Weighted Cryptography with Weight-Independent Complexity

Aarushi Goel* Swagata Sasmal† Mingyuan Wang‡

Abstract

Cryptographic primitives involving multiple participants, such as secure multiparty computation (MPC), threshold signatures, and threshold encryption, are typically designed under the assumption that at least a threshold number of participants remain honest and non-colluding. However, many real-world applications require more expressive access structures beyond simple thresholds. A prominent example is the weighted threshold access structure, where each party is assigned a weight and security holds as long as the total weight of corrupted parties does not exceed a specified threshold.

Despite the practical relevance of such access structures, our understanding of efficient constructions supporting them remains limited. For instance, existing approaches for weighted MPC and weighted threshold encryption incur costs that scale with the total assigned weights to all parties or rely on non-black-box use of cryptography.

In this work, we present the first black-box constructions of the following weighted cryptosystems with weight-independent complexity in the trusted setup model: (i) a weighted MPC protocol with guaranteed output delivery, (ii) a semi-honest weighted threshold encryption scheme and (iii) a semi-honest weighted threshold Schnorr signature scheme.

At the heart of our constructions is a new *succinct computational secret sharing scheme with linear homomorphism* for weighted threshold access structures. We provide two concrete instantiations of this primitive, based on the Decisional Composite Residuosity (DCR) assumption and the Learning With Errors (LWE) assumption, respectively. Furthermore, our constructions extend to any general access structure that can be represented efficiently as a monotone Boolean circuit.

*Rutgers University aarushi.goel@rutgers.edu
†Indian Statistical Institute swagata.sasmal@gmail.com
‡New York University, Shanghai mingyuan.wang@nyu.edu

Contents

1 Introduction **3**
- 1.1 Our Results . . . 4

2 Technical Overview **6**
- 2.1 Weighted Secret Sharing . . . 6
- 2.2 Weighted Cryptosystems . . . 10
- 2.3 Concurrent Work and Further Improvement . . . 12

3 Special-Purpose Linearly Homomorphic Encryption **12**
- 3.1 Overview . . . 12
- 3.2 Definition . . . 16
- 3.3 Instantiation Based on LWE . . . 18
- 3.4 Instantiation Based on DCR . . . 21

4 Computational Secret Sharing with Linear Homomorphism **23**
- 4.1 Basics of Monotone Circuits and Access Structures . . . 23
- 4.2 Definition . . . 23
- 4.3 Construction . . . 25

5 Weighted Secure Multiparty Computation **30**
- 5.1 Definition . . . 31
- 5.2 Construction . . . 31

6 Weighted Threshold Encryption **35**
- 6.1 Definition . . . 36
- 6.2 Construction . . . 37

7 Weighted Threshold Schnorr Signatures **40**
- 7.1 Definition . . . 40
- 7.2 Construction . . . 42

References **44**

1 Introduction

Secure multiparty computation (MPC) protocols [Yao86, GMW87, BGW88, CCD88] enable a group of mutually distrusting parties to jointly compute a function on their private inputs without revealing anything beyond the output of the computation. Security in most existing MPC constructions is based on a *threshold trust assumption*, that is, at least a certain number of parties remain honest and non-colluding.

Other related threshold cryptosystems [Des88, DF90, Fra90, DDFY94] also typically rely on the same trust assumption. For example, threshold signatures [KG21, CKM23] allow a secret signing key to be distributed among multiple parties such that a valid signature can be produced if and only if at least a threshold number of parties participate honestly. Similarly, threshold encryption distributes a decryption key across multiple parties, ensuring that a ciphertext can be decrypted only when a threshold number of parties cooperate. Together with MPC, these primitives have proved to be useful in a wide range of decentralized settings.

Despite their utility, the threshold assumption is restrictive. Many real-world applications require more expressive access structures, where the set of *qualified* parties cannot be described merely as "t-out-of-n". A prominent example is the *weighted threshold access structure*. In this model, each party is assigned a weight, and security is preserved as long as the adversary controls parties whose total weight does not exceed a specified threshold.

Weighted thresholds arise naturally in practice. In stake-based blockchains [KRDO17], consensus protocols often assign voting power in proportion to participants' stake or resources; here, weighted threshold cryptography ensures security unless adversaries control a sufficiently large fraction of the total stake. In electronic voting systems [BVF+25], votes may carry different weights, as in shareholder voting, board elections, or multi-constituency elections. Weighted threshold schemes also appear in federated learning [ZMGN20, XLL+24, KNK+25, AIdK23], where participants contribute datasets of varying size or quality, and in corporate settings, where different organizations hold unequal levels of authority or trust. In all these cases, weighted threshold variants of the above traditional multiparty cryptographic primitives provide a more accurate model of the underlying trust distribution and better capture the desired security and robustness requirements.

Despite the importance of these more expressive access structures, existing constructions of secure multiparty computation protocols for weighted threshold access structures are limited. A standard technique, known as *player virtualization*, generically allows threshold-based MPC to be used in the weighted setting by replicating each party according to its assigned weight. However, this results in MPC protocols whose complexity scales with the total weight, which is highly undesirable. In practice, these weights could be as large as a few million, which makes this approach concretely infeasible. In theory, the weight could be as large as n^n [Mur71, Hås94], meaning that the resulting protocol does not even have a polynomial complexity.

The works of [GJM+23] and [BW06] are the only exceptions that deviate from the player-virtualization approach. However, the complexity of their protocols either still grows linearly in the weights or is superpolynomial (in the number of parties). On the other hand, if one is willing to make non-black-box use of cryptographic primitives, there exist protocols whose complexity does not grow[1] with the weights [Yao89, DJWW25].[2] Surprisingly, as far as we know, even with access to strong primitives such as

[1] Strictly speaking, the complexity grows logarithmically in the weights. However, since the weights are upper bounded by n^n, their logarithm is at most $n \log n$. In this work, we regard such dependence as effectively independent of the weights.

[2] As far as we know, non-black-box construction based on Yao's computational secret sharing [Yao89] is not described anywhere in the literature. We sketch this simple observation in Section 2.

fully homomorphic encryption [Gen09] or indistinguishability obfuscation [BGI+12], black-box constructions of weighted MPC with weight-independent complexity are still unknown.

This motivates the main question that we address in this work:

Can we achieve weight-independent complexity *for weighted secure multiparty computation protocols while only relying on black-box use of cryptography?*

This Work. This work answers this question affirmatively. We present the first black-box construction of a weighted MPC protocol in the trusted setup model with guaranteed output delivery and weight-independent complexity. We further show how our techniques extend to obtain the first black-box constructions of semi-honest *weighted threshold encryption* and semi-honest *weighted threshold Schnorr signature* schemes, both with weight-independent complexity. All of our constructions can be extended to more general monotone access structures as long as they can be described by an efficient monotone Boolean circuit. The complexity of the resulting scheme scales only with the circuit size.

1.1 Our Results

We now elaborate upon our results in more detail.

Computational Weighted Secret Sharing Scheme. At the core of our results lies a novel computational secret sharing scheme for weighted threshold access structures. The computation and communication costs of our scheme scale only with the number of parties, and remain independent of the total weights assigned. Unlike prior approaches [Yao89], our scheme supports a *special form of linear homomorphism*,[3] which makes it particularly well-suited for use in the design of secure multiparty computation protocols.

Our construction builds on Yao's computational secret sharing framework [Yao89] for general access structure described by a monotone Boolean circuit (Definition 5), augmented with a *special-purpose* linearly homomorphic encryption scheme. We show how this encryption scheme can be instantiated under the Learning With Errors (LWE) assumption and the Decisional Composite Residuosity (DCR) assumption. Together with the known polynomial-size monotone Boolean circuit describing a weighted threshold access structure [BW06], we obtain the following result:

(Informal) Theorem 1. *Assuming either the Learning With Errors (LWE) assumption or the Decisional Composite Residuosity (DCR) assumption, there exists a black-box, computational secret sharing scheme for weighted threshold access structures that supports linear homomorphism, where computation and communication scale only (in a fixed polynomial) with the number of parties and are independent of the total assigned weights.*

This result is presented in Section 4.

Beyond Weighted Threshold Access Structure. As with Yao's original scheme, our secret sharing construction naturally extends to any general access structure that can be represented efficiently as a monotone Boolean circuit, and the resulting secret sharing scheme will have a complexity growing linearly

[3] Standard linear secret sharing schemes (e.g., Shamir's scheme [Sha79]) provide *strong linear homomorphism*. This means they achieve both (1) *strong correctness*, allowing *unbounded number of additions*, and (2) *strong security*, ensuring that the resulting secret shares *reveal nothing beyond the sum of the underlying secrets*. In contrast, our notion of linear homomorphism is more restricted, supporting only weaker forms of correctness and security. We defer the precise technical definitions to later sections, but emphasize that these relaxed conditions suffice for applications in MPC and threshold cryptography, as we demonstrate in this work.

in the circuit size. This point applies to both our secret sharing scheme and its applications in MPC and threshold cryptography. We do not repeat it below.

Weighted Secure Multiparty Computation Protocol. Next, we demonstrate how the above computational secret sharing scheme for weighted thresholds can be used to design a black-box weighted MPC protocol with guaranteed output delivery[4] and communication/computation complexity that are independent of the total assigned weights. As noted earlier, all prior constructions of weighted MPC either incur complexity that scales with the total assigned weights or rely on non–black-box use of cryptography. In our protocol, we assume the parties to have access to correlated randomness that can be generated on-the-fly. Let f_{cor} denote the functionality responsible for generating this correlated randomness. We obtain the following result:

(Informal) Theorem 2. *Assuming the Learning with Errors (LWE) assumption or the Decisional Composite Residuosity (DCR) assumption, there exists a black-box weighted MPC protocol in the f_{cor}-hybrid model that achieves guaranteed output delivery against malicious adversaries. The computation and communication in our protocol scale only (in a fixed polynomial) with the number of parties and are independent of the total assigned weights.*

This result is presented in Section 5. We note that while this protocol can also be extended to general access structures, it is only expected to achieve guaranteed output delivery when the set of all honest parties itself forms an authorized set. If the honest parties do not constitute an authorized set, then guaranteed output delivery is impossible to achieve [HLP11].

Weighted Threshold Encryption. As a direct application of our weighted MPC protocol, we show how it can be combined with standard encryption schemes to obtain the first construction of a black-box, semi-honest secure weighted threshold encryption scheme with weight-independent complexity. Instantiating this idea with existing LWE-based encryption scheme [LPR10] yields the following:

(Informal) Theorem 3. *Assuming a trusted setup and the Learning With Errors (LWE) assumption, there exists a black-box weighted threshold encryption scheme with weight-independent complexity, that is secure against semi-honest adversaries.*

Unlike most existing threshold encryption schemes, our protocol does not support non-interactive decryption. In particular, our decryption protocol requires two rounds. We present this result in Section 6.

Weighted Threshold Schnorr Signatures. Finally, we show how our computational secret sharing framework can be applied to obtain weighted threshold signatures. While there has recently been a surge of works on black-box constructions of weighted threshold signatures [MRV+21, DCX+23, GJM+24, GGW24], these approaches are typically based on SNARKs and require modifications to the verification algorithm of the underlying signature scheme, which significantly limits their generality and applicability. In contrast, our approach demonstrates that our computational secret sharing can be used to transform any existing signature scheme into a weighted threshold signature scheme while preserving the original verification algorithm. As a concrete instantiation, we apply our construction to Schnorr signatures [Sch90], obtaining the following result:

[4]Weighted MPC achieving security with abort can be easily done with weight-independent complexity. This is because there exists at least one honest party, and one may simply run an all-but-one corrupt MPC protocol, for which we have several efficient protocols achieving security with abort [FL19, GSZ20, BBC+19, BGIN19, BGIN20, BGIN21, DEN24] .

(Informal) Theorem 4. *Assuming a trusted setup and the hardness of discrete log (DL) plus either learning with error (LWE) or Decisional Composite Residuosity (DCR) assumption, there exists a black-box three-round weighted threshold Schnorr signature scheme with weight-independent complexity secure against semi-honest adversaries in the random oracle model.*

This result is presented in Section 7.

2 Technical Overview

In this section, we present the main ideas underlying our results. The overarching goal of this work is to design multiparty *weighted threshold* cryptosystems.

2.1 Weighted Secret Sharing

Since secret sharing serves as the foundational building block for most such primitives, we begin by reviewing what is known about weighted secret sharing schemes.

Prior Works on Weighted Secret Sharing. Any threshold secret sharing scheme can be extended to the weighted threshold setting using the standard technique of *player virtualization*. However, this approach inevitably increases the share size in proportion to the total weight, which is highly undesirable. Prior works that improve upon this naïve approach can be broadly categorized into (i) information-theoretic schemes and (ii) computational schemes.

- In the *information-theoretic* setting, the only known *linear secret sharing scheme* for weighted threshold access structures beyond the naïve player virtualization is due to [BW06], who obtained a scheme with share size super-polynomial in n, namely $n^{\Theta(\log n)}$. Their construction proceeds by designing a monotone Boolean formula[5] of the same size and then applying known compilers [BL90].

  In the non-linear setting, [GJM+23] provided the only other exception that avoids player virtualization. Their scheme, while non-linear, still enjoys a certain homomorphism sufficient for MPC and threshold cryptosystem applications. However, their focus was mainly on concrete efficiency; in particular, the share size still grows linearly in the total assigned weight (which could still be as large as n^n [Mur71, Hås94]).

  These are the weighted secret sharing schemes that are *MPC-friendly*. There are other information-theoretic secret sharing schemes exist in the literature (e.g., [BHS23]), but they do not provide the necessary homomorphism to be useful in MPC or threshold cryptography.

- In the *computational* setting, Yao [Yao89] proposed a secret sharing scheme for general access structures[6] that can be represented efficiently as a monotone Boolean circuit. This scheme requires only black-box use of one-way functions, and in the special case of weighted threshold access structures, it achieves share sizes that are *independent* of the assigned weights [BW06]. Moreover, the dealer's computational cost is also independent of the total weight.

[5] A monotone Boolean formula is a monotone Boolean circuit (Definition 5) in which the fan-out of every gate is restricted to be 1.

[6] Since this manuscript was never published, we refer readers to [Bei25, Section 9.3] for a detailed presentation of the scheme.

Given Yao's scheme, one can already construct weighted cryptosystems, such as weighted threshold signatures, with weight-independent complexity by making *non-black-box* use of one-way functions. For instance, the dealer may sample and share a secret signing key sk (of some underlying digital signature scheme) using Yao's construction. Then, in the online phase, an authorized subset of users can employ a generic MPC protocol to compute the signing functionality using the reconstructed secret key from their shares. While this approach works, it necessarily requires non-black-box access to the underlying one-way functions.

The main obstacle is that, while Yao's scheme achieves the desired complexity, it does not support *linear homomorphism*, a property typically essential for employing secret sharing in the design of larger cryptographic systems. The goal of this paper is to obtain *black-box* constructions of multiparty weighted cryptosystems with weight-independent complexity.

To overcome this limitation, *our first key idea* is to extend Yao's secret sharing scheme [Yao89] to support linear homomorphism, while preserving its weight-independent complexity. This modified variant serves as the main building block for our constructions of other weighted multiparty cryptographic primitives.

Overview of Yao's [Yao89] Computational Secret Sharing Scheme. The high-level idea behind Yao's construction is inspired by the information-theoretic secret sharing scheme of Benaloh and Leichter [BL90] for general access structures: namely, to represent the access structure as a monotone Boolean circuit over the set of parties. To share a secret s via the BL scheme, the dealer embeds s into the circuit so that any authorized set of parties can evaluate the circuit on their shares to reconstruct s, while any unauthorized set cannot distinguish between shares of s and $s' \neq s$.

Concretely, the dealer begins at the output gate of the circuit (with the output wire holding the secret s) and recursively assigns values to the incoming wires of each gate, proceeding down the circuit. At the leaf level, each wire corresponds to a party, and the associated value becomes that party's share. Thus, reconstruction is achieved by authorized subsets of parties evaluating the circuit using their shares, whereas unauthorized subsets gain no information about s.

The total size of the shares in this scheme is the number of wires in the circuit. A challenge with this recursive approach is that nodes with fan-out greater than one receive multiple values from the recursion, each of which would need to be shared independently. This naïve method can cause the share size to grow exponentially in the depth of the circuit.[7]

Yao's key idea to prevent exponential blow-up was to introduce secret keys and computational assumptions.[8] The dealer samples a secret key for each gate in the circuit, maintaining the invariant that every gate is associated with both a key and a message. The key is shared as follows (see figure. 1):

- AND *Gate:* The dealer additively secret shares each AND gate's key among its incoming wires.
- OR *Gate:* The dealer assigns the same key (of this gate) as the share to each incoming wire.

The message is defined recursively starting from the output gate. At the output gate, the message is the secret s. For all other gates, the message is all the key shares assigned to its outgoing wires. Finally, the dealer encrypts every gate's message under its corresponding key and publishes all ciphertexts. This collection forms the *public part* of the secret sharing scheme. Each party's final share consists of the public

[7] This is precisely why [BW06] obtained share size $n^{\Theta(\log n)}$ as they constructed a monotone Boolean circuit of depth $\Theta(\log n)$.

[8] For readers familiar with Yao's garbled circuits, this is analogous to how computational garbling schemes avoid exponential blow-up in the *information-theoretic* garbling schemes.

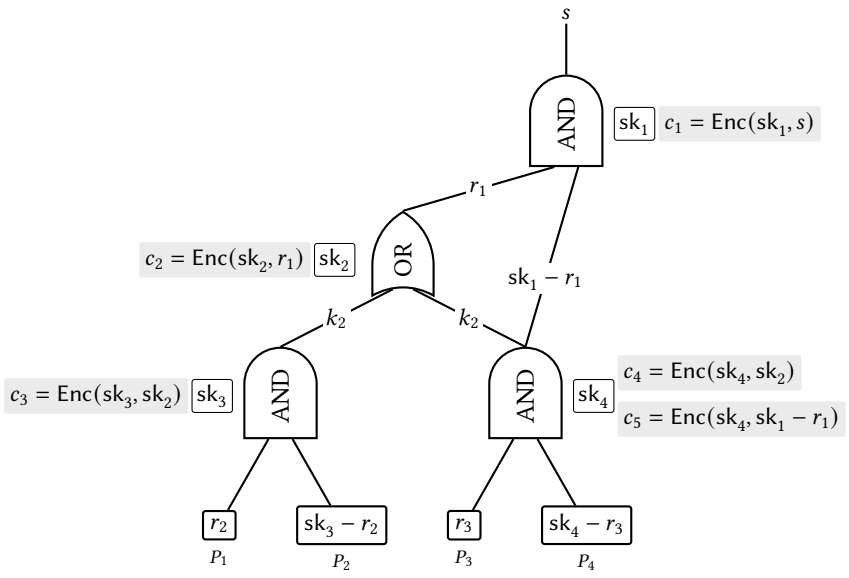

Figure 1: Yao's secret sharing scheme for a monotone Boolean circuit. Every gate carries a key, together with a message; the message of a gate is encrypted under that gate's own key, and the resulting ciphertexts, shaded grey, are published. The total share size grows with the number of gates.

part together with the messages assigned to the input wires corresponding to that party, which is the *secret part* of the secret sharing scheme.

The overall share size grows only with the number of wires in the circuit. In particular, for weighted threshold access structures, this implies that the share size remains independent of the total weights, since there exists a monotone Boolean circuit of fixed polynomial size that describes any weighted threshold access structure [BW06].

Adding Support for Linear Homomorphism in Yao's Scheme. It is easy to see that the construction described above does not naturally support linear homomorphism. Our idea for enabling linear homomorphism is to replace all encryptions in Yao's scheme with a linearly homomorphic public-key encryption scheme (LHE). By doing so, for linearly combining two (or more) secret sharings of secrets s and s', the parties can linearly aggregate both the ciphertexts (i.e., the public part) and the secret-key shares associated with the input wires in the monotone circuit (i.e., the secret part). If the underlying encryption scheme supports linear homomorphism over both the message space and the secret key space, i.e.,

$$\mathsf{Enc}(\mathsf{sk}, \mathsf{msg}) + \mathsf{Enc}(\mathsf{sk}', \mathsf{msg}') = \mathsf{Enc}(\mathsf{sk} + \mathsf{sk}', \mathsf{msg} + \mathsf{msg}'),$$

this idea appears to yield a new sharing of the linearly combined secret. In particular, to reconstruct the combined secret $s + s'$, authorized parties proceed exactly as before: they decrypt the ciphertexts gate by gate, eventually reaching the output gate, which reveals the combined secret $s+s'$. This is possible because the required homomorphism guarantees that decrypting combined ciphertexts using the combined secret keys yields the combined messages.

However, there are important subtleties concerning both security and correctness that must be addressed in order to make this high-level idea work.

- ***Security Concern:*** Suppose the parties hold shares of secrets s_1 and s_2, and apply the above linear homomorphism technique to derive new shares of $s_1 + s_2$. A subtle issue arises: using a generic LHE

scheme may inadvertently leak information about s_1 and s_2 during reconstruction of $s_1 + s_2$. Specifically, reconstruction requires decrypting a linear combination of ciphertexts originating from the sharings of s_1 and s_2. Since these (individual) ciphertexts are also available to the parties, the decryption process could reveal additional information about s_1 and s_2. This leakage stems from the fact that generic LHE schemes do not guarantee the privacy of the underlying ciphertexts when decrypting aggregated ciphertexts.

A familiar reader might realize that similar issues also arise in the context of *arithmetic garbling* [AIK11]. In arithmetic garbling over large fields, each wire may take exponentially many values, making it infeasible to assign a distinct label for each value (as in Boolean garbling). Instead, the label for each value x is defined by an affine function $ax + b$, and the garbled table consists of encryptions of $\mathsf{Enc}(\mathsf{sk}_1, a)$ and $\mathsf{Enc}(\mathsf{sk}_2, b)$. During circuit evaluation,[9] the ciphertexts are combined via a linear combination $(x, 1)$, namely

$$x \cdot \mathsf{Enc}(\mathsf{sk}_1, a) + 1 \cdot \mathsf{Enc}(\mathsf{sk}_2, b),$$

which is then decrypted using $x \cdot \mathsf{sk}_1 + 1 \cdot \mathsf{sk}_2$. The goal is that the evaluator only learns the correct label $ax + b$ and nothing more. This precisely matches both the type of homomorphism and the associated privacy concern that we encounter here.

Recent works on arithmetic garbling [AIK11, BLLL23] develop techniques to address this issue. In particular, they show that the problem can be fixed by appropriately "smudging" the ciphertexts using standard noise smudging techniques. We provide an informal overview of the smudging technique in Section 3. Notably, privacy of reconstruction for a linearly combined ciphertext can be guaranteed *only if the linear combination includes at least one smudged ciphertext whose coefficient needs to be 1.*

Given this, a natural idea is to adapt the existing constructions of LHE from arithmetic garbling into our scheme. Concretely, our secret sharing scheme will support two modes of sharing: a *normal mode*, where all ciphertexts are generated in the standard way, and a *smudged mode*, where all ciphertexts are augmented with smudging. Privacy is then preserved when reconstructing the linear combination of a normal sharing with a smudged sharing (with linear coefficient 1). When using this secret sharing scheme as a building block for other primitives, it is crucial to ensure that every linear combination of secret sharing that we will reconstruct must include at least one smudged sharing which cannot be reused.

In Section 3, we formally define the special-purpose LHE required for our scheme. A notable difference between what we need here and the form of LHE used in arithmetic garbling is that arithmetic garbling only requires *limited homomorphism* (i.e., combining only two ciphertexts), and each ciphertext is combined and decrypted *at most once*. In contrast, our setting requires support for linear combination of *multiple* ciphertexts/secret sharing instances, and a secret sharing instance (for example, a sharing of a secret signing key) may be linearly combined and reconstructed *multiple times*.

Nevertheless, we show that the techniques introduced in [BLLL23] already suffice to obtain the stronger form of linear homomorphism that we require. In particular, we present concrete instantiations of our construction based on the Learning with Errors (LWE) and Decisional Composite Residuosity (DCR) assumptions.

Returning to secret sharing, even after constructing this form of LHE and addressing the aforementioned security concern, there remain additional technical subtleties, which we discuss next.

[9] This is imprecise and leaves out many technical details. We refer the readers to [BLLL23] for more details, but carry on with this informal description.

- ***Correctness Concern:*** Recall that in this garbling-based secret sharing scheme, the secret keys corresponding to gates at one level of the circuit serve as the messages for the gates at the next level. Therefore, the LHE used to instantiate this idea must ensure that both messages and secret keys lie in the same domain. This is how it is done in the arithmetic garbling literature; in particular, the corresponding constructions of LHE that simultaneously satisfy this requirement and address the security concern discussed above only achieve it when the secret key space consists of *bounded integers* This presents a problem: the garbling-based secret sharing scheme relies on *additive secret sharing* for every AND gate, which is not well-defined over bounded integer domains.

  However, we note that in both the LWE- and DCR-based instantiations, the upper bound on the key size (even after linear homomorphic evaluation) is fixed; in particular, it is independent of the message space. Hence, we can always embed the key space into an integer ring $\mathbb{Z}_p$ for a sufficiently large p such that arithmetic over $\mathbb{Z}$ and over $\mathbb{Z}_p$ coincide. Once the keys are lifted as elements of $\mathbb{Z}_p$, additive secret sharing is now well-defined. This resolves the misalignment in domain requirements between Yao's secret sharing construction and the LHE construction.

2.2 Weighted Cryptosystems

We now discuss how our computational secret sharing with linear homomorphism can be used to design other weighted multiparty cryptosystems with weight-independent complexity. We note that, as with our secret sharing scheme, all of these constructions can be extended to general access structures.

Weighted Secure Multiparty Computation. Starting from the seminal works [GMW87, BGW88, CCD88], secret sharing schemes have been indispensable building blocks in the design of secure multiparty computation (MPC) protocols. A common template for constructing such MPC protocols is to begin with secret shares of the parties' inputs and then evaluate the circuit representing the target functionality in a gate-by-gate manner over secret-shared values. When the underlying secret sharing scheme supports linear homomorphism, addition gates can be computed locally: each party simply aggregates its shares of the input wires to obtain a share of the output wire.

Multiplication gates are more challenging. The standard technique for handling them is to use Beaver triples [Bea91]. Specifically, given shares of a random triple (a, b, c) with $c = a \cdot b$, Beaver showed how the parties can compute shares of the output of a multiplication gate using only linear operations on the triples together with the input wire shares.

Since our weighted secret sharing scheme supports linear homomorphism, we can leverage this same template: given weighted shares of Beaver triples, we obtain a weighted MPC protocol with *weight-independent* complexity. To ensure privacy, these Beaver triples must be shared in *smudged mode*.

While this suffices for semi-honest security, recall from Section 1.1 that our goal is to achieve guaranteed output delivery against malicious adversaries. In Section 4, we show that our secret sharing scheme is also *self-authenticating*: given all ciphertexts, one can efficiently verify whether the private components of the revealed shares are consistent. In other words, the public components are binding, allowing the protocol to detect malformed shares during reconstruction and exclude corrupt parties, thereby enabling guaranteed output delivery.

We remark that we can also achieve guaranteed output delivery for general access structures using this framework, but only when the set of all honest parties alone forms an authorized set.

We can also employ a simple semi-honest secure MPC protocol for generating weighted shares of Beaver triples with weight-independent complexity. The parties first use existing techniques [DPSZ12, BCGI18, BCG+19, BCG+20] to obtain additive shares (over $\mathbb{Z}_p$) of random Beaver triples. Each party then

applies our weighted secret sharing scheme to its own additive share of the triple. Finally, leveraging the linearly homomorphic property of our scheme, the parties can locally combine these shares to obtain weighted shares of the Beaver triple. We defer further details to Section 5.

Weighted Threshold Encryption. The next application of our weighted secret sharing scheme is a weighted threshold encryption scheme. As discussed in Section 1.1, no black-box constructions of such schemes with weight-independent complexity were previously known. At a high level, our approach follows the standard template used in threshold encryption. The idea is to begin with a public-key encryption scheme, generate a public/secret key pair, and secret share the secret key among the parties. Given a ciphertext, each party can then compute a partial decryption using its share of the secret key. These partial decryptions must not leak the parties' secret-key shares.[10] Finally, the partial decryptions from an authorized set of parties can be combined to recover the plaintext.

If the underlying secret sharing scheme supports linear homomorphism, and if partial decryptions can be expressed using only linear operations over the ciphertext and the secret-key shares, then each party can compute its partial decryption locally. Importantly, this can be done without interaction and without knowing in advance which set of parties' partial decryptions will ultimately be used for decryption.

We aim to follow this paradigm. However, there is a technical barrier that we must first address. We begin with a standard Regev-style LWE-based encryption scheme [Reg05]. Given a ciphertext of the form $(\overrightarrow{u}, v)$, the partial decryption in corresponding threshold encryption schemes typically involves computing the inner product $\langle \overrightarrow{u}, \mathsf{sk}_i \rangle$, where sk_i is the secret-key share held by party i. However, revealing this inner product directly would leak information about the secret-key share. To prevent such leakage, a standard technique employed in prior works [BGG+18] is to require each party to add a small random noise to its partial decryption. The noise distribution is carefully chosen so that, when the partial decryptions of an authorized set are combined, the total accumulated error remains bounded and decryption correctness is preserved.

For weighted threshold or more general access structures, it is unclear how to select an appropriate noise distribution that guarantees the total accumulated error remains bounded for *every* authorized set. Therefore, we relax the requirement and instead present a scheme with an *interactive* partial decryption procedure, specifically a two-round protocol. In the first round, each party samples a small noise term and secret-shares it among all parties using our weighted secret sharing scheme. In the second round, after receiving these shares, the parties leverage the linear homomorphism of our weighted secret sharing to aggregate the error shares. This aggregated error is then used to mask the inner product between $\overrightarrow{u}$ and their respective secret-key shares.

To the best of our knowledge, even with this interactive partial decryption procedure, no prior work has achieved a *black-box* construction of weighted threshold encryption with weight-independent complexity. We further remark that while we describe the above idea in the context of LWE-based encryption, the same approach can be extended to other public-key encryption schemes, such as those based on the Learning Parity with Noise (LPN) assumption, where partial decryption can also be expressed using only linear operations on the ciphertext and secret-key shares. We refer the reader to Section 6 for details.

Weighted Threshold Signatures. Finally, we construct a weighted threshold Schnorr signature scheme based on our secret sharing scheme. Here, we require that the final output is exactly a standard Schnorr signature. Recall that in the Schnorr scheme, the public key is $\mathsf{pk} = g^{\mathsf{sk}}$, where g is the generator of some cryptographic group $\mathbb{G}$. A Schnorr signature on message msg takes the form (R, σ), and verification checks that $\mathsf{pk}^c \cdot R = g^\sigma$, where $c = H(\mathsf{msg}, \mathsf{pk}, R)$ is the Fiat-Shamir challenge derived via the random oracle.

[10] Observe that this privacy requirement is necessary to rule out trivial solutions in which each party simply outputs its private key share as part of the partial decryption. Refer to Remark 2.

We follow the standard three-round threshold Schnorr signing protocol [CKM23]. In the first round, each participating party commits to its freshly sampled nonce g^{r_i}. In the second round, each party decommits g^{r_i}, and the combined nonce is computed as $R = \prod_i g^{r_i}$. Consequently, the Fiat-Shamir challenge $c = H(\mathsf{msg}, \mathsf{pk}, R)$ becomes publicly known. In the third round, each party uses its secret shares of sk to help reconstruct $c \cdot \mathsf{sk} + \sum_i r_i$.

The key difference in our setting lies in the computation of the third-round messages. Our secret sharing does not allow an authorized set of parties to directly reconstruct $c \cdot \mathsf{sk} + r$ even if each party knows an additive share of r and a weighted threshold share of sk (via our scheme). We circumvent this as follows: in the second round, each party additionally secret shares its nonce r_i using our scheme. Then, by the final round, every party holds secret shares of both sk and all r_i's. At this point, the linear homomorphism property of our scheme allows the parties to reconstruct $c \cdot \mathsf{sk} + \sum_i r_i$. For privacy, we require that every r_i is shared in smudged mode, while the original secret key sk is shared in normal mode. Details are presented in Section 7.

Notation. Throughout the paper, we use λ for the security parameter. For two distributions, we write $A \approx B$ to denote that A and B are computationally indistinguishable. For any positive integer n, $[n]$ denotes the set $\{1, 2, \ldots, n\}$.

2.3 Concurrent Work and Further Improvement

A concurrent and independent work by Farràs and Guiot [FG26] considers the problem of optimizing Yao's computational secret-sharing scheme for weighted access structures. In particular, they observe that Yao's computational secret-sharing scheme, as well as the information-theoretic variant of [BL90], can be extended to Boolean circuits augmented with threshold gates.[11]

Leveraging this observation, they improve the Beimel–Weinreb construction [BW06] for Boolean circuits computing weighted access structures. Specifically, they construct a Boolean circuit augmented with threshold gates of size $O(n^3 \log n)$. Moreover, unlike the Beimel–Weinreb construction, their construction does not hide large constants in the big-O notation, making it more likely to be practically efficient.

We remark that their improvement to the Beimel–Weinreb construction can be readily applied to our scheme, yielding a more efficient computational secret-sharing scheme with linear homomorphism for weighted access structures. As discussed above, the resulting construction avoids large hidden constants and may be practically efficient. We leave this as an exciting direction to explore for future work.

3 Special-Purpose Linearly Homomorphic Encryption

In this section, we present a special-purpose linearly homomorphic encryption scheme that is tailored for our linearly homomorphic computational secret sharing scheme.

3.1 Overview

Both our definition and constructions are inspired by those considered in recent works on arithmetic garbling [AIK11, RS21, BLLL23, MORS24, MORS25]. However, as discussed in Section 2, we require some modifications. For ease of understanding, we first present an overview of our construction before proceeding to formalize the notion of a special-purpose linearly homomorphic encryption.

[11] That is, a gate that outputs 1 if and only if the number of input wires carrying value 1 exceeds a (hardwired) threshold. In particular, AND and OR gates are special cases of threshold gates.

Required Properties for Special-Purpose LHE. As discussed in Section 2, our goal is to augment Yao's computational secret sharing scheme with linear homomorphism by leveraging a linear homomorphic encryption (LHE) scheme. We now spell out the properties required of such an encryption scheme, which in turn motivate our Definition 1 in Section 3.2.

- ***Linear Homomorphism:*** We require the encryption scheme to be homomorphic with respect to both the secret key and the message. Moreover, observe that in the Boolean circuit representation of certain monotone access structures, some gates may have fan-out greater than one. In such cases, following Yao's construction, the secret key associated with a gate must be used to encrypt multiple messages. Consequently, the encryption scheme must support batch encryption, and its homomorphic properties must extend naturally to batched ciphertexts. These requirements are captured in the *linear homomorphism property* in Definition 1.

- ***Decryption-Friendliness:*** A crucial privacy guarantee we require is that the ciphertexts $\mathsf{ct}_1 = \mathsf{Enc}(\mathsf{sk}_1, \mathsf{msg}_1)$ and $\mathsf{ct}_2 = \mathsf{Enc}(\mathsf{sk}_2, \mathsf{msg}_2)$, together with the aggregated secret key $\mathsf{sk}^* = a_1 \cdot \mathsf{sk}_1 + a_2 \cdot \mathsf{sk}_2$, for publicly known coefficients a_1, a_2, reveal only the combined message $\mathsf{msg}^* = a_1 \cdot \mathsf{msg}_1 + a_2 \cdot \mathsf{msg}_2$, and nothing else about the individual messages msg_1 and msg_2. While generic linearly homomorphic encryption schemes do not satisfy this property, certain schemes used in applications such as arithmetic garbling do. This requirement is typically formalized by demanding the existence of a simulator that, given only the combined message msg^*, can efficiently simulate the entire view $(\mathsf{sk}^*, \mathsf{ct}_1, \mathsf{ct}_2)$.

  While this guarantee suffices for applications like arithmetic garbling, it is not strong enough for our purposes. As a motivating example, consider using our secret sharing scheme to share a secret signing key x for a Schnorr signature scheme. During the sharing phase, the signing key x is secret shared, meaning that the ciphertexts corresponding to Yao's sharing of x become public. Later, when a signing session is initiated, the parties jointly generate a random mask r and reconstruct $a \cdot x + r$ (as in a standard Schnorr protocol). In this setting, we must argue that the parties learn nothing beyond the value $a \cdot x + r$.

  Crucially, the simulator must be able to generate this view even when the ciphertexts encrypting x are fixed *a priori*. Moreover, since there may be multiple signing sessions, the simulator must support fully adaptive simulation, producing a consistent view for each session corresponding to a newly revealed value $a' \cdot x + r'$. These requirements are captured by the *decryption-friendliness property* formalized in Definition 1.

- ***Computational Binding:*** Finally, for our MPC application, we aim to achieve the strongest notion of *guaranteed output delivery*. To this end, the underlying secret-sharing scheme must be robust. Our approach to achieving robustness is to make public commitments to the (private) secret shares held by each party. Moreover, these commitments must themselves support homomorphic operations.

  For this reason, we require the LHE setup to additionally specify a public key that serves as a commitment to the secret key, even though the encryption scheme is otherwise symmetric-key. This requirement is captured by the *computational-binding property* in Definition 1.

Two-Mode Encryption. Ideally, we would like to construct an LHE scheme satisfying all of the properties listed above, where both the secret key space and the message space are $\mathbb{Z}_p$. This requirement is important for several reasons.

1. In Yao's construction, the secret key corresponding to an AND gate is additively shared, and these additive shares become (part of) the messages associated with its child gates. To enable additive sharing, the secret key must therefore be treated as a ring element over some $\mathbb{Z}_p$.

2. If the secret key space and the message space do not coincide — specifically, if the key space is defined over some modulus p' that is much larger than p — this may lead to a recursive growth of the key and message spaces as the circuit depth increases. This is because the key space of a gate becomes the message space of its predecessor gates in Yao's construction.

3. Since we linearly combine both keys and messages using the same coefficients, a mismatch between the key space and the message space means that we cannot treat these coefficients as ring elements. Instead, we would have to interpret them as integers in $\mathbb{Z}$ – which as we discuss next, is indeed what we do.

Unfortunately, we currently do not know how to construct LHE schemes that simultaneously satisfy all of the above properties. Instead, existing constructions [BLLL23] treat secret keys as integers drawn from a bounded range, say $\{0, \ldots, \mathsf{Max}\}$. Furthermore, we cannot hope for *decryption-friendliness* to hold for arbitrary pairs of ciphertexts.

To address these limitations, we introduce two operational modes for both key generation and encryption: (i) a ***normal*** mode and (ii) a ***smudging*** mode. In the smudging mode, both the secret key and the ciphertext are masked by adding a small amount of random noise. By the Smudging Lemma (Lemma 1), such noisy keys and ciphertexts remain statistically indistinguishable from their normal counterparts. Consequently, it suffices to guarantee *decryption-friendliness* only when combining a normal ciphertext with a smudged ciphertext. Furthermore, we require that the linear coefficient applied to the smudged ciphertext be exactly one.[12]

Lemma 1 (Smudging Lemma [AJL+12])**.** *Let M be a positive integer. Let $\mathcal{D}$ denote a distribution over $\mathbb{Z}$, which is uniform over $\{a, a+\delta, a+2\cdot\delta, \ldots, a+M\cdot\delta\}$ for some $a, \delta \in \mathbb{Z}$. For any $k \in \mathbb{Z}$, it holds that*

$$\mathsf{SD}\left(\mathcal{D}, \mathcal{D} + k\cdot\delta\right) \leqslant \frac{|k|}{M},$$

where $|k|$ denotes the absolute value of k. For our purposes it suffices to consider $\delta = 1$.

We now further explain why the above restrictions suffice for our purposes:

- Although secret keys are not *a priori* elements of a ring, we can embed them into $\mathbb{Z}_p$ for a sufficiently large modulus p, thereby enabling additive secret sharing over $\mathbb{Z}_p$.
- As discussed above, it is therefore natural to treat the linear coefficients as elements of $\mathbb{Z}$. We additionally impose a bound B on these coefficients to ensure that linear combinations of secret keys computed over $\mathbb{Z}$ agree with the corresponding arithmetic in $\mathbb{Z}_p$. In particular, it suffices to choose $p \geq B \cdot \mathsf{Max}$.
- Finally, to prevent recursive growth of the key and message spaces, we require that the bound Max on the secret keys be independent of the message space.

These properties are reflected throughout Definition 1.

Overview of Construction. We now give a brief overview of the construction of our special-purpose LHE scheme. This construction is inspired by and is closely related to the one in [BLLL23]. Let us start

[12] In the context of arithmetic garbling, the different modes of encryption are implicitly built into the construction, since the garbler always prepares the labels in such a way that the evaluator only ever combines normal ciphertexts with smudging ciphertexts (with linear coefficient equal to one).

by recalling the standard LWE-based private-key encryption scheme: the secret key $\mathsf{sk} \in \mathbb{Z}_q^n$, and the ciphertext is of the form

$$(a, \langle a, \mathsf{sk}\rangle + e + \mathsf{msg} \cdot \Delta) \mod q,$$

where the message $\mathsf{msg} \in \mathbb{Z}_p$, e is a small noise, and $\Delta = \lfloor q/p \rceil$. a is typically sampled during Setup and published to all parties. This allows each party to choose its own secret key sk to encrypt its message msg with respect to the common a. Observe that adding two ciphertexts computed using the same a, i.e., $\mathsf{ct} = (a, \langle a, \mathsf{sk}\rangle + e + \mathsf{msg} \cdot \Delta)$ and $\mathsf{ct}' = (a, \langle a, \mathsf{sk}'\rangle + e' + \mathsf{msg}' \cdot \Delta)$, yields

$$(a, \langle a, \mathsf{sk} + \mathsf{sk}'\rangle + (e + e') + (\mathsf{msg} + \mathsf{msg}') \cdot \Delta) \mod q,$$

which is approximately

$$(a, \langle a, \mathsf{sk} + \mathsf{sk}'\rangle + (e + e') + ((\mathsf{msg} + \mathsf{msg}') \bmod p) \cdot \Delta) \mod q.$$

In other words, this scheme achieves linear homomorphism over both the key space and the message space. The only parameter that requires care is e, which must be chosen as a function of the upper bound B on the linear coefficients; this can be handled in a standard manner.

The main challenge for us lies in ensuring *decryption-friendliness*. When decrypting

$$(a, \langle a, \mathsf{sk} + \mathsf{sk}'\rangle + (e + e') + (\mathsf{msg} + \mathsf{msg}') \cdot \Delta) \mod q$$

using the combined key $\mathsf{sk}+\mathsf{sk}'$, the resulting error term is $e+e'$. It is not immediately clear that the underlying ciphertext $\mathsf{ct} = (a, \langle a, \mathsf{sk}\rangle + e + \mathsf{msg} \cdot \Delta)$ remains pseudorandom given $e + e'$ as auxiliary information. As discussed above, we address this issue in the same spirit as [BLLL23] by employing smudging/noise flooding. Concretely, for the other underlying ciphertext ct', which will be computed in the ***smudging*** mode, we will sample e' from a sufficiently wide distribution so that $e + e' \approx e'$, ensuring that no information about e is revealed. We can now rely on the LWE assumption (see Definition 2) to ensure that ct remains pseudorandom.

Proving Decryption-Friendliness. For arguing decryption-friendliness of the above scheme, our simulator will operate as follows:

- Sample ct as a fresh encryption of 0 (which is possible since no information about e is leaked).
- Sample the combined secret key $\mathsf{sk} + \mathsf{sk}'$.
- Sample a combined ciphertext ct'' by encrypting the plaintext sum (which the simulator receives as input) under $\mathsf{sk} + \mathsf{sk}'$.
- Output $\mathsf{ct}' = \mathsf{ct}'' - \mathsf{ct}$.

An additional complication arises from the approximation error introduced by $(\mathsf{msg} + \mathsf{msg}') \cdot \Delta \approx ((\mathsf{msg}+\mathsf{msg}') \bmod p) \cdot \Delta$. To mask this error, e' must again be sampled from a sufficiently large distribution, which can be done appropriately.

However, this is still insufficient. In particular, the secret keys are defined over $\mathbb{Z}_q$ for $q \gg p$, which conflicts with our requirement that the maximum key size remain independent of the message-space size p, so as to avoid recursive growth of the key length. To address this issue, we rely on the *small-secret variant of LWE*, where secret keys are sampled as small integers. While this will resolve the key-growth issue, it introduces a new challenge — revealing $\mathsf{sk} + \mathsf{sk}'$ would leak information about the individual secret keys.

To fix this, one has to sample sk' from a much larger range so that $\mathsf{sk} + \mathsf{sk}'$ does not leaks any information about sk, i.e., we also need to smudge the secret keys. With some care, this can be done and we refer the readers to Section 3.3 for details. We also present a construction (using similar ideas) based on the DCR assumption in Section 3.4.

3.2 Definition

We now formalize the notion of special-purpose linearly homomorphic encryption.

Definition 1 (Special-Purpose Linearly Homomorphic Encryption Scheme)**.** *A special-purpose linearly homomorphic encryption scheme consists of a tuple of algorithms* (Setup, KeyGen, Enc, Dec, SKEval, PKEval, CTEval, Verify) *as follows.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p)$*: on input the security parameter* λ*, the batch size* m*, an upper bound* B*, and a message space* $\mathbb{Z}_p$*, it outputs a public parameter* pp *that contains the descriptions of the secret-key space* $\mathcal{SK}$*, public-key space* $\mathcal{PK}$ *and ciphertext space* C*. All the following algorithms take* pp *as input implicitly.*

- $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b)$*: on input the security parameter* λ *and a bit* b*, it samples a secret/public key pair* $(\mathsf{sk}, \mathsf{pk})$*. When* $b = 0$*, we call the resulting secret key a* normal secret key*. When* $b = 1$*, we call the resulting secret key a* smudging secret key*. Outputs* $(\mathsf{sk}, \mathsf{pk})$*.*[13]

- $(\mathsf{ct}_1, \ldots, \mathsf{ct}_m) \leftarrow \mathsf{Enc}(\mathsf{sk}, \mathsf{msg}_1, \ldots, \mathsf{msg}_m)$*: on input the secret key*[14] $\mathsf{sk} \in \mathcal{SK}$ *and* m *messages from* $\mathbb{Z}_p$*, it computes* m *ciphertexts. Just like secret keys, there will be two different types of ciphertexts: normal and smudging. When the secret key is smudging (resp., normal), the resulting ciphertext is smudging (resp., normal).*

- $(\mathsf{msg}_1, \ldots, \mathsf{msg}_m) = \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m)$*: on input a secret key* $\mathsf{sk} \in \mathcal{SK}$ *and* m *ciphertexts from* C*, it outputs* m *messages in* $\mathbb{Z}_p$*.*

- $\mathsf{sk}' = \mathsf{SKEval}((a_1, \ldots, a_\ell), (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell))$ *: on input a linear function defined by* ℓ *coefficients over* $\mathbb{Z}$[15] *and* ℓ *secret keys from* $\mathcal{SK}$*, it outputs another secret key* $\mathsf{sk}' \in \mathcal{SK}$*.*

- $\mathsf{pk}' = \mathsf{PKEval}((a_1, \ldots, a_\ell), (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell))$ *: on input a linear function defined by* ℓ *coefficients over* $\mathbb{Z}$ *and* ℓ *public keys from* $\mathcal{PK}$*, it outputs another public key* $\mathsf{pk}' \in \mathcal{PK}$*.*

- $\mathsf{ct}' = \mathsf{CTEval}((a_1, \ldots, a_\ell), (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell))$ *: on input a linear function defined by* ℓ *coefficients over* $\mathbb{Z}$ *and* ℓ *ciphertexts, it outputs another ciphertexts* $\mathsf{ct}' \in C$*.*

- $b \leftarrow \mathsf{Verify}(\mathsf{sk}, \mathsf{pk})$ *: on input a secret/public key pair, it outputs a bit* b *indicating if* sk *is the correct secret key for the corresponding public key* pk*. For a valid* $(\mathsf{sk}, \mathsf{pk})$ *pair,* $b = 1$*.*

The following properties should hold; for brevity, we denote a batch of m *messages by* $\overrightarrow{\mathsf{msg}}$*.*

[13] Without loss of generality, we assume that sk contains the bit b in its description, so we always know if a key is smudging or normal by checking it.

[14] We only need a symmetric encryption scheme, which is why the encryption takes the secret key as input. The public key only serves as a commitment to the secret key.

[15] These coefficients are used to combine keys, messages and ciphertexts. Hence, they are chosen from $\mathbb{Z}$ and the combination is interpreted accordingly in the key, message or ciphertext space.

- ***Bounded Secret Key.*** *The secret keys are bounded integers such that the bound is independent of the message space.*

- ***Linear Homomorphism.*** *For all parameters λ, m, B, p, $\ell \in \mathbb{N}$, public parameters* pp*, messages $\overrightarrow{\mathsf{msg}}_1, \ldots, \overrightarrow{\mathsf{msg}}_\ell$, bits $b_1, \ldots, b_\ell \in \{0,1\}$, and linear coefficients $a_1, \ldots, a_\ell \in \mathbb{Z}$ satisfying that $\sum_i |a_i| \leqslant B$, it holds that*

$$\Pr\left[\mathsf{Dec}(\mathsf{sk}', \mathsf{ct}') = \sum_i a_i \cdot \overrightarrow{\mathsf{msg}}_i \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p) \\ \forall i \in [\ell],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i) \\ \forall i \in [\ell],\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \overrightarrow{\mathsf{msg}}_i) \\ \mathsf{sk}' = \mathsf{SKEval}((a_1, \ldots, a_\ell), (\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell)) \\ \mathsf{ct}' = \mathsf{CTEval}((a_1, \ldots, a_\ell), (\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell)) \end{array}\right] = 1.$$

  Here, $\sum_i a_i \cdot \overrightarrow{\mathsf{msg}}_i$ is over the message space $\mathbb{Z}_p$.

  Note that this implies the standard correctness.

- ***Computationally-binding.*** *For all parameters λ, m, B, p, bits $b_1, \ldots, b_\ell$, linear coefficients $(a_1, \ldots, a_\ell)$ such that $\sum_i |a_i| \leqslant B$ and any PPT adversary $\mathcal{A}$, it holds that*

$$\Pr\left[\begin{array}{c} \mathsf{Verify}(\mathsf{sk}', \mathsf{pk}') = 1 \\ \wedge\ \mathsf{Verify}(\mathsf{sk}'', \mathsf{pk}') = 1 \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p) \\ \forall i \in [\ell],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i) \\ \mathsf{pk}' = \mathsf{PKEval}((a_1, \ldots, a_\ell), (\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell)) \\ (\mathsf{sk}', \mathsf{sk}'') \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{pk}') \end{array}\right] = \mathsf{negl}(\lambda).$$

- ***Decryption-friendliness.*** *For all parameters λ, m, B, p, messages $\overrightarrow{\mathsf{msg}}, \overrightarrow{\mathsf{msg}}_1, \ldots, \overrightarrow{\mathsf{msg}}_\ell$, and linear coefficients $a_1, \ldots, a_\ell \in \mathbb{Z}$ satisfying that $|a_i| \leqslant B$ for all $i \in [\ell]$, there exists stateful PPT simulators $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that the following two distributions are computationally close.*

$$\left\{\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p) \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0) \\ \forall i \in [\ell],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}}) \\ \forall i \in [\ell],\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \overrightarrow{\mathsf{msg}}_i) \\ \forall i \in [\ell],\ \mathsf{sk}'_i = \mathsf{SKEval}((a_i, 1), (\mathsf{sk}, \mathsf{sk}_i)) \\ \textit{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \textit{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array}\right\} \approx \left\{\begin{array}{c} \{\mathsf{pp}, \mathsf{ct}\} \leftarrow \mathsf{Sim}_1\left(1^\lambda, 1^m, B, p\right) \\ (\{\mathsf{ct}_i\}_{i\in[\ell]}, \{\mathsf{sk}'_i\}_{i\in[\ell]}) \leftarrow \\ \mathsf{Sim}_2\left(\mathsf{pp}, \mathsf{ct}, \left\{a_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i\right\}_{i\in[\ell]}\right) \\ \textit{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \textit{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array}\right\}.$$

  Again, we interpret $a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i$ as operations over the message space $\mathbb{Z}_p$. Note that this property is only required to hold when combining ciphertexts encrypted under normal secret keys with ciphertexts encrypted under smudging secret keys, and in this case the coefficient on the smudging secret keys must be 1*.*

Remark. *Note that the decryption friendliness implies the standard notion of IND-CPA security for one-time symmetric key*[16] *encryption. This is because decryption friendliness guarantees the existence of a simulator* Sim_1 *which can output a ciphertext* ct *that is computationally indistinguishable from* $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}})$ *for any message* $\overrightarrow{\mathsf{msg}} \in \mathbb{Z}_p$*. So for a secret key* sk *and any two messages* $\overrightarrow{\mathsf{msg}}_0$ *and* $\overrightarrow{\mathsf{msg}}_1$, $\mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}}_0) \approx \mathsf{Sim}_1 \approx \mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}}_1)$.

3.3 Instantiation Based on LWE

We now present our instantiation of special-purpose linearly homomorphic encryption (see Definition 1) based on LWE.

Definition 2 (Learning With Errors Assumption)**.** *The LWE assumption, parametrized by* $n \in \mathbb{N}$*, modulus* q*, and error distribution* χ *over* $\mathbb{Z}_q$*, states that, for all polynomial* m*, we have*

$$(A, A \cdot s + e) \approx (A, u),$$

where $A \leftarrow \mathbb{Z}_q^{m \times n}$, $s \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi^m$*, and* $u \leftarrow \mathbb{Z}_q^m$.

In fact, we will rely on the following variant where the secret key are small; in particular, binary. This is known to be implied by the standard LWE assumption [BLP+13, Mic18].

Definition 3 (LWE with Small Secrets)**.** *The LWE with small secret assumption, parametrized by* $n \in \mathbb{N}$*, modulus* q*, and error distribution* χ *over* $\mathbb{Z}_q$*, states that, for all polynomial* m*, we have*

$$(A, A \cdot s + e) \approx (A, u),$$

where $A \leftarrow \mathbb{Z}_q^{m \times n}$, $s \leftarrow \mathcal{D}^n$, $e \leftarrow \chi^m$*, and* $u \leftarrow \mathbb{Z}_q^m$*. In particular, we are interested in* $\mathcal{D}$ *that are uniformly sampled from* $\{0, 1, \ldots, L\}$ *with* $L = 1$ *as a special case.*

Construction. The construction proceeds as follows.

- $\mathsf{Setup}(1^\lambda, 1^m, B, p) \to \mathsf{pp}$: Pick an error distribution χ (assume the error distribution χ is bounded between $[-B_{\mathsf{error}}, B_{\mathsf{error}}]$). Set $B_{\mathsf{smudge}} = B \cdot \max(p, B_{\mathsf{error}}) \cdot \lambda^{\omega(1)}$. Set $B_{\mathsf{sk}} = \lambda^{\omega(1)}$. Sample $A \leftarrow \mathbb{Z}_q^{m \times n}$, $A_{\mathsf{pk}} \leftarrow \mathbb{Z}_m^{m \times n}$. Set the modulus q large enough such that $\Delta = \lfloor q/p \rceil$ satisfies

  $$\Delta = B \cdot (B_{\mathsf{error}} + B_{\mathsf{smudge}} + p) \cdot \lambda^{\omega(1)}.$$

  Finally, set $\mathsf{pp} = (n, m, \chi, B_{\mathsf{smudge}}, B_{\mathsf{sk}}, A, A_{\mathsf{pk}}, \Delta)$.

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: for normal secret key ($b = 0$), set $\mathsf{sk} \leftarrow \{0,1\}^n$; for smudging secret key ($b = 1$), set $\mathsf{sk} \leftarrow [B_{\mathsf{sk}}]^n$. Sample $e_{\mathsf{pk}} \leftarrow \chi^m$ and set $\mathsf{pk} = A_{\mathsf{pk}} \cdot \mathsf{sk} + e_{\mathsf{pk}}$.

- $\mathsf{Enc}(\mathsf{sk}, \mathsf{msg}_1, \ldots, \mathsf{msg}_m) \to (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$: Now, if the secret key is a normal secret key, sample $e \leftarrow \chi^m$ and output

  $$(\mathsf{ct}_1, \ldots, \mathsf{ct}_m) = A \cdot \mathsf{sk} + e + \Delta \cdot \overrightarrow{\mathsf{msg}}.$$

  Else, sample $e \leftarrow \chi^m$ and $e_{\mathsf{smudge}} \leftarrow [-B_{\mathsf{smudge}}, B_{\mathsf{smudge}}]$; output

  $$(\mathsf{ct}_1, \ldots, \mathsf{ct}_m) = A \cdot \mathsf{sk} + e + e_{\mathsf{smudge}} + \Delta \cdot \overrightarrow{\mathsf{msg}}.$$

[16] i.e., the secret-key can only be used once.

- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to (\mathsf{msg}_1, \ldots, \mathsf{msg}_m)$: Output $\left\lfloor \frac{(\mathsf{ct}_1, \ldots, \mathsf{ct}_m) - A \cdot \mathsf{sk}}{\Delta} \right\rceil$.
- SKEval, PKEval, CTEval: For SKEval, output $\sum_i a_i \cdot \mathsf{sk}_i$ over $\mathbb{Z}$. For PKEval (resp., CTEval), output $\sum_i a_i \cdot \mathsf{pk}_i$ (resp., $\sum_i a_i \cdot \mathsf{ct}_i$) over $\mathbb{Z}_q$.
- Verify: If $\|\mathsf{pk} - A_{\mathsf{pk}} \cdot \mathsf{sk}\|_\infty \leqslant B_{\mathsf{error}}$, output 1; otherwise, output 0.

Theorem 1. *Assuming the LWE assumption with appropriate parameters (Definition 2) holds, the construction above is a special-purpose linear homomorphic encryption scheme according to Definition 1.*

Proof. We sketch a proof showing that the construction above satisfies all the properties of special-purpose linearly homomorphic encryption (Definition 1).

Bounded Secret Keys: this is clear from the construction.

Linear Homomorphism: Note that one can write Δ exactly as $\frac{q+\delta}{p}$ for some integer $-p/2 \leqslant \delta \leqslant p/2$. Therefore,

$$\left(\sum_i a_i (e + e_{\mathsf{smudge}} + \Delta \cdot \overrightarrow{\mathsf{msg}}_i)\right) \pmod q \in \left(\sum_i a_i \cdot \overrightarrow{\mathsf{msg}}_i \pmod p\right) \cdot \Delta \pm B \cdot (B_{\mathsf{error}} + B_{\mathsf{smudge}} + p),$$

which implies

$$\left\lfloor \frac{\left(\sum_i a_i (e + e_{\mathsf{smudge}} + \Delta \cdot \overrightarrow{\mathsf{msg}}_i)\right) \pmod q}{\Delta} \right\rceil = \sum_i a_i \cdot \overrightarrow{\mathsf{msg}}_i \pmod p$$

as long as

$$B \cdot (B_{\mathsf{error}} + B_{\mathsf{smudge}} + p) = o(\Delta).$$

This is consistent with how we set the parameters.

Computationally-binding: In fact, one can prove statistical binding. It comes routinely from the fact that when p is large enough, the choice of sk such that $\|\mathsf{pk} - A_{\mathsf{pk}} \cdot \mathsf{sk}\|_\infty \leqslant B_{\mathsf{error}}$ is unique. See, for example, [Vai20].

Decryption-friendliness: The simulator proceeds as follows.

$$\mathsf{Sim}_1\left(1^\lambda, 1^m, B, p\right) = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{0}),\ \text{Output } \{\mathsf{pp}, \mathsf{ct}\} \end{array} \right\}.$$

$$\mathsf{Sim}_2\left(\{\mathsf{pp}, \mathsf{ct}\}, \left\{a_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i\right\}_{i \in \ell}\right) = \left\{ \begin{array}{l} \forall i \in [\ell],\ (\mathsf{sk}'_i, \mathsf{pk}'_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1) \\ \forall i \in [\ell],\ \mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{sk}'_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i) \\ \forall i \in [\ell],\ \mathsf{ct}_i = \mathsf{CTEval}((-a_i, 1), (\mathsf{ct}, \mathsf{ct}'_i)) \\ \text{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \text{ and } (\mathsf{sk}'_1, \ldots, sk'_\ell) \end{array} \right\}.$$

To prove the computational indistinguishability, define the following hybrids.

$$H_0 = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0) \\ \forall i \in [\ell],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1),\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}}) \\ \forall i \in [\ell],\ \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}_i, \overrightarrow{\mathsf{msg}}_i),\ \forall i \in [\ell],\ \mathsf{sk}'_i = \mathsf{SKEval}((a_i, 1), (\mathsf{sk}, \mathsf{sk}_i)) \\ \text{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \text{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array} \right\}.$$

$$H_1 = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0) \\ \forall i \in [\ell],\ (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1),\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}}) \\ \forall i \in [\ell],\ \forall i \in [\ell],\ \mathsf{sk}'_i = \mathsf{SKEval}((a_i, 1), (\mathsf{sk}, \mathsf{sk}_i)) \\ \mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{sk}'_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i),\ \forall i \in [\ell],\ \mathsf{ct}_i = \mathsf{CTEval}((-a_i, 1), (\mathsf{ct}, \mathsf{ct}'_i)) \\ \text{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \text{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array} \right\}.$$

$$H_2 = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0),\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{\mathsf{msg}}) \\ \forall i \in [\ell],\ (\mathsf{sk}'_i, \mathsf{pk}'_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1) \\ \mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{sk}'_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i),\ \forall i \in [\ell],\ \mathsf{ct}_i = \mathsf{CTEval}((-a_i, 1), (\mathsf{ct}, \mathsf{ct}'_i)) \\ \text{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \text{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array} \right\}.$$

$$H_3 = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p),\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, \overrightarrow{0}),\ \forall i \in [\ell],\ (\mathsf{sk}'_i, \mathsf{pk}'_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1) \\ \mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{sk}'_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i),\ \forall i \in [\ell],\ \mathsf{ct}_i = \mathsf{CTEval}((-a_i, 1), (\mathsf{ct}, \mathsf{ct}'_i)) \\ \text{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \text{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array} \right\}.$$

The first hybrid is exactly the real game, and the last hybrid is exactly the simulator $(\mathsf{Sim}_1, \mathsf{Sim}_2)$. The closeness is due to

- H_0 and H_1 are statistically close: The only difference is how ct′ are generated. In H_0, it is a homomorphic evaluation on two ciphertexts; in H_1, it is a freshly sampled encryption of the homomorphically evaluated plaintext. They are statistically close due to noise smudging (Lemma 1), i.e., e_{smudge} is superpolynomially larger than and hence will smudge both $\left((a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i) \cdot \Delta \pmod p - a_i \cdot (\overrightarrow{\mathsf{msg}} \cdot \Delta) - \overrightarrow{\mathsf{msg}}_i \cdot \Delta\right) \pmod p$ (which is bounded by $B \cdot p$) and $a_i \cdot e$ (which is bounded by $B \cdot e_{\mathsf{error}}$).

$$SD(H_0, H_1) \le \sum_\ell |a_i| \cdot m \cdot (B_{\mathsf{error}} + p)/B_{\mathsf{smudge}} < m \cdot B \cdot (B_{\mathsf{error}} + p)/B_{\mathsf{smudge}} < m \cdot \lambda^{-\omega(1)}$$

- H_1 and H_2 are statistically close: The only difference is in how the secret key sk′ is sampled. Again, they are statistically close because $\mathsf{sk}_i \leftarrow [B_{\mathsf{sk}}]^n$, where B_{sk} is superpolynomially large, will smudge $a_i \cdot \mathsf{sk}$ where $\mathsf{sk} \leftarrow \{0,1\}^n$ (Lemma 1).

$$SD(H_0, H_1) \le \sum_i n \cdot |a_i|/(B_{\mathsf{sk}} - 1) \le \ell \cdot n \cdot B/(B_{\mathsf{sk}} - 1) = \ell \cdot n \cdot B^{-\omega(1)}$$

- H_2 and H_3 are computationally close: The only difference is due to switching encryption of $\overrightarrow{\mathsf{msg}}$ to encryption of $\overrightarrow{0}$. This is computationally indistinguishable due to the LWE assumption with small secrets (Definition 3). □

3.4 Instantiation Based on DCR

In this section, we present our DCR-based instantiation of special-purpose linearly homomorphic encryption (see Definition 1).

Definition 4 (Decisional Composite Residuosity assumption). *Let $\mathcal{S}$ be the set of λ-bit safe primes.*[17] *Define* $\mathsf{DCR.Setup}(1^\lambda)$ *as* $p_1, p_2 \leftarrow \mathcal{S}$, $N = p_1 \cdot p_2$. *For a polynomial $c(\lambda)$, the c-DCR assumption states that*

$$(N, g^2) \;\approx\; (N, g^{2N^c}),$$

where $(N, p_1, p_2) \leftarrow \mathsf{DCR.Setup}(1^\lambda)$ *and* $g \leftarrow \mathbb{Z}^*_{N^{c+1}}$.

Let $\mathbb{G}$ be the subgroup of $\mathbb{Z}^*_{N^{c+1}}$ generated by $1+N$. We follow [RS21] and define the following functions $\exp : \mathbb{Z}_{N^c} \to \mathbb{G}$ and $\log : \mathbb{G} \to \mathbb{Z}_{N^c}$.

$$\exp(x) \coloneqq \sum_{i=0}^{c} \frac{(Nx)^i}{i!} \qquad \text{and} \qquad \log(1+Nx) \coloneqq \sum_{i=1}^{c} \frac{(-N)^{i-1} \cdot x^i}{i}.$$

They enjoy the following properties (refer to [RS21] for a proof):

- They are inverse functions of each other, i.e., $\log(\exp(x)) = x$ and $\exp(\log(1+Nx)) = 1+Nx$.
- They are group homomorphisms, i.e., $\exp(x+y) = \exp(x) \cdot \exp(y)$ and $\log(1+Nx) + \log(1+Ny) = \log((1+Nx) \cdot (1+Ny))$.

Construction. The construction of special-purpose linearly homomorphic encryption (see Definition 1) proceeds as follows.

- $\mathsf{Setup}(1^\lambda, 1^m, B, p) \to \mathsf{pp}$: Set $B_{\mathsf{smudge}} = B \cdot p \cdot \lambda^{\omega(1)}$. Sample $(N, p_1, p_2) \leftarrow \mathsf{DCR.Setup}(1^\lambda)$. Pick a large enough c such that: set $\Delta = \lfloor N^c/p \rceil$ and it holds that

  $$\Delta = B \cdot (B_{\mathsf{smudge}} + p) \cdot \lambda^{\omega(1)}.$$

  Let

  $$\widehat{g} \leftarrow \mathbb{Z}^*_{N^{c+1}}, \qquad g = (\widehat{g})^{2N^c}.$$

  For every $i \in [m]$, sample $g_i = (h_i)^{2N^c}$, where $h_i \leftarrow \mathbb{Z}^*_{N^{c+1}}$. Set $\mathsf{pp} = (N, c, B_{\mathsf{smudge}}, \Delta, g, g_1, \ldots, g_m)$.

- $\mathsf{KeyGen}(1^\lambda, b) \to (b, \mathsf{sk}, \mathsf{pk})$: If $b = 0$, i.e., we are sampling normal secret key, then $\mathsf{sk} \leftarrow [N/4]$ and $\mathsf{pk} = g^{\mathsf{sk}}$. If $b = 1$, i.e., we are sampling smudging secret key, then $\mathsf{sk} \leftarrow [(N/4) \cdot \lambda^{\omega(1)}]$ and $\mathsf{pk} = g^{\mathsf{sk}}$.

- $\mathsf{Enc}(\mathsf{sk}, b, \mathsf{msg}_1, \ldots, \mathsf{msg}_m) \to (\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$: If the secret key is a normal secret key, output

  $$(g_1^{\mathsf{sk}} \cdot \exp(\mathsf{msg}_1 \cdot \Delta), \ldots, g_1^{\mathsf{sk}} \cdot \exp(\mathsf{msg}_m \cdot \Delta)).$$

  Else, sample $e_i \leftarrow [-B_{\mathsf{smudge}}, B_{\mathsf{smudge}}]$; output

  $$(g_1^{\mathsf{sk}} \cdot \exp(\mathsf{msg}_1 \cdot \Delta + e_1), \ldots, g_1^{\mathsf{sk}} \cdot \exp(\mathsf{msg}_m \cdot \Delta + e_m)).$$

[17] p is a safe prime if $p = 2p' + 1$ and p' is also a prime.

- $\mathsf{Dec}(\mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to (\mathsf{msg}_1, \ldots, \mathsf{msg}_m)$: For all $i \in [m]$, let $\alpha_i = \log(\mathsf{ct}_i \cdot g_i^{-\mathsf{sk}})$. Output $\lfloor \alpha_i / \Delta \rceil$.
- SKEval, PKEval, and CTEval: For SKEval, output $\sum_i a_i \cdot \mathsf{sk}_i$ over $\mathbb{Z}$. For PKEval (resp., CTEval), output $\sum_i a_i \cdot \mathsf{pk}_i$ (resp., $\sum_i a_i \cdot \mathsf{ct}_i$) over $\mathbb{Z}_{N^{c+1}}$.
- Verify: Output 1 if and only if $g^{\mathsf{sk}} = \mathsf{pk}$ over $\mathbb{Z}_{N^{c+1}}$.

Theorem 2. *Assuming the DCR assumption (Definition 4) holds, the construction above is a special-purpose linear homomorphic encryption scheme according to Definition 1.*

Proof. We sketch a proof showing that the construction above satisfies all the properties of special-purpose linearly homomorphic encryption (Definition 1).

Bounded Secret Keys: this is clear from the construction.

Linear Homomorphism: Note that one can write Δ exactly as $\frac{N^c+\delta}{p}$ for some integer $-p/2 \leqslant \delta \leqslant p/2$. Therefore,

$$\left(\sum_i a_i (e_{\mathsf{smudge}} + \Delta \cdot \overrightarrow{\mathsf{msg}}_i)\right) \pmod{N^c} \in \left(\sum_i a_i \cdot \overrightarrow{\mathsf{msg}}_i \pmod{p}\right) \cdot \Delta \pm B \cdot (B_{\mathsf{smudge}} + p),$$

which implies

$$\left\lfloor \frac{\left(\sum_i a_i (e_{\mathsf{smudge}} + \Delta \cdot \overrightarrow{\mathsf{msg}}_i)\right) \pmod{N^c}}{\Delta} \right\rceil = \sum_i a_i \cdot \overrightarrow{\mathsf{msg}}_i$$

as long as

$$B \cdot (B_{\mathsf{smudge}} + p) = o(\Delta).$$

This is consistent with how we set the parameters.

Computationally-binding: Note that g has an order of $\frac{p_1-1}{2} \cdot \frac{p_2-1}{2}$. If the adversary can output two sk' and sk'' such that $g^{\mathsf{sk}'} = g^{\mathsf{sk}''}$, then we know that $\mathsf{sk}' - \mathsf{sk}''$ is divisible by $\frac{p_1-1}{2} \cdot \frac{p_2-1}{2}$. It is well-understood that this information can be leveraged to factor N and hence, break the DCR assumption. For instance, refer to [HLM25, Claim 3.7] for a proof.

Decryption-friendliness: The simulator proceeds as follows.

$$\mathsf{Sim}_1(\lambda, 1^m, B, p) = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^m, B, p) \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda, b = 0) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(b, \mathsf{sk}, \overrightarrow{0}) \\ \text{Output } \{st, \mathsf{pp}, \mathsf{ct}\} \end{array} \right\}.$$

$$\mathsf{Sim}_2\left(st, \mathsf{pp}, \mathsf{ct}, \left\{a_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i\right\}_{i \in [\ell]}\right) = \left\{ \begin{array}{l} \forall i \in [\ell],\ (\mathsf{sk}'_i, \mathsf{pk}'_i) \leftarrow \mathsf{KeyGen}(1^\lambda, b_i = 1) \\ \forall i \in [\ell],\ \mathsf{ct}'_i \leftarrow \mathsf{Enc}(\mathsf{sk}'_i, a_i \cdot \overrightarrow{\mathsf{msg}} + \overrightarrow{\mathsf{msg}}_i) \\ \forall i \in [\ell],\ \mathsf{ct}_i = \mathsf{CTEval}((-a_i, 1), (\mathsf{ct}, \mathsf{ct}'_i)) \\ \text{Output } (\mathsf{ct}, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \text{ and } (\mathsf{sk}'_1, \ldots, \mathsf{sk}'_\ell) \end{array} \right\}.$$

We can prove it in the same way as the proof in the LWE instantiation. First, we switch the homomorphically evaluated ciphertexts to a freshly sampled one; then switch the homomorphically evaluated key to a freshly sampled smudging key; finally switch ct from encrypting $\overrightarrow{\mathsf{msg}}$ to encryption $\overrightarrow{0}$. The first two switches are statistically indistinguishable due to noise smudging (Lemma 1). The last switch is computationally indistinguishable due to the one-time security of the Damgård-Jurik encryption scheme [DJ01, MORS25]. □

4 Computational Secret Sharing with Linear Homomorphism

In this section, we present the definition and construction of a computational secret sharing scheme with linear homomorphism for general access structures. As mentioned earlier, our construction works for any access structures that can be represented as a monotone Boolean circuit and its complexity grows linearly in the circuit size.

4.1 Basics of Monotone Circuits and Access Structures

We start by recalling the definitions of monotone Boolean circuits and access structures.

Definition 5 (Monotone Boolean Circuit)**.** *A monotone Boolean circuit $C : \{0,1\}^n \to \{0,1\}$ is a Boolean circuit with unbounded fan-in and fan-out such that it consists of only* AND *and* OR *gates. In particular, it does not contain* NOT *gates.*

Definition 6 (Monotone Weighted Threshold Function)**.** *A Boolean function $f : \{0,1\}^n \to \{0,1\}$ is called a monotone weighted threshold function if there exist positive weights $w_1, \ldots, w_n \in \mathbb{R}^+$ and threshold $T \in \mathbb{R}$ such that $f(x_1, \ldots, x_n) = 1$ if and only if $\sum_{i=1}^{n} w_i \cdot x_i \geqslant T$.*

It is known [BW06] that any monotone weighted threshold function can be realized by a monotone Boolean circuit with (1) circuit size polynomial in n and independent of the weights w_i and (2) circuit depth $O(\log n)$, which we summarize as following imported theorem.

Theorem 3 ([BW06])**.** *There is a universal polynomial* $\mathsf{poly}(n)$ *such that, for every monotone weighted threshold function $f : \{0,1\}^n \to \{0,1\}$, there exists a monotone Boolean circuit C realizing it, where* $\mathsf{Size}(C) \leqslant \mathsf{poly}(n)$ *and* $\mathsf{Depth}(C) = O(\log n)$.

We also recall the definition of general access structures.

Definition 7 (Access Structures)**.** *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n parties, and let $2^{\mathcal{P}}$ denote its power set. A collection of subsets $\Gamma \subseteq 2^{\mathcal{P}}$ is said to be* monotone *if for all $A, B \subseteq \mathcal{P}$,*

$$A \in \Gamma \text{ and } A \subseteq B \implies B \in \Gamma.$$

Any monotone collection Γ of non-empty subsets of $\mathcal{P}$ is called an access structure *on $\mathcal{P}$. The sets in Γ are referred to as* authorized sets*, while the sets in $2^{\mathcal{P}} \setminus \Gamma$ are referred to as* unauthorized sets*.*

4.2 Definition

Next, we formalize the notion of a computational secret sharing scheme with linear homomorphism for general access structures. In line with our definition of special-purpose linearly homomorphic encryption,

we assume that the share algorithm operates in two modes, producing either *normal secret sharing* or *smudged secret sharing*. As in the encryption setting, privacy of the underlying secrets is required only when the linear combination of shares involves at least one secret shared in the smudged mode. Looking ahead, this restrictive notion of privacy is sufficient for our applications to MPC, threshold encryption, and threshold signature.

For our definition and construction, without loss of generality, we consider a modified access structure wherein each party is an input to an OR gate with another input 0. This ensures that the final private share that each party receives is a full LHE secret key and not an additive share of it. As a result, the public key corresponding to it can be used for private verification of the share.

Definition 8 (LH-CSS)**.** *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n parties, let $\Gamma \subseteq 2^{\mathcal{P}}$ be an access structure (Definition 7) on $\mathcal{P}$, and let $\mathbb{Z}_p$ denote the message space. A computational secret sharing scheme for secrets in $\mathbb{Z}_p$ with respect to (n, Γ) and supporting linear homomorphism consists of a tuple of* PPT *algorithms* $(\mathsf{Setup}, \mathsf{Share}, \mathsf{Recon}, \mathsf{Eval}, \mathsf{Verify})$ *as follows.*

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p)$ *: A randomized algorithm that takes as input the security parameter λ, number of parties n, access structure Γ and message space $\mathbb{Z}_p$, and outputs a public parameter* pp*. We assume that the remaining algorithms implicitly take* pp *as input.*

- $\{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b)$*: A randomized algorithm that takes as input a secret $s \in \mathbb{Z}_p$ and a bit $b \in \{0, 1\}$, and outputs a tuple $((\hat{s}_1, \ldots, \hat{s}_n), \delta)$, where $\hat{s}_i$ is the private share given to the ith party and δ is a public value provided to all parties. If $b = 0$, the resulting shares are called* normal shares*. If $b = 1$, the resulting shares are called* smudged shares*.*

- $s \leftarrow \mathsf{Recon}(\mathcal{T}, \{\hat{s}_i\}_{i \in \mathcal{T}}, \delta)$*: A deterministic algorithm that takes as input an authorized set of parties $\mathcal{T} \subseteq \mathcal{P}$ (i.e., $\mathcal{T} \in \Gamma$), their corresponding private shares $\{\hat{s}_i\}_{i \in \mathcal{T}}$, and the associated public value δ, and outputs the reconstructed secret $s \in \mathbb{Z}_p \cup \{\perp\}$.*

- $(\hat{s}_i, \delta) \leftarrow \mathsf{Eval}(i, (a_1, \ldots, a_\ell), ((\hat{s}_{i,1}, \delta_1), \ldots, (\hat{s}_{i,\ell}, \delta_\ell)))$*: A deterministic algorithm that takes as input a party index $i \in [n]$, a linear function defined by coefficients $a_1, a_2, \ldots, a_\ell \in \mathbb{Z}$, and the ith party's shares $((\hat{s}_{i,1}, \delta_1), \ldots, (\hat{s}_{i,\ell}, \delta_\ell))$ corresponding to ℓ shared secrets. It outputs a new share $(\hat{s}_i, \delta)$.*

  Remark. *Since the δ_i values are public to all parties, the aggregated value δ can be computed publicly.*

- $b \leftarrow \mathsf{Verify}(i, \delta, \hat{s}_i)$*: A deterministic algorithm that takes as input a party index $i \in [n]$, its corresponding private share $\hat{s}_i$, and the associated public value δ, and outputs a bit $b \in \{0, 1\}$.*

  These algorithms should satisfy the following properties:

  1. ***Linear Homomorphism.*** *For all $\ell \in \mathbb{N}$, all parameters λ, n, p, all access structures Γ, any secrets $s_1, \ldots, s_\ell \in \mathbb{Z}_p$, bits $b_1, \ldots, b_\ell \in \{0, 1\}$, any linear coefficients $a_1, \ldots, a_\ell \in \mathbb{Z}$ such that $\sum_i |a_i| \leqslant B$, and any authorized set of parties $\mathcal{T} \in \Gamma$ it holds that:*

$$\Pr\left[\mathsf{Recon}(\mathcal{T}, \{\hat{s}_i\}_{i \in \mathcal{T}}, \delta) = \sum_j a_j \cdot s_j \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \forall j \in [\ell], \{(\hat{s}_{1,j}, \ldots, \hat{s}_{n,j}), \delta\} \leftarrow \mathsf{Share}(s_j, b_j) \\ \forall i \in \mathcal{T}, (\hat{s}_i, \delta) \leftarrow \mathsf{Eval}(i, (a_1, \ldots, a_\ell), ((\hat{s}_{i,1}, \delta_1), \ldots, (\hat{s}_{i,\ell}, \delta_\ell))) \end{array}\right] = 1.$$

     Note that this implies the standard correctness.

2. ***Authenticated Shares.*** *For all parameters λ, n, p, all access structures Γ, any bit $b \in \{0,1\}$, any secret $s \in \mathbb{Z}_p$ and any PPT adversary $\mathcal{A}$, let* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p)$, $\mathcal{I} \leftarrow \mathcal{A}(\mathsf{pp})$, $\{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b)$, *and* $\{\hat{s}'_i\}_{i\in\mathcal{I}} \leftarrow \mathcal{A}(\mathsf{pp}, \{\hat{s}_i\}_{i\in\mathcal{I}}, \delta)$. *If $\mathcal{I} \in 2^{\mathcal{P}} \setminus \Gamma$, then the following holds for each $\mathcal{T} \in \Gamma$:*

$$\Pr\left[\begin{array}{l|l} \mathcal{J} \coloneqq \left\{ i \in \mathcal{I} \cap \mathcal{T} : \mathsf{Verify}(i, \delta, \hat{s}'_i) = 0 \right\} & \\ \qquad \wedge & \\ \left(\begin{array}{c} \mathcal{T} \setminus \mathcal{J} \notin \Gamma \ \wedge s' = \bot \\ \vee \\ \mathcal{T} \setminus \mathcal{J} \in \Gamma \ \wedge s' = s \end{array} \right) & s' \leftarrow \mathsf{Recon}\big(\mathcal{T}, \{\hat{s}_i\}_{i\in\mathcal{T}\setminus\mathcal{I}}, \{\hat{s}'_i\}_{i\in\mathcal{T}\cap\mathcal{I}}, \delta\big) \end{array}\right] \geqslant 1 - \mathsf{negl}(\lambda)$$

3. ***Reconstruction-friendliness.*** *For all parameters λ, n, p, all access structures Γ, any $\ell + 1$ secrets $s, s_1, \ldots, s_\ell \in \mathbb{Z}_p$, any linear coefficients $a_1, \ldots, a_\ell \in \mathbb{Z}$, any authorized set of parties $\mathcal{T} \in \Gamma$, and any PPT adversary $\mathcal{A}$ corrupting an unauthorized set $\mathcal{I} \notin \Gamma$ of parties, there exists a pair of simulators* $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$, *such that the following two distributions are computationally close.*

   (a) Real Distribution:

$$\left\{\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b = 0) \\ \forall j \in [\ell] : \ \{(\hat{s}_{1,j}, \ldots, \hat{s}_{n,j}), \delta_j\} \leftarrow \mathsf{Share}(s_j, b = 1) \\ \forall i \in \mathcal{T}, \ j \in [\ell] : \ (\hat{s}'_{i,j}, \delta'_j) \leftarrow \mathsf{Eval}\big(i, (a_j, 1), ((\hat{s}_i, \delta), (\hat{s}_{i,j}, \delta_j))\big) \\ \textit{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta), \ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]}, \ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array}\right\}$$

   (b) Simulated Distribution:

$$\left\{\begin{array}{c} \{\mathsf{pp}, (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta)\} \leftarrow \mathsf{Sim}_1(1^\lambda, 1^n, \Gamma, p) \\ \left\{(\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j), \ (\{\hat{s}'_i\}_{i\in\mathcal{T}}, \delta') \leftarrow \mathsf{Sim}_2\left(\mathsf{pp}, (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta), a_j, a_j \cdot s + s_j\right)\right\}_{j=1}^{\ell} \\ \textit{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta), \ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]}, \ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array}\right\}$$

   Similar to the encryption case, this property only holds when combining normal shares with smudged shares. In particular, the coefficient on secrets shared in the smudged mode must be equal to 1.

   Finally, we note that Sim_2 *can be invoked either on the share set* $(\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta)$ *output by* Sim_1, *or on a share set that was honestly generated via* $\mathsf{Share}(\cdot, b = 0)$. *Our definition implies this since the two stages of the simulator do not share any secret state.*

Remark 1. *Note that our reconstruction-friendliness definition implies the standard privacy requirement for a computational secret sharing scheme. That is, for any two secrets s_0, s_1, for any unauthorized set $\mathcal{I}$, $\{\hat{s}_{i,0}\}_{i\in\mathcal{I}} \approx \{\hat{s}_{i,1}\}_{i\in\mathcal{I}}$. This is because, reconstruction-friendliness guarantees the existence of* Sim_1 *which can simulate a set of unauthorized shares without knowing the underlying secret s.*

4.3 Construction

In this section, we give a construction of LH-CSS using our special-purpose LHE scheme (LHE.Setup, LHE.KeyGen, LHE.Enc, LHE.Dec, LHE.SKEval, LHE.PKEval, LHE.CTEval, LHE.Verify). Let C be the circuit

realizing the access structure Γ, with depth d and number of nodes g. The gates of the circuit together with the inputs, are referred to as nodes. The input nodes are the n parties in $\mathcal{P}$ denoted by the labels $\mathsf{G}_1, \mathsf{G}_2, \ldots, \mathsf{G}_n$ whereas the gates are labeled with $\mathsf{G}_{n+1}, \ldots, \mathsf{G}_g$ with G_g being the final output gate.

The Construction.

- **Setup.** $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p)$:
  - Sample public parameter $\mathsf{pp}_i \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^\beta, B, p)$ for each node G_i in the circuit. These parameters are chosen as follows:
    - B denotes the upper bound associated with linear homomorphism correctness (note that this is a global parameter)
    - $\mathbb{Z}_p$ is the message space at every gate. Here we rely on the fact that LHE secret keys are integers with a maximum bound independent of the message space and p is appropriately chosen large enough to embed the keys whenever they are encrypted as messages.
- **Sharing.** $\{(\hat{s_1}, \hat{s_2}, \ldots, \hat{s_n}), \delta\} \leftarrow \mathsf{Share}(s, b)$:
  - Initialize a string $\mathsf{str}_i := \varnothing$ for each of the nodes G_i in C, $i \in [g]$.
  - Set the string corresponding to the output gate G_g as $\mathsf{str}_g := s$.
  - For $i = g$ to $i = 1$, do the following:
    - Run $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{LHE.KeyGen}(1^\lambda, \mathsf{pp}_i, b)$ for each of the gates G_i with the corresponding public parameters pp_i and the bit b.
    - Compute $\mathsf{ct}_i \leftarrow \mathsf{LHE.Enc}(\mathsf{sk}_i, \mathsf{str}_i)$. Note that, str_i is a concatenation of f_i strings (i.e., messages), where f_i is its fan-out.
    - Let $\mathsf{G}_{j_1}, \ldots, \mathsf{G}_{j_{g_i}}$ be the sources of the incoming edges into node G_i, where $g_i \geqslant 1$ and $j_1 < \cdots < j_{g_i} < i$.
    - If G_i is an AND gate, update the strings of each of its child nodes as $\mathsf{str}_{j_\alpha} := \mathsf{str}_{j_\alpha}, [\mathsf{sk}_i]_\alpha$ where $\mathsf{sk}_i = \sum_{\alpha=1}^{g_i} [\mathsf{sk}_i]_\alpha$ in $\mathbb{Z}_p$.
    - If G_i is an OR gate, update the strings of its child nodes as $\mathsf{str}_{i_j} := \mathsf{str}_{i_j}, \mathsf{sk}_i$.
    - Set $i = i - 1$ if $i > 1$
  - Set $\delta = \big(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_g, \mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n)\big)$.
  - Output the shares $\{(\hat{s_1}, \hat{s_2}, \ldots, \hat{s_n}), \delta\}$ where $\hat{s_i} := \mathsf{sk}_i$ is the i-th share, and δ is public information.
- **Reconstruction.** $s \leftarrow \mathsf{Recon}(\mathcal{T} \in \Gamma, \{\hat{s_i}\}_{i \in \mathcal{T}}, \delta)$:
  - $(\mathsf{ct}_1, \ldots, \mathsf{ct}_g, \mathsf{pk}) := \delta$
  - Initialize a set $\mathcal{J} := \varnothing$.
  - Parse pk as $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. For all $i \in \mathcal{T}$, run $\mathsf{LHE.Verify}(\hat{s}_i, \mathsf{pk}_i)$ and if the output is 0, do $\mathcal{J} = \mathcal{J} \cup i$.
  - Return $\perp$ if $\mathcal{T} \setminus \mathcal{J} \notin \Gamma$.
  - For $i = 1$ to $i = n$, parse $\hat{s_i}$ as sk_i and compute $\mathsf{str}_i = \mathsf{LHE.Dec}(\mathsf{sk}_i, \mathsf{ct}_i)$.
  - Let C_i denote the sub-circuit with output gate G_i.
  - For the gates $i = n + 1$ to $i = g$, do the following:

- If $\mathcal{G}_i$ is an AND gate and $\mathsf{C}_i(\mathcal{T}) = 1$, for all child nodes G_{j_α} it must be true that $\mathsf{C}_{j_\alpha}(\mathcal{T}) = 1 \; \forall \; \alpha \in [g_i]$.
  - For $\alpha \in [g_i]$ do the following: Fetch $[\mathsf{sk}_i]_\alpha$ from str_{j_α}.
  - Compute $\mathsf{sk}_i = \sum_{\alpha=1}^{g_i} [\mathsf{sk}_i]_\alpha$.
  - Compute $\mathsf{str}_i = \mathsf{LHE.Dec}(\mathsf{sk}_i, \mathsf{ct}_i)$.
- If $\mathcal{G}_i$ is an OR gate and $\mathsf{C}_i(\mathcal{T}) = 1$, $\exists$ a child node G_{j_α} such that $\mathsf{C}_{j_\alpha}(\mathcal{T}) = 1$ for some $\alpha \in [g_i]$.
  - Fetch sk_i from str_{j_α}.
  - Compute $\mathsf{str}_i = \mathsf{LHE.Dec}(\mathsf{sk}_i, \delta_i)$.

- Output $s := str_g$.

- **Linear Evaluation.** $(\hat{s}_i, \delta) \leftarrow \mathsf{Eval}(i, a_1, a_2, ((\hat{s}_{i,1}, \delta_1), (\hat{s}_{i,2}, \delta_2))$:
  - Compute $\hat{s}_i \leftarrow \mathsf{SKEval}(a_1, a_2, \hat{s}_{i,1}, \hat{s}_{i,2})$
  - Compute $\mathsf{ct}_j \leftarrow \mathsf{CTEval}(a_1, a_2, \mathsf{ct}_{1,j}, \mathsf{ct}_{2,j})$ for all $j \in [g]$.
  - Compute $\mathsf{pk}_i \leftarrow \mathsf{PKEval}(a_1, a_2, \mathsf{pk}_{1,i}, \mathsf{pk}_{2,i})$.
  - Output $(\hat{s}_i, \delta)$ where $\delta = (\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_g, \; \mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n))$.
- **Verification.** $0/1 \leftarrow \mathsf{Verify}(i, \hat{s}_i, \delta)$:
  - $(\mathsf{ct}_1, \ldots, \mathsf{ct}_g, \mathsf{pk}) := \delta$ and $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n) := \mathsf{pk}$
  - Parse $\hat{s}_i$ as sk_i.
  - Output $\mathsf{LHE.Verify}(\mathsf{sk}_i, \mathsf{pk}_i)$.

Next, we show that our construction satisfies the required correctness and security properties as per Definition 8.

Theorem 4. *The construction above is a linearly-homomorphic computational secret sharing scheme per Definition 8.*

Proof. We prove that our construction from Section 4.3 satsifies the definition of computational secret sharing with linear homomorphism (Definition 8).

Linear Homomorphism. To see that the construction above satisfies *linear homomorphism*, consider two secrets s_1 and s_2 secret shared among the n parties in $\mathcal{P}$ using our LH-CSS algorithm. Let the individual shares be denoted by $\{(\hat{s}_{1,i}), \delta_1\}$ and $\{(\hat{s}_{2,i}), \delta_2\}$, respectively. On running $\mathsf{Eval}(i, a_1, a_2, \hat{s}_{1,i}, \hat{s}_{2,i})$ we get,

- $\hat{s}_i \leftarrow \mathsf{SKEval}(a_1, a_2, \hat{s}_{i,1}, \hat{s}_{i,2})$
- $\mathsf{ct}_j \leftarrow \mathsf{CTEval}(a_1, a_2, \mathsf{ct}_{j,1}, \mathsf{ct}_{j,2})$ for all $j \in [g]$; $\delta = (\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_g, \mathsf{pk})$
- From *linear homomorphism* of the LHE (Def. 1), we have that $\mathsf{LHE.Dec}(\hat{s}_i, \mathsf{ct}_i) = a_1 \cdot \mathsf{str}_{i,1} \; + \; a_2 \cdot \mathsf{str}_{i,2}$ since $\mathsf{ct}_{i,1}$ and $\mathsf{ct}_{i,2}$ are encryptions of $\mathsf{str}_{i,1}$ and $\mathsf{str}_{i,2}$, respectively.

On running $\mathsf{Recon}(\mathcal{T}, \{(\hat{s}_i, \delta)\}_{i \in \mathcal{T}})$ on these shares we get,

- For all node $i \in [g]$, if $i \leqslant n$, compute $\mathsf{str}_i = \mathsf{LHE.Dec}(\hat{s}_i, \mathsf{ct}_i) = a_1 \cdot \mathsf{str}_{i,1} \; + \; a_2 \cdot \mathsf{str}_{i,2}$.
- For $n + 1 \leqslant i \leqslant g$, do the following:

- If $\mathcal{G}_i$ is an AND gate and $\mathsf{C}_i(\mathcal{T}) = 1$, fetch $[\mathsf{sk}_i]_\alpha = a_1 \cdot [\mathsf{sk}_{i,1}]_\alpha + a_2 \cdot [\mathsf{sk}_{i,2}]_\alpha$ from its child nodes and compute $\mathsf{sk}_i = \bigoplus_{\alpha=1}^{g_i} [\mathsf{sk}_i]_\alpha = a_1 \cdot \mathsf{sk}_{i,1} + a_2 \cdot \mathsf{sk}_{i,2}$. Compute $\mathsf{LHE.Dec}(\mathsf{sk}_i, \delta_i)$ to get $\mathsf{str}_i = a_1 \cdot \mathsf{str}_{i,1} + a_2 \cdot \mathsf{str}_{i,2}$.
- If $\mathcal{G}_i$ is an OR gate and $\mathsf{C}_i(\mathcal{T}) = 1$, fetch $\mathsf{sk}_i = a_1 \cdot \mathsf{sk}_{i,1} + a_2 \cdot \mathsf{sk}_{i,2}$ from one of its child nodes and compute $\mathsf{LHE.Dec}(\mathsf{sk}_i, \delta_i)$ to get $\mathsf{str}_i = a_1 \cdot \mathsf{str}_{i,1} + a_2 \cdot \mathsf{str}_{i,2}$.

- Eventually compute decryption of δ_g under the key $\mathsf{sk}_g = a_1 \cdot \mathsf{sk}_{g,1} + a_2 \cdot \mathsf{sk}_{g,2}$. Again from the *linear homomorphism* of LHE it follows that this decrypts to $a_1 \cdot \mathsf{str}_{g,1} + a_2 \cdot \mathsf{str}_{g,2}$ which is nothing but the value $a_1 \cdot s_1 + a_2 \cdot s_2$.

Authenticated Shares. Our LH-CSS scheme gives an authenticated computational secret sharing scheme as per Definition 8. This is because for any s,

$$\mathsf{LHE.Verify}(\hat{s}_i, \mathsf{pk}_i) = 1 \; \forall i \in [n]$$

where $((\hat{s}_1, \ldots, \hat{s}_n), \delta) \leftarrow \mathsf{Share}(s, b)$ and $\delta = \big(\mathsf{ct}_1, \mathsf{ct}_2, \ldots, \mathsf{ct}_g, \mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_n)\big)$.
Thus for valid shares

$$\mathsf{Verify}(i, \hat{s}_i, \delta) = 1$$

Reconstruction-friendliness. In order to show that our LH-CSS scheme satisfies the *reconstruction-friendliness* property as stated in Definition 8, we construct two simulators Sim_1 and Sim_2 such that for any $\ell + 1$ secrets $s, s_1, \ldots, s_\ell$, any coefficients $a_1, \ldots, a_\ell \in \mathbb{Z}$ and any authorized set $\mathcal{T}$, the following real output distribution is computationally indistinguishable from the one output by $(\mathsf{Sim}_1, \mathsf{Sim}_2)$:

$$\left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b = 0) \\ \forall j \in [\ell] : \; \{(\hat{s}_{1,j}, \ldots, \hat{s}_{n,j}), \delta_j\} \leftarrow \mathsf{Share}(s_j, b = 1) \\ \forall i \in \mathcal{T}, \; j \in [\ell] : \; (\hat{s}'_{i,j}, \delta'_j) \leftarrow \mathsf{Eval}\big(i, (a_j, 1), ((\hat{s}_i, \delta), (\hat{s}_{i,j}, \delta_j))\big) \\ \text{Output: } (\{\hat{s}_i\}_{i \in \mathcal{I}}, \delta), \; (\{\hat{s}_{i,j}\}_{i \in \mathcal{I}}, \delta_j)_{j \in [\ell]}, \; (\{\hat{s}'_{i,j}\}_{i \in \mathcal{T}}, \delta'_j)_{j \in [\ell]} \end{array} \right\}$$

Let $\mathcal{T}$ denote an authorized set of which $\mathcal{I} \notin \Gamma$ is the set of corrupt parties. The simulators $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ work as follows:

$\underline{\mathsf{Sim}_1(1^\lambda, 1^n, \Gamma, p)}$:

- Sim_1 runs setup to sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p)$ as input.
- Generate normal shares of $s = 0$ as $\{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s = 0, b = 0)$.
- Set $\big\{\mathsf{pp}, (\hat{s}_1, \ldots, \hat{s}_{|\mathcal{I}|})\big\}$

$\underline{\mathsf{Sim}_2(\mathsf{pp}, (\hat{s}_1, \ldots, \hat{s}_{|\mathcal{I}|}), a_j, a_j \cdot s + s_j)}$:

- Generates smudged shares of $a_j \cdot s + s_j$ as $\{(\hat{s}'_{1,j}, \ldots, \hat{s}'_{n,j}), \delta'_j\} \leftarrow \mathsf{Share}(a_j \cdot s + s_j, b = 1)$.
- For each $i \in [|\mathcal{I}|]$, set $(\hat{s}_{i,j}, \delta_j) := \mathsf{Eval}(i, (-a_j, 1), (\hat{s}_i, \delta), (\hat{s}'_{i,j}, \delta'_j))$

- Output $(\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta)$, $(\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)$, $(\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)$.

$$(\mathsf{Sim}_1, \{\mathsf{Sim}_2\}_{j=1}^{\ell}) = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(0, b=0) \\ \forall j \in [\ell] : \{(\hat{s}'_{1,j}, \ldots, \hat{s}'_{n,j}), \delta'_j\} \leftarrow \mathsf{Share}(a_j s + s_j, b=1) \\ \forall i \in \mathcal{I},\ j \in [\ell] :: (\hat{s}_{i,j}, \delta_j) \leftarrow \mathsf{Eval}\big(i, (-a_i, 1), ((\hat{s}_i, \delta_i), (\hat{s}'_{i,j}, \delta'_j))\big) \\ \text{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta),\ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]},\ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array} \right\}$$

We define the following hybrids to show computational indistinguishability:

$$H_0 = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b=0);\ \{(\hat{s}_{1,j}, \ldots, \hat{s}_{n,j}), \delta_j\} \leftarrow \mathsf{Share}(s_j, b=1)\ \forall j \in [\ell] \\ \forall i \in \mathcal{T},\ j \in [\ell] :\ (\hat{s}'_{i,j}, \delta'_j) \leftarrow \mathsf{Eval}\big(i, (a_j, 1), ((\hat{s}_i, \delta), (\hat{s}_{i,j}, \delta_j))\big) \\ \text{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta),\ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]},\ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array} \right\}$$

$$H_1 = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b=0);\ \{(\hat{s}_{1,j}, \ldots, \hat{s}_{n,j}), \delta_j\} \leftarrow \mathsf{Share}(s_j, b=1)\ \forall j \in [\ell] \\ \text{Parse } \delta \text{ as } (e_1, \ldots, e_g) \\ \forall m \in [1] : \text{if } C_m(\mathcal{I}) = 0, \mathsf{str} = 0 \text{ ; } e_m := \mathsf{LHE.Enc}\,(k_m, \mathsf{str})\,,\ k_m \leftarrow \mathsf{LHE.KeyGen}(1^\lambda) \\ \forall i \in \mathcal{T},\ j \in [\ell] :\ (\hat{s}'_{i,j}, \delta'_j) \leftarrow \mathsf{Eval}\big(i, (a_j, 1), ((\hat{s}_i, \delta), (\hat{s}_{i,j}, \delta_j))\big) \\ \text{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta),\ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]},\ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array} \right\}$$

$$\vdots$$

$$H_g = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(s, b=0);\ \{(\hat{s}_{1,j}, \ldots, \hat{s}_{n,j}), \delta_j\} \leftarrow \mathsf{Share}(s_j, b=1)\ \forall j \in [\ell] \\ \text{Parse } \delta \text{ as } (e_1, \ldots, e_g) \\ \forall m \in [g] : \text{if } C_m(\mathcal{I}) = 0, \mathsf{str} = 0 \text{ ; } e_m := \mathsf{LHE.Enc}\,(k_m, \mathsf{str})\,,\ k_m \leftarrow \mathsf{LHE.KeyGen}(1^\lambda) \\ \forall i \in \mathcal{T},\ j \in [\ell] :\ (\hat{s}'_{i,j}, \delta'_j) \leftarrow \mathsf{Eval}\big(i, (a_j, 1), ((\hat{s}_i, \delta), (\hat{s}_{i,j}, \delta_j))\big) \\ \text{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta),\ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]},\ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array} \right\}$$

$$H_{g+1} = \left\{ \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p) \\ \{(\hat{s}_1, \ldots, \hat{s}_n), \delta\} \leftarrow \mathsf{Share}(0, b = 0);\ \{(\hat{s}'_{1,j}, \ldots, \hat{s}'_{n,j}), \delta'_j\} \leftarrow \mathsf{Share}(a_j s + s_j, b = 1)\ \forall j \in [\ell] \\ \text{Parse } \delta \text{ as } (e_1, \ldots, e_g) \text{ and } \delta'_j \text{ as } (e'_{1j}, \ldots, e'_{gj}) \\ \forall j \in [\ell], m \in [g] :\ \text{Set } e_{mj} = \mathsf{CTEval}\big(k, (1, -a_j), (e'_{mj}, e_m)\big) \\ \forall j \in [\ell], m \in [g] :\ \text{Set } k_{mj} = \mathsf{SKEval}\big(m, (1, -a_j), (k'_{mj}, k_m)\big) \\ \forall j \in [\ell],\ \forall i \in \mathcal{I} :\ \hat{s}_{i,j} := k_{ij}, \delta_j := (e_{1j}, \ldots, e_{gj}) \\ \text{Output: } (\{\hat{s}_i\}_{i\in\mathcal{I}}, \delta),\ (\{\hat{s}_{i,j}\}_{i\in\mathcal{I}}, \delta_j)_{j\in[\ell]},\ (\{\hat{s}'_{i,j}\}_{i\in\mathcal{T}}, \delta'_j)_{j\in[\ell]} \end{array} \right\}$$

The computational indistinguishability between the real distribution and the simulated distribution can be argued using the decryption-friendliness of our special purpose LHE.

First, note that our simulator does not sample the linearly combined secret sharing (i.e., sharing of $a_j \cdot s + s_j$) using Eval. Instead, it sampled it as if it were a fresh sharing of $a_j \cdot s + s_j$. The distribution of this fresh sharing is indistinguishable from the sharing generated by Eval due to the decryption-friendliness. This is because, for every linearly combined ciphertext (from a normal ciphertext and a smudged ciphertext), their distributions are indistinguishable from a freshly sampled encryption of a smudged ciphertext encryption of the same combined plaintext.

Now, we are going to argue that conditioned on the linearly combined secret sharing, the normal sharing of s can be switched to a normal sharing of 0. We show that this switch is computationally indistinguishable. Namely, we are going to switch the ciphertexts in the public part one by one from the real distribution in secret sharing of s to the simulated distribution of secret sharing of 0, we will only do this only for gates satisfying $C(\mathcal{I}) = 0$). These switches are computationally indistinguishable because of the decryption-friendliness of the LHE scheme which implies one-time security, it first switches the linearly combined ciphertexts to the freshly sampled (smudged) ciphertexts and then switches the normal ciphertexts from encrypting s to encrypting 0. □

If we instantiate Construction 4 with our special-purpose linear homomorphic encryption scheme for weighted threshold access structures with the monotone Boolean circuit in Theorem 3, we obtain the following corollary.

Corollary 1. *Assuming LWE or DCR, there exists an LH-CSS construction for weighted threshold access structures where the computation and communication complexity depends only (in a fixed polynomial) on the number of parties and is independent of the weights.*

5 Weighted Secure Multiparty Computation

We now show how our computational secret sharing with linear homomorphism (LH-CSS) can be used to obtain a secure multiparty computation for arbitrary access structures. For weighted threshold access structures, our techniques yield the first black-box MPC for general functions, with weight-independent complexity.

5.1 Definition

We begin by recalling the definition. As discussed in Section 1.1, when considering weight-independent complexity, designing semi-honest secure MPC protocols or protocols with security with abort is straightforward and therefore not interesting. Hence, we focus on achieving guaranteed output delivery in the presence of malicious adversaries.

Definition 9 (Secure Multiparty Computation for General Access Structures)**.** *Let $\mathcal{P} = \{P_1, \cdots, P_n\}$ be the set of parties and let $f : \mathbb{Z}_p^n \to \mathbb{Z}_p^*$ an n-input functionality that they wish to compute. Let $\Gamma \subseteq 2^{\mathcal{P}}$ be a monotone access structure (see Definition 7) on $\mathcal{P}$. An interactive protocol π securely realizes f in the presence of malicious adversaries, if $\exists$ a PPT simulator* Sim*, such that $\forall$ PPT malicious adversaries $\mathcal{A}$ corrupting any unauthorized set $\mathcal{I} \subset [n]$, $\mathcal{I} \notin \Gamma$ of parties, $\forall 1^\lambda \in \mathbb{N}$ and for all $\{x_i\}_{i \notin \mathcal{I}} \in \mathbb{Z}_p$, the following two distributions are computationally indistinguishable:*

- $\mathsf{Real}_{\pi,\mathcal{A}}(1^\lambda, \{x_i\}_{i\notin\mathcal{I}})$: *Run the protocol π using 1^λ as the security parameter and $\{x_i\}_{i\notin\mathcal{I}}$ as the inputs of the honest parties. The messages of the corrupt parties are chosen by $\mathcal{A}$. Let y_i denote the output of each honest party $P_i \notin \mathcal{I}$ and view_i denotes its view in this protocol. This experiment outputs $\{\{\mathsf{view}_i\}_{i\in\mathcal{I}}, \{y_i\}_{i\notin\mathcal{I}}\}$.*
- $\mathsf{Ideal}_{f,\mathsf{Sim}^{\mathcal{A}}}(1^\lambda, \{x_i\}_{i\notin\mathcal{I}})$: *Run $\mathsf{Sim}^{\mathcal{A}}$ until it extracts $\{x_i\}_{i\in\mathcal{I}}$, compute $(y_1, \ldots, y_n) \leftarrow f(x_1, \ldots, x_n)$. Then give $\{y_i\}_{i\in\mathcal{I}}$ to $\mathsf{Sim}^{\mathcal{A}}$ and let $\{\mathsf{view}_i^*\}_{i\in\mathcal{I}}$ its output. This experiment outputs $\left\{\{\mathsf{view}_i^*\}_{i\in\mathcal{I}}, \{y_i\}_{i\notin\mathcal{I}}\right\}$.*

We say that π achieves guaranteed output delivery*, if $[n] \setminus \mathcal{I} \in \Gamma$ and $\forall i \notin \mathcal{I}, y_i \neq \perp$.*

5.2 Construction

Here we present our construction. We assume that the parties are given access to two sub-functionalities: (1) one that can generated smudged LH-CSS shares of Beaver triples [Bea91] on-the-fly and (2) another that can generate *normal* and *smudged* shares of a random value. We show how these can be used to design an MPC protocol that achieves guaranteed output delivery for general access structures. Towards the end of this section, we demonstrate how these ideal sub-functionality can themselves be instantiated using a semi-honest secure MPC protocol (with weight-independent complexity).

Notation/Building Blocks. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n parties, and let $\Gamma \subseteq 2^{\mathcal{P}}$ be a monotone access structure (see Definition 7) on $\mathcal{P}$. Let $f : \mathbb{Z}_p^n \to \mathbb{Z}_p^*$ be the n-input functionality that the parties wish to compute and let C be an arithmetic circuit representing f. Let (SS.Setup, SS.Share, SS.Recon, SS.Eval) be an LH-CSS w.r.t. Γ. We describe our protocol in the $(f_{\mathsf{Beaver}}, f_{\mathsf{Rand}})$-hybrid model where these functionalities are described in Figures 2 and 3.

Protocol. Let $\mathsf{pp} \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^n, \Gamma, p)$ be the public parameters available to all parties as part of the CRS (common random string).

- **Input Sharing.** Each party instantiates a set of discarded parties $\mathcal{D} = \emptyset$. Let $x_i \in \mathbb{Z}_p$ denote the input held by party i. For each $i \in [n]$, the parties proceed as follows:
  - All parties invoke f_{Rand} and for each $j \in [n]$, party j obtains shares $(\hat{r}_{i,j}^{(0)}, \delta_i^{(0)})$ of a random value r_i.
  - Each party $j \in [n] \setminus i$ sends $\hat{r}_{i,j}^{(0)}$ to party i.

The functionality $f_{\mathsf{Beaver}}(\mathcal{P} = \{P_1, \ldots, P_n\}, \mathsf{pp})$

The functionality f_{Beaver}, running with parties $\{P_1, \ldots, P_n\}$ and public parameters pp proceeds as follows:

- Sample uniformly random values $a, b \leftarrow \mathbb{Z}p$ and compute $c = a \cdot b$.
- For each $w \in \{a, b, c\}$, compute $\{(\hat{w}_1, \ldots, \hat{w}_n), \delta^{(w)}\} \leftarrow \mathsf{SS.Share}(w, 1)$,
- For each $i \in [n]$ send $\Big((\hat{a}_i, \delta^{(a)}), (\hat{b}_i, \delta^{(b)}), (\hat{c}_i, \delta^{(c)})\Big)$ to party i.

Figure 2: Ideal Functionality for obtaining LH-CSS shares of Beaver Triples

The functionality $f_{\mathsf{Rand}}(\mathcal{P} = \{P_1, \ldots, P_n\}, \mathsf{pp})$

The functionality f_{Rand}, running with parties $\{P_1, \ldots, P_n\}$ and public parameters pp proceeds as follows:

- Sample a uniformly random value $r \leftarrow \mathbb{Z}_p$.
- Compute $\{(\hat{r}_1^{(0)}, \ldots, \hat{r}_n^{(0)}), \delta^{(0)}\} \leftarrow \mathsf{SS.Share}(r, 0)$
- For each $i \in [n]$, send $(\hat{r}_i^{(0)}, \delta^{(0)})$ to party i.

Figure 3: Ideal Functionality normal and smudged LH-CSS shares of random values

- For each $j \in [n] \setminus \mathcal{D}$, party i checks $\mathsf{SS.Verify}(j, \delta_i^{(0)}, \hat{r}_{i,j}^{(0)}) \stackrel{?}{=} 1$ and if the check fails, they update $\mathcal{D} \leftarrow \mathcal{D} \cup \{j\}$.
- Party i reconstructs these shares to obtain r_i and broadcasts the value $y_i \coloneqq x_i + r_i$ to all parties.
- For each $j \in [n]$, let $(\hat{1}_j, \delta')$ be the j-th party's share in some deterministic (i.e., non privacy-preserving) sharing of 1. Party j finally computes shares of x_i as: $\hat{x}_{i,j} \leftarrow \mathsf{SS.Eval}\big(i, (y_i, -1), \big((\hat{1}_j, \delta'), (\hat{r}_{i,j}, \delta_i^{(0)})\big)\big)$.

- **Circuit Evaluation.** For each gate in the circuit C, the parties proceed as follows:

1. **Addition:** For each $i \in [n]$, let $(\hat{x}_i, \delta^{(x)})$ and $(\hat{y}_i, \delta^{(y)})$ denote the shares held by party i corresponding to the two input wires of an addition gate. Party i locally computes its share of the output wire as

$$(\hat{z}_i, \delta^{(z)}) \leftarrow \mathsf{SS.Eval}\big(i, (1, 1), \big((\hat{x}_i, \delta^{(x)}), (\hat{y}_i, \delta^{(y)})\big)\big).$$

2. **Scalar Multiplication:** For each $i \in [n]$, let $(\hat{x}_i, \delta^{(x)})$ denote the share held by party i corresponding to the input wire and $\alpha \in \mathbb{Z}_p$ be the public scalar. Party i locally computes its share of the output wire as
$$(\hat{z}_i, \delta^{(z)}) \leftarrow \mathsf{SS.Eval}\big(i, (\alpha, 0), (\hat{x}_i, \delta^{(x)}), (0, \emptyset)\big).$$

3. **Multiplication:** For each $i \in [n]$, let $(\hat{x}_i, \delta^{(x)})$ and $(\hat{y}_i, \delta^{(y)})$ denote the shares held by party i corresponding to the two input wires of a multiplication gate. The parties invoke f_{Beaver} to obtain shares $\Big((\hat{a}_i, \delta^{(a)}), (\hat{b}_i, \delta^{(b)}), (\hat{c}_i, \delta^{(c)})\Big)$ of a random Beaver triple. Each party $i \in [n]$ proceeds as follows:
   - Computes the following and sends the resulting shares to all other parties.
   $$(\hat{d}_i, \delta^{(d)}) \leftarrow \mathsf{SS.Eval}\big(i, (1, -1), \big((\hat{x}_i, \delta^{(x)}), (\hat{a}_i, \delta^{(a)})\big)\big).$$
   $$(\hat{e}_i, \delta^{(e)}) \leftarrow \mathsf{SS.Eval}\big(i, (1, -1), \big((\hat{y}_i, \delta^{(y)}), (\hat{b}_i, \delta^{(b)})\big)\big).$$
   - For each $j \in [n] \setminus \mathcal{D}$, the party checks
   $$\mathsf{SS.Verify}(j, \delta^{(d)}, \hat{d}_j) \stackrel{?}{=} 1 \quad \text{and} \quad \mathsf{SS.Verify}(j, \delta^{(e)}, \hat{e}_j) \stackrel{?}{=} 1.$$
   If either check fails, it updates $\mathcal{D} \leftarrow \mathcal{D} \cup \{j\}$.
   - Denote $\mathcal{T} = [n] \setminus \mathcal{D}$ and compute
   $$d \leftarrow \mathsf{SS.Recon}(\mathcal{T}, \{\hat{d}_i\}_{i \in \mathcal{T}}, \delta^{(d)}) \quad \text{and} \quad e \leftarrow \mathsf{SS.Recon}(\mathcal{T}, \{\hat{e}_i\}_{i \in \mathcal{T}}, \delta^{(e)}).$$
   - Let $(\hat{1}_i, \delta^{(1)})$ be some deterministic (i.e., non privacy-preserving) shares of 1. Finally compute shares of the output wire as
   $$(\hat{z}_i, \delta^{(z)}) \leftarrow \mathsf{SS.Eval}\big(i, (1, d, e, d \cdot e), \big((\hat{c}_i, \delta^{(c)}), (\hat{b}_i, \delta^{(b)}), (\hat{a}_i, \delta^{(a)}), (\hat{1}_i, \delta^{(1)})\big)\big).$$

- **Output Reconstruction:** For each output wire, send shares $(\hat{z}_i, \delta^{(z)})$ of the corresponding wire to all parties.
  - Denote $\mathcal{T} = [n] \setminus \mathcal{D}$. For each $j \in \mathcal{T}$, the party checks
  $$\mathsf{SS.Verify}(j, \delta^{(z)}, \hat{z}_j) \stackrel{?}{=} 1.$$
  If the check fails, it updates $\mathcal{D} \leftarrow \mathcal{D} \cup \{j\}$.
  - Reconstructs
  $$z \leftarrow \mathsf{SS.Recon}(\mathcal{T}, \{\hat{z}_i\}_{i \in \mathcal{T}}, \delta^{(z)}).$$

  If an output wire corresponds to the result of an addition or scalar multiplication gate, the parties first compute a multiplication of this wire with 1 using Beaver triples. They then follow the above procedure of exchanging and reconstructing the shares.

Theorem 5. *This protocol satisfies Definition 9 of a secure multiparty computation protocol for general access structures that achieves guaranteed output delivery against malicious adversaries in the $(f_{\mathsf{Beaver}}, f_{\mathsf{Rand}})$-hybrid model.*

Proof. We prove that our construction from Section 5.2 satsifes Definition 9.

Let $\mathcal{I} \subset [n]$ be an unauthorized subset of parties that the adversary corrupts and let $\mathcal{H} = [n] \setminus \mathcal{I}$ denote the set of honest pareties. Let $(\mathsf{Sim}_1^{\mathsf{SS}}, \mathsf{Sim}_2^{\mathsf{SS}})$ be the simulators that exist from reconstruction-friendliness property of LH-CSS (see Definition 8). The simulator Sim for our MPC protocol will proceed as follows:

- *Simulating Input Sharing Phase:* Initialize an emptyset $\mathcal{D} = \emptyset$. This phase is run by the simulator almost exactly as in the real protocol, except that whenever the adversary invokes f_{Rand}, Sim simulates an honest implementation of f_{Rand}. Moreover, for each honest party $i \in \mathcal{H}$, Sim runs $\mathsf{Sim}_1^{\mathsf{SS}}(\mathsf{pp})$ to compute shares $(\{\hat{x}_{i,j}\}_{j\in\mathcal{I}}, \delta_i^{(x)})$ and sends them to the adversary.

  Assuming $\mathcal{H} \in \Gamma$, for each $i \in \mathcal{I}$, the simulator runs $x_i \leftarrow \mathsf{SS.Recon}(\mathcal{H}, \{\hat{x}_i\}_{i\in\mathcal{H}}, \delta_i^{(x)})$. If $x_i = \perp$, update $\mathcal{D} = \mathcal{D} \cup \{i\}$. Query the ideal functionality compute f on inputs $\{x_i\}_{i\in\mathcal{I}}$ to obtain the output out.

  Observe that at the end of this phase, the simulator knows the shares held by all the corrupt parties for each input wire.

- *Simulating Circuit Evaluation Phase:* For addition and scalar multiplication gates, the simulator leverages its knowledge of the corrupted parties' shares to aggregate them accordingly, thereby preserving the invariant that it knows the adversary's shares for all wires computed or simulated thus far. For multiplication gates, the simulator proceeds as follows:

  - The simulator samples uniformly random values $d, e \leftarrow \mathbb{Z}_p$. It then runs $\mathsf{Sim}_2^{\mathsf{SS}}$ on input d and the shares of the left input wire to this gate held by the adversary to obtain $\{\hat{d}_i, \delta^{(d)}\}_{i\in\mathcal{H}}$ and $\{\hat{a}_i, \delta^{(a)}\}_{i\in\mathcal{I}}$. Similarly, it runs another instance of $\mathsf{Sim}_2^{\mathsf{SS}}$ on input e and the shares of the right input wire to this gate held by the adversary to obtain $\{\hat{e}_i, \delta^{(e)}\}_{i\in\mathcal{H}}$ and $\{\hat{b}_i, \delta^{(b)}\}_{i\in\mathcal{I}}$.
  - For each $i \in \mathcal{I}$, the simulator runs SS.Eval on d, e, $\{\hat{a}_i, \delta^{(a)}\}_{i\in\mathcal{I}}$ and $\{\hat{b}_i, \delta^{(b)}\}_{i\in\mathcal{I}}$ to obtain adversarial shares of $b \cdot d + a \cdot e + d \cdot e$.
  - It samples $z \leftarrow \mathbb{Z}_p$. It then runs $\mathsf{Sim}_2^{\mathsf{SS}}$ on input z and the above computes shares of $b\cdot d+a\cdot e+d\cdot e$ held by the adversary to obtain $\{\hat{z}_i, \delta^{(z)}\}_{i\in\mathcal{H}}$ and $\{\hat{c}_i, \delta^{(c)}\}_{i\in\mathcal{I}}$.
  - When the adversary invokes f_{Beaver}, the simulator returns the above computed shares $\{\hat{a}_i, \delta^{(a)}, \hat{b}_i, \delta^{(b)}, \hat{c}_i, \delta^{(c)}\}_{i\in\mathcal{I}}$.
  - It also sends the above computed shares $\{\hat{d}_i, \delta^{(d)}\}_{i\in\mathcal{H}}$ and $\{\hat{e}_i, \delta^{(e)}\}_{i\in\mathcal{H}}$ to the adversary on behalf of the honest parties during the computation of this gate.

  The remaining steps of this phase are simulated exactly as in the real protocol.

- *Simulating Output Reconstruction:* Run Sim^2 on input out and the shares of the output wire held by the adversary.

If at any step, in the above simulation, the shares sent by the adversary do not match those that the simulator had locally computed for the corresponding party, it includes that party in the set $\mathcal{D}$ to obtain the corresponding shares of the honest parties. It sends these shares to the adversary. This concludes the description of the simulator.

It is straightforward to see that the simulated transcript is indistinguishable from the real one, except that the shares output by f_{Beaver}, f_{Rand}, and some of the honest parties' shares are generated using $(\mathsf{Sim}_1^{\mathsf{SS}}, \mathsf{Sim}_2^{\mathsf{SS}})$. Indistinguishability of these messages follows directly from the reconstruction-friendliness property of LH-CSS.

Another difference is that the final output in the simulated transcript is obtained by querying the ideal functionality on the adversary's extracted inputs and the set of adversarial parties with malformed input shares is determined differently in the real and simulated setting. However, by the authenticated-shares property of LH-CSS, both procedures identify the same set of "discarded" parties. Consequently, the

same set of inputs is used in the output computation. As a result, we conclude that the protocol achieves guaranteed output delivery against malicious adversaries. □

Instantiating the above construction with an LH-CSS scheme for weighted threshold access structures from Corollary 1 yields the following:

Corollary 2. *Assuming the learning with errors assumption (see Definition 2) or the decisional composite residuosity assumption or the DCR assumption(see Definition 4), there exists a secure multiparty computation protocol in the* $(f_{\mathsf{Beaver}}, f_{\mathsf{Rand}})$*-hybrid model for weighted-threshold access structures that achieves guaranteed output delivery against malicious adversaries and has weight-independent communication and computation.*

Instantiating the Pre-Processing Phase. We now present a simple semi-honest secure MPC protocol for transforming additive shares of a secret to LH-CSS shares. We later discuss how this sub-routine can be used along with known techniques to instantiate f_{Beaver} and f_{Rand} with semi-honest secure MPC protocols with weight-independent complexity.

1. *Conversion from Additive to LH-CSS Shares.* For each $i \in [n]$, let $[a]_i$ denote the additive shares of some secret a held by party i. Party i proceeds as follows:

   - It computes $\{(\hat{a}_{i,1}, \ldots, \hat{a}_{i,n}), \delta_i^{(a)}\} \leftarrow \mathsf{SS.Share}([a]_i, 1)$, and sends $\hat{a}_{i,j}, \delta_i^{(a)}$ to party j.
   - Upon receiving corresponding shares from all other parties, party i locally computes LH-CSS shares of a as:

   $$(\hat{a}_i, \delta^{(a)}) \leftarrow \mathsf{SS.Eval}\Big(i, (1, 1, \ldots, 1), \big((\hat{a}_{1,i}, \delta_1^{(a)}), \ldots, (\hat{a}_{n,i}, \delta_n^{(a)})\big)\Big).$$

   From reconstruction friendliness property of LH-CSS, it is easy to see that this protocol remains secure against semi-honest adversaries. Moreover, if the underlying LH-CSS has weight-independent complexity, then so does this protocol.

2. *Instantiating* f_{Rand}. For instantiating f_{Rand}, we can start by using existing techniques [DPSZ12, GIP+14, BTH08] for obtaining additive shares of a random value and then use the above subroutine to transform them into LH-CSS shares.

3. *Instantiating* f_{Beaver}. Similarly, for instantiating f_{Beaver}, the parties can employ existing techniques [DPSZ12, BCGI18, BCG+19, BCG+20] to obtain additive shares (over $\mathbb{Z}_p$) of random Beaver triples and then transform them into LH-CSS shares using the above sub-routine.

6 Weighted Threshold Encryption

In this section, we present the definition and construction of *public-key encryption with distributed decryption w.r.t. arbitrary access structures.* For weighted threshold access structures, this yields the first black-box construction with weight-independent communication and computation.

6.1 Definition

We begin by recalling the definition, which we adapt from the notion of weighted threshold encryption in [GJM+23]. The key difference between our definition and theirs, aside from our generalization to arbitrary access structures, lies in the decryption model. While their scheme supports fully non-interactive partial decryption, our construction requires two rounds of interaction among the parties. Concretely, in the first round, each party can compute its message without knowing the eventual subset of participants in the decryption. The parties who contribute randomness in the first round of partial decryption need not coincide with those who participate in the second round to output the actual partial decryption.

We note that even for this interactive variant of partial decryption, no prior black-box scheme with weight-independent complexity for weighted threshold access structures was known.

Definition 10 (Public-Key Encryption with Distributed Decryption w.r.t. General Access Structures)**.** *A public-key encryption scheme with distributed decryption w.r.t. general access structure consists of a tuple of five PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PartialDec}_1, \mathsf{PartialDec}_2, \mathsf{Reconstruct})$.

- $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n) \leftarrow \mathsf{KeyGen}(1^\lambda, \Gamma)$*: On input the security parameter* 1^λ *and a monotone access structure* $\Gamma \subseteq 2^{[n]}$*, this randomized algorithm outputs a public key* pk *and secret-key shares* $\{\mathsf{sk}_i\}_{i=1}^n$*, where* sk_i *is given to the* i*-th party.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: On input the public key* pk *and a message* m*, this randomized algorithm outputs a ciphertext* c*.*
- $\{\alpha_{i,j}\}_{j\in[n]} \leftarrow \mathsf{PartialDec}_1(i, \mathsf{sk}_i, c)$*: On input a party index* $i \in [n]$*, a secret-key share* sk_i*, and a ciphertext* c*, this randomized algorithm outputs intermediate values* $\{\alpha_{i,j}\}_{j\in[n]}$*, where each* $\alpha_{i,j}$ *is given to the* i*-th party.*
- $\mu_i \leftarrow \mathsf{PartialDec}_2(i, \mathsf{sk}_i, c, \{\alpha_{j,i}\}_{j\in\mathcal{H}})$*: On input a party index* $i \in [n]$*, a secret-key share* sk_i*, a ciphertext* c*, and intermediate values* $\{\alpha_{j,i}\}_{j\in\mathcal{H}}$*, where* $\mathcal{H} \subseteq [n]$*, this deterministic algorithm outputs a partial decryption* μ_i*.*
- $m \leftarrow \mathsf{Reconstruct}(\{\mu_i\}_{i\in\mathcal{T}}, c)$*: A deterministic algorithm that takes as input partial decryptions* $\{\mu_i\}_{i\in\mathcal{T}}$ *from a subset* $\mathcal{T}$ *of parties and a ciphertext* c*, and outputs the message* m *(or* $\perp$ *if* $\mathcal{T} \notin \Gamma$*).*

The scheme should satisfy the following properties:

- ***Statistical Correctness.*** *For any* $\mathcal{H} \subseteq [n]$*, and any monotone access structure* Γ*, any authorized subset* $\mathcal{T} \in \Gamma$*, and any message* m*, it holds that*

$$\Pr\left[m^* = m \;\middle|\; \begin{array}{c} (\mathsf{pk}, \{\mathsf{sk}_i\}_{i=1}^n) \leftarrow \mathsf{KeyGen}(1^\lambda, \Gamma) \\ c \leftarrow \mathsf{Enc}(\mathsf{pk}, m) \\ \forall i \in \mathcal{H} : \{\alpha_{i,j}\}_{j\in[n]} \leftarrow \mathsf{PartialDec}_1(i, \mathsf{sk}_i, c) \\ \forall i \in \mathcal{T} : \mu_i \leftarrow \mathsf{PartialDec}_2(i, \mathsf{sk}_i, c, \{\alpha_{j,i}\}_{j\in\mathcal{H}}) \\ m^* \leftarrow \mathsf{Reconstruct}(\{\mu_i\}_{i\in\mathcal{T}}, c) \end{array}\right] \geqslant 1 - \mathsf{negl}(\lambda).$$

- ***CPA Security.*** *For any stateful (semi-honest) PPT adversary* $\mathcal{A}$ *and any monotone access structure* Γ*, it holds that*

$$\Pr\left[\mathsf{Game}^{\mathrm{IND-CPA}}(1^\lambda, \Gamma) \to 1\right] \leqslant \mathsf{negl}(\lambda).$$

$\mathsf{Game}^{\mathsf{IND-CPA}}(1^\lambda, \Gamma)$:

1. *On receiving the security parameter* λ *and access structure* Γ*, the adversary* $\mathcal{A}$ *outputs a set of corrupt parties* $\mathcal{I} \notin \Gamma$ *to the* IND − CPA *challenger* Chal.
2. Chal *runs* KeyGen *with the sceurity parameter* 1^λ *and the access structure* Γ*. Outputs the public key* pk *and the corrupt secret keys* $\{\mathsf{sk}_i\}_{i\in\mathcal{I}}$ *to* $\mathcal{A}$*.*
3. ***Pre-challenge Queries.*** $\mathcal{A}$ *can (parallelly) request any new decryption sessions on (honestly generated) ciphertexts:*
   - *Sends* $(c, \mathcal{H})$ *to* Chal *for some ciphertext* c *and set* $\mathcal{H} \notin \Gamma$*.*
   - Chal *computes and sends* $\{\alpha_{i,j}\}_{i\in\mathcal{H}\setminus\mathcal{I}, j\in\mathcal{H}\cap\mathcal{I}} \leftarrow \mathsf{PartialDec}_1(i, \mathsf{sk}_i, c)$ *to* $\mathcal{A}$*.*
   - $\mathcal{A}$ *sends the first partial decryption messages to challenger* $\{\alpha_{i,j}\}_{i\in\mathcal{H}\cap\mathcal{I},\, j\in\mathcal{H}\setminus\mathcal{I}}$ *to* Chal*.*
   - *For all* $i \in \mathcal{H} \setminus \mathcal{I}$*,* Chal *computes* $\mu_i \leftarrow \mathsf{PartialDec}_2\big(i, \mathsf{sk}_i, c, \{\alpha_{j,i}\}_{j\in\mathcal{H}}\big)$ *and sends to* $\mathcal{A}$*.*
4. ***Challenge Phase.*** $\mathcal{A}$ *submits two challenge messages* m_0 *and* m_1 *to* Chal*.*
   - Chal *samples a bit* $b \leftarrow \{0,1\}$ *and send* $c_b \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$ *to* $\mathcal{A}$*.*
5. ***Post-challenge Queries.*** $\mathcal{A}$ *may submit partial decryption queries on valid ciphertexts* c *after the challenge phase as long as* $c \neq c_b$ *in a similar manner as the pre-challenge queries.*
6. *Finally, the adversary* $\mathcal{A}$ *outputs a guess* b^**.*
7. Chal *outputs* 1 *if and only if* $b = b^*$*.*

Remark 2. *In other words, we require that even if the adversary augments its corrupted set* $\mathcal{I}$ *with additional parties* $\mathcal{H}$*, the combined set remains unauthorized (so reconstruction is impossible).*

As observed in [GJM+23, Remark 3], plain CPA security in the setting of public-key encryption with distributed decryption (where the adversary has no access to partial decryption oracles) is essentially trivial. Indeed, any CPA-secure public-key encryption scheme can be combined with an arbitrary secret-sharing scheme by generating a key pair, secret-sharing the decryption key, and defining partial decryption to simply output the corresponding share. This naïve construction already achieves plain CPA security.

To avoid such trivialities, we adopt a stronger notion where the adversary is given access to partial decryptions on honestly generated ciphertexts. This stronger CPA-security definition rules out the above construction.

6.2 Construction

We now show how our linearly homomorphic computational secret sharing (LH-CSS) can be combined with the LPR variant [LPR10] of LWE-based Public-Key Encryption to obtain a public-key encryption with distributed decryption w.r.t. arbitrary access structures that can be represented efficiently as a monotone Boolean circuit.

We start by recalling the baseline LWE-based public-key encryption.

LWE-based Public-Key Encryption [LPR10]. Let N be the dimension, p an integer modulus, let χ be a uniform distribution over $\mathbb{Z}$, such that $\mathrm{Supp}(\chi) \subseteq \left(-\sqrt{\frac{p}{4(2N+1)}}, \sqrt{\frac{p}{4(2N+1)}}\right)$.

- **Key Generation** $\mathsf{KeyGen}(1^\lambda)$**.** Sample $A \leftarrow \mathbb{Z}_p^{N\times N}$ uniformly at random, and $e, s \leftarrow \chi^N$. Set $b \leftarrow As + e \bmod p$. Output public key $\mathsf{pk} = (A, b)$ and secret key $\mathsf{sk} = s$.
- **Encryption** $\mathsf{Enc}(\mathsf{pk}, m)$**.** On input a bit $m \in \{0,1\}$, sample $x, e'' \leftarrow \chi^N$, and $e' \leftarrow \chi$. Compute

$$u \leftarrow Ax + e'' \bmod p, \qquad v \leftarrow b^\top x + e' + \left\lfloor \tfrac{p}{2} \right\rceil \cdot m \bmod p.$$

Output ciphertext $c = (u, v) \in \mathbb{Z}_p^N \times \mathbb{Z}_p$.

- **Decryption** $\mathsf{Dec}(\mathsf{sk}, u, v)$**.** Compute $w \leftarrow v - \langle u, s \rangle \bmod p$ and output

$$\hat{m} \leftarrow \begin{cases} 0 & \text{if } w \in \left(-\frac{p}{4}, \frac{p}{4}\right) \pmod p, \\ 1 & \text{if } w \in \left(\frac{p}{4}, \frac{3p}{4}\right) \pmod p. \end{cases}$$

Public-Key Encryption with Distributed Decryption w.r.t. General Access Structures. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of n parties, and let $\Gamma \subseteq 2^{\mathcal{P}}$ be a monotone access structure (Definition 7) on $\mathcal{P}$. Let χ and χ_{smg} be uniform distributions over $\mathbb{Z}$, where $\mathsf{Supp}(\chi) \subseteq (-B, B)$, and $\mathsf{Supp}(\chi_{\mathsf{smg}}) \subseteq (-B_{\mathsf{smg}}, B_{\mathsf{smg}})$, and the parameters satisfy

$$B^2(2N+1) + B_{\mathsf{smg}} \leqslant \tfrac{p}{4} \qquad \text{and} \qquad B/B_{\mathsf{smg}} = \mathsf{negl}(\lambda).$$

Let $(\mathsf{SS.Setup}, \mathsf{SS.Share}, \mathsf{SS.Recon}, \mathsf{SS.Eval}, \mathsf{SS.Verify})$ be an LH-CSS (Definition 8) w.r.t. Γ. We will use a computational secret sharing scheme with linear homomorphism as the mechanism for distributing the decryption key among parties in accordance with Γ.

- **Key Generation** $\mathsf{KeyGen}(1^\lambda, \Gamma)$**.** We use our computational secret sharing scheme with linear homomorphism for distributing the decryption key among parties in accordance with Γ.

  1. Sample $A \leftarrow \mathbb{Z}_p^{N \times N}$ uniformly at random, and $s, e \leftarrow \chi^N$. Set $b \leftarrow As + e \bmod p$ and define $\mathsf{pk} = (A, b)$, $\mathsf{sk} = s$.
  2. Run setup algorithm of LH-CSS: $\mathsf{pp} \leftarrow \mathsf{SS.Setup}(1^\lambda, 1^n, \Gamma, p)$.
  3. Let $s = (s_1, \ldots, s_N)$. For each $j \in [N]$, share the j-th secret key coordinate s_j among parties according to Γ: $(\{\hat{s}_{1,j}, \ldots, \hat{s}_{1,j}\}, \delta_j) \leftarrow \mathsf{SS.Share}(s_j, 0)$.
  4. Set $\hat{s}_i = \{\hat{s}_{i,j}\}_{j \in [N]}$, $\delta = \{\delta_j\}_{j \in [N]}$ and give $\mathsf{sk}_i = (\hat{s}_i, \delta)$ to party P_i.

- **Encryption** $\mathsf{Enc}(\mathsf{pk}, m)$**.** On input a bit $m \in \{0, 1\}$, sample $e' \leftarrow \chi$, $x, e'' \leftarrow \chi^N$. Compute

$$u \leftarrow Ax + e'' \bmod p, \qquad v \leftarrow b^\top x + e' + \left\lfloor \tfrac{p}{2} \right\rceil \cdot m \bmod p.$$

  Output ciphertext $c = (u, v) \in \mathbb{Z}_p^N \times \mathbb{Z}_p$.

- **Partial Decryption**

  1. $\mathsf{PartialDec}_1(i, \mathsf{sk}_i, c)$. Sample $e_i \leftarrow \chi_{\mathsf{smg}}$ and secret share it using LH-CSS: $(\{\hat{e}_{i,1}, \ldots, \hat{e}_{i,n}\}, \varepsilon_i) \leftarrow \mathsf{SS.Share}(e_i, 1)$. For each $j \in \mathcal{H}$, give $\alpha_{i,j} = (\hat{e}_{i,j}, \varepsilon_i)$ to party P_j.
  2. $\mathsf{PartialDec}_2(i, \mathsf{sk}_i, c, \{\alpha_{j,i}\}_{j \in \mathcal{H}})$. Party P_i proceeds as follows:
     (a) Parse $c = (u, v)$, where $u = (u_1, \ldots, u_N)$.
     (b) Parse $\mathsf{sk}_i = (\hat{s}_i, \delta)$, and $\hat{s}_i = \{\hat{s}_{i,j}\}_{j \in [N]}$ and $\delta = \{\delta_j\}_{j \in [N]}$.
     (c) For each $j \in \mathcal{H}$, parse $\alpha_{j,i} = (\hat{e}_{j,i}, \varepsilon_j)$.
     (d) Compute and output

$$(\hat{d}_i, \delta') \leftarrow \mathsf{SS.Eval}\Big(i,\ (u_1, \ldots, u_N, 1, \ldots, 1), \big((\hat{s}_{i,1}, \delta_1), \ldots, (\hat{s}_{i,N}, \delta_N), \{(\hat{e}_{j,i}, \varepsilon_j)\}_{j \in \mathcal{H}}\big)\Big).$$

- **Reconstruction** $\mathsf{Reconstruct}(\{\mu_i\}_{i\in\mathcal{T}}, c)$**.** Given a set $\mathcal{T} \subseteq \mathcal{P}$ with $\mathcal{T} \in \Gamma$, for each $i \in \mathcal{T}$, parse $\mu_i = (\hat{d}_i, \delta')$.

  1. Compute $d \leftarrow \mathsf{SS.Recon}(\mathcal{T}, \{\mu_i\}_{i\in\mathcal{T}}, \delta') = \langle u, s\rangle + \sum_{i\in\mathcal{H}} e_i$.
  2. Then compute $w \leftarrow v - d \bmod p$ and recover the bit

$$\hat{m} = \begin{cases} 0 & \text{if } w \in \left(-\frac{p}{4}, \frac{p}{4}\right) \pmod p, \\ 1 & \text{if } w \in \left(\frac{p}{4}, \frac{3p}{4}\right) \pmod p. \end{cases}$$

Theorem 6. *Let* Γ *be an access structure and* $(\mathsf{SS.Setup}, \mathsf{SS.Share}, \mathsf{SS.Recon}, \mathsf{SS.Eval}, \mathsf{SS.Verify})$ *be an LH-CSS w.r.t.* Γ *(see Definition 8). Assuming the hardness of learning with errors (LWE), the above construction satisfies the definition of public-key encryption with distributed decryption with respect to* Γ *(Definition 10).*

Proof. We now prove correctness and security of this construction.

Statistical Correctness. From correctness of LH-CSS, it is easy to see that the Reconstruct algorithm computes

$$w = v - \langle u, s\rangle - \sum_{i\in\mathcal{H}} e_i \bmod p$$

$$\left(Asx + \langle e, x\rangle + e' + \left\lfloor \tfrac{p}{2} \right\rceil \cdot m - Asx - \langle e'', s\rangle - \sum_{i\in\mathcal{H}} e_i\right) \bmod p$$

$$\left(\langle e, x\rangle + e' - \langle e'', s\rangle - \sum_{i\in\mathcal{H}} e_i + \left\lfloor \tfrac{p}{2} \right\rceil \cdot m\right) \bmod p$$

As long as $B^2(2N+1) + B_{\mathsf{smg}} \leqslant \frac{p}{4}$, the absolute value $\left|\langle e, x\rangle + e' - \langle e'', s\rangle - \sum_{i\in\mathcal{H}} e_i\right|$ is at most $p/4$. In other words, $w \leqslant \frac{p}{4} + \left\lfloor \frac{p}{2} \right\rceil \cdot m \bmod p$ and

$$\hat{m} = \begin{cases} 0 & \text{if } w \in \left(-\frac{p}{4}, \frac{p}{4}\right) \pmod p, \\ 1 & \text{if } w \in \left(\frac{p}{4}, \frac{3p}{4}\right) \pmod p. \end{cases} = m.$$

CPA Security. We now demonstrate via the following sequence of hybrids that an encryption of $m_0 = 0$ remains indistinguishable from that of $m_1 = 1$ for a semi-honest PPT adversary.

0. **Hybrid 0:** This is the real world when message is $m_0 = 0$.

1. **Hybrid 1 (Simulating the partial decryption queries):** This hybrid is identical to the one above, except that the challenger simulates all it query answers. In particular, it sends $\mathsf{PartialDec}_1$ honestly, but for $\mathsf{PartialDec}_2$, it invokes the LH-CSS simulator (for the reconstruction-friendliness) to simulate the honest parties' shares.

   Indistinguishability between hybrid 0 and 1 follows from reconstruction-friendliness of the underlying LH-CSS directly. In particular, since the $\mathcal{H}$ is an unauthorized shares, the simulator does not need the combined secrets as input to simulate the shares (refer to Remark 1).

   Note that all the outputs of the oracle are now independent of the secret key.

2. **Hybrid 2 (Decouple the key pair):** This hybrid is identical to the previous one, except that the public key is now sampled uniformly at random, i.e., $\widetilde{\mathsf{pk}} = (A, y) \leftarrow \mathbb{Z}_p^{N\times N} \times \mathbb{Z}_p^N$.

   Indistinguishability between hybrids 1 and 2 follows routinely from the hardness of LWE, i.e., the public key switches from an LWE sample to a truly random string. This is possible because no other part of the hybrid involves the secret key.

3. **Hybrid 3 (Simulating the ciphertext):** This hybrid is identical to the previous one, except that the encryption is computed as follows: $\mathsf{Enc}(\widetilde{\mathsf{pk}}, m_0) = (\tilde{u}, \tilde{b} + \lfloor \frac{p}{2} \rceil \cdot m_0 \bmod p)$, where $\tilde{u} \leftarrow \mathbb{Z}_p^N$ and $\tilde{b} \leftarrow \mathbb{Z}_p$ are sampled uniformly at random.

   Again, indistinguishability between hybrids 2 and 3 follows from routinely from the hardness of LWE, i.e., since the public key are truly random now, the original ciphertext was an LWE sample, which is switch to truly random due to LWE.

4. **Hybrid 4 (Switch message):** Similar to hybrid 3, except that we now compute encryption of $m_1 = 1$.

   Hybrid 3 and 4 are perfectly indistinguishable because of one-time pad.

5. **Hybrid 5 (Switch back to real ciphertext):** Similar to hybrid 2, but for $m_1 = 1$.

   Hybrid 4 and 5 are indistinguishable for the same reason why hybrid 2 and 3 are indistinguishable.

6. **Hybrid 6 (Switch back to real key pair):** Similar to hybrid 1, but for $m_1 = 1$.

   Hybrid 5 and 6 are indistinguishable for the same reason why hybrid 1 and 2 are indistinguishable.

7. **Hybrid 7 (Switch back to real game):** Similar to hybrid 0, but for $m_1 = 1$. Note that this is the real world when message is $m_1 = 1$.

   Hybrid 6 and 7 are indistinguishable for the same reason why hybrid 0 and 1 are indistinguishable.

Note that the indistinguishability between Hybrid 0 and 7 proves the CPA security. □

Instantiating Theorem 6 with Corollary 1 we get the following:

Corollary 3. *Assuming the hardness of learning with errors (LWE) (Definition 2), there exists a construction of public-key encryption with distributed decryption for weighted threshold access structures (Definition 10)in the trusted setup model with weight-independent complexity.*

7 Weighted Threshold Schnorr Signatures

We recall the definition of a Schnorr signature scheme with a distributed signing process [CKM23, GGW24]. We require the final signature to be exactly a Schnorr signature.

7.1 Definition

Again, the key difference between the definition below and other definitions in the literature is that we consider a general access structure and require the unforgeability to hold as long as the adversary only corrupts an unauthorized set of parties.

Definition 11 (Schnorr Signature with Distributed Signing w.r.t. General Access Structure)**.** *A Schnorr signature with distributed signing process with respect to a general access structure is defined by the following tuple of algorithms and syntax.*

- $\left(\mathsf{sk}, \mathsf{pk}, \left\{\mathsf{sk}_i\right\}_{i\in[n]}\right) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, \Gamma)$*: on input the security parameter* λ*, the number of parties* n *and a general access structure* Γ*,* KeyGen *returns to each party their respective key pairs* $(\mathsf{sk}_i, \mathsf{pk}_i)$ *and the joint public key* pk*.*
- $\left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i\in B} \leftarrow (\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3)$*: This is a three-round signing protocol that can be run by any authorized set* B *of parties. In particular, given such a subset* $B \subseteq [n]$ *and a message* msg *to sign, each party* $i \in B$ *does:*
  - *Round 1:* $(\rho_1^i, \mathsf{st}_i) \leftarrow \mathsf{Sign}_1(B, \mathsf{msg}, \mathsf{sk}_i)$*: this outputs the secret state* st_i *and first-round message* ρ_1^i*.*
  - *Round 2:* $(\rho_2^i, \{\alpha_{i\to j}\}_{j\in B}) \leftarrow \mathsf{Sign}_2\left(B, \mathsf{msg}, \mathsf{sk}_i, \mathsf{st}_i, \left\{\rho_1^i\right\}_{i\in B}\right)$*: given the first-round messages from all parties, this outputs the second-round (broadcast) message* ρ_2^i *and its private messages* $\alpha_{i\to j}$ *to party* j *for every* j*.*
  - *Round 3:* $\rho_3^i \leftarrow \mathsf{Sign}_3\left(B, \mathsf{msg}, \mathsf{sk}_i, \mathsf{st}_i, \left\{\rho_1^i, \rho_2^i\right\}_{i\in B}, \{\alpha_{j\to i}\}_{j\in B}\right)$*: given the transcript from the first two rounds, this outputs the third-round message.*
- $\sigma \leftarrow \mathsf{Aggregate}\left(\mathsf{msg}, \left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i\in B}\right)$*: on input the message* msg *and the (public) transcript of a signing protocol, this algorithm outputs an aggregated signature* σ*.*
- $b \leftarrow \mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma)$*: on input the joint public key* pk*, the message* msg*, and the aggregate signature* σ*, this verification algorithm outputs a bit* b *indicating* accept *or* reject*.*

These algorithms must satisfy the following properties.

- ***Correctness.*** *An honestly generated signature should always verify. That is, for any security parameter* λ*, any general access structure* Γ*, any message* msg*, and any authorized set of signers* B*, it holds that*

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma) = 1 \;\middle|\; \begin{array}{c} \forall i \in [n],\ (\mathsf{sk}, \mathsf{pk}, \left\{(\mathsf{sk}_i, \mathsf{pk}_i)\right\} \leftarrow \mathsf{KeyGen}(1^\lambda, 1^n, \Gamma) \\ \left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i\in B} \leftarrow (\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3) \\ \sigma \leftarrow \mathsf{Aggregate}\left(\mathsf{msg}, \left\{\rho_1^i, \rho_2^i, \rho_3^i\right\}_{i\in B}\right) \end{array}\right] = 1.$$

- ***Security.*** *Any (semi-honest) PPT adversary, who may request an arbitrary polynomial many concurrent signing sessions, should not be able to forge a valid signature with non-negligible probability. In particular, the probability that the adversary wins the following game is* $\mathsf{negl}(\lambda)$*.*

> *Unforgeability Game*
>
> 1. *On input the security parameter* λ*, adversary picks* n*, a general access structure* Γ*, and a (unauthorized) subset of parties to corrupt* $A \subseteq [n]$*.*
> 2. *A trusted setup is sampled* $\mathsf{KeyGen}(1^\lambda, 1^n, \Gamma)$ *and the secret key of the corrupted parties is given to the adversary. The public key is published.*
> 3. *The adversary can repeatedly request new signing sessions. For each session, it specifies a*

message msg *and an authorized subset* $B \subseteq [n]$*. The adversary is rushing in that, for each round, the honest parties in* B *send the messages first. These sessions can be concurrent. Moreover, the adversary does not need to complete the protocol. Let* $Q = \emptyset$*. For each session requested, let* $Q = Q \cup \{\mathsf{msg}\}$*.*

4. *In the end, the adversary outputs a message* msg^* *and a signature* σ^**. The adversary wins the forgery game if (1)* $\mathsf{msg}^* \notin Q$ *and (2)* $\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}^*, \sigma^*) = 1$*, i.e., the adversary produces a valid signature for a message he never requested in any signing session.*

- ***Exact-Schnorr.*** *The final aggregated signature should look exactly like a Schnorr. In particular, there is some cryptographic group* $\mathbb{G}$ *with order* p *and generator* g *such that* $(\mathsf{pk}, \mathsf{sk}) \in \mathbb{G} \times \mathbb{F}_p$ *and* σ *can be parsed as* $(R, \sigma) \in \mathbb{G} \times \mathbb{F}_p$*. The verification checks if* $g^\sigma = R \cdot \mathsf{pk}^c$*, where the challenge* $c = H(\mathsf{pk}, R, \mathsf{msg})$ *is computed by some hash function* H*.*

7.2 Construction

Like any Schnorr signature, our construction relies on the computational hardness of a cryptographic group. We recall the discrete log assumption as follows.

Definition 12. *The discrete log (DL) assumption states that, for any adversary* $\mathcal{A}$*, we have*

$$\Pr\left[\mathcal{A}(1^\lambda, \mathbb{G}, g, g^x) = x \mid (\mathbb{G}, g) \leftarrow \mathsf{Group.Setup}(1^\lambda)\right] = \mathsf{negl}(\lambda).$$

Theorem 7. *Assuming that* (Setup, Share, Recon, Eval, Verify) *is a linearly homomorphic computational secret sharing scheme according to Definition 8,* H *is a random oracle, and the discrete log assumption (Definition 12) holds in* $\mathbb{G}$*, the construction in Figure 4 is a three-round Schnorr signature scheme with distributed signing for general access structure* Γ *according to Definition 11.*

Proof of Theorem 7. The correctness and the complexity of the scheme can be verified trivially. The unforgeability can be proven by standard techniques for proving threshold Schnorr signatures. We provide a proof below.

Suppose there is an adversary $\mathcal{A}'$ that breaks the unforgeability game of the original scheme. We construct an adversary that breaks the DL assumption. Let $\mathcal{A}$ be an adversary for the DL assumption. That is, it receives a challenge $\mathsf{pk}' = g^x$ from the external challenger. It proceeds to simulate a forgery game with $\mathcal{A}'$.

It computes $\left\{(\hat{\mathsf{sk}}_1, \hat{\mathsf{sk}}_2, \ldots, \hat{\mathsf{sk}}_n), \delta\right\} \leftarrow \mathsf{Share}(\mathsf{pp}, \mathsf{sk} = 0, 0)$ and sends the respective secret shares to the corrupted party. The public key is set to be $\mathsf{pk} = (\mathsf{pk}, \delta)$.

Now, for each signing session with signers from B, $\mathcal{A}$ simulates as follows. It sends an arbitrary random string ρ_1^i as the commitment of the honest parties' commitment. After $\mathcal{A}'$ sends its commitment, $\mathcal{A}$ extracts the committed g^{r_i} from the malicious parties. It picks the g^{r_i} for the honest party according to the protocol specification (meaning that $\mathcal{A}$ knows r_i) except for one special honest party. For the last special honest party, it first picks a random challenge c and sets g^{r_i} to be some $g^{y - c \cdot x} = g^y / (g^x)^c$ for a randomly sampled y. Note that the simulator $\mathcal{A}$ does not know the dlog of g^{r_i} for the special honest party (since it is not honestly sampled). It further programs the random oracle to be consistent with the commitment $\rho_2^i = H(\rho_1^i)$ and the Fiat-Shamir challenge $c = H(\mathsf{msg}, \prod_i g^{r_i})$. Finally, to send honest parties' partial signatures in the final round. For all honest parties except for the special honest party, $\mathcal{A}$ proceeds according to the protocol description since it knows the dlog of the nounce. For the special honest party, it

$\underline{\mathsf{KeyGen}(1^\lambda, 1^n, \Gamma) \to \left(\mathsf{pk}, \mathsf{sk}, \left\{\mathsf{sk}_i\right\}_{i\in[n]}\right)}$

- Sample $(\mathbb{G}, g) \leftarrow \mathsf{Group.Setup}(1^\lambda)$, $\mathsf{sk} \leftarrow \mathbb{Z}_p$ and set $\mathsf{pk}' = g^{\mathsf{sk}}$.
- Sample $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^n, \Gamma, p)$.
- Compute $\left\{(\hat{\mathsf{sk}}_1, \hat{\mathsf{sk}}_2, \ldots, \hat{\mathsf{sk}}_n), \delta\right\} \leftarrow \mathsf{Share}(\mathsf{pp}, \mathsf{sk}, 0)$.
- Send $\hat{\mathsf{sk}}_i$ to party i for all $i \in [n]$.
- Set $\mathsf{pk} = (\mathsf{pk}', \delta)$ as the public key.

$\underline{\mathsf{Sign}_1(B, \mathsf{msg}, \hat{\mathsf{sk}}_i) \to \rho_1^i}$

- Sample $r_i \leftarrow \mathbb{Z}_p$ and compute the nonce as $R_i := g^{r_i}$.
- Set $\mathsf{com}_i := H(R_i)$. Broadcast $\rho_1^i := \mathsf{com}_i$, a commitment to the nonce.

$\underline{\mathsf{Sign}_2(B, \mathsf{msg}, \hat{\mathsf{sk}}_i, \rho_1^i) \to \left(\rho_2^i, \left\{\alpha_{i\to j}\right\}_{j\in[B]}\right)}$

- Compute $\left\{(\hat{r}_{i,1}, \hat{r}_{2,i}, \ldots, \hat{r}_{n,i}), \delta_i\right\} \leftarrow \mathsf{Share}(\mathsf{pp}, r_i, 1)$
- For all $j \in [n]$, send $\alpha_{i\to j} := \hat{r}_{i,j}$ and send it to party j.
- Set and broadcast $\rho_2^i := \{R_i, \delta_i\}$.

$\underline{\mathsf{Sign}_3(B, \mathsf{msg}, \hat{\mathsf{sk}}_i, \rho_1^i, \rho_2^i, \left\{\alpha_{j\to i}\right\}_{j\in[B]}) \to \rho_3^i}$

- Compute the aggregate nonce $R := \prod_{i\in B} R_i$.
- Compute the challenge as $c := H(\mathsf{pk}, R, \mathsf{msg})$.
- Compute the partial signature $(\sigma_i, \Delta) := \mathsf{Eval}\left(\{1, 1, \ldots, 1, c\}, \left\{\hat{r}_{j,i}, \delta_j\right\}_{j\in B} \cup \{\hat{\mathsf{sk}}_i, \delta\}\right)$.
- Output $\rho_3^i := (\sigma_i, \Delta)$.

$\underline{\mathsf{Aggregate}\left(\mathsf{msg}, \left\{\rho_1^j, \rho_2^j, \rho_3^j\right\}_{j\in B}\right) \to \sigma}$

- Compute the aggregate signature as $\sigma \leftarrow \mathsf{Recon}\left(B, \left\{\sigma_j\right\}_{j\in B}, \Delta\right)$.
- Output (R, σ).

$\underline{\mathsf{Verify}(\mathsf{pk}, \mathsf{msg}, \sigma') \to 0/1}$

- Parse σ' as (R, σ).
- Compute $c = H(\mathsf{pk}, R, \mathsf{msg})$.
- Verify if $g^\sigma = R \cdot \mathsf{pk}^c$.

Figure 4: Three-round weighted threshold Schnorr signature scheme from our LH-CSS scheme

invokes the simulator for the computational secret sharing with linear homomorphism on $\mathsf{Sim}(y = c\cdot x + r_i)$ to simulate the public part of the secret sharing of r_i and the sum of the shares. It computes the secret shares of r_i by computing it shifted by c times the secret shares of x.

This simulates the adversary's view in a computationally indistinguishable manner due to decryption-friendliness. First, for all but the special honest party, we simulate the view exactly according to the protocol description, so there is no difference between the real world and the simulated world. For the special honest party, first note that its nounce g^{r_i}, although sampled in a different way, is still uniformly random and thus, correctly distributed. Second, the final reconstructed dlog (for the public key combined

with all the nounces) is the correct dlog due to the way we program the special honest party's nounce. So the only remaining part that may be different in the two worlds is the secret share of y. By the decryption friendliness, we know that these two views are computationally indistinguishable.

Now, with a non-negligible probability, $\mathcal{A}'$ will generate a valid forgery (R, σ) under msg^*. Now, we rewind the adversary and reprogram the query on $H(\mathsf{pk}, R, \mathsf{msg}^*)$ from c to a different c'. By the general forking lemma [PS00, BN06, BDL19], we have that the adversary will output another forgery (R, σ') on the same message msg^* with a non-negligible probability. Now, this gives us two satisfying equations

$$(g^x)^c \cdot R = g^\sigma$$
$$(g^x)^{c'} \cdot R = g^{\sigma'}$$

Conditioned on this happening, we solve the discrete log by computing $x = (\sigma' - \sigma)/(c' - c)$, which completes the entire proof. □

Corollary 4. *When one instantiates the linearly homomorphic computational secret sharing scheme using our construction (based on either LWE or DCR), Figure 4 is a three-round Schnorr signature scheme with distributed signing for the weighted threshold access structure, where the complexity of the scheme scales (with a fixed polynomial) with the number of parties. The construction is based on the DL assumption and the LWE/DCR assumption in the random oracle model.*

Acknowledgement. Mingyuan Wang is supported in part by a research grant from Open Philanthropy.

References

[AIdK23] Muhammad Nadeem Ali, Muhammad Imran, Muhammad Salah ud din, and Byung-Seo Kim. Low rate ddos detection using weighted federated learning in sdn control plane in iot network. *Applied Sciences*, 13(3), 2023. URL: https://www.mdpi.com/2076-3417/13/3/1431, doi:10.3390/app13031431. 3

[AIK11] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In Rafail Ostrovsky, editor, *52nd Annual Symposium on Foundations of Computer Science*, pages 120–129, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press. doi:10.1109/FOCS.2011.40. 9, 12

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 483–501, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-29011-4_29. 14

[BBC+19] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear PCPs. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 67–97, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-26954-8_3. 5

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-26954-8_16. 10, 35

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from ring-LPN. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 387–416, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-56880-1_14. 10, 35

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 896–912, Toronto, ON, Canada, October 15–19, 2018. ACM Press. doi:10.1145/3243734.3243868. 10, 35

[BDL19] Mihir Bellare, Wei Dai, and Lucy Li. The local forking lemma and its application to deterministic encryption. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 607–636, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland. doi:10.1007/978-3-030-34618-8_21. 44

[Bea91] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 1991. doi:10.1007/3-540-46766-1_34. 10, 31

[Bei25] Amos Beimel. Secret-sharing schemes for general access structures: An introduction. Cryptology ePrint Archive, Paper 2025/518, 2025. URL: https://eprint.iacr.org/2025/518. 6

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 565–596, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96884-1_19. 11

[BGI+12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012. doi:10.1145/2160158.2160159. 4

[BGIN19] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Practical fully secure three-party computation via sublinear distributed zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 869–886, London, UK, November 11–15, 2019. ACM Press. doi:10.1145/3319535.3363227. 5

[BGIN20] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part III*, volume 12493 of *Lecture Notes in Computer Science*, pages 244–276, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-64840-4_9. 5

[BGIN21] Elette Boyle, Niv Gilboa, Yuval Ishai, and Ariel Nof. Sublinear GMW-style compiler for MPC with preprocessing. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 457–485, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-84245-1_16. 5

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 1–10, Chicago, IL, USA, May 2–4, 1988. ACM Press. doi:10.1145/62212.62213. 3, 10

[BHS23] Fabrice Benhamouda, Shai Halevi, and Lev Stambler. Weighted secret sharing from wiretap channels. In Kai-Min Chung, editor, *ITC 2023: 4th Conference on Information-Theoretic Cryptography*, volume 267 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:19, Aarhus, Denmark, June 6–8, 2023. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ITC.2023.8. 6

[BL90] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In Shafi Goldwasser, editor, *Advances in Cryptology – CRYPTO'88*, volume 403 of *Lecture Notes in Computer Science*, pages 27–35, Santa Barbara, CA, USA, August 21–25, 1990. Springer, New York, USA. doi:10.1007/0-387-34799-2_3. 6, 7, 12

[BLLL23] Marshall Ball, Hanjun Li, Huijia Lin, and Tianren Liu. New ways to garble arithmetic circuits. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 3–34, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-30617-4_1. 9, 12, 14, 15

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 575–584, Palo Alto, CA, USA, June 1–4, 2013. ACM Press. doi:10.1145/2488608.2488680. 18

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. doi:10.1145/1180405.1180453. 44

[BTH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In Ran Canetti, editor, *TCC 2008: 5th Theory of Cryptography Conference*, volume 4948 of *Lecture Notes in Computer Science*, pages 213–230, San Francisco,

CA, USA, March 19–21, 2008. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-78524-8_13. 35

[BVF+25] Samuel Breckenridge, Dani Vilardell, Andrés Fábrega, Amy Zhao, Patrick McCorry, Rafael Solari, and Ari Juels. B-privacy: Defining and enforcing privacy in weighted voting, 2025. URL: https://arxiv.org/abs/2509.17871, arXiv:2509.17871. 3

[BW06] Amos Beimel and Enav Weinreb. Monotone circuits for monotone weighted threshold functions. *Information Processing Letters*, 97(1):12–18, 2006. 3, 4, 6, 7, 8, 12, 23

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th Annual ACM Symposium on Theory of Computing*, pages 11–19, Chicago, IL, USA, May 2–4, 1988. ACM Press. doi:10.1145/62212.62214. 3, 10

[CKM23] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 678–709, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38557-5_22. 3, 12, 40

[DCX+23] Sourav Das, Philippe Camacho, Zhuolun Xiang, Javier Nieto, Benedikt Bünz, and Ling Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023: 30th Conference on Computer and Communications Security*, pages 356–370, Copenhagen, Denmark, November 26–30, 2023. ACM Press. doi:10.1145/3576915.3623096. 5

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th Annual ACM Symposium on Theory of Computing*, pages 522–533, Montréal, Québec, Canada, May 23–25, 1994. ACM Press. doi:10.1145/195058.195405. 3

[DEN24] Anders P. K. Dalskov, Daniel Escudero, and Ariel Nof. Fully secure MPC and zk-FLIOP over rings: New constructions, improvements and extensions. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VIII*, volume 14927 of *Lecture Notes in Computer Science*, pages 136–169, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-68397-8_5. 5

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-48184-2_8. 3

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA. doi:10.1007/0-387-34805-0_28. 3

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-44586-2_9. 23

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. Cryptology ePrint Archive, Paper 2025/944, 2025. URL: https://eprint.iacr.org/2025/944. 3

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-32009-5_38. 10, 35

[FG26] Oriol Farràs and Miquel Guiot. Revisiting beimel-weinreb weighted threshold secret sharing schemes. CSF 2026, 2026. URL: https://eprint.iacr.org/2025/168. 12

[FL19] Jun Furukawa and Yehuda Lindell. Two-thirds honest-majority MPC for malicious adversaries at almost the cost of semi-honest. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 1557–1571, London, UK, November 11–15, 2019. ACM Press. doi:10.1145/3319535.3339811. 5

[Fra90] Yair Frankel. A practical protocol for large group oriented networks. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology – EUROCRYPT'89*, volume 434 of *Lecture Notes in Computer Science*, pages 56–61, Houthalen, Belgium, April 10–13, 1990. Springer Berlin Heidelberg, Germany. doi:10.1007/3-540-46885-4_8. 3

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press. doi:10.1145/1536414.1536440. 4

[GGW24] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-68403-6_14. 5, 40

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In David B. Shmoys, editor, *46th Annual ACM Symposium on Theory of Computing*, pages 495–504, New York, NY, USA, May 31 – June 3, 2014. ACM Press. doi:10.1145/2591796.2591861. 35

[GJM+23] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: MPC, encryption and signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 295–327, Santa Barbara, CA, USA, August 20–24,

2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38557-5_10. 3, 6, 36, 37

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy*, pages 3034–3052, San Francisco, CA, USA, May 19–23, 2024. IEEE Computer Society Press. doi:10.1109/SP54263.2024.00057. 5

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press. doi:10.1145/28395.28420. 3, 10

[GSZ20] Vipul Goyal, Yifan Song, and Chenzhi Zhu. Guaranteed output delivery comes free in honest majority MPC. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 618–646, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. doi:10.1007/978-3-030-56880-1_22. 5

[Hås94] Johan Håstad. On the size of weights for threshold gates. *SIAM J. Discret. Math.*, 7(3):484–492, 1994. doi:10.1137/S0895480192235878. 3, 6

[HLM25] Iftach Haitner, Yehuda Lindell, and Nikolaos Makriyannis. Integer commitments, old and new tools. Cryptology ePrint Archive, Paper 2025/081, 2025. URL: https://eprint.iacr.org/2025/081. 22

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-22792-9_8. 5

[KG21] Chelsea Komlo and Ian Goldberg. Frost: flexible round-optimized schnorr threshold signatures. In *Selected Areas in Cryptography: 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers 27*, pages 34–65. Springer, 2021. 3

[KNK+25] Naeem Khan, Shibli Nisar, Muhammad Asghar Khan, Yasar Abbas Ur Rehman, Fazal Noor, and Gordana Barb. Optimizing federated learning with aggregation strategies: A comprehensive survey. *IEEE Open Journal of the Computer Society*, 6:1227–1247, 2025. doi:10.1109/OJCS.2025.3590102. 3

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 357–388, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Cham, Switzerland. doi:10.1007/978-3-319-63688-7_12. 3

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume

6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-13190-5_1. 5, 37

[Mic18] Daniele Micciancio. On the hardness of learning with errors with binary secrets. *Theory Comput.*, 14(1):1–17, 2018. URL: https://doi.org/10.4086/toc.2018.v014a013, doi:10.4086/TOC.2018.V014A013. 18

[MORS24] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Rate-1 arithmetic garbling from homomorphic secret sharing. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part IV*, volume 15367 of *Lecture Notes in Computer Science*, pages 71–97, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland. doi:10.1007/978-3-031-78023-3_3. 12

[MORS25] Pierre Meyer, Claudio Orlandi, Lawrence Roy, and Peter Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In *Advances in Cryptology – CRYPTO 2025, Part IV*, Lecture Notes in Computer Science, pages 426–458, Santa Barbara, CA, USA, August 2025. Springer, Cham, Switzerland. doi:10.1007/978-3-032-01884-7_14. 12, 23

[MRV+21] Silvio Micali, Leonid Reyzin, Georgios Vlachos, Riad S. Wahby, and Nickolai Zeldovich. Compact certificates of collective knowledge. In *2021 IEEE Symposium on Security and Privacy*, pages 626–641, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press. doi:10.1109/SP40001.2021.00096. 5

[Mur71] Saburo Muroga. *Threshold Logic and Its Applications*. Wiley-Interscience, New York, 1971. 3, 6

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. doi:10.1007/s001450010003. 44

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press. doi:10.1145/1060590.1060603. 11

[RS21] Lawrence Roy and Jaspal Singh. Large message homomorphic secret sharing from DCR and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 687–717, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. doi:10.1007/978-3-030-84252-9_23. 12, 21

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252, Santa Barbara, CA, USA, August 20–24, 1990. Springer, New York, USA. doi:10.1007/0-387-34805-0_22. 5

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. 4

[Vai20] Vinod Vaikuntanathan. Lattices, learning with errors and post-quantum cryptography: Lecture notes. https://people.csail.mit.edu/vinodv/CS294/lecturenotes.pdf, 2020. 19

[XLL+24] Runhua Xu, Bo Li, Chao Li, James B. D. Joshi, Shuai Ma, and Jianxin Li. Tapfed: Threshold secure aggregation for privacy-preserving federated learning. *IEEE Transactions on Dependable and Secure Computing*, 21(5):4309–4323, 2024. doi:10.1109/TDSC.2024.3350206. 3

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167, Toronto, Ontario, Canada, October 27–29, 1986. IEEE Computer Society Press. doi:10.1109/SFCS.1986.25. 3

[Yao89] Andrew Chi-Chih Yao. Unpublished manuscript. Presented at Oberwolfach and DIMACS workshops, 1989. 3, 4, 6, 7

[ZMGN20] Huafei Zhu, Rick Siow Mong Goh, and Wee-Keong Ng. Privacy-preserving weighted federated learning within the secret sharing framework. *IEEE Access*, 8:198275–198284, 2020. doi:10.1109/ACCESS.2020.3034602. 3