# Distributed Monotone-Policy Encryption with Silent Setup from Lattices

Abtin Afshar
UW-Madison*

Rishab Goyal
UW-Madison†

Saikumar Yadugiri
UW-Madison‡

**Abstract**

Distributed cryptography serves as a cornerstone for building trustless systems, yet existing solutions for distributed policy encryption typically require either a trusted dealer or complex interactive setup protocols. Recent advances have introduced the concept of *silent setup*, where users independently generate keys and a joint public key is derived deterministically. However, all current silent-setup constructions rely on bilinear pairings, leaving them vulnerable to quantum adversaries. In this work, we present the first distributed monotone-policy encryption scheme with silent setup based on lattices. Our construction supports arbitrary monotone policies represented as Disjunctive Normal Forms (DNFs) and provides a robust, post-quantum foundation for decentralized access control.

## 1 Introduction

The paradigm of distributed cryptography is essential for eliminating single points of failure in modern digital infrastructure. By splitting sensitive cryptographic capabilities—such as decryption or signing—among a set of $N$ parties, we ensure that security is maintained as long as a sufficient subset of parties remains honest. In the context of encryption, *Distributed Policy Encryption* (DPE) allows a sender to encrypt a message such that it can only be recovered if a specific subset of users, whose identities satisfy a given monotone access policy (e.g., threshold gates, CNF, or DNF), cooperate to decrypt. DPE allows a sender to encrypt a message under an access policy $\phi$, such that decryption is only possible if a set of parties $S \subseteq \{1, \ldots, N\}$ co-operate, where $\phi(S) = 1$.

Traditionally, distributed systems have been hampered by the "setup bottleneck." Classic threshold and policy-based systems generally fall into two categories:

1. **Trusted Setup:** A Central Authority (CA) generates a master secret key, derives shares for all $N$ parties, and then deletes the master key. This introduces a "honeypot" during the setup phase and requires users to trust the CA's honesty and deletion procedures.
2. **Interactive Distributed Key Generation (DKG):** Participants run a multi-party computation protocol to generate keys. While decentralized, DKG protocols are notoriously heavy in communication, require participants to be online simultaneously, and often struggle with scalability as the number of parties $N$ grows.

**The Rise of Silent Setup.** To address these limitations, recent literature has converged on the notion of **Silent Setup** (also known as non-interactive or decentralized setup) [WQZD10, BZ14, MRV+21, DCX+23, GJM+24, GKPW24, BCF+25, DJWW26]. In this model, each user $i$ independently generates their own secret-public key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and publishes $\mathsf{pk}_i$ to a bulletin board. Any encryptor can then deterministically derive a "joint" public key for any subset of users and any policy $\phi$, without any interaction between the users or a dealer. This enables truly ad-hoc distributed encryption: users can join the system by simply posting a public key to a bulletin board, without ever interacting with other participants.

*Email: abtin@cs.wisc.edu.
†Email: rishab@cs.wisc.edu. Research supported by a gift from the Stellar Development Foundation.
‡Email: saikumar@cs.wisc.edu.

**The Post-Quantum Hole.** Despite these advances, a critical challenge remains: *all existing distributed policy encryption schemes with silent setup are built on "quantum-prone" assumptions.* Existing DPE designs for threshold policies [WW26] rely heavily on the algebraic properties of bilinear maps (such as those found in Boneh-Boyen or Waters-style Identity-Based Encryption [BB04, Wat05]) to aggregate keys and ciphertexts. While DPE for more expressive monotone-policies [DJWW26] require heavy cryptographic machinery such as indistinguishability obfuscation [BGI+01, GGH+16]. Consequently, these systems are inherently vulnerable to quantum attacks, as bilinear maps are trivially broken by Shor's algorithm [Sho97], and current obfuscation constructions [JLS21, JLS22, RVV24] crucially rely on bilinear maps.

As the industry and government standards (e.g., NIST) transition toward quantum-resistant primitives, the absence of a lattice-based alternative for silent-setup DPE is a significant void. Constructing such a scheme from lattices is non-trivial; the "key-aggregation" and "policy-embedding" techniques used in pairing-based groups do not translate naturally to the noisy, linear-algebraic structure inherent to lattice-based cryptography. Furthermore, implementing monotone policies (like DNFs) requires a way to "branch" the decryption logic without a central coordinator. While Attribute-Based Encryption (ABE) from lattices handles complex policies, standard ABE requires a *Master Secret Key* (MSK) held by a central authority to issue user keys. Transforming this into a dealerless, silent-setup distributed system requires a fundamental rethinking of how lattice-based ciphertexts interact with independently generated user keys.

**Our Contributions.** In this paper, we bridge this gap by proposing the first *Distributed Monotone-Policy Encryption from Lattices with Silent Setup.* Our construction provides:

- **Post-Quantum Security:** Security is reduced to the hardness of the $\ell$-decomposed Learning With Errors (LWE) problem [AMR25] (a falsifiable lattice assumption) in Random Oracle Model (ROM) [BR93].
- **Expressive Policy Support:** We support arbitrary monotone policies that can be represented as Disjunctive Normal Forms (DNFs).
- **Silent Setup:** Users generate keys locally, and no interaction is required to aggregate the keys or define policies.
- **Decentralized Flexibility:** Senders can choose the policy and the set of participants dynamically at encryption time.

Ultimately, by moving into the lattice regime, we demonstrate that expressive, decentralized access control can be achieved without sacrificing either a trustless setup or post-quantum security.

**Concurrent Work.** In a concurrent and independent work, Champion and Wu [CW26] also construct a distributed monotone-policy encryption scheme from lattices with silent setup. Their results and core techniques are conceptually similar to our work with a few minor differences in presentation. The main differences between the two papers are:

- **Aggregate Public Key.** In our work, we explicitly design a public pre-processing algorithm for ensuring encryption time is independent of the number of users $N$, and policy size $|\phi|$. While in [CW26], the encryption algorithm's runtime scales polynomially with $N$ and $|\phi|$. They also informally sketched an approach similar to ours to enable aggregation.
- **Security.** We prove selective security, while [CW26] prove semi-selective security, where the adversary can declare the challenge terms after receiving CRS and also make partial decryption queries on non-challenge ciphertexts.

**Related Work.** Distributed threshold encryption with silent setup was proposed by Garg et al. [GKPW24]. Distributed Broadcast Encryption (DBE) [WQZD10, BZ14] is a closely related primitive in which the the threshold is 1. There have been lot of works in the domain of DBE from standard pairing-based primitives [WQZD10, KMW23], stronger non-falsifiable primitives such as obfuscation or witness encryption

[BZ14, FWW23], and falsifiable lattice based assumptions [CW24, CHW25, WW25, AMR25]. Schemes that venture beyond DBE have thus far required either heavy cryptographic machinery such as indistinguishability obfuscation or witness encryption [RSY21, ADM+24, DJWW26] or reliance on idealized models [MRV+21, DCX+23, GJM+24, GKPW24, BCF+25], or bilinear pairing based approaches [WW26]. In particular, there are no known falsifiable lattice-based approaches for distributed monotone-policy encryption with silent setup.

## 2 Technical Overview

A Distributed Monotone Policy Encryption (DPE) scheme, for monotone access structures $\mathbb{S}$, consists of five main algorithms $(\mathsf{GSetup}, \mathsf{KGen}, \mathsf{PreProcess}, \mathsf{Enc}, \mathsf{Dec})$. The global setup algorithm $\mathsf{GSetup}$ outputs $\mathsf{crs}$ which the rest of the algorithms use implicitly. Each user locally runs the $\mathsf{KGen}$ algorithm to generate a public-secret key pair $(\mathsf{pk}, \mathsf{sk})$. The fact that $\mathsf{KGen}$ is non-interactive and asynchronous is commonly referred to as *silent setup* [GKPW24]. In the distributed setting, each user posts their public keys on a bulletin board, where any subset of these can be accumulated to create a short "master" public key for that subset. In particular, DPE has a *public-coin* $\mathsf{PreProcess}$ algorithm that uses public keys of users in set $S, \{\mathsf{pk}_i\}_{i\in S}$, and access structure $\mathbb{A} \in \mathbb{S}$ over $|S|$ parties to generate a short master public key $\mathsf{mpk}$ (which includes sampled random coins).[1]

Given a master public key $\mathsf{mpk}$, one can run the $\mathsf{Enc}$ algorithm to conceal a message $\mu \in \{0,1\}$ in a ciphertext $\mathsf{ct}$. Finally, the $\mathsf{Dec}$ algorithm uses $\{\mathsf{sk}_j\}_{j\in S'}, \mathsf{ct}$ and outputs $\mu'$. Prior works [GKPW24, WW26] also define the concept of one-round distributed decryption, where each user locally computes a partial decryption share (without interaction), *and* a set of partial shares for an authorized group of users can be combined to recover the plaintext. As we describe later, our DPE scheme also natively supports such a one-round distributed decryption process. To ensure DPE is non-trivial and meaningful, it is essential that the ciphertext is compact. That is, $|\mathsf{ct}| = o(|\mathbb{A}|)\mathsf{poly}(\lambda)$.

**Security for DPE.** Consider the following game.

**Setup Phase.** $\mathcal{A}$ declares number of users $N \leq 2^\lambda$.

**Query Phase.** $\mathcal{A}$ can create a user $i$ and receive $\mathsf{pk}_i$. $\mathcal{A}$ can also corrupt any user $j$ that was previously created and receives $\mathsf{sk}_j$.

**Challenge Phase.** $\mathcal{A}$ declares a set $S \subseteq [N]$ (of previously created users), and an access policy $\mathbb{A} \in \mathbb{S}$ over $[|S|]$ parties and receives $(\mathsf{mpk}, \mathsf{ct})$ where $\mathsf{mpk} \leftarrow \mathsf{PreProcess}(\{\mathsf{pk}_i\}_{i\in S}, \mathbb{A}), \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mu)$ for a random $\mu \leftarrow \{0,1\}$.

**Post-challenge corruptions.** $\mathcal{A}$ can continue to corrupt users in set $S$.

A DPE scheme is secure as long as any poly-time attacker $\mathcal{A}$, that corrupts users $C$ such that $C \cap S \notin \mathbb{A}$, has negligible advantage in guessing $\mu$. In this work, we consider the *selective* security notion where $\mathcal{A}$ declares challenge set $S$, corruptions $C$, and access policy $\mathbb{A}$ such that $C \cap S \notin \mathbb{A}$ before viewing $\mathsf{crs}$.

**Natural starting point and its failure.** In the lattice-based setting, a natural first idea to design DPE would be to rely on the underlying decryption functions being linear, and combine this with a linear secret sharing scheme. This is indeed what has been successfully used in prior works such as [BGG+18, EY24] where, by generating a linear secret sharing of a master secret key and distributing the decryption capability, one could design a "universal thresholdizer" to generically turn any cryptosystem into its thresholdized counterpart.

While this seems like an excellent first step to design DPE, this fails for a fundamental reason. Prior works relied on a *centralized* setup, which is strictly prohibited in the silent setup model. Essentially, the top-down approach to distribute the master secrete key (commonly used in prior works) is incompatible because in the silent setup model there is no trusted dealer to share a pre-existing key. To circumvent

[1] Prior works [CHW25, WW25] make use of random oracles to de-randomize this public-coin algorithm. In our DPE construction, we can also use a random oracle to de-randomize this algorithm. However, for ease of exposition, we stick to the public-coin notation.

this, our idea is to leverage techniques from recent lattice-based works that design Distributed Broadcast Encryption (DBE) [CW24, CHW25, WW25, AMR25].

## 2.1 A new starting point: DBE

A DBE [WQZD10, BZ14] scheme is a specialization of DPE, where the access policy $\mathbb{A}$ is fixed and is simply $i_1 \vee \ldots \vee i_{|S|}$ where $S = \{i_1, \ldots, i_{|S|}\} \subseteq [N]$. In words, the access policy is an 'OR' tree. Since any user in $S$ can decrypt $\mathsf{ct}$ in DPE and learn the underlying message, thus an adversary cannot corrupt any users in $S$. In words, $C \cap S$ must be $\emptyset$, where $C$ is the set of corrupted users. That is, an adversary cannot corrupt **any** user which can compute a *legitimate* partial decryption.

This ($C \cap S = \emptyset$) is a serious limitation, and makes the task of designing DBE far easier than DPE. However, our insight is that the underlying lattice framework developed in prior DBE schemes is quite powerful, and can be used as an important building block for designing DPE from lattices. To explain our main ideas, we start with a simplified review of the recent Wee-Wu DBE scheme [WW25]. Our presentation slightly deviates from the actual construction presented in [WW25]. We provide this alternate presentation as it helps conveying our main ideas.

**Reviewing [WW25] DBE.** Their DBE construction relies on a new tool called matrix commitment scheme [Wee25] and its security is proven under hardness of decomposed (succinct) LWE assumption [Wee24, AMR25], a new falsifiable lattice assumption. We briefly review these next, and a reader familiar with these concepts can safely skip the next paragraph.

**Matrix commitments and Decomposed LWE.** Let $(n, m, q, \chi, \sigma)$ be Learning with Errors (LWE) [Reg05] parameters and $\mathsf{com.pp} := (\mathbf{W}, \mathbf{D}, \mathbf{B})$ be from the decomposed LWE assumption [AMR25]. Then, there exist three *deterministic* algorithms $(\mathsf{Com}, \mathsf{Verify}, \mathsf{Open})$ which take $\mathsf{com.pp}$ implicitly such that $\forall\ N$, any $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$, $\mathbf{C}\mathbf{V}_N = \mathbf{M} - \mathbf{B}\mathbf{Z}$. Here, $\mathbf{C} = \mathsf{Com}(\mathbf{M}), \mathbf{V}_N = \mathsf{Verify}(1^N), \mathbf{Z} = \mathsf{Open}(\mathbf{M}), \mathbf{C} \in \mathbb{Z}_q^{n \times m}, \mathbf{V}_N \in \mathbb{Z}_q^{m \times N}, \mathbf{Z} \in \mathbb{Z}_q^{m \times N}$. Moreover, $\mathbf{V}_N, \mathbf{Z}$ have bounded norms. Decomposed LWE assumption [AMR25] states that given $(\mathbf{W}, \mathbf{D}, \mathbf{B})$[2], $\{\mathbf{s}^\top \mathbf{B} + \mathsf{err}\} \approx_c \{\mathbf{c}^\top\}$ where $\mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{c} \leftarrow \mathbb{Z}_q^m$ (check Assumption 3.6). In their WW selective DBE scheme for $N$ users, the CRS and keys are sampled as:

- $\mathsf{crs} = (\mathsf{com.pp} := (\mathbf{W}, \mathbf{D}, \mathbf{B}), \mathbf{A}, \mathbf{p}, N)$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{p} \leftarrow \mathbb{Z}_q^n$
- $(\mathsf{pk}_i, \mathsf{sk}_i) = (\mathbf{t}_i, \mathbf{r}_i)$, such that $\mathbf{t}_i := \mathbf{B}\mathbf{r}_i - \mathbf{A}\mathbf{v}_i, \mathbf{r}_i \leftarrow \{\pm 1\}^m$

where $\mathbf{v}_i$ is $i$-th column of $\mathbf{V}_N = \mathsf{Verify}(1^N)$ from matrix commitments. The $\mathsf{Enc}$ algorithm uses public keys $\{\mathbf{t}_i\}_{i \in S}$ to construct a matrix $\mathbf{M} = [\mathbf{m}_1 \mid \ldots \mid \mathbf{m}_N] \in \mathbb{Z}_q^{n \times N}$ where $\mathbf{m}_j = \mathbf{t}_j + \mathbf{p}$ if $j \in S$ and $\mathbf{0}$ otherwise. Then, $\mathsf{Enc}$ computes $\mathbf{C} = \mathsf{Com}(\mathbf{M})$, samples $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and outputs $\mathsf{ct} := (\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3) := (\mathbf{s}^\top \mathbf{B} + \mathsf{err}, \mathbf{s}^\top(\mathbf{A} + \mathbf{C}) + \mathsf{err}, \mathbf{s}^\top \mathbf{p} + \mathsf{err} + \lfloor q/2 \rfloor \mu)$. In order to decrypt given $\mathbf{r}_j$ for $j$-th user, $\mathsf{Dec}$ computes $\mathbf{V}_N = \mathsf{Verify}(1^N)$ and also $\mathbf{M}$ using the public keys $\{\mathbf{t}_i\}_{i \in S}, \mathbf{p}$ similar to above. Using $\mathbf{M}$, $\mathsf{Dec}$ can compute $\mathbf{Z} = \mathsf{Open}(\mathbf{M})$. Then, $\mathsf{Dec}$ computes the large vector $\mathbf{c}^\top = \mathbf{c}_2^\top \mathbf{V}_N + \mathbf{c}_1^\top \mathbf{Z} \approx \mathbf{s}^\top(\mathbf{A}\mathbf{V}_N + \mathbf{M}) \in \mathbb{Z}_q^{1 \times N}$. Note that approximately the $j$-th element of $\mathbf{c}^\top$ is $\mathbf{s}^\top(\mathbf{t}_j + \mathbf{A}\mathbf{v}_j) = \mathbf{s}^\top(\mathbf{B}\mathbf{r}_j + \mathbf{p})$ if $j \in S$. Thus, using $\mathbf{r}_j$ and $\mathbf{c}_1^\top$, $\mathsf{Dec}$ computes $\mathbf{s}^\top \mathbf{p}$ which can be used to retrieve $\mu$ with high probability.

**Proving selective security.** For a challenge set $S \subseteq [N]$, let $(\mathsf{crs}, \{\mathbf{t}_i\}_{i \in S}, \mathsf{ct})$ denote the keys and challenge ciphertext generated by the challenger.

1. The proof starts with an application of leftover hash lemma, where we randomly sample $\mathbf{t}_i \leftarrow \mathbb{Z}_q^n$ for each $i \in S$. This follows from the fact that none of the corresponding secret keys are corrupted.
2. Next, we can apply leftover hash lemma again, but this time to set $\mathbf{A}$ as $\mathbf{B}\mathbf{R}_\mathbf{A} - \mathsf{Com}(\mathbf{M})$, $\mathbf{p}$ as $\mathbf{B}\mathbf{r}_\mathbf{p}$. This crucially relies on the fact that $\mathbf{t}_i$'s have been turned truly random. This step lets us set $\mathsf{ct}$ as $(\mathbf{c}_1^\top, \mathbf{c}_2^\top \approx \mathbf{c}_1^\top \mathbf{R}_\mathbf{A}, c_3 \approx \mathbf{c}_1^\top \mathbf{r}_\mathbf{p} + \lfloor q/2 \rfloor \mu)$.
3. Finally, using decomposed LWE security, we can change $\mathbf{c}_1^\top$ to a random string, which then would mean $c_3$ loses all information about $\mu$.

[2] We deliberately and incorrectly use $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ throughout overview. For precise formulation check Assumption 3.6.

## 2.2 Supporting Monotone Policy Encryption

Note that in DBE any user in $S$ can decrypt and learn $\mu$; while, in DPE, we only want an *authorized set of users* to learn $\mu$ jointly according to access policy $\mathbb{A}$. In DBE, the $j$-th user can compute $\mathbf{s}^\top(\mathbf{B}\mathbf{r}_j + \mathbf{p})$ and using its secret key it can remove the $\mathbf{s}^\top\mathbf{B}\mathbf{r}_j$ term to learn $\mathbf{s}^\top\mathbf{p}$, which is the one-time pad mask used to encrypt $\mu$.

Our idea is to *secret share* this one-time pad mask $\mathbf{s}^\top\mathbf{p}$. That is, we secret share the *"target"* vector $\mathbf{p}$, and only let the $j$-th user learn $\mathbf{s}^\top(\mathbf{B}\mathbf{r}_j+\mathbf{p}_j)$, where $\mathbf{p}_j$ is the $j$-th secret share of $\mathbf{p}$ as per the underlying monotone policy. Now user $j$ can only recover $\mathbf{s}^\top\mathbf{p}_j$, and our hope is that (by relying on linear reconstruction properties of the underlying secret sharing scheme) a set of authorized users can recompute $\mathbf{s}^\top\mathbf{p}$ from a set of $\{\mathbf{s}^\top\mathbf{p}_j\}_{j\in S}$ for an authorized set $S$. To ensure that this reconstruction process does not blow up the errors, it is crucial that the reconstruction coefficient are small. Fortunately, we already have linear secret sharing scheme with binary reconstruction coefficients (called $\{0,1\}$-LSSS) [BL90, Bei96, CK93, BC02, BR07, LW11, BGG+18]. Let us explain how to execute all of this more formally.

**DPE from Decomposed LWE.** During encryption, instead of embedding the same $\mathbf{p}$ in each column of matrix $\mathbf{M}$, we secret share $\mathbf{p}$ and embed one share of $\mathbf{p}$ in each column using a $\{0,1\}$-LSSS. The setup and key generation algorithms are largely unchanged, except we no longer sample a random $\mathbf{p}$ globally; instead, we sample a fresh $\mathbf{p}$ separately for each subset of users (during public key aggregation) as we explain next.

Given $\{\mathsf{pk}_i\}_{i\in S}, \mathbb{A}$, the $\mathsf{PreProcess}$ algorithm samples a random $\mathbf{p} \leftarrow \mathbb{Z}_q^n$, computes $(\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_{|S|}}) = \mathsf{LSSS.Share}(\mathbb{A}, \mathbf{p}; r)$ using random coins $r$, and constructs matrix $\mathbf{M} = [\mathbf{m}_1 \mid \ldots \mid \mathbf{m}_N]$ where $\mathbf{m}_j = \mathbf{t}_j + \mathbf{p}_j$ if $j \in S$ and $\mathbf{0}$ otherwise. $\mathsf{PreProcess}$ then computes $\mathbf{C} = \mathsf{Com}(\mathbf{M})$ sets $\mathsf{mpk} = (\mathbf{C}, \mathbf{p}, r)$. We remind the reader that we are viewing our $\mathsf{PreProcess}$ algorithm to be a randomized, but public-coin algorithm. As we mentioned, this can be generically de-randomized in the Random Oracle Model [BR93]. Given such an accumulated $\mathsf{mpk}$, the encryptor simply samples $\mathbf{s} \leftarrow \mathbb{Z}_q^n$ and constructs $\mathsf{ct}$ as $(\mathbf{p}^\top, \mathbf{s}^\top\mathbf{B} + \mathsf{err}, \mathbf{s}^\top(\mathbf{A} + \mathbf{C}) + \mathsf{err}, \mathbf{s}^\top\mathbf{p} + \lfloor q/2 \rfloor \mu + \mathsf{err})$. That is, an encryptor no longer creates the commitment matrix $\mathbf{C}$ but picks it from $\mathsf{mpk}$.

To decrypt such ciphertexts, we will compute $\mathbf{c}^\top = \mathbf{c}_2^\top\mathbf{V}_N + \mathbf{c}_1^\top\mathbf{Z} \approx \mathbf{s}^\top(\mathbf{A}\mathbf{V}_N + \mathbf{M})$ where $\mathbf{V}_N = \mathsf{Verify}(1^N), \mathbf{Z} = \mathsf{Open}(\mathbf{M})$, as in DBE. Given $\{\mathsf{sk}_j\}_{j\in S'}$, for each $j$, we take the $j$-th entry of $\mathbf{c}^\top$ which is roughly $\mathbf{s}^\top(\mathbf{t}_j + \mathbf{p}_j)$ if $S' \subseteq S$. Then using the secret key $\mathbf{r}_j$ and $\mathbf{c}_1^\top$, we can learn $\{\mathbf{s}^\top\mathbf{p}_j + \mathsf{err}\}_{j\in S'}$. We also remark that we can transform this decryption procedure into one-round 'threshold' decryption using noise smudging techniques similar to [MW16, BGG+18]. Now, because $\{0,1\}$-LSSS has linear reconstruction with coefficients in $\{0,1\}$ [LW11, BGG+18], if $S' \in \mathbb{A}$, we can add the relevant shares to compute $\mathbf{s}^\top\mathbf{p} + \mathsf{err}$ which reveals $\mu$ with high probability.

**How to open matrix M?** Unfortunately, the above process does not work just yet. This is because $\mathsf{Dec}$ cannot compute the matrix $\mathbf{M}$ given just $\mathbf{p}$. Note that $\mathbf{M}$ is publicly computable using $\{\mathbf{t}_i\}_{i\in S}$ and $(\mathbf{p}_{i_1}, \ldots, \mathbf{p}_{i_{|S|}}) = \mathsf{LSSS.Share}(\mathbb{A}, \mathbf{p}; r)$. Thus, it requires the shares $\mathbf{p}_{i_j}$ for each $j \in [|S|]$. For succinctness, we cannot simply include these shares as part of the ciphertext. However, our idea is to simply include the random coins $r$ (used by $\mathsf{LSSS.Share}$) as part of the ciphertext itself. This could be problematic for two reasons – (1) compactness, and (2) security. First, the length of $r$ could be quite big depending upon the access structure. Luckily, there is an easy folklore fix for this. $\mathsf{PreProcess}$ samples a short pseudorandom generator (PRG) seed $\sigma \leftarrow \{0,1\}^\lambda$, use $\sigma$ to generate random coins for LSSS, and include $\sigma$ as part of $\mathsf{mpk}$ and $\mathsf{ct}$. However, the second issue appears to be more worrisome! While exposing secret shares is typically catastrophic, we observe that it does not negatively impact our scheme, and actually serves our underlying strategy very well.

**Why is it okay to release LSSS randomness $r$?** Our key strategy is to use the secret sharing scheme to construct (what we call) a *'dual-mode LWE instance'*. That is, we are not trying to hide $\mathbf{p}$ at all. During decryption, a user $j$ computes a component that intuitively takes the form:

$$j^{th} \text{ user learns } \approx \mathbf{s}^\top(\mathbf{t}_j + \mathbf{p}_j) = \underbrace{\mathbf{s}^\top\mathbf{t}_j}_{\text{Key Term}} + \underbrace{\mathbf{s}^\top\mathbf{p}_j}_{\text{Shared Mask Term}}$$

Our insight is that LSSS let us use the (decomposed) LWE assumption for every user (both, corrupt and honest) to randomize the above term completely. In particular, our strategy is to leverage the fact that for any user, only one of these two components needs to be truly random. For an *honest* user, their public key $\mathbf{t}_j$ is honestly generated and uniform, meaning the left term (Key Term) effectively acts as the LWE challenge to completely hide their right term (Shared Mask Term). Conversely, for a *corrupt* user, the adversary may maliciously control $\mathbf{t}_j$, but since the set of corrupt users must be an unauthorized set, thus, by leveraging LSSS secrecy guarantee, we could set their individual shares $\mathbf{p}_j$ as uniformly random. Thus, for corrupt users, the right term takes over as the LWE challenge to hide the left term. Ultimately, this allows us to hide the shares of honest users using the fact that they are honest, while simultaneously exploiting the public (but uniform) shares of corrupt users to maintain security. Formalizing this intuition, however, requires a clever programming of the shares within the security reduction as we explain next.

### Three barriers for arguing security

Although the above design provides the necessary functionality, proving its security is tricky. We point out three barriers. At a high level, all of these barriers are because of the fact that DPE provides significantly stronger functionality than DBE and this gives an attacker more opportunity to learn non-trivial leakage from a challenge ciphertext. At its core, the difference between DPE and DBE can be boiled down to the the structure of the corrupted set of users; namely, in DBE $C = \emptyset$, whereas in DPE, $C \cap S \notin \mathbb{A}$. In particular, in DPE $C$ does not need to be empty.

**Barrier 1: Correlated Secret Shares.** As mentioned, our $\mathsf{Dec}$ procedure has the special property that if $\mathsf{ct}$ is generated for users $S$, then any user $j \in S$ can decrypt the ciphertext using $\mathsf{sk}_j$ locally. This would mean that an attacker that queries for a non-empty $C$ in the selective security game can figure out $\mathbf{s}^\top \mathbf{p}_j + \mathsf{err}$ on its own by computing a part of decryption honestly. These $\{\mathbf{s}^\top \mathbf{p}_j + \mathsf{err}\}_{j \in C}$ are correlated with $\mathbf{s}^\top \mathbf{p} + \mathsf{err}$ via $\{0,1\}$-LSSS.

Thus, although we do not exactly rely on the security of $\{0,1\}$-LSSS, we still require some properties that lets us generate random shares for any unauthorized set, and a random uncorrelated secret, and later derive the rest of the shares from this information. This way, we can break the correlation and overcome this barrier. To resolve this, we introduce the notion of *Programmable Binary Linear Secret Sharing Schemes* (PLSSS). We elaborate on this below.

**Programmable binary LSSS (PLSSS).** A PLSSS is a $\{0,1\}$-LSSS that is additionally equipped with two *deterministic* programmed sharing algorithms. Fix an access structure $\mathbb{A} \in \mathbb{S}$ and let $T_1, T_2 \subseteq [N]$ be *maximal invalid* share sets (w.r.t. the LSSS matrix for $\mathbb{A}$) with target shares $(\widetilde{\mathsf{sh}}_{1,j})_{j \in T_1}$ and $(\widetilde{\mathsf{sh}}_{2,j})_{j \in T_2}$ respectively. Then the programmed sharing in two modes works as follows:

- Mode 1: $(k_1, \mathsf{sh}_{1,1}, \ldots, \mathsf{sh}_{1,N}) = \mathsf{ProgShare}_1(\mathbb{A}, T_1, (\widetilde{\mathsf{sh}}_{1,j})_{j \in T_1}, j^*, \widetilde{\mathsf{sh}}_{j^*})$, where $j^* \notin T_1$ is one additional index with target value $\widetilde{\mathsf{sh}}_{j^*} \in \mathbb{Z}_q$.
- Mode 2: $(k_2, \mathsf{sh}_{2,1}, \ldots, \mathsf{sh}_{2,N}) = \mathsf{ProgShare}_2(\mathbb{A}, T_2, (\widetilde{\mathsf{sh}}_{2,j})_{j \in T_2}, k)$ where $k \in \mathbb{Z}_q$ is the target secret.

We require (a) consistency of output and input programmed values, i.e., $(\mathsf{sh}_{\ell,1}, \ldots, \mathsf{sh}_{\ell,N})$ contain the programmed values $\widetilde{\mathsf{sh}}_{\ell,j}$ in the intended coordinates for both $\ell = 1, 2$ (i.e., $\widetilde{\mathsf{sh}}_{\ell,j} = \mathsf{sh}_{\ell,j}$ and in Mode 2 also satisfy $k_2 = k$); and (b) mode indistinguishability which means if $(\widetilde{\mathsf{sh}}_{1,j})_{j \in T_1}, \widetilde{\mathsf{sh}}_{j^*}$ and $(\widetilde{\mathsf{sh}}_{2,j})_{j \in T_2}, k$ are sampled uniformly and independently in $\mathbb{Z}_q$, the resulting outputs $(k_1, \mathsf{sh}_{1,1}, \ldots, \mathsf{sh}_{1,N})$ and $(k_2, \mathsf{sh}_{2,1}, \ldots, \mathsf{sh}_{2,N})$ are *identically distributed*.

**PLSSS for DNFs via Lewko–Waters.** For DNF access policies, we instantiate the underlying $\{0,1\}$-LSSS using the Lewko-Waters transformation [LW11]. The resulting LSSS enjoys a useful *linear-independence for unauthorized sets* property: for any invalid share set $T$, the corresponding share rows are linearly independent. Consequently, any *maximal* invalid share set $T$ spans a subspace of co-dimension 1 that still avoids the *secret direction* $\mathbb{1}_1 = (1, 0, \ldots, 0)$: it is missing exactly one independent constraint to force $\mathbb{1}_1$ into the span (i.e., to make the set valid and enable reconstruction). Adding that last constraint preserves independence and

yields a unique solution. Concretely, adding any extra share row $j^* \notin T$ (Mode 1) or the secret constraint $\langle \mathbb{1}_1, \cdot \rangle = k$ (Mode 2) makes the augmented linear system full-rank, and hence uniquely solvable. Moreover, since in both modes the augmented system is full-rank, independent and uniformly random, programming inputs induce a uniformly random solution, and therefore the resulting outputs in Mode 1 and Mode 2 are *identically distributed* (i.e., mode indistinguishability holds).

**Barrier 2: PRGs and Programmability.** Even with the PLSSS abstraction successfully resolving the correlation of secret shares (Barrier 1), we encounter a second major roadblock regarding the secret sharing randomness. To maintain a succinct ciphertext, it is natural to derive random coins required for $\mathsf{LSSS.Share}$ from a PRG seed $\sigma$. However, there is still an issue left here.

Recall that to simulate the challenge ciphertext, our reduction relies on the deterministic algorithms $\mathsf{ProgShare}_1$ and $\mathsf{ProgShare}_2$ to explicitly fix the shares for the maximal invalid set to uniformly random values. If the shares are generated via PRG randomness, the reduction is cryptographically bound. It cannot reverse-engineer or "program" the PRG's output to exactly match these targeted programmed shares ($\widetilde{\mathsf{sh}}_j$) without breaking the PRG itself. Conversely, giving out true, uncompressed random coins of $\mathsf{LSSS.Share}$ allow for this programmability, but the size of these coins grows linearly with the number of parties $|S|$, destroying the compactness of the ciphertext.

Our solution is to operate in the Random Oracle Model (ROM) [BR93]. By using $\mathcal{H}$, a random oracle, the reduction gains the critical ability to dynamically program the oracle's outputs during the security game. Specifically, the reduction can simply set the oracle responses (e.g., $\mathcal{H}(\mathsf{sd}, S)$) to equal the exact simulated shares output by our PLSSS algorithms. This flawlessly aligns the secret sharing distribution with the simulated LWE challenges, all while only requiring the encryptor to include the short seed $\mathsf{sd} \in \{0,1\}^\lambda$ in $\mathsf{ct}$. Constructing a succinct DPE scheme that circumvents this programming barrier in the standard model remains an excellent open question.

**Barrier 3: Commitment Programmability.** In the selective security proof of DBE, in Step (2), we were able to program $\mathbf{C} = \mathsf{Com}(\mathbf{M})$ into the $\mathsf{crs}$. This is because the corruption set $C = \emptyset$, we can set each $\mathbf{t}_i \leftarrow \mathbb{Z}_q^n$ for each $i \in S$. However, we cannot do the same in DPE. As $C$ need not be empty, $\{\mathbf{t}_j\}_{j \in C}$ depend on $\mathbf{A}$ and thus we cannot compute $\mathbf{M}$ before setting the $\mathsf{crs}$. Similar issues have propped up in the context of registered ABE from lattices in [CHW25, WW25]. To resolve this barrier, we make use of the Explainable Re-randomizer scheme (ER) of [CHW25].

An ER scheme allows us to sample a *random* commitment to $\mathbf{0} \in \mathbb{Z}_q^{n \times N}$ for any $N \in \mathbb{N}$. That is, there exists a randomized algorithm $(\mathbf{C}_0, \mathbf{Z}_0) = \mathsf{ER.Sample}(1^\lambda, 1^N; r)$ that uses random coins $r$ such that $|r| = \mathsf{poly}(\lambda, N)$. In addition, there is another algorithm $\mathsf{Explain}$, $r' \leftarrow \mathsf{ER.Explain}(1^\lambda, \mathbf{C}_0, \mathbf{Z}_0)$. These algorithms satisfy the following properties.

**Correctness.** For any $(\mathbf{C}_0, \mathbf{Z}_0) \leftarrow \mathsf{ER.Sample}(1^\lambda, 1^N)$, $\mathbf{C}_0 \mathbf{V}_N = -\mathbf{B}\mathbf{Z}_0$.
**Randomness.** $\left\{(\mathbf{C}_0, \mathbf{Z}_0) : (\mathbf{C}_0, \mathbf{Z}_0) \leftarrow \mathsf{ER.Sample}(1^\lambda, 1^N)\right\} \approx_s \left\{(\mathbf{C}_0, \mathbf{Z}_0) : \mathbf{C}_0 \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{Z}_0 \leftarrow \mathbf{B}^{-1}(-\mathbf{C}_0 \mathbf{V}_N)\right\}$
**Explainability.** $\{(\mathbf{C}_0, \mathbf{Z}_0, r)\} \approx_s \{(\mathbf{C}_0, \mathbf{Z}_0, r')\}$
where $(\mathbf{C}_0, \mathbf{Z}_0) = \mathsf{ER.Sample}(1^\lambda, 1^N; r)$ and $r' \leftarrow \mathsf{ER.Explain}(1^\lambda, \mathbf{C}_0, \mathbf{Z}_0)$.

Using this, we can completely remove $\mathbf{A}$ in the construction and instead sample a random $\mathbf{C}_0$ using random coins from a random oracle. In the security proof, when we know $\mathbf{M}$ completely during challenge phase, we can then sample a random $\mathbf{C}_0$ and explain our sampling using $\mathsf{ER.Explain}$. This lets us set $\mathbf{C}_0 = \mathbf{B}\mathbf{R} - \mathsf{Com}(\mathbf{M})$ and rely on decomposed LWE security.

After modifying the base DPE construction with these fixes, we can construct a selectively-secure DPE scheme for DNF formulas.

**On the necessity of DNFs.** The restriction to Disjunctive Normal Form (DNF) access policies in our construction arised because of the strict constraints our proof strategy places on the underlying secret sharing scheme. Crucially, our system requires a LSSS that supports linear reconstruction with short coefficients, maintains share sizes that are independent of the total number of users $N$, and has linearly independent unauthorized share set rows. The last property is required for program / back-propagate shares and randomness. To the best of our knowledge, the only existing LSSS that successfully achieves all of these properties

simultaneously is the Lewko-Waters [LW11] style $\{0,1\}$-LSSS for DNF formulas [ABN+22, DKW21]. Notably, while this DNF-SS construction is established, its "programmability" property has never been formally proved to the best of our knowledge. Our work provides this formal proof, which may be of independent interest in other cryptographic contexts. While generalizing our framework beyond DNFs remains an interesting open question, it faces strong theoretical barriers, primarily because constructing an LSSS with short reconstruction coefficients and short shares for more expressive policies is notoriously tricky, as demonstrated by Ball, Çakan, and Malkin [BÇM21].

# 3 Preliminaries

**Notation.** By $\mathsf{PPT}$ we denote probabilistic polynomial-time. We denote the security parameter by $\lambda$ and the set of positive integers by $\mathbb{N}$. For any $a, b \in \{0\} \cup \mathbb{N}$, $a \leq b$, we denote by $[a, b]$, the set of all integers from $a$ to $b$ including $a$ and $b$. In other words, $[a, b] = \{a, \ldots, b\}$. We denote by $[n] := [1, n]$. We denote by $x \leftarrow \mathcal{X}$, the process of sampling an element $x$ from the set $\mathcal{X}$, with uniform probability. Similarly, for any $\mathsf{PPT}$ algorithm $\mathcal{A}$, $x \leftarrow \mathcal{A}(y)$ denotes the process of sampling $x$ from the output distribution of $\mathcal{A}$ when run on $y$ with randomly chosen coins. By $\mathsf{negl}(\cdot)$, we denote negligible functions. By $\mathsf{poly}(\cdot)$, we denote positive polynomials. All of our logarithms log are in base 2. We denote the power set $A$ by $2^A$. Consider two efficiently samplable probability distributions $\mathcal{X} = \{\mathcal{X}_\lambda\}_{\lambda\in\mathbb{N}}$ and $\mathcal{Y} = \{\mathcal{Y}_\lambda\}_{\lambda\in\mathbb{N}}$ over support $\Omega$. We say $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable if for any non-uniform $\mathsf{PPT}$ distinguisher $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$, and for large enough $\lambda \in \mathbb{N}$,

$$\left| \Pr_{\alpha\leftarrow\mathcal{X}_\lambda} \left[1 \leftarrow \mathcal{D}(1^\lambda, \alpha)\right] - \Pr_{\alpha\leftarrow\mathcal{Y}_\lambda} \left[1 \leftarrow \mathcal{D}(1^\lambda, \alpha)\right] \right| \leq \mathsf{negl}(\lambda)$$

We write $\mathcal{X} \approx_c \mathcal{Y}$ to denote that distributions $\mathcal{X}$ and $\mathcal{Y}$ computationally indistinguishable. Similarly, by $\approx_s$, we denote statistical indistinguishability. Concretely, $\mathcal{X} \approx_s \mathcal{Y}$ if for large enough $\lambda \in \mathbb{N}$, $\mathsf{SD}(\mathcal{X}_\lambda, \mathcal{Y}_\lambda) = \frac{1}{2} \sum_{\alpha\in\Omega} |\Pr[\alpha \leftarrow \mathcal{X}_\lambda] - \Pr[\alpha \leftarrow \mathcal{Y}_\lambda]| \leq \mathsf{negl}(\lambda)$.

## 3.1 Lattice Preliminaries

For prime power $q \in \mathbb{N}$, we denote the finite field with $q$ elements as $\mathbb{Z}_q$. Following popular convention, we denote column vectors by lowercase bold letters, e.g., $\mathbf{x} \in \mathbb{Z}_q^n$ and denote matrices by uppercase bold letters, e.g., $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$. We use $x_i$ to denote the $i$-th element in $\mathbf{x}$ for $i \in [n]$ and $a_{i,j}$ to denote the element in the $i$-th row and $j$-th column of matrix $\mathbf{A}$ for $i \in [n], j \in [m]$. We sometimes alternately write $\mathbf{x}[i] := x_i$ and $\mathbf{A}[i, j] := a_{i,j}$. We define $\mathbf{1}_n$ to be a vector of all ones, $\mathbf{0}_n$ to be a vector of all zeros, and $\mathbb{1}_{n,j}$ to denote the $n$-dimensional canonical basis vector that has 1 in the $j$-th position, i.e., $\mathbf{1}_n = [1\ 1\ \ldots\ 1]^\top$ and $\mathbb{1}_{n,j} = [\ \ldots\ 1\ \ldots]^\top$. We frequently drop $n$ from subscript in these vectors when dimension is clear. We denote the $n \times n$ identity matrix by $\mathbf{I}_n$ and drop the subscript when dimension is clear. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathsf{vec}(\mathbf{A}) \in \mathbb{Z}_q^{nm}$ denoted the vector obtained by concatenating the columns of $\mathbf{A}$. For any vector $\mathbf{a} \in \mathbb{Z}_q^n$ or matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we use $\|\mathbf{a}\|$ and $\|\mathbf{A}\|$ to denote the maximum of all the entries in $\mathbf{a}$ and $\mathbf{A}$ respectively. We use $\mathbf{A} \otimes \mathbf{B}$ to denote the Kronecker product between matrices $\mathbf{A}, \mathbf{B}$.

**Discrete Gaussian.** We write $\mathcal{D}_{\mathbb{Z},\sigma}$ to denote the discrete gaussian distribution with width parameter $\sigma > 0$. We state the following important lemma about the maximum norm of a value sampled from discrete gaussian distribution.

**Lemma 3.1** ([Lyu12, MP12])**.** For any $k > 0$, $\Pr[|x| > k\sigma : x \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}] \leq 2\exp\left(\frac{-k^2}{2}\right)$.

**Gadget Matrix.** For positive integers $n, m, q$, and $m = \Omega(n \log q)$, we write $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ to denote the gadget matrix [MP12]. It is defined as $\mathbf{G} = (\mathbf{I}_n \otimes \mathbf{g}^\top)$, $\mathbf{g} = [1\ 2\ 2^2\ \ldots\ 2^{\lfloor \log q \rfloor}\ 0\ \ldots\ 0]^\top \in \mathbb{Z}_q^m$.

**Leftover Hash Lemma.** We recall the (generalized) leftover hash lemma adapted from [ILL89, DRS04, GPV08, ABB10].

**Lemma 3.2** (Generalized Leftover Hash Lemma)**.** Let $n, m, q \in \mathbb{N}$ and $q$ is an odd prime.

1. $\{(\mathbf{B}, \mathbf{BR}, \mathbf{e}^\top\mathbf{R})\} \approx_s \{(\mathbf{B}, \mathbf{A}, \mathbf{e}^\top\mathbf{R})\}$ where $m = \Omega(n \log q)$, $\mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{R} \leftarrow \{\pm 1\}^{m\times\ell}$(or $\mathbf{R} \leftarrow \{0,1\}^{m\times\ell}$), $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times\ell}$, $\ell = \mathsf{poly}(n)$ and $\mathbf{e}^\top \in \mathbb{Z}_q^m$ is any fixed vector.
2. $\{(\mathbf{B}, \mathbf{BR})\} \approx_s \{(\mathbf{B}, \mathbf{A})\}$ where $m = \Omega(n \log q)$, $\mathbf{B} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{R} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m\times\ell}, \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times\ell}$, $\ell = \mathsf{poly}(n)$ and $\sigma = \omega(\sqrt{\log m})$.

**Gadget Trapdoors.** Let $(n, m, q)$ be LWE parameters such that $m \geq 3n \log q$. Then, there exists two polynomial time algorithms defined as follows.

$\mathsf{TrapGen}(1^n, 1^m, q) \to (\mathbf{A}, \mathbf{T_A})$**.** The randomized trapdoor generation algorithm takes as input lattice dimension $n$, number of columns $m$, modulus $q$, outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and its trapdoor $\mathbf{T_A}$.

$\mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{y}, \sigma) \to \mathbf{x}$**.** The randomized pre-image sampling algorithm takes as input $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, its trapdoor $\mathbf{T_A}$, a vector $\mathbf{y} \in \mathbb{Z}_q^n$, gaussian width parameter $\sigma$, and outputs a vector $\mathbf{x} \in \mathbb{Z}_q^m$.

These algorithms satisfy the following properties.

**Matrix Distribution.** If $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, then $\|\mathbf{T_A}\| = 1$, $\mathbf{AT_A} = \mathbf{G}$, and

$$\{\mathbf{A} : (\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)\} \approx_s \{\mathbf{A} : \mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}\}$$

**Sampling Correctness.** For all $\sigma > 0$, any $\mathbf{y} \in \mathbb{Z}_q^n$, $\mathbf{Ax} = \mathbf{y}$ where $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ and $\mathbf{x} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{y}, \sigma)$.

**Pre-image Distribution.** Let $(\mathbf{A}, \mathbf{T_A})$ be such that $\mathbf{AT_A} = \mathbf{G}$. Then, for any $\sigma \geq \|\mathbf{T_A}\| m \log n$, any $\mathbf{y} \in \mathbb{Z}_q^n$,

$$\{\mathbf{x} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{y}, \sigma)\} \approx_s \{\mathbf{x} : \mathbf{A}_\sigma^{-1}(\mathbf{y})\}$$

Here, $\mathbf{A}_\sigma^{-1}(\mathbf{y})$ denotes the (possibly inefficient) process of sampling a discrete gaussian vector with width parameter $\sigma$ from the coset lattice $\Lambda_q^{\mathbf{y}}(\mathbf{A}) = \{\mathbf{x} : \mathbf{Ax} = \mathbf{y} \mod q, \mathbf{x} \in \mathbb{Z}^m\}$.

**Remark 3.3.** SamplePre can be naturally extended to sample pre-images for matrices by sampling a pre-image vector for each column independently. We will use this version in our work. Sampling correctness and pre-image distribution properties can be naturally extended to matrices as well.

**Lemma 3.4** ([MP12])**.** Let $n, m, q$ be lattice parameters such that $m \geq 2n \log q$. For all but $\mathsf{negl}(n)$ fraction of matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\sigma \geq \log m$, any vector $\mathbf{y}$ in the column space of $\mathbf{A}$, $\Pr[\|\mathbf{x}\| > \sqrt{m}\sigma : \mathbf{x} \leftarrow \mathbf{A}_\sigma^{-1}(\mathbf{y})] \leq \mathsf{negl}(m)$.

**Assumption 3.5** ([Reg05])**.** The LWE assumption with parameters $(n, m, q, \chi)$ states that

$$\{(\mathbf{A}, \mathbf{s}^\top\mathbf{A} + \mathbf{e}^\top)\} \approx_c \{(\mathbf{A}, \mathbf{c}^\top)\}$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\chi}^m, \mathbf{c} \leftarrow \mathbb{Z}_q^m$.

## 3.2 $\ell$-Decomposed LWE Assumption

We provide the $\ell$-decomposed LWE assumption which was originally proposed in [AMR25]. Our formulation is adapted from [Wee25, WW25].

**Assumption 3.6** ([AMR25], adapted from [Wee25, WW25])**.** Let $(n, m, q, \chi)$ be LWE parameters. Then, the $(\ell, \sigma)$-decomposed LWE assumption states that

$$\{(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{s}^\top\mathbf{B} + \mathbf{e}^\top)\} \approx_c \{(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{c}^\top)\}$$

where $\mathbf{W} \leftarrow \mathbb{Z}_q^{n\times\ell m}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m\times\ell m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\chi}^{\ell^2 m}, \mathbf{c} \leftarrow \mathbb{Z}_q^{\ell^2 m}$ and $\mathbf{B} = \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_\ell)^\top \otimes \mathbf{G} \in \mathbb{Z}_q^{n\times\ell^2 m}$. When $\sigma = \mathsf{poly}(\lambda, \ell, m)$ we simply call this as $\ell$-decomposed LWE assumption.

**Leftover Hash Lemma for Structured B.** We remark that $\mathbf{B}$ defined in above assumption is a structured matrix uniquely defined by $\mathbf{W}, \mathbf{D}$. Thus, it might not be immediately clear that Lemma 3.2 works with $\mathbf{B}$. We provide the following lemmas which show that leftover hash lemma holds even with respect to $\mathbf{B}$.

**Lemma 3.7.** Lemma 3.2, Item 1 holds with respect to $\mathbf{B}$ even given $(\mathbf{W}, \mathbf{D})$.

*Proof.* Full proof is provided in Appendix A. □

**Trapdoor for B.** We will show an alternate way to derive $(\mathbf{W}, \mathbf{D}, \mathbf{B})$ of the $(\ell, \sigma)$-decomposed LWE assumption assumption such that there is a way to construct $\mathbf{T_B}$ such that $\mathbf{BT_B} = \mathbf{G}, \|\mathbf{T_B}\| = 1$. The procedure is described below.

1. Sample $(\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, sample $[\mathbf{W}_2 \mid \ldots \mid \mathbf{W}_\ell] \leftarrow \mathbb{Z}_q^{n \times (\ell-1)m}$. Set $\mathbf{W} := [\mathbf{W}_1 \mid \mathbf{W}_2 \mid \ldots \mid \mathbf{W}_\ell]$.
2. Sample $\mathbf{D}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times m}, [\mathbf{D}_3 \mid \ldots \mid \mathbf{D}_\ell] \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times (\ell-2)m}$. Sample $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ Sample $\mathbf{D}_2 \leftarrow \mathsf{SamplePre}(\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}, \mathbf{A}, \sigma)$. Set $\mathbf{D} = [\mathbf{D}_1 \mid \mathbf{D}_2 \mid \mathbf{D}_3 \mid \ldots \mid \mathbf{D}_\ell]$.
3. Set $\mathbf{B} := \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_\ell)^\top \otimes \mathbf{G}$, $\mathbf{T_B} = \mathbb{1}_{\ell^2,2} \otimes \mathbf{T_A}$. Output $(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B})$.

It is straightforward to see that $\|\mathbf{T_B}\| = \|\mathbf{T_A}\| = 1$. Now, $\mathbf{BT_B} = [\mathbf{W}_1\mathbf{D}_1 + \mathbf{G} \mid \mathbf{W}_1\mathbf{D}_2 \mid \ldots \mid \mathbf{W}_\ell\mathbf{D}_\ell + \mathbf{G}][\mathbf{0} \mid \mathbf{T_A}^\top \mid \mathbf{0} \mid \ldots \mid \mathbf{0}]^\top = \mathbf{W}_1\mathbf{D}_2\mathbf{T_A} = \mathbf{AT_A} = \mathbf{G}$. We also remark that $(\mathbf{W}, \mathbf{D}, \mathbf{B})$ given out by the procedure is statistically close to uniformly random values. Proof is provided in Appendix A.

## 3.3 Matrix Commitment Scheme

We provide the definition of a matrix commitment scheme (MCOM) from decomposed LWE adapted from [Wee25, Appendix C].

**Syntax.** An MCOM scheme consists of the following polynomial time algorithms.

$\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma) \to (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B})$**.** The randomized alternative setup algorithm takes as input LWE parameters $(n, m, q, \sigma)$ where $\sigma = \mathsf{poly}(\lambda, m)$ and outputs $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times 2m^3}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times 2m^3}$, and $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G}, \mathbf{T_B}$ such that $\mathbf{BT_B} = \mathbf{G}$. This is essentially the trapdoor sampling for $\mathbf{B}$ procedure from above.

$\mathsf{Setup}(1^n, 1^m, q, \sigma) \to (\mathbf{W}, \mathbf{D}, \mathbf{B})$**.** The randomized *public-coin* setup algorithm takes as input LWE parameters $(n, m, q, \sigma)$ where $\sigma = \mathsf{poly}(\lambda, m)$ and outputs $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times 2m^3}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times 2m^3}$, and $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G}$. The following algorithms take $(\mathbf{W}, \mathbf{D}, \mathbf{B})$ implicitly.

$\mathsf{Com}(\mathbf{M}) \to \mathbf{C}$**.** The deterministic commitment algorithm takes as input matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$ and outputs commitment matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$.

$\mathsf{Verify}(1^N) \to \mathbf{V}_N$**.** The deterministic verification algorithm takes as input column length $N$ and outputs verification matrix $\mathbf{V}_N \in \mathbb{Z}_q^{m \times N}$.

$\mathsf{Open}(\mathbf{M}) \to \mathbf{Z}$**.** The deterministic opening algorithm takes as input matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$ and outputs opening matrix $\mathbf{Z} \in \mathbb{Z}_q^{4m^5 \times N}$.

**Definition 3.8** (MCOM)**.** A MCOM scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Verify}, \mathsf{Open})$ satisfies the following properties.

**Correctness.** For any $n, q \in \mathbb{N}, m = \Omega(n \log q), \sigma = \mathsf{poly}(n, m), N = \mathsf{poly}(\lambda)$, any $(\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \mathsf{Setup}(1^n, 1^m, q, \sigma)$, any matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$, $\mathbf{CV}_N = \mathbf{M} - \mathbf{BZ}$ where $\mathbf{C} = \mathsf{Com}(\mathbf{M}), \mathbf{V}_N = \mathsf{Verify}(1^N), \mathbf{Z} = \mathsf{Open}(\mathbf{M})$.

**Compactness.** For any $n, q \in \mathbb{N}, m = \Omega(n \log q), \sigma = \mathsf{poly}(n, m), N = \mathsf{poly}(\lambda)$, any $(\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \mathsf{Setup}(1^n, 1^m, q, \sigma)$, any matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$, $\|\mathbf{V}_N\| \le O(\|\mathbf{D}\| m^4 \log q)$ and $\|\mathbf{Z}\| \le O(\|\mathbf{D}\| m^7 \log q \log N)$ where $\mathbf{V}_N = \mathsf{Verify}(1^N)$ and $\mathbf{Z} = \mathsf{Open}(\mathbf{M})$.

**Mode Indistinguishability.** For any $n, q \in \mathbb{N}, m = \Omega(n \log q), \sigma = \mathsf{poly}(n, m)$,

$$\left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}) \ : \ \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \\ \mathsf{Setup}(1^n, 1^m, q, \sigma) \end{array} \right\} \approx_s \left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}) \ : \ \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \\ \widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma) \end{array} \right\}$$

This property follows from Lemma A.6, matrix distribution, and pre-image distribution properties of gadget trapdoors. Full proof is in Appendix A.

### 3.4 Explainable Re-randomizer

We provide the definition of an explainable re-randomizer (ER) scheme from [CHW25, WW25]. Essentially, an ER scheme lets us sample a *random* commitment to a $\mathbf{0} \in \mathbb{Z}_q^{n \times N}$ of the matrix commitment scheme from Section 3.3.

**Syntax.** An ER scheme consists of the following polynomial time algorithms.

$\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma) \to (\mathbf{W}, \mathbf{D}, \mathbf{B})$**.** Same as $\widetilde{\mathsf{Setup}}$ from Section 3.3.

$\mathsf{Setup}(1^n, 1^m, q, \sigma) \to (\mathbf{W}, \mathbf{D}, \mathbf{B})$**.** Same as $\mathsf{Setup}$ from Section 3.3. The following algorithms take $(\mathbf{W}, \mathbf{D}, \mathbf{B})$ implicitly.

$\mathsf{Sample}(1^\lambda, 1^N, \sigma) \to (\mathbf{C}, \mathbf{Z})$**.** The randomized sampling algorithm takes as input security parameter $\lambda$, column length $N$, width parameter $\sigma > 0$, and outputs matrices $\mathbf{C} \in \mathbb{Z}_q^{n \times m}, \mathbf{Z} \in \mathbb{Z}_q^{m \times N}$.

$\mathsf{Explain}(1^\lambda, 1^T, \mathbf{C}, \mathbf{Z}, \sigma) \to r'$**.** The randomized explanation algorithm takes as input $T = \mathsf{poly}(\lambda)$, matrices $\mathbf{C} \in \mathbb{Z}_q^{n \times m}, \mathbf{Z} \in \mathbb{Z}_q^{m \times N}$, width parameter $\sigma > 0$, and outputs $r' \in \{0,1\}^\rho$.

**Definition 3.9** (ER)**.** An ER scheme $(\mathsf{Setup}, \mathsf{Sample}, \mathsf{Explain})$ is said to be an ER scheme if it satisfies the following properties.

**Correctness.** For any $\lambda \in \mathbb{N}, N \in \mathbb{N}, (\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \mathsf{Setup}(1^n, 1^m, q, \sigma)$, there exists a polynomial $\rho = \mathsf{poly}(\lambda, N)$ such that $\forall\ n \geq \lambda, m \geq 2n\lceil \log q \rceil, \sigma > 0$, $\mathbf{C}\mathbf{V}_N = -\mathbf{B}\mathbf{Z}$ where $(\mathbf{C}, \mathbf{Z}) \leftarrow \mathsf{Sample}(1^\lambda, 1^N, \sigma)$, $\mathbf{V}_N = \mathsf{MCOM.Verify}(1^N)$.

**Randomness.** For any $\lambda \in \mathbb{N}, N \in \mathbb{N}, (\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \mathsf{Setup}(1^n, 1^m, q, \sigma)$, there exists a polynomial $\rho = \mathsf{poly}(\lambda, N)$ such that $\forall\ n \geq \lambda, m \geq 2n\lceil \log q \rceil, \|\mathbf{T}\|\, O(m^{12} N^3) < \sigma < 2^\lambda$,

$$\left\{ (\mathbf{C}, \mathbf{Z}) \ : \ \begin{array}{l} r \leftarrow \{0,1\}^\rho, \\ (\mathbf{C}, \mathbf{Z}) = \mathsf{Sample}(1^\lambda, 1^N, \sigma; r) \end{array} \right\} \approx_s \left\{ (\mathbf{C}, \mathbf{Z}) \ : \ \begin{array}{l} \mathbf{C} \leftarrow \mathbb{Z}_q^{n \times m}, \\ \mathbf{V}_N = \mathsf{MCOM.Verify}(1^N), \\ \mathbf{Z} \leftarrow \mathbf{B}_\sigma^{-1}(-\mathbf{C}\mathbf{V}_N) \end{array} \right\}$$

**Explainability.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $N \in \mathbb{N}, T = \mathsf{poly}(\lambda)$, $\|\mathbf{T}\|\, O(m^{12} N^3) < \sigma < 2^\lambda$,

$$\mathsf{SD}(\mathcal{D}_0, \mathcal{D}_1) \leq \frac{1}{T} + \mathsf{negl}(\lambda) \text{ where,}$$

$$\mathcal{D}_0 = \left\{ (\mathbf{C}, \mathbf{Z}, r) \ : \ \begin{array}{l} r \leftarrow \{0,1\}^\rho, \\ (\mathbf{C}, \mathbf{Z}) = \mathsf{Sample}(1^\lambda, 1^N, \sigma; r) \end{array} \right\},$$

$$\mathcal{D}_1 = \left\{ (\mathbf{C}, \mathbf{Z}, r') \ : \ \begin{array}{l} (\mathbf{C}, \mathbf{Z}) \leftarrow \mathsf{Sample}(1^\lambda, 1^N, \sigma), \\ r' \leftarrow \mathsf{Explain}(1^\lambda, 1^T, \mathbf{C}, \mathbf{Z}, \sigma) \end{array} \right\}$$

**Mode Indistinguishability.** Same as Definition 3.8.

# 4 Secret Sharing Schemes

In this section, we review preliminaries on secret sharing and then define additional properties that we require. We also explain why standard constructions already satisfy these properties for the access-structure families we consider.

## 4.1 Secret Sharing Preliminaries

Here, we recall basic notation and definitions for secret sharing. Some definitions are adapted from [LW11, BGG+18].

### 4.1.1 Access Structures

In this work, for simplicity, we identify each party $P_i$ with its index $i \in [N]$. For a party set $S \subseteq [N]$, let $\mathbf{x}_S \in \{0,1\}^N$ be its characteristic vector, i.e., $(\mathbf{x}_S)_i = 1$ if $i \in S$ and $(\mathbf{x}_S)_i = 0$ otherwise.

**Definition 4.1** (Monotone Access Structure)**.** Let $\{1, \ldots, N\}$ be a set of parties. A collection $\mathbb{A} \subseteq 2^{[N]}$ is *monotone* if for any sets $B, C$ satisfying $B \in \mathbb{A}$ and $B \subseteq C \subseteq [N]$, we have $C \in \mathbb{A}$. A *(monotone) access structure* on $[N]$ is a monotone collection $\mathbb{A} \subseteq 2^{[N]} \setminus \emptyset$. Sets in $\mathbb{A}$ are called *valid* (or *authorized*) and sets in $2^{[N]} \setminus \mathbb{A}$ are called *invalid* (or *unauthorized*).

**Definition 4.2** (Efficient Access Structure)**.** Let $[N]$ be a set of parties and let $\mathbb{A}$ be a monotone access structure on $[N]$. We say that $\mathbb{A}$ is *efficient* if there exists a polynomial-size *monotone* Boolean circuit $f_{\mathbb{A}} : \{0,1\}^N \to \{0,1\}$ such that for all $S \subseteq [N]$, $f_{\mathbb{A}}(\mathbf{x}_S) = 1$ if and only if $S \in \mathbb{A}$.

In this work, we only consider efficient access structures.

**Definition 4.3** (Class of Monotone Access Structures)**.** Let $[N]$ be a set of parties. A *class* of monotone access structures is a collection $\mathbb{S} \subseteq 2^{2^{[N]}}$ of monotone access structures on $[N]$.

### 4.1.2 Secret Sharing

We now define secret sharing for access structures [CK93, BC02, BR07].

**Syntax.** A secret sharing scheme for parties $[N]$, secret space $\mathcal{K}$, and class of access structures $\mathbb{S}$ consists of polynomial-time algorithms:

$\mathsf{Share}(k, \mathbb{A}) \to (s_1, \ldots, s_N)$**.** On input a secret $k \in \mathcal{K}$ and an access structure $\mathbb{A} \in \mathbb{S}$, outputs shares $(s_1, \ldots, s_N)$.

$\mathsf{Combine}(\mathbb{A}, (s_i)_{i \in S}) \to k$**.** On input $\mathbb{A} \in \mathbb{S}$ and shares held by a set $S \subseteq [N]$, outputs a value $k \in \mathcal{K}$.

**Definition 4.4** (Secret Sharing)**.** A secret sharing scheme $\mathsf{SS} = (\mathsf{Share}, \mathsf{Combine})$ for parties $[N]$, secret space $\mathcal{K}$, and class $\mathbb{S}$ satisfies:

**Correctness.** For every $\mathbb{A} \in \mathbb{S}$, every $S \in \mathbb{A}$, every $k \in \mathcal{K}$, and every $(s_1, \ldots, s_N) \leftarrow \mathsf{Share}(k, \mathbb{A})$, we have

$$\mathsf{Combine}\big(\mathbb{A}, (s_i)_{i \in S}\big) = k.$$

**Privacy.** For every $\mathbb{A} \in \mathbb{S}$, every $S \notin \mathbb{A}$, and every $k_0, k_1 \in \mathcal{K}$, if $(s_{b,1}, \ldots, s_{b,N}) \leftarrow \mathsf{Share}(k_b, \mathbb{A})$ for $b \in \{0,1\}$, then the distributions $(s_{0,i})_{i \in S}$ and $(s_{1,i})_{i \in S}$ are identical.

### 4.1.3 Linear Secret Sharing

**Definition 4.5** (Linear Secret Sharing Scheme (LSSS))**.** A secret sharing scheme $\mathsf{SS} = (\mathsf{Share}, \mathsf{Combine})$ for parties $[N]$, secret space $\mathcal{K} = \mathbb{Z}_q$ (odd prime $q$), and class $\mathbb{S}$ is a *linear secret sharing scheme* if for every $\mathbb{A} \in \mathbb{S}$ there exist integers $d = d(\mathbb{A})$ and $L^* = L^*(\mathbb{A})$, a matrix $\mathbf{M} \in \mathbb{Z}_q^{L^* \times d}$, and a partition $T_1, \ldots, T_N$ of $[L^*]$ (where $T_i$ is associated with party $i$), such that:

$\mathsf{Share}(k, \mathbb{A})$**.** The sharing algorithm samples $r_2, \ldots, r_d \leftarrow \mathbb{Z}_q$ uniformly, sets $\mathbf{k} := (k, r_2, \ldots, r_d)^\top \in \mathbb{Z}_q^d$, computes

$$\mathbf{w} := \mathbf{M} \cdot \mathbf{k} \in \mathbb{Z}_q^{L^*},$$

and gives party $i$ the share $s_i := (w_j)_{j \in T_i}$.

$\mathsf{Combine}(\mathbb{A}, (s_i)_{i \in S})$**.** For every valid set $S \in \mathbb{A}$, letting $T_S := \bigcup_{i \in S} T_i$, we have

$$(1, 0, \ldots, 0) \in \operatorname{span}\big((\mathbf{M}[j])_{j \in T_S}\big) \subseteq \mathbb{Z}_q^d,$$

where $\mathbf{M}[j]$ denotes the $j$-th row of $\mathbf{M}$. Moreover, the set $S$ can efficiently find coefficients $(c_j)_{j \in T_S} \subseteq \mathbb{Z}_q$ such that

$$\sum_{j \in T_S} c_j \cdot \mathbf{M}[j] = (1, 0, \ldots, 0),$$

and reconstruct the secret as

$$k = \sum_{j \in T_S} c_j \cdot w_j.$$

We call $(c_j)_{j \in T_S}$ the *recovery coefficients*.

**Definition 4.6** (Binary Linear Secret Sharing Scheme)**.** A linear secret sharing scheme $\mathsf{SS} = (\mathsf{Share}, \mathsf{Combine})$ with secret space $\mathcal{K} = \mathbb{Z}_q$ is a *binary* LSSS if for every $\mathbb{A} \in \mathbb{S}$ and every valid set $S \in \mathbb{A}$, there exists a subset $T \subseteq T_S := \bigcup_{i \in S} T_i$ such that

$$k = \sum_{j \in T} w_j,$$

where $\mathbf{w} = \mathbf{M} \cdot (k, r_2, \ldots, r_d)^\top$ is the share vector output by $\mathsf{Share}(k, \mathbb{A})$. Equivalently, $S$ admits recovery coefficients in $\{0, 1\}$.

**Definition 4.7** (Maximal Invalid and Minimal Valid Sets)**.** Let $[N]$ be a set of parties and let $\mathbb{A}$ be a monotone access structure on $[N]$.

- A set of parties $S \subseteq [N]$ is *maximal invalid* if $S \notin \mathbb{A}$ but for every $i \in [N] \setminus S$, we have $S \cup \{i\} \in \mathbb{A}$.
- A set of parties $S \subseteq [N]$ is *minimal valid* if $S \in \mathbb{A}$ and for every $S' \subsetneq S$, we have $S' \notin \mathbb{A}$.

Now fix an LSSS for $\mathbb{A}$ with share matrix $\mathbf{M} \in \mathbb{Z}_q^{L^* \times d}$. For an index set $T \subseteq [L^*]$, we say that $T$ is a *valid share set* if $(1, 0, \ldots, 0) \in \operatorname{span}((\mathbf{M}[j])_{j \in T})$, and *invalid* otherwise.

- A set $T \subseteq [L^*]$ is a *maximal invalid share set* if $T$ is invalid, but for every $i \in [L^*] \setminus T$, the set $T \cup \{i\}$ is valid.
- A set $T \subseteq [L^*]$ is a *minimal valid share set* if $T$ is valid, but for every $T' \subsetneq T$, $T'$ is invalid.

**Remark 4.8** ($\mathrm{LSSS}_N$ Access Structures)**.** For convenience, we refer to $\mathrm{LSSS}_N$ (resp. $\{0,1\}$-$\mathrm{LSSS}_N$) as the class of access structures on $N$ parties that admit an LSSS (resp. a binary LSSS).

**Remark 4.9** (Equal fan-out in the input layer)**.** For simplicity, we assume $|T_i| = L$ for all $i \in [N]$, so $L^* = N \cdot L$. Equivalently, we index shares by $[N \cdot L]$ and view $T_i$ as the $i$-th block of length $L$. We denote the class of such access structures by $\mathbb{S}_L$.

## 4.2 Programmable Binary Linear Secret Sharing Schemes

**Syntax.** A programmable binary linear secret sharing scheme for parties $[N]$, secret space $\mathcal{K}$, and access-structure class $\mathbb{S}_L$, in addition to $(\mathsf{Share}, \mathsf{Combine})$, consists of polynomial-time algorithms:

$\mathsf{ProgShare}_1(\mathbb{A}, T, (\tilde{s}_j)_{j\in T}, j^*, \tilde{s}_{j^*}) \to (k, s_1, \ldots, s_N)$**.** The deterministic programmed sharing algorithm in Mode 1 takes as input an access structure $\mathbb{A} \in \mathbb{S}_L$, a maximal invalid share set $T$ (with respect to the LSSS for $\mathbb{A}$), a collection of programmed shares $(\tilde{s}_j)_{j\in T}$, an index $j^* \notin T$ and a value $\tilde{s}_{j^*}$. It outputs a secret $k \in \mathcal{K}$ and party shares $(s_1, \ldots, s_N)$.

$\mathsf{ProgShare}_2(\mathbb{A}, T, (\tilde{s}_j)_{j\in T}, k) \to (k, s_1, \ldots, s_N)$**.** The deterministic programmed sharing algorithm in Mode 2 takes as input $\mathbb{A} \in \mathbb{S}_L$, a maximal invalid share set $T$, a collection of programmed shares $(\tilde{s}_j)_{j\in T}$, and a secret $k \in \mathcal{K}$. It outputs the same secret $k$ and party shares $(s_1, \ldots, s_N)$.

**Definition 4.10** (Programmability)**.** A binary linear secret sharing scheme $\mathsf{SS} = (\mathsf{Share}, \mathsf{Combine})$ for parties $[N]$, secret space $\mathcal{K} = \mathbb{Z}_q$ (for odd prime $q$), and class $\mathbb{S}_L$ is *programmable* if it satisfies:

**Correctness and consistency.** For every $\mathbb{A} \in \mathbb{S}_L$, every $A \in \mathbb{A}$, every maximal invalid share sets $T_0^*, T_1^* \subseteq [N \cdot L]$ (with respect to the LSSS for $\mathbb{A}$), every $j^* \in [N \cdot L] \setminus T_0^*$, every choice of inputs $\tilde{s}_{b,j} \in \mathbb{Z}_q$ for $b \in \{0,1\}$ and $j \in T_b^*$, every $\tilde{s}_{0,j^*} \in \mathbb{Z}_q$, and every $k \in \mathbb{Z}_q$, let

$$(k_0, s_{0,1}, \ldots, s_{0,N}) := \mathsf{ProgShare}_1(\mathbb{A}, T_0^*, (\tilde{s}_{0,j})_{j\in T_0^*}, j^*, \tilde{s}_{0,j^*})$$

and

$$(k_1, s_{1,1}, \ldots, s_{1,N}) := \mathsf{ProgShare}_2(\mathbb{A}, T_1^*, (\tilde{s}_{1,j})_{j\in T_1^*}, k).$$

Then for each $b \in \{0,1\}$,

$$\mathsf{Combine}\big(\mathbb{A}, (s_{b,i})_{i\in A}\big) = k_b.$$

Moreover, programmed share values are consistent with output party shares: if $j \in T_0^* \cup \{j^*\}$ (resp. $j \in T_1^*$) is the $\ell$-th element of $T_i$ (with a fixed ordering), then $\tilde{s}_{0,j}$ (resp. $\tilde{s}_{1,j}$) equals the $\ell$-th coordinate inside $s_{0,i}$ (resp. $s_{1,i}$). Additionally, $k_1 = k$.

**Mode indistinguishability.** For every $\mathbb{A} \in \mathbb{S}_L$, every maximal invalid share sets $T_0^*, T_1^* \subseteq [N \cdot L]$, and every $j^* \in [N \cdot L] \setminus T_0^*$, the output distributions of the following two experiments are identically distributed:

- $\mathsf{Exp}_1$: sample $(\tilde{s}_{0,j})_{j\in T_0^*} \leftarrow (\mathbb{Z}_q)^{|T_0^*|}$ and $\tilde{s}_{0,j^*} \leftarrow \mathbb{Z}_q$ uniformly, and output
$$(k_0, s_{0,1}, \ldots, s_{0,N}) = \mathsf{ProgShare}_1(\mathbb{A}, T_0^*, (\tilde{s}_{0,j})_{j\in T_0^*}, j^*, \tilde{s}_{0,j^*}).$$
- $\mathsf{Exp}_2$: sample $(\tilde{s}_{1,j})_{j\in T_1^*} \leftarrow (\mathbb{Z}_q)^{|T_1^*|}$ and $k \leftarrow \mathbb{Z}_q$ uniformly, and output
$$(k_1, s_{1,1}, \ldots, s_{1,N}) = \mathsf{ProgShare}_2(\mathbb{A}, T_1^*, (\tilde{s}_{1,j})_{j\in T_1^*}, k).$$

### 4.2.1 Programmable Binary LSSS for DNF Formulas

Here, we justify that Definition 4.10 is satisfied by standard binary LSSS representations of DNF access structures. Concretely, we rely on the Lewko–Waters transformation that converts a monotone Boolean formula (and in particular a DNF formula) into an LSSS matrix representation, introducing one new column per AND gate in the underlying formula. We will use two properties of the resulting LSSS: (i) *binary recovery* (recovery coefficients in $\{0,1\}$), and (ii) a *linear-independence* property for unauthorized sets (defined below), which is known to hold for DNF policies via the Lewko–Waters representation (see, e.g., discussions/uses in [DKW21] and the observations attributed to [ABN+22]).

**A linear-independence property for unauthorized sets.** Fix an LSSS for $\mathbb{A}$ with share matrix $\mathbf{M} \in \mathbb{Z}_q^{L^* \times d}$ and partition $T_1, \ldots, T_N$ of $[L^*]$. Write $\mathbb{1}_1 = (1, 0, \ldots, 0) \in \mathbb{Z}_q^d$.

We assume w.l.o.g. that $\mathbf{M}$ has full column rank, i.e., $\mathrm{rank}(\mathbf{M}) = d$ (otherwise, replace $\mathbf{M}$ by a full-rank column basis that contains the secret direction $\mathbb{1}_1$, and rewrite $\mathbf{k}$ in that basis).

**Definition 4.11** (Linear-independence for unauthorized share sets)**.** We say that $(\mathbf{M}, (T_i)_i)$ has the *LI-unauthorized* property if for every *invalid share set* $T \subseteq [L^*]$ (i.e., $\mathbb{1}_1 \notin \mathrm{span}((\mathbf{M}[j])_{j\in T})$), the rows $(\mathbf{M}[j])_{j\in T}$ are linearly independent over $\mathbb{Z}_q$.

This property is *not* true for arbitrary monotone formulas, but it is known to hold for DNF policies under the Lewko–Waters LSSS representation; see [DKW21], which notes that for DNF formulas the (Lewko–Waters-style) construction yields linearly independent rows for unauthorized sets, and also uses the observation (attributed to [ABN+22]) that one can ensure the property by computing the DNF's monotone LSSS via the Lewko–Waters transformation.

**Rank consequences.** The LI-unauthorized property immediately yields a clean "codimension-one" structure for maximal invalid share sets.

**Lemma 4.12** (Maximal invalid sets have rank $d-1$)**.** Assume $(\mathbf{M}, (T_i)_i)$ satisfies Definition 4.11 and that $\mathrm{rank}(\mathbf{M}) = d$. Let $T \subseteq [L^*]$ be a *maximal invalid share set*. Then:

1. $\mathrm{rank}((\mathbf{M}[j])_{j\in T}) = |T| = d-1$.
2. For every $j^* \in [L^*] \setminus T$, the $d \times d$ matrix

$$\mathbf{A}_{T,j^*} := \begin{pmatrix} \mathbf{M}[j_1] \\ \vdots \\ \mathbf{M}[j_{d-1}] \\ \mathbf{M}[j^*] \end{pmatrix} \quad \text{(rows indexed by } T \text{ in any fixed order, plus } j^*\text{)}$$

   is invertible over $\mathbb{Z}_q$.
3. The $d \times d$ matrix $\mathbf{B}_T$ obtained by stacking the rows of $T$ (in the same fixed order) and then appending $\mathbb{1}_1$ as the last row is invertible.

*Proof.* Since $T$ is invalid, $\mathbb{1}_1 \notin \mathrm{span}((\mathbf{M}[j])_{j\in T})$. By LI-unauthorized, the rows in $T$ are linearly independent, hence $\mathrm{rank}((\mathbf{M}[j])_{j\in T}) = |T|$. Maximality means that for every $j^* \notin T$, the set $T \cup \{j^*\}$ is valid, i.e., $\mathbb{1}_1 \in \mathrm{span}((\mathbf{M}[j])_{j\in T\cup\{j^*\}})$. In particular $j^*$ cannot lie in the span of $T$ (otherwise validity would already hold for $T$), so adding $j^*$ strictly increases rank by 1. Since adding any $j^* \notin T$ increases rank by 1, we have $\mathrm{rank}(T \cup \{j^*\}) = \mathrm{rank}(T) + 1$ for all $j^* \notin T$. We now argue that $\mathrm{rank}(T) = d-1$. Otherwise, if $\mathrm{rank}(T) \le d-2$, then $\dim(\mathrm{span}((\mathbf{M}[j])_{j\in T}, \mathbb{1}_1)) \le d-1$. Maximality implies that for every $j^* \notin T$ we have $\mathbb{1}_1 \in \mathrm{span}((\mathbf{M}[j])_{j\in T\cup\{j^*\}})$, hence $\mathbf{M}[j^*] \in \mathrm{span}((\mathbf{M}[j])_{j\in T}, \mathbb{1}_1)$. Therefore every row of $\mathbf{M}$ lies in the $(\le d-1)$-dimensional subspace $\mathrm{span}((\mathbf{M}[j])_{j\in T}, \mathbb{1}_1)$, contradicting $\mathrm{rank}(\mathbf{M}) = d$. Hence $\mathrm{rank}(T) = d-1$. Since the rows in $T$ are linearly independent (by LI-unauthorized), it follows that $|T| = \mathrm{rank}(T) = d-1$, and thus $\mathrm{rank}(T \cup \{j^*\}) = d$ for all $j^* \notin T$, which implies that $\mathbf{A}_{T,j^*}$ is invertible. Finally, since $\mathbb{1}_1 \notin \mathrm{span}(T)$ and $\mathrm{rank}(T) = d-1$, appending $\mathbb{1}_1$ also increases rank to $d$, hence $\mathbf{B}_T$ is invertible. □

**Instantiating the programmed sharing algorithms.** We now define $\mathsf{ProgShare}_1$ and $\mathsf{ProgShare}_2$ by solving for the unique sharing vector $\mathbf{k} = (k, r_2, \ldots, r_d)^\top$ consistent with the programmed data.

**Mode 1 ($\mathsf{ProgShare}_1$).** On input $(\mathbb{A}, T, (\tilde{s}_j)_{j\in T}, j^*, \tilde{s}_{j^*})$ where $T$ is maximal invalid:

1. Form the invertible matrix $\mathbf{A}_{T,j^*}$ from Lemma 4.12, and form the programming constraint vector $\mathbf{b}_{T,j^*} := ((\tilde{s}_j)_{j\in T}, \tilde{s}_{j^*})^\top \in \mathbb{Z}_q^d$.
2. Compute the unique solution $\mathbf{k} := \mathbf{A}_{T,j^*}^{-1}\mathbf{b}_{T,j^*}$.
3. Set $k := \langle \mathbb{1}_1, \mathbf{k}\rangle$ and compute $\mathbf{w} := \mathbf{M}\mathbf{k}$.
4. Output $(k, s_1, \ldots, s_N)$ where $s_i := (w_j)_{j\in T_i}$.

**Mode 2 ($\mathsf{ProgShare}_2$).** On input $(\mathbb{A}, T, (\tilde{s}_j)_{j\in T}, k)$ where $T$ is maximal invalid:

1. Form the invertible matrix $\mathbf{B}_T$ from Lemma 4.12, and form the programming constraint vector $\mathbf{b}_{T,k} := ((\tilde{s}_j)_{j\in T}, k)^\top \in \mathbb{Z}_q^d$.
2. Compute the unique solution $\mathbf{k} := \mathbf{B}_T^{-1}\mathbf{b}_{T,k}$.
3. Compute $\mathbf{w} := \mathbf{Mk}$ and output $(k, s_1, \ldots, s_N)$ where $s_i := (w_j)_{j\in T_i}$.

Both procedures are deterministic given a fixed row-ordering inside each $T_i$.

**Theorem 4.13.** Assume that the LSSS for $\mathbb{A}$ has $\mathrm{rank}(\mathbf{M}) = d$ and satisfies Definition 4.11. Then the above constructions of $\mathsf{ProgShare}_1$ and $\mathsf{ProgShare}_2$ satisfy Definition 4.10 (correctness/consistency, mode indistinguishability).

*Proof. Correctness and consistency.* In both modes, the algorithm outputs shares of the form $\mathbf{w} = \mathbf{Mk}$ for some vector $\mathbf{k} = (k, r_2, \ldots, r_d)^\top$. By the correctness of the underlying LSSS, every authorized set $A \in \mathbb{A}$ reconstructs $k$ from the shares $(s_i)_{i\in A}$. Consistency with the programmed inputs holds by construction: the defining linear system enforces $w_j = \tilde{s}_j$ for all $j \in T$ (and additionally $w_{j^*} = \tilde{s}_{j^*}$ in Mode 1), and the output shares are exactly the appropriate coordinates of $\mathbf{w}$.

*Mode indistinguishability.* Fix any maximal invalid share sets $T_0^*, T_1^*$ and any $j^* \notin T_0^*$. In $\mathsf{Exp}_1$, the input vector $\mathbf{b}_{T_0^*,j^*} = ((\tilde{s}_{0,j})_{j\in T_0^*}, \tilde{s}_{0,j^*})$ is uniform in $\mathbb{Z}_q^d$. Since $\mathbf{A}_{T_0^*,j^*}$ is invertible, $\mathbf{k} = \mathbf{A}_{T_0^*,j^*}^{-1}\mathbf{b}_{T_0^*,j^*}$ is uniform in $\mathbb{Z}_q^d$. Thus the output $(k_0, s_{0,1}, \ldots, s_{0,N})$ is distributed exactly as $(\langle \mathbb{1}_1, \mathbf{k}\rangle, (\mathbf{Mk})_{T_1}, \ldots, (\mathbf{Mk})_{T_N})$ for uniform $\mathbf{k} \leftarrow \mathbb{Z}_q^d$.

In $\mathsf{Exp}_2$, the input vector $\mathbf{b}_{T_1^*,k} = ((\tilde{s}_{1,j})_{j\in T_1^*}, k)$ is also uniform in $\mathbb{Z}_q^d$ (since $(\tilde{s}_{1,j})_{j\in T_1^*}$ and $k$ are sampled uniformly and independently). Since $\mathbf{B}_{T_1^*}$ is invertible, the resulting $\mathbf{k} = \mathbf{B}_{T_1^*}^{-1}\mathbf{b}_{T_1^*,k}$ is uniform in $\mathbb{Z}_q^d$, hence the output $(k_1, s_{1,1}, \ldots, s_{1,N})$ has the *same* distribution as above with uniform $\mathbf{k}$. Therefore $\mathsf{Exp}_1$ and $\mathsf{Exp}_2$ are identically distributed, independent of the particular choice of $T_0^*, T_1^*$.

□

# 5 Distributed Monotone Policy Encryption with Silent Setup

**Syntax.** A distributed monotone policy encryption scheme with silent setup (DPE) for access policies $\mathbb{S} = \{\mathbb{S}_L\}_{L\in\mathbb{N}}$ has the following polynomial time algorithms.

$\mathsf{GSetup}(1^\lambda, 1^L, N) \to \mathsf{crs}$**.** The possibly randomized global setup algorithm takes as input security parameter $\lambda$, number of users $N$, and outputs the common reference string $\mathsf{crs}$. The following algorithms take $\mathsf{crs}$ implicitly.

$\mathsf{KGen}(i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$**.** The randomized key generation algorithm takes as input $i \in [N]$, and outputs public-secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.

$\mathsf{PreProcess}(\{\mathsf{pk}_i\}_{i\in S}, \mathbb{A}) \to (\mathsf{mpk}, \mathsf{aux})$**.** The randomized *public-coin* pre-processing algorithm takes as input public keys for a users in set $S$, $\{\mathsf{pk}_i\}_{i\in S}$, access structure $\mathbb{A}$ for $|S|$ parties, outputs master public key $\mathsf{mpk}$ and auxiliary information $\mathsf{aux}$.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{aux}, \mu) \to \mathsf{ct}$**.** The randomized encryption algorithm takes as input master public key $\mathsf{mpk}$, auxiliary information $\mathsf{aux}$, message $\mu \in \{0,1\}$, and outputs ciphertext $\mathsf{ct}$.

$\mathsf{PartDec}(\{\mathsf{pk}_i\}_{i\in S}, (j, \mathsf{sk}_j), \mathsf{ct}) \to \mathsf{dsh}_j$**.** The possibly randomized partial decryption algorithm takes as input public keys for users in $S$, $\{\mathsf{pk}_i\}_{i\in S}$, secret key for the $j$-th user $\mathsf{sk}_j$, ciphertext $\mathsf{ct}$, and outputs decryption share $\mathsf{dsh}_j$.

$\mathsf{Recon}(\{\mathsf{dsh}_j\}_{j\in S'}, \mathbb{A}, \mathsf{ct}) \to \mu'$. The deterministic reconstruction algorithm takes as input decryption shares from users in set $S'$, $\{\mathsf{dsh}_j\}_{j\in S'}$, access structure $\mathbb{A}$, ciphertext $\mathsf{ct}$, and outputs message $\mu' \in \{0,1\}\cup\{\perp\}$.

**Definition 5.1** (DPE)**.** A DPE scheme $(\mathsf{GSetup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{Recon})$ is said to be a DPE scheme for access policies $\mathbb{S}$ if it satisfies the following properties.

**Correctness.** For any $\lambda \in \mathbb{N}, L \in \mathbb{N}, L = \mathsf{poly}(\lambda), N \le 2^\lambda$, $\mu \in \{0,1\}$ monotone access policy $\mathbb{A} \in \mathbb{S}_L, S' \in \mathbb{A}$,

$$\Pr\left[\mu \leftarrow \mathsf{Recon}(\{\mathsf{dsh}_j\}_{j\in S'}, \mathbb{A}, \mathsf{ct}) \quad : \quad \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{GSetup}(1^\lambda, 1^L, N), \\ \forall\ i \in S, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(i), \\ (\mathsf{mpk}, \mathsf{aux}) \leftarrow \mathsf{PreProcess}(\{\mathsf{pk}_i\}_{i\in S}, \mathbb{A}), \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{aux}, \mu), \forall\ j \in S', \\ \mathsf{dsh}_j \leftarrow \mathsf{PartDec}(\{\mathsf{pk}_i\}_{i\in S}, (j, \mathsf{sk}_j), \mathsf{ct}) \end{array}\right] = 1$$

**Compactness.** There exists a polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for any $\lambda \in \mathbb{N}, L \in \mathbb{N}, L = \mathsf{poly}(\lambda), N \le 2^\lambda, S \subseteq [N], \mathbb{A} \in \mathbb{S}_L$, $|\mathsf{ct}| = \mathsf{poly}(\lambda, \log N, \log L)$.

**Adaptive Security.** For any stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mu \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(\mathsf{ct}^*, \mathsf{aux}) \quad : \quad \begin{array}{l} (N, 1^L) \leftarrow \mathcal{A}(1^\lambda), C \leftarrow \emptyset, \\ \mathsf{crs} \leftarrow \mathsf{GSetup}(1^\lambda, 1^L, N), \\ (S, \mathbb{A}) \leftarrow \mathcal{A}^{\mathcal{O}(\cdot,\cdot)}(\mathsf{crs}), \\ (\mathsf{mpk}, \mathsf{aux}) \leftarrow \mathsf{PreProcess}(\{\mathsf{pk}_i\}_{i\in S}, \mathbb{A}), \\ \mu \leftarrow \{0,1\}, \mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{aux}, \mu) \end{array}\right] - \frac{1}{2}\right| \le \mathsf{negl}(\lambda)$$

where $\mathcal{O}$ is a stateful oracle that responds to queries of the form

- $(\mathsf{KGen}, i)$**.** Given $i \in [N]$, if $\mathsf{KGen}$ was queried on $i$, send $\mathsf{pk}_i$ to $\mathcal{A}$. Otherwise, sample $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(i)$, store $(\mathsf{pk}_i, \mathsf{sk}_i)$, and send $\mathsf{pk}_i$ to $\mathcal{A}$.
- $(\mathsf{Corr}, i)$**.** Given $i \in [N]$, if $\mathsf{KGen}$ wasn't queried on $i$, send $\perp$ to $\mathcal{A}$. Otherwise, send the corresponding $\mathsf{sk}_i$. Set $C := C \cup \{i\}$.
- $(\mathsf{PartDec}, i, \{\mathsf{pk}_j\}_{j\in S'}, \mathsf{ct})$**.** Given $i \in [N], \{\mathsf{pk}_j\}_{j\in S'}, \mathsf{ct}$, if $\mathsf{KGen}$ wasn't queried on $i$, output $\perp$. Otherwise, if $i \notin S'$, output $\perp$. Otherwise, compute $\mathsf{dsh}_i \leftarrow \mathsf{PartDec}(\{\mathsf{pk}_j\}_{j\in S'}, (i, \mathsf{sk}_i), \mathsf{ct})$ and send $\mathsf{dsh}_j$ to $\mathcal{A}$. If $\mathsf{ct} = \mathsf{ct}^*$, set $C = C \cup \{i\}$.

A stateful PPT machine $\mathcal{A}$ is said to be admissible if for each $i^* \in S$, $(\mathsf{KGen}, i^*)$ query was made before challenge phase and at the end of the game, $C \cap S \notin \mathbb{A}$.

**Definition 5.2** (Selective Security)**.** A DPE scheme $(\mathsf{GSetup}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{Recon})$ is said to be a selectively-secure DPE scheme for access policies $\mathbb{S}$ if it satisfies correctness and compactness properties from Definition 5.2 and the following selective security definition.

**Selective Security.** For any stateful admissible PPT adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mu \leftarrow \mathcal{A}\left(\begin{array}{c}\mathsf{crs}, \{\mathsf{pk}_i\}_{i\in S}, \\ \{\mathsf{sk}_j\}_{j\in C}, \mathsf{aux}, \mathsf{ct}\end{array}\right) \quad : \quad \begin{array}{l} (N, 1^L, \mathbb{A}, S, C) \leftarrow \mathcal{A}(1^\lambda), \\ \mathsf{crs} \leftarrow \mathsf{GSetup}(1^\lambda, 1^L, N), \\ \forall\ i \in S, (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(i), \\ (\mathsf{mpk}, \mathsf{aux}) \leftarrow \mathsf{PreProcess}(\{\mathsf{pk}_i\}_{i\in S}, \mathbb{A}), \\ \mu \leftarrow \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{aux}, \mu) \end{array}\right] - \frac{1}{2}\right| \le \mathsf{negl}(\lambda)$$

A stateful PPT machine $\mathcal{A}$ is said to be admissible if $C \subseteq S \subseteq [N]$ and $C \notin \mathbb{A}$.

# 6 Distributed Monotone Policy Encryption with Silent Setup for Programmable Binary Linear Secret Sharing Schemes

**Parameters.** We define the parameters used in our construction below.

- $n = \mathsf{poly}(\lambda), \chi = \mathsf{poly}(n, \log N, \log L)$ (Decomposed LWE assumption)
- $m = \Omega(n \log q)$ (Leftover hash lemma)
- $\sigma = \mathsf{poly}(\lambda, m)$ (Decomposed LWE assumption)
- $\chi' > \lambda^{\omega(1)} NLO(\sigma \sigma_{\mathsf{er}} \chi m^{14} \log q \log N)$ (Smudging)
- $\sigma_{\mathsf{er}} > \lambda^{\omega(1)} O(\sigma m^{13} \log q \log N)$ (Security)
- $q > 4NL\sqrt{m}\chi'$ (Correctness)

**Construction 6.1.** We provide the construction of a DPE scheme (Definition 5.2) for programmable $\{0,1\}$-LSSS $\mathbb{S} = \{\mathbb{S}_L\}_{L \in \mathbb{N}}$ (Definition 4.10) using $2m^2$-decomposed LWE (Assumption 3.6), matrix commitment scheme MCOM (Definition 3.8), and explainable re-randomizer ER (Definition 3.9) as follows.

$\mathsf{GSetup}(1^\lambda, 1^L, N)$**.** Sample $(\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \mathsf{MCOM.Setup}(1^n, 1^m, q, \sigma)$. Sample hash function $\mathcal{H}_\rho : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ and $\mathcal{H}_{\mathsf{sh}} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \mathbb{Z}_q^n$.

Output $\mathsf{crs} := (N, L, \mathbf{W}, \mathbf{D}, \mathbf{B}, \mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}})$.

$\mathsf{KGen}(i)$**.** Sample $\mathbf{R}_i \leftarrow \{\pm 1\}^{4m^5 \times L}$ and compute $\mathbf{T}_i = \mathbf{B}\mathbf{R}_i$. Output $(\mathsf{pk}_i, \mathsf{sk}_i) := ((i, \mathbf{T}_i), \mathbf{R}_i)$.

$\mathsf{PreProcess}(\{\mathsf{pk}_i\}_{i \in S}, \mathbb{A})$**.** Parse $\mathsf{pk}_i$ as $(i, \mathbf{T}_i)$ for each $i \in S$. If $[|S|] \notin \mathbb{A}$, output $(\bot, \bot)$. Do the following.

1. Sample $\mathsf{sd}_{\mathsf{sh}}, \mathsf{sd}_\rho \leftarrow \{0,1\}^\lambda$.
2. Let $\rho = \mathsf{poly}(\lambda, |S|)$ be the length of randomness used by ER. Sample $r_k = \mathcal{H}_\rho(\mathsf{sd}_\rho, k)$ for each $k \in [\rho]$.
3. Compute $(\mathbf{C}_0, \mathbf{Z}_0) = \mathsf{ER.Sample}(1^\lambda, 1^{|S|}, \sigma_{\mathsf{er}}; (r_1, \ldots, r_\rho))$.
4. Compute a maximally invalid share set $A'$ for $\mathbb{A}$ over $|S|$ parties[3]. Then find the lexicographically smallest index $i^* \in [|S| \cdot L]$ that is not present in $A'$ and set $A = A' \cup \{i^*\}$.
5. For each $j \in A$, compute $\widetilde{\mathsf{sh}}_j = \mathcal{H}_{\mathsf{sh}}(\mathsf{sd}_{\mathsf{sh}}, j)$.
6. Compute $(\mathbf{p}, \mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}) = \mathsf{ProgShare}_1(\mathbb{A}, A', \{\widetilde{\mathsf{sh}}_j\}_{j \in A}, i^*, \widetilde{\mathsf{sh}}_{i^*})$.
7. Compute matrix $\mathbf{X} = [\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_{|S|}] \in \mathbb{Z}_q^{n \times L \cdot |S|}$ with $\mathbf{X}_j = \mathbf{T}_j + \mathsf{sh}_j$.
8. Compute $\mathbf{C} = \mathsf{MCOM.Com}(\mathbf{X})$.

Output $\mathsf{mpk} := \mathbf{C}_0 + \mathbf{C}, \mathsf{aux} := (\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathbf{p})$.

$\mathsf{Enc}(\mathsf{mpk}, \mathsf{aux}, \mu)$**.** If $\mathsf{mpk}$ or $\mathsf{aux}$ is $\bot$, output $\bot$. Otherwise, parse $\mathsf{mpk}$ as $\widehat{\mathbf{C}} \in \mathbb{Z}_q^{n \times m}$ and $\mathsf{aux}$ as $(\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathbf{p})$.

1. Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\chi}^{4m^5}, \mathbf{R}_{\mathbf{C}} \leftarrow \{\pm 1\}^{4m^5 \times m}, \mathbf{r}_{\mathbf{p}} \leftarrow \{\pm 1\}^{4m^5}$.
2. Compute $\mathbf{c}_1^\top = \mathbf{s}^\top \mathbf{B} + \mathbf{e}^\top, \mathbf{c}_2^\top = \mathbf{s}^\top \widehat{\mathbf{C}} + \mathbf{e}^\top \mathbf{R}_{\mathbf{C}}, c_3 = \mathbf{s}^\top \mathbf{p} + \mathbf{e}^\top \mathbf{r}_{\mathbf{p}} + \lfloor q/2 \rfloor \mu$.

Output $\mathsf{ct} := (\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}) \in \mathbb{Z}_q^{1 \times (2m+1)} \times \{0,1\}^{2\lambda}$.

$\mathsf{PartDec}(\{\mathsf{pk}_i\}_{i \in S}, (j, \mathsf{sk}_j), \mathsf{ct})$**.** If $\mathsf{ct}$ is $\bot$, output $\bot$. Otherwise, parse $\mathsf{pk}_i$ as $(i, \mathbf{T}_i)$ for each $i \in S$. Parse $\mathsf{sk}_j$ as $\mathbf{R}_j \in \{\pm 1\}^{4m^5 \times L}$. If $j \notin S$, abort and output $\bot$. Otherwise, parse $\mathsf{ct}$ as $(\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}) \in \mathbb{Z}_q^{1 \times (2m+1)} \times \{0,1\}^{2\lambda}$ and perform the following steps.

1. Perform steps 2-8 from $\mathsf{PreProcess}$ using $\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathcal{H}_{\mathsf{sh}}, \mathcal{H}_\rho$. From these, use $\mathbf{X}, \mathbf{Z}_0$.

[3] We remark that this process is deterministic by a folklore algorithm. Also, check [BGG+18].

2. Compute $\mathbf{V}_{|S|} = \mathsf{MCOM.Verify}(1^{|S|})$ and $\mathbf{Z}_{\mathbf{X}} = \mathsf{MCOM.Open}(\mathbf{X})$.
3. Compute $\mathbf{c}_{exp}^\top = \mathbf{c}_2^\top \mathbf{V}_{|S|} + \mathbf{c}_1^\top(\mathbf{Z}_0 + \mathbf{Z}_{\mathbf{X}}) \in \mathbb{Z}_q^{1\times|S|\cdot L}$.
4. Let $i_j \in [|S|]$ denote the index of $j$ in the set $S$ according to lexicographical ordering. Let $\widetilde{\mathbf{c}}^\top \in \mathbb{Z}_q^{1\times L}$ be the concatenation of $((i_j-1)\cdot L+1)$-th to $i_j \cdot L$-th entries of $\mathbf{c}_{exp}^\top$.
5. Sample $\mathbf{e}' \leftarrow \mathcal{D}_{\mathbb{Z},\chi'}^L$.

Output $\mathsf{dsh}_j := \widetilde{\mathbf{c}}^\top - \mathbf{c}_1^\top \mathbf{R}_j + (\mathbf{e}')^\top \in \mathbb{Z}_q^{1\times L}$.

$\mathsf{Recon}(\{\mathsf{dsh}_j\}_{j\in S'}, \mathbb{A}, \mathsf{ct})$**.** If $\mathsf{ct}$ is $\perp$, output $\perp$. Otherwise, parse $\mathsf{ct}$ as $(\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}) \in \mathbb{Z}_q^{1\times(2m+1)} \times \{0,1\}^{2\lambda}$. If $S' \notin \mathbb{A}$, abort and output $\perp$. Otherwise, compute the reconstruction coefficients $\mathbf{w}^\top \in \{0,1\}^{|S'|\cdot L}$ for $S'$[4]. Parse $\mathbf{w}^\top$ as $(w_{j,k})_{j\in[|S'|],k\in[L]}$. Similarly parse each $\mathsf{dsh}_j$ as $\mathsf{dsh}_{j,k} \in \mathbb{Z}_q^n$ for $k \in [L]$. Compute $c' = \sum_{j\in[|S'|],k\in[L]} w_{j,k}\mathsf{dsh}_{j,k}$. Compute $y = c_3 - c'$. Set $\mu' := 0$ if $-q/4 < y < q/4$ and 1 otherwise. Output $\mu'$.

**Correctness.** Note that $\mathsf{Recon}$ outputs $\perp$ if $S' \notin \mathbb{A}$. So, we will only focus on the case when $S' \in \mathbb{A}$. For each $i \in S$, we have that $\mathsf{pk}_i = (i, \mathbf{T}_i), \mathsf{sk}_i = \mathbf{R}_i$ where $\mathbf{T}_i = \mathbf{B}\mathbf{R}_i$. In $\mathsf{PreProcess}, \mathsf{PartDec}$, $\mathbf{X} = [\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_{|S|}]$ where $\mathbf{X}_j = \mathbf{T}_j + \mathsf{sh}_j$ for $j \in [|S|]$. Here, $\mathsf{sh}_j$ is computed as mentioned in $\mathsf{PreProcess}$ steps 4-6. Similarly, in $\mathsf{PreProcess}, \mathsf{PartDec}$, we have $(\mathbf{C}_0, \mathbf{Z}_0)$ such that $\mathbf{C}_0\mathbf{V}_{|S|} = -\mathbf{B}\mathbf{Z}_0$ by correctness of ER where $\mathbf{V}_{|S|} = \mathsf{MCOM.Verify}(1^{|S|})$. By correctness of MCOM, we have $\mathbf{C}\mathbf{V}_{|S|} = \mathbf{X} - \mathbf{B}\mathbf{Z}_{\mathbf{X}}$ where $\mathbf{C} = \mathsf{MCOM.Com}(\mathbf{X}), \mathbf{Z}_{\mathbf{X}} = \mathsf{MCOM.Verify}(\mathbf{X})$. Note that we also compute $(\mathbf{C}_0, \mathbf{Z}_0)$ using the deterministic procedure from steps 2-3 in $\mathsf{PreProcess}$ so they are the same in $\mathsf{PreProcess}, \mathsf{PartDec}$. Thus,

$$\begin{aligned}
\mathbf{c}_{exp}^\top &= \mathbf{c}_2^\top \mathbf{V}_{|S|} + \mathbf{c}_1^\top(\mathbf{Z}_0 + \mathbf{Z}_X) \\
&= \mathbf{s}^\top(\mathbf{C}_0\mathbf{V}_{|S|} + \mathbf{C}\mathbf{V}_{|S|} + \mathbf{B}\mathbf{Z}_0 + \mathbf{B}\mathbf{Z}_{\mathbf{X}}) + \overbrace{\mathbf{e}^\top(\mathbf{Z}_0 + \mathbf{Z}_{\mathbf{X}} + \mathbf{R}_{\mathbf{C}}\mathbf{V}_{|S|})}^{\widetilde{\mathbf{e}}^\top} \\
&= \mathbf{s}^\top\mathbf{X} + \widetilde{\mathbf{e}}^\top
\end{aligned}$$

Hence, $\widetilde{\mathbf{c}}_j^\top$ from $\mathsf{PartDec}$ with $\mathsf{sk}_j$ is $\mathbf{s}^\top(\mathbf{T}_j + \mathsf{sh}_j) + \widetilde{\mathbf{e}}_j^\top + (\mathbf{e}'_j)^\top$. This means that $\mathsf{dsh}_j = \mathbf{s}^\top(\mathbf{T}_j + \mathsf{sh}_j) + \widetilde{\mathbf{e}}^\top - \mathbf{s}^\top\mathbf{B}\mathbf{R}_j - \mathbf{e}^\top\mathbf{R}_j + (\mathbf{e}'_j)^\top = \mathbf{s}^\top\mathsf{sh}_j + \widetilde{\mathbf{e}}^\top - \mathbf{e}^\top\mathbf{R}_j + (\mathbf{e}'_j)^\top$. In $\mathsf{Recon}$, as $S' \in \mathbb{A}$, we have that $\sum_{j\in[|S'|],k\in[L]} w_{j,k}\mathsf{sh}_{j,k} = \mathbf{p}$ as $\mathbf{p}$ is the secret generated in $\mathsf{PreProcess}, \mathsf{PartDec}$. Thus, $c' = \mathbf{s}^\top\mathbf{p} + \sum_{j\in[|S'|],k\in[L]} w_{j,k}(\widetilde{e}_k - (\mathbf{e}^\top\mathbf{R}_j)_k + e'_{j,k})$ where $\widetilde{e}_k, (\mathbf{e}^\top\mathbf{R}_j)_k, e'_{j,k}$ are the $k$-th elements of $\widetilde{\mathbf{e}}, \mathbf{e}^\top\mathbf{R}_j, \mathbf{e}'_j$ respectively. Hence, we have that $\mu' = \mu$ as long as $q > 4\left\|\sum_{j\in[|S'|],k\in[L]} w_{j,k}(\widetilde{e}_k - (\mathbf{e}^\top\mathbf{R}_j)_k) + e'_{j,k}\right\|$.

We will now bound the $\widetilde{\mathbf{e}}^\top$ and $\sum_{j\in[|S'|],k\in[L]} w_{j,k}(\widetilde{e}_k - (\mathbf{e}^\top\mathbf{R}_j)_k + e'_{j,k})$. With high probability, $\|\mathbf{Z}_0\| \le \sqrt{m}\sigma_{\mathsf{er}}, \|\mathbf{T}\| \le \sqrt{m}\sigma$, and $\|\mathbf{e}\| \le \sqrt{m}\chi, \|\mathbf{e}'\| \le \sqrt{m}\chi$. Moreover, $\|\mathbf{R}_{\mathbf{C}}\| = 1, \|\mathbf{Z}_{\mathbf{X}}\| \le O(\sigma m^{15/2}\log q \log N)$ and $\left\|\mathbf{V}_{|S|}\right\| \le O(\sigma m^{9/2}\log q)$. Thus, $\left\|\mathbf{Z}_0 + \mathbf{Z}_{\mathbf{X}} + \mathbf{R}_{\mathbf{C}}\mathbf{V}_{|S|}\right\| \le O(\sigma\sigma_{\mathsf{er}} m^{15/2}\log q \log N)$. Thus, $\left\|\widetilde{\mathbf{e}}^\top\right\| \le O(\sigma\sigma_{\mathsf{er}}\chi m^{13}\log q \log N)$.

Now, $\left\|\sum_{j\in[|S'|],k\in[L]} w_{j,k}(\widetilde{e}_k - (\mathbf{e}^\top\mathbf{R}_j)_k + e'_{j,k})\right\| \le NL\|\mathbf{e}'\| \le NL\sqrt{m}\chi'$. Thus, if $q > 4NL\sqrt{m}\chi'$, correctness holds.

**Compactness.** Note that size of $\mathsf{ct}$ is $\Theta(m\log q)+2\lambda$. As $m = \Omega(n\log q)$, we have that $|\mathsf{ct}| = \mathsf{poly}(\lambda, \log N, \log L)$. Thus, compactness holds. We also remark that $|\mathsf{crs}| = \mathsf{poly}(\lambda, \log N, \log L)$ and $|\mathsf{pk}| = \mathsf{poly}(\lambda, \log N, L)$ and $|\mathsf{sk}| = \mathsf{poly}(\lambda, \log N, L)$. This dependency of $\mathsf{pk}, \mathsf{sk}$ on $L$ is a consequence of maximum fan-out in $\mathbb{S}_L$.

Moreover, running time of $\mathsf{KGen}, \mathsf{Enc}$ are independent of $N$ as well. On the other hand, running time of $\mathsf{PreProcess}, \mathsf{PartDec}, \mathsf{Recon}$ grows only with $|S|$ and not $N$.

[4]We remark that this is not exactly in line with Definition 4.6. However, we can assume that the values of $\mathbf{w}$ for indices $i$ not in the minimal valid share set are 0.

## 6.1 Security Analysis

**Theorem 6.2.** If $2m^2$-decomposed LWE assumption holds (Assumption 3.6), $\mathsf{MCOM}$ satisfies mode indistinguishability, $\mathsf{ER}$ satisfies randomness and explainability properties, then Construction 6.1 is satisfies selective security (Definition 5.2) for programmable linear secret sharing schemes $\mathbb{S} = \{\mathbb{S}_L\}_{L\in\mathbb{N}}$ (Definition 4.10).

*Proof.* We define the following hybrids.

**$\mathbf{Hyb}_0^{\mathcal{A}}(1^\lambda)$.** This is the security experiment from Definition 5.2 with $b = 0/1$.

**Setup Phase.** $\mathcal{A}$ sends $(N, 1^L, \mathbb{A}, S, C)$. Do the following.

1. Sample $(\mathbf{W}, \mathbf{D}, \mathbf{B}) \leftarrow \mathsf{MCOM.Setup}(1^n, 1^m, q, \sigma)$.
2. For each $i \in S$, sample $\mathbf{R}_i \leftarrow \{\pm 1\}^{4m^5\times L}$ and set $\mathsf{pk}_i := (i, \mathbf{BR}_i), \mathsf{sk}_i := \mathbf{R}_i$.
3. Sample $\mathsf{sd}_{\mathsf{sh}}, \mathsf{sd}_\rho \leftarrow \{0,1\}^\lambda$.
4. Let $\rho = \mathsf{poly}(\lambda, |S|)$ be the length of randomness used by $\mathsf{ER}$. Sample $r_k = \mathcal{H}_\rho(\mathsf{sd}_\rho, k)$ for each $k \in [\rho]$.
5. Compute $(\mathbf{C}_0, \mathbf{Z}_0) = \mathsf{ER.Sample}(1^\lambda, 1^{|S|}, \sigma_{\mathsf{er}}; (r_1, \ldots, r_\rho))$.
6. Compute a maximally invalid share set $A'$ for $\mathbb{A}$ over $|S|$ parties. Then find the lexicographically smallest index $i^* \in [|S| \cdot L]$ that is not present in $A'$ and set $A = A' \cup \{i^*\}$.
7. For each $j \in A$, compute $\widetilde{\mathsf{sh}}_j = \mathcal{H}_{\mathsf{sh}}(\mathsf{sd}_{\mathsf{sh}}, j)$.
8. Compute $(\mathbf{p}, \mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}) = \mathsf{ProgShare}_1(\mathbb{A}, A', \{\widetilde{\mathsf{sh}}_j\}_{j\in A}, i^*, \widetilde{\mathsf{sh}}_{i^*})$.
9. Compute matrix $\mathbf{X} = [\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_{|S|}] \in \mathbb{Z}_q^{n\times L\cdot|S|}$ with $\mathbf{X}_j = \mathbf{T}_j + \mathsf{sh}_j$.
10. Compute $\mathbf{C} = \mathsf{MCOM.Com}(\mathbf{X})$.
11. Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\chi}^{4m^5}, \mathbf{R}_{\mathbf{C}} \leftarrow \{\pm 1\}^{4m^5\times m}, \mathbf{r}_{\mathbf{p}} \leftarrow \{\pm 1\}^{4m^5}$.
12. Compute $\mathbf{c}_1^\top = \mathbf{s}^\top\mathbf{B} + \mathbf{e}^\top, \mathbf{c}_2^\top = \mathbf{s}^\top(\mathbf{C}_0 + \mathbf{C}) + \mathbf{e}^\top\mathbf{R}_{\mathbf{C}}, c_3 = \mathbf{s}^\top\mathbf{p} + \mathbf{e}^\top\mathbf{r}_{\mathbf{p}} + \lfloor q/2\rfloor b$.
13. Set $\mathsf{ct} := (\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}})$.

Send $(\mathsf{crs} := (N, L, \mathbf{W}, \mathbf{D}, \mathbf{B}, \mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}), \{\mathsf{pk}_i\}_{i\in S}, \{\mathsf{sk}_j\}_{j\in C}, \mathsf{aux} := (\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathbf{p}), \mathsf{ct})$ to $\mathcal{A}$. Here, $\mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}$ are handles to random oracles.

**Random Oracle Queries.** Given a query $(\mathsf{sd}, i) \in \{0,1\}^\lambda, \{0,1\}^\lambda$ for $\mathcal{H}_\rho$ or $\mathcal{H}_{\mathsf{sh}}$, if the query is for $\mathcal{H}_\rho$, sample $\beta \leftarrow \{0,1\}$ and send $\beta$. Otherwise, if the query is for $\mathcal{H}_{\mathsf{sh}}$, sample $\mathsf{sh} \leftarrow \mathbb{Z}_q^{n\times L}$ and send $\mathsf{sh}$.

**Guess Phase.** $\mathcal{A}$ outputs its guess $b'$. Output $b' \stackrel{?}{=} b$.

**$\mathbf{Hyb}_1^{\mathcal{A}}(1^\lambda)$.** This is the same as $\mathbf{Hyb}_0^{\mathcal{A}}(1^\lambda)$ except that we use a dictionary $\mathsf{H}$ and store the sampled values from random oracle queries in this dictionary. Changes are highlighted in red.

**Setup Phase.** $\mathcal{A}$ sends $(N, 1^L, \mathbb{A}, S, C)$. Initiate an empty dictionary $\mathsf{H}$. Then proceed exactly as $\mathbf{Hyb}_0^{\mathcal{A}}(1^\lambda)$.

**Random Oracle Queries.** Given a query $(\mathsf{sd}, i) \in \{0,1\}^\lambda, \{0,1\}^\lambda$ for $\mathcal{H}_\rho$ or $\mathcal{H}_{\mathsf{sh}}$, if the query is for $\mathcal{H}_\rho$, check if there is an entry for $(\rho, \mathsf{sd}, i)$ in $\mathsf{H}$. If yes, return that value. Otherwise, sample $\beta \leftarrow \{0,1\}$ and set $\mathsf{H}[(\rho, \mathsf{sd}, i)] = \beta$ and send $\beta$.

Otherwise, if the query is for $\mathcal{H}_{\mathsf{sh}}$, check if there is an entry for $(\mathsf{sh}, \mathsf{sd}, i)$ in $\mathsf{H}$. If yes, return that value. Otherwise, sample $\mathsf{sh} \leftarrow \mathbb{Z}_q^{n\times L}$ store $\mathsf{H}[(\mathsf{sh}, \mathsf{sd}, i)] = \mathsf{sh}$ and send $\mathsf{sh}$.

**Guess Phase.** $\mathcal{A}$ outputs its guess $b'$. Output $b' \stackrel{?}{=} b$.

$\mathbf{Hyb}_2^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_1^{\mathcal{A}}(1^\lambda)$ except that during setup phase, we will randomly sample $(r_1, \ldots, r_\rho) \leftarrow \{0,1\}^\rho$, $(\widetilde{\mathsf{sh}}_i)_{i \in A}$ and program these is $\mathsf{H}$ using $\mathsf{sd}_{\mathsf{sh}}, \mathsf{sd}_\rho$.

$\mathbf{Hyb}_3^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_2^{\mathcal{A}}(1^\lambda)$ except that we sample $\mathsf{pk}_i = (i, \mathbf{T}_i)$ where $\mathbf{T}_i \leftarrow \mathbb{Z}_q^{n \times L}$ for each $i \in S \setminus C$.

$\mathbf{Hyb}_4^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_3^{\mathcal{A}}(1^\lambda)$ except that we will change the way $\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}$ generated. Specifically, we will use $C$ to generate a maximally invalid set $A'$ and sample random shares $\widetilde{\mathsf{sh}}_j \leftarrow \mathbb{Z}_q^n$ for $j \in A'$. We will also sample a random secret $\mathbf{p} \leftarrow \mathbb{Z}_q^n$. Then, $(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}) = \mathsf{ProgShare}_2(\mathbb{A}, A', (\widetilde{\mathsf{sh}}_j)_{j \in A'}, \mathbf{p})$. We will then find $A$ similarly to $\mathsf{PreProcess}$ and program these values.

$\mathbf{Hyb}_5^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_4^{\mathcal{A}}(1^\lambda)$ except that we will use $\mathsf{ER.Explain}$ to generate $(r'_1, \ldots, r'_\rho)$ using $T'$. We will define $T'$ later.

$\mathbf{Hyb}_6^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_5^{\mathcal{A}}(1^\lambda)$ except that we sample $(\mathbf{W}, \mathbf{D}, \mathbf{B})$ along with $\mathbf{T}_{\mathbf{B}}$ using $\widetilde{\mathsf{MCOM.Setup}}(1^n, 1^m, q, \sigma)$.

$\mathbf{Hyb}_7^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_6^{\mathcal{A}}(1^\lambda)$ except that we sample $\mathbf{C}_0 \leftarrow \mathbb{Z}_q^{n \times m}$ and set $\mathbf{Z}_0 \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_{\mathbf{B}}, -\mathbf{C}_0 \mathbf{V}_{|S|}, \sigma_{\mathsf{er}})$. We write the full hybrid for clarity.

**Setup Phase.** $\mathcal{A}$ sends $(N, 1^L, \mathbb{A}, S, C)$. Initiate an empty dictionary $\mathsf{H}$. Do the following.

1. Sample $(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T}_{\mathbf{B}}) \leftarrow \widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma)$.
2. For each $i \in C$, sample $\mathbf{R}_i \leftarrow \{\pm 1\}^{4m^5 \times L}$ and set $\mathsf{pk}_i := (i, \mathbf{B}\mathbf{R}_i), \mathsf{sk}_i := \mathbf{R}_i$. For each $i \in S \setminus C$, sample $\mathbf{T}_i \leftarrow \mathbb{Z}_q^{n \times L}$ and set $\mathsf{pk}_i = (i, \mathbf{T}_i)$.
3. Sample $\mathsf{sd}_{\mathsf{sh}}, \mathsf{sd}_\rho \leftarrow \{0,1\}^\lambda$.
4. Sample $\mathbf{C}_0 \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{Z}_0 \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_{\mathbf{B}}, -\mathbf{C}_0 \mathbf{V}_{|S|}, \sigma_{\mathsf{er}})$.
5. Sample $r' \leftarrow \mathsf{ER.Explain}(1^\lambda, 1^{T'}, \mathbf{C}_0, \mathbf{Z}_0, \sigma_{\mathsf{er}})$ and set $\mathsf{H}[(\rho, \mathsf{sd}_\rho, k)] = r'_k$ for each $k \in [\rho]$ and $\rho = \mathsf{poly}(\lambda, |S|)$.
6. Use $C$ to generate a maximally invalid set $A'$ and sample random shares $\widetilde{\mathsf{sh}}_j \leftarrow \mathbb{Z}_q^n$ for $j \in A'$. Also sample a random secret $\mathbf{p} \leftarrow \mathbb{Z}_q^n$. Then, $(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}) = \mathsf{ProgShare}_2(\mathbb{A}, A', (\widetilde{\mathsf{sh}}_j)_{j \in A'}, \mathbf{p})$.
7. Now, compute a maximally invalid set $\widetilde{A}$ for $\mathbb{A}$ over $|S|$ parties like in $\mathsf{PreProcess}$. Then find the lexicographically smallest index $i^* \in [|S| \cdot L]$ that is not present in $\widetilde{A}$ and set for each $j \in A$, $\mathsf{H}[(\mathsf{sh}, \mathsf{sd}_{\mathsf{sh}}, j)] = \mathsf{sh}_j$.
8. Compute matrix $\mathbf{X} = [\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_{|S|}] \in \mathbb{Z}_q^{n \times L \cdot |S|}$ with $\mathbf{X}_j = \mathbf{T}_j + \mathsf{sh}_j$.
9. Compute $\mathbf{C} = \mathsf{MCOM.Com}(\mathbf{X})$.
10. Sample $\mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\chi}^{4m^5}, \mathbf{R}_{\mathbf{C}} \leftarrow \{\pm 1\}^{4m^5 \times m}, \mathbf{r}_{\mathbf{p}} \leftarrow \{\pm 1\}^{4m^5}$.
11. Compute $\mathbf{c}_1^\top = \mathbf{s}^\top \mathbf{B} + \mathbf{e}^\top, \mathbf{c}_2^\top = \mathbf{s}^\top (\mathbf{C}_0 + \mathbf{C}) + \mathbf{e}^\top \mathbf{R}_{\mathbf{C}}, c_3 = \mathbf{s}^\top \mathbf{p} + \mathbf{e}^\top \mathbf{r}_{\mathbf{p}} + \lfloor q/2 \rfloor b$.
12. Set $\mathsf{ct} := (\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}})$.

Send $(\mathsf{crs} := (N, L, \mathbf{W}, \mathbf{D}, \mathbf{B}, \mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}), \{\mathsf{pk}_i\}_{i \in S}, \{\mathsf{sk}_j\}_{j \in C}, \mathsf{aux} := (\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathbf{p}), \mathsf{ct})$ to $\mathcal{A}$. Here, $\mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}$ are handles to random oracles.

**Random Oracle Queries.** Given a query $(\mathsf{sd}, i) \in \{0,1\}^\lambda, \{0,1\}^\lambda$ for $\mathcal{H}_\rho$ or $\mathcal{H}_{\mathsf{sh}}$, if the query is for $\mathcal{H}_\rho$, check if there is an entry for $(\rho, \mathsf{sd}, i)$ in $\mathsf{H}$. If yes, return that value. Otherwise, sample $\beta \leftarrow \{0,1\}$ and set $\mathsf{H}[(\rho, \mathsf{sd}, i)] = \beta$ and send $\beta$.

Otherwise, if the query is for $\mathcal{H}_{\mathsf{sh}}$, check if there is an entry for $(\mathsf{sh}, \mathsf{sd}, i)$ in $\mathsf{H}$. If yes, return that value. Otherwise, sample $\mathsf{sh} \leftarrow \mathbb{Z}_q^{n \times L}$ store $\mathsf{H}[(\mathsf{sh}, \mathsf{sd}, i)] = \mathsf{sh}$ and send $\mathsf{sh}$.

**Guess Phase.** $\mathcal{A}$ outputs its guess $b'$. Output $b' \stackrel{?}{=} b$.

$\mathbf{Hyb}_8^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_7^{\mathcal{A}}(1^\lambda)$ except that we delay sampling $(\mathbf{C}_0, \mathbf{Z}_0)$ and set $\mathbf{C}_0$ as $\mathbf{BR}_\mathbf{C} - \mathbf{C}$ and sample $\mathbf{Z}_0 \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathbf{X} - \mathbf{B}(\mathbf{R}_\mathbf{C}\mathbf{V}_{|S|} + \mathbf{Z}_\mathbf{X}), \sigma_{\mathsf{er}})$ where $\mathbf{Z}_\mathbf{X} = \mathsf{MCOM.Open}(\mathbf{X})$ and $\mathbf{V}_{|S|} = \mathsf{MCOM.Verify}(1^{|S|})$. Note that $\mathbf{C}_0\mathbf{V}_{|S|} = -\mathbf{B}\mathbf{Z}_0$.

$\mathbf{Hyb}_9^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_8^{\mathcal{A}}(1^\lambda)$ except that we sample $\mathbf{Z}_0 = [\mathbf{Z}_{0,1} \mid \ldots \mid \mathbf{Z}_{0,|S|}] \in \mathbb{Z}_q^{m \times |S| \cdot L}$ as follows.

$$\mathbf{Z}_{0,j} \leftarrow \begin{cases} \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathbf{T}_j + \mathsf{sh}_j, \sigma_{\mathsf{er}}) - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j} & \text{if } j \in S \setminus C \\ \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathsf{sh}_j, \sigma_{\mathsf{er}}) + \mathbf{R}_j - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j} & \text{if } j \in C \end{cases}$$

$\mathbf{Hyb}_{10}^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_9^{\mathcal{A}}(1^\lambda)$ except that we set $\mathbf{T}_j = \mathbf{B}\mathbf{D}'_j - \mathsf{sh}_j$ for $j \in S \setminus C$ and $\mathsf{sh}_j = \mathbf{B}\mathbf{D}'_j$ where every $\mathbf{D}'_j \leftarrow \mathcal{D}_{\mathbb{Z},\sigma_{\mathsf{er}}}^{4m^5 \times L}$. In addition, we set

$$\mathbf{Z}_{0,j} = \begin{cases} \mathbf{D}'_j - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j} & \text{if } j \in S \setminus C \\ \mathbf{D}'_j + \mathbf{R}_j - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j} & \text{if } j \in C \end{cases}$$

where $\mathbf{V}_{|S|,j} \in \mathbb{Z}_q^{4m^5 \times L}$ is the partition corresponding to $i_j$ in $\mathbf{V}_{|S|}$ and $\mathbf{Z}_{\mathbf{X},j}$ is the partition corresponding to $i_j$ in $\mathbf{Z}_\mathbf{X}$.

$\mathbf{Hyb}_{11}^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_{10}^{\mathcal{A}}(1^\lambda)$ except that we sample $(\mathbf{W}, \mathbf{D}, \mathbf{B})$ uniformly randomly.

$\mathbf{Hyb}_{12}^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_{11}^{\mathcal{A}}(1^\lambda)$ except that we set $\mathbf{p} = \mathbf{B}\mathbf{r}_\mathbf{p}, \mathbf{r}_\mathbf{p} \leftarrow \{\pm 1\}^{4m^5}$.

$\mathbf{Hyb}_{13}^{\mathcal{A}}(1^\lambda)$. This is the same as $\mathbf{Hyb}_{12}^{\mathcal{A}}(1^\lambda)$ except that we sample $\mathbf{c}_1^\top \leftarrow \mathbb{Z}_q^m$, set $\mathbf{c}_2^\top = \mathbf{c}_1^\top \mathbf{R}_\mathbf{C}, c_3 = \mathbf{c}_1^\top \mathbf{r}_\mathbf{p}$. We write the full hybrid for clarity.

**Setup Phase.** $\mathcal{A}$ sends $(N, 1^L, \mathbb{A}, S, C)$. Initiate an empty dictionary $\mathsf{H}$. Do the following.

1. Sample $(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T}_\mathbf{B}) \leftarrow \widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma)$.
2. Sample $\mathsf{sd}_{\mathsf{sh}}, \mathsf{sd}_\rho \leftarrow \{0,1\}^\lambda$.
3. Use $C$ to generate a maximally invalid set $A'$ and sample random shares $\widetilde{\mathsf{sh}}_j \leftarrow \mathbb{Z}_q^n$ for $j \in A'$. Also sample a random secret $\mathbf{p} \leftarrow \mathbb{Z}_q^n$. Then, $(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}) = \mathsf{ProgShare}_2(\mathbb{A}, A', (\widetilde{\mathsf{sh}}_j)_{j \in A'}, \mathbf{p})$.
4. Now, compute a maximally invalid set $\widetilde{A}$ for $\mathbb{A}$ over $|S|$ parties like in $\mathsf{PreProcess}$. Then find the lexicographically smallest index $i^* \in [|S| \cdot L]$ that is not present in $\widetilde{A}$ and set for each $j \in A$, $\mathsf{H}[(\mathsf{sh}, \mathsf{sd}_{\mathsf{sh}}, j)] = \mathsf{sh}_j$.
5. For each $i \in C$, sample $\mathbf{R}_i \leftarrow \{\pm 1\}^{4m^5 \times L}$ and set $\mathsf{pk}_i := (i, \mathbf{B}\mathbf{R}_i), \mathsf{sk}_i := \mathbf{R}_i$. For each $i \in S \setminus C$, sample $\mathbf{T}_i = \mathbf{B}\mathbf{D}_i - \mathsf{sh}_i$ and set $\mathsf{pk}_i = (i, \mathbf{T}_i)$.
6. Compute matrix $\mathbf{X} = [\mathbf{X}_1 \mid \ldots \mid \mathbf{X}_{|S|}] \in \mathbb{Z}_q^{n \times L \cdot |S|}$ with $\mathbf{X}_j = \mathbf{T}_j + \mathsf{sh}_j$.
7. Compute $\mathbf{C} = \mathsf{MCOM.Com}(\mathbf{X}), \mathbf{Z}_\mathbf{X} = \mathsf{MCOM.Open}(\mathbf{X}), \mathbf{V}_{|S|} = \mathsf{MCOM.Verify}(1^{|S|})$.
8. Sample $\mathbf{R}_\mathbf{C} \leftarrow \{\pm 1\}^{4m^5 \times m}$, $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_\mathbf{C} - \mathbf{C}, \mathbf{Z}_0 = [\mathbf{Z}_{0,1} \mid \ldots \mid \mathbf{Z}_{0,|S|}] \in \mathbb{Z}_q^{m \times |S| \cdot L}$ as

$$\mathbf{Z}_{0,j} = \begin{cases} \mathbf{D}'_j - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j} & \text{if } j \in S \setminus C \\ \mathbf{D}'_j + \mathbf{R}_j - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j} & \text{if } j \in C \end{cases}$$

9. Sample $r' \leftarrow \mathsf{ER.Explain}(1^\lambda, 1^{T'}, \mathbf{C}_0, \mathbf{Z}_0, \sigma_{\mathsf{er}})$ and set $\mathsf{H}[(\rho, \mathsf{sd}_\rho, k)] = r'_k$ for each $k \in [\rho]$ and $\rho = \mathsf{poly}(\lambda, |S|)$.
10. Sample $\mathbf{c}_1^\top \leftarrow \mathbb{Z}_q^m, \mathbf{c}_2^\top = \mathbf{c}_1^\top \mathbf{R}_{\mathbf{C}}, c_3 = \mathbf{c}_1^\top \mathbf{r}_{\mathbf{p}} + \lfloor q/2 \rfloor b$.
11. Set $\mathsf{ct} := (\mathbf{c}_1^\top, \mathbf{c}_2^\top, c_3, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}})$.

Send $(\mathsf{crs} := (N, L, \mathbf{W}, \mathbf{D}, \mathbf{B}, \mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}), \{\mathsf{pk}_i\}_{i \in S}, \{\mathsf{sk}_j\}_{j \in C}, \mathsf{aux} := (\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathbf{p}), \mathsf{ct})$ to $\mathcal{A}$. Here, $\mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}$ are handles to random oracles.

**Random Oracle Queries.** Given a query $(\mathsf{sd}, i) \in \{0,1\}^\lambda, \{0,1\}^\lambda$ for $\mathcal{H}_\rho$ or $\mathcal{H}_{\mathsf{sh}}$, if the query is for $\mathcal{H}_\rho$, check if there is an entry for $(\rho, \mathsf{sd}, i)$ in $\mathsf{H}$. If yes, return that value. Otherwise, sample $\beta \leftarrow \{0,1\}$ and set $\mathsf{H}[(\rho, \mathsf{sd}, i)] = \beta$ and send $\beta$.

Otherwise, if the query is for $\mathcal{H}_{\mathsf{sh}}$, check if there is an entry for $(\mathsf{sh}, \mathsf{sd}, i)$ in $\mathsf{H}$. If yes, return that value. Otherwise, sample $\mathsf{sh} \leftarrow \mathbb{Z}_q^{n \times L}$ store $\mathsf{H}[(\mathsf{sh}, \mathsf{sd}, i)] = \mathsf{sh}$ and send $\mathsf{sh}$.

**Guess Phase.** $\mathcal{A}$ outputs its guess $b'$. Output $b' \stackrel{?}{=} b$.

**$\mathbf{Hyb}_{14}^{\mathcal{A}}(1^\lambda)$.** This is the same as $\mathbf{Hyb}_{13}^{\mathcal{A}}(1^\lambda)$ except that we sample $\mathbf{p} \leftarrow \mathbb{Z}_q^n$ and $c_3 \leftarrow \mathbb{Z}_q$. Note that in this hybrid all information about $b$ is gone.

**Lemma 6.3.** $\Pr[1 \leftarrow \mathbf{Hyb}_0^{\mathcal{A}}(1^\lambda)] = \Pr[1 \leftarrow \mathbf{Hyb}_1^{\mathcal{A}}(1^\lambda)]$.

*Proof.* These are just semantic changes. So, lemma holds. □

**Lemma 6.4.** $\Pr[1 \leftarrow \mathbf{Hyb}_1^{\mathcal{A}}(1^\lambda)] = \Pr[1 \leftarrow \mathbf{Hyb}_2^{\mathcal{A}}(1^\lambda)]$.

*Proof.* The only difference is that we sample the values of $(r_1, \ldots, r_\rho), \{\mathsf{sh}_j\}_{j \in A}$ from random oracle query phase to setup phase. As this is nothing more than a semantic change, lemma holds. □

**Lemma 6.5.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_2^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_3^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from generalized leftover hash lemma (Lemma 3.7). Specifically, as $\mathbf{B}$ is statistically close to a uniformly random matrix, $(\mathbf{B}, \mathbf{B}\mathbf{R}_i)$ is statistically close to $(\mathbf{B}, \mathbf{T}_i)$ where $\mathbf{T}_i \leftarrow \mathbb{Z}_q^{n \times L}$, $\mathbf{R}_i \leftarrow \{\pm 1\}^{4m^5 \times L}$. □

**Lemma 6.6.** $\Pr[1 \leftarrow \mathbf{Hyb}_3^{\mathcal{A}}(1^\lambda)] = \Pr[1 \leftarrow \mathbf{Hyb}_4^{\mathcal{A}}(1^\lambda)]$.

*Proof.* This follows from the mode indistinguishability property of PLSSS (Definition 4.10). □

**Lemma 6.7.** If $\mathsf{ER}$ satisfies explainability property, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for any $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_4^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_5^{\mathcal{A}}(1^\lambda)]| \leq 1/T' + \mathsf{negl}(\lambda)$.

*Proof.* The only difference between these two hybrids is that in $\mathbf{Hyb}_4^{\mathcal{A}}(1^\lambda)$, we do $(\mathbf{C}_0, \mathbf{Z}_0) = \mathsf{ER.Sample}(1^\lambda, 1^{|S|}, \sigma_{\mathsf{er}}; (r_1, \ldots, r_\rho))$ and program $(r_1, \ldots, r_\rho)$ in $\mathcal{H}_\rho$. In $\mathbf{Hyb}_5^{\mathcal{A}}(1^\lambda)$, we do $(\mathbf{C}_0, \mathbf{Z}_0) \leftarrow \mathsf{ER.Sample}(1^\lambda, 1^{|S|}, \sigma_{\mathsf{er}})$, $(r'_1, \ldots, r'_\rho) \leftarrow \mathsf{ER.Explain}(1^\lambda, 1^{T'}, \mathbf{C}_0, \mathbf{Z}_0, \sigma_{\mathsf{er}})$ and program $(r'_1, \ldots, r'_\rho)$ in $\mathsf{H}_\rho$. Thus, this lemma follows from explainability property of $\mathsf{ER}$. □

**Lemma 6.8.** If $\mathsf{MCOM}$ satisfies mode indistinguishability property, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for any $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_5^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_6^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from mode indistinguishability property of $\mathsf{MCOM}$. Also check Lemma A.5. □

**Lemma 6.9.** If $\mathsf{ER}$ satisfies randomness property, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for any $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_6^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_7^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* Note that as random coins used in $\mathsf{ER.Sample}$ aren't leaked in $\mathbf{Hyb}_6^{\mathcal{A}}(1^\lambda), \mathbf{Hyb}_7^{\mathcal{A}}(1^\lambda)$, this follows from the randomness property of $\mathsf{ER}$. □

**Lemma 6.10.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_7^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_8^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from generalized leftover hash lemma (Lemma 3.7). Specifically, as $\mathbf{B}$ is statistically close to random, $(\mathbf{B}, \mathbf{B}\mathbf{R}_\mathbf{C}, \mathbf{e}^\top \mathbf{R}_\mathbf{C})$ is statistically close to $(\mathbf{B}, \mathbf{C}_0, \mathbf{e}^\top \mathbf{R}_\mathbf{C})$ where $\mathbf{R}_\mathbf{C} \leftarrow \{\pm 1\}^{4m^5 \times m}, \mathbf{C}_0 \leftarrow \mathbb{Z}_q^{n \times m}$. □

**Lemma 6.11.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_8^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_9^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from Lemma A.7 as $\left\|\mathbf{R}_j - \mathbf{R}_\mathbf{C}\mathbf{V}_{|S|,j} - \mathbf{Z}_{\mathbf{X},j}\right\| \leq O(\|\mathbf{Z}_{\mathbf{X},j}\|) = O(\sigma m^{15/2} \log q \log N)$ and $\sigma_{\mathsf{er}} > \lambda^{\omega(1)} O(\sigma m^{13} \log q \log N)$, the lemma holds. □

**Lemma 6.12.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_9^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_{10}^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from Lemma A.6. Specifically, for $j \in C$, $(\mathbf{B}, \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathsf{sh}_j, \sigma_{\mathsf{er}}), \mathsf{sh}_j)$ is statistically close to $(\mathbf{B}, \mathbf{D}'_j, \mathbf{B}\mathbf{D}'_j)$ where $\mathbf{D}'_j \leftarrow \mathcal{D}_{\mathbb{Z}, \sigma_{\mathsf{er}}}^{4m^5 \times L}$ and $\mathsf{sh}_j \leftarrow \mathbb{Z}_q^{n \times L}$.

Similarly, for $j \in S \backslash C$, $(\mathbf{B}, \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathbf{T}_j + \mathsf{sh}_j, \sigma_{\mathsf{er}}), \mathbf{T}_j, \mathsf{sh}_j)$ is statistically close to $(\mathbf{B}, \mathsf{SamplePre}(\mathbf{B}, \mathbf{T}_\mathbf{B}, \mathbf{T}'_j, \sigma_{\mathsf{er}}), \mathbf{T}'_j - \mathsf{sh}_j, \mathsf{sh}_j)$ where $\mathbf{T}_j, \mathbf{T}'_j \leftarrow \mathbb{Z}_q^{n \times L}$. Which is then statistically close to $(\mathbf{B}, \mathbf{D}'_j, \mathbf{B}\mathbf{D}'_j - \mathsf{sh}_j, \mathsf{sh}_j)$ by Lemma A.6 where $\mathbf{D}'_j \leftarrow \mathcal{D}_{\mathbb{Z}, \sigma_{\mathsf{er}}}^{4m^5 \times L}$. □

**Lemma 6.13.** If $\mathsf{MCOM}$ satisfies mode indistinguishability property, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for any $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_{10}^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_{11}^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from mode indistinguishability property of $\mathsf{MCOM}$. Also check Lemma A.5. □

**Lemma 6.14.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_{11}^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_{12}^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from generalized leftover hash lemma (Lemma 3.7). Specifically, since $\mathbf{B}$ is statistically close to random, $(\mathbf{B}, \mathbf{B}\mathbf{r}_\mathbf{p}, \mathbf{e}^\top \mathbf{r}_\mathbf{p})$ is statistically close to $(\mathbf{B}, \mathbf{p}, \mathbf{e}^\top \mathbf{r}_\mathbf{p})$. □

**Lemma 6.15.** Assuming the hardness of $2m^2$-decomposed LWE assumption (Assumption 3.6), for any $\mathsf{PPT}$ adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}_{12}^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_{13}^{\mathcal{A}}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* Assume that there exists an adversary $\mathcal{A}$ such that $|\Pr[1 \leftarrow \mathbf{Hyb}_{12}^{\mathcal{A}}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}_{13}^{\mathcal{A}}(1^\lambda)]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$. Then, we will construct an adversary $\mathcal{B}$ that breaks the $2m^2$-decomposed assumption with non-negligible probability.

$\mathcal{B}$ receives $(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{c}^\top)$ from challenger. It then interacts with $\mathcal{A}$ and receives $(N, 1^L, \mathbb{A}, S, C)$. $\mathcal{B}$ initiates empty dictionary $\mathsf{H}$, samples $\mathsf{sd}_{\mathsf{sh}}, \mathsf{sd}_\rho \leftarrow \{0,1\}^\lambda$. Using $C$, $\mathcal{B}$ generates a maximally invalid set $A'$ and samples $\mathsf{sh}_j = \mathbf{B}\mathbf{D}'_j$, $\mathbf{D}'_j \leftarrow \mathcal{D}_{\mathbb{Z}, \sigma_{\mathsf{er}}}^{4m^5 \times L}$ for $j \in A'$. It also samples $\mathbf{p} = \mathbf{B}\mathbf{r}_\mathbf{p}, \mathbf{r}_\mathbf{p} \leftarrow \{\pm 1\}^{4m^5}$. Using these values, it generates $\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S|}$. Then, $\mathcal{B}$ generates maximally invalid share set $\widetilde{A}$ and adds one more $i^*$ to create $A$ using procedure from $\mathbf{Hyb}_{13}^{\mathcal{A}}(1^\lambda)$. It then programs $\mathsf{H}[(\mathsf{sh}, \mathsf{sd}_{\mathsf{sh}}, j)] = \mathsf{sh}_j$ for $j \in A$.

$\mathcal{B}$ now sample $\mathsf{pk}_i = (i, \mathbf{B}\mathbf{D}_i - \mathsf{sh}_i), \mathbf{D}_i \leftarrow \mathcal{D}_{\mathbb{Z}, \sigma_{\mathsf{er}}}^{4m^5 \times L}$ for $i \in S \setminus C$ and $\mathsf{pk}_i = (i, \mathbf{B}\mathbf{R}_i), \mathsf{sk}_i = \mathbf{R}_i \leftarrow \{\pm 1\}^{4m^5 \times m}$. $\mathcal{B}$ computes $\mathbf{X} = [\mathbf{T}_j + \mathsf{sh}_j \text{ for } j \in S]$, $\mathbf{C} = \mathsf{MCOM.Com}(\mathbf{X}), \mathbf{Z}_\mathbf{X} = \mathsf{MCOM.Open}(\mathbf{X})$, and $\mathbf{V}_{|S|} = \mathsf{MCOM.Verify}(1^{|S|})$. $\mathcal{B}$ samples $\mathbf{R}_\mathbf{C} \leftarrow \{\pm 1\}^{4m^5 \times m}$, computes $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_\mathbf{C} - \mathbf{C}$, $\mathbf{Z}_0$ as mentioned in $\mathbf{Hyb}_{11}^{\mathcal{A}}(1^\lambda)$. It will then sample $r'$ using $\mathsf{ER.Explain}$ and $T'$ and programs this in $\mathsf{H}$.

$\mathcal{B}$ now computes $\mathsf{ct} := (\mathbf{c}^\top, \mathbf{c}^\top \mathbf{R_C}, \mathbf{c}^\top \mathbf{r_p} + \lfloor q/2 \rfloor b, \mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}})$. It sends $\mathsf{crs} := (N, L, \mathbf{W}, \mathbf{D}, \mathbf{B}, \mathcal{H}_\rho, \mathcal{H}_{\mathsf{sh}}), \{\mathsf{pk}_i\}_{i \in S}$, $\{\mathsf{sk}_j\}_{j \in C}, \mathsf{aux} = (\mathsf{sd}_\rho, \mathsf{sd}_{\mathsf{sh}}, \mathbf{p}), \mathsf{ct}$ to $\mathcal{A}$. Any random oracle queries are handled as mentioned in $\mathbf{Hyb}^{\mathcal{A}}_{13}(1^\lambda)$.

If challenger's coin toss is 0, then $\mathcal{B}$ behaves like $\mathbf{Hyb}^{\mathcal{A}}_{12}(1^\lambda)$ and if the coin toss is 1, then $\mathcal{B}$ behaves like $\mathbf{Hyb}^{\mathcal{A}}_{13}(1^\lambda)$. Thus, $\mathcal{B}$ can break the $2m^2$-decomposed LWE assumption with probability $\epsilon(\lambda)$. □

**Lemma 6.16.** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for any adversary $\mathcal{A}$, for all $\lambda \in \mathbb{N}$, $|\Pr[1 \leftarrow \mathbf{Hyb}^{\mathcal{A}}_{13}(1^\lambda)] - \Pr[1 \leftarrow \mathbf{Hyb}^{\mathcal{A}}_{14}(1^\lambda)]| \leq \mathsf{negl}(\lambda)$.

*Proof.* This follows from generalized leftover hash lemma (Lemma 3.7). Specifically, since $\mathbf{B}$ is statistically close to random,

$$\left\{ \left( \begin{bmatrix} \mathbf{B} \\ \mathbf{c}_1^\top \end{bmatrix}, \begin{bmatrix} \mathbf{B} \\ \mathbf{c}_1^\top \end{bmatrix} \mathbf{r_p} \right) \right\} \approx_s \left\{ \left( \begin{bmatrix} \mathbf{B} \\ \mathbf{c}_1^\top \end{bmatrix}, \begin{bmatrix} \mathbf{p} \\ c_3 \end{bmatrix} \right) \right\}$$

□

**Deriving $T'$.** Let $b$-$\mathbf{Hyb}^{\mathcal{A}}_h(1^\lambda)$ denote $\mathbf{Hyb}^{\mathcal{A}}_h(1^\lambda)$ for $h \in [0, 14]$ from above where we encrypt $b \in \{0, 1\}$. Note that $\Pr[1 \leftarrow 0\text{-}\mathbf{Hyb}^{\mathcal{A}}_{14}(1^\lambda)] = \Pr[1 \leftarrow 1\text{-}\mathbf{Hyb}^{\mathcal{A}}_{14}(1^\lambda)]$. By triangle inequality, we have

$$\begin{aligned} |\Pr[1 \leftarrow 0\text{-}\mathbf{Hyb}^{\mathcal{A}}_0(1^\lambda)] - \Pr[1 \leftarrow 1\text{-}\mathbf{Hyb}^{\mathcal{A}}_0(1^\lambda)]| &\leq \sum_{i=1}^{14} |\Pr[1 \leftarrow 0\text{-}\mathbf{Hyb}^{\mathcal{A}}_{i-1}(1^\lambda)] - \Pr[1 \leftarrow 0\text{-}\mathbf{Hyb}^{\mathcal{A}}_i(1^\lambda)]| \\ &\quad + |\Pr[1 \leftarrow 1\text{-}\mathbf{Hyb}^{\mathcal{A}}_{i-1}(1^\lambda)] - \Pr[1 \leftarrow 1\text{-}\mathbf{Hyb}^{\mathcal{A}}_i(1^\lambda)]| \\ &\leq 2/T' + \mathsf{negl}(\lambda) \end{aligned}$$

If there exists an adversary such that $|\Pr[1 \leftarrow 0\text{-}\mathbf{Hyb}^{\mathcal{A}}_0(1^\lambda)] - \Pr[1 \leftarrow 1\text{-}\mathbf{Hyb}^{\mathcal{A}}_0(1^\lambda)]| = \epsilon(\lambda)$ for some non-negligible function $\epsilon(\cdot)$, then there exists a polynomial $T(\lambda)$ and infinite $\Lambda_{\mathcal{A}} \subseteq \mathbb{N}$ such that $\epsilon(\lambda) > 1/T(\lambda)$ for each $\lambda \in \Lambda_{\mathcal{A}}$. Then, we set $T'(\cdot) = 3T(\cdot)$. This would mean for each $\lambda \in \Lambda_{\mathcal{A}}$, $3/T'(\lambda) < \epsilon(\lambda) < 2/T'(\lambda) + \mathsf{negl}(\lambda)$ which is a contradiction. Thus, setting $T'(\cdot) = 3T(\cdot)$ suffices. □

# References


[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[ABN+22] Shweta Agrawal, Rajarshi Biswas, Ryo Nishimaki, Keita Xagawa, Xiang Xie, and Shota Yamada. Cryptanalysis of boyen's attribute-based encryption scheme in TCC 2013. *Designs, Codes and Cryptography*, 90(10):2301–2318, 2022.

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part I*, volume 15484 of *Lecture Notes in Computer Science*, pages 3–31, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

[AGHS13] Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete Gaussian leftover hash lemma over infinite domains. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 97–116, Bengalore, India, December 1–5, 2013. Springer Berlin Heidelberg, Germany.

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Key-homomorphic computations for RAM: Fully succinct randomised encodings and more. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 236–268, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[AR13] Divesh Aggarwal and Oded Regev. A note on discrete gaussian combinations of lattice vectors. *arXiv preprint arXiv:1308.2405*, 2013.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany.

[BC02] Amos Beimel and Benny Chor. Universally ideal secret-sharing schemes. *IEEE Transactions on Information Theory*, 40(3):786–794, 2002.

[BCF+25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. Beast-mev: Batched threshold encryption with silent setup for mev prevention. *Cryptology ePrint Archive*, 2025.

[BÇM21] Marshall Ball, Alper Çakan, and Tal Malkin. Linear threshold secret-sharing with binary reconstruction. In Stefano Tessaro, editor, *ITC 2021: 2nd Conference on Information-Theoretic Cryptography*, volume 199 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:22, Seattle, WA, USA, July 23–26, 2021. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[Bei96] Amos Beimel. Secure schemes for secret sharing and key distribution. *PhD thesis, Israel Institute of Technology, Technion*, 1996.

[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.

[BGI+01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

[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.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[BR07] Mihir Bellare and Phillip Rogaway. Robust computational secret sharing and a unified account of classical secret-sharing goals. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 172–184, 2007.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *34rd Annual International Cryptology Conference, CRYPTO 2014*, pages 480–499. Springer Verlag, 2014.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 3–34, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[CK93] Benny Chor and Eyal Kushilevitz. Secret sharing over infinite domains. *Journal of Cryptology*, 6(2):87–95, 1993.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part III*, volume 15366 of *Lecture Notes in Computer Science*, pages 156–189, Milan, Italy, December 2–6, 2024. Springer, Cham, Switzerland.

[CW26] Jeffrey Champion and David J. Wu. Distributed monotone-policy encryption for DNFs from lattices. Cryptology ePrint Archive, Paper 2026/318, To appear in Eurocrypt, 2026.

[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.

[DJWW26] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. In Goichiro Hanaoka and Bo-Yin Yang, editors, *Advances in Cryptology – ASIACRYPT 2025*, pages 130–162, Singapore, 2026. Springer Nature Singapore.

[DKW21] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority abe for dnf s from lwe. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 177–209. Springer, 2021.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540, Interlaken, Switzerland, May 2–6, 2004. Springer Berlin Heidelberg, Germany.

[EY24] Ehsan Ebrahimi and Anshu Yadav. Strongly secure universal thresholdizer. In Kai-Min Chung and Yu Sasaki, editors, *Advances in Cryptology – ASIACRYPT 2024, Part III*, volume 15486 of *Lecture Notes in Computer Science*, pages 207–239, Kolkata, India, December 9–13, 2024. Springer, Singapore, Singapore.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 498–531, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[GGH+16] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. *SIAM Journal on Computing*, 45(3):882–929, 2016.

[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.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part VII*, volume 14926 of *Lecture Notes in Computer Science*, pages 352–386, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions (extended abstracts). In *21st Annual ACM Symposium on Theory of Computing*, pages 12–24, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from lpn over f p, dlin, and prgs in nc 0. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 670–699. Springer, 2022.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part V*, volume 14442 of *Lecture Notes in Computer Science*, pages 407–441, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[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.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key fhe. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 735–763. Springer, 2016.

[Pei10] Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 80–97, Santa Barbara, CA, USA, August 15–19, 2010. Springer Berlin Heidelberg, Germany.

[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.

[RSY21] Leonid Reyzin, Adam Smith, and Sophia Yakoubov. Turning hate into love: compact homomorphic ad hoc threshold encryption for scalable mpc. In *International Symposium on Cyber Security Cryptography and Machine Learning*, pages 361–378. Springer, 2021.

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and lpn variants. *Cryptology ePrint Archive*, 2024.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*, pages 114–127. Springer, 2005.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\mathsf{depth}, \lambda)$-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part III*, volume 14922 of *Lecture Notes in Computer Science*, pages 178–209, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[Wee25] Hoeteck Wee. Almost optimal KP and CP-ABE for circuits from succinct LWE. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025, Part III*, volume 15603 of *Lecture Notes in Computer Science*, pages 34–62, Madrid, Spain, May 4–8, 2025. Springer, Cham, Switzerland.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption (poster presentation). In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 741–743, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025, Part III*, volume 16002 of *Lecture Notes in Computer Science*, pages 204–235, Santa Barbara, CA, USA, August 17–21, 2025. Springer, Cham, Switzerland.

[WW26] Brent Waters and David J. Wu. Silent threshold cryptography from pairings: Expressive policies in the plain model. Cryptology ePrint Archive, Report 2025/1547, To appear in Eurocrypt, 2026.

# A Additional Proofs

In this section, we provide additional proofs omitted in the main body.

**Proof of Lemma 3.7.** We prove this using the following claims.

**Claim A.1.** For positive integers $n, m$, and odd prime $q$, such that $m = \Omega(n \log q)$, for any $\ell = \mathsf{poly}(\lambda)$, $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times \ell m}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times \ell m}, \mathbf{B} = \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{D}) + (\mathsf{vec}(\mathbf{I}_\ell)^\top \otimes \mathbf{G})$, $\{(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{BR})\} \approx_s \{(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{A})\}$ where $\mathbf{R} \leftarrow \{\pm 1\}^{\ell^2 m \times L}, \sigma = \omega(\sqrt{\log m}), \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times L}$.

*Proof.* It suffices to prove that the distribution of $\mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{D})\mathbf{R}$ is statistically close to uniformly random distribution given $\mathbf{W}, \mathbf{D}$. It is known that linear combination of independent discrete Gaussian samples over a lattice are statistically close to a discrete Gaussian with appropriately increased width, under standard smoothing conditions [Pei10, AGHS13, AR13]. Thus, $(\mathbf{I}_\ell \otimes \mathbf{D})\mathbf{R}$ is a convolution of discrete Gaussian samples and as such is a sufficiently wide Gaussian. As $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times m}$ is a universal hash function, this follows from Lemma 3.2, Item 3. □

**Claim A.2.** For positive integers $n, m$, and odd prime $q$, such that $m = \Omega(n \log q)$, for any $\ell = \mathsf{poly}(\lambda)$ $\mathbf{W} \leftarrow \mathbb{Z}_q^{n \times \ell m}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times \ell m}, \mathbf{B} = \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{D}) + (\mathsf{vec}(\mathbf{I}_\ell))^\top \otimes \mathbf{G})$, $\{(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{BR}, \mathbf{e}^\top \mathbf{R})\} \approx_s \{(\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{A}, \mathbf{e}^\top \mathbf{R})\}$ where $\mathbf{R} \leftarrow \{\pm 1\}^{\ell^2 m \times L}, \sigma = \omega(\sqrt{\log m}), \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times L}$, $\mathbf{e}^\top \in \mathbb{Z}_q^{\ell^2 m}$ is any fixed vector.

*Proof.* This follows by a similar argument as Lemma 3.7 and [ABB10, Lemma 13]. □

**Lemma A.3** ([GPV08, MP12])**.** Let $n, m, q$ be lattice parameters such that $m = \Omega(n \log q)$, $q$ is an odd prime. Then, for $\sigma > \log m$, all but negligible fraction of matrices $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$,

$$\{(\mathbf{B}, \mathbf{B}_\sigma^{-1}(\mathbf{y}), \mathbf{y}) : \mathbf{y} \leftarrow \mathbb{Z}_q^n\} \approx_s \{(\mathbf{B}, \mathbf{z}, \mathbf{Bz}) : \mathbf{z} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^m\}$$

This can also be extended to matrices naturally by sampling column-wise.

**Lemma A.4** ([CHW25, WW25])**.** Let $n, m, q$ be lattice parameters such that $m = \Omega(n \log q)$, $q$ is an odd prime. Then, for any $\mathbf{y} \in \mathbb{Z}_q^n, \mathbf{z} \in \mathbb{Z}_q^m$, all but negligible fraction of matrices $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$,

$$\{(\mathbf{B}, \mathbf{B}_\sigma^{-1}(\mathbf{y} + \mathbf{Bz}))\} \approx_s \{(\mathbf{B}, \mathbf{B}_\sigma^{-1}(\mathbf{y}) + \mathbf{z})\}$$

where $\sigma > \lambda^{\omega(1)} \sqrt{m}\, \|\mathbf{z}\|$. This can also be extended to matrices naturally by sampling column-wise.

**Lemma A.5.** Let $n, m, q$ be lattice parameters such that $m = \Omega(n \log q)$, $q$ is an odd prime. Then, for $\sigma > \log m$,

$$\left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : \mathbf{W} \leftarrow \mathbb{Z}_q^{n \times 2m^3}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times 2m^3}, \mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G}\right\} \approx_s$$
$$\left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \widetilde{\mathsf{MCOM.Setup}}(1^n, 1^m, q, \sigma)\right\}$$

*Proof.* Parse $\mathbf{W}$ as $[\mathbf{W}_1 \mid \ldots \mid \mathbf{W}_{2m^2}] \in \mathbb{Z}_q^{n \times 2m^3}$, $\mathbf{D}$ as $[\mathbf{D}_1 \mid \ldots \mid \mathbf{D}_{2m^2}] \in \mathbb{Z}_q^{m \times 2m^3}$.

$$\left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : \mathbf{W} \leftarrow \mathbb{Z}_q^{n \times 2m^3}, \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times 2m^3}, \mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G}\right\}$$
$$\approx_s \left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : \begin{array}{l} (\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q), \\ \mathbf{W}_2 \mid \ldots \mid \mathbf{W}_{2m^2} \leftarrow \mathbb{Z}_q^{n \times (2m^2-1)m}, \\ \mathbf{D} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times 2m^3}, \\ \mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G} \end{array}\right\} \quad \text{(By matrix distribution of gadget trapdoors)}$$
$$\approx_s \left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : \begin{array}{l} (\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q), \\ [\mathbf{W}_2 \mid \ldots \mid \mathbf{W}_{2m^2}] \leftarrow \mathbb{Z}_q^{n \times m}, \\ \mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{D}_2 \leftarrow \mathsf{SamplePre}(\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}, \mathbf{A}, \sigma), \\ [\mathbf{D}_1 \mid \mathbf{D}_3 \mid \ldots \mid \mathbf{D}_{2m^2}] \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times (2m^2-1)m}, \\ \mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G} \end{array}\right\} \quad \text{(By Lemma A.3)}$$
$$\approx_s \left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : \begin{array}{l} (\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q), \\ [\mathbf{W}_2 \mid \ldots \mid \mathbf{W}_{2m^2}] \leftarrow \mathbb{Z}_q^{n \times m}, \\ (\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q), \\ \mathbf{D}_2 \leftarrow \mathsf{SamplePre}(\mathbf{W}_1, \mathbf{T}_{\mathbf{W}_1}, \mathbf{A}, \sigma), \\ [\mathbf{D}_1 \mid \mathbf{D}_3 \mid \ldots \mid \mathbf{D}_{2m^2}] \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m \times (2m^2-1)m}, \\ \mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{D}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\top \otimes \mathbf{G} \end{array}\right\} \quad \text{(By matrix distribution of gadget trapdoors)}$$
$$\equiv \left\{(\mathbf{W}, \mathbf{D}, \mathbf{B}) : (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \widetilde{\mathsf{MCOM.Setup}}(1^n, 1^m, q, \sigma)\right\} \quad (\text{where } \mathbf{T_B} = \mathbb{1}_{4m^4,2} \otimes \mathbf{T_A})$$

□

**Lemma A.6.** Let $n, m, q$ be lattice parameters such that $m = \Omega(n \log q)$, $q$ is an odd prime. Then, for $\sigma > \log m$,

$$\left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{y}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{y} \leftarrow \mathbb{Z}_q^n, \mathbf{d} \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T_B}, \mathbf{y}, \sigma) \end{array} \right\} \approx_s$$
$$\left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{Bd}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{d} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{4m^5} \end{array} \right\}$$

This can also be extended to matrices naturally by sampling column-wise

*Proof.* Parse $\mathbf{W}$ as $[\mathbf{W}_1 \mid \ldots \mid \mathbf{W}_{2m^2}] \in \mathbb{Z}_q^{n \times 2m^3}$, $\mathbf{D}$ as $[\mathbf{D}_1 \mid \ldots \mid \mathbf{D}_{2m^2}] \in \mathbb{Z}_q^{m \times 2m^3}$. Note that by construction, there exists a matrix $\mathbf{A}, \mathbf{T_A}$ inside $\mathbf{B}, \mathbf{T_B}$. Moreover, $\mathbf{A} = \mathbf{W}_1 \mathbf{D}_2$. Parse $\mathbf{d} \in \mathbb{Z}_q^{4m^5}$ as $\mathbf{d} = [\mathbf{d}_1^\top \mid \ldots \mid \mathbf{d}_{4m^4}^\top]^\top$. Let $\widetilde{\mathbf{y}} = \mathbf{B}(\mathbf{d} - \mathbb{1}_{4m^4,2} \otimes \mathbf{d}_2) \in \mathbb{Z}_q^n$ in each of these hybrids below.

$$\left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{Bd}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{d} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{4m^5} \end{array} \right\}$$
$$\equiv \left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{Ad}_2 + \widetilde{\mathbf{y}}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{d} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{4m^5} \end{array} \right\} \quad (\text{as } \widetilde{\mathbf{y}} + \mathbf{Ad}_2 = \mathbf{Bd})$$
$$\approx_s \left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{y}' + \widetilde{\mathbf{y}}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{y}' \leftarrow \mathbb{Z}_q^n, \mathbf{d}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^m, [\mathbf{d}_3^\top \mid \ldots \mid \mathbf{d}_{4m^4}^\top]^\top \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{(4m^4-2)m}, \\ \mathbf{d}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{y}', \sigma) \end{array} \right\} \quad (\text{By Lemma A.3})$$
$$\equiv \left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{y}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{y} \leftarrow \mathbb{Z}_q^n, \mathbf{d}_1 \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^m, [\mathbf{d}_3^\top \mid \ldots \mid \mathbf{d}_{4m^4}^\top]^\top \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{(4m^4-2)m}, \\ \mathbf{d}_2 \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T_A}, \mathbf{y} - \widetilde{\mathbf{y}}, \sigma) \end{array} \right\}$$
$$\approx_s \left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}, \mathbf{y}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{y} \leftarrow \mathbb{Z}_q^n, \mathbf{d} \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T_B}, \mathbf{y}, \sigma) \end{array} \right\} \quad (\text{By [ABB10]})$$

□

**Lemma A.7.** Let $n, m, q$ be lattice parameters such that $m = \Omega(n \log q)$, $q$ is an odd prime. Then, for any $\mathbf{y} \in \mathbb{Z}_q^n, \mathbf{z} \in \mathbb{Z}_q^{4m^5}$,

$$\left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{d} \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T_B}, \mathbf{y} + \mathbf{Bz}, \sigma) \end{array} \right\} \approx_s$$
$$\left\{ (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{d}) : \begin{array}{l} (\mathbf{W}, \mathbf{D}, \mathbf{B}, \mathbf{T_B}) \leftarrow \mathsf{MCOM.}\widetilde{\mathsf{Setup}}(1^n, 1^m, q, \sigma), \\ \mathbf{d} \leftarrow \mathsf{SamplePre}(\mathbf{B}, \mathbf{T_B}, \mathbf{y}, \sigma) + \mathbf{z} \end{array} \right\}$$

where $\sigma > \lambda^{\omega(1)} \sqrt{m} \, \|\mathbf{z}\|$. This can also be extended to matrices naturally by sampling column-wise.

*Proof.* This is similar to proof of Lemma A.6 except using Lemma A.4. □