Information-Theoretic SFE and PFE with Reduced Communication

Shuaishuai Li[1], Cong Zhang[2], Juntong Lin[3], Anyu Wang[1,2(✉)], and Xiaoyun Wang[1,2]

[1] Zhongguancun Laboratory, Beijing, China
liss@zgclab.edu.cn
[2] Institute for Advanced Study, Tsinghua University, Beijing, China
zhangcong@mail.tsinghua.edu.cn, anyuwang@tsinghua.edu.cn,
xiaoyunwang@tsinghua.edu.cn
[3] School of Cyber Science and Technology, Shandong University, Shandong, China
juntonglin@mail.sdu.edu.cn

Abstract. Secure function evaluation (SFE) and private function evaluation (PFE) are fundamental primitives in multiparty computation. In SFE, multiple parties jointly compute a *public* function over private inputs while revealing nothing beyond the output, whereas in PFE the function itself is *private* and known only to a designated party. In this work, we focus on the semi-honest setting and present more efficient information-theoretic constructions for both SFE and PFE.

For SFE, the classical BGW protocol incurs $O(n^2)$ communication per multiplication gate. The DN protocol (Damgård and Nielsen, Crypto 2007) reduces the amortized communication to linear but still require an additional $O(n^2)$ term, yielding $O(m^*n+n^2)$ communication for m^* multiplication gates. This becomes suboptimal in the regime $m^* = o(n)$. We introduce a simple technique that removes this quadratic overhead, achieving strictly linear $O(m^*n)$ communication.

For PFE, the only existing information-theoretic approach relies on universal circuits, which results in $O(m^5n+n^2)$ complexity for arithmetic circuits. We develop new techniques that avoid universal circuits entirely. Combined with our SFE improvements, this yields an honest-majority PFE protocol achieving $O(m^2n)$ communication for circuit size m. We further obtain improved efficiency in special cases, including a three-party protocol with $O(m^{4/3})$ communication, and an n-party protocol tolerating one corruption with $O(m^{(2n-2)/(2n-3)}n)$ communication.

1 Introduction

1.1 Motivation

Secure function evaluation (SFE) [29] enables a set of mutually distrustful parties to jointly compute a public function over their private inputs while revealing nothing beyond the output. In many cases, however, the function itself is sensitive and must remain hidden from the participants. Private function evaluation (PFE) [1] addresses this stronger privacy requirement by allowing one party to hold a private function while the other parties provide private inputs, ensuring that no information about the function is leaked. PFE has found applications in domains such as software diagnostics [7], medical applications [4], and insurance rate calculation [15]; see [2] for a survey.

Over the past decades, both SFE and PFE have been extensively studied under computational assumptions, where highly efficient protocols are known. For SFE, while the original GMW protocol [12] incurs quadratic communication per multiplication gate, linear communication can be achieved using linearly homomorphic encryption (such as the Paillier scheme [23]). For PFE, a line of works [6, 17, 18, 20–22, 28] achieves communication linear in the circuit size, culminating in the recent work by Li et al. [19] that achieves linear communication in both the circuit size and the number of parties.

In contrast, in the information-theoretic setting, SFE and PFE protocols still have a gap to linear communication. For SFE, the seminal BGW protocol [5] requires $O(n^2)$ communication per multiplication gate, resulting in a total communication cost of $O(m^*n^2)$ for a circuit containing m^* multiplication gates. The DN protocol [9] reduces the communication complexity to $O(m^*n+n^2)$, which is amortized linear. However, the additional $O(n^2)$ term makes their protocol suboptimal when $m^* = o(n)$. This raises the question of whether the quadratic overhead can be eliminated, i.e.,

Can information-theoretic SFE be achieved with $O(m^*n)$ communication?

For PFE, the situation is even more limited. To date, the only general approach for information-theoretic PFE relies on universal circuits (UCs) [27]: the parties securely evaluate a universal circuit that emulates the private target circuit. While this approach yields $O(m \log m)$ communication complexity for Boolean circuits of size m, known arithmetic universal circuits [24, 26] have size $O(m^5)$, leading to $O(m^5)$ communication and computational complexity. As a consequence, UC-based information-theoretic PFE for arithmetic circuits is practically infeasible. This motivates the following question:

Can information-theoretic PFE for arithmetic circuits be realized with significantly lower communication than UC-based constructions?

1.2 Main Results

We answer both of the above questions affirmatively in the semi-honest, static setting.

SFE with strictly-linear communication. Our first result is an information-theoretic SFE protocol with communication cost $O((m^* + \kappa)n)$ when $m^* = o(n)$, which is strictly linear in both m^* and n.

Theorem 1. *For any integers $n > 1$ and $t < n/2$, there exists an information-theoretic n-party SFE protocol that securely computes any arithmetic circuit in the presence of a semi-honest, static adversary controlling up to t parties with $O((m^* + \kappa)n)$ communication complexity, where m^* is the number of multiplication gates in the circuit and κ is the statistical security parameter.*

We remark that the main advantage of our protocol is that it achieves linear communication with respect to both m^* and n in the regime $m^* = o(n)$. When $m^* = \Omega(n)$, the DN protocol [9] should be used, as it already achieves linear communication complexity $O(m^*n + n^2) = O(m^*n)$.

Multiparty PFE with quadratic communication. For multiparty PFE, we present the first information-theoretic protocols for arithmetic circuits that do not rely on universal circuits (UCs). Our protocol achieves communication complexity $O(m^2)$ for any arithmetic circuit of size m, representing a substantial asymptotic improvement over the $O(m^5)$ complexity incurred by UC-based approaches.

Theorem 2. *For any integers $n > 1$ and $t < n/2$, there exists an information-theoretic n-party PFE protocol that securely computes any arithmetic circuit in the presence of a semi-honest, static adversary controlling up to t parties. The communication complexity is $O((m^2 + \kappa)n)$ when $m^2 = o(n)$, and $O(m^2 n)$ when $m^2 = \Omega(n)$, where m is the circuit size and κ is the statistical security parameter.*

Three-party PFE with subquadratic communication. Furthermore, in the three-party setting, we propose a more efficient construction with $O(m^{4/3})$ communication complexity by leveraging information-theoretic symmetric private information retrieval (SPIR) [10]. We also extend this optimized protocol to the n-party setting, achieving a lower communication complexity of $O(m^{(2n-2)/(2n-3)})$, but against an adversarial model that is weaker than honest majority and stronger than single corruption.

Theorem 3. *There exists an information-theoretic three-party PFE protocol with communication complexity $O(m^{4/3})$ that securely computes any arithmetic circuit in the presence of a semi-honest, static adversary controlling at most one party. More generally, in the n-party setting where the adversary may either corrupt only the party owning the circuit, or corrupt any subset of up to $t < n/2$ parties excluding the circuit-owning party, there exists a protocol with communication complexity $O(m^{(2n-2)/(2n-3)}n)$.*

2 Technical Overview

2.1 SFE with Strictly-Linear Communication

The DN protocol [9], like the seminal BGW protocol [5], is also based on Shamir secret sharing [25]. For a secret x over a field $\mathbb{F}$, to generate a Shamir sharing, a dealer first samples a random polynomial g of degree t such that its constant term is x, and each party P_i receives the share $g(i)$. The DN protocol proceeds in three stages. In the input sharing stage, each party generates a Shamir sharing of its input. In the circuit evaluation stage, the parties evaluate the circuit gate by gate, with the output of each gate represented as a Shamir sharing. The linear homomorphism

property of Shamir secret sharing allows the parties to compute addition gates locally without any communication. For multiplication gates, note that the parties can locally compute a sharing of the product; however, the resulting sharing corresponds to a polynomial of degree $2t$. To enable further computation, this $2t$-degree sharing should be converted into a degree-t sharing. The main optimization of the DN protocol over the BGW protocol lies in how this degree-reduction step is performed.

We denote by $\langle x \rangle^t$ a Shamir sharing of a secret x corresponding to a degree-t polynomial (referred to as a t-Shamir sharing), and by $\langle x \rangle^{2t}$ a $2t$-Shamir sharing. In the BGW protocol, to convert a $2t$-Shamir sharing into a t-Shamir sharing, each party P_i generates a t-Shamir sharing of its local share x_i. By Lagrange interpolation, we have $x = \sum_{i \in [n]} \lambda_i x_i$, where $\lambda_i = \prod_{j \in [n] \setminus \{i\}} -j(i-j)^{-1}$ are the Lagrange coefficients. Therefore, the parties can locally compute a t-Shamir sharing of x as $\langle x \rangle^t = \sum_{i \in [n]} \lambda_i \langle x_i \rangle^t$. This approach clearly incurs $O(n^2)$ communication cost, since each party must generate a t-Shamir sharing for its share.

The DN protocol improves the efficiency of this conversion. It first observes that if the parties have prepared a double sharing, consisting of a t-Shamir sharing $\langle r \rangle^t$ and a $2t$-Shamir sharing $\langle r \rangle^{2t}$ for a random value r, then they can convert $\langle x \rangle^{2t}$ into $\langle x \rangle^t$ using only linear communication. Concretely, the parties locally compute $\langle x \rangle^{2t} - \langle r \rangle^{2t}$ and open the value $x - r$. They then reconstruct a t-Shamir sharing of x by computing $\langle x \rangle^t \leftarrow (x - r) + \langle r \rangle^t$. The problem now reduces to how to generate such double sharings. In particular, since each multiplication gate requires one double sharing, they aim to generate such sharings with amortized linear communication.

To achieve this, the DN protocol proceeds as follows. First, each party P_i generates a double sharing of a random value $r^{(i)}$. For security, it is required that the adversary does not know $r^{(i)}$. However, if the adversary corrupts P_i, it learns $r^{(i)}$. Therefore, among the n double sharings generated by the n parties, t of them are known to the adversary, while the remaining $n - t$ are unknown. The question is how to extract these $n - t$ random double sharings. The DN protocol achieves this by computing $n - t$ linear combinations of the n random values. As long as these linear combinations are linearly independent and each depends on all n inputs, the resulting $n-t$ values remain unknown to the adversary. Specifically, these $n-t$ linear combinations are defined using an $(n-t) \times n$ Vandermonde matrix:

$$\mathsf{M} = \begin{pmatrix} 1 & c_1 & c_1^2 & \cdots & c_1^{n-1} \\ 1 & c_2 & c_2^2 & \cdots & c_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & c_{n-t} & c_{n-t}^2 & \cdots & c_{n-t}^{n-1} \end{pmatrix}$$

Here, $c_1, \ldots, c_{n-t}$ are $n - t$ different values. Finally, we note that generating n double sharings incurs $O(n^2)$ communication, and thus the amortized communication is $O(n^2/(n-t)) = O(n)$ (recall that $t < n/2$).

Overall, we can summarize the core idea of the DN protocol as the following observation: from the n double sharings generated by the n parties, one can extract $n - t > n/2$ random double sharings that remain unknown to the adversary. However, since each multiplication gate consumes only one double sharing, this leads to inefficiency when the number of multiplication gates m^* is smaller than $n-t$. In particular, when $m^* = o(n)$, the communication of the DN protocol is $O(n^2 + m^* n) = O(n^2)$, which is higher than the desired linear communication complexity $O(m^* n)$.

Our approach: subsample instead of full participation. To eliminate the quadratic term $O(n^2)$, our starting point is a simple idea: instead of having all parties generate double sharings, we let only $n^* < n$ parties do so. If a constant fraction of these n^* parties are honest, then we can extract $O(n^*)$ uniformly random double sharings. Note that, for semi-honest and static adversaries, we can let any party select these n^* parties.

At first glance, this approach seems problematic, since we do not know in advance which parties are corrupted, and the adversary could potentially corrupt all n^* selected parties. To address this, we let the n^* parties be chosen uniformly at random. We show that, for any constant $c > 2$, with overwhelming probability in n^*, at least n^*/c selected parties are honest. This ensures that we can safely extract $\Theta(n^*)$ uniformly random double sharings.

Since we only need m^* double sharings, it suffices to set $n^* = \Theta(m^*)$. Moreover, to achieve a negligible failure probability, we also require $n^* = \Theta(\kappa)$. Therefore, we set $n^* = \Theta(m^* + \kappa)$ parties to generate double sharings, resulting in a total communication cost of $O((m^* + \kappa)n)$, which is linear in both m^* and n. For concrete efficiency, we will further show how to minimize n^* for a given m^* while ensuring that the failure probability is bounded by $2^{-\kappa}$.

2.2 Multiparty PFE with Quadratic Communication

Existing information-theoretic SFE protocols, including BGW [5] and DN [9], are based on secret sharing. We also leverage secret sharing to construct our PFE protocol. Recall that, unlike SFE, PFE assumes that the circuit being evaluated is a private input held by one of the parties. For convenience, we assume that P_1 owns the circuit, and we refer to the remaining parties $P_2, \ldots, P_n$, who are unaware of the circuit, as circuit-oblivious parties.

Starting point. In secret sharing schemes, a party's share typically varies for different secrets. Consequently, when evaluating each gate, the circuit-oblivious parties $P_2, \ldots, P_n$ cannot determine which shares should be used for the computation. Recently, Li et al. [19] proposed an elegant solution to this problem. Their idea is to use an encryption-based secret sharing scheme. Concretely, for a secret x, the circuit owner P_1 holds a ciphertext of x under some key s, while the circuit-oblivious parties hold a secret sharing of s. The advantage of this encryption-based sharing is that, if the key can be used to encrypt an arbitrary (polynomially bounded) number of plaintexts, then the shares held by the circuit-oblivious parties can remain unchanged for all secret values appearing in the circuit. As a result, when evaluating a gate, the circuit-oblivious parties always take the key sharing as input, and thus they do not face the problem of not knowing which shares should be used for computing the gate.

We refer to secret sharing schemes with the property that the shares of a subset of parties remain fixed as *chained secret sharing*, since these shares behave as if they are "locked by a chain" and never change.

Information-theoretic chained secret sharing. Since we focus on PFE with information-theoretic security, we must rely on an information-theoretically secure chained secret sharing scheme. We follow the approach of [19] and use encryption to construct chained secret sharing. Specifically, we must use an information-theoretically secure encryption scheme, namely the one-time pad (OTP). However, the drawback of OTP is that each key can be used to encrypt only a single message, whereas we need to use the same key to encrypt all intermediate values appearing in the circuit, since this is necessary to ensure that circuit-oblivious parties always take the same inputs when evaluating each gate. To address this issue, our idea is for the parties to directly prepare a *key vector*, where each entry is used to encrypt one value. Specifically, a sharing of x is of form $((k, [\![x]\!]), \langle \mathbf{s} \rangle)$ satisfying that

- $\mathbf{s}$ is a key vector $(s_1, \ldots, s_\alpha)$ and its length α equals the number of wires in the evaluated circuit. Note that α equals the circuits size m plus the number of inputs.
- The index $k \in [\alpha]$ indicates that $[\![x]\!]$ is an encryption of x under the k-th OTP key s_k.
- $\langle \mathbf{s} \rangle$ is a (t, n) sharing of $\mathbf{s}$ such that any t shares reveal no information about $\mathbf{s}$, while any $t+1$ shares determine $\mathbf{s}$. Here, t is the corruption threshold.

Notably, since the circuit owner P_1 knows which key corresponds to which computation gate, the index k is known to P_1 (but unknown to the other parties). Consequently, in the sharing $((k, [\![x]\!]), \langle \mathbf{s} \rangle)$, the share of P_1 consists of $(k, [\![x]\!])$ together with its share in $\langle \mathbf{s} \rangle$, whereas the share of each other party P_i consists solely of its share in $\langle \mathbf{s} \rangle$. When evaluating each gate, the circuit-oblivious parties always take their shares in $\langle \mathbf{s} \rangle$ as input, which enables them to participate in the computation without knowing the circuit.

We remark that the above OTP-based chained secret sharing is not a threshold secret sharing scheme. Recall that a (t, n)-threshold sharing is required to satisfy the properties of correctness and privacy. Correctness requires that the secret can be uniquely determined from any set of $t + 1$ shares. Privacy requires that any t shares reveals no information about the secret. Our construction, however, satisfies only the privacy property and does not satisfy correctness. This is because only P_1 holds the ciphertext $[\![x]\!]$, which implies that a set of $t+1$ shares can reconstruct the secret if and only if it contains the share of P_1.

Although our chained secret sharing scheme does not satisfy the standard correctness requirement, it satisfies a relaxed form of correctness: there *exists* a set of $t+1$ shares that can reconstruct the secret. This relaxed correctness is sufficient for our protocol. We formally define secret sharing with such a relaxed correctness guarantee as *existential secret sharing*.

Evaluating the circuit without leaking gate type. After deciding on the secret sharing scheme to be used, we proceed to show how to evaluate the circuit. In particular, we must ensure that the circuit-oblivious parties do not learn the type of the gate being evaluated, namely, they should not be able to distinguish whether the current gate is an addition gate or a multiplication gate. We note that, as the circuit topology is unknown, only one gate can be evaluated per round to avoid leaking information about the circuit topology. We first consider how to evaluate addition and multiplication gates separately, and then discuss how to hide the gate type.

Suppose the domain is a field $\mathbb{F}$, and the evaluated circuit contains α wires. Recall that we define a sharing of an input x as the form $\langle\!\langle x \rangle\!\rangle = ((k, \tilde{x}), \langle \mathbf{s} \rangle)$, where $\langle \mathbf{s} \rangle = (\langle s_1 \rangle, \ldots, \langle s_\alpha \rangle)$, each $\langle s_i \rangle$ is a (t, n)-sharing of the i-th key s_i, and $\tilde{x} = x - s_k$. Notably, each s_i is a random field element. Assume the input sharings to an addition gate are $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \langle \mathbf{s} \rangle)$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \langle \mathbf{s} \rangle)$. The goal is to compute a sharing of $w = u + v$. Denote this sharing as $\langle\!\langle w \rangle\!\rangle = ((k, \tilde{w}), \langle \mathbf{s} \rangle)$. Since P_1 knows the key index k corresponding to w, and the shares of circuit-oblivious parties are fixed, the only remaining task is to enable P_1 to obtain $\tilde{w}$. Observe that

$$\tilde{w} = w - s_k = u + v - s_k = \tilde{u} + s_{k_1} + \tilde{v} + s_{k_2} - s_k.$$

Denote $\mathbf{e}_i$ the i-th unit vector of length α, then

$$\tilde{w} = \tilde{u} + \mathbf{e}_{k_1}^\top \mathbf{s} + \tilde{v} + \mathbf{e}_{k_2}^\top \mathbf{s} - \mathbf{e}_k^\top \mathbf{s} = \tilde{u} + \tilde{v} + \mathbf{h}^\top \mathbf{s},$$

where $\mathbf{h} = \mathbf{e}_{k_1} + \mathbf{e}_{k_2} - \mathbf{e}_k$. Since P_1 knows $\tilde{u}$ and $\tilde{v}$, it suffices to enable P_1 to obtain $\mathbf{h}^\top \mathbf{s}$. To achieve this, we let the parties compute the inner product $\mathbf{h}^\top \mathbf{s}$ using an information-theoretic secure function evaluation (SFE) protocol, where $\mathbf{h}$ is the private input of P_1. Recall that the parties hold a (t, n)-sharing of $\mathbf{s}$. Assuming this is a Shamir sharing [25], the parties can then apply the BGW protocol [5], with optimizations for achieving linear (in the number of parties) communication [9, 13].

We further consider computing a multiplication gate. Let $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \langle \mathbf{s} \rangle)$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \langle \mathbf{s} \rangle)$ be the input sharings. The goal is to compute a sharing of $w = uv$. Denote this sharing as $\langle\!\langle w \rangle\!\rangle = ((k, \tilde{w}), \langle \mathbf{s} \rangle)$. We need to enable P_1 to learn $\tilde{w}$. Note that

$$\tilde{w} = w - s_k = uv - s_k = (\tilde{u} + s_{k_1})(\tilde{v} + s_{k_2}) - s_k.$$

Still, if we denote $\mathbf{e}_i$ the i-th unit vector of length α, then

$$\tilde{w} = (\tilde{u} + \mathbf{e}_{k_1}^\top \mathbf{s})(\tilde{v} + \mathbf{e}_{k_2}^\top \mathbf{s}) - \mathbf{e}_k^\top \mathbf{s} = \tilde{u}\tilde{v} + \mathbf{h}^\top \mathbf{s} + (\mathbf{e}_{k_1}^\top \mathbf{s})(\mathbf{e}_{k_2}^\top \mathbf{s}),$$

where $\mathbf{h} = \tilde{u}\mathbf{e}_{k_2} + \tilde{v}\mathbf{e}_{k_1} - \mathbf{e}_k$. It suffices to enable P_1 to obtain $\mathbf{h}^\top \mathbf{s} + (\mathbf{e}_{k_1}^\top \mathbf{s})(\mathbf{e}_{k_2}^\top \mathbf{s})$. Similar to the evaluation of an addition gate, this can be achieved by evaluating the function $\mathbf{h}^\top \mathbf{s} + (\mathbf{e}_{k_1}^\top \mathbf{s})(\mathbf{e}_{k_2}^\top \mathbf{s})$ using an information-theoretic SFE protocol.

Finally, we consider how to hide the gate type. Note that computing addition and multiplication gates requires securely evaluating a certain function in each case. Beyond this, the parties do not require any further interaction. Therefore, in order to hide the gate type, we let the parties evaluate the same function for both addition and multiplication gates, with the only difference being that P_1 uses different inputs. We define this function as

$$f((\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2), \mathbf{s}) = \mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}),$$

where $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2)$ is the private input of P_1. When evaluating an addition gate, P_1 sets $\mathbf{h} = \mathbf{e}_{k_1} + \mathbf{e}_{k_2} - \mathbf{e}_k$, and both $\mathbf{c}_1$ and $\mathbf{c}_2$ are defined as all-zero vectors. When evaluating a multiplication gate, P_1 sets $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2) = (\tilde{u}\mathbf{e}_{k_2} + \tilde{v}\mathbf{e}_{k_1} - \mathbf{e}_k, \mathbf{e}_{k_1}, \mathbf{e}_{k_2})$. Denote $y = f((\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2), \mathbf{s})$. We let P_1 output $\tilde{u} + \tilde{v} + y$ for an addition gate and $\tilde{u}\tilde{v} + y$ for a multiplication gate. We can prove that P_1 always obtains the desired output.

2.3 Three-Party PFE with Subquadratic Communication

In our PFE protocol, when evaluating each gate, the parties need to evaluate a public function containing $O(\alpha)$ multiplications, where α denotes the number of wires in the evaluated private circuit. As a result, if the evaluated circuit contains m gates, the overall communication will be $O(m^2)$. We further show that in the three-party setting the communication complexity can be significantly reduced to $O(m^{4/3})$.

Let us return to how to evaluate addition and multiplication gates (without hiding the gate type). Recall that, for input sharings $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \langle \mathbf{s} \rangle)$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \langle \mathbf{s} \rangle)$, we need to let P_1 securely obtain $\tilde{u} + s_{k_1} + \tilde{v} + s_{k_2} - s_k$ (for addition), or $(\tilde{u} + s_{k_1})(\tilde{v} + s_{k_2}) - s_k$ (for multiplication). To this end, we let the parties evaluate a public function with each key s_k (and also s_{k_1} and s_{k_2}) being computed via an inner-product $\mathbf{e}_k^\top \mathbf{s}$. This leads to $O(m)$ communication, since the length of $\mathbf{s}$ equals the number of wires in the evaluated circuit (which is the sum of the number of input wires and the circuit size m). To optimize this, our idea is to design a protocol with *sublinear* communication that allows the parties to obtain a sharing of s_k (as well as s_{k_1} and s_{k_2}), instead of computing these values directly via inner products.

Our core observation is that, in our protocol, P_1 knows the index k, while the other parties hold shares of the vector $\mathbf{s}$. If we conceptually view $\mathbf{s}$ as a database, then the functionality we need closely resembles private information retrieval (PIR) [8]. In an h-server PIR protocol, a client holds an index k, and the h servers hold an identical database $\mathsf{DB} = (\mathsf{DB}_1, \ldots, \mathsf{DB}_\alpha)$. The goal of PIR is to allow the client to obtain the k-th data item DB_k without revealing the index k. However, although standard PIR is closely related to our setting, it does not directly provide the functionality we require. In particular, adapting PIR to our protocol requires addressing three issues.

First, in our PFE protocol, the parties hold a *sharing* of the key vector, while standard PIR assumes that every server stores the entire database. To handle this issue, we first restrict our attention to the three-party setting with parties P_1, P_2, P_3. We then modify the underlying secret sharing scheme. Originally, for a secret x, the sharing has the form $((k, \tilde{x}), \langle \mathbf{s} \rangle)$, where $\langle \mathbf{s} \rangle$ denotes a $(1,3)$-sharing of $\mathbf{s}$. We change the sharing so that P_1 holds $(k, \tilde{x})$, while both P_2 and P_3 hold the key vector $\mathbf{s}$. Clearly, this sharing still satisfies privacy: any single share reveals no information about the secret.

The second issue is that our goal is not to let P_1 learn s_k itself, but rather to let the parties obtain a sharing of s_k (so that P_1 learns no information about s_k). Therefore, we cannot directly define the PIR database to be $\mathbf{s}$, since this would allow P_1 to obtain s_k directly from the PIR protocol. To address this issue, we let P_2 and P_3 first agree on a random value r, and then define the database as $\mathbf{s} - (r, r, \ldots, r)$, that is, the j-th entry of the database is $s_j - r$. In this way, P_1 obtains $s_k - r$ rather than s_k.

The final issue is that standard PIR does not protect the privacy of the database. As a result, if we directly use a PIR protocol, P_1 may obtain additional information about the key vector, which is insecure. Our solution to this issue is to utilize a symmetric PIR (SPIR) protocol [10], which, in addition to the guarantees of PIR, ensures that P_1 can obtain only the k-th entry, namely $s_k - r$.

We have now reduced the task of computing a sharing of s_k to invoking a two-server SPIR protocol. In [10, 11], by letting the two servers (i.e., P_2 and P_3) jointly agree on $O(\alpha^{1/3})$ random values (which incurs $O(\alpha^{1/3})$ communication), the authors obtained an information-theoretic two-server SPIR protocol with communication complexity $O(\alpha^{1/3})$. By applying this protocol, we obtain a protocol for computing a sharing of s_k with communication complexity $O(\alpha^{1/3})$. With a sharing of s_k (as well as s_{k_1} and s_{k_2}), the parties can then securely compute $\tilde{u} + s_{k_1} + \tilde{v} + s_{k_2} - s_k$ (for addition) and $(\tilde{u} + s_{k_1})(\tilde{v} + s_{k_2}) - s_k$ (for multiplication), with communication independent of the circuit size m. Consequently, the communication for evaluating a single gate is reduced to $O(m^{1/3})$.

To complete the protocol, it remains to hide the gate type. Our idea is to let the parties evaluate the same function for both addition and multiplication gates. We define this function as

$$f((c_1, c_2, c), s_{k_1}, s_{k_2}, s_k) = c_1 s_{k_1} + c_2 s_{k_2} + c s_{k_1} s_{k_2} - s_k,$$

where (c_1, c_2, c) is the private input of P_1. When evaluating an addition gate, P_1 sets $(c_1, c_2, c) = (1, 1, 0)$. When evaluating a multiplication gate, P_1 sets $(c_1, c_2, c) = (\tilde{v}, \tilde{u}, 1)$. Denote $y = f((c_1, c_2, c), s_{k_1}, s_{k_2}, s_k)$. We let P_1 output $\tilde{u} + \tilde{v} + y$ for an addition gate and $\tilde{u}\tilde{v} + y$ for a multiplication gate. One can verify that P_1 always obtains the desired output. Moreover, the communication complexity for evaluating a single gate remains $O(m^{1/3})$. Consequently, we obtain an information-theoretic three-party PFE protocol with overall communication complexity $O(m^{4/3})$.

Extending to any number of parties. We can extend our optimized three-party PFE protocol to the general n-party setting. In our three-party protocol, the sharing form of a secret x is $((k, \tilde{x}), \mathbf{s}, \mathbf{s})$. In the n-party setting, we simply generalize it to $((k, \tilde{x}), \mathbf{s}, \ldots, \mathbf{s})$, where party P_1 holds the share $(k, \tilde{x})$, and each of the remaining parties holds the same share $\mathbf{s}$. To ensure that the adversary learns no information about the secret, we require that the adversary either corrupts only P_1, or corrupts a set of at most $t < n/2$ parties that does not include P_1. Note that this corruption model is stronger than the $t = 1$ setting, in which the adversary is allowed to corrupt only a single party.

In the three-party setting, the communication complexity for evaluating a single gate is $O(m^{1/3})$, which comes from invoking an information-theoretic two-server SPIR protocol. In the n-party setting, there are $n - 1$ parties that hold the key vector, and hence the parties can invoke an $(n-1)$-server SPIR scheme. It was shown in [11] that for any constant $h \geq 2$, there exists an information-theoretic h-server SPIR protocol with communication complexity $O(\alpha^{1/(2h-1)})$. This construction additionally requires the servers to share $O(\alpha^{1/(2h-1)})$ random values, which incurs a communication cost of $O(\alpha^{1/(2h-1)} h)$. Applying this result with $h = n - 1$ to our PFE protocol, the communication complexity for evaluating a single gate will be $O(m^{1/(2(n-1)-1)} n) = O(m^{1/(2n-3)} n)$, and hence the total communication complexity is $m \cdot O(m^{1/(2n-3)} n) = O(m^{(2n-2)/(2n-3)} n)$.

Moreover, the work [11] also gives an $O(\log \alpha)$-server SPIR protocol with communication $O(\log^2 \alpha \log\log \alpha)$. This construction requires the servers to share $O(\log^2 \alpha \log\log \alpha)$ random values. Consequently, if the number of

parties n depends on the circuit size m and satisfies $n = O(\log m)$, then we can achieve quasilinear communication complexity $m \cdot O(n \log^2 m \log\log m) = O(m \log^3 m \log\log m)$.

3 Preliminaries

Notations. Let κ be the statistical security parameter. For any integers a, b satisfying $a \le b$, we define $[a, b]$ to be the set $\{a, \ldots, b\}$. When $a = 1$, we abbreviate $[a, b]$ as $[b]$. We use PPT to denote probabilistic polynomial-time. For two random variables X and Y, we write $X \equiv Y$ if they have identical distributions. For a sequence $x_1, \ldots, x_n$ and a set $I \subseteq [n]$, we define x_I as the set of values $\{x_i\}_{i \in I}$.

3.1 Secret Sharing

Due to space limitations, we defer the formal definition of secret sharing to Appendix A. Below, we describe two important secret sharing schemes used in existing information-theoretic SFE [3, 5, 9, 13].

Shamir secret sharing [25]. The secret space is a field $\mathbb{F}$ satisfying $|\mathbb{F}| > n$. To share a secret $x \in \mathbb{F}$, the dealer samples a random polynomial f over $\mathbb{F}$ of degree at most t such that $f(0) = x$. Concretely, the dealer chooses random coefficients $a_1, \ldots, a_t \in \mathbb{F}$ and defines $f(z) = x + \sum_{i=1}^{t} a_i z^i$. Each party P_j is assigned a distinct nonzero point $c_j \in \mathbb{F}^*$, and its share is the evaluation of the polynomial at that point, namely $x_j = f(c_j)$. We refer to $(x_1, \ldots, x_n)$ as a t-Shamir sharing of x, denoted by $\langle x \rangle^t$.

Given any set $I \subseteq [n]$ with $|I| > t$, the parties in I can reconstruct the secret by interpolating the polynomial f from the points $(\alpha_i, x_i)_{i \in I}$ and computing $x = f(0)$. On the other hand, any t shares are actually t random field elements, hence they reveal no information about the secret.

In the DN protocol [9], amortized linear communication is achieved through the efficient generation of double sharings. A double sharing consists of both a t-Shamir sharing and a $2t$-Shamir sharing of the same random value. The generation of a $2t$-Shamir sharing is almost identical to that of a t-Shamir sharing, except that the sampled random polynomial has degree $2t$ rather than t.

Replicated secret sharing [16]. To share a secret x, the dealer first samples $\binom{n}{t}$ random values $\{X_T\}_{T \subseteq [n], |T| = t}$ subject to $x = \sum_{T \subseteq [n], |T| = t} X_T$. For each party P_j, its share consists of all values X_T such that $j \notin T$, namely $\{X_T\}_{j \notin T}$.

From the construction it follows that each value X_T is hidden from exactly the parties indexed by T. Therefore, any set $T \subseteq [n]$ of parties with $|T| = t$ does not know the value X_T, and thus cannot reconstruct the secret x. On the other hand, any set of more than t parties collectively possesses all the values X_T and can reconstruct x by summing them.

Since the share of each party P_j contains exactly those X_T for which $j \notin T$, replicated secret sharing results in a large share size of $\binom{n-1}{t}$. Consequently, this scheme is primarily used when the number of parties n is small, particularly in the case of $n = 3$ (and $t = 1$) [3].

3.2 Chained Secret Sharing

We formally define chained secret sharing in the information-theoretic setting. In particular, we restrict our attention to perfect security. In our definition, we have an explicit upper bound α on the number of secrets. In the scheme used in [19], this bound is implicitly assumed to be polynomial in the security parameter.

Definition 1 (Chained Secret Sharing, Adapted from [19]). *A (t, n, α, η)-chained secret sharing scheme over a secret space $\mathcal{X}$ consists of three PPT algorithms* $(\mathsf{Chain}, \mathsf{Share}, \mathsf{Rec})$ *with the following syntax.*

- $\mathsf{Chain}(\alpha)$. *On input an integer α, output $n - \eta$ state values $\mathsf{st}_1, \ldots, \mathsf{st}_{n-\eta}$ and the η fixed shares, called chains, $x_{n-\eta+1}, \ldots, x_n$.*
- $\mathsf{Share}(j, x, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]})$. *On input the j-th secret $x \in \mathcal{X}$, state values $\mathsf{st}_{[n-\eta]} = \{\mathsf{st}_1, \ldots, \mathsf{st}_{n-\eta}\}$, and chains $x_{[n-\eta+1,n]} = \{x_{n-\eta+1}, \ldots, x_n\}$, output $n - \eta$ shares $x_1, \ldots, x_{n-\eta}$.*
- $\mathsf{Rec}(I, x_I)$. *On input an index set I with $|I| > t$ and the shares $x_I = \{x_i\}_{i \in I}$ in this set, output a value $x \in \mathcal{X}$.*

These algorithms satisfy the following properties.

– ***Correctness.*** *For every $(j, x) \in [\alpha] \times \mathcal{X}$ and every index set $I \subseteq [n]$ with $|I| > t$, the probability*

$$\Pr\left[\mathsf{Rec}(I, x_I) = x : \begin{array}{c}(\mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}) \leftarrow \mathsf{Chain}(\alpha), \\ x_{[n-\eta]} \leftarrow \mathsf{Share}(j, x, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]})\end{array}\right]$$

is equal to 1.

– ***Privacy.*** *For any α secrets $x^{(1)}, \ldots, x^{(\alpha)}$ and every index set $I \subseteq [n]$ with $|I| \leq t$, the distribution*

$$\left\{\{x_{I\setminus[n-\eta]}, (x^{(j)}_{I\cap[n-\eta]})_{j\in[\alpha]}\} : \begin{array}{c}(\mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}) \leftarrow \mathsf{Chain}(\alpha), \\ x^{(1)}_{[n-\eta]} \leftarrow \mathsf{Share}(1, x^{(1)}, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}), \\ \vdots \\ x^{(\alpha)}_{[n-\eta]} \leftarrow \mathsf{Share}(\alpha, x^{(\alpha)}, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]})\end{array}\right\}$$

is independent of the secrets $x^{(1)}, \ldots, x^{(\alpha)}$.

We present a simple impossibility result for chained secret sharing.

Lemma 1. *There exists no (t, n, α, η)-chained secret sharing scheme over a secret space $\mathcal{X}$ when $\alpha > 1, \eta > t, |\mathcal{X}| > 1$.*

Proof. The proof follows from a simple observation: when $\eta > t$, by correctness, the fixed η shares uniquely determine the secret. Hence there is only one possible value of the secret, which contradicts that $\alpha > 1, |\mathcal{X}| > 1$. □

Note that the setting we require is $n > 2, t < n/2, \alpha > 1, \eta = n - 1$, and $|\mathcal{X}| > 1$. Observe that for any integers t, n satisfying $n > 1$ and $t < n/2$, we always have $t < n - 1$. Therefore, in the setting we required, chained secret sharing does not exist. To circumvent this impossibility result, we choose to relax the correctness requirement. See Section 5 for more details.

3.3 Information-Theoretic SFE

In our PFE protocols, the computation of a gate involves the parties evaluating a specific public circuit. We define the general functionality for computing an arbitrary given circuit C. This functionality also takes a particular secret sharing scheme as a parameter. In particular, the functionality allows the parties to submit their inputs either in the clear or as secret sharings jointly provided by the parties. Likewise, the functionality allows the parties to either learn the output or obtain a sharing of the output.

Functionality $\mathcal{F}_C$: Secure Evaluation of C

C is the evaluated circuit.

1. Receive the inputs in plain or shared form (underlying the given secret sharing scheme) from the parties.
2. Evaluate the circuit C on the received inputs, where any input given in shared form is reconstructed before being used in the computation.
3. Send the output or its shared form to the parties.

We can realize the functionality $\mathcal{F}_C$ with information-theoretic security using either the BGW protocol [5, 9, 13] or the AFLNO protocol [3]. The former is based on Shamir secret sharing [25] and applies to any number of parties, whereas the latter is a three-party protocol based on replicated secret sharing [16] that achieves better concrete efficiency. Note that although the original BGW protocol [5] incurs quadratic communication in the number of parties, subsequent optimizations [9, 13] have significantly improved its scalability, achieving communication complexity $O(m^* n + n^2)$ for n parties when the circuit contains m^* multiplication gates.

3.4 Information-Theoretic Multi-Server Symmetric PIR

Our three-party PFE requires invoking information-theoretic multi-server symmetric private information retrieval (SPIR) [10]. In this protocol, a client holding an index $k \in [\alpha]$ privately queries multiple non-colluding servers, each of which holds a replica of the database, in order to retrieve the k-th entry of the database. We define the corresponding functionality as follows.

Functionality $\mathcal{F}_{\text{spir}}^{h}$: Multi-Server SPIR

h is the number of servers.

1. Receive the index k from the client.
2. Receive the database $\mathsf{DB} = (\mathsf{DB}_1, \ldots, \mathsf{DB}_\alpha)$ from the servers.
3. Send DB_k to the client.

In [10, 11], the authors present an information-theoretic h-server SPIR protocol with communication complexity $O(\alpha^{1/(2h-1)})$ for any constant $h \geq 2$, where the servers share $O(\alpha^{1/(2h-1)})$ random values. They also construct an $O(\log \alpha)$-server SPIR protocol with communication complexity $O(\log^2 \alpha \log\log \alpha)$, requiring $O(\log^2 \alpha \log\log \alpha)$ shared random values among the servers. Note that c shared random values among h servers can be generated using $O(ch)$ communication: one server samples c random values and sends them to the other servers. Therefore, we may state that the communication complexity of the h-server SPIR protocol in [10, 11] is $O(h\alpha^{1/(2h-1)})$, while the communication complexity of their $O(\log \alpha)$-server SPIR protocol is $O(\log^3 \alpha \log\log \alpha)$.

4 Multiparty SFE with Strictly-Linear Communication

We present the LDN protocol (short for Linear DN), which optimizes the DN protocol [9] for evaluating small circuits where the number of multiplication gates is much smaller than the number of parties. The LDN protocol modifies only the double-sharing generation procedure of the DN protocol, while leaving the remaining parts of the protocol unchanged. In particular, we introduce an additional parameter $n^* \in [n]$, which depends on both the number of multiplication gates and the security parameter. We will also discuss how to choose n^* to achieve optimal efficiency.

4.1 Optimizing DN for Small Circuits

Our LDN protocol differs from the DN protocol in the way double sharings are generated. For completeness, we first review the DN protocol, with particular emphasis on its double sharing generation procedure. We then show how LDN optimizes the generation of double sharings.

The DN protocol. The DN protocol begins by generating Shamir sharings of the inputs. The circuit is then evaluated gate by gate, with the output of each gate represented as a Shamir sharing, and finally the output sharing is opened. The key point of the DN protocol is the evaluation of multiplication gates, since addition gates can be computed locally.

To evaluate a multiplication gate, the parties first prepare a double sharing $(\langle r \rangle^t, \langle r \rangle^{2t})$ of the same random value r. Suppose the input sharings for a multiplication gate are $\langle u \rangle^t$ and $\langle v \rangle^t$. To compute $\langle uv \rangle^t$, the parties consume a random double sharing $(\langle r \rangle^t, \langle r \rangle^{2t})$. They first designate a distinguished party (say, P_1), and then execute the following protocol:

1. All parties locally compute $\langle e \rangle^{2t} = \langle u \rangle^t \cdot \langle v \rangle^t - \langle r \rangle^{2t}$.
2. The party P_1 collects all shares of $\langle e \rangle^{2t}$ and reconstructs the secret e. It then broadcasts e to all other parties.
3. Upon receiving e, all parties locally compute $\langle z \rangle^t = e - \langle r \rangle^t$.

Correctness follows directly from the properties of Shamir secret sharing. Since each party sends one field element to P_1, and P_1 sends one field element to every party. Therefore, the above protocol requires communication of $2(n-1)$ field elements.

Generation of double sharings in DN. Observe that each multiplication gate consumes one double sharing. In the DN protocol, to efficiently generate double sharings, each party first generates one double sharing. Note that the resulting n double sharings cannot all be directly used for evaluating multiplication gates, since the adversary knows the secret values corresponding to the double sharings generated by corrupted parties, making them insecure for multiplication.

The key observation is that, since at least $n-t$ parties are honest, at least $n-t$ of these double sharings correspond to secret values unknown to the adversary. To extract such $n-t$ "good" double sharings, the DN protocol applies a Vandermonde matrix of dimension $(n-t) \times n$ to the vector of double sharings. This actually computes $n-t$ linearly

independent linear combinations of the original n double sharings, where each resulting double sharing depends on all of the original ones.

The DN protocol elegantly reduces the amortized communication complexity of MPC to linear. However, it is still not fully satisfactory, as it suffers from an efficiency issue: each invocation generates at least $n-t$ double sharings at once. Consequently, when the number of multiplication gates m^* in the circuit is smaller than $n-t$, many of these double sharings are wasted. In particular, when $m^* = o(n)$, the asymptotic communication complexity of the DN protocol remains $O(n^2)$, whereas we would ideally like to achieve communication complexity $O(m^*n)$.

Waste-free generation of double sharings. In the LDN protocol, we introduce a waste-free method for generating double sharings. Specifically, instead of having all n parties generate the initial double sharings, we let only $n^* < n$ parties do so. Observe that, if we can guarantee that at least m^* of these n^* parties are honest, then we can extract m^* double sharings whose underlying random values are unknown to the adversary. This suffices for evaluating m^* multiplication gates.

Unfortunately, we have no way to guarantee the number of honest parties among these selected parties. In fact, if $n^* < n/2$, then in the worst case all of the selected n^* parties may be corrupted. To overcome this difficulty, instead of requiring a deterministic guarantee on the number of honest parties, we only seek to ensure that the probability of having at least m^* honest parties is sufficiently high. In particular, we require this probability to be at least $1 - 2^{-\kappa}$ for a statistical security parameter κ. To achieve this, we rely on two key ideas.

- First, we select these n^* parties uniformly at random.
- Second, we observe that, although at least half of the parties are honest (since $t < n/2$), it is difficult to guarantee that at least $n^*/2$ of the selected parties are honest. Instead, we aim to ensure that at least n^*/c of the selected parties are honest, for some constant $c > 2$.

It suffices to choose parameters such that $n^*/c \geq m^*$ for evaluating the m^* multiplication gates.

Observe that our modification to the DN protocol is simple: we let n^* randomly selected parties generate the initial double sharings, and then use a Vandermonde matrix of dimension $m^* \times n^*$ to extract m^* double sharings whose underlying secret values are unknown to the adversary. In particular, the description and proof of the modified protocol are almost identical to those of the original DN protocol. We remark that our approach relies on the adversary choosing the corrupted parties at the outset, namely, the adversary is *static*. An adaptive adversary could simply decide to corrupt the selected n^* parties after they have been chosen.

The remaining task is how to choose the value of n^*. This is nontrivial, since we must ensure that n^* is sufficiently large so that enough honest parties are selected, while at the same time keeping n^* small enough to achieve linear communication. In particular, we would like to achieve $n^* = O(m^*)$.

4.2 Parameter Choice and Concrete Efficiency

Our problem can be formalized as follows: choose n^* and a constant $c > 1$ such that the probability that at least n^*/c selected parties are honest is at least $1 - 2^{-\kappa}$. Moreover, for a given m^*, we would like to minimize n^* while ensuring that $n^*/c \geq m^*$. We first prove a useful lemma.

Lemma 2. *Let there be n parties, among which at most $t < n/2$ are corrupted. Suppose we randomly select $n^* \leq n$ parties without replacement. Let X denote the number of honest parties in the selected subset. Then, for any constant $c > 2$, we have $\Pr[X \geq n^*/c] \geq 1 - 2^{-\kappa}$, provided that $n^* \geq 4c^2\kappa \ln 2/(c-2)^2$, where κ is the statistical security parameter.*

Proof. Since $t < n/2$, each randomly selected party belongs to the honest set with probability $p > 1/2$ and $\mathbb{E}[X] = n^* \cdot p > n^*/2$. Using the Chernoff bound for sampling without replacement,

$$\Pr[X \leq (1-\delta)\mathbb{E}[X]] \leq \exp(-\delta^2\mathbb{E}[X]/2)$$

for $0 < \delta < 1$. We want $X \geq n^*/c$, so failure occurs when $X < n^*/c$. Note that $\mathbb{E}[X] > n^*/2$, when $c > 2$, we have $0 < n^*/(c\mathbb{E}[X]) < 1$. Thus, we can set $\delta = 1 - n^*/(c\mathbb{E}[X])$, and then we have

$$\Pr[X < n^*/c] \leq \Pr[X \leq n^*/c] \leq \exp(-\delta^2\mathbb{E}[X]/2)$$

It follows from $\mathbb{E}[X] > n^*/2$ that $\delta = 1 - n^*/(c\mathbb{E}[X]) > 1 - 2/c$. Therefore, it holds that

$$\Pr[X < n^*/c] < \exp(-(1-2/c)^2 n^*/4)$$

n	1000			5000			10000		
m^*	100	200	300	100	200	300	100	200	300
DN	1000	1000	1000	5000	5000	5000	10000	10000	10000
LDN	415	674	920	415	674	920	415	674	920
Savings factor	**2.41**	**1.48**	**1.09**	**12.05**	**7.42**	**5.43**	**24.10**	**14.84**	**10.87**

Table 1. Comparison of the number of parties required to generate double sharings in the DN protocol and the LDN protocol. Here, n denotes the total number of parties, and m^* denotes the number of multiplication gates in the circuit.

To ensure that failure probability is less than $2^{-\kappa}$, we require that

$$\exp(-(1-2/c)^2 n^*/4) \leq 2^{-\kappa}.$$

That is, $n^* \geq 4\kappa \ln 2/(1-2/c)^2 = 4c^2\kappa \ln 2/(c-2)^2$. □

Asymptotic efficiency. Lemma 2 shows that if $n^* \geq 4c^2\kappa \ln 2/(c-2)^2$, then n^*/c of the parties are honest with probability at least $1-2^{-\kappa}$. For m^* multiplication gates, we require $n^*/c \geq m^*$, i.e., $n^* \geq cm^*$. Therefore, for any constant $c > 2$, it suffices to choose n^* such that $n^* \geq \max(4c^2\kappa \ln 2/(c-2)^2, cm^*)$. Asymptotically, this implies $n^* = O(m^* + \kappa)$. Hence, we achieve linear communication complexity $O(n^* \cdot n) = O((m^* + \kappa)n)$.

Concrete efficiency. We further consider the concrete efficiency. Specifically, we aim to minimize n^*. Note that n^* needs to satisfy

$$n^* \geq \max(4c^2\kappa \ln 2/(c-2)^2, cm^*), \quad c > 2.$$

Since cm^* is an increasing function in c, and $4c^2\kappa \ln 2/(c-2)^2$ is a decreasing function in c, the minimum is achieved when the two lower bounds are equal:

$$4c^2\kappa \ln 2/(c-2)^2 = cm^*.$$

Note that $c > 2$, hence we take the positive root:

$$c = \frac{2m^* + 2\kappa \ln 2 + 2\sqrt{\kappa \ln 2(2m^* + \kappa \ln 2)}}{m^*}.$$

Substitute back into $n^* \geq cm^*$, we have

$$n^* \geq 2m^* + 2\kappa \ln 2 + 2\sqrt{\kappa \ln 2(2m^* + \kappa \ln 2)}.$$

Since n^* should be an integer, we choose

$$n^* = \lceil 2m^* + 2\kappa \ln 2 + 2\sqrt{\kappa \ln 2(2m^* + \kappa \ln 2)} \rceil.$$

In practice, we typically choose $\kappa = 40$. Substituting this, we obtain

$$n^* = \lceil 2m^* + 80 \ln 2 + 2\sqrt{40 \ln 2(2m^* + 40 \ln 2)} \rceil \leq \lceil 2m^* + 56 + 2\sqrt{56m^* + 769} \rceil.$$

We can now summarize that computing m^* multiplication gates requires $n^* = 2m^* + o(m^*)$ parties, each generating a double sharing. Since generating one double sharing incurs communication of $2(n-1)$, and together with the online stage where P_1 collects all shares and broadcasts them, which costs $2(n-1)$ elements of communication, the total communication required for computing m^* multiplication gates is $6m^*n + o(m^*n)$. That is, the communication cost per party per multiplication gate is 6 elements, which matches the communication efficiency of the DN protocol.

Finally, we present Table 1 comparing the number of parties required to generate double sharings for different values of m^* in the DN protocol and our LDN protocol. In the DN protocol, this number is always n, whereas in the LDN protocol, it is $n^* = 2m^* + o(m^*)$. The results show that our approach significantly improves the efficiency of the DN protocol when $m^* \ll n$.

4.3 Integrating with ATLAS

The ATLAS protocol [13] further improves the concrete efficiency of the DN protocol by reducing the per-party, per-multiplication communication from 6 elements to 4 elements.[4] Their key observation is that when computing a multiplication gate, if the party who collects the secret shares in the online stage (i.e., the distinguished party) is honest, then the corresponding double sharing does not need to ensure that the adversary does not know the underlying secret value. In other words, suppose k parties compute k multiplication gates and each of them in turn acts as the distinguished party. If at most k^* out of these k parties are corrupted, then it suffices that the k double sharings used in the protocol satisfy k^*-wise independence: any subset of k^* of these k sharings is linearly independent. To achieve this, ATLAS first uses the DN method to generate k^* double sharings for which the adversary does not know the corresponding secret values. It then applies a $k \times k^*$ matrix, in which every $k^* \times k^*$ submatrix is invertible, to locally expand these k^* sharings into k double sharings that satisfy the required k^*-wise independence.

We can also apply this technique to our protocol to achieve $4 + \varepsilon$ elements of per-party, per-multiplication communication, for any constant $0 < \varepsilon < 2$. Specifically, to compute m^* multiplication gates, we first choose m^* random parties to serve as the distinguished party in sequence. By Lemma 2, for any $c_1 > 2$, we can guarantee with probability at least $1 - 2^{-\kappa}$ that m^*/c_1 of these parties are honest, provided that $m^* \geq 4c_1^2 \kappa \ln 2/(c_1 - 2)^2$. Therefore, it suffices to generate m^* double sharings that satisfy $m^*(1 - 1/c_1)$-wise independence. In other words, we only need to generate $m^*(1 - 1/c_1)$ double sharings whose corresponding random values are unknown to the adversary. From our previous analysis, this can be achieved by using $2m^*(1 - 1/c_1) + o(m^*)$ randomly selected parties to generate the double sharings. That is, we only need $2m^*(1 - 1/c_1) + o(m^*)$ parties to generate the double sharings. As a result, we can achieve $4(1 - 1/c_1) + 2$ elements of per-party, per-multiplication communication. For any $0 < \varepsilon < 2$, we can set $c_1 = 4/(2 - \varepsilon)$ (which still satisfies $c_1 > 2$), achieving $4 + \varepsilon$ elements of per-party, per-multiplication communication.

Finally, we remark that since we need to select random parties twice, in order to ensure that the total failure probability is at most $2^{-\kappa}$, by the union bound, we must guarantee that the failure probability of each step is at most $2^{-\kappa-1}$, rather than the original $2^{-\kappa}$. This only has a minor impact on the protocol parameters and efficiency.

5 Existential Secret Sharing

To hide the circuit topology in the information-theoretic setting, our PFE protocol requires a (t, n, α, η)-chained secret sharing scheme for $\alpha > 1, t < n/2, \eta = n - 1$. However, since $n - 1 \geq n/2$ holds for any $n \geq 2$, Lemma 1 implies that no chained secret sharing scheme satisfying our requirements exists. To address this issue, we relax the correctness requirement of secret sharing. The standard correctness requirement states that the secret can be reconstructed from *any* $t + 1$ shares. We only require that there *exists* a set of $t + 1$ shares that determines the secret. We refer to such a weakened variant of secret sharing as existential secret sharing.

Definition 2 (Existential Secret Sharing). *A (t, n)-existential secret sharing scheme over a secret space $\mathcal{X}$ consists of two PPT algorithms $(\mathsf{Share}, \mathsf{Rec})$ with the same syntax as a standard secret sharing scheme (Definition 4). These algorithms satisfy the following properties.*

- ***Correctness.*** *There exists an index set $I \subseteq [n]$ with $|I| = t + 1$ such that for every $x \in \mathcal{X}$, it holds that $\Pr[\mathsf{Rec}(I, x_I) = x : (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x)] = 1$.*
- ***Privacy.*** *For every $x \in \mathcal{X}$ and every index set $I \subseteq [n]$ with $|I| \leq t$, it holds that the distribution $\{x_I : (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x)\}$ is independent of the secret x.*

5.1 Chained Existential Secret Sharing

A chained existential secret sharing (CESS) scheme combines the relaxed correctness requirement with the property that some shares remain invariant across different secrets.

Definition 3 (Chained Existential Secret Sharing). *A (t, n, α, η)-chained existential secret sharing (CESS) scheme over a secret space $\mathcal{X}$ consists of three PPT algorithms $(\mathsf{Chain}, \mathsf{Share}, \mathsf{Rec})$ with the same syntax as chained secret sharing (Definition 1). These algorithms satisfy the following properties.*

[4] Before ATLAS, a minor improvement over DN appeared in [14], which reduces the per-party, per-multiplication communication to 5.5 elements.

– ***Correctness.*** *There exists an index set $I \subseteq [n]$ with $|I| = t+1$ such that for every $(j, x) \in [\alpha] \times \mathcal{X}$, the probability*

$$\Pr\left[\mathsf{Rec}(I, x_I) = x : \begin{array}{c} \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]} \leftarrow \mathsf{Chain}(\alpha), \\ (x_1, \ldots, x_{n-\eta}) \leftarrow \mathsf{Share}(j, x, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}) \end{array}\right]$$

is equal to 1.

– ***Privacy.*** *For any α secrets $x^{(1)}, \ldots, x^{(\alpha)}$ and every index set $I \subseteq [n]$ with $|I| \le t$, the distribution*

$$\left\{ \begin{array}{l} \{x_{I \setminus [n-\eta]}, \\ (x^{(j)}_{I \cap [n-\eta]})_{j \in [\alpha]}\} \end{array} : \begin{array}{c} (\mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}) \leftarrow \mathsf{Chain}(\alpha), \\ x^{(1)}_{[n-\eta]} \leftarrow \mathsf{Share}(1, x^{(1)}, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}), \\ \vdots \\ x^{(\alpha)}_{[n-\eta]} \leftarrow \mathsf{Share}(\alpha, x^{(\alpha)}, \mathsf{st}_{[n-\eta]}, x_{[n-\eta+1,n]}) \end{array} \right\}$$

is independent of the secrets $x^{(1)}, \ldots, x^{(\alpha)}$.

From the definition, we know that for any $\eta > \eta'$, a (t, n, α, η)-CESS scheme is also a (t, n, α, η') scheme. Our protocols require a $(t, n, \alpha, n-1)$-CESS scheme.

5.2 An Information-Theoretic CESS Scheme

We construct a perfect $(t, n, \alpha, n-1)$-CESS scheme based on an arbitrary perfect (t, n)-secret sharing scheme.

Theorem 4. *Let $(\mathsf{Share}', \mathsf{Rec}')$ be any perfect (t, n)-secret sharing scheme over a ring $\mathcal{X}$. Then the following three algorithms $(\mathsf{Chain}, \mathsf{Share}, \mathsf{Rec})$ define a perfect $(t, n, \alpha, n-1)$-CESS scheme over $\mathcal{X}$.*

– $\mathsf{Chain}(\alpha)$.
  1. *Sample a random key vector $\mathbf{s} = (s_1, \ldots, s_\alpha) \in \mathcal{X}^\alpha$.*
  2. *Run $\langle s_j \rangle \leftarrow \mathsf{Share}'(s_j)$ for each $j \in [\alpha]$. Denote $\langle s_j \rangle_i$ the i-th share in $\langle s_j \rangle$, and $\mathbf{s}_i = (\langle s_1 \rangle_i, \ldots, \langle s_\alpha \rangle_i)$.*
  3. *Output the state $(\mathbf{s}, \mathbf{s}_1)$ and the fixed shares $(x_2, \ldots, x_n) = (\mathbf{s}_2, \ldots, \mathbf{s}_n)$.*

– $\mathsf{Share}(j, x, ((\mathbf{s}, \mathbf{s}_1), x_2, \ldots, x_n)$.
  1. *Compute $\tilde{x} = x - s_j$.*
  2. *Output the share $x_1 = (j, \tilde{x}, \mathbf{s}_1)$.*

– $\mathsf{Rec}(I, (x_i)_{i \in I})$. *We choose I subject to $1 \in I$.*
  1. *Parse $x_1 = (j, \tilde{x}, \mathbf{s}_1)$ and $x_i = \mathbf{s}_i$ for each $i \in I \setminus \{1\}$.*
  2. *Run $s_j \leftarrow \mathsf{Rec}'(I, (\langle s_j \rangle_i)_{i \in I})$.*
  3. *Output $x = \tilde{x} + s_j$.*

Proof. We prove the correctness and privacy of the scheme respectively.

Correctness. Define the index set I as $I = I' \cup \{1\}$ for an arbitrary size-$|t|$ set $I' \subseteq [2, n]$. We now prove that, given any $t+1$ shares from I, the algorithm Rec correctly reconstructs the secret. Assume x is the secret, and $(x_1, \ldots, x_n) = ((j, \tilde{x}, \mathbf{s}_1), \mathbf{s}_2, \ldots, \mathbf{s}_n)$ is a sharing of x. By the definitions of Chain and Share, if we let $\mathbf{s} = (s_1, \ldots, s_\alpha)$ be the key vector corresponding to the sharing $(\mathbf{s}_1, \ldots, \mathbf{s}_n)$ under $(\mathsf{Share}', \mathsf{Rec}')$, then it holds that $\tilde{x} = x - s_j$. Then by the definition of Rec, the output is $\tilde{x} + s_j = x$. Hence, correctness follows.

Privacy. We consider two cases depending on whether the index set I used for reconstruction contains 1.

If $1 \notin I$, then the shares indexed by I are indeed shares of the key vector under the scheme $(\mathsf{Share}', \mathsf{Rec}')$. These shares do not reveal any information about the secrets, since the key vector is random and therefore independent of the secrets.

If $1 \in I$, then according to the description of the scheme, for any α secrets, the j-th secret is encrypted using the j-th key s_j. By the privacy of the scheme $(\mathsf{Share}', \mathsf{Rec}')$, any set of t parties learns no information about the key vector s. Since s is a random vector, the shares held by these t parties reveal no information about the α secrets. □

6 Multiparty PFE with Quadratic Communication

In this section, we present our multiparty PFE construction, which achieves $O(m^2 n)$ communication complexity for n parties and circuit size m. From now on, let $\mathbf{e}_k$ denote the k-th unit vector.

6.1 Protocol Overview

Our PFE protocol is based on the CESS scheme in Section 5.2. We can instantiate the CESS scheme using linear secret sharing schemes such as Shamir secret sharing [25] or replicated secret sharing [16]. In both cases, linear operations can be performed locally and therefore incur no communication cost. Our protocol consists of three stages: input sharing, circuit evaluation, and output recovery.

Input sharing. Assume the circuit is defined over a field $\mathbb{F}$. Let m be the circuit size and β be the number of inputs. Denote $\alpha = m + \beta$. To generate the CESS sharings of the inputs, the parties need to prepare a key vector of length α for masking the inputs and intermediate values. In particular, this key vector should be available in shared form. To this end, the parties generate α random sharings. We denote this key vector by $\mathbf{s} = (s_1, \ldots, s_\alpha)$, and the sharing of each key s_j by $\langle s_j \rangle$. Recall that in our CESS scheme, the sharing of an input x is of the form $((j, \tilde{x}), \langle \mathbf{s} \rangle)$, where $(j, \tilde{x})$ is held by P_1. Having generated the key vector sharing, it remains to generate $(j, \tilde{x})$ for each input.

We first let P_1 perform a topological sort over all input and intermediate values, mapping the inputs to $[\beta]$ and the intermediate values to $[\beta + 1, \alpha]$, such that the index assigned to each intermediate value is greater than that of every value (input or intermediate) it depends on. For each input x, let its index be j. We then encrypt x using the key s_j, which amounts to computing $\tilde{x} = x - s_j$. We need to let P_1 obtain $\tilde{x}$. Note that j can be safely revealed without leaking any information about the circuit (but the indices corresponding to intermediate values must remain hidden since they rely on the circuit topology). Therefore, we can achieve our goal by letting the party holding x first generate a sharing $\langle x \rangle$ of x. The parties then locally compute $\langle x \rangle - \langle s_j \rangle$ to obtain a sharing of $\tilde{x}$. Finally, they open this sharing to P_1, allowing P_1 to learn $\tilde{x}$.

Circuit evaluation. During the circuit evaluation stage, in order to hide the circuit topology, following the approach of prior PFE work [21], we let the parties evaluate only one gate per round. In particular, recall that P_1 already assign an ordering to all wires in the circuit. We let the parties evaluate the gates following this order. The advantage of doing so is that everyone always knows the index of the output wire of the gate being evaluated. Indeed, for the k-th gate, the index of its output wire is $\beta + k$, and thus the corresponding key is $s_{\beta+k}$. For simplicity, we define $\rho_k = s_{\beta+k}$ for all $k \in [m]$.

Assume the input sharings to the k-th gate are $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \langle \mathbf{s} \rangle)$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \langle \mathbf{s} \rangle)$. For an addition gate, the goal is for P_1 to securely obtain $u + v - \rho_k$. For a multiplication gate, we instead let P_1 securely obtain $uv - \rho_k$. To hide the type of the gate being computed, we have the parties evaluate the same function for both addition and multiplication gates, with P_1 using different inputs. Note that $u = \tilde{u} + s_{k_1}$ and $v = \tilde{v} + s_{k_2}$. We define this function as:

$$f((\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2), \mathbf{s}) = \mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}) - \rho_k,$$

where $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2)$ is the private input of P_1. For an addition gate, P_1 sets $\mathbf{h} = \mathbf{e}_{k_1} + \mathbf{e}_{k_2}$, and lets both $\mathbf{c}_1$ and $\mathbf{c}_2$ be all-zero vectors. For a multiplication gate, P_1 sets $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2) = (\tilde{u}\mathbf{e}_{k_2} + \tilde{v}\mathbf{e}_{k_1}, \mathbf{e}_{k_1}, \mathbf{e}_{k_2})$. Let $y = f((\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2), \mathbf{s})$. Then, for an addition gate, P_1 outputs $\tilde{u} + \tilde{v} + y$, and for a multiplication gate, it outputs $\tilde{u}\tilde{v} + y$. One can verify that P_1 always obtains the correct output.

Output recovery. Let $\langle\!\langle z \rangle\!\rangle = ((k, \tilde{z}), \langle \mathbf{s} \rangle)$ be an output sharing. It suffices to securely evaluate $\tilde{z} + \mathbf{e}_k^\top \mathbf{s}$ in order to recover this output.

6.2 Random Sharing Generation

Our multiparty PFE protocol requires the parties to generate sharings of random values. We define the random sharing generation functionality to accomplish this task. We use the Shamir secret sharing scheme, in which case we can adopt the method used in the DN protocol for generating double sharings. The communication complexity for generating α random sharings is $O(\alpha n + n^2)$, which achieves linear communication when $\alpha = \Omega(n)$. When $\alpha = o(n)$, one can instead use the LDN protocol described in Section 4 to achieve linear communication complexity $O((\alpha + \kappa)n)$.

Functionality $\mathcal{F}_{\text{rsg}}$: Random Sharing Generation

$\mathbb{F}$ is the underlying field.

1. Sample a random value $s \in \mathbb{F}$.
2. Generating a (t, n)-sharing of s, denoted $\langle s \rangle$.
3. Send to each party its corresponding share.

6.3 The Protocol

Protocol 1. Multiparty PFE

Parameter. Let C be the computed circuit over a field $\mathbb{F}$. Let β be the number of inputs, and m be the number of gates. Set $\alpha = \beta + m$.

Setup.

1. The parties invoke the functionality $\mathcal{F}_{\text{rsg}}$ α times to generate α random (t, n)-sharings $\langle s_1 \rangle, \ldots, \langle s_\alpha \rangle$.
2. Denote $\mathbf{s} = (s_1, \ldots, s_\alpha)$ and $\boldsymbol{\sigma} = (s_1, \ldots, s_\beta)$. Define $\rho_k = s_{\beta+k}$ for all $k \in [m]$.

Input sharing.

1. All inputs are computed as (t, n)-sharings. Let the input vector be $\mathbf{x} \in \mathbb{F}^\beta$, and denote the resulting sharing vector by $\langle \mathbf{x} \rangle$.
2. The parties locally compute $\langle \tilde{\mathbf{x}} \rangle \leftarrow \langle \mathbf{x} \rangle - \langle \boldsymbol{\sigma} \rangle$.
3. The parties open the sharing $\langle \tilde{\mathbf{x}} \rangle$ to P_1. Denote $\tilde{\mathbf{x}} = (\tilde{x}_1, \ldots, \tilde{x}_\beta)$.
4. The sharing of the k-th input x_k is $\langle\!\langle x_k \rangle\!\rangle = ((k, \tilde{x}_k), \langle \mathbf{s} \rangle)$.

Circuit evaluation.
For the k-th gate, let $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \langle \mathbf{s} \rangle)$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \langle \mathbf{s} \rangle)$ be the input sharings and the parties compute this gate as follows.

1. P_1 computes the vectors $\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2 \in \mathbb{F}^\alpha$:
   (a) For addition, $\mathbf{h} = \mathbf{e}_{k_1} + \mathbf{e}_{k_2}$, and $\mathbf{c}_1, \mathbf{c}_2$ are all-zero vectors.
   (b) For multiplication, $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2) = (\tilde{u}\mathbf{e}_{k_2} + \tilde{v}\mathbf{e}_{k_1}, \mathbf{e}_{k_1}, \mathbf{e}_{k_2})$.
2. The parties invoke the functionality $\mathcal{F}_{C_1}$ with C_1 being the circuit
$$C_1((\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2), \mathbf{s}) = \mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}) - \rho_k,$$
   where $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2)$ is the private input of P_1, and $\mathbf{s}$ and $\rho_k = s_{\beta+k}$ are provided in the form of (t, n)-sharings. We let P_1 obtain the output, denoted y.
3. P_1 computes $\tilde{w} = \tilde{u} + \tilde{v} + y$ if the computed gate is an addition gate and $\tilde{w} = \tilde{u}\tilde{v} + y$ otherwise.
4. The resulting sharing is $\langle\!\langle w \rangle\!\rangle = ((\beta + k, \tilde{w}), \langle \mathbf{s} \rangle)$.

Output recovery.

1. Let $\langle\!\langle z \rangle\!\rangle = ((k, \tilde{z}), \langle \mathbf{s} \rangle)$ be an output sharing.
2. The parties invoke the functionality $\mathcal{F}_{C_2}$ with C_2 being the circuit $\tilde{z} + \mathbf{e}_k^\top \mathbf{s}$. Note that $\tilde{z} + \mathbf{e}_k^\top \mathbf{s}$ is exactly the output z.

Theorem 5. *Protocol 1 securely computes the circuit C in the ($\mathcal{F}_{\text{rsg}}$,$\mathcal{F}_{C_1}$,$\mathcal{F}_{C_2}$)-hybrid model against any semi-honest, static adversary controlling up to t parties.*

Due to space limitations, we defer the proof to Appendix B.1.

6.4 An Optimization: Halving the Communication Cost

We present an optimization that reduces the communication cost of the circuit evaluation stage, based on the observation that for the k-th gate, its input wires always correspond to indices smaller than $\beta + k$. We reduce the communication cost by roughly a factor of two. Due to space limitations, we defer the technical details to Appendix C.

6.5 Complexity Analysis

The setup stage requires the parties to generate $\beta + m$ sharings. The input sharing stage requires the parties to compute (t, n)-sharings of their inputs and open β sharings to P_1. The circuit evaluation stage requires the parties to compute $3m^2/2 + (3\beta - 1/2)m$ secure multiplications. Finally, reconstructing one output requires computing $\beta + m$ secure multiplications. We assume that there are γ outputs. We use Shamir secret sharing as the underlying secret sharing scheme. We next analyze the asymptotic communication complexity.

Using the DN method, generating $\beta+m$ sharings requires communication $O((\beta+m)n+n^2)$. When $\beta+m \ll n$, one can instead use our LDN protocol to achieve $O((\beta + m + \kappa)n)$ communication. Computing (t, n)-sharings of β inputs and open β sharings to P_1 incur $O(\beta n)$ communication. Using the DN protocol to compute $3m^2/2 + (3\beta - 1/2)m + \gamma(\beta + m)$ multiplication gates requires communication $O((m^2 + (\beta + \gamma)m + \beta\gamma)n + n^2)$. When $m^2 + (\beta + \gamma)m + \beta\gamma \ll n$, one can instead use our LDN protocol to achieve $O((m^2 + (\beta + \gamma)m + \beta\gamma + \kappa)n)$ communication. Finally, we summarize that the asymptotic communication complexity of the protocol is $O((m^2 + (\beta + \gamma)m + \beta\gamma)n + n^2)$ or $O((m^2 + (\beta + \gamma)m + \beta\gamma + \kappa)n)$. Ignoring the contribution of inputs and outputs, the communication complexity simplifies to $O(m^2n + n^2)$ or $O((m^2 + \kappa)n)$.

6.6 Depth-Revealing PFE with Lower Communication

In [19], the authors introduce depth-revealing PFE (drPFE), which relaxes circuit privacy by allowing the circuit depth d to be leaked. They show that drPFE reduces the round complexity of the protocol from $O(m)$ to $O(d)$. By applying this idea to our PFE protocol, we show that it is possible not only to achieve $O(d)$ round complexity, but also to reduce the communication complexity from $O(m^2)$ to $O(dm)$. This represents a significant improvement when $d \ll m$.

Given d, the idea in [19] to achieve $O(d)$ round complexity is simple: in each round, the parties compute $\lceil m/(\varepsilon d)\rceil$ gates, where $\varepsilon > 0$ is an arbitrary constant. In particular, if the number of gates in a layer of the circuit is not a multiple of $\lceil m/(\varepsilon d)\rceil$, then P_1 adds some dummy gates. Note that circuit-oblivious parties (i.e., parties that do not know the circuit) cannot distinguish whether the computed gates are real or dummy. As discussed in [19], this can reduce the round complexity to $O(d)$. Suppose the j-th layer of the circuit contains m_j gates; then the total number of rounds is given by

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

The parameter ε controls the tradeoff between round complexity and computational overhead. To see this, note that the number of computed gates is

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

However, for any constant ε, the number of computed gates remains $O(m)$. Next, we show why revealing d can further reduce the communication complexity of our protocol.

Since the parties compute $\lceil m/(\varepsilon d)\rceil$ gates in each round, when evaluating a gate, the circuit-oblivious parties know that the corresponding key is the k-th key within the current round. Therefore, when computing the k-th gate in each round, the key vector only needs to include the k-th key of each layer, rather than all keys. Naturally, the length of the key vector becomes $(1 + \varepsilon)d$, which implies that computing a single gate now requires the parties to perform $O(d)$ secure multiplications. As a result, the total number of secure multiplications is $O(dm)$, and thus the communication is reduced from $O(m^2)$ to $O(dm)$.

7 Three-Party PFE with Subquadratic Communication

Our multiparty PFE construction achieves communication complexity quadratic in the circuit size. We further show that in the three-party setting, we can achieve subquadratic communication complexity. More specifically, we obtain a three-party PFE with $O(m^{4/3})$ communication.

7.1 A Simple Three-Party CESS Scheme

Our three-party PFE protocol relies on the following three-party CESS scheme.

Theorem 6. *The following three algorithms* $(\mathsf{Chain}, \mathsf{Share}, \mathsf{Rec})$ *define a perfect* $(1, 3, \alpha, 2)$*-CESS scheme over a ring* $\mathcal{X}$*.*

- $\mathsf{Chain}(\alpha)$.
  1. *Sample a random key vector* $\mathbf{s} = (s_1, \ldots, s_\alpha) \in \mathcal{X}^\alpha$.
  2. *Output the state* $\perp$ *and the fixed shares* $(x_2, x_3) = (\mathbf{s}, \mathbf{s})$.
- $\mathsf{Share}(j, x, (\perp, \mathbf{s}, \mathbf{s}))$.
  1. *Compute* $\tilde{x} = x - s_j$.
  2. *Output the share* $x_1 = (j, \tilde{x})$.
- $\mathsf{Rec}(I, (x_i)_{i \in I})$. *We choose* I *subject to* $1 \in I$.
  1. *Parse* $x_1 = (j, \tilde{x})$ *and* $x_i = \mathbf{s}$ *for some* $i \in I \setminus \{1\}$.
  2. *Output* $x = \tilde{x} + s_j$.

Proof. We prove the correctness and privacy of the scheme respectively.

Correctness. Correctness follows from the fact that P_1 holds the ciphertext, while both P_2 and P_3 hold the key vactor. Therefore, P_1 together with either P_2 or P_3 can reconstruct the secret.

Privacy. Privacy follows from the fact that each individual party holds either only the ciphertext or only the key vector, and therefore learns no information about the secret. □

7.2 The Protocol

Protocol 2. Three-Party PFE

Parameter. Let C be the computed circuit over a field $\mathbb{F}$. Let β be the number of inputs, and m be the number of gates. Set $\alpha = \beta + m$.

Setup.

1. P_2 samples a random vector $\mathbf{s} = (s_1, \ldots, s_\alpha) \in \mathbb{F}^\alpha$ and sends it to P_3.
2. P_2 samples 2α random values $r_{1,1}, \ldots, r_{1,\alpha}, r_{2,1}, \ldots, r_{2,\alpha} \in \mathbb{F}$ and sends them to P_3.

Input sharing.

1. Let $\mathbf{x} = (x_1, \ldots, x_\beta) \in \mathbb{F}^\beta$ be the input vector.
2. For each input x_k, the parties perform the following steps.
   (a) If x_k is an input of P_1, then P_2 sends s_k to P_1, and P_1 computes $\tilde{x}_k = x_k - s_k$.
   (b) If x_k is an input of P_i with $i \neq 1$, then P_i sends $\tilde{x}_k = x_k - s_k$ to P_1.
3. The sharing of each input x_k is $\langle\!\langle x_k \rangle\!\rangle = ((k, \tilde{x}_k), \mathbf{s}, \mathbf{s})$.

Circuit evaluation.
For the k-th gate, let $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \mathbf{s}, \mathbf{s})$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \mathbf{s}, \mathbf{s})$ be the input sharings and the parties compute this gate as follows.

1. P_2 and P_3 each compute $s_j^{(1)} = s_j - r_{1,k}$ for all $j \in [\alpha]$. Denote $\mathbf{s}^{(1)} = (s_1^{(1)}, \ldots, s_m^{(1)})$.

2. P_2 and P_3 each compute $s_j^{(2)} = s_j - r_{2,k}$ for all $j \in [\alpha]$. Denote $\mathbf{s}^{(2)} = (s_1^{(2)}, \ldots, s_m^{(2)})$.
3. P_1 and P_2, P_3 invoke the SPIR functionality $\mathcal{F}_{\text{spir}}^2$ twice, where P_1 acts as the client and P_2, P_3 act as the two non-colluding servers. In the b-th invocation, P_1 takes index k_b as input, while P_2 and P_3 use $\mathbf{s}^{(b)}$ as the database, which contains α elements. As a result, P_1 obtains $s_{k_b}^{(b)} = s_{k_b} - r_{b,k}$ from the b-th invocation.
4. The parties invoke the functionality $\mathcal{F}_{C_1}$ with C_1 being the circuit

$$C_1((c_1, c_2, c), s_{k_1}, s_{k_2}, s_k) = c_1 s_{k_1} + c_2 s_{k_2} + c s_{k_1} s_{k_2} - s_{\beta+k},$$

where (c_1, c_2, c) is the private input of P_1. For an addition gate, P_1 sets $(c_1, c_2, c) = (1, 1, 0)$, and for a multiplication gate, P_1 sets $(c_1, c_2, c) = (\tilde{v}, \tilde{u}, 1)$. We let P_1 obtain the output, denoted y.
5. P_1 computes $\tilde{w} = \tilde{u} + \tilde{v} + y$ if the computed gate is an addition gate and $\tilde{w} = \tilde{u}\tilde{v} + y$ otherwise.
6. The resulting sharing is $\langle\!\langle w \rangle\!\rangle = ((\beta + k, \tilde{w}), \mathbf{s}, \mathbf{s})$.

Output recovery.

1. Let $\langle\!\langle z \rangle\!\rangle = ((k, \tilde{z}), \mathbf{s}, \mathbf{s})$ be an output sharing.
2. P_1 and P_2, P_3 invoke the SPIR functionality $\mathcal{F}_{\text{spir}}^2$, where P_1 acts as the client with index k as input, and P_2, P_3 act as the two non-colluding servers with database specified as follows.
   (a) If P_1 wishes to obtain the output, then P_2, P_3 simply take $\mathbf{s}$ as the database.
   (b) If P_2 or P_3 wishes to obtain the output, then P_2 samples a random value r and sends it to P_3. They then take $\mathbf{s}' = (s_1 - r, \ldots, s_\alpha - r)$ as the database.
3. As a result, P_1 obtains s_k if it wishes to obtain the output, and obtains $s'_k = s_k - r$ otherwise.
4. If P_1 wishes to obtain the output, it directly outputs $z = \tilde{z} + s_k$; if P_i (with $i \neq 1$) wishes to obtain the output, then P_1 sends $z' = \tilde{z} + s'_k$ to P_i, and P_i outputs $z' + r$.

Theorem 7. *Protocol 2 securely computes the circuit C in the ($\mathcal{F}_{\text{spir}}^2$,$\mathcal{F}_{C_1}$)-hybrid model against any semi-honest, static adversary controlling up to one party.*

Due to space limitations, we defer the proof to Appendix B.2.

7.3 Complexity Analysis

The setup stage requires the parties to send 3α elements. The input sharing stage requires the parties to send β elements. In the circuit evaluation stage, each gate requires the parties to invoke the SPIR functionality $\mathcal{F}_{\text{spir}}^2$ twice and to evaluate a circuit consisting of 4 multiplication gates. Finally, reconstructing one output requires a single invocation of the functionality $\mathcal{F}_{\text{spir}}^2$. In particular, if P_1 obtains the output, then no additional communication is needed; otherwise, the parties need to send 2 additional elements. We assume that there are γ outputs.

We use the two-server SPIR protocol in [11] to realize the functionality $\mathcal{F}_{\text{spir}}^2$, which achieves communication complexity and shared randomness complexity of $O(\alpha^{1/3})$ for a database of size α. Note that the shared randomness can be realized by having P_2 sample the randomness and send it to P_3. Therefore, the communication complexity of the two-server SPIR protocol in [11] can be regarded as $O(\alpha^{1/3})$. Furthermore, the AFLNO protocol [3] is used to evaluate the involved circuit. Since the circuit contains only a constant number of gates, their communication complexity is $O(1)$. The total communication cost of the protocol is therefore

$$3\alpha + \beta + m \cdot O(\alpha^{1/3}) + \gamma \cdot O(\alpha^{1/3}) = O(\beta + (m + \gamma)(\beta + m)^{1/3}).$$

Ignoring the contribution of inputs and outputs, the communication complexity simplifies to $O(m^{4/3})$.

7.4 Extending to Any Number of Parties

Our three-party PFE protocol can be extended to an arbitrary number of parties while tolerating a single corruption. The key idea is to modify the form of the input sharing from $((\beta + k, \tilde{w}), \mathbf{s}, \mathbf{s})$ to $((\beta + k, \tilde{w}), \mathbf{s}, \ldots, \mathbf{s})$, i.e., all parties

except P_1 hold the key vector. Clearly, any single corrupted party learns no information about the secret. Indeed, this sharing structure actually withstands a stronger adversarial model: the adversary either corrupts only P_1, or corrupts any subset of at most t parties (for achieving information-theoretic security) among those excluding P_1.

The only difference between the resulting n-party protocol and our three-party PFE protocol is that we now require an $(n-1)$-server SPIR scheme instead of a two-server SPIR scheme. In [11], the authors show that for any constant $h \geq 2$, there exists an information-theoretic h-server SPIR protocol with communication complexity and shared randomness complexity $O(\alpha^{1/(2h-1)})$. Note that since there are h servers, generating the shared randomness requires $O(\alpha^{1/(2h-1)}h)$ communication. By applying this construction within our PFE protocol, if ignoring the contribution of inputs and outputs, we obtain a communication complexity of $O(m \cdot m^{1/(2(n-1)-1)}n) = O(m^{(2n-2)/(2n-3)}n)$.

Finally, the work [11] also presents an $O(\log \alpha)$-server SPIR protocol with communication $O(\log^2 \alpha \log\log \alpha)$. This construction requires the servers to share $O(\log^2 \alpha \log\log \alpha)$ random values. If we allow the number of parties n scales with the circuit size m and satisfies $n = O(\log m)$, then the communication becomes quasilinear: $m \cdot O(n \log^2 m \log\log m) = O(m \log^3 m \log\log m)$.

8 Conclusion

We presented new information-theoretic protocols for both SFE and PFE for arithmetic circuits in the semi-honest setting. For SFE, we revisited the classical DN protocol and showed how to eliminate its residual quadratic communication overhead, obtaining communication complexity $O(m^*n)$ for circuits with m^* multiplication gates. To the best of our knowledge, this is the first asymptotic improvement over the DN protocol for small circuits since its introduction. We further developed new techniques for information-theoretic PFE that avoid universal circuits entirely. This leads to an honest-majority protocol with communication complexity $O(m^2n)$ for arithmetic circuits of size m, together with stronger bounds in special settings, including an $O(m^{4/3})$ three-party protocol and an n-party protocol tolerating one corruption with communication $O(m^{(2n-2)/(2n-3)}n)$.

In the computational setting, prior work of Li et al. [19] shows that PFE can asymptotically match SFE, achieving linear communication in both the circuit size and the number of parties. In contrast, a substantial gap between PFE and SFE has remained in the information-theoretic setting. Our work takes a significant step toward closing this gap, reducing the communication complexity from the previous $O(m^5n)$ bound incurred by universal-circuits-based approaches to $O(m^2n)$, and further to $O(m^{4/3})$ in the three-party setting. Our results make us optimistic that achieving information-theoretic PFE with linear (or near-linear $m^{1+o(1)}$) communication is feasible. We leave this as an important open problem.

References

1. Abadi, M., Feigenbaum, J.: Secure circuit evaluation. Journal of Cryptology **3**(1), 1–12 (1990)
2. Alhassan, M.Y., Günther, D., Kiss, Á., Schneider, T.: Efficient and scalable universal circuits. Journal of Cryptology **33**(3), 1216–1271 (2020)
3. Araki, T., Furukawa, J., Lindell, Y., Nof, A., Ohara, K.: High-throughput semi-honest secure three-party computation with an honest majority. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 805–817 (2016)
4. Barni, M., Failla, P., Kolesnikov, V., Lazzeretti, R., Sadeghi, A.R., Schneider, T.: Secure evaluation of private linear branching programs with medical applications. In: European Symposium on Research in Computer Security (ESORICS). pp. 424–439 (2009)
5. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Annual ACM Symposium on Theory of Computing (STOC). pp. 1–10 (1988)
6. Biçer, O., Bingöl, M.A., Kiraz, M.S., Levi, A.: Highly efficient and re-executable private function evaluation with linear complexity. IEEE Transactions on Dependable and Secure Computing **19**(2), 835–847 (2022)
7. Brickell, J., Porter, D.E., Shmatikov, V., Witchel, E.: Privacy-preserving remote diagnostics. In: ACM Conference on Computer and Communications Security (CCS). pp. 498–507 (2007)
8. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: IEEE Annual Foundations of Computer Science (FOCS). pp. 41–50 (1995)
9. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference (CRYPTO). pp. 572–590 (2007)
10. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. In: Annual ACM Symposium on Theory of Computing (STOC). pp. 151–160 (1998)
11. Gertner, Y., Ishai, Y., Kushilevitz, E., Malkin, T.: Protecting data privacy in private information retrieval schemes. Journal of Computer and System Sciences **60**(3), 592–629 (2000)
12. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Annual ACM Symposium on Theory of Computing (STOC). pp. 218–229 (1987)
13. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: Atlas: Efficient and scalable mpc in the honest majority setting. In: Annual International Cryptology Conference (CRYPTO). pp. 244–274 (2021)
14. Goyal, V., Song, Y., Zhu, C.: Guaranteed output delivery comes free in honest majority mpc. In: Annual International Cryptology Conference (CRYPTO). pp. 618–646 (2020)
15. Günther, D., Ágnes Kiss, Scheidel, L., Schneider, T.: Poster: Framework for semi-private function evaluation with application to secure insurance rate calculation. In: ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 2541–2543 (2019)
16. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (Part III: Fundamental Electronic Science) **72**(9), 56–64 (1989)
17. Jia, H., Li, X.: Pfe: Linear active security, double-shuffle proofs, and low-complexity communication. In: International Conference on Information Security and Cryptology (Inscrypt). pp. 227–252 (2023)
18. Katz, J., Malka, L.: Constant-round private function evaluation with linear complexity. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 556–571 (2011)
19. Li, S., Zhang, C., Wang, A., Wang, X.: Private function evaluation with linear complexity. To appear in CRYPTO 2026 (2026)
20. Liu, Y., Wang, Q., Yiu, S.M.: Making private function evaluation safer, faster, and simpler. In: International Conference on Practice and Theory of Public-Key Cryptography (PKC). pp. 349–378 (2022)
21. Mohassel, P., Sadeghian, S.S.: How to hide circuits in mpc an efficient framework for private function evaluation. In: International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 557–574 (2013)
22. Mohassel, P., Sadeghian, S.S., Smart, N.P.: Actively secure private function evaluation. In: International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT). pp. 486–505 (2014)
23. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT). pp. 223–238 (1999)
24. Raz, R.: Elusive functions and lower bounds for arithmetic circuits. In: Annual ACM Symposium on Theory of Computing (STOC). pp. 711–720 (2008)
25. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
26. Shpilka, A., Yehudayoff, A.: Arithmetic circuits: A survey of recent results and open questions. Foundations and Trends in Theoretical Computer Science **5**(3-4), 207–388 (2010)
27. Valiant, L.G.: Universal circuits (preliminary report). In: Annual ACM Symposium on Theory of Computing (STOC). pp. 196–203 (1976)
28. Xu, Y., Jia, H., Li, X., Li, Q., Bao, Y., Hou, X.: Constant-round multiparty private function evaluation with (quasi-)linear complexities. In: Applied Cryptography and Network Security (ACNS). pp. 115–142 (2023)
29. Yao, A.C.: Protocols for secure computations. In: Annual Symposium on Foundations of Computer Science (FOCS). pp. 160–164 (1982)

A Secret Sharing

We recall the definition of secret sharing, with a focus on perfect security.

Definition 4 (Secret Sharing). *A (t, n)-secret sharing scheme over a secret space $\mathcal{X}$ consists of two PPT algorithms* $(\mathsf{Share}, \mathsf{Rec})$ *with the following syntax.*

- $\mathsf{Share}(x)$. *On input a secret $x \in \mathcal{X}$, output n shares $(x_1, \ldots, x_n)$.*
- $\mathsf{Rec}(I, x_I)$. *On input an index set I with $|I| > t$ and the shares $x_I = \{x_i\}_{i \in I}$ in this set, output a value $x \in \mathcal{X}$.*

These algorithms satisfy the following properties.

- ***Correctness.*** *For every $x \in \mathcal{X}$ and every index set $I \subseteq [n]$ with $|I| > t$, it holds that $\Pr[\mathsf{Rec}(I, x_I) = x : (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x)] = 1$.*
- ***Privacy.*** *For every $x \in \mathcal{X}$ and every index set $I \subseteq [n]$ with $|I| \leq t$, it holds that the distribution $\{x_I : (x_1, \ldots, x_n) \leftarrow \mathsf{Share}(x)\}$ is independent of the secret x.*

B Missing Security Proofs

B.1 Security Proof of Protocol 1

Proof. **Correctness.** We prove that the protocol correctly computes the circuit. In the input sharing stage, each input x_k is represented as the sharing $((k, \tilde{x}_k), \langle \mathbf{s} \rangle)$ with $\mathbf{s} = (s_1, \ldots, s_\alpha)$. This sharing satisfies $\tilde{x}_k + s_k = x_k$.

In the circuit evaluation stage, given input sharings $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \langle \mathbf{s} \rangle)$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \langle \mathbf{s} \rangle)$, the parties output the sharing $\langle\!\langle w \rangle\!\rangle = ((\beta + k, \tilde{w}), \langle \mathbf{s} \rangle)$. We now show that when evaluating an addition gate, $\tilde{w} = x + y - s_{\beta+k}$, while when evaluating a multiplication gate, $\tilde{w} = xy - s_{\beta+k}$.

In the case of an addition gate, we have $\tilde{w} = \tilde{u} + \tilde{v} + y$, where $y = \mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}) - \rho_k$. Since P_1 sets $\mathbf{h} = \mathbf{e}_{k_1} + \mathbf{e}_{k_2}$ and $\mathbf{c}_1, \mathbf{c}_2$ as all-zero vectors when evaluating an addition gate, it follows that

$$y = (\mathbf{e}_{k_1}^\top + \mathbf{e}_{k_2}^\top)\mathbf{s} - \rho_k = s_{k_1} + s_{k_2} - s_{\beta+k}.$$

Then, we have

$$\tilde{w} = \tilde{u} + \tilde{v} + y = \tilde{u} + \tilde{v} + s_{k_1} + s_{k_2} - s_{\beta+k} = u + v - s_{\beta+k}.$$

In the case of a multiplication gate, we have $\tilde{w} = \tilde{u}\tilde{v} + y$, where $y = \mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}) - \rho_k$. Since P_1 sets $(\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2) = (\tilde{u}\mathbf{e}_{k_2} + \tilde{v}\mathbf{e}_{k_1}, \mathbf{e}_{k_1}, \mathbf{e}_{k_2})$ when evaluating a multiplication gate, it follows that

$$\begin{aligned} y &= (\tilde{u}\mathbf{e}_{k_2} + \tilde{v}\mathbf{e}_{k_1})^\top \mathbf{s} + (\mathbf{e}_{k_1}^\top \mathbf{s})(\mathbf{e}_{k_2}^\top \mathbf{s}) - \rho_k \\ &= \tilde{u}s_{k_2} + \tilde{v}s_{k_1} + s_{k_1}s_{k_2} - s_{\beta+k} \\ &= (\tilde{u} + s_{k_1})(\tilde{v} + s_{k_2}) - \tilde{u}\tilde{v} - s_{\beta+k} \\ &= uv - \tilde{u}\tilde{v} - s_{\beta+k}. \end{aligned}$$

Then, we have

$$\tilde{w} = \tilde{u}\tilde{v} + y = \tilde{u}\tilde{v} + uv - \tilde{u}\tilde{v} - s_{\beta+k} = uv - s_{\beta+k}.$$

Privacy. We now argue the privacy of the protocol. To this end, we show that there exists a simulator that can generate the view of corrupted parties. We consider two cases, depending on whether P_1 is honest.

$\diamond$ *P_1 is corrupted.* Note that the setup stage does not involve any inputs, and can therefore be perfectly simulated. In the input sharing stage, the messages received by corrupted parties from honest parties include the shares of the corrupted parties when honest parties distribute their input sharings, as well as the shares sent by honest parties when reconstructing $\tilde{\mathbf{x}}$ at P_1. The security of secret sharing guarantees that the corrupted parties' shares are independent of the underlying inputs, and thus the simulator can perfectly simulate these shares even without knowing the inputs. Moreover, since $\langle \boldsymbol{\sigma} \rangle$ is generated via the random sharing functionality $\mathcal{F}_{\mathsf{rsg}}$, and we have $\langle \tilde{\mathbf{x}} \rangle \leftarrow \langle \mathbf{x} \rangle - \langle \boldsymbol{\sigma} \rangle$, it follows that $\langle \tilde{\mathbf{x}} \rangle$ is itself a random sharing. In the ideal execution, the simulator simply samples a random sharing and simulates its opening to P_1.

In the circuit evaluation stage, when evaluating each gate, the only message received by the corrupted parties is the value y that P_1 obtains from the functionality $\mathcal{F}_{C_1}$. According to the protocol execution, y equals either $\tilde{w} - \tilde{u} - \tilde{v}$

or $\tilde{w} - \tilde{u}\tilde{v}$. Moreover, $\tilde{w} = w - s_{\beta+k}$. Since $s_{\beta+k}$ is a random value unknown to the adversary, it follows that y is also uniformly random. Therefore, the simulator can simply sample a random value and simulate $\mathcal{F}_{C_1}$ sending it to P_1.

Finally, in the output recovery stage, when a corrupted party obtains an output z, the simulator (whose input now includes this output z) simply simulates the functionality $\mathcal{F}_{C_2}$ sending z to that party. When an honest party obtains the output, no simulation is needed, since the corrupted parties receive no messages.

$\diamond$ *P_1 is honest.* Still, the setup stage does not involve any inputs, and can therefore be perfectly simulated. In the input sharing stage, the messages received by corrupted parties from honest parties include the shares of the corrupted parties when honest parties distribute their input sharings. The security of secret sharing guarantees that the corrupted parties' shares are independent of the underlying inputs, and thus the simulator can perfectly simulate these shares even without knowing the inputs. In the circuit evaluation stage, since P_1 is honest, the corrupted parties receive no messages, and therefore no simulation is needed. Moreover, the simulation of the output recovery stage is identical to the case where P_1 is corrupted. □

B.2 Security Proof of Protocol 2

Proof. **Correctness.** We prove that the protocol correctly computes the circuit. In the input sharing stage, each input x_k is represented as the sharing $((k, \tilde{x}_k), \mathbf{s}, \mathbf{s})$ with $\mathbf{s} = (s_1, \ldots, s_\alpha)$. This sharing satisfies $\tilde{x}_k + s_k = x_k$.

In the circuit evaluation stage, given input sharings $\langle\!\langle u \rangle\!\rangle = ((k_1, \tilde{u}), \mathbf{s}, \mathbf{s})$ and $\langle\!\langle v \rangle\!\rangle = ((k_2, \tilde{v}), \mathbf{s}, \mathbf{s})$, the parties output the sharing $\langle\!\langle w \rangle\!\rangle = ((\beta+k, \tilde{w}), \mathbf{s}, \mathbf{s})$. We now show that when evaluating an addition gate, $\tilde{w} = x+y-s_{\beta+k}$, while when evaluating a multiplication gate, $\tilde{w} = xy - s_{\beta+k}$.

In the case of an addition gate, we have $\tilde{w} = \tilde{u} + \tilde{v} + y$, where $y = c_1 s_{k_1} + c_2 s_{k_2} + c s_{k_1} s_{k_2} - s_{\beta+k}$. Since P_1 sets $(c_1, c_2, c) = (1, 1, 0)$ when evaluating an addition gate, it follows that $y = s_{k_1} + s_{k_2} - s_{\beta+k}$. Then, we have

$$\tilde{w} = \tilde{u} + \tilde{v} + y = \tilde{u} + \tilde{v} + s_{k_1} + s_{k_2} - s_{\beta+k} = u + v - s_{\beta+k}.$$

In the case of a multiplication gate, we have $\tilde{w} = \tilde{u}\tilde{v} + y$, where $y = c_1 s_{k_1} + c_2 s_{k_2} + c s_{k_1} s_{k_2} - s_{\beta+k}$. Since P_1 sets $(c_1, c_2, c) = (\tilde{v}, \tilde{u}, 1)$ when evaluating a multiplication gate, it follows that

$$\begin{aligned} y &= \tilde{u} s_{k_2} + \tilde{v} s_{k_1} + s_{k_1} s_{k_2} - s_{\beta+k} \\ &= (\tilde{u} + s_{k_1})(\tilde{v} + s_{k_2}) - \tilde{u}\tilde{v} - s_{\beta+k} \\ &= uv - \tilde{u}\tilde{v} - s_{\beta+k}. \end{aligned}$$

Then, we have

$$\tilde{w} = \tilde{u}\tilde{v} + y = \tilde{u}\tilde{v} + uv - \tilde{u}\tilde{v} - s_{\beta+k} = uv - s_{\beta+k}.$$

Privacy. We now argue the privacy of the protocol. To this end, we show that there exists a simulator that can generate the view of corrupted parties. We consider two cases, depending on whether P_1 is honest.

$\diamond$ *P_1 is corrupted.* Note that the setup stage does not involve any inputs, and can therefore be perfectly simulated. In the input sharing stage, P_1 receives s_k when sharing its own input, and receives $\tilde{x}_k = x_k - s_k$ when sharing the inputs of other parties. In both cases, the value received by P_1 is uniformly random, since s_k is a random value shared by P_2 and P_3. Thus, the simulator can perfectly simulate the input sharing stage.

In the circuit evaluation stage, when evaluating each gate, the only messages received by P_1 are the values $s_{k_b}^{(b)} = s_{k_b} - r_{b,k}$ for $b \in \{1, 2\}$ from the functionality $\mathcal{F}_{\text{spir}}^2$, and the value y from the functionality $\mathcal{F}_{C_1}$. For $s_{k_b}^{(b)}$, since $r_{b,k}$ is a random value shared by P_2 and P_3 and is unknown to the adversary, the value $s_{k_b}^{(b)}$ is uniformly random from the adversary's point of view. Hence, the simulator can simply sample a random value $s_{k_b}^{(b)}$ and simulate $\mathcal{F}_{\text{spir}}^2$ sending it to P_1. For y, according to the protocol execution, y equals either $\tilde{w} - \tilde{u} - \tilde{v}$ or $\tilde{w} - \tilde{u}\tilde{v}$, where $\tilde{w} = w - s_{\beta+k}$. Since $s_{\beta+k}$ is a random value unknown to the adversary, it follows that y is also uniformly random. Therefore, the simulator can simply sample a random value y and simulate $\mathcal{F}_{C_1}$ sending it to P_1.

Finally, in the output recovery stage, when P_1 obtains an output z, the only message it receives is s_k from the functionality $\mathcal{F}_{\text{spir}}^2$. Since $z = \tilde{z} + s_k$, the simulator can compute $s_k = z - \tilde{z}$ and simulate $\mathcal{F}_{\text{spir}}^2$ sending s_k to P_1, where $\tilde{z}$ is the simulated ciphertext of the output. When P_2 or P_3 obtains an output, the message received by P_1 is $s'_k = s_k - r$ from the functionality $\mathcal{F}_{\text{spir}}^2$. Since r is a random value shared between P_2 and P_3 and unknown to the adversary, s'_k is uniformly random from the adversary's perspective. Therefore, the simulator can simply sample a random value s'_k and simulate $\mathcal{F}_{\text{spir}}^2$ sending it to P_1.

$\diamond$ P_1 *is honest.* Let P_i be the corrupted party for $i \in \{2,3\}$. Still, the setup stage does not involve any inputs and can therefore be perfectly simulated. In the input sharing and circuit evaluation stages, P_i receives no messages, and thus no simulation is needed. In the output recovery stage, when P_i obtains an output z, it receives from P_1 the value $z' = \tilde{z} + s'_k$. Since $z' + r$ equals the output z, and r is a random value shared between P_i and P_{5-i}, the simulator can directly simulate P_1 sending $z - r$ to P_i, where r is the random value shared between P_i and P_{5-i} in the ideal execution. □

C Halving the Communication Cost of Our Multiparty PFE

The optimization is based on the observation that for the k-th gate with input indices k_1, k_2, we always have $k_1 < \beta + k$ and $k_2 < \beta + k$. Note that each gate requires the parties to evaluate the circuit

$$f((\mathbf{h}, \mathbf{c}_1, \mathbf{c}_2), \mathbf{s}) = \mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}) - \rho_k$$

For an addition gate, the output of this circuit is $\mathbf{e}_{k_1}^\top \mathbf{s} + \mathbf{e}_{k_2}^\top \mathbf{s} - \rho_k$, and for a multiplication gate, the output of the circuit is $\tilde{u}\mathbf{e}_{k_2}^\top \mathbf{s} + \tilde{v}\mathbf{e}_{k_1}^\top \mathbf{s} + (\mathbf{e}_{k_1}^\top \mathbf{s})(\mathbf{e}_{k_2}^\top \mathbf{s}) - \rho_k$. Here, the inner products $\mathbf{e}_{k_1}^\top \mathbf{s}$ and $\mathbf{e}_{k_2}^\top \mathbf{s}$ are used to compute the keys s_{k_1} and s_{k_2}, respectively. Since $k_1 < \beta + k$ and $k_2 < \beta + k$, we can instead use shorter inner products. Let $\mathbf{s}^{(k-1)}$ denote the vector consisting of the first $\beta + k - 1$ elements of $\mathbf{s}$. Then we can use the inner products $\mathbf{e}_{k_1}^\top \mathbf{s}^{(k-1)}$ and $\mathbf{e}_{k_2}^\top \mathbf{s}^{(k-1)}$ are used to compute the keys s_{k_1} and s_{k_2}, respectively (for notational simplicity, we assume that the unit vectors always have the appropriate length, i.e., $\beta + k - 1$).

Originally, for the k-th gate, the parties were required to compute the function $\mathbf{h}^\top \mathbf{s} + (\mathbf{c}_1^\top \mathbf{s})(\mathbf{c}_2^\top \mathbf{s}) - \rho_k$. Now, we can modify this function to $\mathbf{h}^\top \mathbf{s}^{(k-1)} + (\mathbf{c}_1^\top \mathbf{s}^{(k-1)})(\mathbf{c}_2^\top \mathbf{s}^{(k-1)}) - \rho_k$. The original function involves $3(\beta+m)+1$ multiplications, so the protocol in total requires $(3(\beta + m) + 1) \cdot m = 3m^2 + (3\beta + 1)m$ secure multiplications. In contrast, the optimized function for the k-th gate only involves $3(\beta + k - 1) + 1$ multiplications. Therefore, the total number of secure multiplications required by the protocol becomes

$$\sum_{k \in [m]} 3(\beta + k - 1) + 1 = 3m^2/2 + (3\beta - 1/2)m.$$

Clearly, we reduce the communication cost by a factor of about 2 (assuming $m \gg \beta$).