# Optimal Distributed Monotone-Policy Encryption for DNFs and More from Lattices

Jeffrey Champion
UT Austin
jchampion@utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

**Abstract**

Distributed cryptography is a new cryptographic paradigm that enables fine-grained decryption capabilities in a trustless setting. In a distributed monotone-policy encryption scheme, users generate their own public and private keys. Thereafter, one can encrypt a message with respect to an arbitrary set of public keys together with an access policy. Any group of users that satisfies the access policy can recover the message; conversely, the message is computationally hidden from any group of users that does not satisfy the policy. The key requirement is succinctness: the size of the ciphertext should be sublinear in the size of the access policy. Distributed monotone-policy encryption generalizes related notions like distributed broadcast encryption (where the access policy is set membership) and silent threshold encryption (where the access policy is a threshold policy). In this work, we achieve the following:

- First, we give the first *optimal* distributed monotone-policy encryption scheme for the class of DNF policies from the decomposed LWE assumption in the random oracle model. Here, optimal means that the size of the public parameters, the user public keys, and the size of the ciphertext are independent of the size of the policy. As a corollary, we also obtain a (reusable) succinct computational secret sharing scheme for DNFs from decomposed LWE in the random oracle model.
- Next, we show how to adapt our techniques to obtain a distributed monotone-policy encryption scheme for $k$-DNFs in the *plain model* where the size of the ciphertext is $k \cdot L^{1/2}$, $k$ is the maximum size of each min-term, and $L$ is the number of min-terms in the DNF. This is the first scheme from the decomposed LWE assumption in the plain model. If we settle for a much weaker notion of selective security, then we also achieve full succinctness in the plain model (i.e., where the ciphertext size is independent of the size of the DNF).
- By specializing our results to the setting of broadcast encryption, we obtain an *adaptively-secure* distributed broadcast encryption scheme with ciphertext size $|S|^{2/3}$, where $|S|$ is the size of the broadcast set. Security relies on decomposed LWE (with a polynomial modulus-to-noise ratio) in the plain model. This scheme is the first lattice-based scheme with adaptive security that supports an a priori unbounded number of users in the plain model. Previous lattice-based distributed broadcast encryption schemes with adaptive security in the plain model assumed an a priori bound on the number of users (but achieved optimal-size ciphertexts that are independent of the size of the broadcast set).

## 1 Introduction

Distributed cryptography [WQZDF10, BZ14, GKPW24, DJWW25] is a cornerstone of *trustless* cryptographic systems. The goal of distributed cryptography is to augment public-key cryptography with fine-grained decryption capabilities, but in a decentralized way. Namely, in this model, users generate their own public

and private keys (as in standard public-key encryption). Later on, anyone can encrypt an arbitrary message to a subset of public keys $\{\mathsf{pk}_i\}_{i \in S}$ together with an access policy $P$. Any subset of users $T$ that satisfy the policy (e.g., $P(T) = 1$) can decrypt the ciphertext and recover the message. Security says that the message is computationally hidden from any set of users that do not satisfy the policy. The efficiency requirement is that the size of the ciphertext should be *sublinear* in the size of the policy $P$. Unlike the more classic centralized notions like attribute-based encryption [SW05, GPSW06] or functional encryption [BW07, BSW11, O'N10], there is no trusted key-issuing authority or long-term secret keys in distributed cryptography. Distributed cryptography thus provides a compelling method to combine fine-grained access control to encrypted data while simultaneously retaining the decentralized nature of classic public-key cryptography.

**The landscape of distributed cryptography.** Distributed monotone-policy encryption [DJWW25] corresponds to the general setting of distributed cryptography where the policy $P$ can be an arbitrary monotone function. Earlier primitives such as distributed broadcast encryption [WQZDF10, BZ14] and silent threshold encryption correspond to special cases where the policy is a set membership policy or a threshold policy, respectively. In recent years, there has been significant interest in building distributed monotone-policy encryption for different policy families from pairings [WQZDF10, KMW23, GKPW24, GHK+25, WW26], lattice-based assumptions [CW24, CHW25, AMR25, WW25, AMR26, GY26b, GY26a], as well as more general tools such as witness encryption [FWW23, DJWW25] and indistinguishability obfuscation [BZ14, RSY21, ADM+24, DJWW25]. If we focus our attention on lattice-based constructions, the landscape thus far has been limited to two classes of constructions:

- **Distributed broadcast encryption.** A long sequence of works [CW24, CHW25, AMR25, WW25, AMR26, GY26b, GY26a] have shown how to build distributed broadcast encryption from the succinct LWE [Wee24] or the decomposed LWE assumptions [AMR25] in both the plain model [CW24, WW25, AMR25, AMR26, GY26b] and the random oracle model [CHW25, GY26a]. As mentioned before, distributed broadcast encryption represents a simple case of distributed monotone-policy encryption for the set membership policy. The key property here is that decryption only needs to rely on a single user to participate. This restriction simplifies the design of distributed broadcast encryption schemes.
- **Distributed monotone-policy encryption for DNFs.** Very recently, a pair of works [CW26, AGY26] showed how to construct distributed monotone-policy encryption for policy families that require *multiple* users to come together to decrypt. Security of these schemes relies on the decomposed LWE assumption in the random oracle model. Both constructions support the class of DNF policies. They also support an unbounded number of users and have optimal ciphertext size (where the size of the ciphertext is independent of the policy size). However, the size of each user's public key scales *linearly* with the number of times the user's public key appears in the DNF policy. Thus, neither construction achieves optimal parameters.

## 1.1 Our Results

Existing constructions of lattice-based distributed monotone-policy encryption for DNFs [CW26, AGY26] first design a scheme that supports any policy with a linear secret sharing scheme, and then observe that the security proof demands the underlying secret sharing scheme satisfy an additional linear independence property. This extra property limits the final construction to the class of DNFs. In this work, we introduce a new deferred public-key generation technique (see Section 1.2) that enables a number of new results, which we summarize below.

| Scheme | Assumption | \|pp\| | \|pk\| | \|sk\| | \|ct\| | PM | AP | RKA |
|---|---|---|---|---|---|---|---|---|
| [DJWW25] | witness encryption* | 1 | 1 | 1 | $k$ | ✓ | ✓ | ✓ |
| [CW26] | decomposed LWE | 1 | $M$ | 1 | 1 | ✗ | ✗ | ✓ |
| [AGY26] | decomposed LWE | 1 | $M$ | 1 | 1 | ✗ | ✗ | ✗ |
| **Corollary 3.19** | decomposed LWE | 1 | 1 | 1 | 1 | ✗ | ✓ | ✓ |
| **Corollary 4.4**† | decomposed LWE | 1 | 1 | 1 | $k \cdot L^{1/2}$ | ✓ | ✓ | ✓ |
| **Corollary B.14** | decomposed LWE | 1 | $M$ | 1 | 1 | ✓ | ✗ | ✗ |

*This construction additionally relies on leveled homomorphic encryption (which can be built from LWE).
†In Remark 4.5, we sketch an alternative way to re-balance the optimal DMPE scheme to obtain ciphertexts of size $\max\{(kL)^{1/2}, k\}$.

Table 1: Comparison with previous distributed monotone-policy encryption schemes for DNF policies. For each scheme, we report the size of the public parameters pp, the user public key pk, the user secret key sk, and the ciphertext ct as a function of the maximum size $k$ of a min-term in the policy, the total number of min-terms $L$ in a policy, and the total number of times $M$ a user's public key can appear in the policy. For ease of comparison, we suppress $\mathsf{poly}(\lambda)$ factors (as well as polylogarithmic factors), where $\lambda$ is the security parameter. For each scheme, we also indicate whether the construction is secure in the plain model (PM), whether the scheme is secure against adversaries that can adaptively choose the policy (AP), and whether the scheme is shown to be robust against rogue-key attacks (RKA). All listed schemes support a transparent setup and operate in a static corruption model where the adversary cannot adaptively corrupt honest users (i.e., the adversary cannot learn the secret keys associated with honest users).

**Optimal distributed monotone-policy encryption for DNFs in the random oracle model.** First, we show how to construct an *optimal* distributed monotone-policy encryption scheme for DNFs from the decomposed LWE assumption [AMR25] in the random oracle model. This is a scheme where the size of the CRS, the size of the user public keys, and the size of the ciphertext are all independent of the policy size. Previous constructions [CW26, AGY26] (also based on decomposed LWE in the random oracle model) were sub-optimal in that the size of each user's public key scales with a bound on the number of times the user appears in the policy. In particular, this implies a silent threshold encryption scheme [GKPW24] that supports constant thresholds. Prior constructions based on pairings [GKPW24, WW26] or decomposed LWE [CW26, AGY26] require long user public keys that scale linearly (or worse) with the size of the decryption quorum.

Our techniques also allow us to prove security against adversaries that can *adaptively* choose the policy. Previous constructions considered weaker models that require the adversary to declare both the challenge policy as well as the set of corrupted users either at the beginning of the security game [AGY26] or immediately after seeing the public parameters and before making any key-generation or hint-generation queries [CW26]. We give our construction in Construction 3.1 and Corollary 3.19 and refer to Table 1 for a comparison with prior work.

**A corollary: succinct computational secret sharing for DNFs.** An immediate corollary of our optimal distributed monotone-policy encryption scheme for DNFs is that we obtain a *reusable* succinct computational secret sharing (CSS) scheme [ABI+23] for DNF policies from decomposed LWE in the random oracle model. In succinct CSS, given an access structure $f\colon \{0,1\}^n \to \{0,1\}$ defined over $n$ parties and a message $m$, the goal is to output $n$ shares, where the size of the largest share is significantly smaller,

and preferably, independent of the size of some natural computational representation of $f$. As noted in [ABI+23], succinct CSS for DNFs implies a succinct CSS for truth tables (i.e., a scheme that given an arbitrary access structure $f$, represented as a truth table of size $N = 2^n$, produces shares of size $\mathrm{poly}(\log N)$ in time $\mathrm{poly}(N)$). Prior to this work, succinct CSS for DNFs was only known from indistinguishability obfuscation [HIJ+16, BCG+19, ASY22, LNW25, JJMP25] in the random oracle model. For the restricted case of $k$-DNFs where each min-term in the DNF contains at most $k$ variables, the work of [DJWW25] gives a construction from witness encryption in the random oracle model where the share size grows with $k$. Our construction supports general DNF policies. The succinct CSS constructions from [CW26, AGY26] have share sizes that grow with the maximum number of times a variable appears in the policy. When used to support policies like a truth table, this can lead to share size that scales with $\mathrm{poly}(N)$, which is no longer succinct. We summarize our succinct CSS scheme for DNF policies in Corollary 3.21.

**Distributed monotone-policy encryption for DNFs in the plain model.** The structure of our optimal distributed monotone-policy encryption for DNFs also lends itself naturally to a scheme in the plain model where the ciphertexts scale *linearly* with the size of the policy. Specifically, to support $k$-DNFs with $L$ min-terms, the ciphertext size is $k \cdot L$. This by itself is uninteresting since such succinctness is trivially achievable by composing a non-succinct secret sharing scheme for DNFs (e.g., [BL88]) with public-key encryption. However, we show that we can *re-balance* our construction to achieve ciphertext size $k \cdot L^{1/2}$ in the plain model.[1] This is the first construction of distributed monotone-policy encryption for DNFs in the plain model from decomposed LWE. Prior to this work, the only plain-model constructions relied either on witness encryption or indistinguishability obfuscation [DJWW25]. We give our construction in Section 4 and Corollary 4.4.

Here, we also note that if we settle for the *weaker* selective notion of security proposed in [AGY26], then we can in fact achieve optimal ciphertext size from decomposed LWE in the plain model. Critically, the [AGY26] security definition does not allow the adversary to adaptively choose public keys as a function of the honest users' public keys. As we show, achieving this definition turns out to be substantially easier and a simplified variant of our optimal construction satisfies it in the plain model. However, the resulting scheme we obtain is completely insecure against an adversary that can choose public keys as a function of the honest users' keys (i.e., it is trivially broken under a rogue-key attack [RY07, BDN18]). This demonstrates the pitfalls of considering relaxed security definitions that do not properly capture rogue-key attacks, as here, we give an example of a (non-contrived) scheme that satisfies selective security and yet is vulnerable to a rogue-key attack. We refer to [CHW25, §1.2] for similar discussion in the context of registered ABE and give our formal construction in Construction B.2 and Corollary B.14.

**Adaptively-secure unbounded distributed broadcast encryption in the plain model.** Finally, we observe that our re-balanced distributed monotone-policy encryption for DNFs in the plain model also implies a distributed broadcast encryption scheme with *semi-static security* [GW09] in the plain model where the ciphertext for broadcasting to a set $S$ scales with $O(|S|^{1/2})$. Notably, this base scheme still supports an a priori unbounded number of users. We then show that if we are not aiming for optimal succinctness, then the Gentry-Waters compiler [GW09, KMW23] can be used to lift a semi-statically-secure broadcast encryption scheme into an adaptively-secure scheme in the *plain model*.[2] Taken together, we obtain the first *unbounded* adaptively-secure distributed broadcast encryption from decomposed LWE in the plain

[1] As noted in Remark 4.5, the parameters can be set better to get ciphertext size $\max\{(kL)^{1/2}, k\}$.

[2] The original Gentry-Waters compiler achieved optimal succinctness, but needed to rely on the random oracle model.

| Scheme | Assumption | \|pp\| | \|pk\| | \|sk\| | \|ct\| | TP | AD | PM |
|---|---|---|---|---|---|---|---|---|
| [CW24] | succinct LWE | $N^2$ | $N$ | 1 | 1 | ✗ | ✗ | ✓ |
| [CHW25] | succinct LWE | $N^2$ | $N$ | 1 | 1 | ✗ | ✓ | ✗ |
| [AMR25] | decomposed LWE | $N$ | 1 | 1 | 1 | ✓ | ✗ | ✓ |
| [WW25] | decomposed LWE | 1 | 1 | 1 | 1 | ✓ | ✗ | ✓ |
| [AMR26] | decomposed LWE | $N$ | 1 | 1 | 1 | ✓ | ✓ | ✓ |
| [GY26b, GY26a]† | decomposed LWE | 1 | 1 | 1 | 1 | ✓ | ✓ | ✗ |
| [GY26b] | decomposed LWE | $N^2$ | 1 | 1 | 1 | ✓ | ✓ | ✓ |
| **Corollary 5.11*** | decomposed LWE | 1 | 1 | 1 | $\|S\|^{2/3}$ | ✓ | ✓ | ✓ |
| **Remark 5.13†** | decomposed LWE | 1 | 1 | 1 | 1 | ✓ | ✓ | ✗ |

*In Remark 5.12, we describe a plausible route to reduce the ciphertext size to $|S|^{1/2}$ via a non-black-box composition of our re-balancing techniques.

† The work of [GY26a] gives the first adaptively-secure flexible broadcast encryption scheme with optimal parameters from decomposed LWE with a *super-polynomial* modulus-to-noise ratio in the random oracle model. In Remark 5.13, we show how our results immediately imply an adaptively-secure flexible broadcast encryption scheme with optimal parameters from decomposed LWE with a *polynomial* modulus-to-noise ratio in the random oracle model.

Table 2: Comparison with previous lattice-based distributed broadcast encryption schemes. For each scheme, we report the size of the public parameters pp, the user public key pk, the user secret key sk, and the ciphertext ct as a function of the maximum number of users $N$ in the system and the size of the broadcast set $|S|$. For ease of comparison, we suppress $\mathsf{poly}(\lambda)$ factors (as well as polylogarithmic factors), where $\lambda$ is the security parameter. For each scheme, we also indicate whether the public parameters pp can be sampled with a *transparent* setup (TP), whether the scheme is proven to be adaptively secure (AD), and whether the construction is secure in the plain model (PM).

model where the ciphertext size is $O(|S|^{2/3})$.[3] Previous plain-model constructions of adaptively-secure distributed broadcast encryption schemes from decomposed LWE [GY26b, AMR26] can only support an a priori bounded number of users, albeit with optimal ciphertext size. We also note that in the random oracle model, we obtain an adaptively-secure flexible broadcast encryption scheme[4] with optimal parameters from decomposed LWE with a *polynomial* modulus in the random oracle model. This improves upon the recent work of [GY26a] that achieved the same result from decomposed LWE with a *super-polynomial* modulus in the random oracle model. We provide a full comparison in Table 2.

Another appealing feature is that we are able to achieve adaptive security without any new machinery (e.g., the work of [GY26b] relies on programmable pseudorandom set samplers while the work of [AMR26] relies on a new homomorphic evaluation machinery for non-deterministic functions). Our construction only relies on the matrix commitment machinery [Wee25] underlying the distributed broadcast encryption scheme from [WW25].

We also note that the work of [AMR26] shows how to construct a semi-statically-secure distributed broadcast encryption with *optimal* parameters from decomposed LWE in the plain model. To get adaptively-

[3] As we note in Remark 5.12, a *non-black-box* composition of our constructions can plausibly be used to reduce the ciphertext size to $O(|S|^{1/2})$.

[4] Flexible broadcast encryption [FWW23] is a generalization of distributed broadcast encryption that supports encryption to arbitrary collections of public keys. In distributed broadcast encryption, user public keys are additionally associated with an index and encryption is only possible to a set of public keys associated with distinct indices. We refer to Remark 5.4 for more details.

secure broadcast encryption, they then rely on the work of [HWW25], which imposes an a priori bound on the number of users. If we apply our variant of the Gentry-Waters compiler to their scheme, we obtain an unbounded adaptively-secure distributed broadcast encryption scheme in the plain model with ciphertext size $O(|S|^{1/2})$. Note, however, that the [AMR26] construction relies on decomposed LWE with a *super-polynomial* modulus-to-noise ratio. In contrast, our construction can be instantiated with a polynomial modulus.

## 1.2 Technical Overview

Similar to the distributed monotone-policy encryption schemes from [CW26, AGY26], our starting point in this work is the Wee-Wu distributed broadcast encryption scheme from [WW25]. We start by recalling the structure of this scheme. Let $n, m, q$ be lattice parameters where $m = O(n \log q)$. Throughout the overview, we use curly underlines to suppress low-norm errors. Specifically, we write $\underbrace{\mathbf{s}^\mathsf{T}\mathbf{A}}$ to denote $\mathbf{s}^\mathsf{T}\mathbf{A} + \mathbf{e}^\mathsf{T}$ where $\mathbf{e}^\mathsf{T}$ is a low-norm vector over $\mathbb{Z}_q$.

**Matrix commitments.** The key building block underlying our construction is the matrix commitment from [Wee25]. Specifically, a matrix commitment to a matrix $\mathbf{M} \in \mathbb{Z}_q^{n\times N}$ is a matrix $\mathbf{C} \in \mathbb{Z}_q^{n\times m}$ where

$$\mathbf{C} \cdot \mathbf{V}_N = \mathbf{M} - \mathbf{B} \cdot \mathbf{Z}, \tag{1.1}$$

where $\mathbf{B} \in \mathbb{Z}_q^{n\times 4m^5}$, $\mathbf{V}_N \in \mathbb{Z}_q^{m\times N}$ are public matrices, $\mathbf{V}_N$ is low-norm, and $\mathbf{Z} \in \mathbb{Z}_q^{4m^5\times N}$ is a low-norm opening. There are efficient algorithms for computing the commitment $\mathbf{C}$ and the opening $\mathbf{Z}$ from a set of public parameters $\mathsf{pp}_{\mathsf{com}}$ and the input matrix $\mathbf{M}$. These public parameters also specify the matrices $\mathbf{B}$ and $\mathbf{V}_N$. In the case where $\mathbf{M}$ is a *sparse* matrix, the work of [WW25] describes an efficient algorithm to compute any individual column of the opening $\mathbf{Z}$ in time that only scales with the number of non-zero columns in $\mathbf{M}$ and not the width of $\mathbf{M}$; similarly, there is an efficient algorithm for computing the $i^{\text{th}}$ column of $\mathbf{V}_N$ given just the width $N$ and the index $i$. Finally, under the decomposed learning with errors (decomposed LWE) assumption [AMR25], the following distributions are computationally indistinguishable:

$$(\mathsf{pp}_{\mathsf{com}}, \underbrace{\mathbf{s}^\mathsf{T}\mathbf{B}}) \quad \text{and} \quad (\mathsf{pp}_{\mathsf{com}}, \mathbf{u}^\mathsf{T}) \quad \text{where} \quad \mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{u} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{4m^5}.$$

In other words, decomposed LWE states that the plain LWE assumption [Reg05] holds with respect to the matrix $\mathbf{B}$ even given the public parameters $\mathsf{pp}_{\mathsf{com}}$ of a matrix commitment scheme.

**The Wee-Wu distributed broadcast encryption scheme.** Next, we give a brief overview of the structure of the distributed broadcast encryption scheme from [WW25]. In distributed broadcast encryption [WQZDF10, BZ14], each user is associated with an index $i \in [N]$. In this setting, each user $i$ can generate a public key $\mathsf{pk}_i$ and a secret key $\mathsf{sk}_i$ independently of all other users (but with reference to a global set of public parameters). There is a public encryption algorithm that takes as input the public keys $\{\mathsf{pk}_i\}_{i\in S}$ for a group of users along with a message and outputs a ciphertext ct. Any user $i \in S$ can then use their secret key $\mathsf{sk}_i$ to decrypt the ciphertext and recover the message, whereas the message should be computationally hidden from users $i \notin S$. Finally, the efficiency requirement is that the size of the ciphertext should be sublinear in (and ideally, independent of) the size of the broadcast set $S$. We now describe a variant of the distributed broadcast encryption scheme from [WW25]:

- **Public parameters:** The public parameters for the encryption scheme $\mathsf{pp} = (\mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ consist of the public parameters for a matrix commitment scheme, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ for re-randomization, and a public key $\mathbf{p} \in \mathbb{Z}_q^n$ for a dual Regev public-key encryption scheme (cf. [GPV08]). Recall from above that the commitment parameters $\mathsf{pp}_{\mathsf{com}}$ define a matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times 4m^5}$.

- **Key generation:** To generate a key, user $i$ samples $\mathbf{y}_i \xleftarrow{R} \{0,1\}^{4m^5}$ and defines their public key to be $\mathsf{pk}_i := \mathbf{d}_i = \mathbf{B}\mathbf{y}_i - \mathbf{A}\mathbf{v}_{N,i}$, where $\mathbf{v}_{N,i} \in \mathbb{Z}_q^m$ denotes the $i^{\text{th}}$ column of the matrix $\mathbf{V}_N$ from Eq. (1.1).
- **Encryption:** To encrypt a bit $\mu \in \{0,1\}$ to a set of public keys $\{\mathsf{pk}_i\}_{i \in S}$ where $\mathsf{pk}_i = \mathbf{d}_i$ and $S \subseteq [N]$, the encrypter first defines the (sparse) matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$ where the $i^{\text{th}}$ column of $\mathbf{M}$ is $\mathbf{d}_i + \mathbf{p}$ if $i \in S$ and $\mathbf{0}^n$ otherwise. Let $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ be a commitment to $\mathbf{M}$. The encrypter then samples an LWE secret $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ and outputs the ciphertext
$$\mathsf{ct} = \left(\mathbf{s}^\intercal \mathbf{B}\ ,\ \mathbf{s}^\intercal(\mathbf{A}+\mathbf{C})\ ,\ \mathbf{s}^\intercal \mathbf{p} + \mu \cdot \lfloor q/2 \rfloor\right).$$
- **Decryption:** User $i \in S$ can decrypt by first computing the $i^{\text{th}}$ column $\mathbf{z}_i$ of the opening $\mathbf{Z}$. Specifically, from Eq. (1.1), this means
$$\mathbf{C}\mathbf{v}_{N,i} = (\mathbf{d}_i + \mathbf{p}) - \mathbf{B}\mathbf{z}_i = \mathbf{B}\mathbf{y}_i + \mathbf{p} - \mathbf{A}\mathbf{v}_{N,i} - \mathbf{B}\mathbf{z}_i.$$
Recall that $\mathbf{v}_{N,i}$ and $\mathbf{z}_i$ are both low norm. The user can now recover $\mathbf{s}^\intercal \mathbf{p}$ by computing
$$\mathbf{s}^\intercal(\mathbf{A}+\mathbf{C}) \cdot \mathbf{v}_{N,i} + \mathbf{s}^\intercal \mathbf{B} \cdot (\mathbf{z}_i - \mathbf{y}_i) \approx \mathbf{s}^\intercal \mathbf{A}\mathbf{v}_{N,i} + \mathbf{s}^\intercal \mathbf{B}\mathbf{y}_i + \mathbf{s}^\intercal \mathbf{p} - \mathbf{s}^\intercal \mathbf{A}\mathbf{v}_{N,i} - \mathbf{s}^\intercal \mathbf{B}\mathbf{z}_i + \mathbf{s}^\intercal \mathbf{B}\mathbf{z}_i - \mathbf{s}^\intercal \mathbf{B}\mathbf{y}_i = \mathbf{s}^\intercal \mathbf{p}.$$
In combination with $\mathbf{s}^\intercal \mathbf{p} + \mu \cdot \lfloor q/2 \rfloor$, the user can subtract and round to recover $\mu$.

**Extending to distributed monotone-policy encryption.** The recent works of [CW26, AGY26] show how to extend the above template to a distributed monotone-policy encryption scheme for DNF policies. Recall that in this model, the encryption algorithm takes a collection of public keys $\{\mathsf{pk}_i\}_{i \in [N]}$ and a DNF policy $\varphi$ over those public keys (where each public key appears at most once in the DNF). Any subset of users $T \subseteq [N]$ where $\varphi(T) = 1$ should be able to use their individual secret keys to decrypt and recover the message. The approach taken in [CW26, AGY26] is roughly as follows:

- The public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ contain the matrix commitment parameters and the dual-Regev public key $\mathbf{p} \in \mathbb{Z}_q^n$. The translation matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is now sampled at encryption time.
- Each user's public key $\mathsf{pk}_i$ is simply $\mathbf{d}_i = \mathbf{B}\mathbf{y}_i$ (*without* the $\mathbf{A}\mathbf{v}_{N,i}$ term). In particular, users' public keys are no longer bound to a slot index.
- At encryption time, the encrypter first *secret shares* the target vector $\mathbf{p} \in \mathbb{Z}_q^n$ according to the policy $\varphi$. Let $\mathbf{p}_1, \ldots, \mathbf{p}_N \in \mathbb{Z}_q^n$ be the resulting shares (which are also included as part of the ciphertext). Next, the encrypter commits to the matrix $\mathbf{M}$ where the $i^{\text{th}}$ column is $\mathbf{d}_i + \mathbf{p}_i$. In addition, the encryption algorithm samples a translation matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with low-norm pre-images $\mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,i})$ for all $i \in [N]$. The work of [CHW25] shows how to jointly sample $(\mathbf{A}, \mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,1}), \ldots, \mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,N}))$ using $\mathsf{pp}_{\mathsf{com}}$. Specifically, we can view $\mathbf{A}$ as a *randomized* matrix commitment to the all-zeroes matrix. The recoding vectors $\mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,i})$ are needed to enable decryption.

With these modifications, each user $i$ can use their secret key and obtain $\mathbf{s}^\intercal \mathbf{p}_i$ via a similar decryption procedure as described above. We can view this as a noisy share of $\mathbf{s}^\intercal \mathbf{p}$. Given sufficiently many shares, the users are able to recover $\mathbf{s}^\intercal \mathbf{p}$, and from there, the message. As described so far, the size of the ciphertext is linear in $N$, which is no better than the trivial construction. To obtain shorter ciphertexts, the works of [CW26, AGY26] compress the ciphertext with the aid of the random oracle:

- First, the shares $\mathbf{p}_1, \ldots, \mathbf{p}_N$ of the target vector $\mathbf{p}$ are derived using the random oracle. The ciphertext only needs to include the seed used to derive these shares. The security analysis in turn relies on the fact that we can construct an "explainable" secret sharing scheme for the class of DNF policies.
- Second, the matrix $\mathbf{A}$ and the low-norm pre-images $\mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,i})$ for all $i \in [N]$ are also derived from the random oracle (specifically, the random oracle outputs the randomness for a sampling algorithm that outputs $\mathbf{A}$ together with the pre-images). The explainable sampling procedure of [CHW25, WW25] shows how to use a uniform random string to sample (as well as reverse sample) $(\mathbf{A}, \mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,1}), \ldots, \mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{N,N}))$ from the commitment parameters $\mathsf{pp}_{\mathsf{com}}$.

The above template assumes that each user's public key appears once in the DNF policy. To extend to the setting where the user's public key appears up to $K$ times in the policy, the works of [CW26, AGY26] rely on virtualization where the user generates $K$ different public keys $\mathbf{d}_{i,1}, \ldots, \mathbf{d}_{i,K}$, and $\mathbf{d}_{i,j}$ is used for the $j^{\text{th}}$ occurrence of the user's public key in the policy. Observe that this approach assumes an a priori bound on the number of times each user appears in a given policy, and moreover, the size of the user's public key scales linearly with $K$. For this reason, these works do not achieve optimal parameters for DNF policies.

**This work: deferred public-key generation.** In the blueprint described above, the encryption algorithm begins by committing to the public keys of the users appearing in the decryption policy. The central idea in this work is to introduce an extra layer of indirection and design a mechanism that allows the encrypter to "refresh" each user's public key at encryption time. This refreshing mechanism allows us to (1) build an *optimal* distributed monotone-policy encryption scheme for DNFs; and (2) prove security against adversaries that can *adaptively* choose the challenge policy and make key-generation queries. We proceed as follows:

- Instead of sampling a vector $\mathbf{y}_i \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ as their secret key, each user in the system now samples a matrix $\mathbf{Y}_i \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$. Their public key is the *matrix* $\mathbf{T}_i = \mathbf{B}\mathbf{Y}_i \in \mathbb{Z}_q^{n \times m}$.
- By taking the public key to be a matrix $\mathbf{T}_i \in \mathbb{Z}_q^{n \times m}$, the encrypter can derive *fresh* public keys for user $i$ on the fly at encryption time. Specifically, suppose that user $i$'s public key appears $K$ times in the policy. At encryption time, the encrypter will sample $K$ low-norm vectors $\mathbf{u}_{i,1}, \ldots, \mathbf{u}_{i,K} \in \mathbb{Z}_q^m$ and compute $K$ sub-keys $\mathbf{d}_{i,1}, \ldots, \mathbf{d}_{i,K}$ where $\mathbf{d}_{i,j} = \mathbf{T}_i\mathbf{u}_{i,j}$. In addition, the encrypter includes the low-norm vectors $\mathbf{u}_{i,j}$ with the ciphertext. As written, the size of the ciphertext now scales with the size of the policy, and thus, no longer satisfies succinctness. To compress, we use the random oracle to derive the low-norm vectors $\mathbf{u}_{i,j} = H(\gamma, i, j)$, where $\gamma$ is a short seed included with the ciphertext.
- Correctness relies on the fact that user $i$ still has a low-norm preimage of each $\mathbf{d}_{i,j}$ with respect to $\mathbf{B}$. Specifically, by definition, $\mathbf{d}_{i,j} = \mathbf{T}_i\mathbf{u}_{i,j} = \mathbf{B}(\mathbf{Y}_i\mathbf{u}_{i,j})$, so the vector $\mathbf{Y}_i\mathbf{u}_{i,j} \in \mathbb{Z}_q^{4m^5}$ is the secret key associated with $\mathbf{d}_{i,j}$. Correctness now proceeds as described above.

This approach immediately gives a distributed monotone-policy encryption scheme for DNFs where the size of the user's public key is independent of the number of times the user appears in the policy. Essentially, the encrypter can effectively sample a fresh public key for each occurrence of the user in the policy.

The ability to re-randomize public keys at *encryption* time is also helpful from the perspective of the security analysis. Specifically, if we consider the static corruption model where the adversary cannot corrupt honest parties to learn their secret key,[5] the reduction algorithm can sample the public key for

[5]Many lattice-based constructions of distributed broadcast encryption [CW24, WW25] and all constructions of distributed monotone-policy encryption from lattices are analyzed in the static corruption model [CW26, AGY26].

user $i$ to be a random matrix $\mathbf{T}_i \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $\mathsf{td}_i$. The trapdoor gives the reduction the ability to program the vectors $\mathbf{d}_{i,j}$ to arbitrary values of its choosing by using the trapdoor to sample $\mathbf{u}_{i,j} \leftarrow \mathbf{T}_i^{-1}(\mathbf{d}_{i,j})$. Since this programming can now occur at *encryption* time as opposed to key-generation time, we are able to prove security even if the policy is chosen *adaptively* (and in the case of distributed broadcast encryption, we obtain a scheme with semi-static security). Previous approaches [CW26, AGY26] relied on the ability to program the public keys for honest users. Since this programming must happen at key-generation time, these works required the adversary to declare its challenge policy prior to making key-generation queries. Our technique simultaneously enables stronger security and optimal parameters.

**A simpler method to support DNF policies.** While we believe we can prove security of the above template from decomposed LWE in the random oracle model (following [CW26, AGY26]), we describe here a simpler variant that is more tailored to the access structure of DNF policies. The advantage of the simpler variant is that it straightforwardly yields a *plain model* construction with re-balancing.

Specifically, while the above template is described in terms of general linear secret sharing schemes, security only holds for the subclass of DNF policies. In this work, we show that if we design a scheme specifically for DNF policies, then we can obtain a simpler construction. In this work, we model a DNF policy $\varphi$ with $L$ min-terms over $N$ variables as a tuple of $L$ sets $(S_1, \ldots, S_L)$ where $S_i \subseteq [N]$ is the set of variables associated with the $i^{\text{th}}$ min-term. Suppose that $\mathbf{d}_1, \ldots, \mathbf{d}_N \in \mathbb{Z}_q^n$ are the user public keys associated with the $N$ variables and suppose $\mathbf{y}_1, \ldots, \mathbf{y}_N$ are their associated secret keys.

The main observation is that to satisfy the policy $\varphi$, the decryption quorum should possess the secret keys $\mathbf{y}_i$ for all $i \in S_j$ for some $j \in [L]$. In effect then, we can define the public key $\tilde{\mathbf{d}}_j = \sum_{i \in S_j} \mathbf{d}_i$ associated with the $j^{\text{th}}$ min-term of $\varphi$, where the associated secret key is $\sum_{i \in S_j} \mathbf{y}_i$. At this point, we can view the encryption to the DNF policy $\varphi$ as a distributed broadcast encryption ciphertext where the public keys are $\tilde{\mathbf{d}}_1, \ldots, \tilde{\mathbf{d}}_L$. To prove security (and support the setting where a variable might appear multiple times in the policy $\varphi$), we will use our deferred public-key generation technique described above. This yields the following construction:

- **Public parameters:** The public parameters $\mathsf{pp} = (\mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ are exactly the same as in the distributed monotone-policy encryption scheme sketched above (where the re-randomizing matrix $\mathbf{A}$ is sampled at encryption time rather than setup time). As usual, let $\mathbf{B} \in \mathbb{Z}_q^{n \times 4m^5}$ be the matrix defined by $\mathsf{pp}$.

- **Key generation:** To generate a key, user $i$ samples $\mathbf{Y}_i \xleftarrow{\text{R}} \{0,1\}^{4m^5 \times m}$ and defines their public key to be $\mathsf{pk}_i := \mathbf{T}_i = \mathbf{B}\mathbf{Y}_i$. The user's secret key is $\mathbf{Y}_i$.

- **Encryption:** To encrypt a bit $\mu \in \{0,1\}$ to a set of public keys $\{\mathsf{pk}_i\}_{i \in [N]}$ where $\mathsf{pk}_i = \mathbf{T}_i$ and a DNF policy $(S_1, \ldots, S_L)$, the encryption algorithm first samples a random translation matrix $\mathbf{A} \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$. In addition, it also samples low-norm vectors $\mathbf{u}_{i,j} \in \mathbb{Z}_q^m$ for all $i \in [N]$ and $j \in [L]$. Finally, it defines the *dense* matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$ where the $j^{\text{th}}$ column $\mathbf{m}_j \in \mathbb{Z}_q^n$ of $\mathbf{M}$ is

$$\mathbf{m}_j = \sum_{i \in S_j} \mathbf{T}_i \mathbf{u}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_{L,j},$$

  where $\mathbf{v}_{L,j}$ is the $j^{\text{th}}$ column of the matrix $\mathbf{V}_L$ from Eq. (1.1). Let $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ be a commitment to $\mathbf{M}$. Finally, the encrypter samples encryption randomness $\mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_q^n$ and outputs the ciphertext

$$\mathsf{ct} = \left(\mathbf{A}\,,\ \{\mathbf{u}_{i,j}\}_{j \in [L], i \in S_j}\,,\ \mathbf{s}^\intercal \mathbf{B}\,,\ \mathbf{s}^\intercal(\mathbf{A} + \mathbf{C})\,,\ \mathbf{s}^\intercal \mathbf{p} + \mu \cdot \lfloor q/2 \rfloor\right). \tag{1.2}$$

As written, the size of the ciphertext scales linearly with the size of the policy (due to the $\mathbf{u}_{i,j}$ vectors). However, these can be sampled using a random oracle, which yields a scheme with succinct ciphertexts.

Similar to [CW26, AGY26], we can also preprocess the set of public keys together with the access policy to derive a succinct encryption key. Given the succinct encryption key, computing the ciphertext then requires time that is independent of policy size. We provide more details in Remark 3.20.

- **Decryption:** We define the decryption hint $\mathbf{h}_i$ for a user $i \in [N]$ to be

$$\mathbf{h}_i^\intercal = \underbrace{\mathbf{s}^\intercal \mathbf{B}} \cdot \mathbf{Y}_i \approx \mathbf{s}^\intercal \mathbf{T}_i \in \mathbb{Z}_q^m,$$

where $\mathbf{Y}_i \in \{0,1\}^{4m^5 \times m}$ is the user's secret key and $\mathbf{T}_i = \mathbf{B}\mathbf{Y}_i$ is the user's public key. Take any group of users $S_j \subseteq [N]$ associated with a min-term in the policy. By construction of the commitment $\mathbf{C}$, we have from Eq. (1.1) that

$$\mathbf{C}\mathbf{v}_{L,j} = \mathbf{m}_j - \mathbf{B}\mathbf{z}_j = \sum_{i \in S_j} \mathbf{T}_i \mathbf{u}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_{L,j} - \mathbf{B}\mathbf{z}_j.$$

Now, given the decryption hints $\mathbf{h}_i \approx \mathbf{s}^\intercal \mathbf{T}_i$ for all $i \in S_j$, the decrypter can compute

$$\begin{aligned} &\underbrace{\mathbf{s}^\intercal (\mathbf{A} + \mathbf{C})} \cdot \mathbf{v}_{L,j} + \underbrace{\mathbf{s}^\intercal \mathbf{B}} \mathbf{z}_j - \sum_{i \in S_j} \mathbf{h}_i^\intercal \mathbf{u}_{i,j} \\ &\quad \approx \mathbf{s}^\intercal \mathbf{A}\mathbf{v}_{L,j} + \mathbf{s}^\intercal \sum_{i \in S_j} \mathbf{T}_i \mathbf{u}_{i,j} + \mathbf{s}^\intercal \mathbf{p} - \mathbf{s}^\intercal \mathbf{A}\mathbf{v}_{L,j} - \mathbf{s}^\intercal \mathbf{B}\mathbf{z}_j + \mathbf{s}^\intercal \mathbf{B}\mathbf{z}_j - \mathbf{s}^\intercal \sum_{i \in S_j} \mathbf{T}_i \mathbf{u}_{i,j} = \mathbf{s}^\intercal \mathbf{p}. \end{aligned}$$

As before, this is sufficient to recover the message.

**Security analysis via public-key equivocation.** Next, we give a high-level sketch of our security analysis. We work in the *static* security model where we do not allow the adversary to make corruption queries on honest users. However, we do allow the adversary the ability to *adaptively* choose the public keys for corrupted users as well as the challenge policy. The key technical challenge in proving security is simulating the challenge ciphertext component $\underbrace{\mathbf{s}^\intercal (\mathbf{A} + \mathbf{C})}$. Here, $\mathbf{C}$ is a commitment to a matrix that depends on the adversary's public keys and the reduction has no control over the public keys chosen by the adversary. Thus, the reduction algorithm will need to pick $\mathbf{A}$ strategically to properly simulate this quantity.

In the reduction to decomposed LWE, the reduction algorithm obtains commitment parameters $\mathsf{pp}_{\mathsf{com}}$, the matrix $\mathbf{B}$, and the challenge $\mathbf{c}^\intercal$, which is either an LWE sample $\underbrace{\mathbf{s}^\intercal \mathbf{B}}$ or a random vector. When simulating the challenge ciphertext, the reduction aims to set things up so that $\mathbf{A} + \mathbf{C} = \mathbf{B}\mathbf{R}$ for some low-norm matrix $\mathbf{R}$ that it knows. In this case, the reduction can simulate $\mathbf{s}^\intercal(\mathbf{A} + \mathbf{C})$ by computing $\underbrace{\mathbf{s}^\intercal \mathbf{B}} \cdot \mathbf{R}$.

The natural approach to achieve this would be to have the reduction sample $\mathbf{R} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$ and set $\mathbf{A} = \mathbf{B}\mathbf{R} - \mathbf{C}$ in the challenge ciphertext. This is the standard programming strategy used to prove selective security of related primitives (e.g., see [CW24, WW25]). Unfortunately, this technique does not apply to our construction above. This is because the matrix $\mathbf{C}$ depends on the public keys chosen by the adversary *and* the matrix $\mathbf{A}$. To overcome this, we instead follow the template proposed in the recent work on equivocal broadcast encryption [GY26b]. We describe their template and our implementation strategy below:

- **The [GY26b] equivocation strategy.** The idea in [GY26b] is to have the reduction first sample a random matrix $\mathbf{M}^* \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times L}$ and then take $\mathbf{C}^*$ to be a commitment to $\mathbf{M}^*$. Next, it programs

$\mathbf{A} = \mathbf{BR} - \mathbf{C}^*$. The idea now is to engineer the ciphertext in such a way that the encryption algorithm would naturally commit to the matrix $\mathbf{M}^*$.[6] This is the crux of the [GY26b] *equivocation* strategy. The works of [GY26b, GY26a] show how to apply this proof strategy to distributed broadcast encryption.

- **Our equivocation strategy.** In this work, we show how to implement this equivocation proof strategy directly using our deferred public-key generation technique. Specifically, if $(S_1, \ldots, S_L)$ is an admissible policy, then each set $S_j$ must contain the public key for an honest user. Now, by design, the encryption algorithm commits to the matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$ whose $j^{\text{th}}$ column is

$$\mathbf{m}_j = \sum_{i \in S_j} \mathbf{T}_i \mathbf{u}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_{L,j}. \tag{1.3}$$

  Suppose user $i^* \in S_j$ is *honest*. Then the reduction algorithm would choose $\mathbf{u}_{i^*,j}$ such that

$$\mathbf{T}_{i^*} \mathbf{u}_{i^*,j} = \mathbf{m}_j^* - \left( \sum_{i \in S_j \setminus \{i^*\}} \mathbf{T}_i \mathbf{u}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_{L,j} \right).$$

  For this choice of $\mathbf{u}_{i^*,j}$, the vector $\mathbf{m}_j$ computed via Eq. (1.3) is precisely the target vector $\mathbf{m}_j^*$. As mentioned previously, in the security analysis, we first switch to a hybrid where the reduction algorithm replaces the honest users' public keys $\mathbf{T}_{i^*}$ with matrices for which it knows the corresponding trapdoors. This will allow it to efficiently sample $\mathbf{u}_{i^*,j}$ to implement the strategy above. Finally, setting $\mathbf{p} = \mathbf{Br}$ for a short vector $\mathbf{r} \in \{0,1\}^{4m^5}$ is sufficient to simulate the remaining components of the challenge ciphertext. We give the formal construction and proof in Section 3. Note that in the full scheme, we also need to handle hint-generation queries, but these can be handled using the same mechanism as in [CW26] (i.e., include a NIZK proof of well-formedness on the ciphertext and use noise smudging to prevent the decryption hint from leaking information about the secret key).

A crucial difference between our equivocation mechanism and that from [GY26b] is our ability to program the honest user's public key to target *any* vector of our choosing. This is critical because the reduction programs the honest users' public keys to *cancel out* the adversary's public keys. In contrast, the equivocation strategy of [GY26b] is designed for the simpler case of (semi-static) distributed broadcast encryption where there are no adversarial public keys; instead, the equivocation strategy just needs to "select" between one of two possible reduction-generated keys (depending on whether the adversary chooses to include or exclude a user).

We can also compare our proof strategy with that of [CW26, AGY26] which used the ciphertext-rerandomization technique developed in [CHW25]. Essentially, we can think of [CHW25] as randomizing the commitment (i.e., take $\mathbf{A}$ to be a random commitment to the all-zeroes matrix and publish pre-images $\mathbf{B}^{-1}(\mathbf{A}\mathbf{v}_{L,i})$ in the ciphertext). In contrast, our technique instead randomizes the honest users' public keys at encryption time. The advantage of randomizing the honest users' public keys as opposed to the commitment is that we are able to prove security even if the adversary can *adaptively* choose the policy. The previous works required the adversary to commit to the policy before honest public keys are generated in order to program the honest keys to carry out the security proof (specifically, to program in a suitable smudging factor). By deferring the public-key generation to *encryption* time (where the policy is known), we are able to avoid noise smudging and prove security even when the policy is chosen adaptively.

[6] Specifically, in the work of [GY26b], each column of the matrix $\mathbf{M}$ that is committed to by the encryption algorithm is either a translated user public key or a uniform random vector.

**Re-balancing in the plain model.** Our construction above relies on the random oracle to compress the public-key derivation vectors $\mathbf{u}_{i,j}$. If we simply included $\mathbf{u}_{i,j}$ for all $j \in [L]$ and $i \in S_j$ in the ciphertext, then the size of the ciphertext scales with $kL$, where $k$ is the maximum size of a min-term and $L$ is the total number of min-terms. While this is uninteresting from a succinctness perspective, security no longer relies on the random oracle model. To get a scheme with non-trivial succinctness in the plain model, we simply "re-balance" this construction. Specifically, take any policy $(S_1, \ldots, S_L)$, where $|S_j| \leq k$ for all $j \in [L]$. We partition $(S_1, \ldots, S_L)$ into $L^{1/2}$ disjoint collections, each containing $L^{1/2}$ min-terms. The key observation now is that we can *reuse* the public-key derivation vectors $\mathbf{u}_{i,j}$ across all $L^{1/2}$ collections. The security analysis follows via a simple hybrid argument, and specifically, we leverage the fact that the encryption algorithm is a *public* function of the user's public keys, the policy, and the public-key derivation vectors $\mathbf{u}_{i,j}$. Thus, with re-balancing, we share a single translation matrix $\mathbf{A}$ together with one set of $\mathbf{u}_{i,j}$ across all $L^{1/2}$ collections. For each of the $L^{1/2}$ collections, we sample a fresh encryption secret $\mathbf{s}_w$ and include the associated components

$$(\mathbf{s}_w^\top \mathbf{B}, \mathbf{s}_w^\top (\mathbf{A} + \mathbf{C}_w), \mathbf{s}_w^\top \mathbf{p} + \mu \cdot \lfloor q/2 \rfloor)$$

specifically for that collection, where $\mathbf{C}_w$ is the commitment for that collection. This yields a scheme with ciphertexts of size $k \cdot L^{1/2}$ in the plain model. We give the construction and details in Section 4. Note that even though the size of the ciphertext depends on the size of the policy, this is determined at encryption time and not setup time; the scheme itself does not impose any a priori bounds on the policy size (or the number of users).

**Application to adaptively-secure distributed broadcast encryption.** Since our distributed monotone-policy encryption scheme for DNFs supports an adaptive choice of policy (but does not allow adaptive corruptions of honest users), this immediately implies a distributed broadcast encryption scheme with semi-static security [GW09, KMW23] (i.e., distributed broadcast encryption where the adversary can adaptively choose the broadcast set but does not get to corrupt any honest users). Next, we observe that we can apply a similar type of re-balancing to the classic Gentry-Waters compiler (which compiles a semi-statically-secure broadcast encryption scheme to an adaptively-secure one) [GW09] to obtain a scheme with security in the *plain model* (but without optimal ciphertext size). We describe this variant of the Gentry-Waters compiler in Section 5. Applied to our semi-statically-secure distributed broadcast encryption scheme, we obtain the first adaptively-secure distributed broadcast encryption scheme in the plain model from decomposed LWE that supports an unbounded number of users and has ciphertext size $|S|^{2/3}$, where $|S|$ is the size of the broadcast set. As we note in Remark 5.12, a non-black-box composition of the two re-balancing steps plausibly gives a construction with ciphertext size $|S|^{1/2}$.

**Distributed monotone-policy encryption with short ciphertexts in the plain model.** Finally, we note that if we settle for a much weaker notion of security for distributed monotone-policy encryption where the adversary has to declare the keys for corrupted users *before* seeing the public parameters, then it is straightforward to obtain a scheme with short ciphertexts in the plain model. This is the notion of selective security considered in [AGY26]. Our observation is that if we allow the reduction to choose the public keys for the honest parties *after* it sees the public keys for the adversary, then there is no need for deferred public-key sampling anymore. The reduction can simply choose the public keys for the honest parties to induce the desired cancellation in the security analysis. In combination with our basic template above, this yields a scheme with short ciphertexts (independent of the policy size) in the *plain* model. The

previous scheme of [AGY26] relied on the random oracle.[7] We describe the details in Appendix B.

We remark here that the security notion from [AGY26] of requiring the adversary to choose the public keys for malicious users before seeing those of the honest users is a very weak security notion which fails to capture security against "rogue-key attacks" [RY07, BDN18]. The very same strategy used to prove security under this definition can also be used to implement a rogue-key attack on the scheme. While we can heuristically prevent such an attack by requiring the adversary to provide a NIZK proof of knowledge of the decryption key associated with their public key, it is unclear how to leverage this to give a security reduction.

In our view, the right security definition for distributed cryptographic primitives should capture rogue-key attacks. We hope that our work here provides an example of a simple and natural construction of a distributed cryptographic scheme that is secure under definitions that do not allow rogue-key attacks, and yet, becomes trivially insecure under a rogue-key attack.

## 2 Preliminaries

We begin with some basic notation and definitions. Our presentation from here until Section 2.1 is taken largely verbatim from [CHW25, §3]. We write $\lambda$ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n] \coloneqq \{1, \ldots, n\}$. For a set $S$, we write $2^S$ to denote the power set of $S$ (i.e., the set of all subsets of $S$). We write $\{x_i\}_{i \in S}$ to denote the set of *ordered pairs* $(i, x_i)$ for all $i \in S$; namely, we associate each element $x_i$ with its unique index $i \in S$. For a finite set $S$, we write $x \xleftarrow{R} S$ to denote that $x$ is sampled uniformly from $S$, and we write $x \leftarrow \mathcal{D}$ to denote that $x$ is sampled from the distribution $\mathcal{D}$. We write $\mathsf{poly}(\lambda)$ to denote a fixed polynomial in $\lambda$ and $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We say that two distribution ensembles $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{0,\lambda}] - \Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{1,\lambda}]| = \mathsf{negl}(\lambda).$$

We say that they are statistically indistinguishable if their statistical distance $\Delta(\mathcal{D}_0, \mathcal{D}_1)$ is bounded by $\mathsf{negl}(\lambda)$. We say an event occurs with overwhelming probability if the probability of its complement occurring is negligible.

**Simulation-sound extractable NIZKs.** Next, we recall the notion of a simulation-sound extractable non-interactive zero-knowledge (NIZK) argument for NP [BFM88, FLS90, Sah99, DDO+01]. We give the definition below.

**Definition 2.1** (Simulation-Sound Extractable NIZK)**.** Let $C = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ where $C_\lambda$ is a set of Boolean circuits. A simulation-sound extractable NIZK argument $\Pi_{\mathsf{NIZK}}$ for $C$ is a tuple of efficient algorithms $\Pi_{\mathsf{NIZK}} = (\mathsf{Setup}, \mathsf{TrapSetup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Extract})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$: On input the security parameter $\lambda$, the setup algorithm outputs a common reference string crs.
- $\mathsf{TrapSetup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter $\lambda$, the trapdoor setup algorithm outputs a common reference string crs and a trapdoor td.

[7] As we discuss in Appendix B, we actually achieve a strictly stronger security definition than that considered in [AGY26] while giving a scheme in the plain model. Namely, we allow the adversary to choose the public keys after seeing the scheme parameters as well as make hint-generation queries in the security game.

- $\mathsf{Prove}(\mathsf{crs}, C, x, w) \to \pi$: On input the common reference string crs, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, the prove algorithm outputs a proof $\pi$.
- $\mathsf{Verify}(\mathsf{crs}, C, x, \pi) \to b$: On input the common reference string crs, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a statement $x \in \{0,1\}^n$, and a proof $\pi$, the verification algorithm outputs a bit $b \in \{0,1\}$.
- $\mathsf{Sim}(\mathsf{td}, C, x) \to \pi$: On input the trapdoor td, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, and a statement $x \in \{0,1\}^n$, the simulation algorithm outputs a proof $\pi$.
- $\mathsf{Extract}(\mathsf{td}, C, x, \pi) \to w$: On input the trapdoor td, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, a statement $x \in \{0,1\}^n$, and a proof $\pi$, the extraction algorithm outputs a witness $w \in \{0,1\}^h$ (or a special symbol $\perp$).

We require that $\Pi_{\mathsf{NIZK}}$ satisfy the following properties:

- **Completeness:** For all $\lambda \in \mathbb{N}$, all Boolean circuits $C \in \mathcal{C}_\lambda$ with type $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, all statements $x \in \{0,1\}^n$ and witnesses $w \in \{0,1\}^h$ where $C(x,w) = 1$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, x, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, x, w) \end{array}\right] = 1.$$

- **Zero knowledge:** For a security parameter $\lambda$, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the zero-knowledge security game as follows:
  - If $b = 0$, the challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$. If $b = 1$, the challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda)$. The challenger gives crs to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ can now make adaptive queries of the form $(C, x, w)$, where $C \in \mathcal{C}_\lambda$ is a Boolean circuit, $x \in \{0,1\}^n$ is a statement, and $w \in \{0,1\}^h$ is a witness. On each query, the challenger proceeds as follows:
    - The challenger first checks if $C(x,w) = 1$. If not, the challenger responds with $\perp$.
    - If $b = 0$, the challenger replies with $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, x, w)$. If $b = 1$, the challenger replies with $\pi \leftarrow \mathsf{Sim}(\mathsf{td}, C, x)$.
  - After $\mathcal{A}$ is finished making queries, it outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say that $\Pi_{\mathsf{NIZK}}$ satisfies computational zero knowledge if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$ in the zero-knowledge security game.
- **Simulation extractability:** For a security parameter $\lambda$ and an adversary $\mathcal{A}$, we define the simulation-extractability game as follows:
  - The challenger samples $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda)$ and gives crs to $\mathcal{A}$. The challenger also initializes an empty list $Q$.
  - Algorithm $\mathcal{A}$ can now make adaptive queries $(C, x)$ where $C \in \mathcal{C}_\lambda$ is a Boolean circuit and $x \in \{0,1\}^n$ is a statement. The challenger replies with $\pi \leftarrow \mathsf{Sim}(\mathsf{td}, C, x)$ and adds $(C, x, \pi)$ to $Q$.

– After $\mathcal{A}$ is finished making queries, it outputs a Boolean circuit $C \in \mathcal{C}_\lambda$, a statement $x \in \{0,1\}^n$, and a proof $\pi$.

– The challenger computes $w = \mathsf{Extract}(\mathsf{td}, C, x, \pi)$ and outputs $b' = 1$ if $\mathsf{Verify}(\mathsf{crs}, C, x, \pi) = 1$, $(C, x, \pi) \notin Q$ and $C(x, w) = 0$. Otherwise, the challenger outputs $b' = 0$.

We say that $\Pi_{\mathsf{NIZK}}$ is simulation extractable if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b' = 1] = \mathsf{negl}(\lambda)$ in the simulation-extractability game.

When the common reference string output by $\mathsf{Setup}(1^\lambda)$ is a uniform random string, we say that the NIZK has a *transparent* setup. The works of [PS19, WWW25, BCD+25] show how to construct a NIZK with a transparent setup from the plain LWE assumption. The work of [DDO+01] shows how to construct a simulation-sound extractable NIZK for NP from any NIZK for NP together with a public-key encryption scheme. Finally, the work of [GGI+15] shows how to combine a NIZK for NP with a (leveled) homomorphic encryption scheme to obtain a NIZK argument where the size of the proof scales with the size of the witness (and the depth of the circuit), but not the statement size. We summarize the instantiation we use here.

**Fact 2.2** (Simulation-Sound Extractable NIZK from LWE)**.** Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ where $\mathcal{C}_\lambda$ is the set of all Boolean circuits with depth at most $d = d(\lambda)$ for some fixed polynomial $d(\lambda)$. Then, under the plain LWE assumption with a sub-exponential modulus-to-noise ratio, there exists a simulation-sound extractable NIZK for $\mathcal{C}$ with the following properties:

- **Transparent setup:** The CRS is a *uniform* random string of size $\mathsf{poly}(\lambda, d)$.
- **Proof size:** For all security parameters $\lambda$, all crs in the support of $\mathsf{Setup}(1^\lambda)$ and all $C \in \mathcal{C}_\lambda$, the size of a proof output by $\mathsf{Prove}(\mathsf{crs}, C, x, w)$ is at most $|w| + \mathsf{poly}(\lambda, d)$.

## 2.1 Lattice Preliminaries

We now recall some basic facts about lattices. Parts of this section are taken verbatim from [CW26, §2.1] and [CHW25, §3.1]. Throughout this work, we use bold uppercase letters (e.g., $\mathbf{A}, \mathbf{B}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{u}, \mathbf{v}$) to denote vectors. We use non-boldface letters to denote their components (e.g., $\mathbf{v} = [v_1, \ldots, v_n]$). For a vector $\mathbf{v} \in \mathbb{R}^n$, we write $\|\mathbf{v}\| = \max_i |v_i|$ to denote the $\ell_\infty$-norm of $\mathbf{v}$, and for a matrix $\mathbf{V}$ we write $\|\mathbf{V}\| = \max_{i,j} |V_{i,j}|$. We write $\|\mathbf{v}\|_2$ to denote the $\ell_2$-norm of $\mathbf{v}$ (i.e., $\|\mathbf{v}\|_2^2 := \sum_{i \in [n]} v_i^2$). When $\mathbf{v} \in \mathbb{Z}_q^n$, we write $\|\mathbf{v}\|$ (resp., $\|\mathbf{v}\|_2$) to denote the $\ell_\infty$-norm (resp., $\ell_2$-norm) of the vector obtained by associating each component $v_i$ with its unique representative in the interval $(-q/2, q/2]$.

**Kronecker products and vectorization.** For matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{B} \in \mathbb{Z}_q^{k \times \ell}$, we write $\mathbf{A} \otimes \mathbf{B} \in \mathbb{Z}_q^{nk \times m\ell}$ to denote their Kronecker product. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we write $\mathsf{vec}(\mathbf{A})$ to denote the vectorization of $\mathbf{A}$ (i.e., the vector $\mathbf{a} \in \mathbb{Z}_q^{nm}$ obtained by concatenating together the columns of $\mathbf{A}$ in left-to-right order).

**Discrete Gaussians and gadget matrices.** We write $D_{\mathbb{Z},\sigma}$ to denote the discrete Gaussian distribution over $\mathbb{Z}$ with width parameter $\sigma > 0$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a target vector $\mathbf{y} \in \mathbb{Z}_q^n$ in the column-space of $\mathbf{A}$, we write $\mathbf{A}_\sigma^{-1}(\mathbf{y})$ to denote a random variable $\mathbf{x} \leftarrow D_{\mathbb{Z},\sigma}^m$ conditioned on $\mathbf{A}\mathbf{x} = \mathbf{y} \bmod q$. We extend $\mathbf{A}_\sigma^{-1}(\cdot)$ to matrices by applying $\mathbf{A}_\sigma^{-1}(\cdot)$ to each column of the input. For positive integers $n, q \in \mathbb{N}$, let $\mathbf{G}_n = \mathbf{I}_n \otimes \mathbf{g}^\intercal \in \mathbb{Z}_q^{n \times m'}$ be the gadget matrix [MP12] where $\mathbf{I}_n$ is the identity matrix of dimension $n$, $\mathbf{g}^\intercal = [1, 2, \ldots, 2^{\lceil \log q \rceil - 1}]$, and $m' = n \lceil \log q \rceil$. For $m > m'$, we overload $\mathbf{G}_n$ to denote the padded gadget matrix $\mathbf{G}_n = [\mathbf{I}_n \otimes \mathbf{g}^\intercal \mid \mathbf{0}^{n \times (m - m')}]$. We now recall some basic properties of the discrete Gaussian distribution.

**Lemma 2.3** (Gaussian Tail Bound [MP12, Lemma 2.6, adapted]). *For all $\sigma > 0$ and any $\lambda \in \mathbb{N}$,*

$$\Pr[|x| > \sqrt{\lambda}\sigma : x \leftarrow D_{\mathbb{Z},\sigma}] \leq 2^{-\lambda}.$$

**Lemma 2.4** (Gaussian Samples [GPV08, adapted]). *Let $n, m, q, \sigma$ be lattice parameters such that $\sigma \geq \log m$, $m \geq 2n \log q$, and $q$ is prime. There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all but a $q^{-n}$-fraction of matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, the statistical distance between the following distributions is $\mathsf{negl}(n)$:*

$$\left\{(\mathbf{x}, \mathbf{A}\mathbf{x}) : \mathbf{x} \leftarrow D_{\mathbb{Z},\sigma}^m\right\} \quad \text{and} \quad \left\{(\mathbf{x}, \mathbf{y}) : \mathbf{y} \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{x} \leftarrow \mathbf{A}_\sigma^{-1}(\mathbf{y})\right\}.$$

**Lemma 2.5** (Noise Smudging [BDE+18]). *Let $\lambda$ be a security parameter, $a \in \mathbb{Z}$ be an integer such that $|a| \leq B$, and $\sigma \geq B \cdot \lambda^{\omega(1)}$. Then, the statistical distance between the following distributions is $\mathsf{negl}(\lambda)$:*

$$\{e : e \leftarrow D_{\mathbb{Z},\sigma}\} \quad \text{and} \quad \{a + e : e \leftarrow D_{\mathbb{Z},\sigma}\}.$$

**Lattice trapdoors.** We recall the notion of a gadget trapdoor [MP12]:

**Lemma 2.6** (Gadget Trapdoor [Ajt96, GPV08, MP12]). *Let $n, m, q$ be lattice parameters with $m \geq 3n \log q$. There exist efficient algorithms $(\mathsf{TrapGen}, \mathsf{SamplePre})$ with the following syntax:*

- $\mathsf{TrapGen}(1^n, q, m) \to (\mathbf{A}, \mathbf{T})$*: On input the lattice dimension $n$, the modulus $q$, and the number of samples $m$, the trapdoor-generation algorithm outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $\mathbf{T} \in \mathbb{Z}_q^{m \times m'}$ where $m' = n\lceil \log q \rceil$.*
- $\mathsf{SamplePre}(\mathbf{A}, \mathbf{T}, \mathbf{y}, \sigma) \to \mathbf{x}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a trapdoor $\mathbf{T} \in \mathbb{Z}_q^{m \times m'}$, a target vector $\mathbf{y} \in \mathbb{Z}_q^n$, and a Gaussian width parameter $\sigma$, the preimage-sampling algorithm outputs a vector $\mathbf{x} \in \mathbb{Z}_q^m$.*

*Moreover, the above algorithms satisfy the following properties:*

- **Trapdoor distribution:** *If $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$ and $\mathbf{A}' \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, then $\Delta(\mathbf{A}, \mathbf{A}') = \mathsf{negl}(n)$. Moreover, $\mathbf{A}\mathbf{T} = \mathbf{G}_n \in \mathbb{Z}_q^{n \times m'}$ and $\|\mathbf{T}\| = 1$.*
- **Preimage sampling:** *For all matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{T} \in \mathbb{Z}_q^{m \times m'}$, width parameter $\sigma > 0$, and all target vectors $\mathbf{y} \in \mathbb{Z}_q^n$ in the column span of $\mathbf{A}$, the output $\mathbf{x} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T}, \mathbf{y}, \sigma)$ satisfies $\mathbf{A}\mathbf{x} = \mathbf{y}$.*
- **Preimage distribution:** *There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $(\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T} \in \mathbb{Z}_q^{m \times m'})$ where $\mathbf{T}$ is a gadget trapdoor for $\mathbf{A}$ (i.e., $\mathbf{A}\mathbf{T} = \mathbf{G}_n$), all $\sigma \geq m\|\mathbf{T}\| \log n$ and all target vectors $\mathbf{y} \in \mathbb{Z}_q^n$, the statistical distance between the following distributions is at most $\mathsf{negl}(n)$:*

$$\{\mathbf{x} \leftarrow \mathsf{SamplePre}(\mathbf{A}, \mathbf{T}, \mathbf{y}, \sigma)\} \quad \text{and} \quad \{\mathbf{x} \leftarrow \mathbf{A}_\sigma^{-1}(\mathbf{y})\}.$$

**Decomposed LWE.** The learning with errors (LWE) assumption [Reg05] with parameters $(n, m, q, \sigma)$ states that the following distributions are computationally indistinguishable:

$$(\mathbf{B}, \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal) \text{ and } (\mathbf{B}, \mathbf{v}^\intercal) \quad \text{where} \quad \mathbf{B} \xleftarrow{R} \mathbb{Z}_q^{n \times m},\ \mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n,\ \mathbf{e} \xleftarrow{R} D_{\mathbb{Z},\sigma}^m,\ \mathbf{v} \xleftarrow{R} \mathbb{Z}_q^m.$$

For parameters $\ell \in \mathbb{N}$ and $s > 0$, the $(\ell, s)$-decomposed LWE assumption from [AMR25] asserts that LWE is hard when the matrix $\mathbf{B}$ is sampled in a structured manner:

$$\mathbf{B} := \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{R}) + \mathrm{vec}(\mathbf{I}_\ell)^\intercal \otimes \mathbf{G} \quad \text{where} \quad \mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell m},\ \mathbf{R} \leftarrow D_{\mathbb{Z},s}^{m \times \ell m}.$$

We now give the formal statement of the assumption:

**Assumption 2.7** (Decomposed LWE [AMR25])**.** Let $\lambda$ be a security parameter and let $n = n(\lambda), m = m(\lambda), q = q(\lambda), \sigma = \sigma(\lambda)$ be lattice parameters. Let $s > 0$ be a Gaussian width parameter and $\ell \in \mathbb{N}$ be a dimension. We say that the $(\ell, s)$-decomposed LWE assumption with parameters $(n, m, q, \sigma)$ holds if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$:

$$\left|\Pr[\mathcal{A}(1^\lambda, \mathbf{W}, \mathbf{R}, \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal) = 1] - \Pr[\mathcal{A}(1^\lambda, \mathbf{W}, \mathbf{R}, \mathbf{v}^\intercal) = 1]\right| = \mathsf{negl}(\lambda),$$

where $\mathbf{B} = \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_\ell)^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times \ell^2 m}$, $\mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell m}$, $\mathbf{R} \leftarrow D_{\mathbb{Z},s}^{m \times \ell m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z},\sigma}^{\ell^2 m}$, and $\mathbf{v} \xleftarrow{R} \mathbb{Z}_q^{\ell^2 m}$. We will sometimes write "$\ell$-decomposed LWE" as shorthand to refer to an instance of the $(\ell, s)$-decomposed LWE assumption for some (implicitly-defined) width parameter $s$.

**Leftover hash lemma.** Next, we recall the (generalized) leftover hash lemma [HILL99, DORS08, ABB10] along with a version of the lemma that holds for the matrix $\mathbf{B}$ used in the decomposed LWE assumption:

**Lemma 2.8** (Generalized Leftover Hash Lemma [ABB10, Lemma 13, adapted])**.** *Let $n, m, q$ be integers such that $m \geq 2n \log q$ and $q > 2$ is prime. Then, for all fixed vectors $\mathbf{e} \in \mathbb{Z}_q^m$ and all $k = \mathsf{poly}(n)$, the statistical distance between the following distributions is $\mathsf{negl}(n)$:*

$$\left\{(\mathbf{A}, \mathbf{AK}, \mathbf{e}^\intercal\mathbf{K}) : \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}, \mathbf{K} \xleftarrow{R} \{0,1\}^{m \times k}\right\} \quad \text{and} \quad \left\{(\mathbf{A}, \mathbf{U}, \mathbf{e}^\intercal\mathbf{K}) : \begin{array}{c} \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}, \mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{n \times k} \\ \mathbf{K} \xleftarrow{R} \{0,1\}^{m \times k} \end{array}\right\}.$$

**Corollary 2.9** (Leftover Hash Lemma for Decomposed LWE Matrix [CW26])**.** *Let $n, m, q$ be integers such that $m \geq 2n \log q$ and $q$ is prime. Let $\ell \in \mathbb{N}$ and suppose $s \geq \log m$. Then, for all fixed vectors $\mathbf{e} \in \mathbb{Z}_q^{\ell^2 m}$ and all $k = \mathsf{poly}(n)$, the statistical distance between the following distributions is $\mathsf{negl}(n)$:*

- $\left\{(\mathbf{W}, \mathbf{R}, \mathbf{BK}, \mathbf{e}^\intercal\mathbf{K}) : \mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell m},\ \mathbf{R} \leftarrow D_{\mathbb{Z},s}^{m \times \ell m},\ \mathbf{K} \xleftarrow{R} \{0,1\}^{\ell^2 m \times k}\right\}$*; and*
- $\left\{(\mathbf{W}, \mathbf{R}, \mathbf{U}, \mathbf{e}^\intercal\mathbf{K}) : \mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell m},\ \mathbf{R} \leftarrow D_{\mathbb{Z},s}^{m \times \ell m},\ \mathbf{U} \xleftarrow{R} \mathbb{Z}_q^{n \times k},\ \mathbf{K} \xleftarrow{R} \{0,1\}^{\ell^2 m \times k}\right\}$,

*where $\mathbf{B} = \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_\ell)^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times \ell^2 m}$ in both distributions.*

**Matrix commitments.** A primary building block we use in this work is the matrix commitment by Wee [Wee25]. In Wee's construction, a commitment to a matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$ is a matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ such that

$$\mathbf{C} \cdot \mathbf{V}_L = \mathbf{M} - \mathbf{B} \cdot \mathbf{Z},$$

where $\mathbf{B}, \mathbf{V}_L$ are matrices defined by a set of *public* parameters pp, and $\mathbf{Z}$ is a low-norm opening. The matrix $\mathbf{V}_L$ also has low norm. We summarize the main properties from [Wee25, WW25] below. Specifically, we describe the version from [Wee25, Appendix C.2] where the public parameters pp can be described by a uniform random string. This latter property will enable constructions with a transparent setup.

**Theorem 2.10** (Sparse Matrix Commitment [Wee25, WW25, adapted])**.** *Let $n, m, q$ be lattice parameters where $m \geq 2n \log q$. There exists a triple of efficient and* deterministic *algorithms* $(\mathsf{Com}, \mathsf{Ver}, \mathsf{Open})$ *with the following syntax and properties:*

- $\mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \to \mathbf{C}$*: On input the public parameters $\mathsf{pp}_{\mathsf{com}}$ and a matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$, the* $\mathsf{Com}$ *algorithm outputs a commitment matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ in time $\mathsf{poly}(K, m, \log q, \log L)$, where $K$ is the number of non-zero columns in $\mathbf{M}$.*[8]

[8]Here, we assume that the description of $\mathbf{M}$ is provided in the form of a list of non-zero columns.

- $\mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, L, i) \to \mathbf{v}_{L,i}$: *On input the public parameters* $\mathsf{pp}_{\mathsf{com}}$*, the length parameter* $L \in \mathbb{N}$ *(in* binary*), and an index* $i \in [L]$*, the* Ver *algorithm outputs a vector* $\mathbf{v}_{L,i} \in \mathbb{Z}_q^m$ *in time* $\mathsf{poly}(m, \log q, \log L)$.
- $\mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}, i) \to \mathbf{z}_i$: *On input the public parameters* $\mathsf{pp}_{\mathsf{com}}$*, a matrix* $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$*, and an index* $i \in [L]$*, the* Open *algorithm outputs a vector* $\mathbf{z}_i \in \mathbb{Z}_q^{4m^5}$ *in time* $\mathsf{poly}(K, m, \log q, \log L)$*, where* $K$ *is the number of non-zero columns in* $\mathbf{M}$.

*For all* $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$ *where* $\mathbf{W} \in \mathbb{Z}_q^{n \times 2m^3}$, $\mathbf{R} \in \mathbb{Z}_q^{m \times 2m^3}$, *and* $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times 4m^5}$*, all parameters* $L \in \mathbb{N}$*, all matrices* $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_L] \in \mathbb{Z}_q^{n \times L}$*, all indices* $i \in [L]$*, and setting*

$$\begin{aligned} \mathbf{C} &= \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) && \in \mathbb{Z}_q^{n \times m} \\ \mathbf{v}_{L,i} &= \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, L, i) && \in \mathbb{Z}_q^{m} \\ \mathbf{z}_i &= \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}, i) && \in \mathbb{Z}_q^{4m^5}, \end{aligned}$$

*the following hold:*

$$\mathbf{C}\mathbf{v}_{L,i} = \mathbf{m}_i - \mathbf{B}\mathbf{z}_i \quad \textit{and} \quad \|\mathbf{v}_{L,i}\| \le O(\|\mathbf{R}\| \cdot m^4 \log q) \quad \textit{and} \quad \|\mathbf{z}_i\| \le O(\|\mathbf{R}\| \cdot m^7 \log q \log L).$$

**Explainable discrete Gaussian samplers.** We recall the notion of an explainable discrete Gaussian sampler from [LW22].

**Definition 2.11** (Explainable Discrete Gaussian Sampler [LW22, adapted])**.** Let $\lambda$ be a security parameter and $m$ be a dimension. A $\rho$-explainable discrete Gaussian sampler $\Pi_{\mathsf{DGS}}$ with randomness length $\rho(\lambda, m)$ is a pair of efficient algorithms $\Pi_{\mathsf{DGS}} = (\mathsf{Sample}, \mathsf{Explain})$ with the following syntax:

- $\mathsf{Sample}(1^\lambda, 1^m, \sigma; r) \to \mathbf{x}$: On input a security parameter $\lambda$, a dimension $m$, a width parameter $\sigma$, and randomness $r \in \{0,1\}^{\rho(\lambda,m)}$, the sampling algorithm outputs a vector $\mathbf{x} \in \mathbb{Z}^m$.
- $\mathsf{Explain}(1^\lambda, 1^\kappa, \mathbf{x}, \sigma) \to r$: On input a security parameter $\lambda$, a precision parameter $\kappa$, a sample $\mathbf{x} \in \mathbb{Z}^m$, and a width parameter $\sigma$, the explain algorithm outputs a string $r \in \{0,1\}^{\rho(\lambda,m)}$.

Moreover, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all width parameters $\sigma \le 2^\lambda$, the statistical distance between the following distributions is bounded by $1/\kappa + \mathsf{negl}(\lambda)$:

$$\left\{ (\mathbf{x}, r) : \begin{array}{c} r \xleftarrow{\mathrm{R}} \{0,1\}^{\rho(\lambda,m)} \\ \mathbf{x} = \mathsf{Sample}(1^\lambda, 1^m, \sigma; r) \end{array} \right\} \quad \text{and} \quad \left\{ (\mathbf{x}, r) : \begin{array}{c} \mathbf{x} \leftarrow D_{\mathbb{Z},\sigma}^m \\ r \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, \mathbf{x}, \sigma) \end{array} \right\}.$$

**Theorem 2.12** (Explainable Discrete Gaussian Sampler [LW22])**.** *There exists a* $\rho$*-explainable discrete Gaussian sampler* $\Pi_{\mathsf{DGS}}$ *for* $\rho \in \mathsf{poly}(\lambda, m)$.

## 2.2 Distributed Monotone-Policy Encryption

In this section, we recall the notion of distributed monotone-policy encryption with one-round distributed decryption from [DJWW25, Appendix B]:

**Definition 2.13** (Distributed Monotone-Policy Encryption [DJWW25, adapted])**.** Let $\lambda$ be a security parameter and $\mathcal{P}$ be a family of monotone access policies. We model each policy $P \in \mathcal{P}$ as a monotone Boolean function. When $P\colon \{0,1\}^n \to \{0,1\}$ is a function on $n$-bit inputs and $T \subset [n]$, we will sometimes overload notation and define $P(T)$ to be $P(\beta_1, \ldots, \beta_n)$ where $\beta_i = 1$ if $i \in T$ and $\beta_i = 0$ otherwise. A distributed monotone-policy encryption scheme with policy space $\mathcal{P}$ is a tuple of efficient algorithms $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the setup algorithm outputs a set of public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the public parameters pp, the key-generation algorithm outputs a public key $\mathsf{pk}_i$ and a secret key $\mathsf{sk}_i$.
- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu) \to \mathsf{ct}$: On input the public parameters pp, a policy $P \in \mathcal{P}$ (on $n$-bit inputs), a tuple of $n$ public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and a message $\mu \in \{0,1\}$, the encryption algorithm outputs a ciphertext ct. Note that the input length $n$ is determined by $P$ and is *not* fixed by the scheme.
- $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct}) \to \mathsf{ht}_i$: On input the public parameters pp, a secret key $\mathsf{sk}_i$, and a ciphertext ct, the hint-generation algorithm outputs a decryption hint $\mathsf{ht}_i$.
- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}) \to \mu$: On input the public parameters pp, a policy $P \in \mathcal{P}$, a tuple of $n$ public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, a set $T \subseteq [n]$, decryption hints $\mathsf{ht}_i$ for $i \in T$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \{0,1\}$.

Moreover, $\Pi_{\mathsf{DMPE}}$ should satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all policies $P \in \mathcal{P}$ (on $n$-bit inputs), all sets $T \subseteq [n]$ where $P(T) = 1$, all messages $\mu \in \{0,1\}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, and all public keys $\mathsf{pk}_i$ for $i \notin T$,

$$\Pr\left[\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}) = \mu\right] = 1,$$

  where $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for every $i \in T$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$, and $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$ for all $i \in T$.

- **Security:** For a security parameter $\lambda$, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the security game as follows:
  - **Setup phase:** At the beginning of the game, the challenger samples the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and initializes a counter $\mathsf{ctr} = 0$ and an (empty) list $C = \emptyset$. The list $C$ is used to keep track of corrupted keys. The challenger gives pp to $\mathcal{A}$.
  - **Pre-challenge query phase:** The adversary can now make the following queries:
    - **Key-generation query:** In a key-generation query, the challenger increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, and responds with $\mathsf{pk}_{\mathsf{ctr}}$.
    - **Corruption query:** In a corruption query, the adversary specifies a counter value $\mathsf{ctr}' \le \mathsf{ctr}$, and the challenger replies with $\mathsf{sk}_{\mathsf{ctr}'}$. The adversary adds $\mathsf{ctr}'$ to $C$.
    - **Hint query:** In a hint query, the adversary specifies a ciphertext ct and a counter value $\mathsf{ctr}' \le \mathsf{ctr}$. The challenger replies with $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_{\mathsf{ctr}'}, \mathsf{ct})$.
  - **Challenge phase:** In the challenge phase, the adversary specifies a challenge policy $P \in \mathcal{P}$ on $n$ input bits. In addition, for each $i \in [n]$, algorithm $\mathcal{A}$ either specifies a public key $\mathsf{pk}_i$ or a counter value $\mathsf{ctr}_i$. For each $i \in [n]$ where algorithm $\mathcal{A}$ specifies a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$, the challenger sets $\mathsf{pk}_i = \mathsf{pk}_{\mathsf{ctr}_i}$. The challenger replies to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), b)$.
  - **Post-challenge query phase:** The adversary can continue to make corruption and hint queries in this phase. If the adversary asks for a hint on the challenge ciphertext $\mathsf{ct}^*$, the counter value $\mathsf{ctr}'$ is added to $C$. (Note that post-challenge key-generation queries are not useful).

– **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

Let $\beta_i = 1$ if the adversary specified a public key $\mathsf{pk}_i$ or a corrupted counter value $\mathsf{ctr}_i$ where $\mathsf{ctr}_i \in C$ during the challenge phase. For indices $i \in [n]$ where $\mathcal{A}$ specified an uncorrupted counter value $\mathsf{ctr}_i \notin C$, let $\beta_i = 0$. We say that $\mathcal{A}$ is admissible if $P(\beta_1, \ldots, \beta_n) = 0$. We say that $\Pi_{\mathsf{DMPE}}$ is secure if for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

in the above security game.

- **Succinctness:** There exists a polynomial $\mathsf{poly}(\cdot,\cdot)$ such that for all $\lambda \in \mathbb{N}$, public parameters pp in the support of $\mathsf{Setup}(1^\lambda)$, policies $P \in \mathcal{P}$ (on $n$-bit inputs), public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$, and messages $\mu \in \{0,1\}$, the size of the ciphertext ct output by $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mu)$ is $o(|P|) \cdot \mathsf{poly}(\lambda, \log n)$.

**Definition 2.14** (Static Security)**.** Let $\Pi_{\mathsf{DMPE}}$ be a distributed monotone-policy encryption scheme with policy space $\mathcal{P}$. We say that $\Pi_{\mathsf{DMPE}}$ satisfies static security if $\Pi_{\mathsf{DMPE}}$ is secure against adversaries that do not make any corruption queries during the query phase and moreover, the adversary cannot make any hint queries on the challenge ciphertext in the post-challenge query phase. Importantly, the adversary can still choose public keys itself (for any non-accepting subset of parties) during the challenge phase.

# 3 Optimal Distributed Monotone-Policy Encryption for DNFs

We now give our construction of a distributed monotone-policy encryption scheme for DNF policies with optimal parameters and standard static security in the random oracle model.

**Construction 3.1** (Distributed Monotone-Policy Encryption)**.** Let $\lambda$ be a security parameter. We define the following:

- Let $(n, m, q, \sigma_{\mathsf{LWE}})$ be LWE parameters (which may be functions of $\lambda$). Let $\sigma_{\mathsf{pp}}, \sigma_{\mathsf{ht}} > 0$ be Gaussian width parameters.
- Let $\mathcal{P}_{\mathsf{DNF}}$ be the set of DNF formulas on up to $2^\lambda$-bit inputs. We will represent a DNF $P \in \mathcal{P}_{\mathsf{DNF}}$ on input variables $x_1, \ldots, x_N$ as a collection of $\ell$ sets $S_1, \ldots, S_\ell \subseteq [N]$, where $S_1, \ldots, S_\ell$ represents the policy $\bigvee_{i \in [\ell]} \left(\bigwedge_{j \in S_i} x_j\right)$.
- Let $C_{\mathsf{ct}}$ be the Boolean circuit that takes a statement $(\mathbf{B}, \mathrm{ct}_{\mathsf{main}}, \beta)$, a witness $(\mathbf{s}, \mathbf{e})$, and outputs 1 if $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal$ and $\|\mathbf{e}\| \leq \beta$, where $\mathrm{ct}_{\mathsf{main}} = (\tau, \mathbf{A}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$, with $\tau \in \{0,1\}^\lambda$, $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{c}_1^\intercal \in \mathbb{Z}_q^{4m^5}$, $\mathbf{c}_2^\intercal \in \mathbb{Z}_q^m$, and $c_3 \in \mathbb{Z}_q$. Note that $C_{\mathsf{ct}}$ can be computed by a circuit of depth $\mathsf{poly}(n, m, \log q)$.
- Let $\Pi_{\mathsf{NIZK}} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.TrapSetup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify}, \mathsf{NIZK.Sim}, \mathsf{NIZK.Extract})$ be a simulation-sound extractable NIZK argument for a family $C = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ where $C_\lambda$ contains the circuits $C_{\mathsf{ct}}$.
- Let $(\mathsf{Sample}, \mathsf{Explain})$ be a $\rho'$-explainable discrete Gaussian sampler (Definition 2.11) and let $\rho = \rho(\lambda, m)$ be an upper bound on the value of $\rho'$ for the sampler instances used in the construction.
- Let $H \colon \{0,1\}^\lambda \times \mathbb{N}^2 \to \{0,1\}^\rho$ be a hash function which we model as a random oracle in the security analysis.

We construct a distributed monotone-policy encryption scheme $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ for $\mathcal{P}_{\mathsf{DNF}}$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter $\lambda$, the setup algorithm samples

$$\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda),\ \mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3}.$$

  If $\|\mathbf{R}\| > \sqrt{m}\cdot\sigma_{\mathsf{pp}}$, set $\mathbf{R} = \mathbf{0}^{m\times 2m^3}$. Let

$$\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2}\otimes\mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n\times 4m^5} \quad \text{and} \quad \mathsf{pp}_{\mathsf{com}} = (\mathbf{W},\mathbf{R},\mathbf{B}).$$

  Finally, it outputs $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$.

- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W},\mathbf{R},\mathbf{B})$, the key-generation algorithm samples $\mathbf{Y} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5\times m}$ and computes $\mathbf{T} = \mathbf{B}\mathbf{Y} \in \mathbb{Z}_q^{n\times m}$. It outputs the public key $\mathsf{pk} = \mathbf{T}$ and the secret key $\mathsf{sk} = \mathbf{Y}$.

- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1,\ldots,\mathsf{pk}_N), \mu)$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W},\mathbf{R},\mathbf{B})$, a policy $P = (S_1,\ldots,S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$, a collection of public keys $\mathsf{pk}_j = \mathbf{T}_j$ for $j \in [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm samples the following:

$$\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{e} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{LWE}}}^{4m^5},\ \mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times m},\ \mathbf{K}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5\times m},\ \mathbf{k}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}.$$

  If $\|\mathbf{e}\| > \sqrt{m}\cdot\sigma_{\mathsf{LWE}}$, it sets $\mathbf{e} = \mathbf{0}^{4m^5}$. Next, it samples $\tau \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$ and for $i \in [\ell],\ j \in S_i$ it computes

$$\mathbf{u}_{i,j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; H(\tau,i,j)).$$

  It sets $\mathbf{u}_{i,j} = \mathbf{0}^m$ if $\|\mathbf{u}_{i,j}\| > \sqrt{m}\cdot\sigma_{\mathsf{pp}}$ and computes $\mathbf{d}_{i,j} = \mathbf{T}_j\cdot\mathbf{u}_{i,j}$. For each $i \in [\ell]$, it sets $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and

$$\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell] \in \mathbb{Z}_q^{n\times\ell} \quad \text{where} \quad \forall i \in [\ell] : \mathbf{m}_i = \sum_{j\in S_i} \mathbf{d}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \in \mathbb{Z}_q^n. \tag{3.1}$$

  It computes the commitment

$$\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \in \mathbb{Z}_q^{n\times m}. \tag{3.2}$$

  Then, it sets $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$, $\mathbf{c}_2^\intercal = \mathbf{s}^\intercal(\mathbf{A}+\mathbf{C}) + \mathbf{e}^\intercal\mathbf{K}_{\mathbf{A}}$, and $c_3 = \mathbf{s}^\intercal\mathbf{p} + \mathbf{e}^\intercal\mathbf{k}_{\mathbf{p}} + \mu\cdot\lfloor q/2\rfloor$. Finally, it sets $\mathsf{ct}_{\mathsf{main}} = (\tau, \mathbf{A}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$ and computes the proof

$$\pi_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), (\mathbf{s},\mathbf{e})),$$

  and outputs $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$.

- $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W},\mathbf{R},\mathbf{B})$, a secret key $\mathsf{sk}_i = \mathbf{Y}_i$, and a ciphertext $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}} = (\tau, \mathbf{A}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$, the hint-generation algorithm checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), \pi_{\mathsf{ct}}) = 1.$$

  If the check fails, it outputs $\perp$. Otherwise, it samples $\mathbf{e} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}^m$, sets $\mathbf{e} = \mathbf{0}^m$ if $\|\mathbf{e}\| > \sqrt{m}\cdot\sigma_{\mathsf{ht}}$, and outputs $\mathbf{c}_1^\intercal\mathbf{Y}_i + \mathbf{e}^\intercal$.

- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct})$: On input the parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$, a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$, a collection of public keys $\mathsf{pk}_j = \mathbf{T}_j$ for $j \in [N]$, a list of decryption hints $\mathsf{ht}_i = \mathbf{h}_i^\intercal$ for $i \in T \subseteq [N]$, and a ciphertext $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}} = (\tau, \mathbf{A}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$, the decryption algorithm outputs $\perp$ if $P(T) = 0$. Otherwise, for each $i' \in [\ell]$ and $j \in S_{i'}$, it computes

$$\mathbf{u}_{i',j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; H(\tau, i', j)),$$

sets $\mathbf{u}_{i',j} = \mathbf{0}^m$ if $\|\mathbf{u}_{i',j}\| > \sqrt{m} \cdot \sigma_{\mathsf{pp}}$, and computes $\mathbf{d}_{i',j} = \mathbf{T}_j \cdot \mathbf{u}_{i',j}$. It then defines $\mathbf{M}$ as in Eq. (3.1) and computes

$$\mathbf{v}_{\mathsf{ind}} = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, \mathsf{ind}) \quad \text{and} \quad \mathbf{z}_{\mathsf{ind}} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}, \mathsf{ind}),$$

where $\mathsf{ind} \in [\ell]$ is the smallest index such that $S_{\mathsf{ind}} \subseteq T$. Compute

$$\mu' = c_3 - \mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{c}_2^\intercal \mathbf{v}_{\mathsf{ind}} + \sum_{i \in S_{\mathsf{ind}}} \mathbf{h}_i^\intercal \cdot \mathbf{u}_{\mathsf{ind},i}.$$

Output 0 if $-q/4 < \mu' < q/4$ and 1 otherwise.

**Theorem 3.2** (Correctness). *Suppose $q > N \cdot O(m^{13} \sigma_{\mathsf{pp}} \sigma_{\mathsf{ht}} \sigma_{\mathsf{LWE}} \log q \log \ell)$ and $\Pi_{\mathsf{NIZK}}$ is complete. Then, Construction 3.1 is correct.*

*Proof.* Take any $\lambda \in \mathbb{N}$, any $N, \ell \leq 2^\lambda$, any $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$ on $N$-bit inputs, any set $T \subseteq [N]$ where $P(T) = 1$, any message $\mu \in \{0,1\}$, any pp in the support of $\mathsf{Setup}(1^\lambda)$, any $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for $i \in T$, and any choice of $\mathsf{pk}_i$ for $i \notin T$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \mu)$ and for each $i \in T$, let $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$. Now, consider the output of

$$\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}).$$

We analyze each step individually:

- Let $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. By definition of Setup, we have $\|\mathbf{R}\| \leq \sqrt{m}\sigma_{\mathsf{pp}}$.
- For $i \in [N]$, we have $\mathsf{pk}_i = \mathbf{T}_i \in \mathbb{Z}_q^{n \times m}$. Additionally, for $i \in T$ we have that $\mathbf{T}_i = \mathbf{B}\mathbf{Y}_i$ where $\mathbf{Y}_i \in \{0,1\}^{4m^5 \times m}$.
- For each $i \in [\ell]$, let $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$.
- By definition, the encryption algorithm samples $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow D_{\mathbb{Z}, \sigma_{\mathsf{LWE}}}^{4m^5}$, $\tau \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$, $\mathbf{K}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$, and $\mathbf{k}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$, where $\|\mathbf{e}\| \leq \sqrt{m}\sigma_{\mathsf{LWE}}$ and $\|\mathbf{K}_{\mathbf{A}}\|, \|\mathbf{k}_{\mathbf{p}}\| \leq 1$. Next, for each $i \in [\ell]$ and $j \in S_i$, the encryption algorithm computes $\mathbf{u}_{i,j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; H(\tau, i, j))$. It sets $\mathbf{u}_{i,j} = \mathbf{0}^m$ if $\|\mathbf{u}_{i,j}\| > \sqrt{m}\sigma_{\mathsf{pp}}$. Then, it computes

$$\mathbf{d}_{i,j} = \mathbf{T}_j \cdot \mathbf{u}_{i,j}.$$

Finally, it defines $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell]$ as in Eq. (3.1), where for each $i \in [\ell]$

$$\mathbf{m}_i = \textstyle\sum_{j \in S_i} \mathbf{d}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \in \mathbb{Z}_q^n,$$

and computes $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \in \mathbb{Z}_q^{n \times m}$. The ciphertext has the form $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where

$$\begin{aligned} \mathsf{ct}_{\mathsf{main}} &= (\tau, \mathbf{A}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3) \\ &= \left(\tau, \mathbf{A}, \mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal, \mathbf{s}^\intercal(\mathbf{A} + \mathbf{C}) + \mathbf{e}^\intercal \mathbf{K}_{\mathbf{A}}, \mathbf{s}^\intercal \mathbf{p} + \mathbf{e}^\intercal \mathbf{k}_{\mathbf{p}} + \mu \cdot \lfloor q/2 \rfloor\right). \end{aligned}$$

By completeness of $\Pi_{\mathsf{NIZK}}$, we have

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \pi_{\mathsf{ct}}) = 1.$$

- By definition of GetHint, for each $i \in T$, the hint $\mathsf{ht}_i = \mathbf{h}_i^\intercal \in \mathbb{Z}^m$ satisfies

$$\mathbf{h}_i^\intercal = \mathbf{c}_1^\intercal \mathbf{Y}_i + \mathbf{e}_i^\intercal = \mathbf{s}^\intercal \mathbf{B}\mathbf{Y}_i + \mathbf{e}^\intercal \mathbf{Y}_i + \mathbf{e}_i^\intercal = \mathbf{s}^\intercal \mathbf{T}_i + \mathbf{e}^\intercal \mathbf{Y}_i + \mathbf{e}_i^\intercal,$$

  where $\mathbf{e}_i \in \mathbb{Z}^m$ is the per-hint smudging noise satisfying $\|\mathbf{e}_i\| \le \sqrt{m} \cdot \sigma_{\mathsf{ht}}$.

- Since $P(T) = 1$, there exists an index $\mathsf{ind} \in [\ell]$ where $S_{\mathsf{ind}} \subseteq T$. Let $\mathsf{ind}$ be the smallest such index. The Decrypt algorithm computes $\mathbf{z}_{\mathsf{ind}} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}, \mathsf{ind})$. By correctness of the matrix commitment (Theorem 2.10), we have $\mathbf{C}\mathbf{v}_{\mathsf{ind}} = \mathbf{m}_{\mathsf{ind}} - \mathbf{B}\mathbf{z}_{\mathsf{ind}}$, and therefore

$$\mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} = (\mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal)\mathbf{z}_{\mathsf{ind}} = \mathbf{s}^\intercal(\mathbf{m}_{\mathsf{ind}} - \mathbf{C}\mathbf{v}_{\mathsf{ind}}) + \mathbf{e}^\intercal \mathbf{z}_{\mathsf{ind}}.$$

  In addition, $\mathbf{m}_{\mathsf{ind}} = \sum_{i \in S_{\mathsf{ind}}} \mathbf{d}_{\mathsf{ind},i} + \mathbf{p} - \mathbf{A}\mathbf{v}_{\mathsf{ind}}$ by Eq. (3.1). Note that for each $i \in S_{\mathsf{ind}} \subseteq T$, the public key $\mathbf{T}_i = \mathbf{B}\mathbf{Y}_i$ together with the identity $\mathbf{d}_{\mathsf{ind},i} = \mathbf{T}_i \cdot \mathbf{u}_{\mathsf{ind},i}$ yields

$$\mathbf{s}^\intercal \mathbf{T}_i \cdot \mathbf{u}_{\mathsf{ind},i} = \mathbf{s}^\intercal \mathbf{d}_{\mathsf{ind},i}.$$

  Combining the above, we thus have

$$\mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} = \mathbf{s}^\intercal \Big( \textstyle\sum_{i \in S_{\mathsf{ind}}} \mathbf{d}_{\mathsf{ind},i} + \mathbf{p} - (\mathbf{A} + \mathbf{C})\mathbf{v}_{\mathsf{ind}} \Big) + \mathbf{e}^\intercal \mathbf{z}_{\mathsf{ind}}.$$

  Consider now the expression computed by the decryption algorithm. We have

$$\begin{aligned}
\mu' &= c_3 - \mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{c}_2^\intercal \mathbf{v}_{\mathsf{ind}} + \sum_{i \in S_{\mathsf{ind}}} \mathbf{h}_i^\intercal \mathbf{u}_{\mathsf{ind},i} \\
&= c_3 - \mathbf{s}^\intercal \Big( \textstyle\sum_{i \in S_{\mathsf{ind}}} \mathbf{d}_{\mathsf{ind},i} + \mathbf{p} - (\mathbf{A} + \mathbf{C})\mathbf{v}_{\mathsf{ind}} \Big) - \mathbf{e}^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{s}^\intercal(\mathbf{A} + \mathbf{C})\mathbf{v}_{\mathsf{ind}} - \mathbf{e}^\intercal \mathbf{K}_{\mathbf{A}} \mathbf{v}_{\mathsf{ind}} + \textstyle\sum_{i \in S_{\mathsf{ind}}} \mathbf{h}_i^\intercal \mathbf{u}_{\mathsf{ind},i} \\
&= c_3 - \mathbf{s}^\intercal \left( \sum_{i \in S_{\mathsf{ind}}} \mathbf{d}_{\mathsf{ind},i} + \mathbf{p} \right) - \mathbf{e}^\intercal (\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}} \mathbf{v}_{\mathsf{ind}}) + \sum_{i \in S_{\mathsf{ind}}} (\mathbf{s}^\intercal \mathbf{T}_i + \mathbf{e}^\intercal \mathbf{Y}_i + \mathbf{e}_i^\intercal) \mathbf{u}_{\mathsf{ind},i} \\
&= c_3 - \mathbf{s}^\intercal \mathbf{p} - \mathbf{e}^\intercal \left( \mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}} \mathbf{v}_{\mathsf{ind}} - \sum_{i \in S_{\mathsf{ind}}} \mathbf{Y}_i \mathbf{u}_{\mathsf{ind},i} \right) + \sum_{i \in S_{\mathsf{ind}}} \mathbf{e}_i^\intercal \mathbf{u}_{\mathsf{ind},i} \\
&= \mu \cdot \lfloor q/2 \rfloor - \mathbf{e}^\intercal \left( \mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}} \mathbf{v}_{\mathsf{ind}} - \sum_{i \in S_{\mathsf{ind}}} \mathbf{Y}_i \mathbf{u}_{\mathsf{ind},i} - \mathbf{k}_{\mathbf{p}} \right) + \sum_{i \in S_{\mathsf{ind}}} \mathbf{e}_i^\intercal \mathbf{u}_{\mathsf{ind},i}.
\end{aligned}$$

Correctness holds as long as

$$\left| \mathbf{e}^\intercal \left( \mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}} \mathbf{v}_{\mathsf{ind}} - \textstyle\sum_{i \in S_{\mathsf{ind}}} \mathbf{Y}_i \mathbf{u}_{\mathsf{ind},i} - \mathbf{k}_{\mathbf{p}} \right) + \sum_{i \in S_{\mathsf{ind}}} \mathbf{e}_i^\intercal \mathbf{u}_{\mathsf{ind},i} \right| < q/4,$$

so we bound the left side of the expression. By Theorem 2.10, $\|\mathbf{v}_{\mathsf{ind}}\| \le O(\|\mathbf{R}\| \cdot m^4 \log q) = O(\sigma_{\mathsf{pp}} \cdot m^{9/2} \log q)$ and $\|\mathbf{z}_{\mathsf{ind}}\| \le O(\|\mathbf{R}\| \cdot m^7 \log q \log \ell) = O(\sigma_{\mathsf{pp}} \cdot m^{15/2} \log q \log \ell)$. Combining with $\|\mathbf{K}_{\mathbf{A}}\|, \|\mathbf{k}_{\mathbf{p}}\| \le 1$ and $\|\mathbf{Y}_i\| \le 1$, $\left\|\mathbf{u}_{\mathsf{ind},i}\right\| \le \sqrt{m}\sigma_{\mathsf{pp}}$, we have

$$\begin{aligned}
\|\mathbf{K}_{\mathbf{A}} \mathbf{v}_{\mathsf{ind}}\| &\le m \cdot \|\mathbf{K}_{\mathbf{A}}\| \cdot \|\mathbf{v}_{\mathsf{ind}}\| = O(\sigma_{\mathsf{pp}} \cdot m^{11/2} \log q), \\
\left\|\mathbf{Y}_i \mathbf{u}_{\mathsf{ind},i}\right\| &\le m \cdot \|\mathbf{Y}_i\| \cdot \left\|\mathbf{u}_{\mathsf{ind},i}\right\| = O(m^{3/2} \sigma_{\mathsf{pp}}),
\end{aligned}$$

and hence

$$\left\|\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}} - \textstyle\sum_{i\in S_{\mathsf{ind}}} \mathbf{Y}_i\mathbf{u}_{\mathsf{ind},i} - \mathbf{k}_{\mathbf{p}}\right\| \le O\big(\sigma_{\mathsf{pp}} \cdot m^{15/2} \log q \log \ell + N \cdot m^{3/2}\sigma_{\mathsf{pp}}\big),$$

since $|S_{\mathsf{ind}}| \le N$. Since $\mathbf{e} \in \mathbb{Z}^{4m^5}$ and $\|\mathbf{e}\| \le \sqrt{m}\sigma_{\mathsf{LWE}}$, we have

$$\left|\mathbf{e}^{\mathsf{T}}\big(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}} - \textstyle\sum_{i\in S_{\mathsf{ind}}} \mathbf{Y}_i\mathbf{u}_{\mathsf{ind},i} - \mathbf{k}_{\mathbf{p}}\big)\right| \le O\big(\sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}} \cdot m^{13} \log q \log \ell + N \cdot \sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}} \cdot m^7\big).$$

For the remaining sum, $\left|\mathbf{e}_i^{\mathsf{T}}\mathbf{u}_{\mathsf{ind},i}\right| \le m \cdot \sqrt{m}\sigma_{\mathsf{ht}} \cdot \sqrt{m}\sigma_{\mathsf{pp}} = O(m^2\sigma_{\mathsf{ht}}\sigma_{\mathsf{pp}})$, so $\left|\sum_{i\in S_{\mathsf{ind}}} \mathbf{e}_i^{\mathsf{T}}\mathbf{u}_{\mathsf{ind},i}\right| \le O(N \cdot m^2\sigma_{\mathsf{ht}}\sigma_{\mathsf{pp}})$. Taken all together, we have

$$\left|\mathbf{e}^{\mathsf{T}}\left(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}} - \textstyle\sum_{i\in S_{\mathsf{ind}}} \mathbf{Y}_i\mathbf{u}_{\mathsf{ind},i} - \mathbf{k}_{\mathbf{p}}\right) + \sum_{i\in S_{\mathsf{ind}}} \mathbf{e}_i^{\mathsf{T}}\mathbf{u}_{\mathsf{ind},i}\right| < O\big(N \cdot \sigma_{\mathsf{pp}}\sigma_{\mathsf{ht}}\sigma_{\mathsf{LWE}} \cdot m^{13} \log q \log \ell\big).$$

Correctness follows by setting $q > N \cdot O(\sigma_{\mathsf{pp}}\sigma_{\mathsf{ht}}\sigma_{\mathsf{LWE}} \cdot m^{13} \log q \log \ell)$. □

**Theorem 3.3** (Static Security). *Suppose $\Pi_{\mathsf{NIZK}}$ is zero-knowledge and simulation-extractable, and that the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption holds with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$. Suppose also that $n \ge \lambda$, $m \ge 3n \log q$, $q > 2$ is prime, $\sigma_{\mathsf{pp}} \ge m \log n$, $\sigma_{\mathsf{LWE}} > \log m$, and $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$. Then, Construction 3.1 satisfies static security (Definition 2.14).*

*Proof.* Take any efficient and admissible adversary $\mathcal{A}$ for the static security game, and suppose $\mathcal{A}$ wins the game with non-negligible advantage $\varepsilon$. We now define a sequence of hybrid experiments parameterized by a bit $b \in \{0,1\}$ and a precision parameter $\kappa = \kappa(\lambda)$ (used for the explainable Gaussian sampler Explain). We omit the index $\kappa$ when the hybrid definition is independent of the choice of $\kappa$.

- $\mathsf{Hyb}_0^{(b)}$: This is the static security game with challenge bit $b \in \{0,1\}$:
  - **Setup phase:** On input the security parameter $1^\lambda$, the challenger samples

    $$\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda),\ \mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3}.$$

    If $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$, the challenger sets $\mathbf{R} = \mathbf{0}^{m\times 2m^3}$ instead. Let

    $$\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^{\mathsf{T}} \otimes \mathbf{G} \in \mathbb{Z}_q^{n\times 4m^5}$$

    and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. The challenger gives the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ to $\mathcal{A}$ and initializes a counter $\mathsf{ctr} = 0$.

    *Random oracle queries:* Additionally, the challenger maintains a table for the random oracle $H$, which is initially empty. Whenever algorithm $\mathcal{A}$ queries $H$ on a tuple $(\tau, i, j)$, the challenger checks whether $(\tau, i, j)$ already appears in the table. If so, it replies with the stored value. Otherwise, it samples $\gamma \xleftarrow{\mathrm{R}} \{0,1\}^\rho$, stores $(\tau, i, j) \mapsto \gamma$ in the table, and replies with $\gamma$.
  - **Key-generation queries:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $\mathbf{Y}_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5\times m}$, and sends $\mathsf{pk}_{\mathsf{ctr}} = \mathbf{T}_{\mathsf{ctr}} = \mathbf{B}\mathbf{Y}_{\mathsf{ctr}}$ to $\mathcal{A}$.
  - **Hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, \mathsf{ind})$ for $\mathsf{ind} \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^{\mathsf{T}}, \hat{\mathbf{c}}_2^{\mathsf{T}}, \hat{c}_3)$. Then it checks that

    $$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1,$$

    and replies with $\perp$ if the check fails. If the check passes, the challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}^m$, sets $\hat{\mathbf{e}}_{\mathsf{ht}} = \mathbf{0}^m$ if $\|\hat{\mathbf{e}}_{\mathsf{ht}}\| > \sqrt{m} \cdot \sigma_{\mathsf{ht}}$, and replies with $\hat{\mathbf{c}}_1^{\mathsf{T}}\mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^{\mathsf{T}}$.

– **Challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$ over $N$ variables and specifies either a public key $\mathsf{pk}_i^* = \mathbf{T}_i^*$ or a counter value $\mathsf{ctr}_i$ for each $i \in [N]$. For each $i \in [N]$ where $\mathcal{A}$ specified a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$, the challenger sets $\mathsf{pk}_i^* = \mathbf{T}_i^* = \mathbf{T}_{\mathsf{ctr}_i}$. The challenger constructs the challenge ciphertext $\mathsf{ct}^*$ by first sampling

$$\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n\,,\ \mathbf{e} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{LWE}}}^{4m^5}\,,\ \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}\,,\ \mathbf{K}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{4m^5\times m}\,,\ \mathbf{k}_{\mathbf{p}} \xleftarrow{R} \{0,1\}^{4m^5}.$$

If $\|\mathbf{e}\| > \sqrt{m}\cdot\sigma_{\mathsf{LWE}}$, the challenger sets $\mathbf{e} = \mathbf{0}^{4m^5}$. Next, the challenger samples $\tau^* \xleftarrow{R} \{0,1\}^\lambda$ and for $i \in [\ell],\ j \in S_i$ it computes

$$\mathbf{u}_{i,j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; H(\tau^*, i, j)),$$

sets $\mathbf{u}_{i,j} = \mathbf{0}^m$ if $\|\mathbf{u}_{i,j}\| > \sqrt{m}\cdot\sigma_{\mathsf{pp}}$, and sets $\mathbf{d}_{i,j} = \mathbf{T}_j^*\cdot\mathbf{u}_{i,j}$. For each $i \in [\ell]$, the challenger sets $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and

$$\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell] \in \mathbb{Z}_q^{n\times\ell} \quad \text{where} \quad \forall i \in [\ell] : \mathbf{m}_i = \sum_{j\in S_i} \mathbf{d}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \in \mathbb{Z}_q^n.$$

The challenger computes the commitment $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \in \mathbb{Z}_q^{n\times m}$. Then, the challenger sets $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$, $\mathbf{c}_2^\intercal = \mathbf{s}^\intercal(\mathbf{A}+\mathbf{C}) + \mathbf{e}^\intercal\mathbf{K}_{\mathbf{A}}$, and $c_3 = \mathbf{s}^\intercal\mathbf{p} + \mathbf{e}^\intercal\mathbf{k}_{\mathbf{p}} + b\cdot\lfloor q/2\rfloor$. Finally, the challenger sets $\mathsf{ct}_{\mathsf{main}}^* = (\tau^*, \mathbf{A}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$, computes the proof

$$\pi_{\mathsf{ct}}^* \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}^*, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e})),$$

and gives $\mathsf{ct}^* = (\pi_{\mathsf{ct}}^*, \mathsf{ct}_{\mathsf{main}}^*)$ to $\mathcal{A}$.

– **(Post-challenge) hint-generation queries:** Algorithm $\mathcal{A}$ can continue to make hint queries on ciphertexts $(\hat{\mathsf{ct}}, \mathsf{ind})$ where $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$. The challenger answers these queries exactly like the pre-challenge hint-generation queries.

– **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. Note that if $\mathcal{A}$ aborts early, then the output of the experiment is 0.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$ except for the following modifications:

  – The challenger samples the random oracle seed $\tau^* \xleftarrow{R} \{0,1\}^\lambda$ for the challenge ciphertext at setup time.

  – In the challenge phase, after the adversary outputs the policy $P = (S_1, \ldots, S_\ell)$, the challenger then samples $\gamma_{i,j}^* \xleftarrow{R} \{0,1\}^\rho$ for $i \in [\ell]$, $j \in S_i$. Next, the challenger sets

$$\mathbf{u}_{i,j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; \gamma_{i,j}^*).$$

  Next, the challenger answers queries to the random oracle $H$ using the following modified procedure:

  – If the adversary queries $H$ on a triple $(\tau^*, \cdot, \cdot)$ before the challenge phase, the challenger halts the experiment with output 0.

  – If the adversary queries $H$ on the triple $(\tau^*, i, j)$ for $i \in [\ell]$, $j \in S_i$ after the challenge phase, the challenger replies with $\gamma_{i,j}^*$. All other queries to the random oracle are handled as before.

Finally, the challenger also halts with output 0 if any pre-challenge hint-generation query includes $\tau^*$ as part of the ciphertext.

- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger replaces the NIZK proof in the challenge ciphertext with a simulated NIZK proof. Specifically, the challenger proceeds as follows:
  - During setup, the challenger samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$.
  - During the challenge phase, after constructing $\mathsf{ct}^*_{\mathsf{main}}$, the challenger now runs

$$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}})).$$

- $\mathsf{Hyb}_3^{(b)}$: Same as $\mathsf{Hyb}_2^{(b)}$ except the challenger extracts the encryption randomness for hint-generation queries. Specifically, when the adversary makes a hint-generation query $(\hat{\mathsf{ct}}, \mathsf{ind})$ for an index $\mathsf{ind} \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{c}_3)$. Then, after checking that $\hat{\pi}_{\mathsf{ct}}$ verifies, it computes

$$(\hat{\mathbf{s}}, \hat{\mathbf{e}}) = \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}).$$

  If $\|\hat{\mathbf{e}}\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \hat{\mathbf{s}}^\intercal \mathbf{B} + \hat{\mathbf{e}}^\intercal$, the challenger outputs 0.

- $\mathsf{Hyb}_4^{(b)}$: Same as $\mathsf{Hyb}_3^{(b)}$ except the challenger no longer truncates $\hat{\mathbf{e}}_{\mathsf{ht}}$. That is, when answering hint-generation queries, the challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and replies with $\hat{\mathbf{c}}_1^\intercal \mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$ (irrespective of $\|\hat{\mathbf{e}}_{\mathsf{ht}}\|$).

- $\mathsf{Hyb}_5^{(b)}$: Same as $\mathsf{Hyb}_4^{(b)}$ except the challenger now simulates the hints using the extracted randomness. In particular, for a hint-generation query $(\hat{\mathsf{ct}}, \mathsf{ind})$ where $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ and $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{c}_3)$, the challenger first checks the proof $\hat{\pi}_{\mathsf{ct}}$. If the proof verifies, then the challenger extracts $(\hat{\mathbf{s}}, \hat{\mathbf{e}})$ as in $\mathsf{Hyb}_4^{(b)}$. Then it samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and replies with $\hat{\mathbf{s}}^\intercal \mathbf{T}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$. At this point, the challenger's logic in the experiment no longer requires knowledge of the secret keys $\mathbf{Y}_{\mathsf{ind}}$.

- $\mathsf{Hyb}_{6,\kappa}^{(b)}$: Same as $\mathsf{Hyb}_5^{(b)}$ except the challenger derives the randomness $\gamma^*_{i,j}$ using the explainable Gaussian sampler for the first honest user in each clause. Specifically, for each $i \in [\ell]$, let $\mathsf{idx}_i \in S_i$ be the smallest index in $S_i$ for which $\mathcal{A}$ specified a counter value $\mathsf{ctr}_i$ in the challenge phase (i.e., the smallest uncorrupted index in $S_i$, which exists since $\mathcal{A}$ is admissible). In the challenge phase, the challenger now samples

$$\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}} \quad \text{and} \quad \gamma^*_{i,\mathsf{idx}_i} \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, \mathbf{u}_{i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}}).$$

  As in $\mathsf{Hyb}_5^{(b)}$, the challenger sets $\mathbf{u}_{i,\mathsf{idx}_i} = \mathbf{0}^m$ if $\|\mathbf{u}_{i,\mathsf{idx}_i}\| > \sqrt{m} \cdot \sigma_{\mathsf{pp}}$.

- $\mathsf{Hyb}_{7,\kappa}^{(b)}$: Same as $\mathsf{Hyb}_{6,\kappa}^{(b)}$ except the challenger no longer truncates $\mathbf{u}_{i,\mathsf{idx}_i}$ after sampling. In addition, in the setup phase, the challenger no longer checks if $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$, and when constructing the challenge ciphertext $\mathsf{ct}^*$, it no longer checks if $\|\mathbf{e}\| > \sqrt{m}\sigma_{\mathsf{LWE}}$.

- $\mathsf{Hyb}_{8,\kappa}^{(b)}$: Same as $\mathsf{Hyb}_{7,\kappa}^{(b)}$ except the challenger samples $\mathbf{T}_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ when answering key-generation queries.

- $\mathsf{Hyb}^{(b)}_{9,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{8,\kappa}$ except the challenger changes how it samples $\mathbf{u}_{i,\mathsf{idx}_i}$ and $\mathbf{d}_{i,\mathsf{idx}_i}$ for the special index $\mathsf{idx}_i \in S_i$ in each min-term $i \in [\ell]$. Specifically, in the challenge phase, the challenger now samples $\mathbf{d}_{i,\mathsf{idx}_i} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow (\mathbf{T}^*_{\mathsf{idx}_i})^{-1}_{\sigma_{\mathsf{pp}}}(\mathbf{d}_{i,\mathsf{idx}_i})$. The remaining $\mathbf{u}_{i,j}$ for $j \in S_i \setminus \{\mathsf{idx}_i\}$ are sampled as in $\mathsf{Hyb}^{(b)}_{8,\kappa}$.

- $\mathsf{Hyb}^{(b)}_{10,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{9,\kappa}$ except the challenger samples $(\mathbf{T}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$ when answering key-generation queries. In the challenge phase, it labels $\mathsf{td}^*_i = \mathsf{td}_{\mathsf{ctr}_i}$.

- $\mathsf{Hyb}^{(b)}_{11,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{10,\kappa}$ except the challenger uses SamplePre to sample $\mathbf{u}_{i,\mathsf{idx}_i}$ in the challenge phase. Specifically, for each $i \in [\ell]$, the challenger samples $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow \mathsf{SamplePre}(\mathbf{T}^*_{\mathsf{idx}_i}, \mathsf{td}^*_{\mathsf{idx}_i}, \mathbf{d}_{i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}})$ instead of sampling $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow (\mathbf{T}^*_{\mathsf{idx}_i})^{-1}_{\sigma_{\mathsf{pp}}}(\mathbf{d}_{i,\mathsf{idx}_i})$.

- $\mathsf{Hyb}^{(b)}_{12,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{11,\kappa}$ except the challenger programs $\mathbf{d}_{i,\mathsf{idx}_i}$ in each min-term $i \in [\ell]$. Specifically, in the challenge phase, the challenger starts by sampling $\mathbf{d}^*_i \xleftarrow{R} \mathbb{Z}_q^n$ for each $i \in [\ell]$. Then, for all $i \in [\ell]$ and $j \in S_i \setminus \{\mathsf{idx}_i\}$, it samples $\gamma^*_{i,j} \xleftarrow{R} \{0,1\}^\rho$ and computes $\mathbf{u}_{i,j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; \gamma^*_{i,j})$, $\mathbf{d}_{i,j} = \mathbf{T}^*_j \cdot \mathbf{u}_{i,j}$. Afterwards, it sets

$$\mathbf{d}_{i,\mathsf{idx}_i} = \mathbf{d}^*_i - \sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{d}_{i,j} + \mathbf{A}\mathbf{v}_i - \mathbf{p}$$

  for each $i \in [\ell]$.

- $\mathsf{Hyb}^{(b)}_{13,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{12,\kappa}$ except the challenger samples $\mathbf{k}_{\mathbf{p}} \xleftarrow{R} \{0,1\}^{4m^5}$ at setup time and sets $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}$. At the start of the challenge phase, the challenger defines $\mathbf{D}^* = [\mathbf{d}^*_1 \mid \cdots \mid \mathbf{d}^*_\ell]$, sets $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$, and sets $\mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}} - \mathbf{C}$. The challenger uses the matrix $\mathbf{C}$ when constructing the challenge ciphertext.

- $\mathsf{Hyb}^{(b)}_{14,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{13,\kappa}$ except the challenger samples the component $\mathbf{c}_1^\intercal \xleftarrow{R} \mathbb{Z}_q^{4m^5}$, then computes $\mathbf{c}_2^\intercal = \mathbf{c}_1^\intercal \mathbf{K}_{\mathbf{A}}$ and $c_3 = \mathbf{c}_1^\intercal \mathbf{k}_{\mathbf{p}} + b \cdot \lfloor q/2 \rfloor$.

- $\mathsf{Hyb}^{(b)}_{15,\kappa}$: Same as $\mathsf{Hyb}^{(b)}_{14,\kappa}$ except the challenger samples $\mathbf{p} \xleftarrow{R} \mathbb{Z}_q^n$ and $c_3 \xleftarrow{R} \mathbb{Z}_q$.

We write $\mathsf{Hyb}^{(b)}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid $\mathsf{Hyb}^{(b)}_i$ with adversary $\mathcal{A}$ (and an implicit security parameter $\lambda$). We now bound the difference between the output distributions of each adjacent pair of hybrid experiments.

**Lemma 3.4.** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}^{(b)}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_1(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* By construction, $\mathsf{Hyb}^{(b)}_0$ and $\mathsf{Hyb}^{(b)}_1$ are identical unless algorithm $\mathcal{A}$ queries the random oracle on a triple of the form $(\tau^*, \cdot, \cdot)$ before the challenge phase or if it makes a pre-challenge hint-generation query on a ciphertext that includes $\tau^*$. Conditioned on these events not happening, the distributions of random oracle queries in the two experiments are identically distributed. It suffices to bound the probability that the adversary makes such a query:

- Let $Q_{\mathsf{ro}}$ be a bound on the total number of random oracle queries and $Q_{\mathsf{ht}}$ be a bound on the number of pre-challenge hint-generation queries that algorithm $\mathcal{A}$ makes.

- Since the challenger samples $\tau^* \xleftarrow{R} \{0,1\}^\lambda$ in the challenge phase, the probability that $\mathcal{A}$ queries $H$ on a triple of the form $(\tau^*, \cdot, \cdot)$ before the challenge phase, or that $\mathcal{A}$ submits a pre-challenge hint-generation query on a ciphertext that includes $\tau^*$, is at most $(Q_{\mathsf{ro}} + Q_{\mathsf{ht}})/2^\lambda$.

Since $\mathcal{A}$ is efficient, $(Q_{\mathsf{ro}} + Q_{\mathsf{ht}})/2^\lambda = \mathsf{negl}(\lambda)$ and the lemma follows. □

**Lemma 3.5.** *Suppose* $\Pi_{\mathsf{NIZK}}$ *satisfies zero-knowledge. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Suppose $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \delta$ for some non-negligible $\delta$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks zero-knowledge of $\Pi_{\mathsf{NIZK}}$:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{B}$ receives the common reference string $\mathsf{crs}_{\mathsf{NIZK}}$ from the zero-knowledge challenger and starts running $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ executes the setup phase exactly as in $\mathsf{Hyb}_1^{(b)}$ except it uses $\mathsf{crs}_{\mathsf{NIZK}}$ from the challenger instead of sampling its own. Algorithm $\mathcal{B}$ gives the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ to $\mathcal{A}$.
- **Key-generation, hint, and challenge phases:** Algorithm $\mathcal{B}$ executes these phases exactly as in $\mathsf{Hyb}_1^{(b)}$, except when constructing the proof $\pi_{\mathsf{ct}}^*$ in the challenge ciphertext, algorithm $\mathcal{B}$ submits $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}^*, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e}))$ to the zero-knowledge challenger to get $\pi_{\mathsf{ct}}^*$.
- **Output phase:** At the end of the game, if algorithm $\mathcal{A}$ outputs $b' \in \{0,1\}$, $\mathcal{B}$ also outputs $b'$.

By construction, $C_{\mathsf{ct}}((\mathbf{B}, \mathsf{ct}_{\mathsf{main}}^*, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e})) = 1$. If the zero-knowledge challenger sampled the CRS and answered proof queries honestly, then algorithm $\mathcal{B}$ simulates $\mathsf{Hyb}_1^{(b)}$. If the zero-knowledge challenger sampled the CRS with a trapdoor and answered proof queries with simulated proofs, then algorithm $\mathcal{B}$ simulates $\mathsf{Hyb}_2^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks zero-knowledge with the same advantage $\delta$. □

**Lemma 3.6.** *Suppose* $\Pi_{\mathsf{NIZK}}$ *satisfies simulation-sound extractability. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Suppose $|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]| = \delta$ for some non-negligible $\delta$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks simulation-sound extractability of $\Pi_{\mathsf{NIZK}}$:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{B}$ gets the common reference string $\mathsf{crs}_{\mathsf{NIZK}}$ from the simulation-sound extractability challenger and starts running $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ executes the setup phase exactly as in $\mathsf{Hyb}_2^{(b)}$ except it uses $\mathsf{crs}_{\mathsf{NIZK}}$ from the challenger instead of sampling its own. Algorithm $\mathcal{B}$ gives the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ to $\mathcal{A}$.
- **Key-generation, hint, and challenge phases:** Algorithm $\mathcal{B}$ executes these phases as in $\mathsf{Hyb}_2^{(b)}$, except when simulating the proof $\pi_{\mathsf{ct}}^*$ in the challenge ciphertext, algorithm $\mathcal{B}$ submits the tuple $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}^*, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}))$ to the simulation-sound extractability challenger to get $\pi_{\mathsf{ct}}^*$.
- **Output phase:** Algorithm $\mathcal{B}$ samples $j \xleftarrow{R} [Q]$ (where $Q$ is the number of hint-generation queries made by $\mathcal{A}$), parses the $j^{\text{th}}$ hint-generation query as $((\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}}), \mathsf{ind})$ and then gives the triple $(C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}})$ to the simulation-sound extractability challenger.

**Admissibility.** The only query that algorithm $\mathcal{B}$ makes to the simulation-sound extractability challenger is $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}))$, where $\mathsf{ct}^*_{\mathsf{main}} = (\tau^*, \mathbf{A}, \mathbf{c}_1^\mathsf{T}, \mathbf{c}_2^\mathsf{T}, c_3)$ contains the algebraic components of the challenge ciphertext $\mathsf{ct}^*$. Algorithm $\mathcal{B}$ outputs $\big((C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}})), \hat{\pi}_{\mathsf{ct}}\big)$, where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^\mathsf{T}, \hat{\mathbf{c}}_2^\mathsf{T}, \hat{c}_3)$ comes from the $j^{\text{th}}$ hint-generation query $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$. We now argue that this is an admissible strategy:

- If the $j^{\text{th}}$ hint-generation query $\hat{\mathsf{ct}}$ is a pre-challenge query, then both experiments abort with output 0 if $\hat{\tau} = \tau^*$. Thus, we can assume $\hat{\tau} \neq \tau^*$, which means $\hat{\mathsf{ct}}_{\mathsf{main}} \neq \mathsf{ct}^*_{\mathsf{main}}$ and the statements differ.
- If the $j^{\text{th}}$ hint-generation query is a post-challenge query, then $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$. This means that either $\hat{\pi}_{\mathsf{ct}} \neq \pi^*_{\mathsf{ct}}$ or $\hat{\mathsf{ct}}_{\mathsf{main}} \neq \mathsf{ct}^*_{\mathsf{main}}$.

In both cases, algorithm $\mathcal{B}$ outputs either a statement that is different from the one it provided to the challenger during the challenge phase or a proof that is different from the one it received from the challenger. Thus, algorithm $\mathcal{B}$ always outputs an admissible triple.

**Advantage of $\mathcal{B}$.** By construction, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_2^{(b)}$ for $\mathcal{A}$. Moreover, $\mathsf{Hyb}_2^{(b)}$ and $\mathsf{Hyb}_3^{(b)}$ differ only if for some hint-generation query $((\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}}), \mathsf{ind})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^\mathsf{T}, \hat{\mathbf{c}}_2^\mathsf{T}, \hat{c}_3)$, the extracted pair $(\hat{\mathbf{s}}, \hat{\mathbf{e}})$ satisfies $\|\hat{\mathbf{e}}\| > \sqrt{m}\cdot\sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\mathsf{T} \neq \hat{\mathbf{s}}^\mathsf{T}\mathbf{B} + \hat{\mathbf{e}}^\mathsf{T}$. Thus, with probability at least $\delta$, there exists $j^* \in [Q]$ such that the $(j^*)^{\text{th}}$ hint-generation query satisfies the following:

- The proof verifies: $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1$.
- The extracted pair $(\hat{\mathbf{s}}, \hat{\mathbf{e}})$ satisfies $C_{\mathsf{ct}}((\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), (\hat{\mathbf{s}}, \hat{\mathbf{e}})) = 0$.

Since algorithm $\mathcal{B}$ samples $j \xleftarrow{\mathrm{R}} [Q]$ uniformly, algorithm $\mathcal{B}$ wins the simulation-sound extractability game with probability at least $\delta/Q$, which is non-negligible since $Q = \mathsf{poly}(\lambda)$. □

**Lemma 3.7.** *Suppose $m \geq \lambda$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The two hybrids differ only in whether the challenger truncates $\hat{\mathbf{e}}_{\mathsf{ht}}$ when answering hint-generation queries. For each hint-generation query, the challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$, so by Lemma 2.3 and a union bound, the probability that $\|\hat{\mathbf{e}}_{\mathsf{ht}}\| > \sqrt{m}\sigma_{\mathsf{ht}}$ is at most $m \cdot 2^{-m} = \mathsf{negl}(\lambda)$. Since the adversary makes at most a polynomial number of hint-generation queries, the lemma now follows by a hybrid argument. □

**Lemma 3.8.** *Suppose $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* For every hint-generation query $((\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}}), \mathsf{ind})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^\mathsf{T}, \hat{\mathbf{c}}_2^\mathsf{T}, \hat{c}_3)$ that algorithm $\mathcal{A}$ makes (and where the challenger does not halt), the extracted pair $(\hat{\mathbf{s}}, \hat{\mathbf{e}})$ satisfies $\hat{\mathbf{c}}_1^\mathsf{T} = \hat{\mathbf{s}}^\mathsf{T}\mathbf{B} + \hat{\mathbf{e}}^\mathsf{T}$ and $\|\hat{\mathbf{e}}\| \leq \sqrt{m}\sigma_{\mathsf{LWE}}$. Therefore, the reply in $\mathsf{Hyb}_4^{(b)}$ satisfies

$$\hat{\mathbf{c}}_1^\mathsf{T}\mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\mathsf{T} = \hat{\mathbf{s}}^\mathsf{T}\mathbf{B}\mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}^\mathsf{T}\mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\mathsf{T} = \hat{\mathbf{s}}^\mathsf{T}\mathbf{T}_{\mathsf{ind}} + \hat{\mathbf{e}}^\mathsf{T}\mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\mathsf{T},$$

whereas the challenger's reply in $\mathsf{Hyb}_5^{(b)}$ is $\hat{\mathbf{s}}^\mathsf{T}\mathbf{T}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\mathsf{T}$. The only difference between the two hybrids is the omission of $\hat{\mathbf{e}}^\mathsf{T}\mathbf{Y}_{\mathsf{ind}}$ in $\mathsf{Hyb}_5^{(b)}$. Since $\mathbf{Y}_{\mathsf{ind}} \in \{0,1\}^{4m^5 \times m}$ and $\|\hat{\mathbf{e}}\| \leq \sqrt{m}\sigma_{\mathsf{LWE}}$, $\|\hat{\mathbf{e}}^\mathsf{T}\mathbf{Y}_{\mathsf{ind}}\| \leq 4m^5 \cdot \sqrt{m}\sigma_{\mathsf{LWE}} =$

$4m^{11/2}\sigma_{\mathsf{LWE}}$. Since $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$, we can appeal to Lemma 2.5 and conclude that the distributions of $\hat{\mathbf{e}}^\mathsf{T}\mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\mathsf{T}$ and $\hat{\mathbf{e}}_{\mathsf{ht}}^\mathsf{T}$ are statistically indistinguishable when $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$. Since the adversary makes at most a polynomial number of hint-generation queries, the lemma now follows by a hybrid argument. □

**Lemma 3.9.** *Suppose* (Sample, Explain) *is an explainable discrete Gaussian sampler (Definition 2.11). Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_5^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{6,\kappa}^{(b)}(\mathcal{A}) = 1]| = \ell/\kappa + \mathsf{negl}(\lambda).$$

*Proof.* The lemma follows by the explainability property (Definition 2.11) applied to each pair $(\mathbf{u}_{i,\mathsf{idx}_i}, \gamma^*_{i,\mathsf{idx}_i})$ for $i \in [\ell]$. When $\gamma^*_{i,\mathsf{idx}_i} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and $\mathbf{u}_{i,\mathsf{idx}_i} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; \gamma^*_{i,\mathsf{idx}_i})$, the pair is distributed according to the specification in $\mathsf{Hyb}_5^{(b)}$. Alternatively, when $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}}$ and $\gamma^*_{i,\mathsf{idx}_i} \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, \mathbf{u}_{i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}})$, the pair is distributed according to the specification of $\mathsf{Hyb}_{6,\kappa}^{(b)}$. By Definition 2.11, the statistical distance between each of these pairs is $1/\kappa + \mathsf{negl}(\lambda)$. Since $\ell = \mathsf{poly}(\lambda)$, the claim now follows by a hybrid argument. □

**Lemma 3.10.** *Suppose $m \geq \lambda$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_{6,\kappa}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{7,\kappa}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The two hybrids differ only in whether the challenger truncates $\mathbf{u}_{i,\mathsf{idx}_i}$ for some $i \in [\ell]$, $\mathbf{R}$, or $\mathbf{e}$ when their norms exceed the prescribed thresholds. We bound the probability of each of these events:

- Since $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}}$, by Lemma 2.3 and a union bound, $\|\mathbf{u}_{i,\mathsf{idx}_i}\| > \sqrt{m} \cdot \sigma_{\mathsf{pp}}$ with probability at most $m \cdot 2^{-m}$ for each $i \in [\ell]$.
- Since $\mathbf{R} \leftarrow D^{m \times 2m^3}_{\mathbb{Z},\sigma_{\mathsf{pp}}}$, by Lemma 2.3 and a union bound, $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$ with probability at most $2^{-m} \cdot 2m^4$.
- Since $\mathbf{e} \leftarrow D^{4m^5}_{\mathbb{Z},\sigma_{\mathsf{LWE}}}$, by Lemma 2.3 and a union bound, $\|\mathbf{e}\| > \sqrt{m}\sigma_{\mathsf{LWE}}$ with probability at most $2^{-m} \cdot 4m^5$.

Since $\ell = \mathsf{poly}(\lambda)$ and $m \geq \lambda$, the lemma follows by a union bound over all of these events. □

**Lemma 3.11.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} > \log m$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_{7,\kappa}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{8,\kappa}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The only difference between these two experiments is the distribution of $\mathbf{T}_{\mathsf{ctr}}$. In $\mathsf{Hyb}_{7,\kappa}^{(b)}$, the challenger samples $\mathbf{Y}_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$ and sets $\mathbf{T}_{\mathsf{ctr}} = \mathbf{B}\mathbf{Y}_{\mathsf{ctr}}$, whereas in $\mathsf{Hyb}_{8,\kappa}^{(b)}$, the challenger samples $\mathbf{T}_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ directly. These two distributions are statistically indistinguishable by Corollary 2.9. Since the adversary makes polynomially-many key-generation queries, the lemma now follows by a standard hybrid argument. □

**Lemma 3.12.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} > \log m$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_{8,\kappa}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{9,\kappa}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows directly by Lemma 2.4. Specifically, for each $i \in [\ell]$, with overwhelming probability over the choice of $\mathbf{T}^*_{\mathsf{idx}_i} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, the following distributions are statistically indistinguishable:

$$\left\{(\mathbf{u}_{i,\mathsf{idx}_i}, \mathbf{T}^*_{\mathsf{idx}_i} \cdot \mathbf{u}_{i,\mathsf{idx}_i}) : \mathbf{u}_{i,\mathsf{idx}_i} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}}\right\} \text{ and } \left\{(\mathbf{u}_{i,\mathsf{idx}_i}, \mathbf{d}_{i,\mathsf{idx}_i}) : \begin{array}{c} \mathbf{d}_{i,\mathsf{idx}_i} \xleftarrow{R} \mathbb{Z}_q^n, \\ \mathbf{u}_{i,\mathsf{idx}_i} \leftarrow (\mathbf{T}^*_{\mathsf{idx}_i})^{-1}_{\sigma_{\mathsf{pp}}}(\mathbf{d}_{i,\mathsf{idx}_i}) \end{array}\right\}.$$

The left and right distributions correspond to $\mathsf{Hyb}^{(b)}_{8,\kappa}$ and $\mathsf{Hyb}^{(b)}_{9,\kappa}$ respectively. The lemma follows by a hybrid argument since $\ell = \mathsf{poly}(\lambda)$. □

**Lemma 3.13.** *Suppose $m \geq 3n \log q$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{9,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{10,\kappa}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The only difference between these two experiments is the distribution of $\mathbf{T}_{\mathsf{ctr}}$. In $\mathsf{Hyb}^{(b)}_{9,\kappa}$, the challenger samples $\mathbf{T}_{\mathsf{ctr}} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ uniformly when answering each key-generation query, whereas in $\mathsf{Hyb}^{(b)}_{10,\kappa}$, the challenger samples $(\mathbf{T}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$. By the trapdoor distribution property in Lemma 2.6, the marginal distribution of $\mathbf{T}_{\mathsf{ctr}}$ in the two experiments is statistically indistinguishable. Since the adversary makes polynomially-many key-generation queries, the lemma now follows by a standard hybrid argument. □

**Lemma 3.14.** *Suppose $m \geq 3n \log q$ and $\sigma_{\mathsf{pp}} \geq m \log n$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{10,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{11,\kappa}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The only difference between these two experiments is the distribution of $\mathbf{u}_{i,\mathsf{idx}_i}$ for $i \in [\ell]$:

$$\mathsf{Hyb}^{(b)}_{10,\kappa} : \mathbf{u}_{i,\mathsf{idx}_i} \leftarrow (\mathbf{T}^*_{\mathsf{idx}_i})^{-1}_{\sigma_{\mathsf{pp}}}(\mathbf{d}_{i,\mathsf{idx}_i})$$
$$\mathsf{Hyb}^{(b)}_{11,\kappa} : \mathbf{u}_{i,\mathsf{idx}_i} \leftarrow \mathsf{SamplePre}(\mathbf{T}^*_{\mathsf{idx}_i}, \mathsf{td}^*_{\mathsf{idx}_i}, \mathbf{d}_{i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}}).$$

Since the challenger samples $(\mathbf{T}^*_{\mathsf{idx}_i}, \mathsf{td}^*_{\mathsf{idx}_i}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$ in both hybrids, by Lemma 2.6, these two distributions are statistically close. The lemma now follows via a standard hybrid argument over the $\ell = \mathsf{poly}(\lambda)$ key-generation queries the adversary makes. □

**Lemma 3.15.** *For all polynomials $\kappa = \kappa(\lambda)$ and $b \in \{0,1\}$, $\Pr[\mathsf{Hyb}^{(b)}_{11,\kappa}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}^{(b)}_{12,\kappa}(\mathcal{A}) = 1]$.*

*Proof.* In $\mathsf{Hyb}^{(b)}_{11,\kappa}$, the challenger samples $\mathbf{d}_{i,\mathsf{idx}_i} \xleftarrow{R} \mathbb{Z}_q^n$. In $\mathsf{Hyb}^{(b)}_{12,\kappa}$, the challenger sets

$$\mathbf{d}^*_i = \mathbf{d}^*_i - \textstyle\sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{d}_{i,j} + \mathbf{A}\mathbf{v}_i - \mathbf{p},$$

where $\mathbf{d}^*_i \xleftarrow{R} \mathbb{Z}_q^n$ (and independent of all other components). In both cases, the distribution of $\mathbf{d}_{i,\mathsf{idx}_i}$ is uniform over $\mathbb{Z}_q^n$ and the claim holds. □

**Lemma 3.16.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} > \log m$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{12,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{13,\kappa}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* We first show that the matrix $\mathbf{C}$ used in the challenge ciphertext is identically distributed in $\mathsf{Hyb}^{(b)}_{12,\kappa}$ and $\mathsf{Hyb}^{(b)}_{13,\kappa}$. In $\mathsf{Hyb}^{(b)}_{12,\kappa}$, the challenger computes $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ where $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell]$ and $\mathbf{m}_i = \sum_{j \in S_i} \mathbf{d}_{i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_i$. By construction of $\mathbf{d}_{i,\mathsf{idx}_i}$, this means for all $i \in [\ell]$,

$$\begin{aligned}\mathbf{m}_i &= \textstyle\sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{d}_{i,j} + \mathbf{d}_{i,\mathsf{idx}_i} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \\ &= \textstyle\sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{d}_{i,j} + \left(\mathbf{d}^*_i - \sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{d}_{i,j} + \mathbf{A}\mathbf{v}_i - \mathbf{p}\right) + \mathbf{p} - \mathbf{A}\mathbf{v}_i \\ &= \mathbf{d}^*_i.\end{aligned}$$

Thus $\mathbf{M} = \mathbf{D}^*$ and $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$ in $\mathsf{Hyb}^{(b)}_{12,\kappa}$, which exactly matches the value of $\mathbf{C}$ sampled in $\mathsf{Hyb}^{(b)}_{13,\kappa}$. Hence the only remaining differences between $\mathsf{Hyb}^{(b)}_{12,\kappa}$ and $\mathsf{Hyb}^{(b)}_{13,\kappa}$ are the distributions of $\mathbf{p}$ and $\mathbf{A}$:

- In $\mathsf{Hyb}^{(b)}_{12,\kappa}$, $\mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ are sampled uniformly and independently of $(\mathbf{W}, \mathbf{R}, \mathbf{C}, \mathbf{k}_{\mathbf{p}}, \mathbf{K}_{\mathbf{A}})$.
- In $\mathsf{Hyb}^{(b)}_{13,\kappa}$, $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}$ and $\mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}} - \mathbf{C}$, where $\mathbf{k}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ and $\mathbf{K}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$.

Let $\mathbf{K} = [\mathbf{k}_{\mathbf{p}} \mid \mathbf{K}_{\mathbf{A}}] \in \{0,1\}^{4m^5 \times (m+1)}$. By construction, the joint distribution of $(\mathbf{p}, \mathbf{A} + \mathbf{C})$ is uniform over $\mathbb{Z}_q^{n \times (m+1)}$ in $\mathsf{Hyb}^{(b)}_{12,\kappa}$ and equals $\mathbf{B}\mathbf{K}$ in $\mathsf{Hyb}^{(b)}_{13,\kappa}$. By Corollary 2.9 the distribution

$$\{(\mathbf{W}, \mathbf{R}, \mathbf{B}\mathbf{K}, \mathbf{e}^\mathsf{T}\mathbf{K}) : \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m \times 2m^3},\ \mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times (m+1)}\}$$

is statistically indistinguishable from

$$\{(\mathbf{W}, \mathbf{R}, \mathbf{U}, \mathbf{e}^\mathsf{T}\mathbf{K}) : \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m \times 2m^3},\ \mathbf{U} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times (m+1)},\ \mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times (m+1)}\}.$$

The first distribution corresponds to the distribution of the public parameters and challenge ciphertext components in $\mathsf{Hyb}^{(b)}_{13,\kappa}$ while the second corresponds to those in $\mathsf{Hyb}^{(b)}_{12,\kappa}$. We conclude that $\mathsf{Hyb}^{(b)}_{12,\kappa}$ and $\mathsf{Hyb}^{(b)}_{13,\kappa}$ are statistically indistinguishable. □

**Lemma 3.17.** *Suppose the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$ holds. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{13,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{14,\kappa}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Suppose $|\Pr[\mathsf{Hyb}^{(b)}_{13,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{14,\kappa}(\mathcal{A}) = 1]| = \delta$ for some non-negligible $\delta$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathbf{W}, \mathbf{R}, \mathbf{c}^\mathsf{T})$ from the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE challenger and starts running $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ sets $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\mathsf{T} \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times 4m^5}$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. Then algorithm $\mathcal{B}$ samples

  $$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda),\ \mathbf{k}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5},\ \tau^* \xleftarrow{\mathrm{R}} \{0,1\}^\lambda.$$

  Algorithm $\mathcal{B}$ defines $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}$ and sets $\mathsf{ctr} = 0$. It gives $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ to $\mathcal{A}$.
- **Random oracle queries:** Algorithm $\mathcal{B}$ initializes a table for queries. Whenever algorithm $\mathcal{A}$ queries $H$ on a tuple $(\tau, i, j)$, algorithm $\mathcal{B}$ checks whether $(\tau, i, j)$ already appears in the table. If so, it replies with the stored value. Otherwise, it samples $\gamma \xleftarrow{\mathrm{R}} \{0,1\}^\rho$, stores $(\tau, i, j) \mapsto \gamma$ in the table, and replies with $\gamma$. However, if a tuple $(\tau^*, \cdot, \cdot)$ is queried before the challenge phase, algorithm $\mathcal{B}$ aborts and outputs 0. If $(\tau^*, i, j)$ is queried after the challenge phase, algorithm $\mathcal{B}$ replies with $\gamma^*_{i,j}$ which are sampled in the challenge phase.

- **Key-generation queries:** When algorithm $\mathcal{A}$ makes a key-generation query, algorithm $\mathcal{B}$ increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $(\mathbf{T}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$, and sends $\mathsf{pk}_{\mathsf{ctr}} = \mathbf{T}_{\mathsf{ctr}}$ to $\mathcal{A}$.
- **Hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint query $(\hat{\mathsf{ct}}, \mathsf{ind})$ for $\mathsf{ind} \le \mathsf{ctr}$, algorithm $\mathcal{B}$ parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{c}_3)$. Then it checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1,$$

  and replies with $\perp$ if the check fails. For pre-challenge hint queries, algorithm $\mathcal{B}$ aborts and outputs 0 if $\hat{\tau} = \tau^*$. Otherwise, algorithm $\mathcal{B}$ computes

$$(\hat{\mathbf{s}}, \hat{\mathbf{e}}) = \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}).$$

  If $\|\hat{\mathbf{e}}\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \hat{\mathbf{s}}^\intercal \mathbf{B} + \hat{\mathbf{e}}^\intercal$, algorithm $\mathcal{B}$ outputs 0. Otherwise, algorithm $\mathcal{B}$ samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow D^m_{\mathbb{Z}, \sigma_{\mathsf{ht}}}$ and sends $\hat{\mathbf{s}}^\intercal \mathbf{T}_{\mathsf{ind}} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$ to $\mathcal{A}$.
- **Challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$ over $N$ variables and specifies either a public key $\mathsf{pk}_i^* = \mathbf{T}_i^*$ or a counter value $\mathsf{ctr}_i$ for each $i \in [N]$. For each $i \in [N]$ where $\mathcal{A}$ specified a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i^* = \mathbf{T}_i^* = \mathbf{T}_{\mathsf{ctr}_i}$ and $\mathsf{td}_i^* = \mathsf{td}_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ constructs the challenge ciphertext $\mathsf{ct}^*$ as follows:
  - Sample $\mathbf{d}_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ for each $i \in [\ell]$ and $\mathbf{K}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$.
  - Let $\mathbf{D}^* = [\mathbf{d}_1^* \mid \cdots \mid \mathbf{d}_\ell^*]$ and set $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$. Let $\mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}} - \mathbf{C}$.
  - For each $i \in [\ell]$, let $\mathsf{idx}_i \in S_i$ be the smallest index in $S_i$ for which $\mathcal{A}$ specified a counter value $\mathsf{ctr}_i$ in the challenge phase. For $i \in [\ell]$ and $j \in S_i \backslash \{\mathsf{idx}_i\}$, algorithm $\mathcal{B}$ samples $\gamma_{i,j}^* \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and sets

$$\mathbf{u}_{i,j} = \mathsf{Sample}(1^\lambda, 1^m, \sigma_{\mathsf{pp}}; \gamma_{i,j}^*) \quad \text{and} \quad \mathbf{d}_{i,j} = \mathbf{T}_j^* \cdot \mathbf{u}_{i,j},$$

    unless $\|\mathbf{u}_{i,j}\| > \sqrt{m} \cdot \sigma_{\mathsf{pp}}$, in which case it sets $\mathbf{d}_{i,j} = \mathbf{0}^n$.
  - For each $i \in [\ell]$, algorithm $\mathcal{B}$ computes $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and sets

$$\mathbf{d}_{i,\mathsf{idx}_i} = \mathbf{d}_i^* - \sum_{j \in S_i \backslash \{\mathsf{idx}_i\}} \mathbf{d}_{i,j} + \mathbf{A}\mathbf{v}_i - \mathbf{p}.$$

    Then, algorithm $\mathcal{B}$ samples $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow \mathsf{SamplePre}(\mathbf{T}_{\mathsf{idx}_i}^*, \mathsf{td}_{\mathsf{idx}_i}^*, \mathbf{d}_{i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}})$ and computes the randomness $\gamma_{i,\mathsf{idx}_i}^* \leftarrow \mathsf{Explain}(1^\lambda, 1^\kappa, \mathbf{u}_{i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}})$ for each $i \in [\ell]$. Algorithm $\mathcal{B}$ sets $\mathsf{ct}_{\mathsf{main}}^* = (\tau^*, \mathbf{A}, \mathbf{c}^\intercal, \mathbf{c}^\intercal \mathbf{K}_{\mathbf{A}}, \mathbf{c}^\intercal \mathbf{k}_{\mathbf{p}} + b \cdot \lfloor q/2 \rfloor)$ and then computes

$$\pi_{\mathsf{ct}}^* \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}^*, \sqrt{m} \cdot \sigma_{\mathsf{LWE}})).$$

  Finally, algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* = (\pi_{\mathsf{ct}}^*, \mathsf{ct}_{\mathsf{main}}^*)$.
- **Post-challenge hint-generation queries:** Algorithm $\mathcal{B}$ handles post-challenge hint queries exactly the same as pre-challenge queries, except instead of aborting when $\hat{\tau} = \tau^*$, it aborts when $\hat{\mathsf{ct}} = \mathsf{ct}^*$.
- **Output phase:** At the end of the game, if algorithm $\mathcal{A}$ outputs $b' \in \{0,1\}$, $\mathcal{B}$ also outputs $b'$.

By construction, algorithm $\mathcal{B}$ efficiently simulates the setup phase, the public keys, and the hint queries exactly as in $\mathsf{Hyb}^{(b)}_{13,\kappa}$ or $\mathsf{Hyb}^{(b)}_{14,\kappa}$. Consider the distribution of the challenge ciphertext components in the two experiments. By construction of $\mathbf{A}, \mathbf{C}$, and $\mathbf{p}$, if $\mathbf{c}^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$ where $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{e} \leftarrow D^{4m^5}_{\mathbb{Z},\sigma_{\mathsf{LWE}}}$, then

$$\mathbf{c}^\intercal\mathbf{K}_\mathbf{A} = \mathbf{s}^\intercal\mathbf{B}\mathbf{K}_\mathbf{A} + \mathbf{e}^\intercal\mathbf{K}_\mathbf{A} = \mathbf{s}^\intercal(\mathbf{A} + \mathbf{C}) + \mathbf{e}^\intercal\mathbf{K}_\mathbf{A}$$
$$\mathbf{c}^\intercal\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor = \mathbf{s}^\intercal\mathbf{B}\mathbf{k}_\mathbf{p} + \mathbf{e}^\intercal\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor = \mathbf{s}^\intercal\mathbf{p} + \mathbf{e}^\intercal\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor,$$

so algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{13,\kappa}$. If $\mathbf{c}^\intercal \xleftarrow{R} \mathbb{Z}_q^{4m^5}$, then algorithm $\mathcal{B}$ constructs the challenge ciphertext exactly as described in $\mathsf{Hyb}^{(b)}_{14,\kappa}$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{14,\kappa}$. Thus, algorithm $\mathcal{B}$ breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE with advantage $\delta$. □

**Lemma 3.18.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} > \log m$. Then, for all polynomials $\kappa = \kappa(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{14,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{15,\kappa}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The hybrids are indistinguishable by Corollary 2.9 and Lemma 2.8. Specifically, by setting $\ell = 2m^2$ and $s = \sigma_{\mathsf{pp}}$ in Corollary 2.9, we have that the following distributions are statistically indistinguishable for any $\mathbf{c}_1 \in \mathbb{Z}_q^{4m^5}$:

$$\left\{(\mathbf{W}, \mathbf{R}, \mathbf{B}\mathbf{k}_\mathbf{p}, \mathbf{c}_1^\intercal\mathbf{k}_\mathbf{p}) : \begin{array}{c}\mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D^{m\times 2m^3}_{\mathbb{Z},\sigma_{\mathsf{pp}}} \\ \mathbf{k}_\mathbf{p} \xleftarrow{R} \{0,1\}^{4m^5}\end{array}\right\} \text{ and } \left\{(\mathbf{W}, \mathbf{R}, \mathbf{p}, \mathbf{c}_1^\intercal\mathbf{k}_\mathbf{p}) : \begin{array}{c}\mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D^{m\times 2m^3}_{\mathbb{Z},\sigma_{\mathsf{pp}}} \\ \mathbf{k}_\mathbf{p} \xleftarrow{R} \{0,1\}^{4m^5},\ \mathbf{p} \xleftarrow{R} \mathbb{Z}_q^n\end{array}\right\},$$

where $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G}$. By Lemma 2.8, the following distributions are statistically close:

$$\left\{(\mathbf{c}_1^\intercal, \mathbf{c}_1^\intercal\mathbf{k}_\mathbf{p}) : \mathbf{c}_1 \xleftarrow{R} \mathbb{Z}_q^{4m^5},\ \mathbf{k}_\mathbf{p} \xleftarrow{R} \{0,1\}^{4m^5}\right\} \text{ and } \left\{(\mathbf{c}_1^\intercal, c_3) : \mathbf{c}_1 \xleftarrow{R} \mathbb{Z}_q^{4m^5},\ c_3 \xleftarrow{R} \mathbb{Z}_q\right\}.$$

The left and right distributions correspond to $\mathsf{Hyb}^{(b)}_{14,\kappa}$ and $\mathsf{Hyb}^{(b)}_{15,\kappa}$ respectively. The lemma follows by appealing to Corollary 2.9 to switch the first pair of distributions, and then appealing to Lemma 2.8 to switch the second pair of distributions. □

**Completing the proof.** To complete the proof of Theorem 3.3, we first observe that the challenger's behavior in $\mathsf{Hyb}^{(b)}_{15,\kappa}(\mathcal{A})$ is independent of $b \in \{0,1\}$, so $\Pr[\mathsf{Hyb}^{(0)}_{15,\kappa}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}^{(1)}_{15,\kappa}(\mathcal{A}) = 1]$. Now, recall the assumption that algorithm $\mathcal{A}$ wins the static security game with advantage $\varepsilon$. This implies

$$|\Pr[\mathsf{Hyb}^{(0)}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1)}_0(\mathcal{A}) = 1]| = \varepsilon(\lambda).$$

Since $\varepsilon$ is non-negligible, there exists some polynomial $\kappa' = \kappa'(\lambda)$ such that for infinitely-many $\lambda \in \mathbb{N}$, $\varepsilon(\lambda) \geq 1/\kappa'(\lambda)$. Let $L$ be an upper bound on the number of min-terms in a challenge policy that $\mathcal{A}$ can output (which we can always upper bound by the running time of $\mathcal{A}$) and take

$$\kappa(\lambda) = (2L+1) \cdot \kappa'(\lambda).$$

By Lemmas 3.4 to 3.18, we have for all $\lambda \in \mathbb{N}$ (and recalling that for $i \le 5$, $\mathsf{Hyb}^{(b)}_{i,\kappa}(\mathcal{A}) \equiv \mathsf{Hyb}^{(b)}_{i}(\mathcal{A})$),

$$\begin{aligned}\varepsilon(\lambda) = |\Pr[\mathsf{Hyb}^{(0)}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1)}_0(\mathcal{A}) = 1]| &\le \sum_{i=0}^{14} |\Pr[\mathsf{Hyb}^{(0)}_{i,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(0)}_{i+1,\kappa}(\mathcal{A}) = 1]| \\ &\quad + |\Pr[\mathsf{Hyb}^{(0)}_{15,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1)}_{15,\kappa}(\mathcal{A}) = 1]| \\ &\quad + \sum_{i=0}^{14} |\Pr[\mathsf{Hyb}^{(1)}_{i+1,\kappa}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(1)}_{i,\kappa}(\mathcal{A}) = 1]| \\ &\le 2L/\kappa(\lambda) + \delta(\lambda),\end{aligned}$$

where $\delta(\lambda) = \mathsf{negl}(\lambda)$ is a negligible function. Thus, for infinitely many $\lambda \in \mathbb{N}$, we have that

$$2L/\kappa(\lambda) + \delta(\lambda) \ge \varepsilon(\lambda) \ge 1/\kappa'(\lambda) = (2L+1)/\kappa(\lambda).$$

Hence $\delta(\lambda) \ge 1/\kappa(\lambda)$ for infinitely many $\lambda$, contradicting the fact that $\delta$ is negligible, which proves the theorem. □

**Parameter instantiation.** We now provide one possible way to set the lattice parameters in Construction 3.1 so as to satisfy the conditions in Theorems 3.2 and 3.3. Let $\lambda$ be the security parameter, $N \le 2^\lambda$ be the number of variables, and $\varepsilon \in (0,1)$ be a constant. We set

$$\begin{aligned} n &= (\lambda \log N)^{1/\varepsilon} \cdot \mathsf{poly}(\log \lambda, \log N) \\ m &= 3n \log q \\ \sigma_{\mathsf{LWE}} &= \mathsf{poly}(n) \\ \sigma_{\mathsf{pp}} &= m \log n \\ \sigma_{\mathsf{ht}} &= 2^\lambda \cdot m^{11/2} \cdot \sigma_{\mathsf{LWE}} \\ q &= 2^\lambda \cdot \mathsf{poly}(n, N). \end{aligned}$$

Note that $q = 2^{O(n^\varepsilon)}$, so we require a sub-exponential modulus-to-noise ratio. In conjunction with the NIZK from Fact 2.2, we obtain the following instantiation:

**Corollary 3.19** (Optimal Distributed Monotone-Policy Encryption for DNFs)**.** *Let $\lambda$ be a security parameter. Then, under the decomposed LWE assumption with a sub-exponential modulus-to-noise ratio, there exists a statically-secure distributed monotone-policy encryption scheme in the random oracle model for policies that can be computed by arbitrary DNF formulas over up to $N = 2^\lambda$ variables. The sizes of the public parameters, public keys, and ciphertext are all* $\mathsf{poly}(\lambda)$*. Moreover, the scheme has a transparent setup.*

**Remark 3.20** (Public-Key Aggregation)**.** Similar to [CW26, AGY26], we can extend Construction 3.1 to support an explicit preprocessing algorithm that takes a collection of public keys together with an policy and outputs a succinct encryption key. Given the succinct encryption key, the encryption algorithm runs in time that is independent of the number of parties and the size of the policy. In the context of Construction 3.1, the preprocessing algorithm would take the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ and the access policy $P \in \mathcal{P}_{\mathsf{DNF}}$, and output the commitment $\mathbf{C}$ as computed in Eq. (3.2). Specifically, the commitment $\mathbf{C}$ in the encryption algorithm of Construction 3.1 only depends on the public keys and the policy. Given the commitment $\mathbf{C}$, computing the remaining ciphertext components $\mathbf{c}_1, \mathbf{c}_2, c_3, \pi_{\mathsf{ct}}$ requires time that is independent of the size of the number of users and the size of the policy.

**Corollary 3.21** (Succinct Computational Secret Sharing for DNFs)**.** *Let $\lambda$ be a security parameter. Then, under the decomposed LWE assumption with a sub-exponential modulus-to-noise ratio, there exists a statically-secure* reusable *succinct secret sharing scheme in the random oracle model for policies that can be computed by arbitrary DNF formulas over up to $N = 2^\lambda$ variables. The sizes of the public long-term share, secret long-term shares, and online share are all* $\mathsf{poly}(\lambda)$.

*Proof (Sketch).* Each user's share consists of the public parameters pp for the distributed monotone-policy encryption scheme, a user public key $\mathsf{pk}_i = \mathbf{T}_i$, and the associated secret key $\mathsf{sk}_i = \mathbf{Y}_i$. The (non-reusable) message-dependent share of a message $\mu$ is an encryption of $\mu$ under pp to the full set of users. Share reconstruction maps to decryption. If the underlying distributed monotone-policy encryption has optimal parameters, then the sizes of the public parameters, the user's public key, the user's secret key, and the ciphertext are all independent of the policy size (and the number of parties). This ensures each user's share is succinct.

As written, there is a problem because decryption (and thus, share reconstruction) requires knowledge of the public keys of all of the users. Including this information as part of each user's share or as a global public share would compromise succinctness. However, in the setting of secret sharing, we can achieve share compression by working in the random oracle model. Specifically, we modify the scheme as follows:

- Instead of sampling the matrix commitment parameters $\mathsf{pp}_{\mathsf{com}}$ as in Construction 3.1, the dealer instead samples them from a statistically indistinguishable distribution where the associated matrix $\mathbf{B}$ has a trapdoor (known to the dealer). This can be done via the procedure described in [AGY26, §3.2].
- Next, the dealer derives each user's public key $\mathbf{T}_i$ using the random oracle and samples their secret key $\mathbf{Y}_i$ to be $\mathbf{Y}_i \leftarrow \mathbf{B}^{-1}(\mathbf{T}_i)$ using the trapdoor for $\mathbf{B}$. Observe that this still satisfies the invariant $\mathbf{B}\mathbf{Y}_i = \mathbf{T}_i$ needed for correctness.
- When parties come together to reconstruct, they can derive the public keys $\mathbf{T}_1, \ldots, \mathbf{T}_N$ for all of the users from the random oracle. Decryption then follows via the same procedure as for the distributed monotone-policy encryption scheme.

The security analysis follows via a similar argument as that for the distributed monotone-policy encryption scheme. We provide a brief sketch below:

- In the first hybrid, we switch the matrix commitment parameters $\mathsf{pp}_{\mathsf{com}}$ back to the distribution in Construction 3.1 (where the matrix $\mathbf{B}$ no longer has a trapdoor). Since the matrix $\mathbf{B}$ no longer has a trapdoor, the challenger reverts to first sampling the secret keys $\mathbf{Y}_i \in \mathbb{Z}_q^{4m^5 \times m}$ from a discrete Gaussian distribution and then programming the random oracle to output $\mathbf{T}_i = \mathbf{B}\mathbf{Y}_i$ for the $i^{\text{th}}$ user's public key. These two distributions are statistically indistinguishable by [GPV08]. We note that the [GPV08] sampling lemmas also hold for the matrix $\mathbf{B}$ induced by $\mathsf{pp}_{\mathsf{com}}$ [CW26, Corollary 2.10].
- At this point, the shares consist of the public parameters, the public keys, the secret keys, and ciphertexts for Construction 3.1. By the same argument as in the proof of Theorem 3.3, we can argue that the shares of any unauthorized set of users computationally hide the underlying secret. In fact, we can even show the stronger definition where the shares computationally hide the message even if the adversary can request online shares from honest users derived from message-dependent shares of its choosing, so long as the message-dependent shares are distinct from the challenge share. This is the analog of hint queries in the encryption setting and is important for capturing reusability of the offline shares. □

# 4 Distributed Monotone-Policy Encryption for DNFs in the Plain Model

In this section, we show that Construction 3.1 can be re-balanced to yield a non-trivial construction of distributed monotone-policy encryption in the plain model. The re-balancing blows up the ciphertext size by a factor of $k \cdot L^{1/2}$, where $k$ is a bound on the size of the largest min-term and $L$ is the number of min-terms. For simplicity, we will assume that $\ell = L^{1/2}$ is an integer and every min-term has exactly $k$ variables in it. In Remark 4.5, we describe how to improve this blowup to $\max\{(kL)^{1/2}, k\}$.

**Construction 4.1** (Distributed Monotone-Policy Encryption)**.** Let $\lambda$ be a security parameter. We define the following:

- Let $(n, m, q, \sigma_{\mathsf{LWE}})$ be LWE parameters (which may be functions of $\lambda$). Let $\sigma_{\mathsf{pp}}, \sigma_{\mathsf{ht}} > 0$ be Gaussian width parameters.
- Let $\mathcal{P}_{\mathsf{DNF}}$ be a family of monotone Boolean functions on up to $2^\lambda$-bit inputs described by a DNF formula. As in Construction 3.1, we will represent a DNF as a collection of sets $S_1, \ldots, S_L \subseteq [N]$, but we additionally require that $|S_i| = k$ and $L = \ell^2$ for some $\ell \in \mathbb{N}$.
- Let $C_{\mathsf{ct},k,\ell}$ be the Boolean circuit that takes a statement $(\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \beta)$, a witness $(\mathbf{s}_w, \mathbf{e}_w)_{w\in[\ell]}$, and outputs 1 if for all $w \in [\ell]$, $\mathbf{c}_{w,1}^\intercal = \mathbf{s}_w^\intercal \mathbf{B} + \mathbf{e}_w^\intercal$ and $\|\mathbf{e}_w\| \le \beta$, where $\mathsf{ct}_{\mathsf{main}} = (\tau, \mathbf{A}, \mathbf{u}, \{\mathbf{c}_{w,1}^\intercal, \mathbf{c}_{w,2}^\intercal, c_{w,3}\}_{w\in[\ell]})$, with $\tau \in \{0,1\}^\lambda$, $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\{\mathbf{u}_{i,j}\}_{i\in[\ell],j\in[k]} \in (\mathbb{Z}_q^m)^{k\ell}$, $\mathbf{c}_{w,1}^\intercal \in \mathbb{Z}_q^{4m^5}$, $\mathbf{c}_{w,2}^\intercal \in \mathbb{Z}_q^m$, and $c_{w,3} \in \mathbb{Z}_q$. Note that $C_{\mathsf{ct},k,\ell}$ can be computed by a circuit of depth $\mathrm{poly}(n, m, \log q, \log \ell)$.
- Let $\Pi_{\mathsf{NIZK}} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.TrapSetup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify}, \mathsf{NIZK.Sim}, \mathsf{NIZK.Extract})$ be a simulation-sound extractable NIZK argument for a family $C = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ where $C_\lambda$ contains the circuits $C_{\mathsf{ct},k,\ell}$.

We construct a distributed monotone-policy encryption scheme $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ for $\mathcal{P}_{\mathsf{DNF}}$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter $\lambda$, the setup algorithm samples

$$\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda),\ \mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3}.$$

  If $\|\mathbf{R}\| > \sqrt{m}\cdot\sigma_{\mathsf{pp}}$, set $\mathbf{R} = \mathbf{0}^{m\times 2m^3}$. Let

$$\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathrm{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n\times 4m^5} \quad \text{and} \quad \mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B}).$$

  Finally, it outputs $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$.

- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$, the key-generation algorithm samples $\mathbf{Y} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5\times m}$ and computes $\mathbf{T} = \mathbf{B}\mathbf{Y} \in \mathbb{Z}_q^{n\times m}$. It outputs the public key $\mathsf{pk} = \mathbf{T}$ and the secret key $\mathsf{sk} = \mathbf{Y}$.
- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \mu)$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$, a policy $P = (S_1, \ldots, S_L) \in \mathcal{P}_{\mathsf{DNF}}$, a collection of public keys $\mathsf{pk}_j = \mathbf{T}_j$ for $j \in [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm first parses the policy into chunks $P_1, \ldots, P_\ell$, where $P_w = (S_{w,1}, \ldots, S_{w,\ell}) = (S_{(w-1)\ell+1}, \ldots, S_{w\ell})$ for $w \in [\ell]$. It samples the following shared components:

$$\mathbf{u}_{i,j} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^m \ \ \forall i \in [\ell], j \in [k],\ \ \tau \xleftarrow{\mathrm{R}} \{0,1\}^\lambda, \mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times m},$$

  sets $\mathbf{u}_{i,j} = \mathbf{0}^m$ if $\|\mathbf{u}_{i,j}\| > \sqrt{m}\cdot\sigma_{\mathsf{pp}}$. Then, for each $w \in [\ell]$, it does the following:

1. Sample $\mathbf{s}_w \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{K}_{\mathsf{A}}^{(w)} \xleftarrow{R} \{0,1\}^{4m^5 \times m}, \mathbf{k}_{\mathsf{p}}^{(w)} \xleftarrow{R} \{0,1\}^{4m^5}, \mathbf{e}_w \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{LWE}}}^{4m^5}$ and set $\mathbf{e}_w = \mathbf{0}^{4m^5}$ if $\|\mathbf{e}_w\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$.
2. Let $\mathbf{T}_{w,i,j}$ be the $j^{\text{th}}$ public key indicated by $S_{w,i}$. For $i \in [\ell], j \in [k]$ set $\mathbf{d}_{w,i,j} = \mathbf{T}_{w,i,j} \cdot \mathbf{u}_{i,j}$.
3. For each $i \in [\ell]$, set $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and
$$\mathbf{M}_w = [\mathbf{m}_{w,1} \mid \cdots \mid \mathbf{m}_{w,\ell}] \in \mathbb{Z}_q^{n \times \ell} \text{ where } \forall i \in [\ell] : \mathbf{m}_{w,i} = \sum_{j \in [k]} \mathbf{d}_{w,i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \in \mathbb{Z}_q^n. \quad (4.1)$$
Compute $\mathbf{C}_w = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}_w) \in \mathbb{Z}_q^{n \times m}$.
4. Set $\mathbf{c}_{w,1}^\mathsf{T} = \mathbf{s}_w^\mathsf{T}\mathbf{B} + \mathbf{e}_w^\mathsf{T}$, $\mathbf{c}_{w,2}^\mathsf{T} = \mathbf{s}_w^\mathsf{T}(\mathbf{A} + \mathbf{C}_w) + \mathbf{e}_w^\mathsf{T}\mathbf{K}_{\mathsf{A}}^{(w)}$, and $c_{w,3} = \mathbf{s}_w^\mathsf{T}\mathbf{p} + \mathbf{e}_w^\mathsf{T}\mathbf{k}_{\mathsf{p}}^{(w)} + \mu \cdot \lfloor q/2 \rfloor$.

Finally, it sets
$$\mathsf{ct}_{\mathsf{main}} = (\tau, \mathbf{A}, \{\mathbf{u}_{i,j}\}_{i \in [\ell], j \in [k]}, \{\mathbf{c}_{w,1}^\mathsf{T}, \mathbf{c}_{w,2}^\mathsf{T}, c_{w,3}\}_{w \in [\ell]}) \quad (4.2)$$
and computes the proof
$$\pi_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), (\mathbf{s}_w, \mathbf{e}_w)_{w \in [\ell]}),$$
and outputs $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$.

- $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$, a secret key $\mathsf{sk}_i = \mathbf{Y}_i$, and a ciphertext $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}}$ is parsed as in Eq. (4.2), the hint-generation algorithm checks that
$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \pi_{\mathsf{ct}}) = 1.$$
If the check fails, it outputs $\perp$. Otherwise, for $w \in [\ell]$ it samples $\mathbf{e}_w \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}^m$ and sets $\mathbf{e}_w = \mathbf{0}^m$ if $\|\mathbf{e}_w\| > \sqrt{m} \cdot \sigma_{\mathsf{ht}}$. It outputs $\{\mathbf{c}_{w,1}^\mathsf{T}\mathbf{Y}_i + \mathbf{e}_w^\mathsf{T}\}_{w \in [\ell]}$.
- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct})$: On input the parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$, a policy $P = (S_1, \ldots, S_L) \in \mathcal{P}_{\mathsf{DNF}}$, a collection of public keys $\mathsf{pk}_j = \mathbf{T}_j$ for $j \in [N]$, a list of decryption hints $\mathsf{ht}_i = \{\mathbf{h}_{w,i}^\mathsf{T}\}_{w \in [\ell]}$ for $i \in T \subseteq [N]$, and a ciphertext $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}}$ is parsed as in Eq. (4.2), the decryption algorithm outputs $\perp$ if $P(T) = 0$. Otherwise, it parses the policy into chunks $P_1, \ldots, P_\ell$, where $P_w = (S_{w,1}, \ldots, S_{w,\ell})$, and takes $(w^*, \mathsf{ind}) \in [\ell] \times [\ell]$ to be the smallest pair such that $S_{w^*,\mathsf{ind}} \subseteq T$. Then, for $i \in [\ell], j \in [k]$ it sets $\mathbf{d}_{w^*,i,j} = \mathbf{T}_{w^*,i,j} \cdot \mathbf{u}_{i,j}$, where $\mathbf{T}_{w^*,i,j}$ denotes the $j^{\text{th}}$ public key indicated by $S_{w^*,i}$. It defines $\mathbf{M}_{w^*}$ as in Eq. (4.1) and computes
$$\mathbf{v}_{\mathsf{ind}} = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, \mathsf{ind}) \quad \text{and} \quad \mathbf{z}_{\mathsf{ind}} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}_{w^*}, \mathsf{ind}).$$
Finally, let $\mathsf{idx}_i \in [N]$ be the $i^{\text{th}}$ index in $S_{w^*,\mathsf{ind}}$ and compute
$$\mu' = c_{w^*,3} - \mathbf{c}_{w^*,1}^\mathsf{T}\mathbf{z}_{\mathsf{ind}} - \mathbf{c}_{w^*,2}^\mathsf{T}\mathbf{v}_{\mathsf{ind}} + \sum_{i \in [k]} \mathbf{h}_{w^*,\mathsf{idx}_i}^\mathsf{T} \cdot \mathbf{u}_{\mathsf{ind},i}.$$
Output 0 if $-q/4 < \mu' < q/4$ and 1 otherwise.

**Theorem 4.2** (Correctness). *Suppose $q > k \cdot O(m^{13}\sigma_{\mathsf{pp}}\sigma_{\mathsf{ht}}\sigma_{\mathsf{LWE}} \log q \log L)$ and $\Pi_{\mathsf{NIZK}}$ is complete. Then, Construction 4.1 is correct.*

**Theorem 4.3** (Static Security). *Suppose $\Pi_{\mathsf{NIZK}}$ is zero-knowledge and simulation-extractable, and that the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption holds with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$. Suppose also that $n \geq \lambda$, $m \geq 3n \log q$, $q > 2$ is prime, $\sigma_{\mathsf{pp}} \geq m \log n$, $\sigma_{\mathsf{LWE}} > \log m$, and $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$. Then, Construction 4.1 satisfies static security (Definition 2.14).*

We defer the correctness and security proofs to Appendices A.1 and A.2.

**Parameter instantiation.** We now provide one possible way to set the lattice parameters in Construction 4.1 so as to satisfy the conditions in Theorems 4.2 and 4.3. Let $\lambda$ be the security parameter, $k \leq 2^\lambda$ be a bound on the maximum size of a min-term, $L \leq 2^\lambda$ be a bound on the number of min-terms and $\varepsilon \in (0,1)$ be a constant. We set

$$\begin{aligned} n &= (\lambda \log k)^{1/\varepsilon} \cdot \mathsf{poly}(\log \lambda, \log k) \\ m &= 3n \log q \\ \sigma_{\mathsf{LWE}} &= \mathsf{poly}(n) \\ \sigma_{\mathsf{pp}} &= m \log n \\ \sigma_{\mathsf{ht}} &= 2^\lambda \cdot m^{11/2} \cdot \sigma_{\mathsf{LWE}} \\ q &= 2^\lambda \cdot \mathsf{poly}(n, k, \log L). \end{aligned}$$

Note that $q = 2^{O(n^\varepsilon)}$, so we require a sub-exponential modulus-to-noise ratio. In conjunction with the NIZK from Fact 2.2, we obtain the following instantiation:

**Corollary 4.4** (Distributed Monotone-Policy Encryption for DNFs in the Plain Model)**.** *Let $\lambda$ be a security parameter. Then, under the decomposed LWE assumption with a sub-exponential modulus-to-noise ratio, there exists a statically-secure distributed monotone-policy encryption scheme for policies that can be computed by arbitrary DNF formulas with up to $L = 2^\lambda$ min-terms and up to a maximum min-term size of $k = 2^\lambda$ variables. The size of the public parameters and public keys is $\mathsf{poly}(\lambda)$. The size of a ciphertext is $k \cdot L^{1/2} \cdot \mathsf{poly}(\lambda)$. Moreover, the scheme has a transparent setup.*

**Remark 4.5** (Better Re-Balancing)**.** One can be more careful when re-balancing Construction 4.1 to get a slightly better ciphertext size. In particular, grouping $\ell = \lceil (L/k)^{1/2} \rceil$ min-terms together yields $\lceil L/\ell \rceil$ total groups, which gives a ciphertext of size $(k\ell + L/\ell) \cdot \mathsf{poly}(\lambda) = (kL)^{1/2} \cdot \mathsf{poly}(\lambda) = |P|^{1/2} \cdot \mathsf{poly}(\lambda)$ so long as $k < L$. When $k \geq L$, the ciphertext size becomes $k \cdot \mathsf{poly}(\lambda)$.

# 5 Adaptively-Secure Unbounded Distributed Broadcast Encryption

Recall that the notion of distributed broadcast encryption [WQZDF10, BZ14] corresponds to the special case of distributed monotone-policy encryption for $k$-DNFs where $k = 1$ (and where there is no need for a separate hint-generation procedure). The notion of static security for distributed monotone-policy encryption (Definition 2.14) corresponds to the notion of semi-static security for distributed broadcast encryption [GW09, KMW23]. Specifically, in this setting, the adversary can adaptively choose the subset of users associated with the challenge ciphertext, but is not allowed to corrupt any users. A classic result of Gentry and Waters [GW09] shows how to generically transform any broadcast encryption scheme with semi-static security into an adaptively secure scheme in the random oracle model. The work of [KMW23] showed that the same transformation also applies to distributed broadcast encryption. Recently, the work of [HWW25] also showed how to implement the Gentry-Waters blueprint in the plain model using projective PRGs, but the use of projective PRGs introduces linear-size public parameters to the scheme (when instantiated with lattice-based projective PRGs). As such, existing lattice-based distributed broadcast encryption schemes with adaptive security [GY26a, AMR26] only support a bounded number of users.

**A re-balanced version of Gentry-Waters.** In this section, we describe our "re-balanced" variant of the classic Gentry-Waters transformation that works in the plain model. Unlike [HWW25], our re-balanced

version is information-theoretic and thus does not require using new primitives such as projective PRGs. Notably, this means that if the underlying semi-statically-secure distributed broadcast encryption scheme supports an unbounded number of users, the transformed scheme also supports an unbounded number of users. On the flip side, the ciphertext size in the transformed scheme scales polynomially with $|S|$ whereas the transformations from [GW09, KMW23, HWW25] preserved the ciphertext size of the underlying scheme.

Taken together, we obtain an adaptively-secure distributed broadcast encryption scheme in the plain model that supports an a priori unbounded number of users with ciphertext size $|S|^{2/3}$, where $S$ is the number of users in the broadcast set. In Remark 5.12, we describe how to plausibly improve this to $|S|^{1/2}$ through a non-black-box composition of Construction 4.1 with the Gentry-Waters compiler. We start by recalling the definition of a distributed broadcast encryption scheme.

**Distributed broadcast encryption.** We begin by recalling the notion of distributed broadcast encryption [WQZDF10, BZ14, KMW23]. In distributed broadcast encryption, each user is associated with an index and the key-generation algorithm depends on this index. Moreover, the scheme only supports encryption to collections of public keys associated with *distinct* indices. In our definition, we relax the succinctness requirement to allow the ciphertext size to depend polynomially on the size of the set. This is a relaxation compared to prior works on distributed broadcast encryption, but was commonly considered in early works on centralized broadcast encryption.

**Definition 5.1** (Distributed Broadcast Encryption [BZ14, KMW23, WW25])**.** Let $\lambda$ be a security parameter. A distributed broadcast encryption scheme $\Pi_{\mathsf{DBE}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, N) \to \mathsf{pp}$: On input the security parameter $\lambda$ and the number of users $N$ (in *binary*), the setup algorithm outputs the public parameters pp. We assume that pp contains $1^\lambda$ and $N$.
- $\mathsf{KeyGen}(\mathsf{pp}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the public parameters pp and an index $i \in [N]$, the key-generation algorithm outputs a public key $\mathsf{pk}_i$ and secret key $\mathsf{sk}_i$.
- $\mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu) \to \mathsf{ct}$: On input the public parameters pp, a collection of public keys $\mathsf{pk}_i$ for $i \in S$, and a message $\mu \in \{0,1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mathsf{ct}, (i^*, \mathsf{sk}_{i^*})) \to \mu$: On input the public parameters pp, a collection of public keys $\mathsf{pk}_i$ for $i \in S$, a ciphertext ct, and a secret key $\mathsf{sk}_{i^*}$ for an index $i^*$, the decryption algorithm outputs a message $\mu \in \{0,1\}$.

We require that $\Pi_{\mathsf{DBE}}$ satisfy the following properties:

- **Correctness:** For all parameters $\lambda, N \in \mathbb{N}$, all pp in the support of $\mathsf{Setup}(1^\lambda, N)$, all sets $S \subseteq [N]$, all indices $i^* \in S$, all public keys $\mathsf{pk}_i$ for $i \in S \setminus \{i^*\}$, and all messages $\mu \in \{0,1\}$,

$$\Pr\left[\mathsf{Decrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mathsf{ct}, (i^*, \mathsf{sk}_{i^*})) = \mu : \begin{array}{c} (\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i^*) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu) \end{array}\right] = 1.$$

- **Adaptive security:** For a security parameter $\lambda$, a bound $N$, and a bit $b \in \{0,1\}$, we define the adaptive security game between an adversary $\mathcal{A}$ and a challenger as follows:
  - **Setup phase:** The challenger starts by sampling $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N)$ and gives $(1^\lambda, N, \mathsf{pp})$ to the adversary. The challenger also initializes an empty table T to keep track of key-generation queries.

- **Query phase:** The adversary can now make adaptive queries to the following oracles:
  - **Key-generation query:** On input an index $i \in [N]$, the challenger first checks if $\mathsf{T}[i]$ is already defined. If so, it looks up $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{T}[i]$. Otherwise, it samples $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ and sets $\mathsf{T}[i] = (\mathsf{pk}_i, \mathsf{sk}_i)$. The challenger responds with $\mathsf{pk}_i$.
  - **Corruption query:** On input an index $i \in [N]$, the challenger looks up $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{T}[i]$ and responds with $\mathsf{sk}_i$. Note that without loss of generality, we will require that the adversary has issued a key-generation query on the index $i$ before it is allowed to issue a corruption query on index $i$. In this case, $\mathsf{T}[i]$ is guaranteed to be defined.
- **Challenge phase:** After the adversary is done making queries, it outputs a set of indices $S^* \subseteq [N]$. Without loss of generality, we require that the adversary has previously issued a key-generation query on every index $i \in S^*$. Let $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{T}[i]$ for each $i \in S^*$. The challenger then computes $\mathsf{ct}_b \leftarrow \mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S^*}, b)$ and sends $\mathsf{ct}_b$ to $\mathcal{A}$.
- **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say an adversary $\mathcal{A}$ is admissible if it does not make a corruption query on any index $i \in S^*$. We say $\Pi_{\mathsf{DBE}}$ is adaptively secure if for all efficient and admissible adversaries $\mathcal{A}$ and all polynomials $N = N(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 1] - \Pr[b' = 1 : b = 0]| = \mathsf{negl}(\lambda)$$

in the adaptive security game.

- $\varepsilon$-**Succinctness:** There exists a fixed polynomial $\mathsf{poly}(\cdot)$ and constant $\varepsilon \in [0,1)$ such that for all $\lambda, N \in \mathbb{N}$, all subsets $S \subseteq [N]$, all public parameters $\mathsf{pp}$ in the support of $\mathsf{Setup}(1^\lambda, N)$, all public keys $\mathsf{pk}_i$, all messages $\mu \in \{0,1\}$, and all ciphertexts $\mathsf{ct}$ in the support of $\mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu)$, it holds that $|\mathsf{ct}| \leq |S|^\varepsilon \cdot \mathsf{poly}(\lambda)$.

**Semi-static security.** Next, we recall the notion of semi-static security [GW09, KMW23]. In the semi-static security game, the adversary can make key-generation queries, but not corruption queries. We give the formal definition below:

**Definition 5.2** (Semi-Static Security)**.** Let $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a distributed broadcast encryption scheme. For a security parameter $\lambda$, a bound $N$, a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the semi-static security game between an adversary $\mathcal{A}$ and a challenger to be identical to the adaptive security game from Definition 5.1, except the adversary is not allowed to make any corruption queries during the query phase. Finally, we say $\Pi_{\mathsf{DBE}}$ is semi-statically secure if for all efficient adversaries $\mathcal{A}$ and all polynomials $N = N(\lambda)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 1] - \Pr[b' = 1 : b = 0]| = \mathsf{negl}(\lambda)$$

in the semi-static security game.

**Semi-static distributed broadcast encryption in the plain model.** As noted earlier, distributed broadcast encryption is a special case of distributed monotone-policy encryption for DNFs where each min-term contains a single variable. Thus Construction 4.1 and Corollary 4.4 immediately imply a distributed broadcast encryption scheme in the plain model with semi-static security. Note that since the hint-generation

step is not needed for distributed broadcast encryption, we can drop the NIZK proof of well-formedness as well as the noise smudging for the hint generation. Thus, we obtain the following corollary:

**Corollary 5.3** (Semi-Static Distributed Broadcast Encryption in the Plain Model)**.** *Let $\lambda$ be a security parameter. Then, under the decomposed LWE assumption with a polynomial modulus-to-noise ratio, there exists a semi-static distributed broadcast encryption scheme for sets of up to $|S| = 2^\lambda$ users. The size of the public parameters and public keys is $\mathsf{poly}(\lambda)$. The size of a ciphertext is $|S|^{1/2} \cdot \mathsf{poly}(\lambda)$. Moreover, the scheme has a transparent setup.*

*Proof.* This follows by specializing Construction 4.1 to the setting where the policy is a simple disjunction (i.e., set $k = 1$) and removing the hint-generation algorithm. Since we do not need to support hint-generation, we can additionally avoid the need for a smudging modulus. The analogous version of Theorem 4.3 can now be instantiated from decomposed LWE with a polynomial modulus-to-noise ratio. □

**Remark 5.4** (Flexible Broadcast Encryption)**.** Flexible broadcast encryption [FWW23] is a generalization of distributed broadcast encryption where users are no longer associated with a specific index. Specifically, the key-generation algorithm in a flexible broadcast encryption scheme just takes the public parameters of the scheme as input. In turn, the encryption algorithm takes a list of public keys for the recipients (without needing to associate public keys with indices). Since the key-generation algorithm in Construction 4.1 only depends on the public parameters (and does not need to be associated with an index), the broadcast encryption scheme obtained in Corollary 5.3 is in fact a flexible broadcast encryption scheme.

**From semi-static security to adaptive security.** We now describe our re-balanced variant of the Gentry-Waters transformation [GW09] (specifically, the Kolonelos-Malavolta-Wee variant for distributed broadcast encryption [KMW23]) that produces a distributed broadcast encryption with longer (but still sublinear-size) ciphertexts and security in the plain model.

**Construction 5.5** (A Re-Balanced Gentry-Waters Compiler)**.** Let $\Pi_{\mathsf{DBE,SS}} = (\mathsf{Setup}_{\mathsf{SS}}, \mathsf{KeyGen}_{\mathsf{SS}}, \mathsf{Encrypt}_{\mathsf{SS}}, \mathsf{Decrypt}_{\mathsf{SS}})$ be a distributed broadcast encryption scheme with semi-static security. Let $\alpha \in (0, 1)$ be a re-balancing constant. We construct a distributed broadcast encryption scheme $\Pi_{\mathsf{DBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda, N)$: On input the security parameter $\lambda$ and the number of users $N$, sample and output the public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}_{\mathsf{SS}}(1^\lambda, 2N)$.
- $\mathsf{KeyGen}(\mathsf{pp}, i)$: On input the public parameters pp and an index $i \in [N]$, sample $(\mathsf{pk}_{i,0}, \mathsf{sk}_{i,0}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i-1)$ and $(\mathsf{pk}_{i,1}, \mathsf{sk}_{i,1}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i)$. Sample a random bit $b \xleftarrow{R} \{0,1\}$. Output the public key $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ and the secret key $\mathsf{sk}_i = (b, \mathsf{sk}_{i,b})$.
- $\mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu)$: On input the public parameters pp, a collection of public keys $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ for $i \in S$, and a message $\mu \in \{0,1\}$, the encryption algorithm proceeds as follows:
  - Let $K = \lceil |S|^\alpha \rceil$ and $L = \lceil |S|/K \rceil$.
  - Let $i_1, \ldots, i_{|S|}$ be the elements of $S$ in lexicographic order. For each $k \in [K]$, let

$$S_k = \left\{ i_{(k-1)\cdot L + j} : j \in [L], (k-1)\cdot L + j \le |S| \right\} \tag{5.1}$$

    be the $k^{\text{th}}$ block of indices. In particular, $S_1, \ldots, S_K$ form a partition of $S$. Take the elements of $S_k$ to be $\left(i_1^{(k)}, \ldots, i_{|S_k|}^{(k)}\right)$.

- Sample $t \xleftarrow{R} \{0,1\}^L$. For each $k \in [K]$ and $\beta \in \{0,1\}$, define the set of public keys to be

$$T_{k,\beta} = \left\{ \left( 2i_j^{(k)} - 1 + (t_j \oplus \beta), \mathsf{pk}_{i_j^{(k)}, t_j \oplus \beta} \right) : j \in [|S_k|] \right\}. \tag{5.2}$$

- For each $k \in [K]$ and each $\beta \in \{0,1\}$, compute $\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, \mu)$.
- Output the ciphertext $\mathsf{ct} = (t, \{\mathsf{ct}_{k,0}, \mathsf{ct}_{k,1}\}_{k \in [K]})$.

- $\mathsf{Decrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mathsf{ct}, (i^*, \mathsf{sk}_{i^*}))$: On input the public parameters $\mathsf{pp}$, a collection of public keys $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ for $i \in S$, a ciphertext $\mathsf{ct}$, and a secret key $\mathsf{sk}_{i^*} = (b, \mathsf{sk}_{i^*,b})$ for an index $i^* \in S$, the decryption algorithm proceeds as follows:
  - Parse the ciphertext as $\mathsf{ct} = (t, \{\mathsf{ct}_{k,0}, \mathsf{ct}_{k,1}\}_{k \in [K]})$, where $t \in \{0,1\}^L$ and $K = \lceil |S|^\alpha \rceil$, $L = \lceil |S|/K \rceil$ as in Encrypt.
  - Compute $S_1, \ldots, S_K$ as in Eq. (5.1) and as in Encrypt, let $S_k = \left(i_1^{(k)}, \ldots, i_{|S_k|}^{(k)}\right)$. Define $T_{k,\beta}$ as in Eq. (5.2).
  - Let $k^* \in [K]$ and $j^* \in [|S_{k^*}|]$ be the unique pair with $i^* = i_{j^*}^{(k^*)}$.
  - Let $c = b \oplus t_{j^*}$. Output $\mathsf{Decrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k^*,c}, \mathsf{ct}_{k^*,c}, (2i^* - 1 + b, \mathsf{sk}_{i^*,b}))$.

**Theorem 5.6** (Correctness). *If $\Pi_{\mathsf{DBE,SS}}$ is correct, then Construction 5.5 is also correct.*

*Proof.* Fix any $\lambda, N \in \mathbb{N}$, any public parameter $\mathsf{pp}$ in the support of $\mathsf{Setup}(1^\lambda, N)$, any set $S \subseteq [N]$, any index $i^* \in S$, any collection of public keys $\mathsf{pk}_i$ for $i \in S \setminus \{i^*\}$, and any message $\mu \in \{0,1\}$. Let $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i^*)$ and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu)$. Our goal is to show that

$$\mathsf{Decrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mathsf{ct}, (i^*, \mathsf{sk}_{i^*})) = \mu.$$

By construction, the public parameters $\mathsf{pp}$ are in the support of $\mathsf{Setup}_{\mathsf{SS}}(1^\lambda, 2N)$. By definition of KeyGen, the public key for $i^*$ has the form $\mathsf{pk}_{i^*} = (\mathsf{pk}_{i^*,0}, \mathsf{pk}_{i^*,1})$ where $(\mathsf{pk}_{i^*,\beta}, \mathsf{sk}_{i^*,\beta}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i^* - 1 + \beta)$ for $\beta \in \{0,1\}$, and the secret key has the form $\mathsf{sk}_{i^*} = (b^*, \mathsf{sk}_{i^*,b^*})$ for some $b^* \in \{0,1\}$. By definition of Encrypt, the ciphertext has the form $\mathsf{ct} = (t, \{\mathsf{ct}_{k,0}, \mathsf{ct}_{k,1}\}_{k \in [K]})$ where $t \in \{0,1\}^L$ and, for each $k \in [K]$ and $\beta \in \{0,1\}$, $\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, \mu)$ with $T_{k,\beta}$ defined as in Encrypt.

Since $S_1, \ldots, S_K$ form a partition of $S$ and $i^* \in S$, there exists a unique pair $(k^*, j^*) \in [K] \times [|S_{k^*}|]$ such that $i_{j^*}^{(k^*)} = i^*$. The decryption algorithm computes $c = b^* \oplus t_{j^*}$ and outputs

$$\mathsf{Decrypt}_{\mathsf{SS}}\left(\mathsf{pp}, T_{k^*,c}, \mathsf{ct}_{k^*,c}, (2i^* - 1 + b^*, \mathsf{sk}_{i^*,b^*})\right).$$

By construction, the encryption algorithm sets

$$T_{k^*,c} = \left\{ \left( 2i_j^{(k^*)} - 1 + (t_j \oplus c), \mathsf{pk}_{i_j^{(k^*)}, t_j \oplus c} \right) : j \in [|S_{k^*}|] \right\}.$$

Consider the entry corresponding to $j = j^*$. By construction, the decryption algorithm chooses $(k^*, j^*)$ so that $i_{j^*}^{(k^*)} = i^*$. Since $c = b^* \oplus t_{j^*}$, this means

$$t_{j^*} \oplus c = t_{j^*} \oplus (b^* \oplus t_{j^*}) = b^*.$$

Therefore, the $(j^*)^{\text{th}}$ entry of $T_{k^*,c}$ is $(2i^* - 1 + b^*, \mathsf{pk}_{i^*,b^*})$. Since $(\mathsf{pk}_{i^*,b^*}, \mathsf{sk}_{i^*,b^*}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i^* - 1 + b^*)$, $\mathsf{ct}_{k^*,c} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k^*,c}, \mu)$, and $(2i^* - 1 + b^*, \mathsf{pk}_{i^*,b^*}) \in T_{k^*,c}$, we appeal to correctness of $\Pi_{\mathsf{DBE,SS}}$ to conclude that $\mathsf{Decrypt}_{\mathsf{SS}}\left(\mathsf{pp}, T_{k^*,c}, \mathsf{ct}_{k^*,c}, (2i^* - 1 + b^*, \mathsf{sk}_{i^*,b^*})\right)$ outputs $\mu$, as required. □

**Theorem 5.7** (Adaptive Security). *Suppose* $\Pi_{\mathsf{DBE,SS}}$ *satisfies semi-static security. Then, Construction 5.5 is adaptively secure.*

*Proof.* Take any polynomial $N = N(\lambda)$. Let $\mathcal{A}$ be an efficient and admissible adversary for the adaptive security game for Construction 5.5 with $N$ users. We start by defining a sequence of hybrid experiments, each parameterized by a bit $b \in \{0,1\}$.

- $\mathsf{Hyb}_0^{(b)}$: This is the adaptive security game with challenge bit $b$:
  - **Setup phase:** The challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}_{\mathsf{SS}}(1^\lambda, 2N)$ and gives $(1^\lambda, N, \mathsf{pp})$ to $\mathcal{A}$. It also initializes an empty table $\mathsf{T}$.
  - **Query phase:** The adversary makes adaptive queries to the following oracles:
    - **Key-generation query:** On input an index $i \in [N]$, if $\mathsf{T}[i]$ is defined, the challenger looks up $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{T}[i]$. Otherwise, it samples $(\mathsf{pk}_{i,0}, \mathsf{sk}_{i,0}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i-1)$, $(\mathsf{pk}_{i,1}, \mathsf{sk}_{i,1}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i)$, and $b_i \xleftarrow{\mathrm{R}} \{0,1\}$, sets $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ and $\mathsf{sk}_i = (b_i, \mathsf{sk}_{i,b_i})$, and stores $\mathsf{T}[i] = (\mathsf{pk}_i, \mathsf{sk}_i)$. The challenger replies with $\mathsf{pk}_i$.
    - **Corruption query:** On input an index $i \in [N]$, if $\mathsf{T}[i] = (\mathsf{pk}_i, \mathsf{sk}_i)$ is defined, the challenger replies with $\mathsf{sk}_i$. Otherwise, it replies with $\perp$.
  - **Challenge phase:** The adversary outputs a challenge set $S^* \subseteq [N]$. Without loss of generality, we assume that the adversary $\mathcal{A}$ has made a key-generation query on every index $i \in S^*$. To construct the challenge ciphertext, the challenger sets $K = \lceil |S^*|^\alpha \rceil$ and $L = \lceil |S^*|/K \rceil$, computes the partition $S_1^*, \ldots, S_K^*$ of $S^*$ as in Eq. (5.1), samples $t \xleftarrow{\mathrm{R}} \{0,1\}^L$, and constructs the sets $T_{k,\beta}$ for $(k,\beta) \in [K] \times \{0,1\}$ as in Eq. (5.2). Finally, for each $(k,\beta) \in [K] \times \{0,1\}$, the challenger computes
$$\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, b),$$
    and sends $\mathsf{ct} = (t, \{\mathsf{ct}_{k,\beta}\}_{(k,\beta)\in[K]\times\{0,1\}})$ to $\mathcal{A}$.
  - **Output phase:** At the end of the experiment, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.
- $\mathsf{Hyb}_{k^*,0}^{(b)}$ for $k^* \in [K]$: Same as $\mathsf{Hyb}_0^{(b)}$ except the challenger now sets
$$\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, 0)$$
  whenever $k < k^*$ and when $(k,\beta) = (k^*, 0)$.
- $\mathsf{Hyb}_{k^*,1}^{(b)}$ for $k^* \in [K]$: Same as $\mathsf{Hyb}_0^{(b)}$, except the challenger now sets
$$\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, 0)$$
  whenever $k \le k^*$. The remaining ciphertexts are constructed as in $\mathsf{Hyb}_0^{(b)}$.

We write $\mathsf{Hyb}_{k^*,\beta}^{(b)}(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of $\mathsf{Hyb}_{k^*,\beta}^{(b)}$ with adversary $\mathcal{A}$. We define $\mathsf{Hyb}_{0,1}^{(b)} \equiv \mathsf{Hyb}_0^{(b)}$. We now analyze each adjacent pair of hybrid experiments.

**Lemma 5.8.** *Suppose* $\Pi_{\mathsf{DBE,SS}}$ *satisfies semi-static security. Then, for all* $k^* \in [K]$ *and all* $b \in \{0,1\}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*
$$\left|\Pr[\mathsf{Hyb}_{k^*-1,1}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k^*,0}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

*Proof.* The only difference between $\mathsf{Hyb}^{(b)}_{k^*-1,1}$ and $\mathsf{Hyb}^{(b)}_{k^*,0}$ is the distribution of $\mathsf{ct}_{k^*,0}$. In $\mathsf{Hyb}^{(b)}_{k^*-1,1}$, the ciphertext $\mathsf{ct}_{k^*,0}$ is an encryption of $b$, whereas in $\mathsf{Hyb}^{(b)}_{k^*,0}$, it is an encryption of 0. When $b = 0$, the two experiments are identical by construction. It suffices to consider the case where $b = 1$. Suppose

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{k^*-1,1}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{k^*,0}(\mathcal{A}) = 1]\right| \geq \varepsilon$$

for some non-negligible $\varepsilon$. We construct an efficient semi-static adversary $\mathcal{B}$ for $\Pi_{\mathsf{DBE,SS}}$ (with $2N$ users) as follows:

- **Setup phase:** On input $(1^\lambda, 2N, \mathsf{pp})$, algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on input $(1^\lambda, N, \mathsf{pp})$. Algorithm $\mathcal{B}$ also maintains an (empty) table $\mathsf{T}$ to keep track of key-generation queries.
- **Key-generation queries:** On a key-generation query for $i \in [N]$, algorithm $\mathcal{B}$ first checks if $\mathsf{T}[i]$ is already defined. If so, it looks up $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{T}[i]$ and responds with $\mathsf{pk}_i$. Otherwise, algorithm $\mathcal{B}$ samples $b_i \xleftarrow{R} \{0,1\}$, issues a key-generation query on $2i-1+(1-b_i)$ to its challenger to receive the public key $\mathsf{pk}_{i,1-b_i}$. In addition, it samples $(\mathsf{pk}_{i,b_i}, \mathsf{sk}_{i,b_i}) \leftarrow \mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i-1+b_i)$ on its own. Algorithm $\mathcal{B}$ then sets $\mathsf{pk}_i = (\mathsf{pk}_{i,0}, \mathsf{pk}_{i,1})$ and $\mathsf{sk}_i = (b_i, \mathsf{sk}_{i,b_i})$, sets $\mathsf{T}[i] = (\mathsf{pk}_i, \mathsf{sk}_i)$, and replies with $\mathsf{pk}_i$.
- **Corruption queries:** On a corruption query for $i \in [N]$, algorithm $\mathcal{B}$ looks up $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{T}[i]$ and responds with $\mathsf{sk}_i = (b_i, \mathsf{sk}_{i,b_i})$.
- **Challenge phase:** When $\mathcal{A}$ outputs a challenge set $S^* \subseteq [N]$, algorithm $\mathcal{B}$ proceeds as follows.
  - First, algorithm $\mathcal{B}$ computes $K = \lceil |S^*|^\alpha \rceil$, $L = \lceil |S^*|/K \rceil$, and defines the sets $S^*_1, \ldots, S^*_K$ according to Eq. (5.1). Take the elements of $S^*_k$ to be $\left(i^{(k)}_1, \ldots, i^{(k)}_{|S^*_k|}\right)$.
  - Algorithm $\mathcal{B}$ constructs the selector $t \in \{0,1\}^L$ as follows. For each $j \in [|S^*_{k^*}|]$, algorithm $\mathcal{B}$ sets $t_j = 1 - b_{i^{(k^*)}_j}$. For the remaining indices $j \in [L] \setminus [|S^*_{k^*}|]$, algorithm $\mathcal{B}$ sets $t_j \xleftarrow{R} \{0,1\}$.
  - Algorithm $\mathcal{B}$ submits the set of indices

    $$\widetilde{S}^* = \left\{2i^{(k^*)}_j - 1 + t_j : j \in [|S^*_{k^*}|]\right\} \subseteq [2N]$$

    as its challenge set to the semi-static security challenger, who replies with a ciphertext $\mathsf{ct}_{k^*,0}$.
  - For each $(k, \beta) \in [K] \times \{0,1\}$ with $(k, \beta) \neq (k^*, 0)$, algorithm $\mathcal{B}$ constructs the public-key set $T_{k,\beta}$ as in Eq. (5.2) and computes $\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, \mu_{k,\beta})$, where $\mu_{k,\beta} = 0$ if $k < k^*$ and $\mu_{k,\beta} = b$ otherwise. Algorithm $\mathcal{B}$ sends $\mathsf{ct} = (t, \{\mathsf{ct}_{k,0}, \mathsf{ct}_{k,1}\}_{k\in[K]})$ to $\mathcal{A}$.
- **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

We now argue that algorithm $\mathcal{B}$ correctly simulates either an execution of $\mathsf{Hyb}^{(b)}_{k^*-1,1}$ or $\mathsf{Hyb}^{(b)}_{k^*,0}$.

- First, the challenger samples the public parameter $\mathsf{pp} \leftarrow \mathsf{Setup}_{\mathsf{SS}}(1^\lambda, 2N)$, which coincides with the distribution in both experiments. Next, for each $i \in [N]$, the public keys $\mathsf{pk}_{i,0}$ and $\mathsf{pk}_{i,1}$ are sampled by running $\mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i-1)$ and $\mathsf{KeyGen}_{\mathsf{SS}}(\mathsf{pp}, 2i)$, respectively. This matches the behavior of the honest key-generation algorithm. Finally, algorithm $\mathcal{B}$ samples the bit $b_i \xleftarrow{R} \{0,1\}$ associated with each key in the same manner as the honest key-generation algorithm. Thus, algorithm $\mathcal{B}$ perfectly simulates the query phase for $\mathcal{A}$.

- Consider now the distribution of the challenge ciphertext. Consider first the distribution of the selector bit string $t \in \{0,1\}^L$. In $\mathsf{Hyb}^{(b)}_{k^*-1,1}$ or $\mathsf{Hyb}^{(b)}_{k^*,0}$, the challenger samples $t \xleftarrow{R} \{0,1\}^L$. We claim that this matches the distribution in the reduction. Take any index $j \in [L]$.
  - Suppose $j \in [|S^*_{k^*}|]$. In this case, algorithm $\mathcal{B}$ sets $t_j = 1 - b_{i_j^{(k^*)}}$, where $b_{i_j^{(k^*)}} \xleftarrow{R} \{0,1\}$ is the random bit algorithm $\mathcal{B}$ sampled when responding to $\mathcal{A}$'s key-generation query on $i_j^{(k^*)}$. Since $i_j^{(k^*)} \in S^*_{k^*} \subseteq S^*$, and algorithm $\mathcal{A}$ is admissible, algorithm $\mathcal{A}$ must not have issued a corruption query on $i_j^{(k^*)}$. This means the bit $b_{i_j^{(k^*)}}$ is information-theoretically hidden from the view of $\mathcal{A}$ (and independent of all other components in the view of $\mathcal{A}$). Correspondingly, the distribution of $t_j = 1 - b_{i_j^{(k^*)}}$ is uniform over $\{0,1\}$ and independent of $\mathcal{A}$'s view, which matches the distribution in $\mathsf{Hyb}^{(b)}_{k^*-1,1}$ or $\mathsf{Hyb}^{(b)}_{k^*,0}$.
  - Suppose $j \in [L] \setminus [|S^*_{k^*}|]$. In this case, algorithm $\mathcal{B}$ samples $t_j \xleftarrow{R} \{0,1\}$ independent of all other components in the view of $\mathcal{A}$, which matches the distribution in $\mathsf{Hyb}^{(b)}_{k^*-1,1}$ or $\mathsf{Hyb}^{(b)}_{k^*,0}$.

  Next, consider the ciphertexts $\mathsf{ct}_{k,\beta}$ where $(k,\beta) \neq (k^*,0)$. By construction, algorithm $\mathcal{B}$ constructs these using the procedure as in $\mathsf{Hyb}^{(b)}_{k^*-1,1}$ and $\mathsf{Hyb}^{(b)}_{k^*,0}$. Finally, consider the distribution of $\mathsf{ct}_{k^*,0}$. Since $t_j = 1 - b_{i_j^{(k^*)}}$ for $j \in [|S^*_{k^*}|]$ and $i_1^{(k^*)}, \ldots, i_{|S^*_{k^*}|}^{(k^*)}$ are the elements of $S^*_{k^*}$, the challenge set is

$$\widetilde{S}^* = \left\{2i_j^{(k^*)} - 1 + t_j : j \in [|S^*_{k^*}|]\right\} = \left\{2i - 1 + (1 - b_i) : i \in S^*_{k^*}\right\}.$$

  By construction of algorithm $\mathcal{B}$'s key-generation queries, this means the semi-static security challenger constructs the public-key set to be

$$\widetilde{T}_{k^*,0} = \left\{\left(2i - 1 + (1 - b_i), \mathsf{pk}_{i,1-b_i}\right) : i \in S^*_{k^*}\right\} = \left\{\left(2i_j^{(k^*)} - 1 + t_j, \mathsf{pk}_{i_j^{(k^*)},t_j}\right) : j \in [|S^*_{k^*}|]\right\},$$

  which precisely coincides with the definition of $T_{k^*,0}$ in Eq. (5.2). We now consider two cases:
  - If the challenger computes $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, \widetilde{T}_{k^*,0}, 1)$, then $\mathsf{ct}_{k^*,0}$ is distributed according to $\mathsf{Hyb}^{(b)}_{k^*-1,1}$. Recall that we are considering the case where $b = 1$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{k^*-1,1}$.
  - If the challenger computes $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, \widetilde{T}_{k^*,0}, 0)$, then $\mathsf{ct}_{k^*,0}$ is distributed according to $\mathsf{Hyb}^{(b)}_{k^*,0}$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{k^*,0}$.

We conclude that algorithm $\mathcal{B}$ breaks semi-static security of $\Pi_{\mathsf{DBE,SS}}$ with the same advantage $\varepsilon$. □

**Lemma 5.9.** *Suppose $\Pi_{\mathsf{DBE,SS}}$ satisfies semi-static security. Then, for all $k^* \in [K]$ and all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{k^*,0}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{k^*,1}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

*Proof.* Follows by the same argument as the proof of Lemma 5.8, except algorithm $\mathcal{B}$ targets $\mathsf{ct}_{k^*,1}$. Concretely, $\mathcal{B}$ now sets $t_j = b_{i_j^{(k^*)}}$ (rather than $1 - b_{i_j^{(k^*)}}$) for $j \in [|S^*_{k^*}|]$. This way, the indices in $T_{k^*,1}$ coincide with the indices on which $\mathcal{B}$ issued key-generation queries. Since $\mathsf{ct}_{k^*,1}$ is now the target ciphertext, algorithm $\mathcal{B}$ constructs the remaining ciphertext $\mathsf{ct}_{k^*,0}$ itself as an encryption of 0. Namely, algorithm $\mathcal{B}$ sets $\mu_{k,\beta} = 0$ for all ciphertexts with $k \leq k^*$ and $\mu_{k,\beta} = b$ for all $k > k^*$. The remaining analysis is identical. □

To complete the proof, observe that the experiment $\mathsf{Hyb}_{K,1}^{(b)}$ is identical with $b = 0$ and with $b = 1$ (i.e., the challenge ciphertext is independent of the bit $b$, since every $\mathsf{ct}_{k,\beta}$ in the two experiments is an encryption of 0). Since $N = \mathrm{poly}(\lambda)$ and $K < N$, the claim now follows from Lemmas 5.8 and 5.9 and a hybrid argument. □

**Theorem 5.10** (Succinctness)**.** *Suppose $\Pi_{\mathsf{DBE,SS}}$ satisfies $\varepsilon$-succinctness for some constant $\varepsilon \in [0, 1)$. Let $\alpha = (1 - \varepsilon)/(2 - \varepsilon)$. Then, Construction 5.5 satisfies $\delta$-succinctness where $\delta = 1/(2 - \varepsilon)$.*

*Proof.* Fix any $\lambda, N \in \mathbb{N}$, any $S \subseteq [N]$, any pp in the support of $\mathsf{Setup}(1^\lambda, N)$, any public keys $\mathsf{pk}_i$ for $i \in S$, any message $\mu \in \{0, 1\}$, and any ct in the support of $\mathsf{Encrypt}(\mathsf{pp}, \{(i, \mathsf{pk}_i)\}_{i \in S}, \mu)$. By definition of Encrypt, the ciphertext has the form

$$\mathsf{ct} = (t, \{\mathsf{ct}_{k,0}, \mathsf{ct}_{k,1}\}_{k \in [K]}),$$

where $K = \lceil |S|^\alpha \rceil = O(|S|^\alpha)$, $L = \lceil |S|/K \rceil = O(|S|^{1-\alpha})$, $t \in \{0, 1\}^L$, and $\mathsf{ct}_{k,\beta} \leftarrow \mathsf{Encrypt}_{\mathsf{SS}}(\mathsf{pp}, T_{k,\beta}, \mu)$ with $|T_{k,\beta}| = |S_k| \le L$ for each $k \in [K]$ and $\beta \in \{0, 1\}$. The total ciphertext size is therefore

$$|\mathsf{ct}| = L + \sum_{k \in [K]} \left( |\mathsf{ct}_{k,0}| + |\mathsf{ct}_{k,1}| \right). \tag{5.3}$$

By $\varepsilon$-succinctness of $\Pi_{\mathsf{DBE,SS}}$, there exists a fixed polynomial $\mathrm{poly}_{\mathsf{SS}}(\cdot)$ such that

$$|\mathsf{ct}_{k,\beta}| \le |S|^{\varepsilon(1-\alpha)} \cdot \mathrm{poly}_{\mathsf{SS}}(\lambda)$$

for all $k \in [K]$ and $\beta \in \{0, 1\}$. Substituting back into Eq. (5.3), we have

$$|\mathsf{ct}| = L + \sum_{k \in [K]} \left( |\mathsf{ct}_{k,0}| + |\mathsf{ct}_{k,1}| \right) \le O(|S|^{1-\alpha}) + O(|S|^{\alpha+\varepsilon(1-\alpha)}) \cdot \mathrm{poly}_{\mathsf{SS}}(\lambda).$$

Let $\alpha = (1 - \varepsilon)/(2 - \varepsilon)$. In this case,

$$1 - \alpha = \frac{1}{2 - \varepsilon} = \delta \quad \text{and} \quad \alpha + \varepsilon(1 - \alpha) = \frac{(1 - \varepsilon) + \varepsilon}{2 - \varepsilon} = \frac{1}{2 - \varepsilon} = \delta.$$

We conclude

$$|\mathsf{ct}| \le O(|S|^{1-\alpha}) + O(|S|^{\alpha+\varepsilon(1-\alpha)}) \cdot \mathrm{poly}_{\mathsf{SS}}(\lambda) = |S|^\delta \cdot \mathrm{poly}(\lambda),$$

as required. □

By applying the above compiler to Corollary 5.3, we obtain the following instantiation:

**Corollary 5.11** (Adaptively-Secure Distributed Broadcast Encryption in the Plain Model)**.** *Let $\lambda$ be a security parameter. Then, under the decomposed LWE assumption with a polynomial modulus-to-noise ratio, there exists an adaptively-secure distributed broadcast encryption scheme for sets of up to $|S| = 2^\lambda$ users. The size of the public parameters and public keys is $\mathrm{poly}(\lambda)$. The size of a ciphertext is $|S|^{2/3} \cdot \mathrm{poly}(\lambda)$. Moreover, the scheme has a transparent setup.*

**Remark 5.12** (Reducing Ciphertext Size)**.** We believe that it is possible to obtain a scheme with ciphertext size $|S|^{1/2} \cdot \mathrm{poly}(\lambda)$ by in-lining the compiler from Construction 5.5 directly into Construction 4.1. Both constructions rely on re-balancing and both exploit the fact that the different copies of the base scheme can *reuse* a common set of base parameters. However, for the sake of modularity and ease of exposition, we chose to describe the two transformations separately.

**Remark 5.13** (Optimal Flexible Broadcast Encryption Scheme in the Random Oracle Model)**.** As shown in Corollary 5.3 and Remark 5.4, we can obtain a flexible broadcast encryption scheme with semi-static security in the plain model by specializing Construction 4.1 to the special case of a simple disjunction policy. We can also obtain a semi-statically-secure flexible broadcast encryption scheme with optimal parameters in the random oracle model by specializing our distributed monotone-policy encryption scheme from Construction 3.1 to the case of disjunctions. As in the case of Corollary 5.3, for the special case of broadcast encryption, we do not need to support hint-generation queries, and as such, Theorem 3.3 no longer requires using a super-polynomial modulus $q$. Specifically, Theorem 3.3 assumed a super-polynomial modulus for noise smudging to support hint-generation queries in the security analysis. In conjunction with the Gentry-Waters compiler [GW09, KMW23], this yields an adaptively-secure flexible broadcast encryption scheme with optimal parameters from decomposed LWE with a *polynomial* modulus-to-noise ratio in the random oracle model. This improves upon the prior work of [GY26a] that achieved the same result, but relied on decomposed LWE with a super-polynomial modulus-to-noise ratio in the random oracle model.

## Acknowledgments

We acknowledge the use of Claude for copy-editing to improve the exposition of this work. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, an Amazon Research Award, and a gift from the Stellar Development Foundation.

## References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

[ABI+23] Benny Applebaum, Amos Beimel, Yuval Ishai, Eyal Kushilevitz, Tianren Liu, and Vinod Vaikuntanathan. Succinct computational secret sharing. In *STOC*, 2023.

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. In *ASIACRYPT*, 2024.

[AGY26] Abtin Afshar, Rishab Goyal, and Saikumar Yadugiri. Distributed monotone-policy encryption with silent setup from lattices. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/372.pdf.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, 1996.

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Key-homomorphic computations for RAM: Fully succinct randomised encodings and more. In *CRYPTO*, 2025.

[AMR26] Damiano Abram, Giulio Malavolta, and Lawrence Roy. How to authenticate a non-deterministic computation: Shift-hiding functions, compressed LWE sampling, broadcast encryption, and obfuscation. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/741.pdf.

[ASY22] Damiano Abram, Peter Scholl, and Sophia Yakoubov. Distributed (correlation) samplers: How to remove a trusted dealer in one round. In *EUROCRYPT*, 2022.

[BCD+25] Pedro Branco, Arka Rai Choudhuri, Nico Döttling, Abhishek Jain, Giulio Malavolta, and Akshayaram Srinivasan. Black-box non-interactive zero knowledge from vector trapdoor hash. In *EUROCRYPT*, 2025.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO*, 2019.

[BDE+18] Jonathan Bootle, Claire Delaplace, Thomas Espitau, Pierre-Alain Fouque, and Mehdi Tibouchi. LWE without modular reduction and improved side-channel attacks against BLISS. In *ASIACRYPT*, 2018.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*, 2018.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, 1988.

[BL88] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *CRYPTO*, 1988.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, 2007.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[CW26] Jeffrey Champion and David J. Wu. Distributed monotone-policy encryption for DNFs from lattices. In *EUROCRYPT*, 2026.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. In *ASIACRYPT*, 2025.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1), 2008.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, 1990.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam D. Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *J. Cryptol.*, 28(4), 2015.

[GHK+25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. In *CRYPTO*, 2025.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[GW09] Craig Gentry and Brent Waters. Adaptive security in broadcast encryption systems (with short ciphertexts). In *EUROCRYPT*, 2009.

[GY26a] Rishab Goyal and Saikumar Yadugiri. Adaptively-secure flexible and identity-based broadcast encryption from decomposed LWE. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/862.pdf.

[GY26b] Rishab Goyal and Saikumar Yadugiri. Equivocal broadcast encryption: Adaptively-secure optimal distributed broadcast encryption from lattices. In *CRYPTO*, 2026.

[HIJ+16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In *ITCS*, 2016.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28(4), 1999.

[HWW25] Yao-Ching Hsieh, Brent Waters, and David J. Wu. A generic approach to adaptively-secure broadcast encryption in the plain model. In *EUROCRYPT*, 2025.

[JJMP25] Abhishek Jain, Zhengzhong Jin, Surya Mathialagan, and Omer Paneth. On succinct obfuscation via propositional proofs. In *FOCS*, 2025.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[LNW25] George Lu, Shafik Nassar, and Brent Waters. Succinct computational secret sharing for monotone circuits. In *ASIACRYPT*, 2025.

[LW22] George Lu and Brent Waters. How to sample a discrete gaussian (and more) from a random oracle. In *TCC*, 2022.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. *IACR Cryptol. ePrint Arch.*, 2010, 2010.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In *CRYPTO*, 2019.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[RSY21] Leonid Reyzin, Adam Smith, and Sophia Yakoubov. Turning HATE into LOVE: Compact homomorphic ad hoc threshold encryption for scalable MPC. In *CSCML*, 2021.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.

[Wee24] Hoeteck Wee. Circuit ABE with $\mathsf{poly}(\mathsf{depth}, \lambda)$-sized ciphertexts and keys from lattices. In *CRYPTO*, 2024.

[Wee25] Hoeteck Wee. Almost optimal KP and CP-ABE for circuits from succinct LWE. In *EUROCRYPT*, 2025.

[WQZDF10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[WW26] Brent Waters and David J. Wu. Silent threshold cryptography from pairings: Expressive policies in the plain model. In *EUROCRYPT*, 2026.

[WWW25] Brent Waters, Hoeteck Wee, and David J. Wu. New techniques for preimage sampling: Improved NIZKs and more from LWE. In *EUROCRYPT*, 2025.

# A Analysis of Construction 4.1

In this section, we give the proofs of Theorems 4.2 and 4.3.

## A.1 Proof of Theorem 4.2 (Correctness)

Take any $\lambda \in \mathbb{N}$, any $k, L \leq 2^\lambda$, any $P = (S_1, \ldots, S_L) \in \mathcal{P}_{\mathsf{DNF}}$ on $N$-bit inputs, any set $T \subseteq [N]$ where $P(T) = 1$, any message $\mu \in \{0,1\}$, any $\mathsf{pp}$ in the support of $\mathsf{Setup}(1^\lambda)$, any $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for $i \in T$, and any choice of $\mathsf{pk}_i$ for $i \notin T$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \mu)$ and for each $i \in T$, let $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$. Now, consider the output of

$$\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}).$$

We parse the policy into chunks $P_1, \ldots, P_\ell$, where $P_w = (S_{w,1}, \ldots, S_{w,\ell})$ for $w \in [\ell]$ and analyze each step individually:

- Let $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. By definition of Setup, we have $\|\mathbf{R}\| \le \sqrt{m}\sigma_{\mathsf{pp}}$.
- For $i \in [N]$, we have $\mathsf{pk}_i = \mathbf{T}_i \in \mathbb{Z}_q^{n \times m}$. Additionally, for $i \in T$ we have that $\mathbf{T}_i = \mathbf{B}\mathbf{Y}_i$ where $\mathbf{Y}_i \in \{0,1\}^{4m^5 \times m}$.
- For each $i \in [\ell]$, let $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$.
- By definition, Encrypt first samples $\mathbf{u}_{i,j} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}}$ $\forall\, i \in [\ell], j \in [k]$, $\tau \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$, $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$, where $\|\mathbf{u}_{i,j}\| \le \sqrt{m}\sigma_{\mathsf{pp}}$. For $w \in [\ell]$, it samples $\mathbf{s}_w \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n, \mathbf{K}_{\mathbf{A}}^{(w)} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}, \mathbf{k}_{\mathbf{p}}^{(w)} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}, \mathbf{e}_w \leftarrow D^{4m^5}_{\mathbb{Z},\sigma_{\mathsf{LWE}}}$, where $\|\mathbf{e}_w\| \le \sqrt{m}\sigma_{\mathsf{LWE}}$ and $\|\mathbf{K}_{\mathbf{A}}^{(w)}\|, \|\mathbf{k}_{\mathbf{p}}^{(w)}\| \le 1$. Then, it sets $\mathbf{T}_{w,i,j}$ to be the $j^{\text{th}}$ public key indicated by $S_{w,i}$ and computes $\mathbf{d}_{w,i,j} = \mathbf{T}_{w,i,j} \cdot \mathbf{u}_{i,j}$ for $i \in [\ell], j \in [k]$. Finally, for $w \in [\ell]$ it defines $\mathbf{M}_w$ as in Eq. (4.1) and computes $\mathbf{C}_w = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}_w) \in \mathbb{Z}_q^{n \times m}$. The ciphertext has the form $(\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}}$ is parsed as in Eq. (4.2) where
$$(\mathbf{c}_{w,1}^\intercal, \mathbf{c}_{w,2}^\intercal, c_{w,3}) = (\mathbf{s}_w^\intercal \mathbf{B} + \mathbf{e}_w^\intercal,\ \mathbf{s}_w^\intercal(\mathbf{A} + \mathbf{C}_w) + \mathbf{e}_w^\intercal \mathbf{K}_{\mathbf{A}}^{(w)},\ \mathbf{s}_w^\intercal \mathbf{p} + \mathbf{e}_w^\intercal \mathbf{k}_{\mathbf{p}}^{(w)} + \mu \cdot \lfloor q/2 \rfloor).$$
By completeness of $\Pi_{\mathsf{NIZK}}$, we have
$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \pi_{\mathsf{ct}}) = 1.$$
- By definition of GetHint, for each $i \in T$, the hint $\mathsf{ht}_i = \{\mathbf{h}_{w,i}^\intercal\}_{w \in [\ell]}$ satisfies
$$\mathbf{h}_{w,i}^\intercal = \mathbf{c}_{w,1}^\intercal \mathbf{Y}_i + \mathbf{e}_{w,i}^\intercal = \mathbf{s}_w^\intercal \mathbf{B}\mathbf{Y}_i + \mathbf{e}_w^\intercal \mathbf{Y}_i + \mathbf{e}_{w,i}^\intercal = \mathbf{s}_w^\intercal \mathbf{T}_i + \mathbf{e}_w^\intercal \mathbf{Y}_i + \mathbf{e}_{w,i}^\intercal,$$
where $\mathbf{e}_{w,i} \in \mathbb{Z}^m$ is the per-hint smudging noise satisfying $\|\mathbf{e}_{w,i}\| \le \sqrt{m} \cdot \sigma_{\mathsf{ht}}$.
- Since $P(T) = 1$, there exists a smallest pair $(w^*, \mathsf{ind}) \in [\ell] \times [\ell]$ such that $S_{w^*,\mathsf{ind}} \subseteq T$. The decryption algorithm computes $\mathbf{z}_{\mathsf{ind}} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}_{w^*}, \mathsf{ind})$. By correctness of the matrix commitment (Theorem 2.10), we have $\mathbf{C}_{w^*}\mathbf{v}_{\mathsf{ind}} = \mathbf{m}_{w^*,\mathsf{ind}} - \mathbf{B}\mathbf{z}_{\mathsf{ind}}$, and therefore
$$\mathbf{c}_{w^*,1}^\intercal \mathbf{z}_{\mathsf{ind}} = (\mathbf{s}_{w^*}^\intercal \mathbf{B} + \mathbf{e}_{w^*}^\intercal)\mathbf{z}_{\mathsf{ind}} = \mathbf{s}_{w^*}^\intercal(\mathbf{m}_{w^*,\mathsf{ind}} - \mathbf{C}_{w^*}\mathbf{v}_{\mathsf{ind}}) + \mathbf{e}_{w^*}^\intercal \mathbf{z}_{\mathsf{ind}}.$$
In addition, $\mathbf{m}_{w^*,\mathsf{ind}} = \sum_{i \in [k]} \mathbf{d}_{w^*,\mathsf{ind},i} + \mathbf{p} - \mathbf{A}\mathbf{v}_{\mathsf{ind}}$ by Eq. (4.1). By definition, $\mathsf{idx}_i \in [N]$ is the $i^{\text{th}}$ index in $S_{w^*,\mathsf{ind}}$, so $\mathbf{h}_{w^*,\mathsf{idx}_i}^\intercal = \mathbf{s}_{w^*}^\intercal \mathbf{T}_{w^*,\mathsf{ind},i} + \mathbf{e}_{w^*}^\intercal \mathbf{Y}_{\mathsf{idx}_i} + \mathbf{e}_{\mathsf{idx}_i}^\intercal$. Consider the expression computed by the decryption algorithm. We have
$$\begin{aligned}
\mu' &= c_{w^*,3} - \mathbf{c}_{w^*,1}^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{c}_{w^*,2}^\intercal \mathbf{v}_{\mathsf{ind}} + \sum_{i \in [k]} \mathbf{h}_{w^*,\mathsf{idx}_i}^\intercal \mathbf{u}_{\mathsf{ind},i} \\
&= c_{w^*,3} - \mathbf{s}_{w^*}^\intercal(\mathbf{m}_{w^*,\mathsf{ind}} - \mathbf{C}_{w^*}\mathbf{v}_{\mathsf{ind}}) - \mathbf{e}_{w^*}^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{s}_{w^*}^\intercal(\mathbf{A} + \mathbf{C}_{w^*})\mathbf{v}_{\mathsf{ind}} - \mathbf{e}_{w^*}^\intercal \mathbf{K}_{\mathbf{A}}^{(w^*)} \mathbf{v}_{\mathsf{ind}} + \sum_{i \in [k]} \mathbf{h}_{w^*,\mathsf{idx}_i}^\intercal \mathbf{u}_{\mathsf{ind},i} \\
&= c_{w^*,3} - \mathbf{s}_{w^*}^\intercal \left( \sum_{i \in [k]} \mathbf{d}_{w^*,\mathsf{ind},i} + \mathbf{p} \right) - \mathbf{e}_{w^*}^\intercal (\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}^{(w^*)} \mathbf{v}_{\mathsf{ind}}) + \sum_{i \in [k]} (\mathbf{s}_{w^*}^\intercal \mathbf{T}_{w^*,\mathsf{ind},i} + \mathbf{e}_{w^*}^\intercal \mathbf{Y}_{\mathsf{idx}_i} + \mathbf{e}_{\mathsf{idx}_i}^\intercal) \mathbf{u}_{\mathsf{ind},i} \\
&= c_{w^*,3} - \mathbf{s}_{w^*}^\intercal \mathbf{p} - \mathbf{e}_{w^*}^\intercal \left( \mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}^{(w^*)} \mathbf{v}_{\mathsf{ind}} - \sum_{i \in [k]} \mathbf{Y}_{\mathsf{idx}_i} \mathbf{u}_{\mathsf{ind},i} \right) + \sum_{i \in [k]} \mathbf{e}_{\mathsf{idx}_i}^\intercal \mathbf{u}_{\mathsf{ind},i} \\
&= \mu \cdot \lfloor q/2 \rfloor - \mathbf{e}_{w^*}^\intercal \left( \mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}^{(w^*)} \mathbf{v}_{\mathsf{ind}} - \sum_{i \in [k]} \mathbf{Y}_{\mathsf{idx}_i} \mathbf{u}_{\mathsf{ind},i} - \mathbf{k}_{\mathbf{p}}^{(w^*)} \right) + \sum_{i \in [k]} \mathbf{e}_{\mathsf{idx}_i}^\intercal \mathbf{u}_{\mathsf{ind},i}.
\end{aligned}$$

Correctness holds as long as

$$\left|\mathbf{e}_{w^*}^\top\left(\mathbf{z}_{\mathsf{ind}}+\mathbf{K}_{\mathbf{A}}^{(w^*)}\mathbf{v}_{\mathsf{ind}}-\sum_{i\in[k]}\mathbf{Y}_{\mathsf{idx}_i}\mathbf{u}_{\mathsf{ind},i}-\mathbf{k}_{\mathbf{p}}^{(w^*)}\right)+\sum_{i\in[k]}\mathbf{e}_{\mathsf{idx}_i}^\top\mathbf{u}_{\mathsf{ind},i}\right|<q/4,$$

so we bound the left side of the expression. By Theorem 2.10, $\|\mathbf{v}_{\mathsf{ind}}\|\le O(\|\mathbf{R}\|\cdot m^4\log q)=O(\sigma_{\mathsf{pp}}\cdot m^{9/2}\log q)$ and $\|\mathbf{z}_{\mathsf{ind}}\|\le O(\|\mathbf{R}\|\cdot m^7\log q\log\ell)=O(\sigma_{\mathsf{pp}}\cdot m^{15/2}\log q\log\ell)$. Combining with $\|\mathbf{K}_{\mathbf{A}}^{(w^*)}\|,\|\mathbf{k}_{\mathbf{p}}^{(w^*)}\|\le 1$ and $\|\mathbf{Y}_i\|\le 1$, $\left\|\mathbf{u}_{\mathsf{ind},i}\right\|\le\sqrt{m}\sigma_{\mathsf{pp}}$, we have

$$\left\|\mathbf{K}_{\mathbf{A}}^{(w^*)}\mathbf{v}_{\mathsf{ind}}\right\|\le m\cdot\left\|\mathbf{K}_{\mathbf{A}}^{(w^*)}\right\|\cdot\|\mathbf{v}_{\mathsf{ind}}\|=O(\sigma_{\mathsf{pp}}\cdot m^{11/2}\log q),$$
$$\left\|\mathbf{Y}_{\mathsf{idx}_i}\mathbf{u}_{\mathsf{ind},i}\right\|\le m\cdot\left\|\mathbf{Y}_{\mathsf{idx}_i}\right\|\cdot\left\|\mathbf{u}_{\mathsf{ind},i}\right\|=O(m^{3/2}\sigma_{\mathsf{pp}}),$$

and hence

$$\left\|\mathbf{z}_{\mathsf{ind}}+\mathbf{K}_{\mathbf{A}}^{(w^*)}\mathbf{v}_{\mathsf{ind}}-\textstyle\sum_{i\in[k]}\mathbf{Y}_{\mathsf{idx}_i}\mathbf{u}_{\mathsf{ind},i}-\mathbf{k}_{\mathbf{p}}^{(w^*)}\right\|\le O\big(\sigma_{\mathsf{pp}}\cdot m^{15/2}\log q\log\ell+k\cdot m^{3/2}\sigma_{\mathsf{pp}}\big).$$

Since $\mathbf{e}_{w^*}\in\mathbb{Z}^{4m^5}$ and $\|\mathbf{e}_{w^*}\|\le\sqrt{m}\sigma_{\mathsf{LWE}}$, we have

$$\left|\mathbf{e}_{w^*}^\top\big(\mathbf{z}_{\mathsf{ind}}+\mathbf{K}_{\mathbf{A}}^{(w^*)}\mathbf{v}_{\mathsf{ind}}-\textstyle\sum_{i\in[k]}\mathbf{Y}_{\mathsf{idx}_i}\mathbf{u}_{\mathsf{ind},i}-\mathbf{k}_{\mathbf{p}}^{(w^*)}\big)\right|\le O\big(\sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}}\cdot m^{13}\log q\log\ell+k\cdot\sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}}\cdot m^7\big).$$

For the remaining sum, $\left|\mathbf{e}_i^\top\mathbf{u}_{\mathsf{ind},i}\right|\le m\cdot\sqrt{m}\sigma_{\mathsf{ht}}\cdot\sqrt{m}\sigma_{\mathsf{pp}}=O(m^2\sigma_{\mathsf{ht}}\sigma_{\mathsf{pp}})$, so $\left|\sum_{i\in[k]}\mathbf{e}_i^\top\mathbf{u}_{\mathsf{ind},i}\right|\le O(k\cdot m^2\sigma_{\mathsf{ht}}\sigma_{\mathsf{pp}})$. Taken all together, we have

$$\left|\mathbf{e}_{w^*}^\top\left(\mathbf{z}_{\mathsf{ind}}+\mathbf{K}_{\mathbf{A}}^{(w^*)}\mathbf{v}_{\mathsf{ind}}-\sum_{i\in[k]}\mathbf{Y}_{\mathsf{idx}_i}\mathbf{u}_{\mathsf{ind},i}-\mathbf{k}_{\mathbf{p}}^{(w^*)}\right)+\sum_{i\in[k]}\mathbf{e}_{\mathsf{idx}_i}^\top\mathbf{u}_{\mathsf{ind},i}\right|<O\big(k\cdot\sigma_{\mathsf{pp}}\sigma_{\mathsf{ht}}\sigma_{\mathsf{LWE}}\cdot m^{13}\log q\log\ell\big).$$

Correctness follows by setting $q>k\cdot O(\sigma_{\mathsf{pp}}\sigma_{\mathsf{ht}}\sigma_{\mathsf{LWE}}\cdot m^{13}\log q\log L)$.

## A.2 Proof of Theorem 4.3 (Static Security)

Take any efficient and admissible adversary $\mathcal{A}$ for the static security game, and suppose $\mathcal{A}$ wins the game with non-negligible advantage $\varepsilon$. We now define a sequence of hybrid experiments parameterized by a bit $b\in\{0,1\}$.

- $\mathsf{Hyb}_0^{(b)}$: This is the static security game with challenge bit $b\in\{0,1\}$:
  - **Setup phase:** On input the security parameter $1^\lambda$, the challenger samples

$$\mathsf{crs}_{\mathsf{NIZK}}\leftarrow\mathsf{NIZK.Setup}(1^\lambda),\ \mathbf{p}\xleftarrow{\mathrm{R}}\mathbb{Z}_q^n,\ \mathbf{W}\xleftarrow{\mathrm{R}}\mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R}\leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3}.$$

    If $\|\mathbf{R}\|>\sqrt{m}\sigma_{\mathsf{pp}}$, the challenger sets $\mathbf{R}=\mathbf{0}^{m\times 2m^3}$ instead. Let

$$\mathbf{B}=\mathbf{W}(\mathbf{I}_{2m^2}\otimes\mathbf{R})+\mathsf{vec}(\mathbf{I}_{2m^2})^\top\otimes\mathbf{G}\in\mathbb{Z}_q^{n\times 4m^5}$$

    and $\mathsf{pp}_{\mathsf{com}}=(\mathbf{W},\mathbf{R},\mathbf{B})$. The challenger gives the public parameters $\mathsf{pp}=(\mathsf{crs}_{\mathsf{NIZK}},\mathsf{pp}_{\mathsf{com}},\mathbf{p})$ to $\mathcal{A}$ and initializes a counter $\mathsf{ctr}=0$.

– **Key-generation queries:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $\mathbf{Y}_{\mathsf{ctr}} \xleftarrow{R} \{0,1\}^{4m^5 \times m}$, and sends $\mathsf{pk}_{\mathsf{ctr}} = \mathbf{T}_{\mathsf{ctr}} = \mathbf{B}\mathbf{Y}_{\mathsf{ctr}}$ to $\mathcal{A}$.

– **Hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, \mathsf{ind})$ for $\mathsf{ind} \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where

$$\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\tau}, \hat{\mathbf{A}}, \{\hat{\mathbf{u}}_{i,j}\}_{i\in[\ell],j\in[k]}, \{\hat{\mathbf{c}}_{w,1}^\top, \hat{\mathbf{c}}_{w,2}^\top, \hat{c}_{w,3}\}_{w\in[\ell]}) \tag{A.1}$$

Then it checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1,$$

and replies with $\perp$ if the check fails. Otherwise, the challenger samples error $\hat{\mathbf{e}}_w \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and sets $\hat{\mathbf{e}}_w = \mathbf{0}^m$ if $\|\hat{\mathbf{e}}_w\| > \sqrt{m}\cdot\sigma_{\mathsf{ht}}$ for $w \in [\ell]$. It replies with $\{\hat{\mathbf{c}}_{w,1}^\top \mathbf{Y}_{\mathsf{ind}} + \hat{\mathbf{e}}_w^\top\}_{w\in[\ell]}$.

– **Challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P = (S_1, \ldots, S_L) \in \mathcal{P}_{\mathsf{DNF}}$ over $N$ variables, where $|S_i| = k$ and $L = \ell^2$ for $\ell \in \mathbb{N}$, and specifies either a public key $\mathsf{pk}_i^* = \mathbf{T}_i^*$ or a counter value $\mathsf{ctr}_i$ for each $i \in [N]$. For each $i \in [N]$ where $\mathcal{A}$ specified a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$, the challenger sets $\mathsf{pk}_i^* = \mathbf{T}_i^* = \mathbf{T}_{\mathsf{ctr}_i}$. The challenger parses the policy into chunks $P_1, \ldots, P_\ell$, where $P_w = (S_{w,1}, \ldots, S_{w,\ell}) = (S_{(w-1)\ell+1}, \ldots, S_{w\ell})$ for $w \in [\ell]$. To construct the challenge ciphertext $\mathsf{ct}^*$, it first samples the following shared components:

$$\mathbf{u}_{i,j} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}} \;\; \forall i \in [\ell], j \in [k], \;\; \tau \xleftarrow{R} \{0,1\}^\lambda, \mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m},$$

and sets $\mathbf{u}_{i,j} = \mathbf{0}^m$ if $\|\mathbf{u}_{i,j}\| > \sqrt{m}\cdot\sigma_{\mathsf{pp}}$. Then, for $w \in [\ell]$, the challenger does the following:

1. Sample $\mathbf{s}_w \xleftarrow{R} \mathbb{Z}_q^n, \mathbf{K}_{\mathbf{A}}^{(w)} \xleftarrow{R} \{0,1\}^{4m^5\times m}, \mathbf{k}_{\mathbf{p}}^{(w)} \xleftarrow{R} \{0,1\}^{4m^5}, \mathbf{e}_w \leftarrow D^{4m^5}_{\mathbb{Z},\sigma_{\mathsf{LWE}}}$ and set $\mathbf{e}_w = \mathbf{0}^{4m^5}$ if $\|\mathbf{e}_w\| > \sqrt{m}\cdot\sigma_{\mathsf{LWE}}$.
2. Let $\mathbf{T}^*_{w,i,j}$ be the $j^{\text{th}}$ public key indicated by $S_{w,i}$. For $i \in [\ell], j \in [k]$ set $\mathbf{d}_{w,i,j} = \mathbf{T}^*_{w,i,j}\cdot\mathbf{u}_{i,j}$.
3. For each $i \in [\ell]$, set $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and

$$\mathbf{M}_w = [\mathbf{m}_{w,1} \mid \cdots \mid \mathbf{m}_{w,\ell}] \in \mathbb{Z}_q^{n\times\ell} \;\text{ where }\; \forall i \in [\ell] : \mathbf{m}_{w,i} = \sum_{j\in[k]} \mathbf{d}_{w,i,j} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \in \mathbb{Z}_q^n.$$

   Compute $\mathbf{C}_w = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}_w) \in \mathbb{Z}_q^{n\times m}$.
4. Set $\mathbf{c}_{w,1}^\top = \mathbf{s}_w^\top\mathbf{B} + \mathbf{e}_w^\top$, $\mathbf{c}_{w,2}^\top = \mathbf{s}_w^\top(\mathbf{A} + \mathbf{C}_w) + \mathbf{e}_w^\top\mathbf{K}_{\mathbf{A}}^{(w)}$, and $c_{w,3} = \mathbf{s}_w^\top\mathbf{p} + \mathbf{e}_w^\top\mathbf{k}_{\mathbf{p}}^{(w)} + b\cdot\lfloor q/2\rfloor$.

Finally, the challenger sets

$$\mathsf{ct}^*_{\mathsf{main}} = (\tau, \mathbf{A}, \{\mathbf{u}_{i,j}\}_{i\in[\ell],j\in[k]}, \{\mathbf{c}_{w,1}^\top, \mathbf{c}_{w,2}^\top, c_{w,3}\}_{w\in[\ell]})$$

and computes the proof

$$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), (\mathbf{s}_w, \mathbf{e}_w)_{w\in[\ell]}),$$

and gives $\mathsf{ct}^* = (\pi^*_{\mathsf{ct}}, \mathsf{ct}^*_{\mathsf{main}})$ to $\mathcal{A}$.

– **(Post-challenge) hint-generation queries:** Algorithm $\mathcal{A}$ can continue to make hint queries on ciphertexts $(\hat{\mathsf{ct}}, \mathsf{ind})$ where $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$. The challenger answers these queries exactly like the pre-challenge hint-generation queries.

- **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. Note that if $\mathcal{A}$ aborts early, then the output of the experiment is 0.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$ except the challenger samples the tag $\tau \xleftarrow{R} \{0,1\}^\lambda$ for the challenge ciphertext at setup time. The challenger now halts with output 0 if any pre-challenge hint-generation query includes $\tau$ as part of the ciphertext.

- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger replaces the NIZK proof in the challenge ciphertext with a simulated NIZK proof. Specifically, the challenger proceeds as follows:

  - During setup, the challenger samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$.
  - During the challenge phase, after constructing $\mathsf{ct}^*_{\mathsf{main}}$, the challenger now runs

$$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}})).$$

- $\mathsf{Hyb}_3^{(b)}$: Same as $\mathsf{Hyb}_2^{(b)}$ except the challenger extracts the encryption randomness for hint-generation queries:

  - When the adversary makes a hint-generation query $(\hat{\mathsf{ct}}, \mathsf{ind})$ for an index $\mathsf{ind} \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}}$ is parsed according to Eq. (A.1). Then, for $w \in [\ell]$ it computes

$$(\tilde{\mathbf{s}}_w, \tilde{\mathbf{e}}_w) = \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}).$$

    If $\|\tilde{\mathbf{e}}_w\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_{w,1}^\mathsf{T} \neq \tilde{\mathbf{s}}_w^\mathsf{T}\mathbf{B} + \tilde{\mathbf{e}}_w^\mathsf{T}$ for any $w \in [\ell]$, the challenger outputs 0.

- $\mathsf{Hyb}_4^{(b)}$: Same as $\mathsf{Hyb}_3^{(b)}$ except when answering hint-generation queries, the challenger samples $\hat{\mathbf{e}}_w \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and does not check $\|\hat{\mathbf{e}}_w\|$ for any $w \in [\ell]$.

- $\mathsf{Hyb}_5^{(b)}$: Same as $\mathsf{Hyb}_4^{(b)}$ except the challenger now simulates the hints using the extracted randomness. In particular, for a hint-generation query $(\hat{\mathsf{ct}}, \mathsf{ind})$ where $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ and $\hat{\mathsf{ct}}_{\mathsf{main}}$ is parsed as in Eq. (A.1), the challenger first checks the proof $\hat{\pi}_{\mathsf{ct}}$. If the proof verifies, then the challenger extracts $(\tilde{\mathbf{s}}_w, \tilde{\mathbf{e}}_w)$ as in $\mathsf{Hyb}_4^{(b)}$. Then it samples $\hat{\mathbf{e}}_w \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ for $w \in [\ell]$ and replies with $\{\tilde{\mathbf{s}}_w^\mathsf{T}\mathbf{T}_{\mathsf{ind}} + \hat{\mathbf{e}}_w^\mathsf{T}\}_{w\in[\ell]}$. At this point, the challenger's logic in the experiment no longer requires knowledge of the secret keys $\mathbf{Y}_{\mathsf{ind}}$.

  **Indexing:** In the following analysis, we additionally index hybrids by $w \in [\ell]$. Let $\mathsf{Hyb}_{5,w}^{(b)}$ denote $\mathsf{Hyb}_5^{(b)}$ where, for each $w' \in [\ell]$, the challenger sets the component

$$c_{w',3} = \mathbf{s}_{w'}^\mathsf{T}\mathbf{p} + \mathbf{e}_{w'}^\mathsf{T}\mathbf{k}_{\mathbf{p}}^{(w')} + \mu_{w'} \cdot \lfloor q/2 \rfloor$$

  in the challenge ciphertext, where

$$\mu_{w'} = \begin{cases} 1 & w' < w \\ b & w' = w \\ 0 & w' > w. \end{cases}$$

  In other words, the blocks before $w$ encrypt 1, block $w$ encrypts the challenge bit $b$, and the blocks after $w$ encrypt 0. In particular, $\mathsf{Hyb}_{5,1}^{(0)} \equiv \mathsf{Hyb}_5^{(0)}$, $\mathsf{Hyb}_{5,w}^{(1)} \equiv \mathsf{Hyb}_{5,w+1}^{(0)}$ for $w \in [\ell-1]$, and $\mathsf{Hyb}_{5,\ell}^{(1)} \equiv \mathsf{Hyb}_5^{(1)}$.

- $\mathsf{Hyb}^{(b)}_{6,w}$: Same as $\mathsf{Hyb}^{(b)}_{5,w}$ except the challenger no longer truncates $\mathbf{u}_{i,\mathsf{idx}_i}$ for $i \in [\ell]$ after sampling, where for each $i \in [\ell]$, $\mathsf{idx}_i \in [k]$ is the smallest index in $S_{w,i}$ for which $\mathcal{A}$ specified a counter value in the challenge phase (i.e., the smallest uncorrupted position in $S_{w,i}$, which must exist by admissibility of $\mathcal{A}$). In addition, in the setup phase, the challenger no longer checks if $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$, and when constructing the challenge ciphertext, it no longer checks if $\|\mathbf{e}_w\| > \sqrt{m}\sigma_{\mathsf{LWE}}$.

- $\mathsf{Hyb}^{(b)}_{7,w}$: Same as $\mathsf{Hyb}^{(b)}_{6,w}$ except the challenger samples $\mathbf{T}_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times m}$ when answering key-generation queries.

- $\mathsf{Hyb}^{(b)}_{8,w}$: Same as $\mathsf{Hyb}^{(b)}_{7,w}$ except the challenger changes how it samples $\mathbf{u}_{i,\mathsf{idx}_i}$ and $\mathbf{d}_{i,\mathsf{idx}_i}$ for the special index $\mathsf{idx}_i \in [k]$ in each min-term $i \in [\ell]$. Specifically, in the challenge phase, the challenger now samples $\mathbf{d}_{w,i,\mathsf{idx}_i} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow (\mathbf{T}^*_{w,i,\mathsf{idx}_i})^{-1}_{\sigma_{\mathsf{pp}}}(\mathbf{d}_{w,i,\mathsf{idx}_i})$. The remaining $\mathbf{u}_{i,j}$ for $j \in [k] \setminus \{\mathsf{idx}_i\}$ are sampled as in $\mathsf{Hyb}^{(b)}_{7,w}$.

- $\mathsf{Hyb}^{(b)}_{9,w}$: Same as $\mathsf{Hyb}^{(b)}_{8,w}$ except the challenger samples $(\mathbf{T}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$ when answering key-generation queries. In the challenge phase, it labels $\mathsf{td}^*_{w,i,j}$ analogously to $\mathbf{T}^*_{w,i,j}$ for the honest users.

- $\mathsf{Hyb}^{(b)}_{10,w}$: Same as $\mathsf{Hyb}^{(b)}_{9,w}$ except the challenger uses SamplePre to sample $\mathbf{u}_{i,\mathsf{idx}_i}$ in the challenge phase. Specifically, for each $i \in [\ell]$, the challenger samples

$$\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow \mathsf{SamplePre}(\mathbf{T}^*_{w,i,\mathsf{idx}_i}, \mathsf{td}^*_{w,i,\mathsf{idx}_i}, \mathbf{d}_{w,i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}})$$

  instead of sampling $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow (\mathbf{T}^*_{w,i,\mathsf{idx}_i})^{-1}_{\sigma_{\mathsf{pp}}}(\mathbf{d}_{w,i,\mathsf{idx}_i})$.

- $\mathsf{Hyb}^{(b)}_{11,w}$: Same as $\mathsf{Hyb}^{(b)}_{10,w}$ except the challenger programs $\mathbf{d}_{w,i,\mathsf{idx}_i}$ in each min-term $i \in [\ell]$. Specifically, in the challenge phase, the challenger starts by sampling $\mathbf{d}^*_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ for each $i \in [\ell]$. Then, for all $i \in [\ell]$ and $j \in [k] \setminus \{\mathsf{idx}_i\}$, it samples $\mathbf{u}_{i,j} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}}$ and sets $\mathbf{d}_{w,i,j} = \mathbf{T}^*_{w,i,j} \cdot \mathbf{u}_{i,j}$. Afterwards, it sets

$$\mathbf{d}_{w,i,\mathsf{idx}_i} = \mathbf{d}^*_i - \sum_{j\in[k]\setminus\{\mathsf{idx}_i\}} \mathbf{d}_{w,i,j} + \mathbf{A}\mathbf{v}_i - \mathbf{p}$$

  for each $i \in [\ell]$.

- $\mathsf{Hyb}^{(b)}_{12,w}$: Same as $\mathsf{Hyb}^{(b)}_{11,w}$ except the challenger samples $\mathbf{k}^{(w)}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ at setup time and sets $\mathbf{p} = \mathbf{B}\mathbf{k}^{(w)}_{\mathbf{p}}$. At the start of the challenge phase, the challenger defines $\mathbf{D}^* = [\mathbf{d}^*_1 \mid \cdots \mid \mathbf{d}^*_\ell]$, sets $\mathbf{C}_w = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$, and sets $\mathbf{A} = \mathbf{B}\mathbf{K}^{(w)}_{\mathbf{A}} - \mathbf{C}_w$. The challenger uses the matrix $\mathbf{C}_w$ when constructing the challenge ciphertext.

- $\mathsf{Hyb}^{(b)}_{13,w}$: Same as $\mathsf{Hyb}^{(b)}_{12,w}$ except the challenger samples the component $\mathbf{c}^\intercal_{w,1} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{4m^5}$, then computes $\mathbf{c}^\intercal_{w,2} = \mathbf{c}^\intercal_{w,1}\mathbf{K}^{(w)}_{\mathbf{A}}$ and $c_{w,3} = \mathbf{c}^\intercal_{w,1}\mathbf{k}^{(w)}_{\mathbf{p}} + b \cdot \lfloor q/2 \rfloor$.

- $\mathsf{Hyb}^{(b)}_{14,w}$: Same as $\mathsf{Hyb}^{(b)}_{13,w}$ except the challenger samples $\mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and $c_{w,3} \xleftarrow{\mathrm{R}} \mathbb{Z}_q$.

We write $\mathsf{Hyb}^{(b)}_i(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid $\mathsf{Hyb}^{(b)}_i$ with adversary $\mathcal{A}$ (and an implicit security parameter $\lambda$). We now bound the difference between the output distributions of each adjacent pair of hybrid experiments.

**Lemma A.1.** *There exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.4. □

**Lemma A.2.** *Suppose* $\Pi_{\mathsf{NIZK}}$ *satisfies zero-knowledge. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.5. □

**Lemma A.3.** *Suppose* $\Pi_{\mathsf{NIZK}}$ *satisfies simulation-sound extractability. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.6. □

**Lemma A.4.** *Suppose* $m \geq \lambda$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.7. □

**Lemma A.5.** *Suppose* $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$ *and all* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.8. □

**Lemma A.6.** *Suppose* $m \geq \lambda$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$, $w \in [\ell]$, *and* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{5,w}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{6,w}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.10. □

**Lemma A.7.** *Suppose* $n \geq \lambda, m \geq 2n\log q, q > 2$ *is prime, and* $\sigma_{\mathsf{pp}} > \log m$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$, $w \in [\ell]$, *and* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{6,w}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{7,w}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.11. □

**Lemma A.8.** *Suppose* $n \geq \lambda, m \geq 2n\log q, q > 2$ *is prime, and* $\sigma_{\mathsf{pp}} > \log m$. *Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $b \in \{0,1\}$, $w \in [\ell]$, *and* $\lambda \in \mathbb{N}$,

$$|\Pr[\mathsf{Hyb}_{7,w}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{8,w}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.12. □

**Lemma A.9.** *Suppose $m \geq 3n \log q$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$, $w \in [\ell]$, and $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{8,w}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{9,w}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.13. □

**Lemma A.10.** *Suppose $m \geq 3n \log q$ and $\sigma_{\mathsf{pp}} \geq m \log n$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$, $w \in [\ell]$, and $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{9,w}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{10,w}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.14. □

**Lemma A.11.** *For all $w \in [\ell]$ and $b \in \{0,1\}$, $\Pr[\mathsf{Hyb}^{(b)}_{10,w}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}^{(b)}_{11,w}(\mathcal{A}) = 1]$.*

*Proof.* Follows via a similar argument as in the proof of Lemma 3.15. □

**Lemma A.12.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} > \log m$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$, $w \in [\ell]$, and $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{11,w}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{12,w}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.16. □

**Lemma A.13.** *Suppose the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$ holds. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$, $w \in [\ell]$, and $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{12,w}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{13,w}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Suppose $|\Pr[\mathsf{Hyb}^{(b)}_{12,w}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{13,w}(\mathcal{A}) = 1]| = \delta$ for some non-negligible $\delta$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathbf{W}, \mathbf{R}, \mathbf{c}^\intercal)$ from the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE challenger and starts running $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ sets $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times 4m^5}$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. Then algorithm $\mathcal{B}$ samples

  $$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda),\ \mathbf{k}_{\mathbf{p}}^{(w)} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5},\ \tau \xleftarrow{\mathrm{R}} \{0,1\}^\lambda.$$

  Algorithm $\mathcal{B}$ defines $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}^{(w)}$ and sets $\mathsf{ctr} = 0$. It gives $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{p})$ to $\mathcal{A}$.
- **Key-generation queries:** When algorithm $\mathcal{A}$ makes a key-generation query, algorithm $\mathcal{B}$ increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $(\mathbf{T}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$, and sends $\mathsf{pk}_{\mathsf{ctr}} = \mathbf{T}_{\mathsf{ctr}}$ to $\mathcal{A}$.
- **Hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint query $(\hat{\mathsf{ct}}, \mathsf{ind})$ for $\mathsf{ind} \leq \mathsf{ctr}$, algorithm $\mathcal{B}$ parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}}$ is parsed according to Eq. (A.1). Then it checks that

  $$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1,$$

and replies with $\perp$ if the check fails. For pre-challenge hint queries, algorithm $\mathcal{B}$ aborts and outputs 0 if $\hat{\tau} = \tau$. Otherwise, for $w \in [\ell]$, algorithm $\mathcal{B}$ computes

$$(\tilde{\mathbf{s}}_w, \tilde{\mathbf{e}}_w) = \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}).$$

If $\|\tilde{\mathbf{e}}_w\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_{w,1}^\mathsf{T} \neq \tilde{\mathbf{s}}_w^\mathsf{T}\mathbf{B} + \tilde{\mathbf{e}}_w^\mathsf{T}$ for any $w \in [\ell]$, algorithm $\mathcal{B}$ outputs 0. Otherwise, algorithm $\mathcal{B}$ samples $\hat{\mathbf{e}}_w \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ for $w \in [\ell]$ and replies with $\{\tilde{\mathbf{s}}_w^\mathsf{T}\mathbf{T}_{\mathsf{ind}} + \hat{\mathbf{e}}_w^\mathsf{T}\}_{w\in[\ell]}$.

- **Challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P = (S_1, \ldots, S_L) \in \mathcal{P}_{\mathsf{DNF}}$ over $N$ variables and specifies either a public key $\mathsf{pk}_i^* = \mathbf{T}_i^*$ or a counter value $\mathsf{ctr}_i$ for each $i \in [N]$. For each $i \in [N]$ where $\mathcal{A}$ specified a counter value $\mathsf{ctr}_i \leq \mathsf{ctr}$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i^* = \mathbf{T}_i^* = \mathbf{T}_{\mathsf{ctr}_i}$ and $\mathsf{td}_i^* = \mathsf{td}_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ parses the policy into chunks $P_1, \ldots, P_\ell$, where $P_w = (S_{w,1}, \ldots, S_{w,\ell})$ for $w \in [\ell]$ and constructs the challenge ciphertext $\mathsf{ct}^*$ as follows:
  - Sample $\mathbf{d}_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ for each $i \in [\ell]$ and $\mathbf{K}_{\mathbf{A}}^{(w)} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$.
  - Let $\mathbf{D}^* = [\mathbf{d}_1^* \mid \cdots \mid \mathbf{d}_\ell^*]$ and set $\mathbf{C}_w = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$. Let $\mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}}^{(w)} - \mathbf{C}_w$.
  - For each $i \in [\ell]$, let $\mathsf{idx}_i \in [k]$ be the smallest position in $S_{w,i}$ for which $\mathcal{A}$ specified a counter value in the challenge phase. For $i \in [\ell]$ and $j \in [k] \setminus \{\mathsf{idx}_i\}$, algorithm $\mathcal{B}$ samples $\mathbf{u}_{i,j} \leftarrow D^m_{\mathbb{Z},\sigma_{\mathsf{pp}}}$ and sets $\mathbf{d}_{w,i,j} = \mathbf{T}^*_{w,i,j} \cdot \mathbf{u}_{i,j}$.
  - For each $i \in [\ell]$, algorithm $\mathcal{B}$ computes $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and sets

$$\mathbf{d}_{w,i,\mathsf{idx}_i} = \mathbf{d}_i^* - \sum_{j \in [k] \setminus \{\mathsf{idx}_i\}} \mathbf{d}_{w,i,j} + \mathbf{A}\mathbf{v}_i - \mathbf{p}.$$

    Then, algorithm $\mathcal{B}$ samples $\mathbf{u}_{i,\mathsf{idx}_i} \leftarrow \mathsf{SamplePre}(\mathbf{T}^*_{w,i,\mathsf{idx}_i}, \mathsf{td}^*_{w,i,\mathsf{idx}_i}, \mathbf{d}_{w,i,\mathsf{idx}_i}, \sigma_{\mathsf{pp}})$. Algorithm $\mathcal{B}$ sets $\mathbf{c}_{w,1}^\mathsf{T} = \mathbf{c}^\mathsf{T}$, $\mathbf{c}_{w,2}^\mathsf{T} = \mathbf{c}^\mathsf{T}\mathbf{K}_{\mathbf{A}}^{(w)}$ and $c_{w,3} = \mathbf{c}^\mathsf{T}\mathbf{k}_{\mathbf{p}}^{(w)} + b \cdot \lfloor q/2 \rfloor$.
  - For $w' < w$, algorithm $\mathcal{B}$ samples the components $(\mathbf{c}_{w',1}^\mathsf{T}, \mathbf{c}_{w',2}^\mathsf{T}, c_{w',3})$ as in $\mathsf{Hyb}_0^{(1)}$. For $w'' > w$, algorithm $\mathcal{B}$ samples the components $(\mathbf{c}_{w'',1}^\mathsf{T}, \mathbf{c}_{w'',2}^\mathsf{T}, c_{w'',3})$ as in $\mathsf{Hyb}_0^{(0)}$. Algorithm $\mathcal{B}$ sets

$$\mathsf{ct}^*_{\mathsf{main}} = (\tau, \mathbf{A}, \{\mathbf{u}_{i,j}\}_{i\in[\ell], j\in[k]}, \{\mathbf{c}_{w,1}^\mathsf{T}, \mathbf{c}_{w,2}^\mathsf{T}, c_{w,3}\}_{w\in[\ell]})$$

    and computes the proof

$$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct},k,\ell}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}})).$$

  Finally, algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* = (\pi^*_{\mathsf{ct}}, \mathsf{ct}^*_{\mathsf{main}})$.
- **Post-challenge hint-generation queries:** Algorithm $\mathcal{B}$ handles post-challenge hint queries exactly the same as pre-challenge queries, except instead of aborting when $\hat{\tau} = \tau$, it aborts when $\hat{\mathsf{ct}} = \mathsf{ct}^*$.
- **Output phase:** At the end of the game, if algorithm $\mathcal{A}$ outputs $b' \in \{0,1\}$, $\mathcal{B}$ also outputs $b'$.

By construction, algorithm $\mathcal{B}$ efficiently simulates the setup phase, the public keys, and the hint queries exactly as in $\mathsf{Hyb}_{12,w}^{(b)}$ or $\mathsf{Hyb}_{13,w}^{(b)}$. If $\mathbf{c}^\mathsf{T}$ is a decomposed LWE sample, then algorithm $\mathcal{B}$ constructs the challenge ciphertext exactly as described in $\mathsf{Hyb}_{12,w}^{(b)}$. If $\mathbf{c}^\mathsf{T}$ is uniform random, then algorithm $\mathcal{B}$ constructs the challenge ciphertext exactly as described in $\mathsf{Hyb}_{13,w}^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE with advantage $\delta$. $\square$

**Lemma A.14.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} > \log m$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$, $w \in [\ell]$, and $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}^{(b)}_{13,w}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{14,w}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.18. □

Since the challenger's behavior in $\mathsf{Hyb}^{(b)}_{14,w}(\mathcal{A})$ is independent of $b \in \{0,1\}$, we have $\Pr[\mathsf{Hyb}^{(0)}_{14,w}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}^{(1)}_{14,w}(\mathcal{A}) = 1]$ for all $w \in [\ell]$. Since $\ell = \mathsf{poly}(\lambda)$, security follows by Lemmas A.1 to A.14 and a standard hybrid argument.

# B DMPE for DNFs with Short Ciphertexts in the Plain Model

We now show how to construct distributed monotone-policy encryption with short ciphertexts in the plain model at the cost of a security notion that does not capture rogue-key attacks. For ease of exposition, we assume that each variable (i.e., each user's public key) appears at most once in the policy. It is straightforward to support formulas where each variable appears at most $K$ times by having each user generate $K$ independent public keys, one for each copy of the variable. This generic transformation increases the public key size of each user by a factor of $K$, but does not affect the public parameter or ciphertext size. This is the approach taken in both [CW26, AGY26]. Compared to the construction in Section 4, this construction has optimal ciphertexts at the cost of potentially large public keys and much weaker security.

**Security notion.** We start by stating the security notion we obtain for the construction in this section.

**Definition B.1** (Semi-Policy-and-Very-Query-Selective Security)**.** We say that a distributed monotone-policy encryption scheme $\Pi_{\mathsf{DMPE}}$ is *semi-policy-and-very-query-selective secure in the static model* if the adversary in Definition 2.13 is static and moreover, the adversary declares its challenge policy $P$, the indices associated with the uncorrupted users, and its keys for the corrupted users at the beginning of the pre-challenge phase (but *after* seeing the public parameters). The challenger then sends back the honest keys and challenge ciphertext. In this model, the adversary is still allowed to choose the public keys for the corrupted users and make post-challenge hint queries that do not include the challenge ciphertext.

We now give our formal construction:

**Construction B.2** (Distributed Monotone-Policy Encryption)**.** Let $\lambda$ be a security parameter. We define the following:

- Let $(n, m, q, \sigma_{\mathsf{LWE}})$ be LWE parameters (which may be functions of $\lambda$). Let $\sigma_{\mathsf{pp}}, \sigma_{\mathsf{ht}} > 0$ be Gaussian width parameters.
- Let $\mathcal{P}_{\mathsf{DNF}}$ be the set of DNF formulas on up to $2^\lambda$-bit inputs where each input variable is used at most once. We will represent a DNF $P \in \mathcal{P}_{\mathsf{DNF}}$ on input variables $x_1, \ldots, x_N$ as a collection of $\ell$ *mutually disjoint* sets $S_1, \ldots, S_\ell \subseteq [N]$, where $S_1, \ldots, S_\ell$ represents the policy $\bigvee_{i \in [\ell]} \left(\bigwedge_{j \in S_i} x_j\right)$. The condition $S_1, \ldots, S_\ell$ being mutually disjoint is equivalent to each variable appearing at most once in the DNF formula.
- Let $C_{\mathsf{ct}}$ be the Boolean circuit that takes a statement $(\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \beta)$, a witness $(\mathbf{s}, \mathbf{e})$, and outputs 1 if $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal$ and $\|\mathbf{e}\| \leq \beta$, where $\mathsf{ct}_{\mathsf{main}} = (\mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$ with $\mathbf{c}_1^\intercal \in \mathbb{Z}_q^{4m^5}$, $\mathbf{c}_2^\intercal \in \mathbb{Z}_q^m$, and $c_3 \in \mathbb{Z}_q$.

- Let $\Pi_{\mathsf{NIZK}} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.TrapSetup}, \mathsf{NIZK.Prove}, \mathsf{NIZK.Verify}, \mathsf{NIZK.Sim}, \mathsf{NIZK.Extract})$ be a simulation-sound extractable NIZK argument for a family $C = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ where $C_\lambda$ contains the circuits $C_{\mathsf{ct}}$. Note that $C_{\mathsf{ct}}$ can be computed by a circuit of depth $\mathsf{poly}(n, m, \log q)$.

We construct a DMPE scheme $\Pi_{\mathsf{DMPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ for $\mathcal{P}_{\mathsf{DNF}}$ as follows:

- $\mathsf{Setup}(1^\lambda)$: On input the security parameter $\lambda$, the setup algorithm samples

$$\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda),\ \mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times m},\ \mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n},\ \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3}.$$

  If $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$, set $\mathbf{R} = \mathbf{0}^{m\times 2m^3}$. Let

$$\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\mathsf{T} \otimes \mathbf{G} \in \mathbb{Z}_q^{n\times 4m^5} \quad \text{and} \quad \mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B}).$$

  Finally, it outputs $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$.

- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$, the key-generation algorithm samples $\mathbf{y} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ and computes $\mathbf{t} = \mathbf{B}\mathbf{y} \in \mathbb{Z}_q^n$. It outputs the public key $\mathsf{pk} = \mathbf{t}$ and the secret key $\mathsf{sk} = \mathbf{y}$.

- $\mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \mu)$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$, a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$, a collection of public keys $\mathsf{pk}_j = \mathbf{t}_j$ for $j \in [N]$, and a message $\mu \in \{0,1\}$, the encryption algorithm samples

$$\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{e} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{LWE}}}^{4m^5},\ \mathbf{K}_\mathbf{A} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5\times m},\ \mathbf{k}_\mathbf{p} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}.$$

  If $\|\mathbf{e}\| > \sqrt{m}\cdot\sigma_{\mathsf{LWE}}$, it sets $\mathbf{e} = \mathbf{0}^{4m^5}$. Next, for each $i \in [\ell]$, it sets $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$ and

$$\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell] \in \mathbb{Z}_q^{n\times\ell} \quad \text{where} \quad \forall i \in [\ell] : \mathbf{m}_i = \sum_{j\in S_i} \mathbf{t}_j + \mathbf{p} - \mathbf{A}\mathbf{v}_i \in \mathbb{Z}_q^n. \tag{B.1}$$

  It computes the commitment

$$\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \in \mathbb{Z}_q^{n\times m}.$$

  Then, it sets $\mathbf{c}_1^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T}$, $\mathbf{c}_2^\mathsf{T} = \mathbf{s}^\mathsf{T}(\mathbf{A}+\mathbf{C}) + \mathbf{e}^\mathsf{T}\mathbf{K}_\mathbf{A}$, and $c_3 = \mathbf{s}^\mathsf{T}\mathbf{p} + \mathbf{e}^\mathsf{T}\mathbf{k}_\mathbf{p} + \mu\cdot\lfloor q/2\rfloor$. Finally, it sets $\mathsf{ct}_{\mathsf{main}} = (\mathbf{c}_1^\mathsf{T}, \mathbf{c}_2^\mathsf{T}, c_3)$ and computes the proof

$$\pi_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e})),$$

  and outputs $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$.

- $\mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$: On input the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ where $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$, a secret key $\mathsf{sk}_i = \mathbf{y}_i$, and a ciphertext $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}} = (\mathbf{c}_1^\mathsf{T}, \mathbf{c}_2^\mathsf{T}, c_3)$, the hint-generation algorithm checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m}\cdot\sigma_{\mathsf{LWE}}), \pi_{\mathsf{ct}}) = 1.$$

  If the check fails, it outputs $\perp$. Otherwise, it samples $e \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}$, sets $e = 0$ if $|e| > \sqrt{m}\cdot\sigma_{\mathsf{ht}}$, and outputs $\mathbf{c}_1^\mathsf{T}\mathbf{y}_i + e$.

- $\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct})$: On input the parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$, a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$, a collection of public keys $\mathsf{pk}_j = \mathbf{t}_j$ for $j \in [N]$, a list of decryption hints $\mathsf{ht}_i = h_i$ for $i \in T \subseteq [N]$, and a ciphertext $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where $\mathsf{ct}_{\mathsf{main}} = (\mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$, the decryption algorithm outputs $\perp$ if there does not exist an index $\mathsf{ind} \in [\ell]$ where $S_{\mathsf{ind}} \subseteq T$. Otherwise, let $\mathsf{ind} \in [\ell]$ be the smallest such index. Then, for each $i \in [\ell]$, it sets $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$. Define the matrix $\mathbf{M}$ as in Eq. (B.1). Compute $\mathbf{z}_{\mathsf{ind}} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}, \mathsf{ind})$ and

$$\mu' = c_3 - \mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{c}_2^\intercal \mathbf{v}_{\mathsf{ind}} + \sum_{i \in S_{\mathsf{ind}}} h_i.$$

  Output 0 if $-q/4 < \mu' < q/4$ and 1 otherwise.

**Theorem B.3** (Correctness). *Suppose $q > N \cdot O(m^{13} \sigma_{\mathsf{pp}} \sigma_{\mathsf{ht}} \sigma_{\mathsf{LWE}} \log q \log N)$ and $\Pi_{\mathsf{NIZK}}$ is complete. Then, Construction B.2 is correct.*

*Proof.* Take any $\lambda \in \mathbb{N}$, any $N \le 2^\lambda$, any $P \in \mathcal{P}_{\mathsf{DNF}}$ on $N$-bit inputs, any set $T \subseteq [N]$ where $P(T) = 1$, any message $\mu \in \{0,1\}$, any pp in the support of $\mathsf{Setup}(1^\lambda)$, any $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for $i \in T$, and any choice of $\mathsf{pk}_i$ for $i \notin T$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \mu)$ and for each $i \in T$, let $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{pp}, \mathsf{sk}_i, \mathsf{ct})$. Now, consider the output of

$$\mathsf{Decrypt}(\mathsf{pp}, P, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), \{(i, \mathsf{ht}_i)\}_{i \in T}, \mathsf{ct}).$$

We analyze each step individually:

- Let $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. By definition of Setup, we have $\|\mathbf{R}\| \le \sqrt{m}\sigma_{\mathsf{pp}}$.
- For $i \in [N]$, we have $\mathsf{pk}_i = \mathbf{t}_i \in \mathbb{Z}_q^n$. Additionally, for $i \in T$ we have $\mathbf{t}_i = \mathbf{B}\mathbf{y}_i$ where $\mathbf{y}_i \in \{0,1\}^{4m^5}$.
- For each $i \in [\ell]$, let $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i) \in \mathbb{Z}_q^m$.
- By definition, Encrypt samples $\mathbf{s}, \mathbf{e}, \mathbf{K}_{\mathbf{A}}, \mathbf{k}_{\mathbf{p}}$ such that $\|\mathbf{e}\| \le \sqrt{m}\sigma_{\mathsf{LWE}}$ and $\|\mathbf{K}_{\mathbf{A}}\|, \|\mathbf{k}_{\mathbf{p}}\| \le 1$. The ciphertext has the form $\mathsf{ct} = (\pi_{\mathsf{ct}}, \mathsf{ct}_{\mathsf{main}})$ where

$$\mathsf{ct}_{\mathsf{main}} = (\mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3) = \left(\mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal, \mathbf{s}^\intercal(\mathbf{A} + \mathbf{C}) + \mathbf{e}^\intercal \mathbf{K}_{\mathbf{A}}, \mathbf{s}^\intercal \mathbf{p} + \mathbf{e}^\intercal \mathbf{k}_{\mathbf{p}} + \mu \cdot \lfloor q/2 \rfloor\right),$$

  $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$, and $\mathbf{M}$ is as defined in Eq. (B.1). By completeness of $\Pi_{\mathsf{NIZK}}$, we have

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \pi_{\mathsf{ct}}) = 1.$$

- By definition of GetHint, for all $i \in T$,

$$\mathsf{ht}_i = h_i = \mathbf{c}_1^\intercal \mathbf{y}_i + e_i = \mathbf{s}^\intercal \mathbf{B}\mathbf{y}_i + \mathbf{e}^\intercal \mathbf{y}_i + e_i = \mathbf{s}^\intercal \mathbf{t}_i + \mathbf{e}^\intercal \mathbf{y}_i + e_i,$$

  for some $|e_i| < \sqrt{m} \cdot \sigma_{\mathsf{ht}}$.

- Since $P(T) = 1$, there exists an index $\mathsf{ind} \in [\ell]$ where $S_{\mathsf{ind}} \subseteq T$. Let ind be the smallest such index. The Decrypt algorithm computes $\mathbf{z}_{\mathsf{ind}} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}, \mathsf{ind})$. By correctness of the matrix commitment (Theorem 2.10), we have $\mathbf{C}\mathbf{v}_{\mathsf{ind}} = \mathbf{m}_{\mathsf{ind}} - \mathbf{B}\mathbf{z}_{\mathsf{ind}}$, and therefore

$$\mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} = (\mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal)\mathbf{z}_{\mathsf{ind}} = \mathbf{s}^\intercal(\mathbf{m}_{\mathsf{ind}} - \mathbf{C}\mathbf{v}_{\mathsf{ind}}) + \mathbf{e}^\intercal \mathbf{z}_{\mathsf{ind}}.$$

In addition, $\mathbf{m}_{\mathsf{ind}} = \sum_{j \in S_{\mathsf{ind}}} \mathbf{t}_j + \mathbf{p} - \mathbf{A}\mathbf{v}_{\mathsf{ind}}$ by Eq. (B.1). Substituting these expressions into the formula for $\mu'$ computed by Decrypt, we have

$$\begin{aligned}
\mu' &= c_3 - \mathbf{c}_1^\intercal \mathbf{z}_{\mathsf{ind}} - \mathbf{c}_2^\intercal \mathbf{v}_{\mathsf{ind}} + \sum_{i \in S_{\mathsf{ind}}} h_i \\
&= c_3 - \mathbf{s}^\intercal(\mathbf{m}_{\mathsf{ind}} - \mathbf{C}\mathbf{v}_{\mathsf{ind}} + (\mathbf{A}+\mathbf{C})\mathbf{v}_{\mathsf{ind}}) - \mathbf{e}^\intercal(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}}) + \sum_{i \in S_{\mathsf{ind}}} h_i \\
&= c_3 - \mathbf{s}^\intercal\left(\sum_{i \in S_{\mathsf{ind}}} \mathbf{t}_i + \mathbf{p}\right) - \mathbf{e}^\intercal(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}}) + \sum_{i \in S_{\mathsf{ind}}} \mathbf{s}^\intercal \mathbf{t}_i + \mathbf{e}^\intercal \mathbf{y}_i + e_i \\
&= c_3 - \mathbf{s}^\intercal \mathbf{p} - \mathbf{e}^\intercal\left(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}} - \sum_{i \in S_{\mathsf{ind}}} \mathbf{y}_i\right) + \sum_{i \in S_{\mathsf{ind}}} e_i \\
&= \mu \cdot \lfloor q/2 \rfloor - \mathbf{e}^\intercal\left(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}} - \sum_{i \in S_{\mathsf{ind}}} \mathbf{y}_i - \mathbf{k}_{\mathbf{p}}\right) + \sum_{i \in S_{\mathsf{ind}}} e_i.
\end{aligned}$$

Correctness holds as long as $\left|\mathbf{e}^\intercal\left(\mathbf{z}_{\mathsf{ind}} + \mathbf{K}_{\mathbf{A}}\mathbf{v}_{\mathsf{ind}} - \sum_{i \in S_{\mathsf{ind}}} \mathbf{y}_i - \mathbf{k}_{\mathbf{p}}\right) + \sum_{i \in S_{\mathsf{ind}}} e_i\right| < q/4$, so we bound the left side of the expression. By Theorem 2.10 we have

$$\begin{aligned}
\|\mathbf{v}_{\mathsf{ind}}\| &\le O(\|\mathbf{R}\| \cdot m^4 \log q) = O(\sigma_{\mathsf{pp}} \cdot m^{9/2} \log q) \\
\|\mathbf{z}_{\mathsf{ind}}\| &\le O(\|\mathbf{R}\| \cdot m^7 \log q \log N) = O(\sigma_{\mathsf{pp}} \cdot m^{15/2} \log q \log N).
\end{aligned}$$

Correctness follows by setting $q > N \cdot O(\sigma_{\mathsf{pp}}\sigma_{\mathsf{ht}}\sigma_{\mathsf{LWE}} \cdot m^{13} \log q \log N)$. □

**Theorem B.4** (Semi-Policy-and-Very-Query-Selective Security)**.** *Suppose $\Pi_{\mathsf{NIZK}}$ is zero-knowledge and simulation-extractable, and that the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption holds with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$. Suppose also that $n \ge \lambda$, $m \ge 2n \log q$, $q > 2$ is prime, $\sigma_{\mathsf{pp}}, \sigma_{\mathsf{LWE}} \ge \log m$, and $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$. Then, Construction B.2 satisfies semi-policy-and-very-query-selective security in the static security model (Definition B.1).*

*Proof.* Take any efficient adversary $\mathcal{A}$ for the semi-policy-and-very-query-selective security game in the static model (Definition B.1), and suppose $\mathcal{A}$ wins the game with non-negligible advantage $\varepsilon$. Let $L$ be a bound on the number of min-terms in a challenge policy that $\mathcal{A}$ can output. Similar to the proof of Theorem 3.3, we now define a sequence of hybrid experiments, where each hybrid is parameterized by a bit $b \in \{0, 1\}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the semi-policy-and-very-query-selective security game with challenge bit $b \in \{0, 1\}$:
  - **Setup phase:** On input the security parameter $1^\lambda$, the challenger samples

$$\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$$
$$\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m},\ \mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n,\ \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m \times 2m^3}.$$

    If $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$, the challenger sets $\mathbf{R} = \mathbf{0}^{m \times 2m^3}$ instead. Let

$$\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times 4m^5}$$

    and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. The challenger gives the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ to $\mathcal{A}$.

– **Key-generation and challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$ over $N$ variables, a set of honest slots $\mathcal{H} \subseteq [N]$, and a set of public keys $\mathsf{pk}_i = \mathbf{t}_i$ for $i \in [N] \setminus \mathcal{H}$. If $P([N] \setminus \mathcal{H}) = 1$, the challenger outputs 0. Otherwise, for $i \in \mathcal{H}$, the challenger samples $\mathbf{y}_i \xleftarrow{R} \{0,1\}^{4m^5}$ and sets $\mathbf{t}_i = \mathbf{B}\mathbf{y}_i \in \mathbb{Z}_q^n$. The challenger then constructs the challenge ciphertext $\mathsf{ct}^*$ by first sampling

$$\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n\ ,\ \mathbf{e} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{LWE}}}^{4m^5}\ ,\ \mathbf{K}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{4m^5 \times m}\ ,\ \mathbf{k}_{\mathbf{p}} \xleftarrow{R} \{0,1\}^{4m^5}.$$

If $\|\mathbf{e}\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$, the challenger sets $\mathbf{e} = \mathbf{0}^{4m^5}$. Next, the challenger sets $\mathbf{m}_i = \sum_{j \in S_i} \mathbf{t}_j + \mathbf{p} - \mathbf{A}\mathbf{v}_i$ for $i \in [\ell]$, defines $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell]$, and computes the commitment

$$\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \in \mathbb{Z}_q^{n \times m}.$$

Then, the challenger sets $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$, $\mathbf{c}_2^\intercal = \mathbf{s}^\intercal(\mathbf{A} + \mathbf{C}) + \mathbf{e}^\intercal\mathbf{K}_{\mathbf{A}}$, and $c_3 = \mathbf{s}^\intercal\mathbf{p} + \mathbf{e}^\intercal\mathbf{k}_{\mathbf{p}} + b \cdot \lfloor q/2 \rfloor$. Finally, the challenger sets $\mathsf{ct}^*_{\mathsf{main}} = (\mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, c_3)$ and computes the proof

$$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e})),$$

and gives $\mathsf{ct}^* = (\pi^*_{\mathsf{ct}}, \mathsf{ct}^*_{\mathsf{main}})$ along with $\{\mathsf{pk}_i\}_{i \in \mathcal{H}}$ to $\mathcal{A}$.

– **(Post-challenge) hint queries:** When algorithm $\mathcal{A}$ makes a hint query $(\hat{\mathsf{ct}}, i)$ where $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$ for $i \in \mathcal{H}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{c}_3)$. Then it checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1,$$

and replies with $\perp$ if the check fails. If the check passes, the challenger samples $e \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}$, sets $e = 0$ if $|e| > \sqrt{m} \cdot \sigma_{\mathsf{ht}}$, and replies with $\hat{\mathbf{c}}_1^\intercal\mathbf{y}_i + e$.

– **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. Note that if $\mathcal{A}$ aborts early, then the output of the experiment is 0.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$ except the challenger samples $\ell' \xleftarrow{R} [L]$ along with vectors $\mathbf{d}_i^* \xleftarrow{R} \mathbb{Z}_q^n$ for $i \in [\ell']$ at setup time. Additionally, when the adversary outputs the policy $P = (S_1, \ldots, S_\ell)$, the challenger checks if $\ell' = \ell$ and halts with output 0 if not.

- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger replaces the NIZK proof in the challenge ciphertext with a simulated NIZK proof. Specifically, the challenger proceeds as follows:

  – During setup, the challenger samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$.

  – When constructing the challenge ciphertext, after computing $\mathsf{ct}^*_{\mathsf{main}}$, the challenger computes

  $$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}})).$$

- $\mathsf{Hyb}_3^{(b)}$: Same as $\mathsf{Hyb}_2^{(b)}$ except the challenger extracts the encryption randomness for hint queries:

  – When the adversary makes a hint query $(\hat{\mathsf{ct}}, i)$ where $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$ for an index $i \in \mathcal{H}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{c}_3)$. Then, it computes

  $$(\mathbf{s}, \mathbf{e}) = \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}).$$

  If $\|\mathbf{e}\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$, the challenger outputs 0.

- $\mathsf{Hyb}_4^{(b)}$: Same as $\mathsf{Hyb}_3^{(b)}$ except the challenger no longer truncates $e$ when answering hint-generation queries. Specifically, when answering hint queries, the challenger samples $e \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and replies with $\hat{\mathbf{c}}_1^\intercal \mathbf{y}_i + e$ (irrespective of $|e|$). In addition, in the setup phase, the challenger no longer checks if $\|\mathbf{R}\| > \sqrt{m}\sigma_{\mathsf{pp}}$, and when constructing the challenge ciphertext $\mathsf{ct}^*$, it no longer checks if $\|\mathbf{e}\| > \sqrt{m}\sigma_{\mathsf{LWE}}$.

- $\mathsf{Hyb}_5^{(b)}$: Same as $\mathsf{Hyb}_4^{(b)}$ except the challenger now simulates the hints using the extracted randomness. In particular, for a hint query $(\hat{\mathsf{ct}}, i)$ where $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ and $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{c}_3)$, the challenger first validates the proof $\hat{\pi}_{\mathsf{ct}}$. If the proof verifies, then the challenger extracts $(\mathbf{s}, \mathbf{e})$ as in $\mathsf{Hyb}_3^{(b)}$. Then, it samples $e \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and replies with $\mathbf{s}^\intercal \mathbf{t}_i + e$. At this point, the hint queries no longer depend on the secret key $\mathbf{y}_i$.

- $\mathsf{Hyb}_6^{(b)}$: Same as $\mathsf{Hyb}_5^{(b)}$ except an honest key in each min-term is programmed. Specifically, for each $i \in [\ell]$, let $\mathsf{idx}_i \in S_i$ be the smallest index where $\mathsf{idx}_i \in S_i \cap \mathcal{H}$. For $i \in [\ell]$, set $\mathbf{t}_{\mathsf{idx}_i} = \mathbf{d}_i^* - \sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{t}_j + \mathbf{A}\mathbf{v}_i - \mathbf{p}$.

- $\mathsf{Hyb}_7^{(b)}$: Same as $\mathsf{Hyb}_6^{(b)}$ except the challenger samples $\mathbf{k}_{\mathbf{p}} \xleftarrow{R} \{0,1\}^{4m^5}, \mathbf{K}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{4m^5 \times m}$ at setup time, defines $\mathbf{D}^* = [\mathbf{d}_1^* \mid \cdots \mid \mathbf{d}_{\ell'}^*]$, sets $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$, and $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}, \mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}} - \mathbf{C}$. The challenger uses the matrix $\mathbf{C}$ when constructing the challenge ciphertext.

- $\mathsf{Hyb}_8^{(b)}$: Same as $\mathsf{Hyb}_7^{(b)}$ except the challenger samples the component $\mathbf{c}_1 \xleftarrow{R} \mathbb{Z}_q^{4m^5}$, then computes $\mathbf{c}_2^\intercal = \mathbf{c}_1^\intercal \mathbf{K}_{\mathbf{A}}$ and $c_3 = \mathbf{c}_1^\intercal \mathbf{k}_{\mathbf{p}} + b \cdot \lfloor q/2 \rfloor$.

- $\mathsf{Hyb}_9^{(b)}$: Same as $\mathsf{Hyb}_8^{(b)}$ except the challenger samples $\mathbf{p} \xleftarrow{R} \mathbb{Z}_q^n$ and $c_3 \xleftarrow{R} \mathbb{Z}_q$.

We write $\mathsf{Hyb}^{(b)}(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of hybrid $\mathsf{Hyb}^{(b)}$ with adversary $\mathcal{A}$ (and an implicit security parameter $\lambda$). We now bound the difference between the output distributions of each adjacent pair of hybrid experiments.

**Lemma B.5.** *For all $b \in \{0,1\}$, we have* $\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] = L \cdot \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]$.

*Proof.* The adversary's view in both hybrids is identical. Moreover, the challenger in $\mathsf{Hyb}_1^{(b)}$ outputs 1 whenever the challenger in $\mathsf{Hyb}_0^{(b)}$ outputs 1, and moreover $\ell' = \ell$. The challenger in $\mathsf{Hyb}_1^{(b)}$ samples $\ell' \xleftarrow{R} [L]$ independently of the view of the adversary. By assumption, the adversary always chooses a value $\ell \in [L]$. This means

$$\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] \cdot \Pr[\ell' = \ell] = \frac{1}{L} \cdot \Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1].$$

The claim holds. □

**Lemma B.6.** *Suppose $\Pi_{\mathsf{NIZK}}$ satisfies zero-knowledge. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.5. □

**Lemma B.7.** *Suppose $\Pi_{\mathsf{NIZK}}$ satisfies simulation-sound extractability. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.6. □

**Lemma B.8.** *Suppose $m \geq \lambda$. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proofs of Lemma 3.7 (for removing the norm check on $e$) and Lemma 3.10 (for removing the norm check on $\mathbf{R}$ and $\mathbf{e}$). □

**Lemma B.9.** *Suppose $\sigma_{\mathsf{ht}} > \lambda^{\omega(1)} \cdot m^{11/2}\sigma_{\mathsf{LWE}}$. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_5^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.8. □

**Lemma B.10.** *Suppose $n \geq \lambda, m \geq 2n\log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} \geq \log m$. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_5^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* The hybrids are indistinguishable by Corollary 2.9 by setting $\ell = 2m^2$ and $s = \sigma_{\mathsf{pp}}$. In particular, we have the following for any fixed $\mathbf{e} \in \mathbb{Z}_q^{4m^5}$ and $i \in \mathcal{H}$:

$$\left\{(\mathbf{W}, \mathbf{R}, \mathbf{B}\mathbf{y}_i) : \begin{array}{c}\mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3} \\ \mathbf{y}_i \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}\end{array}\right\} \text{and} \left\{(\mathbf{W}, \mathbf{R}, \mathbf{d}_i) : \begin{array}{c}\mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n\times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{pp}}}^{m\times 2m^3} \\ \mathbf{d}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n}\end{array}\right\},$$

where $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G}$. Fix any $i \in [\ell]$ and let $\mathsf{idx}_i \in \mathcal{H}$ be the first honest key in the $i^{\text{th}}$ min-term. We argue that the left and right distributions above correspond to the marginal distribution of $(\mathbf{W}, \mathbf{R}, \mathbf{t}_{\mathsf{idx}_i})$ in $\mathsf{Hyb}_5^{(b)}$ and $\mathsf{Hyb}_6^{(b)}$, respectively, conditioned on all other variables in the experiment.

- In $\mathsf{Hyb}_5^{(b)}$, the challenger samples $\mathbf{y}_{\mathsf{idx}_i} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ and sets $\mathbf{t}_{\mathsf{idx}_i} = \mathbf{B}\mathbf{y}_{\mathsf{idx}_i}$. Hence the joint distribution of $(\mathbf{W}, \mathbf{R}, \mathbf{t}_{\mathsf{idx}_i})$ in $\mathsf{Hyb}_5^{(b)}$ matches the left distribution above.
- In $\mathsf{Hyb}_6^{(b)}$, the challenger samples $\mathbf{d}_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ at setup time (independently of all other variables) and sets

  $$\mathbf{t}_{\mathsf{idx}_i} = \mathbf{d}_i^* - \textstyle\sum_{j\in S_i\setminus\{\mathsf{idx}_i\}} \mathbf{t}_j + \mathbf{A}\mathbf{v}_i - \mathbf{p}.$$

  Since $\mathbf{d}_i^*$ is uniform in $\mathbb{Z}_q^n$ and independent of the remaining quantities, $\mathbf{t}_{\mathsf{idx}_i}$ is uniform in $\mathbb{Z}_q^n$ and independent of $(\mathbf{W}, \mathbf{R})$. Hence the joint distribution of $(\mathbf{W}, \mathbf{R}, \mathbf{t}_{\mathsf{idx}_i})$ in $\mathsf{Hyb}_6^{(b)}$ matches the right distribution above.

By Corollary 2.9, the two distributions are statistically indistinguishable. Finally, since $\mathsf{idx}_1, \ldots, \mathsf{idx}_\ell$ are all distinct (since $S_1, \ldots, S_\ell$ are mutually disjoint), the claim now follows by a hybrid argument over all $\ell = \mathsf{poly}(\lambda)$ min-terms. □

**Lemma B.11.** *Suppose $n \geq \lambda, m \geq 2n\log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} \geq \log m$. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_6^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_7^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* This follows by a similar argument as in the proof of Lemma 3.16. Concretely, we first observe that the matrix $\mathbf{C}$ used in the challenge ciphertext is identically distributed in $\mathsf{Hyb}_6^{(b)}$ and $\mathsf{Hyb}_7^{(b)}$. In $\mathsf{Hyb}_6^{(b)}$, the challenger computes $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ where $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_\ell]$ and $\mathbf{m}_i = \sum_{j \in S_i} \mathbf{t}_j + \mathbf{p} - \mathbf{A}\mathbf{v}_i$. After the programming step in $\mathsf{Hyb}_6^{(b)}$, for each $i \in [\ell]$,

$$\begin{aligned}\mathbf{m}_i &= \textstyle\sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{t}_j + \mathbf{t}_{\mathsf{idx}_i} + \mathbf{p} - \mathbf{A}\mathbf{v}_i \\ &= \textstyle\sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{t}_j + \left(\mathbf{d}_i^* - \sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{t}_j + \mathbf{A}\mathbf{v}_i - \mathbf{p}\right) + \mathbf{p} - \mathbf{A}\mathbf{v}_i \\ &= \mathbf{d}_i^*.\end{aligned}$$

Thus $\mathbf{M} = \mathbf{D}^*$ and $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$ in $\mathsf{Hyb}_6^{(b)}$, which exactly matches the value of $\mathbf{C}$ sampled in $\mathsf{Hyb}_7^{(b)}$. Hence the only remaining differences between $\mathsf{Hyb}_6^{(b)}$ and $\mathsf{Hyb}_7^{(b)}$ are the distributions of $\mathbf{p}$ and $\mathbf{A}$:

- In $\mathsf{Hyb}_6^{(b)}$, $\mathbf{p} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and $\mathbf{A} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$ are sampled uniformly and independently of $(\mathbf{W}, \mathbf{R}, \mathbf{C}, \mathbf{k}_{\mathbf{p}}, \mathbf{K}_{\mathbf{A}})$.
- In $\mathsf{Hyb}_7^{(b)}$, $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}$ and $\mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}} - \mathbf{C}$, where $\mathbf{k}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ and $\mathbf{K}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$.

Let $\mathbf{K} = [\mathbf{k}_{\mathbf{p}} \mid \mathbf{K}_{\mathbf{A}}] \in \{0,1\}^{4m^5 \times (m+1)}$. By construction, the joint distribution of $(\mathbf{p}, \mathbf{A} + \mathbf{C})$ is uniform over $\mathbb{Z}_q^{n \times (m+1)}$ in $\mathsf{Hyb}_6^{(b)}$ and equals $\mathbf{B}\mathbf{K}$ in $\mathsf{Hyb}_7^{(b)}$. By Corollary 2.9 the distribution

$$\{(\mathbf{W}, \mathbf{R}, \mathbf{B}\mathbf{K}, \mathbf{e}^\mathsf{T}\mathbf{K}) : \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z}, \sigma_{\mathsf{pp}}}^{m \times 2m^3},\ \mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times (m+1)}\}$$

is statistically indistinguishable from

$$\{(\mathbf{W}, \mathbf{R}, \mathbf{U}, \mathbf{e}^\mathsf{T}\mathbf{K}) : \mathbf{W} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3},\ \mathbf{R} \leftarrow D_{\mathbb{Z}, \sigma_{\mathsf{pp}}}^{m \times 2m^3},\ \mathbf{U} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times (m+1)},\ \mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times (m+1)}\}.$$

The first distribution corresponds to the distribution of the public parameters and challenge ciphertext components in $\mathsf{Hyb}_7^{(b)}$ while the second corresponds to those in $\mathsf{Hyb}_6^{(b)}$. We conclude that $\mathsf{Hyb}_6^{(b)}$ and $\mathsf{Hyb}_7^{(b)}$ are statistically indistinguishable. □

**Lemma B.12.** *Suppose the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$ holds. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_7^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* This follows by a similar argument as in the proof of Lemma 3.17. Suppose $|\Pr[\mathsf{Hyb}_7^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8^{(b)}(\mathcal{A}) = 1]| = \delta$ for some non-negligible $\delta$. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathbf{W}, \mathbf{R}, \mathbf{c}^\mathsf{T})$ from the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE challenger and starts running $\mathcal{A}(1^\lambda)$. Algorithm $\mathcal{B}$ sets $\mathbf{B} = \mathbf{W}(\mathbf{I}_{2m^2} \otimes \mathbf{R}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\mathsf{T} \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times 4m^5}$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$. Then algorithm $\mathcal{B}$ samples

  $$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda),\ \ell' \xleftarrow{\mathrm{R}} [L],\ \mathbf{d}_i^* \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n \text{ for } i \in [\ell'],$$
  $$\mathbf{K}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m},\ \mathbf{k}_{\mathbf{p}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}.$$

  Algorithm $\mathcal{B}$ defines $\mathbf{D}^* = [\mathbf{d}_1^* \mid \cdots \mid \mathbf{d}_{\ell'}^*]$ and sets $\mathbf{C} = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{D}^*)$, $\mathbf{A} = \mathbf{B}\mathbf{K}_{\mathbf{A}} - \mathbf{C}$, and $\mathbf{p} = \mathbf{B}\mathbf{k}_{\mathbf{p}}$. It gives the public parameters $\mathsf{pp} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}, \mathbf{A}, \mathbf{p})$ to $\mathcal{A}$.

- **Key-generation and challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P = (S_1, \ldots, S_\ell) \in \mathcal{P}_{\mathsf{DNF}}$, a set of honest slots $\mathcal{H} \subseteq [N]$, and a set of public keys $\mathsf{pk}_i = \mathbf{t}_i$ for $i \in [N] \setminus \mathcal{H}$. If $P([N] \setminus \mathcal{H}) = 1$ or $\ell \neq \ell'$, algorithm $\mathcal{B}$ outputs 0. For each $i \in [\ell]$, let $\mathsf{idx}_i \in [N]$ be the smallest index where $\mathsf{idx}_i \in S_i \cap \mathcal{H}$. Then for $j \in \mathcal{H} \setminus \{\mathsf{idx}_1, \ldots, \mathsf{idx}_\ell\}$, algorithm $\mathcal{B}$ samples $\mathbf{y}_j \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ and sets $\mathbf{t}_j = \mathbf{B}\mathbf{y}_j$. Finally, for each $i \in [\ell]$, algorithm $\mathcal{B}$ sets $\mathbf{v}_i = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, \ell, i)$ and

$$\mathbf{t}_{\mathsf{idx}_i} = \mathbf{d}_i^* - \textstyle\sum_{j \in S_i \setminus \{\mathsf{idx}_i\}} \mathbf{t}_j + \mathbf{A}\mathbf{v}_i - \mathbf{p}.$$

  Algorithm $\mathcal{B}$ then sets $\mathsf{ct}^*_{\mathsf{main}} = (\mathbf{c}^\mathsf{T},\ \mathbf{c}^\mathsf{T}\mathbf{K}_\mathbf{A},\ \mathbf{c}^\mathsf{T}\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor)$ and computes

$$\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}})).$$

  Finally, algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with the public keys $\{\mathsf{pk}_i\}_{i \in \mathcal{H}}$ together with the challenge ciphertext $\mathsf{ct}^* = (\pi^*_{\mathsf{ct}}, \mathsf{ct}^*_{\mathsf{main}})$.

- **(Post-challenge) hint queries:** When algorithm $\mathcal{A}$ makes a hint query $(\hat{\mathsf{ct}}, i)$ where $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$ for $i \in \mathcal{H}$, algorithm $\mathcal{B}$ parses $\hat{\mathsf{ct}} = (\hat{\pi}_{\mathsf{ct}}, \hat{\mathsf{ct}}_{\mathsf{main}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\mathbf{c}}_1^\mathsf{T}, \hat{\mathbf{c}}_2^\mathsf{T}, \hat{c}_3)$, and checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1.$$

  If the check fails, algorithm $\mathcal{B}$ replies with $\perp$. Otherwise, algorithm $\mathcal{B}$ computes

$$(\mathbf{s}, \mathbf{e}) = \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{m} \cdot \sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}).$$

  If $\|\mathbf{e}\| > \sqrt{m} \cdot \sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\mathsf{T} \neq \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T}$, algorithm $\mathcal{B}$ outputs 0. Otherwise, algorithm $\mathcal{B}$ samples $e \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{ht}}}$ and replies with $\mathbf{s}^\mathsf{T}\mathbf{t}_i + e$.

- **Output phase:** At the end of the game, if algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, algorithm $\mathcal{B}$ also outputs $b'$.

By construction, Algorithm $\mathcal{B}$ efficiently simulates the setup phase, the public keys, and the hint queries exactly as in $\mathsf{Hyb}_7^{(b)}$ or $\mathsf{Hyb}_8^{(b)}$. Consider the distribution of the challenge ciphertext components in the two distributions. By construction of $\mathbf{A}, \mathbf{C}$, and $\mathbf{p}$, if $\mathbf{c}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T}$ where $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and $\mathbf{e} \leftarrow D_{\mathbb{Z},\sigma_{\mathsf{LWE}}}^{4m^5}$, then

$$\mathbf{c}^\mathsf{T}\mathbf{K}_\mathbf{A} = \mathbf{s}^\mathsf{T}\mathbf{B}\mathbf{K}_\mathbf{A} + \mathbf{e}^\mathsf{T}\mathbf{K}_\mathbf{A} = \mathbf{s}^\mathsf{T}(\mathbf{A} + \mathbf{C}) + \mathbf{e}^\mathsf{T}\mathbf{K}_\mathbf{A}$$
$$\mathbf{c}^\mathsf{T}\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor = \mathbf{s}^\mathsf{T}\mathbf{B}\mathbf{k}_\mathbf{p} + \mathbf{e}^\mathsf{T}\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor = \mathbf{s}^\mathsf{T}\mathbf{p} + \mathbf{e}^\mathsf{T}\mathbf{k}_\mathbf{p} + b \cdot \lfloor q/2 \rfloor,$$

so algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_7^{(b)}$. If $\mathbf{c}^\mathsf{T} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{4m^5}$, then algorithm $\mathcal{B}$ constructs the challenge ciphertext exactly as described in $\mathsf{Hyb}_8^{(b)}$. In this case, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_8^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE with advantage $\delta$. □

**Lemma B.13.** *Suppose $n \geq \lambda, m \geq 2n \log q, q > 2$ is prime, and $\sigma_{\mathsf{pp}} \geq \log m$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$,*

$$|\Pr[\mathsf{Hyb}_8^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

*Proof.* Follows via a similar argument as in the proof of Lemma 3.18. □

Clearly $\Pr[\mathsf{Hyb}_9^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_9^{(1)}(\mathcal{A}) = 1]$, since the challenger's behavior in $\mathsf{Hyb}_9^{(b)}(\mathcal{A})$ is independent of $b \in \{0,1\}$. Since $L = \mathsf{poly}(\lambda)$, security now follows by Lemmas B.5 to B.13 and a standard hybrid argument. □

**Parameter instantiation.** We now provide one possible way to set the lattice parameters in Construction B.2 so as to satisfy the conditions in Theorems B.3 and B.4. Let $\lambda$ be the security parameter, $N \leq 2^\lambda$ be the number of variables, and $\varepsilon \in (0,1)$ be a constant. We set

$$\begin{aligned} n &= (\lambda \log N)^{1/\varepsilon} \cdot \mathsf{poly}(\log \lambda, \log N) \\ m &= 2n \log q \\ \sigma_{\mathsf{LWE}} &= \mathsf{poly}(n) \\ \sigma_{\mathsf{pp}} &= \log m \\ \sigma_{\mathsf{ht}} &= 2^\lambda \cdot m^{11/2} \cdot \sigma_{\mathsf{LWE}} \\ q &= 2^\lambda \cdot \mathsf{poly}(n, N). \end{aligned}$$

Note that $q = 2^{O(n^\varepsilon)}$, so we require a sub-exponential modulus-to-noise ratio. Combined with the NIZK scheme from Fact 2.2, we obtain the following corollary:

**Corollary B.14** (Distributed Monotone-Policy Encryption for DNFs in the Plain Model with Short Ciphertexts)**.** *Let $\lambda$ be a security parameter. Then, under the decomposed LWE assumption and a sub-exponential modulus-to-noise ratio (in the plain model), there exists a semi-policy-and-very-query-selective distributed monotone-policy encryption scheme in the static model for policies that can be computed by DNF formulas (with up to $N = 2^\lambda$ variables). The size of the public parameters and the size of the ciphertext are $\mathsf{poly}(\lambda)$ and the size of each user's public key is $K \cdot \mathsf{poly}(\lambda)$, where $K$ is a bound on the number of times each variable can appear in the DNF computing a policy. Moreover, the scheme has a transparent setup.*