Silent Threshold Encryption from Lattices

Jeffrey Champion
UT Austin
jchampion@utexas.edu

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

Shota Yamada*
AIST
yamada-shota@aist.go.jp

Abstract

Silent threshold encryption is a generalization of threshold encryption where the public encryption key associated with a group of users is a deterministic function of their individual public keys. The main efficiency requirement is that the ciphertext size should be sublinear in (and ideally, independent of) the size of the decryption quorum N. Existing constructions of silent threshold encryption for arbitrary threshold policies have either relied on bilinear maps or on heavyweight tools such as witness encryption and indistinguishability obfuscation. Recently, several works have shown how to support *constant* thresholds from the decomposed learning with errors (LWE) problem.

In this work, we show how to construct a silent threshold encryption scheme from the decomposed LWE assumption with ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$, where N is the total number of users, T is the threshold, and λ is the security parameter. Our scheme achieves non-trivial succinctness for all thresholds $T = N^{\varepsilon}$ for any constant $\varepsilon < 1$. More generally, our scheme extends beyond threshold policies to any monotone policy family that has a succinct (computational) secret sharing scheme; the ciphertext in this case scales with the maximum number of corrupted shares.

The core building block in our work is a new bounded-collusion registered functional encryption (FE) scheme with succinct ciphertexts. Specifically, for N users and a collusion bound Q, we obtain a registered FE scheme that supports depth-d Boolean circuits on ℓ-bit inputs with ciphertext size $Q \cdot \tilde{O}(d) + \ell \cdot \mathsf{poly}(\lambda, d, \log N)$. Security relies on the decomposed LWE assumption in the random oracle model. Previously, bounded-collusion registered FE for general circuits was known only from bilinear maps, evasive LWE, or indistinguishability obfuscation.

1 Introduction

Threshold cryptography [Des87, Fra89, DF89, DDFY94] is a standard technique to protect cryptographic keys by splitting them into multiple independent shares, each held by a different party. In a T-out-of-N threshold public-key encryption scheme, anyone can encrypt a message under the public key such that it can only be decrypted with the cooperation of at least T decrypters. For the scheme to be non-trivial, we require the size of the ciphertext to be *sublinear* in the number of decrypters N. Without this efficiency requirement, there is a trivial construction where each member of the decryption committee holds an (independent) public/secret key-pair and the ciphertext simply consists of vanilla encryptions of a T-out-of-N Shamir secret sharing [Sha79] of the message under the N public keys (where the i^{th} share is encrypted under the public key of decrypter i).

Traditional threshold encryption assumes some kind of centralized setup, where either a trusted dealer generates and issues the shares of the decryption keys to the different parties, or the parties themselves engage in an interactive distributed key-generation protocol to generate the shares. The former requires a trusted party while the latter requires parties to interact and be aware of one another. Moreover, in many conventional settings, the setup needs to be repeated whenever users join or leave the system.

Silent threshold encryption. Silent threshold encryption [GKPW24] (also known as registered threshold encryption [BLM+24]) is a generalization of threshold cryptography that is more suitable for *distributed* settings. Specifically, in this setting, we assume there is a common reference string (CRS). Each user can generate their own public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$ with respect to the CRS. Then, given an arbitrary collection of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ along

* Part of this work was done while visiting UT Austin.

with a threshold T, there is a public, deterministic algorithm that aggregates these public keys together into an *aggregated encryption key* mpk for this group of users. The aggregated public key now functions as the public key for a T-out-of-N threshold encryption system, and each party's secret key sk_i functions as their decryption share. Specifically, anyone can encrypt a message μ to the aggregated public key mpk and obtain a ciphertext ct. Given the ciphertext ct, any decrypter can use their secret key sk_i to decrypt and output a partial decryption σ_i. Given T partial decryptions $\{\sigma_i\}$, there is a public aggregation algorithm to recover the message μ.

Compared to the traditional "top-down" model of threshold encryption where a trusted dealer generates shares of a decryption key and issues the shares to the users, we can view silent threshold encryption as a "bottom-up" model where users choose their own public/private keys, and the overall master public key is derived as a deterministic function of the users' public keys. The advantage of this model is that it removes the need for a trusted dealer (or interactive protocol) to generate shares. Moreover, when users join or leave the system, it is straightforward to derive a new public key that reflects the current composition of the decryption quorum.

Constructions of silent threshold encryption. In the last few years, a number of works have studied constructions of silent threshold encryption from assumptions like indistinguishability obfuscation [RSY21, ADM+24, DJWW25] and from bilinear maps [GKPW24, BLM+24, BCF+25, WW26, GWWW26]. If we restrict our attention to lattice-based constructions, the results are far more limited. Here, existing constructions can only support constant threshold policies [CW26a, AGY26a, CW26b] or the more restricted case of threshold $T = 1$ [CW24, CHW25, AMR25, WW25, CW26b, GY26c, AMR26, GY26a] (which is known as distributed broadcast encryption [WQZD10, BZ14]). All of these aforementioned lattice-based schemes rely on either the succinct learning with errors (succinct LWE) assumption [Wee24] or the decomposed LWE assumption [AMR25]. The work of [HSW25] considers a relaxed version of silent threshold encryption with "soft thresholds" where there is an $\Omega(N)$ gap between the threshold needed to decrypt and that needed for security (i.e., the "ramp threshold" setting), and moreover, correctness only holds for most, but not all, decryption quorums of a sufficient size. This relaxed notion can be realized just from public-key encryption.

1.1 Our Results

In this work, we give the first lattice-based silent threshold encryption scheme from the decomposed LWE assumption in the random oracle model where the ciphertext size is $\tilde{O}(T) + \mathrm{poly}(\lambda, \log N)$, N is the total number of users, and T is the threshold. Notably, our scheme achieves non-trivial succinctness (i.e., ciphertext size $o(N)$) for all thresholds $T = N^{\epsilon}$ for any constant $\epsilon < 1$. This is the first lattice-based silent threshold encryption scheme that supports super-constant thresholds. Our scheme supports any number of users N, and the size of the public parameters scale polylogarithmically with N. Moreover, the scheme has a transparent setup procedure. We summarize the properties of our construction in the following theorem:

Theorem 1.1 (Silent Threshold Encryption from Lattices, Informal)**.** *Let λ be a security parameter. Under the decomposed LWE assumption (with a sub-exponential modulus-to-noise ratio), there exists a silent threshold encryption scheme in the random oracle model that supports decryption quorums with up to N users with the following properties:*

- **Common reference string:** *The CRS is a uniform random string of size* $\mathrm{poly}(\lambda, \log N)$*. Since the CRS is uniform, the scheme has a* transparent *setup.*
- **Public key size:** *The size of each user's public key is* $\mathrm{poly}(\lambda, \log N)$*.*
- **Aggregated public key size:** *The size of the aggregated public key is* $\mathrm{poly}(\lambda, \log N)$*.*
- **Ciphertext size:** *A ciphertext with threshold T has size $\tilde{O}(T) + \mathrm{poly}(\lambda, \log N)$, where $\tilde{O}(\cdot)$ suppresses constant and polylogarithmic factors.*

Moreover, our scheme generalizes to support dynamic *thresholds where the threshold T is specified at encryption time (with the same properties as above).*

Technical building block: registered FE. The main technical tool underlying our construction is a bounded-collusion registered functional encryption scheme with succinct ciphertexts. Specifically, in registered functional

encryption (FE) [DPY24, FFM+23], users can generate their own public/private key-pair $(\mathsf{pk}, \mathsf{sk})$ and then register their public key pk along with an associated function $f : \{0,1\}^\ell \to \{0,1\}$ with a key curator. The key curator can then aggregate the list of public keys and their associated functions $(\mathsf{pk}_1, f_1), \ldots, (\mathsf{pk}_N, f_N)$ to obtain an aggregated master public key mpk. Anyone can encrypt an input $\mathbf{x} \in \{0,1\}^\ell$ to the master public key mpk. A user with secret key sk_i (and associated function f_i) can decrypt the ciphertext and learn $f_i(\mathbf{x})$. In this work, we show how to construct a registered FE scheme in the bounded-collusion model where security holds if the adversary controls at most Q keys in the system, and we allow the ciphertext size to grow with the collusion bound Q. We say the registered FE scheme has succinct ciphertexts (in the sense of [GKP+13]) if the size of the ciphertext only scales with the *depth* of the functions f, and *not* the size of f. We summarize our instantiation in the following theorem:

Theorem 1.2 (Bounded-Collusion Registered FE from Lattices, Informal)**.** *Let λ be a security parameter and $\mathcal{F}$ be a family of functions on inputs of length $\ell = \ell(\lambda)$ that can be computed by a Boolean circuit of depth $d = d(\lambda)$ and size $s = s(\lambda)$. Then, under the decomposed LWE assumption (with a sub-exponential modulus-to-noise ratio), there exists a bounded-collusion registered FE scheme for $\mathcal{F}$ in the random oracle model that supports N users with the following properties:*

- **Common reference string:** *The CRS is a uniform random string of size* $\mathrm{poly}(\lambda, d, \log \ell, \log s, \log N)$. *Since the CRS is uniform, the scheme has a* transparent *setup.*
- **Public key size:** *The size of each user's public key is* $\mathrm{poly}(\lambda, d, \log \ell, \log s, \log N)$.
- **Aggregated public key size:** *To achieve security with collusion bound Q, the size of the aggregated public key is* $\mathrm{poly}(\lambda, d, \log \ell, \log s, \log N)$.
- **Ciphertext size:** *A ciphertext with collusion bound Q has size $Q \cdot \tilde{O}(d) + \ell \cdot \mathrm{poly}(\lambda, d, \log N)$, where $\tilde{O}(\cdot)$ suppresses constant and polylogarithmic factors.*

The work of [BLM+24] shows a direct implication between (bounded-collusion) registered FE and silent threshold encryption. Prior to this work, all constructions of (bounded-collusion) registered FE relied on the *private-coin* evasive LWE assumption [ZCHZ24, PST25] (as well as the random oracle model in the case of [ZCHZ24]). Unfortunately, recent attacks have raised substantial concerns over the plausibility of the evasive LWE assumption [VWW22, BÜW24, BDJ+25, DJM+25, AMYY25, HJL25, HHY26]. Moreover, in the case of [ZCHZ24], the underlying bounded-collusion registered FE does not consider security in the presence of adversarially-chosen public keys, and as such, it is insufficient for silent threshold encryption. Even if [ZCHZ24] could be extended to support adversarially-chosen public keys, the resulting silent threshold encryption scheme would still have asymptotically-longer ciphertexts than our scheme (see Appendix A). Theorem 1.2 gives the first (bounded-collusion) registered FE scheme for general circuits from a falsifiable lattice assumption (decomposed LWE) in the random oracle model.

We also note that bounded-collusion registered FE is considerably more challenging to construct than vanilla bounded-collusion functional encryption. For instance, plain bounded-collusion FE is known just from the minimal assumption of public-key encryption [SS10, GVW12]. In contrast, even *1-bounded* registered FE (i.e., a scheme where security holds against an adversary that is allowed to corrupt just a single user) already implies (distributed) broadcast encryption. Existing lattice-based constructions of broadcast encryption (either centralized or distributed) all rely on succinct or decomposed LWE [Wee24, CW24, CHW25, AMR25, WW25, CW26b, GY26c, AMR26, GY26a], and it would be a major breakthrough to realize this primitive from plain LWE (let alone public-key encryption).

An application to collusion-resistant traitor tracing. The work of [BLM+24] also shows how to construct a registered traitor tracing scheme from any registered FE scheme that supports quadratic functions. Substituting Theorem 1.2 into their compiler yields a (bounded-collusion) registered traitor tracing scheme from the decomposed LWE assumption in the random oracle model.

Concurrent work. In a pair of concurrent and independent works, Afshar, Goyal, and Yadugiri [AGY26b, GY26b] also show how to construct silent threshold encryption from decomposed LWE in the random oracle model. They do so by extending the approach from [AGY26a, CW26a] to threshold policies using a form of scaled secret sharing. We describe the main differences between our silent threshold encryption scheme and theirs:

- **Ciphertext size.** The use of scaled secret sharing in [AGY26b, GY26b] leads to large reconstruction coefficients. To instantiate their construction for N users and threshold T, they need to set the decomposed LWE modulus q in their scheme to have bit length $\min\{T^2, N-T\}$. When instantiated with decomposed LWE (and recalling that $n > \log q$ for security), this leads to ciphertext size at least $T^{22} \cdot \mathsf{poly}(\lambda)$ or $(N-T)^{11} \cdot \mathsf{poly}(\lambda)$ and a public key size of at least $T^8 \cdot \mathsf{poly}(\lambda)$ or $(N-T)^4 \cdot \mathsf{poly}(\lambda)$. The ciphertext size can be improved to $T^6 \cdot \mathsf{poly}(\lambda)$ or $(N-T)^3 \cdot \mathsf{poly}(\lambda)$ by instead using succinct LWE, at the cost of needing structured public parameters whose size scales with $\mathsf{poly}(T)$. In this work, we focus on the small threshold case and obtain ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$, which achieves non-trivial succinctness whenever $T = N^\varepsilon$ for any constant $\varepsilon < 1$. Moreover, we do so with transparent, $\mathsf{poly}(\lambda, \log N)$-sized public parameters and $\mathsf{poly}(\lambda, \log N)$-sized public keys.
- **Public key aggregation.** In the original definition of silent threshold encryption [GKPW24], there is a *deterministic* aggregation algorithm that takes as input the users' public keys and aggregates them into a master public key for the quorum. This is the notion we consider in this work. The works of [AGY26b, GY26b] consider a randomized aggregation algorithm. It is plausible that they can derandomize the aggregation process using the random oracle, but this is not explicitly proven in their work. Having a deterministic aggregation process is important to avoid the possibility of a malicious aggregator using "bad" aggregation randomness to subvert the scheme. This is a central requirement in trustless cryptographic systems.
- **Security.** Both our work and [GY26b] consider a form of static security. In both works, the adversary is not allowed to adaptively corrupt users in the security game. In our work, we impose the additional restriction where the adversary must declare the indices for the corrupted slots at setup time, whereas the work of [GY26b] allows the adversary to adaptively choose the indices of the corrupted users. Both works allow the adversary complete freedom to choose the public keys for the corrupted users (as a function of the honest users' public keys). Note that both works consider a stronger security notion than [AGY26b].

1.2 Technical Overview

In this section, we give an overview of our approach for building silent threshold encryption. We start by explaining the limitations of the distributed monotone-policy encryption scheme for DNF policies from [CW26a, AGY26a]. In a distributed monotone-policy encryption scheme, users choose their own public/secret keys. Later, anyone can publicly encrypt a message with respect to an arbitrary set of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ together with a monotone access policy P over those users. Any subset of users that satisfies the policy can decrypt the ciphertext and the ciphertext should computationally hide the message from any unauthorized set of users. Silent threshold encryption is a special case of distributed monotone-policy encryption for threshold policies. (Technically, silent threshold encryption requires an additional algorithm to aggregate public keys, but this is a second-order bit.)

Noisy shares and their limitation. The basic strategy from the prior works of [CW26a, AGY26a] is to take a vector $\mathbf{p} \in \mathbb{Z}_q^n$ (in the public parameters) and split it into shares $\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_N \in \mathbb{Z}_q^n$. The encrypter then constructs a *succinct* ciphertext that allows user i to recover the noisy inner product $\mathbf{s}^\intercal \hat{\mathbf{p}}_i$. Here, $\mathbf{s} \in \mathbb{Z}_q^n$ is an LWE secret (sampled at encryption time) and we write $\mathbf{s}^\intercal \hat{\mathbf{p}}_i$ to denote $\mathbf{s}^\intercal \hat{\mathbf{p}}_i + e_i$ for some $\widetilde{\text{small}}$ error $e_i \in \mathbb{Z}$. The ciphertext additionally contains $\mathbf{s}^\intercal \mathbf{p} + \mu \cdot \lfloor q/2 \rfloor$, where $\mu \in \widetilde{\{0,1\}}$ is the message. If the secret sharing scheme is linear (i.e., for any authorized set of $\widetilde{\text{users}}$ S, there exist coefficients $\omega_i \in \mathbb{Z}$ such that $\sum_{i \in S} \omega_i \hat{\mathbf{p}}_i = \mathbf{p}$), and moreover, the reconstruction coefficients ω_i are small, then the authorized users can compute $\mathbf{s}^\intercal \mathbf{p} \approx \sum_{i \in S} \omega_i \mathbf{s}^\intercal \hat{\mathbf{p}}_i$ and recover the message. For this template to work, it is critical that the reconstruction coefficients ω_i are $\widetilde{\text{small}}$ relative to the modulus q. Unfortunately, this is incompatible with standard Shamir secret sharing (over $\mathbb{Z}_q$) since the Lagrange interpolation coefficients are large.

While there are many techniques to reduce the size of the interpolation coefficients (cf. Section 1.3), instantiating the [CW26a, AGY26a] template with existing approaches leads to an insecure construction. This is because the security analysis in [CW26a, AGY26a] additionally requires that any set of unauthorized shares $\hat{\mathbf{p}}_i$ be linearly independent. If the unauthorized shares are linearly dependent, then there is a zeroizing attack on the scheme. Unfortunately, existing approaches for threshold secret sharing do not simultaneously have small reconstruction coefficients and satisfy the linear independence property. Thus, in this work, we take a different approach and instead ask whether the decrypters can obtain *noise-free* shares. If so, then the users would be able to use standard Shamir interpolation to decrypt.

Noise-free shares via registered functional encryption. If we consider the problem from the perspective of building a mechanism to allow users to obtain noise-free shares, then a natural approach is to leverage some type of functional encryption [BW07, BSW11, O'N10]. Previously, the work of Branco et al. [BLM+24] showed such an implication between registered linear functional encryption and silent threshold encryption. We start by describing their template as applied to any linear secret sharing scheme. Specifically, we model a linear secret sharing scheme over N parties by a share-generating matrix $\mathbf{M} \in \mathbb{Z}_p^{N \times T}$. To share a message $\mu \in \mathbb{Z}_p$, sample $r_2, \ldots, r_T \xleftarrow{R} \mathbb{Z}_p$ and consider the vector $\mathbf{r} = [\mu, r_2, \ldots, r_T]^\intercal$. The share for party i is $\mathbf{m}_i^\intercal \mathbf{r} \in \mathbb{Z}_p$ where $\mathbf{m}_i \in \mathbb{Z}_p^T$ is the i^{th} row of $\mathbf{M}$. Given the secret shares for any authorized group of users, the message μ is a linear function of the parties' shares.

- Each user in the system generates a public/private key-pair $(\mathsf{pk}, \mathsf{sk})$ for the registered linear FE scheme. This serves as their public/private key-pair for the silent threshold encryption scheme.
- Given a collection of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_N)$ and a share-generating matrix $\mathbf{M} \in \mathbb{Z}_p^{N \times T}$, the aggregation algorithm runs the aggregation algorithm for the registered linear FE scheme with $\mathbf{m}_i^\intercal \in \mathbb{Z}_p^T$ as the linear function associated with party i. The aggregated public key is then the public key associated with the quorum.
- To encrypt a message $\mu \in \mathbb{Z}_p$, the encrypter samples $r_2, \ldots, r_T \xleftarrow{R} \mathbb{Z}_p$ and encrypts the vector $\mathbf{r} = [\mu, r_2, \ldots, r_T]^\intercal$. Let ct be the resulting ciphertext.
- Each party i can decrypt ct with their secret key sk_i to learn their share $\mathbf{m}_i^\intercal \mathbf{r}$. Any subset of parties that satisfies the policy can now combine their shares to recover the message μ.

The work of [BLM+24] shows how to construct a registered linear FE scheme from pairings. Combined with standard Shamir secret sharing, they obtain a silent threshold encryption scheme from pairings with ciphertext size $T \cdot \mathsf{poly}(\lambda)$.

This work: bounded-collusion registered FE for circuits. Currently, the only constructions of registered FE [ZCHZ24, PST25] rely on the evasive LWE assumption. This remains the case even if we consider a relaxation to the bounded-collusion setting. In this work, we show how to construct a bounded-collusion registered FE scheme that supports arbitrary circuits, where the ciphertext encrypting an input $\mathbf{x}$ has size $Q \cdot \tilde{O}(d) + |\mathbf{x}| \cdot \mathsf{poly}(\lambda, d)$; here, Q is the collusion bound and d is an a priori bound on the depth of the circuits we support. Observe that this is sufficient to instantiate the general template from [BLM+24]. For the case of threshold encryption, the collusion bound Q coincides with the threshold T, so in combination with Shamir secret sharing, we obtain a silent threshold encryption scheme with ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$. Strictly speaking, we instantiate the above compiler with Shamir secret sharing where the share randomness is derived from a pseudorandom function (PRF). In this case, the ciphertext just needs to contain an encryption of a short PRF key (as opposed to the share randomness which would have size $\Omega(T \log N)$).

Background: matrix commitments and homomorphic evaluation. Our construction of bounded-collusion registered FE for general circuits combines techniques from the registered attribute-based encryption (ABE) scheme from [WW25] and the dual-use technique for constructing predicate encryption from [BTVW17]. We start here by recalling some basic preliminaries on lattice-based homomorphic evaluation underlying these schemes that we will also use:

- **Matrix commitments:** A key building block in the [WW25] registered ABE scheme is the matrix commitment scheme introduced in [Wee25]. Specifically, given a collection of public parameters $\mathsf{pp}_{\mathsf{com}}$, there exists an efficient algorithm that takes as input a matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$ and outputs a commitment $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$ and a low-norm opening $\mathbf{Z}_{\mathbf{M}} \in \mathbb{Z}^{4m^5 \times L}$ where
$$\mathbf{C}\mathbf{V}_L = \mathbf{M} - \mathbf{B}\mathbf{Z}_{\mathbf{M}}, \tag{1.1}$$
where $\mathbf{B} \in \mathbb{Z}_q^{n \times 4m^5}$ and $\mathbf{V}_L \in \mathbb{Z}^{m \times L}$ are matrices determined by the public parameters (moreover, $\mathbf{V}_L$ is low-norm). In addition, under the succinct LWE [Wee24] or the decomposed LWE assumption [AMR25], the LWE assumption holds with respect to the matrix $\mathbf{B}$ given $\mathsf{pp}_{\mathsf{com}}$ (i.e., $(\mathbf{B}, \mathbf{s}^\intercal \mathbf{B})$ is pseudorandom given $\mathsf{pp}_{\mathsf{com}}$).
- **Homomorphic evaluation:** The construction also relies on the lattice-based homomorphic evaluation machinery from [GSW13, BGG+14], and specifically, on the extension to matrix-valued functions from [BTVW17].

Namely, for every matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$, every function $f\colon \{0,1\}^\ell \to \mathbb{Z}_q^{n \times t}$ that can be computed by a Boolean circuit of depth d, and every input $\mathbf{x} \in \{0,1\}^\ell$, there is a low-norm matrix $\mathbf{H}_{\mathbf{A},f,\mathbf{x}} \in \mathbb{Z}^{\ell m \times t}$ where

$$(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A},f,\mathbf{x}} = \mathbf{A}_f - f(\mathbf{x}) \in \mathbb{Z}_q^{n \times t}. \tag{1.2}$$

Here, $\mathbf{G}$ denotes the gadget matrix [MP12] and $\mathbf{A}_f \in \mathbb{Z}_q^{n \times t}$ is a matrix that only depends on $\mathbf{A}$ and f (Theorem 2.15). The norm of $\mathbf{H}_{\mathbf{A},f,\mathbf{x}}$ scales as $m^{O(d)}$. When $f\colon \{0,1\}^\ell \to \{0,1\}$ is a Boolean function, we recover the usual relation $(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G})\mathbf{H}_{\mathbf{A},f,\mathbf{x}} = \mathbf{A}_f - f(\mathbf{x}) \cdot \mathbf{G}$ by taking $t = m$ and considering the function $\mathbf{x} \mapsto f(\mathbf{x}) \cdot \mathbf{G}$.

From registered ABE to registered FE. Next, we recall the basic structure of the registered (key-policy) ABE scheme from [CHW25, WW25]. In key-policy registered ABE, users are associated with policies and ciphertexts are associated with attributes. Decryption is successful whenever the ciphertext attribute satisfies the user's policy. In the [CHW25, WW25] constructions, a ciphertext encrypting a message $\mu \in \{0,1\}$ with attribute $\mathbf{x} \in \{0,1\}^\ell$ has the form

$$\mathsf{ct} = \big(\underbrace{\mathbf{s}^\intercal \mathbf{B}}\ ,\ \underbrace{\mathbf{s}^\intercal(\mathbf{C}_0 + \mathbf{C}_1)}\ ,\ \underbrace{\mathbf{s}^\intercal(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G})}\ ,\ \underbrace{\mathbf{s}^\intercal \mathbf{p}} + \mu \cdot \lfloor q/2 \rfloor\big).$$

Here, $\mathbf{B}$ is the matrix defined by the public parameters of the matrix commitment scheme, $\mathbf{C}_1$ is a matrix commitment to the public keys of the individual users and their respective policies, $\mathbf{C}_0$ is a term used to blind $\mathbf{C}_1$, and $\mathbf{A}, \mathbf{p}$ are fixed matrices (in the public parameters) used to encode the attribute and the message, respectively. Specifically, $\mathbf{C}_1$ commits to the matrix whose i^{th} column is $\mathbf{p} + \mathbf{B}\mathbf{r}_i - \mathbf{A}_{f_i}\mathbf{G}^{-1}(\mathbf{d}_i) - \mathbf{C}_0\mathbf{v}_i$, where $\mathbf{v}_i$ is the i^{th} column of the matrix $\mathbf{V}_L$ from Eq. (1.1), and $\mathbf{d}_i \in \mathbb{Z}_q^n$ is also a fixed vector from the public parameters. A user associated with a policy f_i where $f_i(\mathbf{x}) = 0$ can decrypt as follows:

- Using the homomorphic evaluation relation and the fact that $f_i(\mathbf{x}) = 0$, the user computes

$$\underbrace{\mathbf{s}^\intercal(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G})} \cdot \mathbf{H}_{\mathbf{A},f_i,\mathbf{x}}\mathbf{G}^{-1}(\mathbf{d}_i) \approx \mathbf{s}^\intercal\big(\mathbf{A}_{f_i} - f_i(\mathbf{x}) \cdot \mathbf{G}\big) \cdot \mathbf{G}^{-1}(\mathbf{d}_i) = \mathbf{s}^\intercal \mathbf{A}_{f_i}\mathbf{G}^{-1}(\mathbf{d}_i).$$

- Using the matrix commitment relation from Eq. (1.1), the user takes their opening $\mathbf{z}_i$ for $\mathbf{C}_1$ and computes

$$\begin{aligned}\underbrace{\mathbf{s}^\intercal(\mathbf{C}_0 + \mathbf{C}_1)}\mathbf{v}_i + \underbrace{\mathbf{s}^\intercal \mathbf{B}}\mathbf{z}_i &\approx \mathbf{s}^\intercal(\mathbf{C}_0\mathbf{v}_i + \mathbf{p} + \mathbf{B}\mathbf{r}_i - \mathbf{A}_{f_i}\mathbf{G}^{-1}(\mathbf{d}_i) - \mathbf{C}_0\mathbf{v}_i - \mathbf{B}\mathbf{z}_i) + \mathbf{s}^\intercal \mathbf{B}\mathbf{z}_i \\ &= \mathbf{s}^\intercal(\mathbf{p} + \mathbf{B}\mathbf{r}_i - \mathbf{A}_{f_i}\mathbf{G}^{-1}(\mathbf{d}_i)).\end{aligned}$$

Combining the above relations and using their secret key $\mathbf{r}_i$ to compute $\mathbf{s}^\intercal \mathbf{B}\mathbf{r}_i \approx \underbrace{\mathbf{s}^\intercal \mathbf{B}}\mathbf{r}_i$, the user recovers a noisy version of $\mathbf{s}^\intercal \mathbf{p}$, and from there, the message. To extend these ideas to registered FE (for functions with single-bit outputs), we follow the template used to build predicate encryption [GVW15, BTVW17]:

- In the registered ABE scheme, the user homomorphically evaluates their function f_i on a *public* input $\mathbf{x}$. In registered FE, we need to support a computation on a *secret* input $\mathbf{x}$. Following [GVW15, BTVW17], we can support computation on hidden inputs by replacing the input $\mathbf{x}$ with a (leveled) homomorphic encryption of $\mathbf{x}$. Let $\mathbf{\Psi}$ be the encryption of $\mathbf{x}$. In the ciphertext, we replace the encoding of $\mathbf{x}$ (i.e., the encoding $\mathbf{s}^\intercal(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G})$) with an encoding of $\mathbf{\Psi}$ (and also give out $\mathbf{\Psi}$ as part of the ciphertext). Instead of associating the function f_i with user i, we now associate the function $\hat{f}_i$ that homomorphically evaluates f_i (on an encryption of $\mathbf{x}$). We can view $\hat{f}_i$ as outputting a *vector* in $\mathbb{Z}_q^n$.
- Specifically, let $\boldsymbol{\varphi}$ be the binary representation of the ciphertext $\mathbf{\Psi}$ and we replace the encoding of $\mathbf{x}$ with an encoding of $\boldsymbol{\varphi}$. Then, the decrypter can apply the matrix-valued homomorphic evaluation procedure and compute

$$\underbrace{\mathbf{s}^\intercal(\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G})} \cdot \mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} \approx \mathbf{s}^\intercal\big(\mathbf{a}_{\hat{f}_i} - \hat{f}_i(\boldsymbol{\varphi})\big). \tag{1.3}$$

When $\mathbf{\Psi}$ is an encryption under the Gentry-Sahai-Waters homomorphic encryption scheme [GSW13] where the secret key is $\mathbf{s}$, the dual-use technique of [BTVW17] shows how to compute an element $\underline{\psi} \in \mathbb{Z}_q$ from $\mathbf{\Psi}$ and f_i such that

$$\underline{\psi} - \mathbf{s}^\intercal \hat{f}_i(\boldsymbol{\varphi}) \approx f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor\,. \tag{1.4}$$

Combining Eqs. (1.3) and (1.4), the decrypter can now compute

$$\underbrace{\mathbf{s}^\intercal(\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G})}\cdot \mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} + \underline{\psi} \approx \mathbf{s}^\intercal\big(\mathbf{a}_{\hat{f}_i} - \hat{f}_i(\boldsymbol{\varphi})\big) + \underline{\psi} \approx \mathbf{s}^\intercal \mathbf{a}_{\hat{f}_i} + f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor.$$

Now, by taking $\mathbf{C}_1$ to be a commitment to a matrix whose i^{th} column is $\mathbf{B}\mathbf{r}_i + \mathbf{a}_{\hat{f}_i} - \mathbf{C}_0\mathbf{v}_i$, we can use the same strategy as before to allow the i^{th} user to recover a noisy version of $f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor$. The user now rounds to learn the *noise-free* value $f_i(\mathbf{x})$, as required.

- The above strategy suffices for correctness, but is insecure. Specifically, in functional encryption, we require that the decrypter only learn $f_i(\mathbf{x})$ and nothing more about the input $\mathbf{x}$. Unfortunately, the above template reveals more than just $f_i(\mathbf{x})$; the user also learns information about the noise associated with the ciphertext $\boldsymbol{\Psi}$. As such, we cannot argue that the ciphertext $\boldsymbol{\Psi}$ hides the message $\mathbf{x}$. This leakage is precisely the reason why the predicate encryption schemes in [GVW15, BTVW17] only provide *weak* attribute-hiding and not strong attribute-hiding. Thus, to lift to registered FE, we need to introduce an additional masking term to the ciphertext to hide the leakage on the noise for the homomorphic encryption scheme. Roughly speaking, for each key that the adversary has, we need to introduce one unit of randomness to *smudge* away this leakage. This limits our scheme to bounded-collusion security. In particular, security against an adversary controlling Q keys requires a ciphertext whose size grows with Q.

We are now ready to give our full construction. For ease of exposition, we first describe the construction with a *randomized* aggregation algorithm and where the collusion bound is fixed to $Q = 1$. Afterwards, we describe how we derandomize the aggregation algorithm using the random oracle and how to support larger collusion bounds using cover-free sets.

- **Public parameters:** The public parameters for the scheme consist of the public parameters $\mathsf{pp}_{\mathsf{com}}$ for the matrix commitment scheme. Let $\mathbf{B} \in \mathbb{Z}_q^{n \times 4m^5}$ be the matrix associated with $\mathsf{pp}_{\mathsf{com}}$.

- **Key generation:** To generate a public key, the user samples $\mathbf{K} \xleftarrow{\text{R}} \{0,1\}^{4m^5 \times m}$. The user's secret key is $\mathsf{sk} = \mathbf{K}$ and their public key is $\mathbf{P} = \mathbf{B}\mathbf{K} \in \mathbb{Z}_q^{n \times m}$. Here, our approach differs from [CHW25, WW25] in that we take the user public key to be a matrix rather than a vector. This allows us to leverage the deferred public-key sampling technique from [CW26b], which will be important for proving *simulation security* of our registered FE scheme.

- **Aggregation:** Given a collection of public keys $\mathbf{P}_1, \ldots, \mathbf{P}_N \in \mathbb{Z}_q^{n \times m}$, together with the set of associated functions $f_1, \ldots, f_N \colon \{0,1\}^\ell \to \{0,1\}$, the aggregation algorithm proceeds as follows:

  - **Sample an encoding matrix.** Let $\hat{\ell}$ be the length of a ciphertext (for the [GSW13] encryption scheme) for encrypting an input of length ℓ. Sample an encoding matrix $\mathbf{A} \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times \hat{\ell} m}$.
  - **Derive a fresh public key for each user.** For each $i \in [N]$, sample a low-norm vector $\mathbf{u}_i \in \mathbb{Z}^m$ and let $\mathbf{d}_i = \mathbf{P}_i\mathbf{u}_i \in \mathbb{Z}_q^n$ be the effective public key for user i.
  - **Sample a masking vector.** Sample a masking vector $\mathbf{w} \xleftarrow{\text{R}} \mathbb{Z}_q^n$. In the security analysis, this vector will be used to both program in the value of the function evaluation for corrupted users and to introduce a smudging term for smudging the [GSW13] decryption error. We refer to the overview of the security analysis for more details on the critical role $\mathbf{w}$ plays.
  - **Commit to the public keys.** Sample an offset $\mathbf{C}_0 \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$. Then, for each $i \in [N]$, let $\hat{f}_i$ be the function that homomorphically evaluates f_i on an encrypted ℓ-bit input and let $\mathbf{a}_{\hat{f}_i} \in \mathbb{Z}_q^n$ be the vector associated with the function $\hat{f}_i$ with respect to $\mathbf{A}$ (see Eq. (1.3)). Let $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$ be the matrix whose i^{th} column is

    $$\mathbf{m}_i = \mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \mathbf{w} - \mathbf{C}_0\mathbf{v}_i \in \mathbb{Z}_q^n, \tag{1.5}$$

    where $\mathbf{v}_i \in \mathbb{Z}^m$ is the i^{th} column of the matrix $\mathbf{V}_N$ from Eq. (1.1). Let $\mathbf{C}_1 \in \mathbb{Z}_q^{n \times m}$ be a commitment to $\mathbf{M}$ and let $\mathbf{z}_i$ be the i^{th} column of the associated opening. Finally, let $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1 \in \mathbb{Z}_q^{n \times m}$.

  The aggregated master public key is $\mathsf{mpk} = (\mathbf{A}, \mathbf{C}, \mathbf{w})$ and the helper decryption key for user i is $\mathsf{hsk}_i = (\mathbf{v}_i, \mathbf{z}_i, \mathbf{u}_i, f_i)$.

- **Encryption:** To encrypt an input $\mathbf{x} \in \{0,1\}^\ell$, the encrypter samples $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, computes a [GSW13] encryption $\mathbf{\Psi}$ of $\mathbf{x}$ with respect to the secret key $\mathbf{s}$ (and public matrix $\mathbf{B}$). Concretely, the encryption algorithm samples $\mathbf{R}_{\mathbf{\Psi}} \xleftarrow{R} \{0,1\}^{4m^5 \times \ell m}$ and sets

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{B} \\ \underline{\mathbf{s}^\mathsf{T}\mathbf{B}} \end{bmatrix} \mathbf{R}_{\mathbf{\Psi}} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G} \in \mathbb{Z}_q^{(n+1)\times \ell m}.$$

  Let $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ be the binary representation of $\mathbf{\Psi}$. Then, it outputs the ciphertext

$$\mathsf{ct} = \left(\mathbf{\Psi}\ ,\ \underline{\mathbf{s}^\mathsf{T}\mathbf{B}}\ ,\ \underline{\mathbf{s}^\mathsf{T}\mathbf{C}}\ ,\ \underline{\mathbf{s}^\mathsf{T}(\mathbf{A} - \boldsymbol{\varphi}^\mathsf{T} \otimes \mathbf{G})}\ ,\ \underline{\mathbf{s}^\mathsf{T}\mathbf{w}}\right). \tag{1.6}$$

  Here, the component $\underline{\mathbf{s}^\mathsf{T}\mathbf{w}}$ will include a *large* smudging error which will be used to smudge out the noise in the [GSW13] ciphertext. By construction, the size of the ciphertext is $\mathsf{poly}(\lambda, \ell, d)$, where d is a bound on the depth of the Boolean circuit computing f_i.

- **Decryption:** To decrypt, user i uses their secret key $\mathbf{K}_i$ together with the helper decryption key $\mathsf{hsk}_i = (\mathbf{v}_i, \mathbf{z}_i, \mathbf{u}_i, f_i)$. By Eq. (1.1) and the fact that $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1$, we have

$$\mathbf{C}\mathbf{v}_i = \mathbf{C}_0\mathbf{v}_i + \mathbf{C}_1\mathbf{v}_i = \mathbf{C}_0\mathbf{v}_i + \mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \mathbf{w} - \mathbf{C}_0\mathbf{v}_i - \mathbf{B}\mathbf{z}_i = \mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \mathbf{w} - \mathbf{B}\mathbf{z}_i.$$

  Since $\mathbf{K}_i$, $\mathbf{v}_i$, $\mathbf{z}_i$ are all low norm and $\mathbf{P}_i = \mathbf{B}\mathbf{K}_i$, this means

$$\underline{\mathbf{s}^\mathsf{T}\mathbf{C}} \cdot \mathbf{v}_i - \underline{\mathbf{s}^\mathsf{T}\mathbf{B}} \cdot \mathbf{K}_i\mathbf{u}_i - \underline{\mathbf{s}^\mathsf{T}\mathbf{w}} + \underline{\mathbf{s}^\mathsf{T}\mathbf{B}} \cdot \mathbf{z}_i \approx \mathbf{s}^\mathsf{T}\mathbf{a}_{\hat{f}_i}.$$

  Next, the user computes a noisy encoding of $\mathbf{s}^\mathsf{T}(\mathbf{a}_{\hat{f}_i} - \hat{f}_i(\boldsymbol{\varphi}))$ using Eq. (1.3). Taken together, the user learns $\underline{\mathbf{s}^\mathsf{T}\hat{f}_i(\boldsymbol{\varphi})}$. Following Eq. (1.4), the user computes $\underline{\psi}$ from $\mathbf{\Psi}$ and f_i, and computes $\underline{\psi} - \underline{\mathbf{s}^\mathsf{T}\hat{f}_i(\boldsymbol{\varphi})} \approx f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor$.

Security analysis. We now provide a brief sketch of the security analysis for the above construction. In this work, we focus on *simulation security*, where we require that the ciphertext (encrypting an input $\mathbf{x}$) can be simulated given only $\{f_i(\mathbf{x})\}_{i \in C}$, where $C \subseteq [N]$ denotes the set of indices that the adversary corrupted and f_i is the function associated with the corrupted slot $i \in C$. Moreover, we consider selective security in the static model; this means the adversary both declares the input $\mathbf{x}$ as well as the corrupted indices C at the beginning of the security game. However, the adversary can still adaptively choose the public keys (and the functions f_i) for the corrupted indices.

For the construction described above, the collusion bound is $Q = 1$, so the adversary declares a singleton set $C = \{i\}$ as the set of corrupted indices. The adversary also declares the challenge input $\mathbf{x}$. We now must construct an efficient simulator that simulates the keys for the honest users together with the challenge ciphertext given only $y_i = f_i(\mathbf{x})$, where f_i is the function the adversary selects for user i. Unlike registered ABE (and related notions), we cannot simply argue that the ciphertext is a pseudorandom string; this is because the adversary can decrypt and learn $f_i(\mathbf{x})$. In particular, this means we must be able to "program" the value of $f_i(\mathbf{x})$ into the ciphertext itself. This is also the reason why the ciphertext size in (simulation-secure) registered FE necessarily scales with the collusion bound Q (see also Remark 3.8).

We now describe our general strategy. Throughout this work, we work in the registered-key model [RY07] where we require the adversary to provide the key-generation randomness associated with any public key it chooses in the security game. Such a scheme is also said to satisfy semi-malicious security. It is straightforward to compile a semi-malicious scheme to one with full security using non-interactive zero-knowledge proofs of knowledge [RY07, LWW25] (see Remark 3.7). We start by describing an alternative (but statistically indistinguishable) way to sample the public parameters, the keys for the honest users, and the components of the challenge ciphertext. Note that this procedure will still require knowledge of the input $\mathbf{x}$, but will be a stepping stone to the eventual simulation procedure that will only require knowledge of $f_i(\mathbf{x})$:

- We set $\mathsf{pp}_{\mathsf{com}}$ exactly as in the real scheme. Let $\mathbf{B}$ be the matrix associated with $\mathsf{pp}_{\mathsf{com}}$.
- Following the deferred public key sampling technique from [CW26b], we sample the public keys $\mathbf{P}_j \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor td_j for all honest users $j \neq i$. By standard lattice trapdoor sampling techniques [Ajt96, GPV08, MP12], the marginal distribution of $\mathbf{P}_j$ is statistically close to uniform.

- We sample a random matrix $\mathbf{M} \xleftarrow{R} \mathbb{Z}_q^{n\times N}$ and compute $\mathbf{C}_1 \in \mathbb{Z}_q^{n\times m}$ to be a commitment to $\mathbf{M}$. Next, we sample $\mathbf{R}_C \xleftarrow{R} \{0,1\}^{4m^5\times m}$ and define $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_C - \mathbf{C}_1$. For this choice of variables, observe that $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1 = \mathbf{B}\mathbf{R}_C$. By the leftover hash lemma,[1] $\mathbf{C}_0$ and $\mathbf{C}_1$ are statistically close to their distribution in the real scheme.
- When constructing the challenge ciphertext, we sample the [GSW13] ciphertext $\mathbf{\Psi}$ exactly as in the real scheme. Namely, we sample $\mathbf{R}_\Psi \xleftarrow{R} \{0,1\}^{4m^5\times \ell m}$ and set

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{B} \\ \underset{\sim}{\mathbf{s}^\intercal\mathbf{B}} \end{bmatrix} \mathbf{R}_\Psi + \mathbf{x}^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{(n+1)\times \ell m}.$$

  Let $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ be the binary representation of $\mathbf{\Psi}$. Then, sample $\mathbf{R}_A \xleftarrow{R} \{0,1\}^{4m^5\times \hat{\ell} m}$ and let $\mathbf{A} = \mathbf{B}\mathbf{R}_A + \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}$. By the leftover hash lemma, these components are statistically indistinguishable from those in the real scheme.
- When constructing the challenge ciphertext, we sample the randomizing terms as follows:
  - For the corrupted user i, the simulator samples a low-norm $\mathbf{u}_i \in \mathbb{Z}^m$ and sets

$$\mathbf{w} = \mathbf{m}_i - \mathbf{P}_i\mathbf{u}_i - \mathbf{a}_{\hat{f}_i} + \mathbf{C}_0\mathbf{v}_i \in \mathbb{Z}_q^n. \tag{1.7}$$

  - For the honest users $j \neq i$, the simulator uses the trapdoor td_j to sample a low-norm $\mathbf{u}_j \in \mathbb{Z}^m$ such that $\mathbf{P}_j\mathbf{u}_j = \mathbf{m}_j - \mathbf{a}_{\hat{f}_j} - \mathbf{w} + \mathbf{C}_0\mathbf{v}_j \in \mathbb{Z}_q^n$.

  For this choice of $\mathbf{u}_1, \ldots, \mathbf{u}_N, \mathbf{w}$, observe that the columns of $\mathbf{M}$ are precisely the ones that would be computed according to Eq. (1.5). This is the equivocation strategy developed in [GY26c, CW26b]. The critical point is we use $\mathbf{w}$ to program the column of $\mathbf{M}$ corresponding to the corrupted user and we use $\mathbf{u}_j$ to program the columns of $\mathbf{M}$ corresponding to honest users.
- Let $\mathbf{K}_i \in \{0,1\}^{4m^5\times m}$ be the secret key associated with the adversary's chosen public key $\mathbf{P}_i = \mathbf{B}\mathbf{K}_i$. This is available since we are working in the registered-key model. From Eq. (1.7), we now have

$$\begin{aligned}
\mathbf{w} &= \mathbf{m}_i - \mathbf{P}_i\mathbf{u}_i - \mathbf{a}_{\hat{f}_i} + \mathbf{C}_0\mathbf{v}_i \\
&= \mathbf{C}_1\mathbf{v}_i + \mathbf{B}\mathbf{z}_i - \mathbf{P}_i\mathbf{u}_i - \mathbf{a}_{\hat{f}_i} + \mathbf{C}_0\mathbf{v}_i && \text{since } \mathbf{C}_1\mathbf{v}_i = \mathbf{m}_i - \mathbf{B}\mathbf{z}_i \\
&= \mathbf{B}\mathbf{R}_C\mathbf{v}_i + \mathbf{B}\mathbf{z}_i - \mathbf{B}\mathbf{K}_i\mathbf{u}_i - \mathbf{a}_{\hat{f}_i} && \text{since } \mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1 = \mathbf{B}\mathbf{R}_C \text{ and } \mathbf{P}_i = \mathbf{B}\mathbf{K}_i \\
&= \mathbf{B}(\mathbf{z}_i + \mathbf{R}_C\mathbf{v}_i - \mathbf{K}_i\mathbf{u}_i) - (\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}) \cdot \mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} - \hat{f}_i(\boldsymbol{\varphi}) && \text{by Eq. (1.2)} \\
&= \mathbf{B}(\mathbf{z}_i + \mathbf{R}_C\mathbf{v}_i - \mathbf{K}_i\mathbf{u}_i - \mathbf{R}_A\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}}) - \hat{f}_i(\boldsymbol{\varphi}) && \text{since } \mathbf{A} = \mathbf{B}\mathbf{R}_A + \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}.
\end{aligned}$$

  Now compute $\underset{\sim}{\psi} \in \mathbb{Z}_q$ from $\mathbf{\Psi}$ and f_i (where $\underset{\sim}{\psi} \in \mathbb{Z}_q$ is the component in Eq. (1.4)). We now simulate the term

$$\begin{aligned}
\underset{\sim}{\mathbf{s}^\intercal\mathbf{w}} &\approx \underset{\sim}{\mathbf{s}^\intercal\mathbf{B}} \cdot \left(\mathbf{z}_i + \mathbf{R}_C\mathbf{v}_i - \mathbf{K}_i\mathbf{u}_i - \mathbf{R}_A\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}}\right) - \mathbf{s}^\intercal\hat{f}_i(\boldsymbol{\varphi}) \\
&\approx \underset{\sim}{\mathbf{s}^\intercal\mathbf{B}} \cdot \left(\mathbf{z}_i + \mathbf{R}_C\mathbf{v}_i - \mathbf{K}_i\mathbf{u}_i - \mathbf{R}_A\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}}\right) + f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor - \underset{\sim}{\psi} && \text{by Eq. (1.4).}
\end{aligned}$$

  To argue that this has the correct distribution, we will require that the noise in $\underset{\sim}{\mathbf{s}^\intercal\mathbf{w}}$ is sufficient to smudge the noise in $\underset{\sim}{\mathbf{s}^\intercal\mathbf{B}} \cdot \left(\mathbf{z}_i + \mathbf{R}_C\mathbf{v}_i - \mathbf{K}_i\mathbf{u}_i - \mathbf{R}_A\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}}\right)$. In particular, this means that the noise in $\underset{\sim}{\mathbf{s}^\intercal\mathbf{w}}$ smudges out the noise accumulated by [GSW13] evaluation. This is the reason why the decrypted ciphertext does not leak information about the [GSW13] noise (and thus, ensures that we can simulate the ciphertext given only the value $f_i(\mathbf{x})$).

There are two important properties to observe in the above distribution:

[1] Technically, we are invoking the leftover hash lemma on the non-random matrix $\mathbf{B}$. This is nonetheless valid due to the specific structure of $\mathbf{B}$ (see Lemma 2.18).

- First, the public parameters, the honest users' keys, and the components of the challenge ciphertext can also be simulated given knowledge of $\mathsf{pp}_{\mathsf{com}}$ (which determines $\mathbf{B}$) and $\underset{\sim}{\mathbf{s}}^\intercal\mathbf{B}$. Thus, by the decomposed LWE assumption, the above distribution is computationally indistinguishable from one where we substitute $\underset{\sim}{\mathbf{s}}^\intercal\mathbf{B}$ with a uniform random vector $\mathbf{v}^\intercal \xleftarrow{R} \mathbb{Z}_q^{4m^5}$. After this change, the [GSW13] ciphertext $\boldsymbol{\Psi}$ would be constructed as

$$\boldsymbol{\Psi} = \begin{bmatrix} \mathbf{B} \\ \mathbf{v}^\intercal \end{bmatrix} \mathbf{R}_{\boldsymbol{\Psi}} + \mathbf{x}^\intercal \otimes \mathbf{G}. \tag{1.8}$$

- Second, when simulating the challenge ciphertext, the only component that depends on the input $\mathbf{x}$ is the [GSW13] ciphertext $\boldsymbol{\Psi}$. The remaining components ($\mathbf{A}$ and $\underset{\sim}{\mathbf{s}}^\intercal\underset{\sim}{\mathbf{w}}$) can be simulated given only $\boldsymbol{\Psi}$, f_i, and $f_i(\mathbf{x})$. Moreover, nothing else in the experiment depends on the encryption randomness $\mathbf{R}_{\boldsymbol{\Psi}}$. Thus, we can appeal to the leftover hash lemma and argue that $\boldsymbol{\Psi}$ in Eq. (1.8) is statistically close to uniform. In this distribution, all of the components can be simulated given just $f_i(\mathbf{x})$, and simulation security holds.

We give the formal construction and analysis in Section 3.

Bounded collusion via cover-free sets. The basic template described above allows the adversary to corrupt a single slot $i \in [N]$. The simulator then programs the function value $f_i(\mathbf{x})$ into the vector $\mathbf{w}$. Suppose now that the adversary can corrupt up to Q users $C \subseteq [N]$. To carry out the same proof strategy, we need to program the function evaluations $f_i(\mathbf{x})$ for each corrupted index $i \in C$. To do so, we need at least Q vectors $\mathbf{w}_1, \ldots, \mathbf{w}_Q$, one associated with each corrupted slot. Thus, a natural approach is to replace the global vector $\mathbf{w}$ in Eq. (1.5) with a slot-specific vector $\mathbf{w}_i \in \mathbb{Z}_q^n$. Namely, at aggregation time, the aggregation algorithm would commit to the matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times N}$ whose i^{th} column is

$$\mathbf{m}_i = \mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \mathbf{w}_i - \mathbf{C}_0\mathbf{v}_i \in \mathbb{Z}_q^n.$$

For correctness, the ciphertext would then need to include $\underset{\sim}{\mathbf{s}}^\intercal\underset{\sim}{\mathbf{w}_i}$ for each $i \in [N]$. This is a problem because the ciphertext size is now linear in the total number of users. Instead, we need a solution that only relies on a small number of vectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$ where $M = \tilde{O}(Q)$. In this case, the ciphertext size scales with the collusion bound rather than the total number of users. We can achieve this by associating a set $S_i \subseteq [M]$ with each index $i \in [N]$. Then, at aggregation time, the aggregation algorithm modifies the vector $\mathbf{m}_i$ to be

$$\mathbf{m}_i = \mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \sum_{j \in S_i} \mathbf{w}_j - \mathbf{C}_0\mathbf{v}_i \in \mathbb{Z}_q^n.$$

The property we require now is that for every $i \in C$, there exists an index $s_i \in S_i$ such that $s_i \notin S_j$ for all $j \in C \setminus \{i\}$. Then, in the security analysis, we can simply program the function value $f_i(\mathbf{x})$ using $\mathbf{w}_{s_i}$, exactly as before.

The remaining question is how to choose the parameter M and the sets $S_1, \ldots, S_N$ so this property holds (with high probability). One approach is to choose the sets $S_1, \ldots, S_N \subseteq [M]$ to be from a Q-cover-free family [KS64, EFF85]. This ensures that every S_i is guaranteed to not be contained in the union of any collection of Q other sets. Then, we can always associate a unique index s_i with S_i. The drawback of using cover-free sets is that constructing a Q-cover-free family of sets requires choosing $M = \tilde{\Omega}(Q^2)$ [DR82]. In this case, the ciphertext size in our construction grows quadratically with the collusion bound. In this work, since we are already limited to static security where the adversary declares the set of corrupted users C at the beginning of the security game, we can do better. In particular, since the set of corrupted indices C is fixed in advance, we only require a set family $S_1, \ldots, S_N \subseteq [M]$ where for all $i \in C$, it is the case that $S_i \nsubseteq \bigcup_{j \in C \setminus \{i\}} S_j$. This is a "local" cover-free property and can be satisfied using standard Q-wise independent hash functions with $M = \tilde{O}(Q)$. This in turn yields a Q-bounded-collusion registered FE scheme where the ciphertext size scales quasi-linearly with the collusion bound.

Finally, we note that combinatorial approaches (using cover-free sets) to lift from 1-bounded security to Q-bounded security are well known in the setting of (centralized) functional encryption [GVW12, AR17, AV19]. In particular, the work of [AV19] shows how to obtain a bounded-collusion centralized functional encryption scheme for Boolean circuits with ciphertexts of size $Q \cdot \mathsf{poly}(\lambda, s)$, where s is a bound on the size of the Boolean circuits the scheme supports. In this work, we are working in the registered setting and moreover, for our application to silent threshold encryption, we additionally require the size of the ciphertext to only grow with a bound on the depth of the circuit rather than the size. As such, the [AV19] approach is not immediately applicable. While it is possible to adapt the existing compiler (see also Appendix A), we believe it is simpler to directly integrate the cover-free sets into the algebraic structure of our scheme.

Derandomizing aggregation. In registration-based cryptography [GHMR18, HLWW23] (and silent threshold encryption [GKPW24]), aggregation should be a public deterministic algorithm. While the aggregation algorithm we described above is randomized, it is straightforward to derandomize it by simply deriving the aggregation randomness from a public hash function evaluated on the inputs to the aggregation algorithm (i.e., the user's public keys and their associated functions). In the security analysis, we model the hash function as a random oracle. This is similar to the approach taken in several previous works that also relied on the random oracle to derandomize the aggregation procedure [CHW25, WW25, CW26a, AGY26a, CW26b]. The security reduction will in turn program the outputs of the random oracle to implement the simulation strategy described above.

One technical caveat that comes up in this work is our reduction strategy needs the ability to program all *potential* random oracle queries that correspond to a set of valid public keys and policies. This is because we are considering a simulation-based notion of security. Previous works on registered ABE and distributed monotone-policy encryption all considered an indistinguishability-based notion of security and there, it was sufficient to have the reduction "guess" the query to the random oracle that corresponds to the adversary's challenge query, and take a $1/Q_{\mathrm{ro}}$ loss in the security reduction, where Q_{ro} is the number of random-oracle queries the adversary makes. Such a strategy is no longer viable if we consider a simulation-based security definition because the $1/Q_{\mathrm{ro}}$ loss in the reduction translates into a simulator that is successful with probability $\approx 1/Q_{\mathrm{ro}}$, which does not provide a meaningful security guarantee. To get around this, we need the ability to program the random oracle on *every* random oracle query that could potentially be the challenge query. To do this, our proof strategy critically relies on the deferred public-key sampling technique from [CW26b]. Namely, since the honest users' effective public key $\mathbf{d}_i = \mathbf{P}_i\mathbf{u}_i$ is refreshed at each query, the reduction has the ability to equivocate on each query. In contrast, the earlier techniques of [CHW25, WW25] fix the honest users' public keys at key-generation time, which limits the reduction to only being able to program a single query.

One-round distributed decryption. Finally, for our application to silent threshold encryption, we require a *distributed* decryption procedure. Specifically, in silent threshold encryption, given a ciphertext, each user should be able to publish a decryption hint that only depends on their individual secret key and the ciphertext (and nothing else about other parties in the decryption quorum). Then, there is a separate recover algorithm that takes a collection of decryption hints (from a sufficiently-large quorum of users) and uses the hints to recover the underlying message. Translated to the setting of registered functional encryption, this boils down to decomposing the decryption algorithm into two steps (see also Definition 3.9):

- **Hint generation:** Given only the ciphertext ct and the user's secret key sk (but *not* the helper decryption key hsk), the hint-generation algorithm should output a decryption hint ht. This procedure will map to the local decryption algorithm in the application to silent threshold encryption.
- **Message recovery:** Given the ciphertext ct, the decryption hint ht, and the helper decryption key hsk, the message-recovery algorithm recovers the message. Notably, this step does not depend on the user's secret key. In fact, it only depends on *public* quantities (recall that a user's helper decryption key is output by the aggregation algorithm, which is a public algorithm). In the application to silent threshold encryption, this step corresponds to aggregating the hints from different decrypters together to recover the message.

The decryption relation in our scheme satisfies this decomposition. During decryption, the only quantity that depends on the user's secret key $\mathbf{K}_i$ is $\mathsf{ht}_i = \underset{\sim}{\mathbf{s}^\mathsf{T}\mathbf{B}} \cdot \mathbf{K}_i$. Given ht_i, the remainder of the decryption algorithm only requires knowledge of ct and the helper decryption key hsk_i.

In silent threshold encryption, the standard security notion allows the adversary to request decryption hints from honest users. In the context of registered functional encryption, this translates to requiring security against an adversary that additionally has access to a hint-generation oracle. Handling such hint-generation queries follows via the same strategy as in prior works [CW26a, CW26b]. Specifically, observe that if the ciphertext is well-formed, then the decryption hint is simply $\mathsf{ht}_i = \underset{\sim}{\mathbf{s}^\mathsf{T}\mathbf{B}} \cdot \mathbf{K}_i \approx \mathbf{s}^\mathsf{T}\mathbf{P}_i$, where $\mathbf{P}_i = \mathbf{B}\mathbf{K}_i$ is the public key for user i. If the ciphertext was honestly-generated (with $\mathbf{s}$ as part of the encryption randomness), then the output of the decryption query is essentially just a function of the encryption randomness $\mathbf{s}$ and the public key $\mathbf{P}_i$, which the adversary could have computed itself without knowledge of the secret key. To argue this formally, we augment ciphertexts with a non-interactive zero-knowledge (NIZK) proof of knowledge of the encryption randomness (which includes $\mathbf{s}$); in the security proof,

the reduction algorithm extracts $\mathbf{s}$ from the proof and responds with $\mathbf{s}^\intercal \mathbf{P}_i$. Using noise smudging, we can ensure that this is statistically close to the actual decrypted value. We refer to the proof of Theorem 3.13 for the full details.

From registered FE to silent threshold encryption. Finally, to obtain our main implication to silent threshold encryption from registered functional encryption, we can appeal to the general template from [BLM+24]. Specifically, we proceed as follows:

- A user's public/secret key is simply a public/secret key-pair for the registered functional encryption scheme.
- Suppose we want to form a decryption quorum for a group of users with public keys $(\mathbf{P}_1, \ldots, \mathbf{P}_N)$ and a threshold T. We associate user i with the function $f_i(\mu, k)$ that takes as input the message $\mu \in \{0,1\}$ together with a key k for a pseudorandom function (PRF). The function f_i outputs the i^{th} Shamir share of μ generated using randomness derived from the PRF with key k and with threshold T. The aggregated public key mpk (with collusion bound $T-1$) functions as the public key for the decryption quorum.
- To encrypt a message μ with respect to mpk, the encrypter samples a PRF key k and encrypts the pair (μ, k).
- Observe that each user i can locally decrypt to obtain the i^{th} Shamir share of the message. Given T such shares, we can interpolate and recover the message. If the underlying registered functional encryption scheme supports split decryption, then we can decompose this process into a hint-generation and aggregation process.

If the PRF can be computed by a circuit with depth $\mathsf{poly}(\log \lambda)$ (e.g., [BPR12]), then we only require our registered functional encryption scheme to support circuits of depth $\mathsf{poly}(\log \lambda, \log N)$. Instantiated with our bounded-collusion registered functional encryption scheme, we obtain a silent threshold encryption scheme with ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$ (see Corollary 4.13).

Extending to dynamic thresholds. Finally, we can support dynamic thresholds (i.e., where the threshold is specified at encryption time) using a standard power-of-two construction (cf. [BLM+24, §6.2.1]). Roughly speaking, the aggregation algorithm prepares $\lceil \log N \rceil$ schemes, where the i^{th} scheme supports thresholds of up to 2^i. In this case, we provide the threshold as an additional input to the encryption algorithm (i.e., we now encrypt the triple (μ, T, k)), and the function associated with each user generates Shamir shares for the chosen threshold T. The i^{th} scheme is instantiated with a registered functional encryption scheme with collusion bound 2^i (which is good for all thresholds up to $T \le 2^i$). This increases the size of the master public key by a factor of $\log N$ and the ciphertext size by at most a factor of 2. We provide more details in Remark 4.14.

1.3 Additional Related Work

As mentioned in Section 1.2, a natural approach to build silent threshold encryption is to combine the distributed monotone-policy encryption schemes from [CW26a, AGY26a] with a secret sharing scheme for threshold policies. Correctness holds as long as the secret sharing scheme has small reconstruction coefficients and security holds as long as any collection of unauthorized shares is linearly independent. Unfortunately, existing approaches for designing threshold secret sharing fall short along one or both of these axes. In the following, let $\mathbf{p} \in \mathbb{Z}_q^n$ be the vector and let $\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_N \in \mathbb{Z}_q^n$ be the shares. We discuss some of the possibilities below:

- **Bit decomposition:** A folklore approach (cf. [BÇM21]) is to use bit decomposition to reduce the size of the Shamir interpolation coefficients. Namely, suppose $\hat{\mathbf{p}}_1, \ldots, \hat{\mathbf{p}}_N$ are Shamir shares of $\mathbf{p}$. Normally, the Lagrange interpolation coefficients ω_i are large. However, we can replace each interpolation coefficient ω_i with its binary decomposition and each share $\hat{\mathbf{p}}_i$ with a vector of shares corresponding to $\hat{\mathbf{p}}_i$ scaled up by powers of two: $(\hat{\mathbf{p}}_i, 2\hat{\mathbf{p}}_i, \ldots, 2^{\lceil \log q \rceil - 1}\hat{\mathbf{p}}_i)$. However, the resulting shares are now linearly dependent, which breaks security.
- **Clearing out the denominators:** Recall that for a T-out-of-N threshold policy, the Lagrange interpolation coefficients ω_i can be written as a ratio of two products, where each product is over T small integers. The problem is that over $\mathbb{Z}_q$, the modular inversion leads to large coefficients, which breaks correctness. A standard technique to obtain small reconstruction coefficients over $\mathbb{Z}_q$ is to clear out the denominators [ABV+12,

BGG+18]. Specifically, the encrypter could first scale each share $\hat{\mathbf{p}}_i$ by a common denominator for the Lagrange interpolation coefficients such that the resulting reconstruction coefficients have magnitude at most $2^{\tilde{O}(T)}$. The problem is the encrypter does not know in advance which set of users will reconstruct. Thus, the scaling factor applied to each share must simultaneously clear the denominators for all possible authorized sets of T users. If we consider the standard interpolation basis $\{1, \ldots, N\}$, a common denominator is $(N!)^2$. Thus, the encrypter would scale each share $\hat{\mathbf{p}}_i$ by $(N!)^2$. For correctness to hold, this means the bit length of the modulus q must now satisfy $\log q \geq \log((N!)^2) = \Omega(N \log N)$. This leads to a scheme where the ciphertext size is $\Omega(N \log N)$, which is *worse* than the trivial silent threshold encryption scheme of just encrypting Shamir shares of the message to each user's public key.

- **Lagrange interpolation over a cyclotomic ring:** A similar approach is to work over a cyclotomic ring and consider Lagrange interpolation over a subtractive set [AL21, KLNO24, CLW25]. For power-of-two cyclotomics (with suitably-chosen parameters), the prior works show that the bit length of the Lagrange interpolation coefficients scales with the threshold T. The limitation, however, is that the size of the subtractive set in these approaches is bounded by the ring dimension n (which corresponds to the security parameter). As a result, the maximum number of users the scheme supports is also bounded by the ring dimension. If we want to support N users, we need to work over a ring of dimension at least N; in this case, the ciphertexts also have size $\Omega(N)$, so this approach no longer confers any (asymptotic) advantage over the trivial approach.
- **Secret sharing with small reconstruction coefficients:** One could also consider alternative secret sharing schemes with small reconstruction coefficients. For instance, the work of [BGG+18] gives one way to do this by combining Valiant's construction of a monotone formula for computing a threshold function [Val84, Gol20] together with the classic linear secret sharing scheme for monotone formulas [BL88], which has $\{0, 1\}$-reconstruction coefficients. Another possibility is to consider a scheme that supports ramp threshold policies with small reconstruction coefficients [ANP23]. Unfortunately, neither of these schemes satisfies the additional linear independence property required by [CW26a, AGY26a].

In summary, we are not aware of any linear secret sharing scheme for threshold policies that supports small reconstruction coefficients and satisfies the additional linear independence property for unauthorized shares.

2 Preliminaries

We begin with some basic notation and definitions. We follow similar conventions as in [CW26a, CW26b] and parts of this section are taken verbatim from [CW26a, CW26b]. We write λ 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 *index-value pairs* (i, x_i). When S is a finite set, 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 λ 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 write $\tilde{O}(\cdot)$ to suppress polylogarithmic factors. 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. We recall the notion of a simulation-sound extractable non-interactive zero-knowledge (NIZK) argument for NP [BFM88, FLS90, Sah99, DDO+01]. Our definition is taken verbatim from [CW26b]:

Definition 2.1 (Simulation-Sound Extractable NIZK)**.** Let $C = \{C_\lambda\}_{\lambda \in \mathbb{N}}$ where C_λ is a set of Boolean circuits. A simulation-sound extractable NIZK argument Π_{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 λ, the setup algorithm outputs a common reference string crs.
- $\mathsf{TrapSetup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter λ, the trapdoor setup algorithm outputs a common reference string crs and a trapdoor td.
- $\mathsf{Prove}(\mathsf{crs}, C, x, w) \to \pi$: 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 π.
- $\mathsf{Verify}(\mathsf{crs}, C, x, \pi) \to b$: 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 π, 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 π.
- $\mathsf{Extract}(\mathsf{td}, C, x, \pi) \to w$: 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 π, the extraction algorithm outputs a witness $w \in \{0,1\}^h$ (or a special symbol $\perp$).

We require that Π_{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 λ, 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 Π_{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 λ 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, π) 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 π.
  - 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 Π_{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 (i.e., a rate-1 NIZK). In this work, we will use the following:

Fact 2.2 (Simulation-Sound Extractable Rate-1 NIZKs 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 λ, all crs in the support of $\mathsf{Setup}(1^\lambda)$ and all $C \in \mathcal{C}_\lambda$, a proof output by $\mathsf{Prove}(\mathsf{crs}, C, x, w)$ has size at most $|w| + \mathsf{poly}(\lambda, d)$.

2.1 Cover-Free Set Families

The construction and security analysis of our scheme will rely on cover-free set families [KS64]. We start by recalling the general definition below:

Definition 2.3 (Cover-Free Family [KS64, adapted])**.** Let $N, M, Q \in \mathbb{N}$. A family of sets $\{S_1, \ldots, S_N\} \subseteq 2^{[M]}$ is Q-cover-free if for all $i \in [N]$ and all $\mathcal{J} \subseteq [N] \setminus \{i\}$ where $|\mathcal{J}| \le Q$, we have that $S_i \notin \bigcup_{j\in\mathcal{J}} S_j$.

Local cover-free family. The works of [KS64, EFF85] give an explicit construction of a Q-cover-free family of sets $\{S_1, \ldots, S_N\} \subseteq 2^{[M]}$ where $M = O(Q^2 \log^2 N)$. Moreover, a lower bound from [DR82] shows that any Q-cover-free family of sets necessitates taking $M = \Omega\big(\frac{Q^2}{\log Q} \log N\big)$. In this work, we consider a relaxation of the requirements that allows us to set $M = \tilde{O}(Q)$, which translates to better parameters for our silent threshold encryption scheme. Specifically, we consider a *randomized* algorithm that samples the family of sets, and moreover, we only require the cover-free property to hold with overwhelming probability for a fixed *subfamily* of sets. We define the syntax and requirement below and then show how to realize our notion from any k-wise independent hash family.

Definition 2.4 (Local Cover-Free Family)**.** Let $N, M, Q, D \in \mathbb{N}$. A local cover-free family with parameters (N, M, Q, D) and public parameters of size ρ consists of an efficient deterministic algorithm Expand with the following syntax:

- $\mathsf{Expand}(\mathsf{pp}, i) \to S$: On input the public parameters $\mathsf{pp} \in \{0,1\}^\rho$ and an index $i \in [N]$, the expand algorithm outputs a set $S \subseteq [M]$ where $|S| \le D$.

We say that the local cover-free family satisfies ε-local cover-freeness if for all fixed sets of indices $\mathcal{I} \subseteq [N]$ where $|\mathcal{I}| \le Q$,

$$\Pr\left[S_i \notin \bigcup_{j\in\mathcal{I}\setminus\{i\}} S_j \text{ for all } i \in \mathcal{I} : \begin{array}{c}\mathsf{pp} \xleftarrow{R} \{0,1\}^\rho \\ \forall i \in \mathcal{I} : S_i = \mathsf{Expand}(\mathsf{pp}, i)\end{array}\right] \ge 1 - \varepsilon.$$

Constructing a local cover-free family. We now show how to construct a local cover-free family. Our construction relies on an almost k-wise independent hash function, which we define below.

Definition 2.5 (Almost k-Wise Independent Hash Function)**.** Let $k, N, M \in \mathbb{N}$ and $\varepsilon \ge 0$. A family of functions $\mathcal{H}$ with domain $[N]$ and codomain $[M]$ is ε-almost k-wise independent if for all distinct $x_1, \ldots, x_k \in [N]$, the statistical distance between the following distributions is at most ε:

- Sample $h \xleftarrow{R} \mathcal{H}$ and output $(h(x_1), \ldots, h(x_k))$.

- Sample $y_1, \ldots, y_k \xleftarrow{R} [M]$. Output $(y_1, \ldots, y_k)$.

When $\varepsilon = 0$, we say the family is k-wise independent.

Fact 2.6 (Almost k-Wise Independent Hash Function [WC81])**.** For all $k, N, M \in \mathbb{N}$ and $\varepsilon > 0$, there exists an explicit ε-almost k-wise independent hash family $\mathcal{H}$ with domain $[N]$ and codomain $[M]$ where each function $h \in \mathcal{H}$ can be described by $k \cdot O(\log N + \log(kM/\varepsilon))$ bits and evaluated in time $k \cdot \mathrm{polylog}(N, M, 1/\varepsilon)$. Moreover, there exists an efficient algorithm to sample $h \xleftarrow{R} \mathcal{H}$ using $k \cdot O(\log N + \log(kM/\varepsilon))$ uniformly random bits.

Fact 2.7 (Combinatorial Number System)**.** For all $M, D \in \mathbb{N}$ with $M \geq D$, there is an explicit bijection $\phi\colon [\binom{M}{D}] \to \binom{[M]}{D}$, where $\binom{[M]}{D}$ denotes the set of all D-element subsets of $[M]$. Both ϕ and ϕ^{-1} can be evaluated in time $\mathrm{poly}(M, D)$.

Construction 2.8 (Local Cover-Free Family)**.** Let $N, M, Q, D \in \mathbb{N}$ be parameters such that $M \geq QD$. Let $\mathcal{H}$ be an ε-almost Q-wise independent hash family with domain $[N]$ and codomain $[\binom{M}{D}]$. Let ρ be the randomness complexity for sampling uniformly from $\mathcal{H}$. We construct a local cover-free sampler Expand with parameters (N, M, Q, D) and public parameter size ρ as follows:

- $\mathsf{Expand}(\mathsf{pp}, i)$: On input $\mathsf{pp} \in \{0,1\}^\rho$, sample a hash function $h \xleftarrow{R} \mathcal{H}$ using randomness pp. Then, output $\phi(h(i)) \subseteq [M]$, where $\phi\colon [\binom{M}{D}] \to \binom{[M]}{D}$ is the mapping from Fact 2.7.

Lemma 2.9 (Local Cover-Free Family)**.** *Construction 2.8 satisfies ε'-local cover-freeness where $\varepsilon' = Q \cdot (QD/M)^D + \varepsilon$.*

Proof. Consider first the case where $\mathcal{H}$ is a Q-wise independent hash family. Take any set of indices $\mathcal{I} \subseteq [N]$ where $|\mathcal{I}| \leq Q$ and suppose $h \xleftarrow{R} \mathcal{H}$. For each $i \in \mathcal{I}$, let $S_i = \mathsf{Expand}(\mathsf{pp}, i) = \phi(h(i))$. Fix an index $i \in \mathcal{I}$ and consider the set $\mathcal{J}_i = \mathcal{I} \setminus \{i\}$ of the remaining at most $Q - 1$ indices. We bound the probability that $S_i \subseteq \bigcup_{j \in \mathcal{J}_i} S_j$. Since $|\mathcal{J}_i| \leq Q - 1$ and each S_j has exactly D elements, we have $|\bigcup_{j \in \mathcal{J}_i} S_j| \leq (Q-1)D < QD$. Since h is Q-wise independent and ϕ is a bijection, the distribution of S_i is *independent* of $\bigcup_{j \in \mathcal{J}_i} S_j$ and moreover, is uniform over the set of all subsets of $[M]$ of size D. Therefore,

$$\Pr\left[S_i \subseteq \textstyle\bigcup_{j \in \mathcal{J}_i} S_j\right] \leq \frac{\binom{QD}{D}}{\binom{M}{D}} \leq \left(\frac{QD}{M}\right)^D.$$

By a union bound over all $i \in \mathcal{I}$, we have

$$\Pr[\exists i \in \mathcal{I} : S_i \subseteq \textstyle\bigcup_{j \in \mathcal{J}_i} S_j] \leq Q \cdot (QD/M)^D,$$

where the probability is taken over the choice of $h \xleftarrow{R} \mathcal{H}$. The above analysis assumes that $\mathcal{H}$ is Q-wise independent. When $\mathcal{H}$ is ε-almost Q-wise independent, we have

$$\Pr[\exists i \in \mathcal{I} : S_i \subseteq \textstyle\bigcup_{j \in \mathcal{J}_i} S_j] \leq Q \cdot (QD/M)^D + \varepsilon,$$

which proves the claim. □

Corollary 2.10 (Local Cover-Free Family)**.** *Let N be the domain and Q be the collusion bound where $Q \leq N$. Take any $\varepsilon \in (0,1)$. Then there is an ε-local cover-free sampler with parameters (N, M, Q, D) where*

$$D = O(\log(Q/\varepsilon)) \quad \text{and} \quad M = O(Q \log(Q/\varepsilon)),$$

and whose public parameter size is $\rho = Q \cdot O\big(\log N + \log^2(Q/\varepsilon)\big)$.

Proof. Instantiate Construction 2.8 with $D = \lceil \log(Q/\varepsilon) \rceil + 1$, $M = 2QD$, and an $(\varepsilon/2)$-almost Q-wise independent hash family $\mathcal{H}$ with codomain $[\binom{M}{D}]$. Since $M = 2QD$, we have $(QD/M)^D = 2^{-D}$, so by Lemma 2.9 the resulting failure probability is

$$\varepsilon' = Q \cdot (QD/M)^D + \varepsilon/2 = Q \cdot 2^{-D} + \varepsilon/2 \leq \varepsilon/2 + \varepsilon/2 = \varepsilon,$$

since $D \geq \log(Q/\varepsilon) + 1 = \log(2Q/\varepsilon)$, so $Q \cdot 2^{-D} = 2^{\log Q - D} \leq \varepsilon/2$. If we instantiate $\mathcal{H}$ using Fact 2.6, we can sample from $\mathcal{H}$ using ρ uniformly random bits where

$$\rho = Q \cdot O\big(\log N + \log\big(Q\tbinom{M}{D}/\varepsilon\big)\big) = Q \cdot O\big(\log N + \log^2(Q/\varepsilon)\big),$$

and $\log \binom{M}{D} \leq D \log M = O(\log^2(Q/\varepsilon))$. □

2.2 Lattice Preliminaries

We now recall some basic facts about lattices. Our presentation here is adapted from [CW26a, §2.1] with some parts taken verbatim. 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 ℓ_∞-norm of $\mathbf{v}$, and for a matrix $\mathbf{V}$ we write $\|\mathbf{V}\| = \max_{i,j} \left|V_{i,j}\right|$.

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. 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. Next, we recall some basic properties of the discrete Gaussian distribution:

Lemma 2.11 (Gaussian Tail Bound [MP12, Lemma 2.6, adapted])**.** *Let n, m, q be lattice parameters where $m \geq 2n \log q$. For all but a $\mathsf{negl}(n)$-fraction of matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, for all $\sigma > \log m$, all $\lambda \in \mathbb{N}$, and all vectors $\mathbf{y} \in \mathbb{Z}_q^n$ in the span of $\mathbf{A}$,*

$$\Pr[\|\mathbf{u}\| > \sqrt{\lambda}\sigma : \mathbf{u} \leftarrow \mathbf{A}_\sigma^{-1}(\mathbf{y})] \leq m \cdot 2^{-\lambda}.$$

Moreover, for all $\sigma > 0$ and $\lambda \in \mathbb{N}$,

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

Lemma 2.12 (Gaussian Samples [GPV08, adapted])**.** *Let n, m, q, σ 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 $\mathsf{negl}(n)$:*

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

Truncated discrete Gaussian distribution. We write $\bar{D}_{\mathbb{Z},\sigma,B}$ to denote the *truncated* discrete Gaussian distribution with truncation bound B. To sample from $\bar{D}_{\mathbb{Z},\sigma,B}$, we first sample $x \leftarrow D_{\mathbb{Z},\sigma}$ and output x if $|x| \leq B$ and 0 otherwise. By Lemma 2.11, the statistical distance between the distributions $\bar{D}_{\mathbb{Z},\sigma,B}$ and $D_{\mathbb{Z},\sigma}$ is at most $2^{-\lambda} = \mathsf{negl}(\lambda)$ when $B \geq \sqrt{\lambda}\sigma$. By a hybrid argument, for all $n, m = \mathsf{poly}(\lambda)$ and all $B \geq \sqrt{\lambda}\sigma$, the statistical distance between the distributions $\bar{D}_{\mathbb{Z},\sigma,B}^{n \times m}$ and $D_{\mathbb{Z},\sigma}^{n \times m}$ is also $\mathsf{negl}(\lambda)$. We will also use the following smudging lemma. This was originally shown for the discrete Gaussian distribution, but also extends to the truncated distribution when the two distributions are statistically indistinguishable (via a standard hybrid argument). We state the version we use below:

Lemma 2.13 (Noise Smudging [BDE+18, adapted])**.** *Let λ be a security parameter. Let $x \in \mathbb{Z}$ be an integer and let $\sigma > 0$ be a width parameter such that $\sigma \geq |x| \cdot \lambda^{\omega(1)}$. Take any $B \geq \sqrt{\lambda}\sigma$. Then, the statistical distance between the following distributions is $\mathsf{negl}(\lambda)$:*

$$\left\{e : e \leftarrow \bar{D}_{\mathbb{Z},\sigma,B}\right\} \quad \textit{and} \quad \left\{x + e : e \leftarrow \bar{D}_{\mathbb{Z},\sigma,B}\right\}.$$

Lattice trapdoors. 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 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 $\mathbf{G}_n = [\mathbf{I}_n \otimes \mathbf{g}^\intercal \mid \mathbf{0}^{n \times (m - m')}]$. When the dimensions of $\mathbf{G}_n$ are clear from context, we omit the subscript and simply write $\mathbf{G}$. We write $\mathbf{G}_n^{-1} : \mathbb{Z}_q^n \to \{0,1\}^m$ to denote the operator that expands each component of the input into its binary decomposition, appropriately padded with 0s if $\mathbf{G}_n$ is padded. In particular, $\mathbf{G}_n \cdot \mathbf{G}_n^{-1}(\mathbf{x}) = \mathbf{x}$ for all $\mathbf{x} \in \mathbb{Z}_q^n$. Next, we recall the notion of a gadget trapdoor [MP12]:

Lemma 2.14 (Gadget Trapdoor [Ajt96, GPV08, MP12]). *Let n, m, q be lattice parameters with $m \geq 3n \log q$. There exist efficient algorithms* (TrapGen, 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 $\mathbf{T} \in \mathbb{Z}^{m \times m'}$ 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 $\mathbf{T} \in \mathbb{Z}^{m \times m'}$, a target vector $\mathbf{y} \in \mathbb{Z}_q^n$, and a Gaussian width parameter σ, the preimage-sampling algorithm outputs a vector $\mathbf{x} \in \mathbb{Z}^m$.*

Moreover, the above algorithms satisfy the following properties:

- **Trapdoor distribution:** *If $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$ and $\mathbf{A}' \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$, 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 $\mathbf{T} \in \mathbb{Z}^{m \times m'}$, all width parameters $\sigma > 0$, 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 $(\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T} \in \mathbb{Z}^{m \times m'})$ 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 $\mathsf{negl}(n)$:*

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

Homomorphic computation. Our construction of bounded-collusion registered functional encryption will rely on the lattice homomorphic evaluation machinery developed in [GSW13, BGG+14, BTVW17]. In this work, we will use the version for matrix-valued functions from [BTVW17]:

Theorem 2.15 (Homomorphic Evaluation of Matrix-Valued Functions [GSW13, BGG+14, BTVW17]). *Let λ be a security parameter, n, m, q be lattice parameters where $m \geq n\lceil \log q \rceil$, and $\ell, t \in \mathbb{N}$ be dimensions. There exists a pair of efficient and deterministic algorithms* (EvalF, EvalFX) *with the following properties:*

- $\mathsf{EvalF}(\mathbf{A}, f) \to \mathbf{A}_f$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$ and a matrix-valued function $f\colon \{0,1\}^\ell \to \mathbb{Z}_q^{n \times t}$, output a matrix $\mathbf{A}_f \in \mathbb{Z}_q^{n \times t}$.*
- $\mathsf{EvalFX}(\mathbf{A}, f, \mathbf{x}) \to \mathbf{H}_{\mathbf{A},f,\mathbf{x}}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$, a function $f\colon \{0,1\}^\ell \to \mathbb{Z}_q^{n \times t}$, and an input $\mathbf{x} \in \{0,1\}^\ell$, output a matrix $\mathbf{H}_{\mathbf{A},f,\mathbf{x}} \in \mathbb{Z}^{\ell m \times t}$.*

For all matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times \ell m}$, all functions $f\colon \{0,1\}^\ell \to \mathbb{Z}_q^{n \times t}$ that can be computed by a Boolean circuit of depth d and size s, and all inputs $\mathbf{x} \in \{0,1\}^\ell$, the matrices $\mathbf{A}_f = \mathsf{EvalF}(\mathbf{A}, f)$ and $\mathbf{H}_{\mathbf{A},f,\mathbf{x}} = \mathsf{EvalFX}(\mathbf{A}, f, \mathbf{x})$ satisfy the following key relation:

$$(\mathbf{A} - \mathbf{x}^\intercal \otimes \mathbf{G}) \cdot \mathbf{H}_{\mathbf{A},f,\mathbf{x}} = \mathbf{A}_f - f(\mathbf{x}). \tag{2.1}$$

In addition, $\|\mathbf{H}_{\mathbf{A},f,\mathbf{x}}\| \leq m^{O(d)} \cdot \mathsf{poly}(s)$. Finally, for any fixed f, the function $\mathbf{A} \mapsto \mathsf{EvalF}(\mathbf{A}, f)$ can be computed by a Boolean circuit of depth $d \cdot \mathsf{polylog}(m)$ and size $\mathsf{poly}(s, m)$.

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

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

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

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

We now give the statement of the assumption, except we substitute the truncated discrete Gaussian distribution in place of the discrete Gaussian distribution. As long as the truncation bound is chosen such that the truncated distribution and the original distribution are statistically close, and n, m, ℓ are polynomially bounded, the two formulations are equivalent. We give the statement we use below:

Assumption 2.16 (Decomposed LWE [AMR25, adapted])**.** Let λ 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 (ℓ, s)-decomposed LWE assumption with parameters (n, m, q, σ) 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 $\mathbf{B} = \mathbf{W}(\mathbf{I}_\ell \otimes \mathbf{R}) + \mathrm{vec}(\mathbf{I}_\ell)^\intercal \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times \ell^2 m}$, $\mathbf{W} \xleftarrow{R} \mathbb{Z}_q^{n \times \ell m}$, $\mathbf{R} \leftarrow \bar{D}^{m \times \ell m}_{\mathbb{Z}, s, \sqrt{\lambda} s}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \bar{D}^{\ell^2 m}_{\mathbb{Z}, \sigma, \sqrt{\lambda}\sigma}$, and $\mathbf{v} \xleftarrow{R} \mathbb{Z}_q^{\ell^2 m}$. We will sometimes write "ℓ-decomposed LWE" as shorthand to refer to an instance of the (ℓ, s)-decomposed LWE assumption for some (implicitly-defined) width parameter s.

Leftover hash lemma. Next, we recall a basic form of the leftover hash lemma from [HILL99], along with a generalized version [DORS08, ABB10] that also applies to the structured matrix $\mathbf{B}$ from the decomposed LWE assumption. The generalized version we use was formally shown in [CW26a]. We state the version from [CW26a], but adapted to the setting of truncated discrete Gaussians.

Lemma 2.17 (Leftover Hash Lemma [HILL99, adapted])**.** *Take $m, q \in \mathbb{N}$ where $m \geq 2 \log q$. Then, for all $k = \mathsf{poly}(m)$, the statistical distance between the following distributions is $\mathsf{negl}(m)$:*

$$\left\{(\mathbf{a}, \mathbf{a}^\intercal\mathbf{K}) : \mathbf{a} \xleftarrow{R} \mathbb{Z}_q^m, \mathbf{K} \xleftarrow{R} \{0,1\}^{m \times k}\right\} \quad \text{and} \quad \left\{(\mathbf{a}, \mathbf{u}^\intercal) : \mathbf{a} \xleftarrow{R} \mathbb{Z}_q^m, \mathbf{u} \xleftarrow{R} \mathbb{Z}_q^k\right\}.$$

Lemma 2.18 (Leftover Hash Lemma for Decomposed LWE Matrix [CW26a, adapted])**.** *Let λ be a security parameter and let n, m, q be integers such that $n \geq \lambda$, $m \geq 2n \log q$, 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 $k = \mathsf{poly}(\lambda)$, the statistical distance between the following distributions is $\mathsf{negl}(\lambda)$:*

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

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

Matrix commitments. The main lattice building block we use is the matrix commitment scheme of 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 $\mathbf{B} \in \mathbb{Z}_q^{n \times 4m^5}$, $\mathbf{V}_L \in \mathbb{Z}^{m \times L}$ are matrices defined by a set of public parameters $\mathsf{pp}_{\mathsf{com}}$, and $\mathbf{Z} \in \mathbb{Z}^{4m^5 \times L}$ is a low-norm opening. We summarize the main properties below (using the version from [Wee25, Appendix C.2] where the public parameters can be described by a uniform random string):

Theorem 2.19 (Matrix Commitment [Wee25, 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 Com algorithm outputs a commitment matrix $\mathbf{C} \in \mathbb{Z}_q^{n \times m}$.*
- $\mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, L) \to \mathbf{V}_L$*: On input the public parameters $\mathsf{pp}_{\mathsf{com}}$ and the length parameter $L \in \mathbb{N}$, the Ver algorithm outputs a matrix $\mathbf{V}_L \in \mathbb{Z}^{m \times L}$.*
- $\mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \to \mathbf{Z}$*: On input the public parameters $\mathsf{pp}_{\mathsf{com}}$ and a matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$, the Open algorithm outputs a matrix $\mathbf{Z} \in \mathbb{Z}^{4m^5 \times L}$.*

For all $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}, \mathbf{R}, \mathbf{B})$ *where* $\mathbf{W} \in \mathbb{Z}_q^{n \times 2m^3}$, $\mathbf{R} \in \mathbb{Z}^{m \times 2m^3}$, *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* $\mathbf{M} \in \mathbb{Z}_q^{n \times L}$, *and setting*

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

the following hold:

$$\mathbf{C}\mathbf{V}_L = \mathbf{M} - \mathbf{B}\mathbf{Z} \quad \textit{and} \quad \|\mathbf{V}_L\| \le O(\|\mathbf{R}\| \cdot m^4 \log q) \quad \textit{and} \quad \|\mathbf{Z}\| \le O(\|\mathbf{R}\| \cdot m^7 \log q \log L).$$

Explainable discrete Gaussian sampling. Similar to [BCTW16, AWY20, WWW22, CHW25, WW25, CW26a, AGY26a, GY26a], we will use a random oracle to sample from a discrete Gaussian distribution. In the security reduction, we will in turn need to program the discrete Gaussian samples output by the random oracle. To implement this, we need to use an explainable discrete Gaussian sampler [LW22]. In this work, we only require an explainable sampler for polynomially-bounded width parameters, so we can use the simple inversion sampler from [WWW22, §5.1] to instantiate our scheme. We give the formal description we require below:

Definition 2.20 (Explainable Discrete Gaussian Sampler [LW22, adapted])**.** Let λ be a security parameter and m be an output dimension. A ρ-explainable discrete Gaussian sampler Π_{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 λ, a dimension m, a width parameter σ, and randomness $r \in \{0,1\}^{\rho(\lambda,m)}$, the sampling algorithm outputs a vector $\mathbf{x} \in \mathbb{Z}^m$ where $\|\mathbf{x}\| \le \sqrt{\lambda}\sigma$.
- $\mathsf{Explain}(1^\lambda, \mathbf{x}, \sigma) \to r$: On input a security parameter λ, a sample $\mathbf{x} \in \mathbb{Z}^m$, and a width parameter σ, 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 $\sigma = \mathsf{poly}(\lambda)$, the statistical distance between the following distributions is bounded by $\mathsf{negl}(\lambda)$:[2]

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

Theorem 2.21 (Explainable Discrete Gaussian Sampler [WWW22, §5.1])**.** *For all* $m = \mathsf{poly}(\lambda)$*, there exists a* ρ*-explainable discrete Gaussian sampler* Π_{DGS} *where* $\rho(\lambda, m) = \mathsf{poly}(\lambda, m)$*.*

3 Succinct Bounded-Collusion Registered Functional Encryption

In this section, we formally define a succinct bounded-collusion registered functional encryption (FE) scheme [DPY24, ZLZ+24, BLM+24]. As in prior work [GHMR18, HLWW23, DPY24, ZLZ+24, BLM+24], we consider the simpler slotted setting where instead of supporting dynamic registration, there is a one-time aggregation algorithm that takes a collection of N public keys together with their associated functions $f_1, \ldots, f_N$ and aggregates them into a single succinct public key. In the bounded-collusion model, we fix a collusion bound Q (provided as input to the aggregation algorithm), and only require security to hold against an adversary that registers up to Q public keys in the system. We allow the size of the ciphertext to scale with Q. We note that allowing the ciphertext to scale with Q is necessary if we consider simulation-based security (cf. [AGVW13] and Remark 3.8). We give the formal syntax below:

[2]The original definition from [LW22] allows the Explain algorithm to additionally take a precision parameter 1^κ as input and only requires the statistical distance between these two distributions to be bounded by $1/\kappa$. This is sufficient for many applications. In our application, we consider a simulation-based security notion, and our security analysis requires the stronger property that the two distributions are *statistically indistinguishable*. When the discrete Gaussian has polynomial-size support, this is possible using standard inversion sampling [WWW22].

Definition 3.1 (Slotted Succinct Bounded-Collusion Registered FE)**.** Let λ be a security parameter and τ be a function-family parameter. Let $\mathcal{F} = \{\mathcal{F}_\tau\}_{\tau\in\mathbb{N}}$ be a family of Boolean circuits where $\mathcal{F}_\tau$ is a collection of Boolean functions on inputs of length $\ell = \ell(\tau)$ and single-bit outputs. A slotted succinct bounded-collusion registered FE scheme for $\mathcal{F}$ is a tuple of efficient algorithms $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\tau, N) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$, the function-family parameter $\tau \in \mathbb{N}$, and the number of slots $N \in \mathbb{N}$ (in binary), the setup algorithm outputs the public parameters pp. We assume that the public parameters include 1^λ, 1^τ, and N.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: On input the public parameters pp, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}) \to b$: On input the public parameters pp and a public key pk, the validity-checking algorithm outputs a bit $b \in \{0,1\}$.
- $\mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N)) \to (\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)$: On input a collusion bound $Q \in \mathbb{N}$, the public parameters pp, a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, and their associated functions $f_1, \ldots, f_N \in \mathcal{F}_\tau$, the aggregation algorithm outputs the master public key mpk and helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N$. This algorithm is deterministic.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x}) \to \mathsf{ct}$: On input the master public key mpk and an input $\mathbf{x} \in \{0,1\}^\ell$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) \to b$: On input a secret key sk, a helper decryption key hsk, and a ciphertext ct, the decryption algorithm outputs a bit $b \in \{0,1\}$.

We require that Π_{RFE} satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$ and function-family parameters $\tau \in \mathbb{N}$, all $N \in \mathbb{N}$, all $Q \le N$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^\tau, N)$, all indices $i \in [N]$, all $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, all pk_j for $j \neq i$, all functions $f_1, \ldots, f_N \in \mathcal{F}_\tau$, all inputs $\mathbf{x} \in \{0,1\}^\ell$, and setting $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N))$, we have that

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct}) = f_i(\mathbf{x}) : \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x})] = 1. \tag{3.1}$$

- **Completeness:** For all security parameters $\lambda \in \mathbb{N}$, all function-family parameters $\tau \in \mathbb{N}$, all $N \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^\tau, N)$, and all $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, we have $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}) = 1$.
- **Simulation security:** For an adversary $\mathcal{A}$, a simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{cor}}, \mathcal{S}_{\mathsf{ct}})$, and a bit $b \in \{0,1\}$, we define the simulation-security game as follows:
  - **Setup:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs the function-family parameter 1^τ and a slot count N. Then, the challenger samples the public parameters as follows:
    - If $b = 0$, the challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^\tau, N)$.
    - If $b = 1$, the challenger samples $(\mathsf{pp}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{pp}}(1^\lambda, 1^\tau, N)$.

    The challenger gives pp to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} = 0$, an empty dictionary $\mathsf{D}_{\mathsf{key}}$, and a set of corrupted slots $C = \emptyset$.
  - **Pre-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    - **Key-generation query:** When $\mathcal{A}$ makes a key-generation query, the challenger starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$. Then, it proceeds as follows:
      - If $b = 0$, the challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and adds $\mathsf{ctr} \mapsto (\mathsf{pk}, \mathsf{sk})$ to $\mathsf{D}_{\mathsf{key}}$.
      - If $b = 1$, the challenger samples $(\mathsf{pk}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathsf{st}_{\mathcal{S}})$ and adds $\mathsf{ctr} \mapsto (\mathsf{pk}, \perp)$ to $\mathsf{D}_{\mathsf{key}}$.

The challenger replies with $(\mathsf{ctr}, \mathsf{pk})$.

* **Corruption query:** In a corruption query, the adversary specifies an index $1 \le c \le \mathsf{ctr}$. The challenger responds as follows:
  - If $b = 0$, the challenger looks up the tuple $(\mathsf{pk}, \mathsf{sk}) = \mathsf{D}_{\mathsf{key}}[c]$.
  - If $b = 1$, the challenger runs $(\mathsf{sk}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{cor}}(\mathsf{st}_{\mathcal{S}}, c, \varnothing)$. Note that the third argument to $\mathcal{S}_{\mathsf{cor}}$ is only set for post-challenge corruption queries.

  In both cases, the challenger responds with sk.

– **Challenge phase:** The adversary first outputs a collusion bound 1^Q where $Q \le N$. Then, for each slot $i \in [N]$, the adversary must specify a tuple $(c_i^*, f_i^*, \mathsf{pk}_i^*)$ where $f_i^* \in \mathcal{F}_\tau$ and either $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$ to reference a challenger-generated key or $c_i^* = \perp$ to reference a key outside this set. In addition, the adversary specifies a challenge input $\mathbf{x}^* \in \{0,1\}^\ell$. For each $i \in [N]$, the challenger then checks the following:

  * If $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$, then the challenger looks up $(\mathsf{pk}', \mathsf{sk}') = \mathsf{D}_{\mathsf{key}}[c_i^*]$. If $\mathsf{pk}_i^* \neq \mathsf{pk}'$, then the challenger halts with output 0. If the adversary previously made a corruption query on index c_i^*, then the challenger adds the index i to C.
  * If $c_i^* = \perp$, then the challenger checks that $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i^*) = 1$ and outputs 0 if not. If the check passes, the challenger adds index i to C.

  The challenger checks that $|C| \le Q$ and outputs 0 if not. The challenger then constructs the challenge ciphertext as follows:

  * If $b = 0$, the challenger computes

$$(\mathsf{mpk}^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_N^*) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*))$$

    and $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}^*, \mathbf{x}^*)$.
  * If $b = 1$, the challenger runs $(\mathsf{ct}^*, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{ct}}(\mathsf{st}_{\mathcal{S}}, 1^Q, \{(c_i^*, f_i^*, \mathsf{pk}_i^*, y_i^*)\}_{i \in [N]})$, where $y_i^* = f_i^*(\mathbf{x}^*)$ if $i \in C$ and $y_i^* = \perp$ otherwise.

– **Post-challenge query phase:** Adversary $\mathcal{A}$ can continue to make corruption queries as in the pre-challenge query phase. Let $S_c \subseteq [N]$ be the set of slots associated with a counter value c in the challenge phase. The challenger responds to these queries as follows:

  * If $b = 0$, the challenger looks up the tuple $(\mathsf{pk}, \mathsf{sk}) = \mathsf{D}_{\mathsf{key}}[c]$.
  * If $b = 1$, the challenger computes $(\mathsf{sk}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{cor}}(\mathsf{st}_{\mathcal{S}}, c, \{f_i^*(\mathbf{x}^*)\}_{i \in S_c})$.

  The challenger adds the slots in S_c to C and outputs 0 if $|C| > Q$. In both cases, the challenger responds with sk.

– **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. If $\mathcal{A}$ aborts without outputting a bit, then we take $b' = 0$.

We say that Π_{RFE} satisfies simulation security if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{cor}}, \mathcal{S}_{\mathsf{ct}})$ such that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda) \tag{3.2}$$

in the simulation security game defined above.

- **Succinctness:** There exists a polynomial $\mathsf{poly}(\cdot,\cdot,\cdot,\cdot)$ such that for all security parameters $\lambda \in \mathbb{N}$, all function-family parameters $\tau \in \mathbb{N}$, all $N \in \mathbb{N}$, all $Q \le N$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, 1^\tau, N)$, all public keys pk_i for $i \in [N]$, all functions $f_1, \ldots, f_N \in \mathcal{F}_\tau$, all inputs $\mathbf{x} \in \{0,1\}^\ell$, and setting $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N))$, we have that $|\mathsf{mpk}|, |\mathsf{hsk}_i| = \mathsf{poly}(\lambda, \tau, Q, \log N)$ for all $i \in [N]$.

Remark 3.2 (Registered FE in the Random Oracle Model)**.** When defining simulation security for a registered FE scheme in the random oracle model [BR93], we allow the simulator $\mathcal{S}$ to simulate the adversary's queries to the random oracle. Formally, the simulator $\mathcal{S}$ includes an additional algorithm $\mathcal{S}_{\mathsf{ro}}$ that takes as input the state $\mathsf{st}_{\mathcal{S}}$ and an input x and responds with a value y together with an updated state $\mathsf{st}_{\mathcal{S}}$. Then, when $b = 1$, the challenger answers the adversary's random oracle queries using $\mathcal{S}$. When $b = 0$, the challenger answers random oracle queries with a fresh random value (for each distinct input).

Relaxed notions of security. We now define a few relaxations of the simulation security model:

- **Static security:** In the static security game, we require the adversary to declare the indices of the corrupted slots ahead of time. The adversary still picks the public keys and functions for the corrupted slots during the challenge phase (which could depend on the public keys for honest users). This relaxation is a common one in the setting of both registered ABE and silent threshold cryptography (cf. [GJM+24, GLWW24, WW26, SWW26]).
- **Input-selective security:** We also consider an input-selective notion of security where the adversary has to declare its challenge input $\mathbf{x}^*$ in advance. This is a standard relaxation in lattice-based (registered) ABE (e.g., [BGG+14, CHW25]), a weaker form of functional encryption. We note that using complexity leveraging, we can generically lift an input-selective scheme into an adaptively-secure scheme (Remark 3.5).
- **Semi-malicious security:** Finally, for ease of exposition, we work in the semi-malicious model where in the security game (Definition 3.1), the adversary is required to provide the challenger with the key-generation randomness it used to sample the keys for malicious users. This is sometimes also referred to as the registered-key model [RY07]. As noted in prior works [RY07, LWW25], it is straightforward to upgrade any scheme with semi-malicious security into a fully secure scheme by having users include a simulation-extractable NIZK proof of well-formedness in their public keys (Remark 3.7). Working in the semi-malicious model simplifies both the scheme description and its security analysis.

We now define each of these notions formally.

Definition 3.3 (Static Security)**.** Let Π_{RFE} be a slotted succinct bounded-collusion registered FE scheme. We define the *static security* game for Π_{RFE} by restricting the adversary in the security game from Definition 3.1 as follows:

- At the beginning of the setup phase, the adversary commits to the set of corrupted slots $C \subseteq [N]$.
- The adversary cannot make any corruption queries in either phase. In particular, there is no post-challenge query phase in the static security game.
- During the challenge phase, the adversary is required to set $c_i^* = \bot$ for all $i \in C$ and $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$ for all $i \notin C$.

We say that Π_{RFE} satisfies static security if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{ct}})$ such that for all efficient adversaries $\mathcal{A}$, Eq. (3.2) holds in the static simulation security experiment. Note that in this setting, the simulator algorithm $\mathcal{S}_{\mathsf{pp}}$ now takes the tuple $(1^\lambda, 1^\tau, N, C)$ as its input. Since there are no corruption queries in the static security game, we drop the simulator algorithm $\mathcal{S}_{\mathsf{cor}}$ from Definition 3.1.

Definition 3.4 (Input-Selective Security)**.** Let Π_{RFE} be a slotted succinct bounded-collusion registered FE scheme. We define the *input-selective security* game for Π_{RFE} by restricting the adversary in the security game from Definition 3.1 as follows:

- During the setup phase, the adversary must commit to the challenge input $\mathbf{x}^* \in \{0,1\}^\ell$.

We say that Π_{RFE} satisfies input-selective security if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{cor}}, \mathcal{S}_{\mathsf{ct}})$ such that for all efficient adversaries $\mathcal{A}$, Eq. (3.2) holds in the input-selective simulation security experiment.

Remark 3.5 (Adaptive Security via Complexity Leveraging)**.** Using standard complexity leveraging, any scheme that satisfies input-selective security (Definition 3.4) is also secure against an adversary that adaptively chooses the challenge input $\mathbf{x}^* \in \{0,1\}^\ell$, so long as the input length ℓ is a priori bounded. Complexity leveraging incurs a 2^ℓ loss in the security reduction, so we will have to rely on sub-exponential security of the underlying input-selective scheme, and moreover, the scheme parameters will increase by a factor of $\mathsf{poly}(\ell)$.

Definition 3.6 (Semi-Malicious Correctness and Security)**.** Let Π_{RFE} be a slotted succinct bounded-collusion registered FE scheme. We now define *semi-malicious* variants of the correctness and simulation-security requirements from Definition 3.1:

- **Semi-malicious correctness:** We say Π_{RFE} satisfies semi-malicious correctness if Eq. (3.1) holds for collections of public keys pk_j for $j \neq i$ where there exists sk_j such that $(\mathsf{pk}_j, \mathsf{sk}_j)$ is in the support of $\mathsf{KeyGen}(\mathsf{pp})$.
- **Semi-malicious security:** We define the *semi-malicious* simulation-security game by modifying the challenge phase of the security game from Definition 3.1 to require that the adversary additionally provide the key-generation randomness for every key it chooses (i.e., every index $i \in [N]$ where $c_i^* = \bot$). Concretely, if $c_i^* = \bot$, the adversary must provide (sk_i^*, r_i) such that $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) = \mathsf{KeyGen}(\mathsf{pp}; r_i)$. Moreover, in the case where $b = 1$, the simulator $\mathcal{S}_{\mathsf{ct}}$ is additionally given the randomness r_i for each slot i where $c_i^* = \bot$.

  We say that Π_{RFE} satisfies semi-malicious security if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{cor}}, \mathcal{S}_{\mathsf{ct}})$ such that for all efficient adversaries $\mathcal{A}$, Eq. (3.2) holds in the semi-malicious simulation security experiment.

Since completeness stipulates that $\mathsf{IsValid}$ always outputs 1 on every honestly-generated key, when we consider semi-malicious security and correctness, we will omit the $\mathsf{IsValid}$ algorithm from the scheme description.

Remark 3.7 (From Semi-Malicious to Malicious Security)**.** Any scheme that satisfies semi-malicious correctness and security (Definition 3.6) can be generically upgraded into one that satisfies the standard correctness and security requirements from Definition 3.1 using a simulation-extractable NIZK proof of knowledge. Specifically, the public key would additionally include a NIZK proof of knowledge of the associated key-generation randomness. In the security reduction, the challenger and/or simulator would then extract the key-generation randomness from the NIZK proof. We refer to [LWW25] for a formal exposition of this transformation in the setting of multi-authority registered ABE.

Remark 3.8 (Ciphertext Size and the Collusion Bound)**.** In Definition 3.1, we allow the ciphertext size to grow with the collusion bound Q. This dependence is inherent. In particular, the lower bound from [AGVW13] for simulation-secure *non-adaptive* functional encryption extends naturally to the setting of statically-secure slotted registered FE with simulation security whenever the scheme is expressive enough to support a function family that contains a pseudorandom function. Namely, if we require simulation security against an adversary that (statically) corrupts at most Q slots, the size of the ciphertext is necessarily $\Omega(Q)$. This follows via the same incompressibility argument as in [AGVW13]. We give a brief sketch of their argument here:

- Let PRF be a pseudorandom function. Consider the adversary that registers Q (honestly-generated) public keys where the i^{th} key is associated with the function $f_i(k) := \mathsf{PRF}(k, i)$. The adversary makes no key-generation queries or corruption queries.
- In the challenge phase, the adversary requests an encryption of a random PRF key k.
- If the slotted registered FE scheme satisfies simulation security, then the simulator needs to simulate a ciphertext ct that decrypts to $y_1, \ldots, y_Q$ where $y_i = \mathsf{PRF}(k, i)$ with respect to the adversary's secret keys. However, if PRF is a secure pseudorandom function, then the string $y_1, \ldots, y_Q$ is pseudorandom, and thus, incompressible. This means the size of ct must be at least $\Omega(Q)$.

Split decryption and CCA security. For our application to silent threshold encryption, it will be helpful to decompose the decryption algorithm Decrypt into two separate steps:

- **Hint generation:** The first step is a "hint-generation" algorithm that only depends on the ciphertext ct and the user's secret key sk, but *not* the user's helper decryption key. This step outputs a decryption hint ht.
- **Message recovery:** Given the ciphertext ct, the decryption hint ht, and the helper decryption key hsk, the recover algorithm recovers the message. Since the helper decryption key hsk is a public quantity, the message-recovery algorithm does not rely on any user secrets.

Looking ahead to our application to silent threshold encryption, this split decryption procedure will be needed to support a one-round distributed decryption procedure where the individual decrypters implement the local decryption procedure (that only requires knowledge of their secret key) and the aggregator applies the recover algorithm to the partial decryptions and combines them together to obtain the underlying message. Moreover, we also define a notion of CCA security where we require security to hold even if the adversary has access to a decryption oracle in the simulation security game.

Definition 3.9 (Split Decryption)**.** Let $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a registered functional encryption scheme. We say Π_{RFE} satisfies split decryption if there exist two efficient algorithms with the following syntax:

- $\mathsf{GetHint}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{ht}$: On input a secret key sk and a ciphertext ct, the hint-generation algorithm outputs a decryption hint ht.
- $\mathsf{Recover}(\mathsf{ht}, \mathsf{hsk}, \mathsf{ct}) \to b$: On input a decryption hint ht, a helper decryption key hsk, and a ciphertext ct, the output-recovery algorithm outputs a bit $b \in \{0, 1\}$. This algorithm is deterministic.

Moreover, if Π_{RFE} satisfies split decryption, we require that the decryption algorithm Decrypt be defined as follows:

$$\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) := \mathsf{Recover}(\mathsf{GetHint}(\mathsf{sk}, \mathsf{ct}), \mathsf{hsk}, \mathsf{ct}).$$

Since Decrypt is defined implicitly in terms of (GetHint, Recover), we will define registered functional encryption with split decryption by the tuple $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Recover})$ which does not contain an explicit Decrypt algorithm.

Definition 3.10 (CCA Security)**.** Let $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Recover})$ be a registered functional encryption scheme with split decryption. For an adversary $\mathcal{A}$, a simulator algorithm $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{cor}}, \mathcal{S}_{\mathsf{ct}}, \mathcal{S}_{\mathsf{ht}})$, and a bit $b \in \{0, 1\}$, we define the CCA security game exactly as we defined the simulation-security game in Definition 3.1, except we allow the adversary to additionally make hint-generation queries in the pre-challenge and post-challenge query phases:

- **Hint-generation query:** In a hint-generation query, the adversary specifies a ciphertext ct and an index $1 \le c \le \mathsf{ctr}$. The challenger responds as follows:
  - If $b = 0$, the challenger looks up the tuple $(\mathsf{pk}, \mathsf{sk}) = \mathsf{D}_{\mathsf{key}}[c]$ and replies with $\mathsf{GetHint}(\mathsf{sk}, \mathsf{ct})$.
  - If $b = 1$, the challenger runs $(\mathsf{ht}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{ht}}(\mathsf{st}_{\mathcal{S}}, c, \mathsf{ct})$ and replies with ht.

  For post-challenge hint-generation queries, we additionally require that $\mathsf{ct} \neq \mathsf{ct}^*$ where ct^* is the challenge ciphertext the challenger constructed in the challenge phase.[3]

We say that Π_{RFE} satisfies CCA security if there exists an efficient simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{cor}}, \mathcal{S}_{\mathsf{ct}}, \mathcal{S}_{\mathsf{ht}})$ such that for all efficient adversaries $\mathcal{A}$, Eq. (3.2) holds in the CCA security game. We also define static CCA security to be the analogous extension of Definition 3.3 to allow hint-generation queries.

3.1 Constructing Succinct Bounded-Collusion Registered FE

In this section, we show how to construct a succinct bounded-collusion registered FE scheme for general circuits. The public parameter and key sizes are independent of the collusion bound Q, while the ciphertext size grows with $Q \cdot \tilde{O}(d) + \ell \cdot \mathsf{poly}(\lambda, d)$, where d is a bound on the circuit depth and ℓ is a bound on the input length.

Construction 3.11 (Slotted Succinct Bounded-Collusion Registered FE)**.** Let λ be a security parameter, τ be a function-family parameter, and N be the number of slots. We define the following quantities:

[3] Here, we could also define a stronger notion that allows post-challenge hint-generation queries on the challenge ciphertext ct^*. In this case, if the adversary specified a tuple of the form $(c, f_i^*, \mathsf{pk}_i^*)$ during the challenge phase, then the simulator $\mathcal{S}_{\mathsf{ht}}$ would additionally be provided the value $f_i^*(\mathbf{x}^*)$. In this work, we consider the simpler definition here where we disallow hint-generation queries on the challenge ciphertext.

- Let $\mathcal{F} = \{\mathcal{F}_\tau\}_{\tau\in\mathbb{N}}$ be a family of Boolean functions where $\mathcal{F}_\tau$ is a collection of functions $f\colon \{0,1\}^{\ell(\tau)} \to \{0,1\}$ on inputs of length $\ell = \ell(\tau)$ and which can be computed by a Boolean circuit of depth at most $d = d(\tau)$ and size at most $s(\tau)$.
- Let $(n, m, q, \sigma_{\mathsf{LWE}})$ be lattice parameters and $\sigma_{\mathsf{pp}}, \sigma_{\mathsf{agg}}, \sigma_{\mathsf{smudge}}, \sigma_{\mathsf{ht}} > 0$ be Gaussian width parameters. These parameters are functions of λ, τ, and N, and we assume throughout that n, m, and $\log q$ are polynomially bounded in λ, τ, and $\log N$ (as are the input length $\ell = \ell(\tau)$, the depth $d = d(\tau)$, and the size $s = s(\tau)$ associated with the function family).
- Let $\Pi_{\mathsf{DGS}} = (\mathsf{DGS.Sample}, \mathsf{DGS.Explain})$ be a ρ-explainable discrete Gaussian sampler (Definition 2.20).
- 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 NP (Definition 2.1). In this construction, we will consider the following Boolean relation:
  - Let C_{ct} be the Boolean circuit that takes a statement $(\mathbf{B}, \mathrm{ct}_{\mathsf{main}}, \beta)$, a witness $(\mathbf{s}, \mathbf{e}) \in \mathbb{Z}_q^n \times \mathbb{Z}^{4m^5}$, and outputs 1 if $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$, $\beta \in \mathbb{Z}$, and $\|\mathbf{e}\| \le \beta$. Here, $\mathrm{ct}_{\mathsf{main}} = (\boldsymbol{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c_1', \ldots, c_M')$, with $\boldsymbol{\Psi} \in \mathbb{Z}_q^{(n+1)\times\ell m}$, $\mathbf{c}_1^\intercal \in \mathbb{Z}_q^{4m^5}$, $\mathbf{c}_2^\intercal \in \mathbb{Z}_q^m$, $\mathbf{c}_3^\intercal \in \mathbb{Z}_q^{\hat{\ell}m}$, and $c_1', \ldots, c_M' \in \mathbb{Z}_q$. Note that C_{ct} can be computed by a circuit of depth $\mathsf{poly}(n, m, \log q)$.
- Let $H_1\colon \{0,1\}^* \times \{0,1\}^* \to \mathbb{Z}_q^n$, $H_2\colon \{0,1\}^* \times \mathbb{N} \to \{0,1\}$, and $H_3\colon \{0,1\}^* \times \mathbb{N} \to \{0,1\}^\rho$ be hash functions (which we model as random oracles in the security analysis).
- In the following description, we will consider gadget matrices $\mathbf{G}_n$ and $\mathbf{G}_{n+1}$ with heights n and $n+1$, respectively. For ease of exposition, we will pad *both* gadget matrices to have the *same* width $m > (n+1)\lceil\log q\rceil$.

We construct a slotted bounded-collusion registered FE scheme with split decryption $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Recover})$ for $\mathcal{F}$ as follows:[4]

- $\mathsf{Setup}(1^\lambda, 1^\tau, N)$: On input the security parameter λ, the function-family parameter τ, and the number of slots N, the setup algorithm samples

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

  It sets $\mathbf{B} = \mathbf{W}_{\mathsf{com}}(\mathbf{I}_{2m^2} \otimes \mathbf{R}_{\mathsf{com}}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G}_n \in \mathbb{Z}_q^{n\times 4m^5}$ and $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}, \mathbf{B})$. It outputs $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$, the key-generation algorithm samples $\mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5\times m}$ and computes $\mathbf{P} = \mathbf{B}\mathbf{K} \in \mathbb{Z}_q^{n\times m}$. It outputs the public key $\mathsf{pk} = \mathbf{P}$ and the secret key $\mathsf{sk} = (\mathbf{K}, \mathsf{pp})$.
- $\mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N))$: On input the collusion bound Q, the public parameters $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$, the public keys $\mathsf{pk}_i = \mathbf{P}_i \in \mathbb{Z}_q^{n\times m}$ and associated function $f_i \in \mathcal{F}_\tau$ for each $i \in [N]$, the aggregation algorithm proceeds as follows:
  - Let $\hat{\ell} = (n+1)\ell m\lceil\log q\rceil$. Then for each $i \in [N]$, define the function

$$f_i'\colon \{0,1\}^\ell \to \mathbb{Z}_q^{n+1} \quad \text{where} \quad f_i'(\mathbf{x}) := f_i(\mathbf{x}) \cdot \lfloor q/2\rfloor \cdot \boldsymbol{\eta}_{n+1}, \tag{3.3}$$

    where $\boldsymbol{\eta}_{n+1} \in \{0,1\}^{n+1}$ is the $(n+1)^{\mathrm{st}}$ elementary basis vector. Next, define the function $\hat{f}_i\colon \{0,1\}^{\hat{\ell}} \to \mathbb{Z}_q^n$ as follows:

[4] Throughout, we work in the semi-malicious model, so we omit the explicit IsValid algorithm.

On input $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$:

1. Let $\boldsymbol{\Psi} \in \mathbb{Z}_q^{(n+1)\times \ell m}$ be the matrix with binary representation $\boldsymbol{\varphi}$.
2. Let $\boldsymbol{\psi}_{f_i'} = \mathsf{EvalF}(\boldsymbol{\Psi}, f_i') \in \mathbb{Z}_q^{n+1}$.
3. Output $\bar{\boldsymbol{\psi}}_{f_i'} \in \mathbb{Z}_q^n$, where $\bar{\boldsymbol{\psi}}_{f_i'}$ denotes the first n components of $\boldsymbol{\psi}_{f_i'}$.

Figure 1: The homomorphic evaluation function $\hat{f}_i$. By Theorem 2.15, $\hat{f}_i$ can be computed by a Boolean circuit of depth $\hat{d} = d \cdot \mathrm{polylog}(m)$ and size $\hat{s} = \mathrm{poly}(s, m)$.

– Define

$$\begin{aligned} \xi &= (\mathsf{pp}, Q, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N)) \\ \mathbf{A} &= [H_1(\xi, (1,1)) \mid \cdots \mid H_1(\xi, (1, \hat{\ell}m))] \in \mathbb{Z}_q^{n \times \hat{\ell}m}. \end{aligned} \tag{3.4}$$

Then, for each $i \in [N]$, let

$$\mathbf{a}_{\hat{f}_i} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i) \in \mathbb{Z}_q^n.$$

– Let CFF.Expand be the local cover-free sampler from Corollary 2.10 with collusion bound Q, domain size N, and $\varepsilon = 2^{-\log^2(\lambda)} = \mathsf{negl}(\lambda)$. Let ρ_{CFF} be the associated public parameter size and M be the size of the output universe. Compute the public parameters

$$\mathsf{pp}_{\mathsf{CFF}} = H_2(\xi, 1) \| \cdots \| H_2(\xi, \rho_{\mathsf{CFF}}).$$

Then, for each $i \in [N]$, let

$$S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i) \subseteq [M].$$

Next, for each $j \in [M]$, compute a smudging term $\mathbf{w}_j = H_1(\xi, (2, j)) \in \mathbb{Z}_q^n$.

– Compute $\mathbf{V}_N = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N) \in \mathbb{Z}^{m \times N}$ and for each $i \in [N]$, let $\mathbf{v}_i \in \mathbb{Z}^m$ be the i^{th} column of $\mathbf{V}_N$. Next, derive the matrix

$$\mathbf{C}_0 = [H_1(\xi, (3,1)) \mid \cdots \mid H_1(\xi, (3,m))] \in \mathbb{Z}_q^{n \times m}.$$

For each $i \in [N]$, compute

$$\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi, i)) \in \mathbb{Z}^m. \tag{3.5}$$

By Definition 2.20, we have $\|\mathbf{u}_i\| \le \sqrt{\lambda}\sigma_{\mathsf{agg}}$.

– Define the matrix

$$\mathbf{M} = \left[\mathbf{P}_1\mathbf{u}_1 - \mathbf{C}_0\mathbf{v}_1 + \mathbf{a}_{\hat{f}_1} + \textstyle\sum_{j\in S_1} \mathbf{w}_j \mid \cdots \mid \mathbf{P}_N\mathbf{u}_N - \mathbf{C}_0\mathbf{v}_N + \mathbf{a}_{\hat{f}_N} + \sum_{j\in S_N} \mathbf{w}_j\right] \in \mathbb{Z}_q^{n\times N}.$$

Compute the commitment $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ and let $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1$. Then, compute $\mathbf{Z} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M}) \in \mathbb{Z}^{4m^5 \times N}$ and for each $i \in [N]$, let $\mathbf{z}_i \in \mathbb{Z}^{4m^5}$ be the i^{th} column of $\mathbf{Z}$.

Output the master public key $\mathsf{mpk} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{w}_1, \ldots, \mathbf{w}_M)$ along with the helper decryption keys $\mathsf{hsk}_i = (\mathbf{A}, \mathbf{v}_i, \mathbf{z}_i, \mathbf{u}_i, S_i, f_i)$ for all $i \in [N]$.

- $\mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x})$: On input the master public key $\mathsf{mpk} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{w}_1, \ldots, \mathbf{w}_M)$ and an input $\mathbf{x} \in \{0,1\}^\ell$, sample the following quantities:

$$\begin{aligned} &\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n \\ &\mathbf{e} \leftarrow \bar{D}^{4m^5}_{\mathbb{Z}, \sigma_{\mathsf{LWE}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}}, \ e_{\mathbf{w}_1}, \ldots, e_{\mathbf{w}_M} \leftarrow \bar{D}_{\mathbb{Z}, \sigma_{\mathsf{smudge}}, \sqrt{\lambda}\sigma_{\mathsf{smudge}}} \\ &\mathbf{R}_{\boldsymbol{\Psi}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times \ell m}, \ \mathbf{R}_{\mathbf{C}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}, \ \mathbf{R}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times \hat{\ell}m}. \end{aligned}$$

Compute an encryption of the input $\mathbf{x}$:

$$\boldsymbol{\Psi} = \begin{bmatrix} \mathbf{B} \\ \mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal \end{bmatrix} \mathbf{R}_{\boldsymbol{\Psi}} + \mathbf{x}^\intercal \otimes \mathbf{G}_{n+1} \in \mathbb{Z}_q^{(n+1)\times \ell m}.$$

Let $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ be the binary representation of $\boldsymbol{\Psi}$. Set

$$\mathsf{ct}_{\mathsf{main}} = \left(\boldsymbol{\Psi}, \mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal,\ \mathbf{s}^\intercal \mathbf{C} + \mathbf{e}^\intercal \mathbf{R}_{\mathbf{C}},\ \mathbf{s}^\intercal (\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}_n) + \mathbf{e}^\intercal \mathbf{R}_{\mathbf{A}},\ \mathbf{s}^\intercal \mathbf{w}_1 + e_{\mathbf{w}_1},\ \ldots,\ \mathbf{s}^\intercal \mathbf{w}_M + e_{\mathbf{w}_M}\right).$$

Finally, compute the proof

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

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

- $\mathsf{GetHint}(\mathsf{sk}_i, \mathsf{ct})$: On input the secret key $\mathsf{sk}_i = (\mathbf{K}_i, \mathsf{pp})$ where $\mathbf{K}_i \in \{0,1\}^{4m^5 \times m}$ and $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$, and a ciphertext $\mathsf{ct} = (\mathsf{ct}_{\mathsf{main}}, \pi_{\mathsf{ct}})$, where $\mathsf{ct}_{\mathsf{main}} = (\boldsymbol{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c'_1, \ldots, c'_M)$, the hint-generation algorithm checks that

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

If the check fails, it outputs $\perp$. Otherwise, it samples $\mathbf{e}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z}, \sigma_{\mathsf{ht}}, \sqrt{\lambda}\sigma_{\mathsf{ht}}}$. It then outputs the hint

$$\mathbf{p}_i^\intercal = \mathbf{c}_1^\intercal \mathbf{K}_i + \mathbf{e}_{\mathsf{ht}}^\intercal \in \mathbb{Z}_q^m.$$

- $\mathsf{Recover}(\mathsf{ht}, \mathsf{hsk}, \mathsf{ct})$: On input a hint $\mathsf{ht} = \mathbf{p} \in \mathbb{Z}_q^m$, a helper decryption key $\mathsf{hsk} = (\mathbf{A}, \mathbf{v}, \mathbf{z}, \mathbf{u}, S, f)$, and a ciphertext $\mathsf{ct} = (\mathsf{ct}_{\mathsf{main}}, \pi_{\mathsf{ct}})$ where $\mathsf{ct}_{\mathsf{main}} = (\boldsymbol{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c'_1, \ldots, c'_M)$, the recovery algorithm proceeds as follows:
  - Let $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ be the binary representation of $\boldsymbol{\Psi}$, and let f' (Eq. (3.3)) and $\hat{f}$ (Fig. 1) be the functions associated with f as defined in Aggregate. Compute

$$\mathbf{h}_{\mathbf{A}, \hat{f}, \boldsymbol{\varphi}} = \mathsf{EvalFX}(\mathbf{A}, \hat{f}, \boldsymbol{\varphi}) \in \mathbb{Z}^{\hat{\ell} m} \quad \text{and} \quad \boldsymbol{\psi}_{f'} = \mathsf{EvalF}(\boldsymbol{\Psi}, f') \in \mathbb{Z}_q^{n+1}.$$

    Let $d_0 \in \mathbb{Z}_q$ be the $(n+1)^{\text{st}}$ component of $\boldsymbol{\psi}_{f'}$.
  - Compute

$$d_1 = \mathbf{c}_2^\intercal \mathbf{v} + \mathbf{c}_1^\intercal \mathbf{z} - \mathbf{p}^\intercal \mathbf{u} - \textstyle\sum_{j \in S} c'_j - \mathbf{c}_3^\intercal \mathbf{h}_{\mathbf{A}, \hat{f}, \boldsymbol{\varphi}} \in \mathbb{Z}_q. \tag{3.6}$$

  - Compute $d = d_0 - d_1 \in \mathbb{Z}_q$ and output 1 if $q/4 < d \le 3q/4$ and 0 otherwise.

Theorem 3.12 (Correctness). *Suppose $n \ge \lambda$, $m > 2(n+1)\log q$, $\sigma_{\mathsf{LWE}}, \sigma_{\mathsf{pp}}, \sigma_{\mathsf{agg}}, \sigma_{\mathsf{smudge}}, \sigma_{\mathsf{ht}} > 1$, and Π_{NIZK} is complete. Then, there exists*

$$q_{\mathsf{min}} = \tilde{O}(Q) \cdot m^{O(d \cdot \mathrm{polylog}(m))} \cdot \mathrm{poly}(s) \cdot \ell \cdot \sigma_{\mathsf{LWE}} \sigma_{\mathsf{pp}} \sigma_{\mathsf{agg}} \sigma_{\mathsf{smudge}} \sigma_{\mathsf{ht}}$$

such that for all $q > q_{\mathsf{min}}$, Construction 3.11 is correct.

Proof. Fix $\lambda, \tau, N \in \mathbb{N}$, $Q \le N$, an index $i \in [N]$, functions $f_1, \ldots, f_N \in \mathcal{F}_\tau$, and an input $\mathbf{x} \in \{0,1\}^\ell$. Take any $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ with $\mathsf{pp}_{\mathsf{com}} = (\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}, \mathbf{B})$ in the support of $\mathsf{Setup}(1^\lambda, 1^\tau, N)$ and any $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$. Fix any collection of public keys $\mathsf{pk}_j = \mathbf{P}_j \in \mathbb{Z}_q^{n \times m}$ for $j \ne i$. Let

$$(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N)).$$

Next, let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mathbf{x})$ and $\mathsf{ht} \leftarrow \mathsf{GetHint}(\mathsf{sk}_i, \mathsf{ct})$. We show that $\mathsf{Recover}(\mathsf{ht}, \mathsf{hsk}_i, \mathsf{ct}) = f_i(\mathbf{x})$:

- **Setup and public keys.** By definition of Setup, $\|\mathbf{R}_{\mathsf{com}}\| \le \sqrt{\lambda}\sigma_{\mathsf{pp}}$. By definition of KeyGen, $\mathsf{sk}_i = (\mathbf{K}_i, \mathsf{pp})$ where $\mathbf{K}_i \in \{0,1\}^{4m^5 \times m}$ and $\mathsf{pk}_i = \mathbf{P}_i = \mathbf{B}\mathbf{K}_i$.

- **Aggregation.** Write $\mathsf{hsk}_i = (\mathbf{A}, \mathbf{v}_i, \mathbf{z}_i, \mathbf{u}_i, S_i, f_i)$. By construction, $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1$ where $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ and the i^{th} column of $\mathbf{M}$ is

$$\mathbf{m}_i = \mathbf{P}_i\mathbf{u}_i - \mathbf{C}_0\mathbf{v}_i + \mathbf{a}_{\hat{f}_i} + \textstyle\sum_{j\in S_i}\mathbf{w}_j \quad \text{where} \quad \mathbf{a}_{\hat{f}_i} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i).$$

  Since $\mathbf{v}_i$ and $\mathbf{z}_i$ are the i^{th} columns of $\mathbf{V}_N = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N)$ and $\mathbf{Z} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$, respectively, Theorem 2.19 gives

$$\mathbf{C}\mathbf{v}_i = (\mathbf{C}_0 + \mathbf{C}_1)\mathbf{v}_i = \mathbf{C}_0\mathbf{v}_i + \mathbf{m}_i - \mathbf{B}\mathbf{z}_i = \mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \textstyle\sum_{j\in S_i}\mathbf{w}_j - \mathbf{B}\mathbf{z}_i, \tag{3.7}$$

  with $\|\mathbf{v}_i\| \le O(\sqrt{\lambda}\sigma_{\mathsf{pp}} m^4 \log q)$ and $\|\mathbf{z}_i\| \le O(\sqrt{\lambda}\sigma_{\mathsf{pp}} m^7 \log q \log N)$. From Eq. (3.5) and Definition 2.20, we have $\|\mathbf{u}_i\| \le \sqrt{\lambda}\sigma_{\mathsf{agg}}$.

- **Encryption.** Parse $\mathsf{ct} = (\mathsf{ct}_{\mathsf{main}}, \pi_{\mathsf{ct}})$, $\mathsf{ct}_{\mathsf{main}} = (\mathbf{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c'_1, \ldots, c'_M)$, and

$$\begin{aligned}
&\mathbf{\Psi} = \begin{bmatrix} \mathbf{B} \\ \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal \end{bmatrix} \mathbf{R}_{\mathbf{\Psi}} + \mathbf{x}^\intercal \otimes \mathbf{G}_{n+1} \\
&\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal, \quad \mathbf{c}_2^\intercal = \mathbf{s}^\intercal\mathbf{C} + \mathbf{e}^\intercal\mathbf{R}_{\mathbf{C}}, \quad \mathbf{c}_3^\intercal = \mathbf{s}^\intercal(\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}_n) + \mathbf{e}^\intercal\mathbf{R}_{\mathbf{A}}, \\
&c'_j = \mathbf{s}^\intercal\mathbf{w}_j + e_{\mathbf{w}_j},
\end{aligned}$$

  where $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ is the binary representation of $\mathbf{\Psi}$. By construction, we have $\|\mathbf{R}_{\mathbf{C}}\|, \|\mathbf{R}_{\mathbf{A}}\|, \|\mathbf{R}_{\mathbf{\Psi}}\| \le 1$ and $\|\mathbf{e}\| \le \sqrt{\lambda}\sigma_{\mathsf{LWE}}$ and $|e_{\mathbf{w}_j}| \le \sqrt{\lambda}\sigma_{\mathsf{smudge}}$. By completeness of Π_{NIZK}, we have

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

- **Hint generation.** By definition of GetHint, it first samples $\mathbf{e}_{\mathsf{ht}} \in \mathbb{Z}^m$ where $\|\mathbf{e}_{\mathsf{ht}}\| \le \sqrt{\lambda}\sigma_{\mathsf{ht}}$. Then, it outputs

$$\mathsf{ht}_i = \mathbf{p}_i^\intercal = \mathbf{c}_1^\intercal\mathbf{K}_i + \mathbf{e}_{\mathsf{ht}}^\intercal = \mathbf{s}^\intercal\mathbf{B}\mathbf{K}_i + \mathbf{e}^\intercal\mathbf{K}_i + \mathbf{e}_{\mathsf{ht}}^\intercal = \mathbf{s}^\intercal\mathbf{P}_i + \mathbf{e}^\intercal\mathbf{K}_i + \mathbf{e}_{\mathsf{ht}}^\intercal.$$

- **Recovery.** Consider the values d_0, d_1, d computed by $\mathsf{Recover}(\mathsf{ht}_i, \mathsf{hsk}_i, \mathsf{ct})$. By Eq. (3.7) we have

$$\begin{aligned}
\mathbf{c}_2^\intercal\mathbf{v}_i + \mathbf{c}_1^\intercal\mathbf{z}_i &= \mathbf{s}^\intercal\mathbf{C}\mathbf{v}_i + \mathbf{s}^\intercal\mathbf{B}\mathbf{z}_i + \mathbf{e}^\intercal\mathbf{R}_{\mathbf{C}}\mathbf{v}_i + \mathbf{e}^\intercal\mathbf{z}_i \\
&= \mathbf{s}^\intercal\big(\mathbf{P}_i\mathbf{u}_i + \mathbf{a}_{\hat{f}_i} + \textstyle\sum_{j\in S_i}\mathbf{w}_j - \mathbf{B}\mathbf{z}_i\big) + \mathbf{s}^\intercal\mathbf{B}\mathbf{z}_i + \mathbf{e}^\intercal\mathbf{R}_{\mathbf{C}}\mathbf{v}_i + \mathbf{e}^\intercal\mathbf{z}_i \\
&= \mathbf{s}^\intercal\mathbf{P}_i\mathbf{u}_i + \mathbf{s}^\intercal\mathbf{a}_{\hat{f}_i} + \textstyle\sum_{j\in S_i}\mathbf{s}^\intercal\mathbf{w}_j + \varepsilon_1,
\end{aligned}$$

  where $\varepsilon_1 = \mathbf{e}^\intercal\mathbf{R}_{\mathbf{C}}\mathbf{v}_i + \mathbf{e}^\intercal\mathbf{z}_i$. Next, we can write

$$\mathbf{p}_i^\intercal\mathbf{u}_i = \mathbf{s}^\intercal\mathbf{P}_i\mathbf{u}_i + \varepsilon_2 \quad \text{where} \quad \varepsilon_2 = (\mathbf{e}^\intercal\mathbf{K}_i + \mathbf{e}_{\mathsf{ht}}^\intercal)\mathbf{u}_i.$$

  We can also write

$$\textstyle\sum_{j\in S_i} c'_j = \sum_{j\in S_i}\mathbf{s}^\intercal\mathbf{w}_j + \varepsilon_3 \quad \text{where} \quad \varepsilon_3 = \sum_{j\in S_i} e_{\mathbf{w}_j}.$$

  Since $\mathbf{a}_{\hat{f}_i} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i)$ and $\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} = \mathsf{EvalFX}(\mathbf{A}, \hat{f}_i, \boldsymbol{\varphi})$, Theorem 2.15 gives $(\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}_n)\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} = \mathbf{a}_{\hat{f}_i} - \hat{f}_i(\boldsymbol{\varphi})$, so

$$\mathbf{c}_3^\intercal\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} = \mathbf{s}^\intercal\mathbf{a}_{\hat{f}_i} - \mathbf{s}^\intercal\hat{f}_i(\boldsymbol{\varphi}) + \varepsilon_4 \quad \text{where} \quad \varepsilon_4 = \mathbf{e}^\intercal\mathbf{R}_{\mathbf{A}}\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}}.$$

  Substituting into the definition of d_1 from Eq. (3.6), we now have

$$\begin{aligned}
d_1 &= \mathbf{c}_2^\intercal\mathbf{v}_i + \mathbf{c}_1^\intercal\mathbf{z}_i - \mathbf{p}_i^\intercal\mathbf{u}_i - \textstyle\sum_{j\in S_i} c'_j - \mathbf{c}_3^\intercal\mathbf{h}_{\mathbf{A},\hat{f}_i,\boldsymbol{\varphi}} \\
&= \mathbf{s}^\intercal\mathbf{P}_i\mathbf{u}_i + \mathbf{s}^\intercal\mathbf{a}_{\hat{f}_i} + \textstyle\sum_{j\in S_i}\mathbf{s}^\intercal\mathbf{w}_j - \mathbf{s}^\intercal\mathbf{P}_i\mathbf{u}_i - \sum_{j\in S_i}\mathbf{s}^\intercal\mathbf{w}_j - \mathbf{s}^\intercal\mathbf{a}_{\hat{f}_i} + \mathbf{s}^\intercal\hat{f}_i(\boldsymbol{\varphi}) + \varepsilon_1 - \varepsilon_2 - \varepsilon_3 - \varepsilon_4 \\
&= \mathbf{s}^\intercal\hat{f}_i(\boldsymbol{\varphi}) + \varepsilon_1 - \varepsilon_2 - \varepsilon_3 - \varepsilon_4.
\end{aligned} \tag{3.8}$$

Next, consider $d_0 \in \mathbb{Z}_q$. By construction, d_0 is the $(n+1)^{\text{st}}$ component of $\boldsymbol{\psi}_{f_i'} = \mathsf{EvalF}(\boldsymbol{\Psi}, f_i')$. Let $\bar{\mathbf{s}}^\mathsf{T} = [-\mathbf{s}^\mathsf{T} \mid 1]$. Since $\boldsymbol{\Psi} - \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1} = \begin{bmatrix} \mathbf{B} \\ \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T} \end{bmatrix} \mathbf{R}_{\boldsymbol{\Psi}}$, we have $\bar{\mathbf{s}}^\mathsf{T}(\boldsymbol{\Psi} - \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1}) = \mathbf{e}^\mathsf{T}\mathbf{R}_{\boldsymbol{\Psi}}$. Writing $\mathbf{h}_{\boldsymbol{\Psi}, f_i', \mathbf{x}} = \mathsf{EvalFX}(\boldsymbol{\Psi}, f_i', \mathbf{x})$ and using Theorem 2.15, we have

$$(\boldsymbol{\Psi} - \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1})\mathbf{h}_{\boldsymbol{\Psi}, f_i', \mathbf{x}} = \boldsymbol{\psi}_{f_i'} - f_i'(\mathbf{x}) = \boldsymbol{\psi}_{f_i'} - f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor \cdot \boldsymbol{\eta}_{n+1}.$$

Since $\boldsymbol{\eta}_{n+1}$ is the $(n+1)^{\text{st}}$ basis vector, this means

$$\mathbf{e}^\mathsf{T}\mathbf{R}_{\boldsymbol{\Psi}}\mathbf{h}_{\boldsymbol{\Psi}, f_i', \mathbf{x}} = \bar{\mathbf{s}}^\mathsf{T}(\boldsymbol{\Psi} - \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1})\mathbf{h}_{\boldsymbol{\Psi}, f_i', \mathbf{x}} = \bar{\mathbf{s}}^\mathsf{T}\boldsymbol{\psi}_{f_i'} - f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor .$$

Thus, we can now write

$$\bar{\mathbf{s}}^\mathsf{T}\boldsymbol{\psi}_{f_i'} = f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor + \varepsilon_5 \quad \text{where} \quad \varepsilon_5 = \mathbf{e}^\mathsf{T}\mathbf{R}_{\boldsymbol{\Psi}}\mathbf{h}_{\boldsymbol{\Psi}, f_i', \mathbf{x}}.$$

By definition, $\hat{f}_i(\boldsymbol{\varphi}) = \bar{\boldsymbol{\psi}}_{f_i'}$ is the first n components of $\boldsymbol{\psi}_{f_i'}$ and d_0 is the last component of $\boldsymbol{\psi}_{f_i'}$. This means $\bar{\mathbf{s}}^\mathsf{T}\boldsymbol{\psi}_{f_i'} = d_0 - \mathbf{s}^\mathsf{T}\hat{f}_i(\boldsymbol{\varphi})$ and

$$d_0 = \bar{\mathbf{s}}^\mathsf{T}\boldsymbol{\psi}_{f_i'} + \mathbf{s}^\mathsf{T}\hat{f}_i(\boldsymbol{\varphi}) = f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor + \varepsilon_5 + \mathbf{s}^\mathsf{T}\hat{f}_i(\boldsymbol{\varphi}).$$

Combined with Eq. (3.8), this means

$$d = d_0 - d_1 = f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor + \varepsilon_5 - \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4.$$

Let $\varepsilon^* = \varepsilon_5 - \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \varepsilon_4$. It suffices to bound $|\varepsilon^*|$.

- By construction, the function f_i' in Eq. (3.3) has depth $d + O(1)$ and size $s + O(n \log q)$. By Theorem 2.15, the function $\hat{f}_i$ from Fig. 1 has depth $\hat{d} = d \cdot \mathrm{polylog}(m)$ and size $\hat{s} = \mathrm{poly}(s, m)$. By Theorem 2.15, this means

$$\|\mathbf{h}_{\mathbf{A}, \hat{f}_i, \boldsymbol{\varphi}}\|, \|\mathbf{h}_{\boldsymbol{\Psi}, f_i', \mathbf{x}}\| \le m^{O(d \cdot \mathrm{polylog}(m))} \cdot \mathrm{poly}(s).$$

- From Corollary 2.10, we have that $M = \tilde{O}(Q)$. This means $|S_i| \le \tilde{O}(Q)$.
- Since $\mathbf{R}_\mathsf{C} \in \{0,1\}^{4m^5 \times m}$ and $\mathbf{K}_i \in \{0,1\}^{4m^5 \times m}$, we have $\|\mathbf{e}^\mathsf{T}\mathbf{R}_\mathsf{C}\|, \|\mathbf{e}^\mathsf{T}\mathbf{K}_i\| \le 4m^5 \cdot \sqrt{\lambda}\sigma_{\mathsf{LWE}}$. Taken together with the bounds on $\|\mathbf{v}_i\|, \|\mathbf{z}_i\|, \|\mathbf{u}_i\|, \|\mathbf{e}_{\mathsf{ht}}\|$ from above, we can now bound

$$\begin{aligned}
|\varepsilon_1| &\le \lambda \cdot \sigma_{\mathsf{LWE}}\sigma_{\mathsf{pp}} \cdot \log q \log N \cdot m^{O(1)} \\
|\varepsilon_2| &\le \lambda \cdot \sigma_{\mathsf{agg}}(\sigma_{\mathsf{LWE}} + \sigma_{\mathsf{ht}}) \cdot m^{O(1)} \\
|\varepsilon_3| &\le \sqrt{\lambda} \cdot \sigma_{\mathsf{smudge}} \cdot \tilde{O}(Q) \\
|\varepsilon_4|, |\varepsilon_5| &\le \sqrt{\lambda} \cdot \sigma_{\mathsf{LWE}}\ell \cdot m^{O(d \cdot \mathrm{polylog}(m))} \cdot \mathrm{poly}(s).
\end{aligned}$$

Since each of the width parameters $\sigma_{\mathsf{LWE}}, \sigma_{\mathsf{pp}}, \sigma_{\mathsf{agg}}, \sigma_{\mathsf{smudge}}, \sigma_{\mathsf{ht}}$ is at least 1 and using the fact that $m > 2(n+1)\log q > \lambda$, this means

$$|\varepsilon^*| \le \tilde{O}(Q) \cdot m^{O(d \cdot \mathrm{polylog}(m))} \cdot \mathrm{poly}(s) \cdot \ell \cdot \sigma_{\mathsf{LWE}}\sigma_{\mathsf{pp}}\sigma_{\mathsf{agg}}\sigma_{\mathsf{smudge}}\sigma_{\mathsf{ht}}.$$

Taking q_{min} to be four times this bound, we have $|\varepsilon^*| < q/4$ for all $q > q_{\mathsf{min}}$. Finally, $d = f_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor + \varepsilon^*$ with $|\varepsilon^*| < q/4$. If $f_i(\mathbf{x}) = 0$, then $|d| < q/4$ and Recover outputs 0; if $f_i(\mathbf{x}) = 1$, then $q/4 < d \le 3q/4$ and Recover outputs 1. In both cases $\mathsf{Recover}(\mathsf{ht}, \mathsf{hsk}_i, \mathsf{ct}) = f_i(\mathbf{x})$ and correctness holds. □

Theorem 3.13 (Security). *Suppose Π_{DGS} is a ρ-explainable discrete Gaussian sampler (Definition 2.20), Π_{NIZK} is zero-knowledge and simulation-sound extractable (Definition 2.1), and that the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption holds with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$. Suppose also that the following constraints hold:*

- *Lattice parameters: $n \ge \lambda$, $m \ge 3n \log q$, and $q > 2$ is prime.*

- *Width parameters:* $\sigma_{\mathsf{pp}} > \log m,\ m \log n < \sigma_{\mathsf{agg}} = \mathsf{poly}(\lambda, d),\ \sigma_{\mathsf{ht}} \geq 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} \cdot \lambda^{\omega(1)}$, *and*

$$\sigma_{\mathsf{smudge}} \geq \lambda^{\omega(1)} \cdot \tilde{O}(Q\ell) \cdot m^{O(d \cdot \mathrm{polylog}(m))} \cdot \mathsf{poly}(s) \cdot \sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}}\sigma_{\mathsf{agg}}.$$

Then, Construction 3.11 satisfies input-selective static CCA security in the semi-malicious model and modeling the hash functions H_1, H_2, H_3 as random oracles (Remark 3.2).

Proof. We start by defining the simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{ro}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{ct}}, \mathcal{S}_{\mathsf{ht}})$.[5]

- $\mathcal{S}_{\mathsf{pp}}(1^\lambda, 1^\tau, N, C)$: On input the security parameter λ, the function-family parameter τ, a slot count N, and a set of corrupted slots $C \subseteq [N]$, the simulator samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$, $\mathbf{W}_{\mathsf{com}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3}$, $\mathbf{R}_{\mathsf{com}} \leftarrow \bar{D}^{m \times 2m^3}_{\mathbb{Z}, \sigma_{\mathsf{pp}}, \sqrt{\lambda}\sigma_{\mathsf{pp}}}$, and sets

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

  The simulator initializes $\mathsf{ctr} = 0$ and initializes empty dictionaries $\mathsf{D}_{\mathsf{key}}, \mathsf{D}_{\mathsf{agg}}, \mathsf{D}_{\mathsf{ro}}$. The simulator samples $\mathbf{y} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{4m^5}$, $\mathbf{\Psi} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{(n+1) \times \ell m}$, and outputs the public parameters pp together with the initial state $\mathsf{st}_{\mathcal{S}}$, where

$$\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}}) \quad \text{and} \quad \mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ctr}, \mathsf{D}_{\mathsf{key}}, \mathsf{D}_{\mathsf{agg}}, \mathsf{D}_{\mathsf{ro}}, \mathbf{y}, \mathbf{\Psi}).$$

- $\mathcal{S}_{\mathsf{ro}}(\mathsf{st}_{\mathcal{S}}, \gamma)$: On input a state $\mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ctr}, \mathsf{D}_{\mathsf{key}}, \mathsf{D}_{\mathsf{agg}}, \mathsf{D}_{\mathsf{ro}}, \mathbf{y}, \mathbf{\Psi})$ and an oracle input $\gamma \in \{0,1\}^*$ (which specifies an oracle among H_1, H_2, H_3), the simulator maintains the dictionary D_{ro} of responses. By default, whenever H_1 (resp., H_2, H_3) is queried on an input that is not already present in D_{ro}, the simulator samples a uniformly random value in $\mathbb{Z}_q^n$ (resp., $\{0,1\}$, $\{0,1\}^\rho$) and records the input-output pair in D_{ro}. On a previously-seen input, the simulator always responds with the existing value in D_{ro}. Throughout, the simulator also consults the same dictionary D_{ro} whenever it needs to evaluate the random oracle itself. In addition, when the query is made, the simulator first checks the following conditions:

  (i) The query is of the form $H_1(\xi^*, \cdot)$, $H_2(\xi^*, \cdot)$, or $H_3(\xi^*, \cdot)$ for some string $\xi^* \in \{0,1\}^*$, and none of the preceding random-oracle queries have first input ξ^*.

  (ii) The string ξ^* is of the form $\xi^* = (\mathsf{pp}, Q, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*))$, where $|C| \leq Q \leq N$, and moreover, $\mathsf{pk}_i^* \in \mathbb{Z}_q^{n \times m}$ and $f_i^* \in \mathcal{F}_\tau$ for all $i \in [N]$.

  (iii) For every $i \in [N] \setminus C$, there exists a counter $c \in [\mathsf{ctr}]$ such that pk_i^* is the public key stored in $\mathsf{D}_{\mathsf{key}}[c]$.

  If any check fails, the simulator replies with the value determined above together with the current state $\mathsf{st}_{\mathcal{S}}$. If all checks pass, the simulator adds $\xi^* \mapsto \mathsf{st}$ to $\mathsf{D}_{\mathsf{agg}}$, where st is an empty state that will be populated with the components generated below. The simulator then computes $\mathsf{pp}_{\mathsf{CFF}} = H_2(\xi^*, 1) \| \cdots \| H_2(\xi^*, \rho_{\mathsf{CFF}})$ and for each $i \in [N]$, sets $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i) \subseteq [M]$. The simulator checks that for all $i \in C$, there exists an index $j_i^* \in S_i \setminus \bigcup_{i' \in C \setminus \{i\}} S_{i'}$. Define j_i^* to be the smallest such index if there are multiple. If no such index exists for some $i \in C$, the simulator outputs $\perp$. Otherwise, the simulator does the following:

  - Sample $\mathbf{R}_{\mathbf{C}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$, $\mathbf{R}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times \hat{\ell} m}$, and $\mathbf{r}_{\mathbf{w}_j} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$ for each $j \in [M] \setminus \{j_i^*\}_{i \in C}$. Let $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ be the binary representation of $\mathbf{\Psi}$, and set $\mathbf{A} = \mathbf{B}\mathbf{R}_{\mathbf{A}} + \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}_n$.
  - Sample $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ for all $i \in [N]$, and set $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_N]$, $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$, $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_{\mathbf{C}} - \mathbf{C}_1$.
  - For each $i \in [N]$, let f_i' and $\hat{f}_i^*$ be the functions associated with f_i^* (Eq. (3.3) and Fig. 1), let $\mathbf{a}_{\hat{f}_i^*} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i^*)$, let $\mathbf{v}_i$ be the i^{th} column of $\mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N)$, and parse $\mathsf{pk}_i^* = \mathbf{P}_i^*$.

[5] We adopt the convention that whenever a simulator algorithm fails, it outputs $\perp$ in place of *both* its output and the updated state $\mathsf{st}_{\mathcal{S}}$. Any subsequent simulator algorithm that receives $\mathsf{st}_{\mathcal{S}} = \perp$ as input immediately outputs $\perp$ without performing any further computation. In the security experiment, this means the experiment halts and outputs 0. This is distinct from $\mathcal{S}_{\mathsf{ht}}$ replying with a hint of $\perp$ on a ciphertext whose proof does not verify, which mirrors the behavior of GetHint and leaves $\mathsf{st}_{\mathcal{S}}$ unchanged.

– For each index $j \in [M] \setminus \{j_i^*\}_{i \in C}$ set $\mathbf{w}_j = \mathbf{Br}_{\mathbf{w}_j}$. The simulator programs

$$[H_1(\xi^*, (1,1)) \mid \cdots \mid H_1(\xi^*, (1, \hat{\ell}m))] \coloneqq \mathbf{A}$$
$$[H_1(\xi^*, (3,1)) \mid \cdots \mid H_1(\xi^*, (3,m))] \coloneqq \mathbf{C}_0,$$

and $H_1(\xi^*, (2,j)) \coloneqq \mathbf{w}_j$ for each $j \in [M] \setminus \{j_i^*\}_{i \in C}$.

– For each $i \in C$ set $\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi^*, i))$ and compute

$$\mathbf{w}_{j_i^*} = \mathbf{m}_i - \mathbf{P}_i^* \mathbf{u}_i + \mathbf{C}_0 \mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{w}_j.$$

The simulator then programs $H_1(\xi^*, (2, j_i^*)) \coloneqq \mathbf{w}_{j_i^*}$.

– For each $i \in [N] \setminus C$, let $c \in [\mathsf{ctr}]$ be the first counter where $\mathsf{D_{key}}[c] = (\mathbf{P}_i^*, \mathbf{T}_i^*)$ for some trapdoor $\mathbf{T}_i^*$. Then, the simulator sets

$$\mathbf{d}_i = \mathbf{m}_i + \mathbf{C}_0 \mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j \in S_i} \mathbf{w}_j,$$

computes $\mathbf{u}_i \leftarrow \mathsf{SamplePre}\big(\mathbf{P}_i^*, \mathbf{T}_i^*, \mathbf{d}_i, \sigma_{\mathsf{agg}}\big)$, sets $\mathbf{u}_i = \mathbf{T}_i^* \mathbf{G}_n^{-1}(\mathbf{d}_i)$ if $\|\mathbf{u}_i\| > \sqrt{\lambda}\sigma_{\mathsf{agg}}$, and programs $H_3(\xi^*, i) \coloneqq \mathsf{DGS.Explain}(1^\lambda, \mathbf{u}_i, \sigma_{\mathsf{agg}})$.

All of the quantities computed above are recorded in $\mathsf{D_{agg}}[\xi^*]$, and all programmed input-output pairs are recorded in $\mathsf{D_{ro}}$. The simulator then replies with the value determined above together with the updated state $\mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td_{NIZK}}, \mathsf{ctr}, \mathsf{D_{key}}, \mathsf{D_{agg}}, \mathsf{D_{ro}}, \mathbf{y}, \boldsymbol{\Psi})$.

- $\mathcal{S}_{\mathsf{kg}}(\mathsf{st}_{\mathcal{S}})$: On input a state $\mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td_{NIZK}}, \mathsf{ctr}, \mathsf{D_{key}}, \mathsf{D_{agg}}, \mathsf{D_{ro}}, \mathbf{y}, \boldsymbol{\Psi})$, the simulator increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $(\mathbf{P}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$, sets $\mathsf{pk} = \mathbf{P}$, and adds $\mathsf{ctr} \mapsto (\mathsf{pk}, \mathbf{T})$ to $\mathsf{D_{key}}$. It replies with $(\mathsf{pk}, \mathsf{st}_{\mathcal{S}})$, where $\mathsf{st}_{\mathcal{S}}$ now contains the incremented counter ctr and the updated dictionary $\mathsf{D_{key}}$.

- $\mathcal{S}_{\mathsf{ct}}(\mathsf{st}_{\mathcal{S}}, 1^Q, \{(c_i^*, f_i^*, \mathsf{pk}_i^*, y_i)\}_{i \in [N]}, \{(\mathsf{sk}_i^*, r_i)\}_{i \in C})$: On input a state $\mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td_{NIZK}}, \mathsf{ctr}, \mathsf{D_{key}}, \mathsf{D_{agg}}, \mathsf{D_{ro}}, \mathbf{y}, \boldsymbol{\Psi})$, a collusion bound 1^Q, a list of counter-function-key-output tuples $\{(c_i^*, f_i^*, \mathsf{pk}_i^*, y_i)\}_{i \in [N]}$, and a list of pairs $\{(\mathsf{sk}_i^*, r_i)\}_{i \in C}$, the simulator checks that $Q \le N$, $|C| \le Q$, $f_i^* \in \mathcal{F}_\tau$, $c_i^* = \perp$ for all $i \in C$, $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$ for all $i \notin C$, and halts with output $\perp$ otherwise. For each $i \in [N]$, the simulator then checks the following:

  – If $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$, the simulator looks up $(\mathsf{pk}', \mathbf{T}') = \mathsf{D_{key}}[c_i^*]$ and halts with output $\perp$ if $\mathsf{pk}_i^* \neq \mathsf{pk}'$.

  – If $c_i^* = \perp$, the simulator halts with output $\perp$ if $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) \neq \mathsf{KeyGen}(\mathsf{pp}; r_i)$.

  If the checks pass, the simulator sets

$$\xi_{\mathsf{chal}}^* = (\mathsf{pp}, Q, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*)).$$

Namely, ξ_{chal}^* is the specific aggregation string associated with the adversary's challenge query. The simulator checks that ξ_{chal}^* is contained in $\mathsf{D_{agg}}$, and halts with output $\perp$ if not. If the check passes, then the simulator parses $\mathsf{sk}_i^* = \mathbf{K}_i^* \in \{0,1\}^{4m^5 \times m}$ for all $i \in C$, samples $e_{\mathbf{w}_1}, \ldots, e_{\mathbf{w}_M} \leftarrow \bar{D}_{\mathbb{Z}, \sigma_{\mathsf{smudge}}, \sqrt{\lambda}\sigma_{\mathsf{smudge}}}$, and sets

$$\begin{aligned}
\mathbf{c}_1^\mathsf{T} &= \mathbf{y}^\mathsf{T} \\
\mathbf{c}_2^\mathsf{T} &= \mathbf{y}^\mathsf{T} \mathbf{R}_\mathbf{C} \\
\mathbf{c}_3^\mathsf{T} &= \mathbf{y}^\mathsf{T} \mathbf{R}_\mathbf{A} \\
c_j' &= \mathbf{y}^\mathsf{T} \mathbf{r}_{\mathbf{w}_j} + e_{\mathbf{w}_j} \qquad \forall j \in [M] \setminus \left\{j_i^*\right\}_{i \in C},
\end{aligned}$$

where the components multiplying $\mathbf{y}^\mathsf{T}$ are retrieved from $\mathsf{D_{agg}}[\xi_{\mathsf{chal}}^*]$. For each $i \in C$, the simulator computes $\underline{\psi}_{f_i'} = \boldsymbol{\eta}_{n+1}^\mathsf{T} \mathsf{EvalF}(\boldsymbol{\Psi}, f_i')$ and $\mathbf{h}_{\mathbf{A}, \hat{f}_i^*, \boldsymbol{\varphi}} = \mathsf{EvalFX}(\mathbf{A}, \hat{f}_i^*, \boldsymbol{\varphi})$. It then sets $\mathbf{z}_i \in \mathbb{Z}^{4m^5}$ to be the i^{th} column of $\mathsf{Open}(\mathsf{pp_{com}}, \mathbf{M})$ and computes the remaining ciphertext components as

$$c_{j_i^*}' = \mathbf{y}^\mathsf{T} \Big(\mathbf{z}_i - \mathbf{K}_i^* \mathbf{u}_i + \mathbf{R}_\mathbf{C} \mathbf{v}_i - \mathbf{R}_\mathbf{A} \mathbf{h}_{\mathbf{A}, \hat{f}_i^*, \boldsymbol{\varphi}} - \textstyle\sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{r}_{\mathbf{w}_j} \Big) + y_i \cdot \lfloor q/2 \rfloor - \underline{\psi}_{f_i'} + e_{\mathbf{w}_{j_i^*}} \in \mathbb{Z}_q.$$

Finally, the simulator sets $\mathsf{ct}^*_{\mathsf{main}} = (\boldsymbol{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c'_1, \ldots, c'_M)$ and computes a simulated proof

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

The simulator outputs $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{main}}, \pi^*_{\mathsf{ct}})$ and the updated state $\mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ctr}, \mathsf{D}_{\mathsf{key}}, \mathsf{D}_{\mathsf{agg}}, \mathsf{D}_{\mathsf{ro}}, \mathbf{y}, \boldsymbol{\Psi})$.

- $\mathcal{S}_{\mathsf{ht}}(\mathsf{st}_{\mathcal{S}}, c, \mathsf{ct})$: On input a state $\mathsf{st}_{\mathcal{S}} = (\mathsf{pp}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ctr}, \mathsf{D}_{\mathsf{key}}, \mathsf{D}_{\mathsf{agg}}, \mathsf{D}_{\mathsf{ro}}, \mathbf{y}, \boldsymbol{\Psi})$, a counter $c \le \mathsf{ctr}$, and a ciphertext ct, the simulator parses $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ and $\mathsf{ct} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = \big(\hat{\boldsymbol{\Psi}}, \hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{\mathbf{c}}_3^\intercal, \hat{c}'_1, \ldots, \hat{c}'_M\big)$. Then, the simulator checks that

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

If not, the simulator responds with the hint $\perp$ together with the unchanged state $\mathsf{st}_{\mathcal{S}}$. If the proof verifies, the simulator computes

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

If $\|\tilde{\mathbf{e}}\| > \sqrt{\lambda}\sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \tilde{\mathbf{s}}^\intercal\mathbf{B} + \tilde{\mathbf{e}}^\intercal$, then the simulator halts with output $\perp$. Otherwise, the simulator looks up $(\mathsf{pk}, \mathbf{T}) = \mathsf{D}_{\mathsf{key}}[c]$, parses $\mathsf{pk} = \mathbf{P}$, samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z}, \sigma_{\mathsf{ht}}, \sqrt{\lambda}\sigma_{\mathsf{ht}}}$, and replies with $\tilde{\mathbf{s}}^\intercal\mathbf{P} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$ together with the (unchanged) state $\mathsf{st}_{\mathcal{S}}$.

Take now any efficient adversary $\mathcal{A}$ for the input-selective static security game. We show that the real distribution and the simulated distribution in Definition 3.1 are computationally indistinguishable. In the security analysis, we model all three hash functions H_1, H_2, H_3 as random oracles. Since $\mathcal{A}$ is efficient, there are polynomials $N = N(\lambda)$ and $Q_{\mathsf{ro}} = Q_{\mathsf{ro}}(\lambda)$ that bound the number of keys algorithm $\mathcal{A}$ outputs and the number of random oracle queries $\mathcal{A}$ makes, respectively. For ease of exposition, we also make the following simplifying assumptions on the behavior of the adversary $\mathcal{A}$:

- Algorithm $\mathcal{A}$ does not query H_1, H_2, or H_3 on the same input more than once.
- Prior to the challenge phase, algorithm $\mathcal{A}$ makes at least one random-oracle query to one of H_1, H_2, or H_3 whose first input is the tuple

  $$\xi^*_{\mathsf{chal}} = \big(\mathsf{pp}, Q, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (f_1, \ldots, f_N)\big)$$

  associated with the challenge query (specifically, the tuple ξ^*_{chal} that is constructed by the Aggregate algorithm in Eq. (3.4)). Specifically, this is a query of the form $H_1(\xi^*_{\mathsf{chal}}, \cdot)$, $H_2(\xi^*_{\mathsf{chal}}, \cdot)$, or $H_3(\xi^*_{\mathsf{chal}}, \cdot)$.

Both assumptions are without loss of generality since for any efficient algorithm $\mathcal{A}$ that does not conform to these requirements, we can construct an efficient wrapper algorithm $\mathcal{A}'$ that does satisfy the requirements and induces the same output distribution.

Hybrid experiments. We now define our sequence of hybrid experiments.

- Hyb_0: This is the input-selective static CCA security experiment in the semi-malicious model from Definitions 3.3, 3.6 and 3.10 corresponding to the case where $b = 0$ (i.e., the challenger answers the adversary's queries using the real algorithm). We give the formal specification below:
  - **Setup phase:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a function-family parameter 1^τ, a slot count N, a set of corrupted slots $C \subseteq [N]$, and a challenge input $\mathbf{x} \in \{0,1\}^\ell$. The challenger then samples

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

    It then sets

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

    and gives the public parameters $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} = 0$ and an empty dictionary $\mathsf{D}_{\mathsf{key}}$.

– **Pre-challenge query phase:** The adversary can now make random oracle queries, key-generation queries, and hint-generation queries. In the static security model, the adversary cannot make corruption queries. The challenger then responds as follows:

  * **Random-oracle queries:** The challenger maintains a table of its random-oracle responses. Whenever $\mathcal{A}$ queries H_1 (resp., H_2, H_3) on an input that is not already present in the table, the challenger samples a uniformly random value in $\mathbb{Z}_q^n$ (resp., $\{0,1\}$, $\{0,1\}^\rho$), records the input-output pair in the table, and replies with this value. On a previously-seen input it replies with the recorded value. The challenger consults the same table whenever it evaluates the random oracles itself (e.g., when constructing the challenge ciphertext), so that its own evaluations remain consistent with the values returned to $\mathcal{A}$.
  * **Key-generation queries:** When $\mathcal{A}$ makes a key-generation query, the challenger increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $\mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$ and computes $\mathbf{P} = \mathbf{BK} \in \mathbb{Z}_q^{n \times m}$. The challenger sets $\mathsf{pk} = \mathbf{P}$ and $\mathsf{sk} = (\mathbf{K}, \mathsf{pp})$ and adds $\mathsf{ctr} \mapsto (\mathsf{pk}, \mathsf{sk})$ to $\mathsf{D}_{\mathsf{key}}$. It replies to $\mathcal{A}$ with $(\mathsf{ctr}, \mathsf{pk})$.
  * **Hint-generation queries:** When $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ for $c \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\mathbf{\Psi}}, \hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{\mathbf{c}}_3^\intercal, \hat{c}'_1, \ldots, \hat{c}'_M)$ and looks up $(\mathsf{pk}, \mathsf{sk}) = \mathsf{D}_{\mathsf{key}}[c]$. The challenger parses $\mathsf{sk} = (\mathbf{K}, \mathsf{pp})$ and $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$. Then, it checks that

$$\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1. \tag{3.9}$$

    If the check fails, the challenger replies with $\perp$. If the check passes, the challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{ht}},\sqrt{\lambda}\sigma_{\mathsf{ht}}}$ and replies with $\hat{\mathbf{c}}_1^\intercal \mathbf{K} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$.

– **Challenge phase:** After the adversary is finished making pre-challenge queries, it outputs a collusion bound 1^Q with $Q \le N$. The challenger checks that $|C| \le Q$ and halts with output 0 otherwise. Then, for each slot $i \in [N]$, the adversary specifies a tuple $(c_i^*, f_i^*, \mathsf{pk}_i^*)$ where $f_i^* \in \mathcal{F}_\tau$, $c_i^* = \perp$ for all $i \in C$, and $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$ for all $i \notin C$. For each $i \in C$, the adversary additionally provides (sk_i^*, r_i) such that $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) = \mathsf{KeyGen}(\mathsf{pp}; r_i)$. For each $i \in [N]$, the challenger then checks the following:

  * If $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up $(\mathsf{pk}', \mathsf{sk}') = \mathsf{D}_{\mathsf{key}}[c_i^*]$ and halts with output 0 if $\mathsf{pk}_i^* \neq \mathsf{pk}'$.
  * If $c_i^* = \perp$, the challenger halts with output 0 if $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) \neq \mathsf{KeyGen}(\mathsf{pp}; r_i)$.

  The challenger then constructs the challenge ciphertext. Concretely, it parses $\mathsf{pk}_i^* = \mathbf{P}_i$ for each $i \in [N]$ and proceeds as follows:

  * Let $\hat{\ell} = (n+1)\ell m \lceil \log q \rceil$, and for each $i \in [N]$ let f_i' and $\hat{f}_i^*$ be the functions associated with f_i^*, as defined in Eq. (3.3) and Fig. 1.
  * Set $\xi_{\mathsf{chal}}^* = (\mathsf{pp}, Q, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*))$ and compute

$$\mathbf{A} = [H_1(\xi_{\mathsf{chal}}^*, (1,1)) \mid \cdots \mid H_1(\xi_{\mathsf{chal}}^*, (1, \hat{\ell}m))] \in \mathbb{Z}_q^{n \times \hat{\ell}m}.$$

    Then, for each $i \in [N]$, compute

$$\mathbf{a}_{\hat{f}_i^*} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i^*) \in \mathbb{Z}_q^n.$$

  * Let $\mathsf{CFF.Expand}$ be the local cover-free sampler from Corollary 2.10 with collusion bound Q, domain size N, and $\varepsilon = 2^{-\log^2(\lambda)} = \mathsf{negl}(\lambda)$. Let ρ_{CFF} be the associated public parameter size and M be the size of the output universe. Compute the public parameters $\mathsf{pp}_{\mathsf{CFF}} = H_2(\xi_{\mathsf{chal}}^*, 1) \| \cdots \| H_2(\xi_{\mathsf{chal}}^*, \rho_{\mathsf{CFF}})$. Then, for each $i \in [N]$, let $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i) \subseteq [M]$.
  * For each $j \in [M]$, compute a smudging term $\mathbf{w}_j = H_1(\xi_{\mathsf{chal}}^*, (2, j)) \in \mathbb{Z}_q^n$.
  * Compute $\mathbf{V}_N = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N) \in \mathbb{Z}^{m \times N}$ and for each $i \in [N]$, let $\mathbf{v}_i \in \mathbb{Z}^m$ be the i^{th} column of $\mathbf{V}_N$. Construct the matrix

$$\mathbf{C}_0 = [H_1(\xi_{\mathsf{chal}}^*, (3,1)) \mid \cdots \mid H_1(\xi_{\mathsf{chal}}^*, (3,m))] \in \mathbb{Z}_q^{n \times m}.$$

    For each $i \in [N]$, compute $\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi_{\mathsf{chal}}^*, i)) \in \mathbb{Z}^m$.

* Define the matrix

$$\mathbf{M} = \left[\mathbf{P}_1\mathbf{u}_1 - \mathbf{C}_0\mathbf{v}_1 + \mathbf{a}_{\hat{f}_1^*} + \sum_{j\in S_1}\mathbf{w}_j \mid \cdots \mid \mathbf{P}_N\mathbf{u}_N - \mathbf{C}_0\mathbf{v}_N + \mathbf{a}_{\hat{f}_N^*} + \sum_{j\in S_N}\mathbf{w}_j\right] \in \mathbb{Z}_q^{n\times N},$$

and compute the commitment $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$. Let $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1$.

* Next, sample

$$\begin{aligned}
\mathbf{s} &\xleftarrow{R} \mathbb{Z}_q^n \\
\mathbf{e} \leftarrow \bar{D}^{4m^5}_{\mathbb{Z},\sigma_{\mathsf{LWE}},\sqrt{\lambda}\sigma_{\mathsf{LWE}}}&,\ e_{\mathbf{w}_1},\ldots,e_{\mathbf{w}_M} \leftarrow \bar{D}_{\mathbb{Z},\sigma_{\mathsf{smudge}},\sqrt{\lambda}\sigma_{\mathsf{smudge}}} \\
\mathbf{R}_{\mathbf{\Psi}} \xleftarrow{R} \{0,1\}^{4m^5\times \ell m}&,\ \mathbf{R}_{\mathbf{C}} \xleftarrow{R} \{0,1\}^{4m^5\times m},\ \mathbf{R}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{4m^5\times \hat{\ell} m}.
\end{aligned}$$

Set

$$\mathbf{\Psi} = \begin{bmatrix}\mathbf{B}\\ \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal\end{bmatrix}\mathbf{R}_{\mathbf{\Psi}} + \mathbf{x}^\intercal \otimes \mathbf{G}_{n+1} \in \mathbb{Z}_q^{(n+1)\times \ell m}.$$

Let $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ be the binary representation of $\mathbf{\Psi}$. Set

$$\begin{aligned}
\mathsf{ct}^*_{\mathsf{main}} &= \left(\mathbf{\Psi},\ \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal,\ \mathbf{s}^\intercal\mathbf{C} + \mathbf{e}^\intercal\mathbf{R}_{\mathbf{C}},\ \mathbf{s}^\intercal(\mathbf{A} - \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}_n) + \mathbf{e}^\intercal\mathbf{R}_{\mathbf{A}},\ \mathbf{s}^\intercal\mathbf{w}_1 + e_{\mathbf{w}_1},\ldots,\mathbf{s}^\intercal\mathbf{w}_M + e_{\mathbf{w}_M}\right) \\
&= \left(\mathbf{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c'_1,\ldots,c'_M\right)
\end{aligned}$$

and compute the proof

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

The challenger gives the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{main}}, \pi^*_{\mathsf{ct}})$ to $\mathcal{A}$.

– **Post-challenge query phase:** The adversary can continue to make random oracle queries and hint-generation queries in the post-challenge phase subject to the restriction that its hint-generation queries $(\hat{\mathsf{ct}}, c)$ cannot be on the challenge ciphertext (i.e., $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$). The challenger answers these queries exactly as in the pre-challenge phase.

– **Output phase:** At the end of the experiment, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. If $\mathcal{A}$ aborts without outputting a bit, the output is 0.

- Hyb_1: Same as Hyb_0 except the challenger replaces the NIZK proof in the challenge ciphertext with a simulated proof:

  – During the setup phase, the challenger now 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 computes the proof π^*_{ct} as

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

- Hyb_2: Same as Hyb_1 except the challenger extracts the encryption randomness underlying the ciphertexts when answering hint-generation queries. Specifically, when $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$, the challenger first parses $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ and replies with $\perp$ if $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) \neq 1$. If the proof verifies, the challenger additionally extracts

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

and halts with output 0 if $\|\tilde{\mathbf{e}}\| > \sqrt{\lambda}\sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \tilde{\mathbf{s}}^\intercal\mathbf{B} + \tilde{\mathbf{e}}^\intercal$. If the checks pass, then the challenger answers the query as in Hyb_1.

- Hyb_3: Same as Hyb_2 except when answering key-generation queries, the challenger now adds the mapping $\mathsf{ctr} \mapsto (\mathsf{pk}, \perp)$ to $\mathsf{D}_{\mathsf{key}}$ (instead of $\mathsf{ctr} \mapsto (\mathsf{pk}, \mathsf{sk})$). When answering hint-generation queries, the challenger now uses the extracted randomness. In particular, for a hint-generation query $(\hat{\mathsf{ct}}, c)$ where $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$, the challenger first checks the proof $\hat{\pi}_{\mathsf{ct}}$ (i.e., by checking Eq. (3.9)). If the proof verifies, then the challenger extracts $(\tilde{\mathbf{s}}, \tilde{\mathbf{e}})$ as in Hyb_2. Next it looks up $(\mathsf{pk}, \perp) = \mathsf{D}_{\mathsf{key}}[c]$ and parses $\mathsf{pk} = \mathbf{P} \in \mathbb{Z}_q^{n \times m}$. The challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z}, \sigma_{\mathsf{ht}}, \sqrt{\lambda}\sigma_{\mathsf{ht}}}$ and replies with $\tilde{\mathbf{s}}^\intercal \mathbf{P} + \hat{\mathbf{e}}^\intercal_{\mathsf{ht}}$.

  At this point, nothing in the experiment depends on the secret keys for the honest parties. For completeness, we provide a more detailed description of the challenger's behavior in this hybrid here:

  - **Setup phase:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a function-family parameter 1^τ, a slot count N, a set of corrupted slots $C \subseteq [N]$, and a challenge input $\mathbf{x} \in \{0,1\}^\ell$. The challenger samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$, $\mathbf{W}_{\mathsf{com}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3}$, and $\mathbf{R}_{\mathsf{com}} \leftarrow \bar{D}^{m \times 2m^3}_{\mathbb{Z}, \sigma_{\mathsf{pp}}, \sqrt{\lambda}\sigma_{\mathsf{pp}}}$. It then sets

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

    and gives the public parameters $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ to $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an empty dictionary $\mathsf{D}_{\mathsf{key}}$.
  - **Pre-challenge query phase:** The challenger responds to the adversary's queries as follows:
    - **Random-oracle queries:** The challenger maintains a table of its random-oracle responses. Whenever $\mathcal{A}$ queries H_1 (resp., H_2, H_3) on an input that is not already present in the table, the challenger samples a uniformly random value in $\mathbb{Z}_q^n$ (resp., $\{0,1\}$, $\{0,1\}^\rho$), records the input-output pair in the table, and replies with this value. On a previously-seen input it replies with the recorded value. The challenger consults the same table whenever it evaluates the random oracles itself (e.g., when constructing the challenge ciphertext), so that its own evaluations remain consistent with the values returned to $\mathcal{A}$.
    - **Key-generation queries:** When $\mathcal{A}$ makes a key-generation query, the challenger increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $\mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$, and computes $\mathbf{P} = \mathbf{B}\mathbf{K}$. It sets $\mathsf{pk} = \mathbf{P}$, adds $\mathsf{ctr} \mapsto (\mathsf{pk}, \perp)$ to $\mathsf{D}_{\mathsf{key}}$, and replies to $\mathcal{A}$ with $(\mathsf{ctr}, \mathsf{pk})$.
    - **Hint-generation queries:** When $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ for $c \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\mathbf{\Psi}}, \hat{\mathbf{c}}_1^\intercal, \hat{\mathbf{c}}_2^\intercal, \hat{\mathbf{c}}_3^\intercal, \hat{c}'_1, \ldots, \hat{c}'_M)$. Then, the challenger checks that

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

      If not, the challenger responds with $\perp$. If the proof verifies, the challenger computes

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

      If $\|\tilde{\mathbf{e}}\| > \sqrt{\lambda}\sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \tilde{\mathbf{s}}^\intercal \mathbf{B} + \tilde{\mathbf{e}}^\intercal$, then the challenger halts with output 0. Otherwise, the challenger looks up $(\mathsf{pk}, \perp) = \mathsf{D}_{\mathsf{key}}[c]$, parses $\mathsf{pk} = \mathbf{P}$, samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z}, \sigma_{\mathsf{ht}}, \sqrt{\lambda}\sigma_{\mathsf{ht}}}$, and replies with $\tilde{\mathbf{s}}^\intercal \mathbf{P} + \hat{\mathbf{e}}^\intercal_{\mathsf{ht}}$.
  - **Challenge phase:** The challenger constructs the challenge ciphertext $\mathsf{ct}^*_{\mathsf{main}}$ exactly as in the real experiment (Hyb_0), but now computes a *simulated* proof

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

    The rest of the challenge phase proceeds identically to that in Hyb_0.
  - **Post-challenge query phase:** The adversary may continue to make random-oracle and hint-generation queries, which the challenger answers exactly as in the pre-challenge phase, subject to the restriction that $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$ for each hint-generation query.
  - **Output phase:** At the end of the experiment, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment. If $\mathcal{A}$ aborts without outputting a bit, the output is 0.

- Hyb_4: Same as Hyb_3 except that when answering a key-generation query, instead of sampling $\mathbf{K} \xleftarrow{R} \{0,1\}^{4m^5 \times m}$ and setting $\mathbf{P} = \mathbf{BK}$, the challenger samples $\mathbf{P} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$ uniformly at random. As before, it increments ctr = ctr+1, sets pk = $\mathbf{P}$, adds ctr $\mapsto$ (pk, $\perp$) to $\mathsf{D}_{\mathsf{key}}$, and replies with (ctr, pk).

- Hyb_5: Same as Hyb_4 except that the challenger initializes a dictionary $\mathsf{D}_{\mathsf{agg}}$ at setup time. Moreover, when $\mathcal{A}$ makes a random-oracle query, the challenger performs the following additional checks:

  (i) The query is of the form $H_1(\xi^*, \cdot)$, $H_2(\xi^*, \cdot)$, or $H_3(\xi^*, \cdot)$ for some string $\xi^* \in \{0,1\}^*$, and none of the preceding random-oracle queries have first input ξ^*.

  (ii) The string ξ^* parses as $\xi^* = \big(\mathsf{pp}, Q, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*)\big)$, where $|C| \le Q \le N$, and moreover, $\mathsf{pk}_i^* \in \mathbb{Z}_q^{n \times m}$ and $f_i^* \in \mathcal{F}_\tau$ for all $i \in [N]$.

  (iii) For every $i \in [N] \setminus C$, there exists some counter $c \in [\mathsf{ctr}]$ such that pk_i^* is the public key the challenger sampled on the c^{th} key-generation query (i.e., pk_i^* is the public key stored in $\mathsf{D}_{\mathsf{key}}[c]$).

  If all checks pass, the challenger adds $\xi^* \mapsto \mathsf{st}$ to $\mathsf{D}_{\mathsf{agg}}$, where st is an empty state. Moreover, in the challenge phase, the challenger aborts if ξ^*_{chal} does not exist in $\mathsf{D}_{\mathsf{agg}}$. Throughout the proof, we refer to queries that pass the above three checks as *aggregation-valid* random oracle queries with prefix ξ^*.

- Hyb_6: Same as Hyb_5 except that, after the adversary makes an aggregation-valid random oracle query with prefix ξ^*, the challenger samples the cover-free set parameters associated with ξ^* and then tries to assign a unique index to each corrupted slot. Concretely, when answering this query, the challenger computes $\mathsf{pp}_{\mathsf{CFF}} = H_2(\xi^*, 1) \| \cdots \| H_2(\xi^*, \rho_{\mathsf{CFF}})$ and $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i) \subseteq [M]$ for each $i \in C$. Then, for each $i \in C$, the challenger attempts to find an index $j_i^* \in S_i$ such that $j_i^* \notin S_{i'}$ for every $i' \in C \setminus \{i\}$. If no such index exists for some $i \in C$, the challenger aborts with output 0.

- Hyb_7: Same as Hyb_6 except the challenger changes how it samples the vectors $\mathbf{u}_i$ for the honest slots when answering aggregation-valid random oracle queries. Specifically, after $\mathcal{A}$ makes an aggregation-valid random oracle query with prefix $\xi^* = \big(\mathsf{pp}, Q, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*)\big)$, the challenger proceeds as follows:

  - By Condition (i), this is the first random oracle query with prefix ξ^*, so the adversary has not queried for the values of $H_1(\xi^*, \cdot)$, $H_2(\xi^*, \cdot)$, and $H_3(\xi^*, \cdot)$ before. This means the challenger has the ability to "program" the values of the random oracles at these points.
  - For each $i \in [N]$, parse $\mathsf{pk}_i^* = \mathbf{P}_i^* \in \mathbb{Z}_q^{n \times m}$.
  - For each index $i \in [N] \setminus C$, the challenger now samples the vector $\mathbf{u}_i \leftarrow \bar{D}^m_{\mathbb{Z}, \sigma_{\mathsf{agg}}, \sqrt{\lambda}\sigma_{\mathsf{agg}}}$ and programs the value $H_3(\xi^*, i) := \mathsf{DGS.Explain}(1^\lambda, \mathbf{u}_i, \sigma_{\mathsf{agg}})$.

  From now on, the challenger adds all programming-related components to the state $\mathsf{D}_{\mathsf{agg}}[\xi^*]$ and refers to $\mathsf{D}_{\mathsf{agg}}[\xi^*_{\mathsf{chal}}]$ when using these components to construct the challenge ciphertext.

- Hyb_8: Same as Hyb_7 except the challenger samples $\mathbf{d}_i \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{u}_i \leftarrow (\mathbf{P}_i^*)^{-1}_{\sigma_{\mathsf{agg}}}(\mathbf{d}_i)$ for $i \in [N] \setminus C$ when answering aggregation-valid random oracle queries.

- Hyb_9: Same as Hyb_8 except when responding to key-generation queries, the challenger now samples $(\mathbf{P}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$. It sets $\mathsf{pk} = \mathbf{P} \in \mathbb{Z}_q^{n \times m}$ and adds ctr $\mapsto$ (pk, $\mathbf{T}$) to $\mathsf{D}_{\mathsf{key}}$.

- Hyb_{10}: Same as Hyb_9 except the challenger truncates $\mathbf{u}_i$ when answering aggregation-valid random oracle queries. In this experiment, for each $i \in [N] \setminus C$, the challenger first looks up the first counter $c \in [\mathsf{ctr}]$ where $\mathsf{D}_{\mathsf{key}}[c] = (\mathbf{P}_i^*, \mathbf{T}_i^*)$ for some trapdoor $\mathbf{T}_i^*$. Then, the challenger samples $\mathbf{u}_i \leftarrow (\mathbf{P}_i^*)^{-1}_{\sigma_{\mathsf{agg}}}(\mathbf{d}_i)$ and sets $\mathbf{u}_i = \mathbf{T}_i^* \mathbf{G}_n^{-1}(\mathbf{d}_i)$ if $\|\mathbf{u}_i\| > \sqrt{\lambda}\sigma_{\mathsf{agg}}$. Note that in this case, $\mathbf{P}_i^* \mathbf{u}_i = \mathbf{d}_i$ since $\mathbf{P}_i^* \mathbf{T}_i^* = \mathbf{G}_n$.

- Hyb_{11}: Same as Hyb_{10} except the challenger now samples $\mathbf{u}_i \leftarrow \mathsf{SamplePre}(\mathbf{P}_i^*, \mathbf{T}_i^*, \mathbf{d}_i, \sigma_{\mathsf{agg}})$ for $i \in [N] \setminus C$ when answering aggregation-valid random oracle queries.

- Hyb_{12}: Same as Hyb_{11} except the challenger changes how it samples $\mathbf{w}_{j_i^*} \in \mathbb{Z}_q^n$ for each slot $i \in C$ when answering aggregation-valid random oracle queries. Specifically, after $\mathcal{A}$ makes an aggregation-valid random oracle query with prefix ξ^* and the challenger checks the cover-free property, the challenger proceeds as follows:
  - As in Hyb_{11}, write $\xi^* = \left(\mathsf{pp}, Q, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (f_1^*, \ldots, f_N^*)\right)$ where $\mathsf{pk}_i^* = \mathbf{P}_i^*$. The challenger now defines the following quantities (in the same manner as Hyb_{11}):
    - Let $\mathbf{A} = [H_1(\xi^*, (1,1)) \mid \cdots \mid H_1(\xi^*, (1, \hat{\ell}m))]$. For each $i \in [N]$, compute $\mathbf{a}_{\hat{f}_i^*} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i^*)$.
    - Compute $\mathbf{V}_N = \mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N) \in \mathbb{Z}^{m \times N}$ and for each $i \in [N]$, let $\mathbf{v}_i \in \mathbb{Z}^m$ be the i^{th} column of $\mathbf{V}_N$. Let $\mathbf{C}_0 = [H_1(\xi^*, (3,1)) \mid \cdots \mid H_1(\xi^*, (3,m))]$.
    - For each $i \in C$, let $\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi^*, i)) \in \mathbb{Z}^m$.
    - For each $i \in C$, let $j_i^* \in [M]$ be the special index associated with corrupted slot i (as defined in Hyb_6).
    - For each $j \in [M] \setminus \{j_i^*\}_{i \in C}$, compute $\mathbf{w}_j = H_1(\xi^*, (2, j))$.
    - For each $i \in C$, let $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i)$.
  - For each $i \in C$, the challenger samples $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ (after generating $\mathbf{A}$) and sets

$$\mathbf{w}_{j_i^*} = \mathbf{m}_i - \mathbf{P}_i^* \mathbf{u}_i + \mathbf{C}_0 \mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{w}_j.$$

    Note that this is well-defined since by construction of j_i^*, for all $i \in C$,

$$S_i \setminus \{j_i^*\} \subseteq [M] \setminus \{j_{i'}^*\}_{i' \in C}.$$

    The challenger now programs $H_1(\xi^*, (2, j_i^*)) \coloneqq \mathbf{w}_{j_i^*}$.

- Hyb_{13}: Same as Hyb_{12} except the challenger programs $\mathbf{d}_i$ for each $i \in [N] \setminus C$. Specifically, after $\mathcal{A}$ makes an aggregation-valid random oracle query and the challenger has generated $\mathbf{A}$, the challenger samples $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ for each $i \in [N] \setminus C$. After the challenger computes the $\mathbf{w}_j$ vectors for all $j \in [M]$ (via the procedure in Hyb_{12}), the challenger computes $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i)$ for each index $i \in [N] \setminus C$ and then sets

$$\mathbf{d}_i = \mathbf{m}_i + \mathbf{C}_0 \mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j \in S_i} \mathbf{w}_j.$$

- Hyb_{14}: Same as Hyb_{13} except the challenger programs $\mathbf{A}$, $\mathbf{C}_0$, and $\mathbf{w}_j$ for $j \in [M] \setminus \{j_i^*\}_{i \in C}$. In particular, after the setup phase (before any random oracle queries), the challenger samples the challenge ciphertext components $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \bar{D}^{4m^5}_{\mathbb{Z}, \sigma_{\mathsf{LWE}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}}$, and $\mathbf{R}_\Psi \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times \ell m}$. The challenger sets

$$\Psi = \begin{bmatrix} \mathbf{B} \\ \mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal \end{bmatrix} \mathbf{R}_\Psi + \mathbf{x}^\intercal \otimes \mathbf{G}_{n+1} \in \mathbb{Z}_q^{(n+1) \times \ell m},$$

  and sets $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ to be the binary representation of Ψ. Note that this is possible because the adversary commits to the challenge input $\mathbf{x}$ at the beginning of the input-selective security game. Moreover, whenever $\mathcal{A}$ makes an aggregation-valid random oracle query with prefix ξ^*, the challenger samples the candidate challenge ciphertext components $\mathbf{R}_{\mathbf{C}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$ and $\mathbf{R}_{\mathbf{A}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times \hat{\ell}m}$. The challenger sets $\mathbf{A} = \mathbf{B}\mathbf{R}_{\mathbf{A}} + \boldsymbol{\varphi}^\intercal \otimes \mathbf{G}_n$ and programs

$$[H_1(\xi^*, (1,1)) \mid \cdots \mid H_1(\xi^*, (1, \hat{\ell}m))] \coloneqq \mathbf{A}.$$

  For each $i \in [N]$, the challenger samples $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$. The challenger now sets $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_N]$, sets $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$, and sets $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_{\mathbf{C}} - \mathbf{C}_1$. The challenger additionally programs

$$[H_1(\xi^*, (3,1)) \mid \cdots \mid H_1(\xi^*, (3,m))] \coloneqq \mathbf{C}_0.$$

  Finally, for each index $j \in [M] \setminus \{j_i^*\}_{i \in C}$, the challenger samples $\mathbf{r}_{\mathbf{w}_j} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5}$, sets $\mathbf{w}_j = \mathbf{B}\mathbf{r}_{\mathbf{w}_j}$, and programs $H_1(\xi^*, (2, j)) \coloneqq \mathbf{w}_j$.

- Hyb_{15}: Same as Hyb_{14} except the challenger changes how it constructs the challenge ciphertext components $c'_j = \mathbf{s}^\mathsf{T}\mathbf{w}_j + e_{\mathbf{w}_j}$ for $j \in [M]$. Specifically, for each $i \in C$, the challenger computes the $(n+1)^{\text{th}}$ component $\underline{\psi}_{f'_i}$ of $\boldsymbol{\psi}_{f'_i} = \mathsf{EvalF}(\boldsymbol{\Psi}, f'_i)$, computes $\mathbf{h}_{\mathbf{A},\hat{f}^*_i,\boldsymbol{\varphi}} = \mathsf{EvalFX}(\mathbf{A}, \hat{f}^*_i, \boldsymbol{\varphi}) \in \mathbb{Z}^{\hat{\ell}m}$, lets $\mathbf{z}_i \in \mathbb{Z}^{4m^5}$ be the i^{th} column of $\mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$, and sets

$$c'_{j^*_i} = (\mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T})\left(\mathbf{z}_i - \mathbf{K}^*_i\mathbf{u}_i + \mathbf{R}_\mathbf{C}\mathbf{v}_i - \mathbf{R}_\mathbf{A}\mathbf{h}_{\mathbf{A},\hat{f}^*_i,\boldsymbol{\varphi}} - \sum_{j \in S_i \setminus \{j^*_i\}} \mathbf{r}_{\mathbf{w}_j}\right) + f^*_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor - \underline{\psi}_{f'_i} + e_{\mathbf{w}_{j^*_i}},$$

where $\mathbf{K}^*_i$ is the secret key sk^*_i the adversary provides for index $i \in C$.[6] For each $j \in [M] \setminus \{j^*_i\}_{i \in C}$, the challenger now sets

$$c'_j = (\mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T})\mathbf{r}_{\mathbf{w}_j} + e_{\mathbf{w}_j}.$$

In this hybrid, the challenge ciphertext can be simulated just given $(\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}, \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T})$ and we are ready to appeal to decomposed LWE. For completeness, we provide a more detailed description of the challenger's behavior in this hybrid here:

  - **Setup phase:** On input 1^λ, the adversary $\mathcal{A}$ outputs a function-family parameter 1^τ, a slot count N, a corrupted set $C \subseteq [N]$, and a challenge input $\mathbf{x} \in \{0,1\}^\ell$. The challenger samples the components $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$, $\mathbf{W}_{\mathsf{com}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3}$, $\mathbf{R}_{\mathsf{com}} \leftarrow \bar{D}^{m \times 2m^3}_{\mathbb{Z},\sigma_{\mathsf{pp}},\sqrt{\lambda}\sigma_{\mathsf{pp}}}$, and sets

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

    The challenger gives $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ to $\mathcal{A}$, and initializes $\mathsf{ctr} = 0$ and empty dictionaries $\mathsf{D}_{\mathsf{key}}, \mathsf{D}_{\mathsf{agg}}$. The challenger samples $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \bar{D}^{4m^5}_{\mathbb{Z},\sigma_{\mathsf{LWE}},\sqrt{\lambda}\sigma_{\mathsf{LWE}}}$, and $\mathbf{R}_{\boldsymbol{\Psi}} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times \ell m}$. Let $\mathbf{y}^\mathsf{T} := \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T} \in \mathbb{Z}_q^{4m^5}$. The challenger sets

$$\boldsymbol{\Psi} = \begin{bmatrix} \mathbf{B} \\ \mathbf{y}^\mathsf{T} \end{bmatrix} \mathbf{R}_{\boldsymbol{\Psi}} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1} \in \mathbb{Z}_q^{(n+1) \times \ell m},$$

    and sets $\boldsymbol{\varphi} \in \{0,1\}^{\hat{\ell}}$ to be the binary representation of $\boldsymbol{\Psi}$.
  - **Pre-challenge query phase:** The challenger answers queries as follows:
    * **Random-oracle queries:** The challenger maintains a table of its random-oracle responses. Whenever $\mathcal{A}$ queries H_1 (resp., H_2, H_3) on an input that is not already present in the table, the challenger samples a uniformly random value in $\mathbb{Z}_q^n$ (resp., $\{0,1\}$, $\{0,1\}^\rho$), records the input-output pair in the table, and replies with this value. On a previously-seen input it replies with the recorded value. The challenger consults the same table whenever it evaluates the random oracles itself, so that its own evaluations remain consistent with the values returned to $\mathcal{A}$.

      In addition, when $\mathcal{A}$ makes a random-oracle query, the challenger first checks the following conditions:

      (i) The query is of the form $H_1(\xi^*, \cdot)$, $H_2(\xi^*, \cdot)$, or $H_3(\xi^*, \cdot)$ for some string $\xi^* \in \{0,1\}^*$, and none of the preceding random-oracle queries have first input ξ^*.

      (ii) The string ξ^* is of the form $\xi^* = (\mathsf{pp}, Q, (\mathsf{pk}^*_1, \ldots, \mathsf{pk}^*_N), (f^*_1, \ldots, f^*_N))$, where $|C| \le Q \le N$, and moreover, $\mathsf{pk}^*_i \in \mathbb{Z}_q^{n \times m}$ and $f^*_i \in \mathcal{F}_\tau$ for all $i \in [N]$.

      (iii) For every $i \in [N] \setminus C$, there exists a counter $c \in [\mathsf{ctr}]$ such that pk^*_i is the public key stored in $\mathsf{D}_{\mathsf{key}}[c]$.

[6] Recall that in the semi-malicious model, the adversary submits a pair (sk^*_i, r_i) for each $i \in C$ in the challenge phase, and the challenger halts with output 0 unless $(\mathsf{pk}^*_i, \mathsf{sk}^*_i) = \mathsf{KeyGen}(\mathsf{pp}; r_i)$. Thus, whenever the challenger does not abort, we have that $\mathsf{sk}^*_i = \mathbf{K}^*_i \in \{0,1\}^{4m^5 \times m}$ and $\mathbf{P}^*_i = \mathbf{B}\mathbf{K}^*_i$ for all $i \in C$.

If any check fails, the challenger responds with the value determined above. If all checks pass, the challenger adds $\xi^* \mapsto \mathsf{st}$ to $\mathsf{D}_{\mathsf{agg}}$, where st is an empty state that will be populated with the components generated below. The challenger then computes $\mathsf{pp}_{\mathsf{CFF}} = H_2(\xi^*, 1) \| \cdots \| H_2(\xi^*, \rho_{\mathsf{CFF}})$ and for each $i \in [N]$, sets $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i) \subseteq [M]$. The challenger checks that for all $i \in C$, there exists an index $j_i^* \in S_i \setminus \bigcup_{i' \in C \setminus \{i\}} S_{i'}$. Define j_i^* to be the smallest such index if there are multiple. If no such index exists for some $i \in C$, the challenger halts with output 0. Otherwise, the challenger does the following:

- Sample $\mathbf{R}_{\mathbf{C}} \xleftarrow{R} \{0,1\}^{4m^5 \times m}$, $\mathbf{R}_{\mathbf{A}} \xleftarrow{R} \{0,1\}^{4m^5 \times \hat{\ell} m}$, and $\mathbf{r}_{\mathbf{w}_j} \xleftarrow{R} \{0,1\}^{4m^5}$ for each $j \in [M] \setminus \{j_i^*\}_{i \in C}$. Set $\mathbf{A} = \mathbf{B}\mathbf{R}_{\mathbf{A}} + \boldsymbol{\varphi}^{\mathsf{T}} \otimes \mathbf{G}_n$.
- Sample $\mathbf{m}_i \xleftarrow{R} \mathbb{Z}_q^n$ for all $i \in [N]$, and set $\mathbf{M} = [\mathbf{m}_1 \mid \cdots \mid \mathbf{m}_N]$, $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$, $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_{\mathbf{C}} - \mathbf{C}_1$.
- For each $i \in [N]$, let f_i' and $\hat{f}_i^*$ be the functions associated with f_i^* (Eq. (3.3) and Fig. 1), let $\mathbf{a}_{\hat{f}_i^*} = \mathsf{EvalF}(\mathbf{A}, \hat{f}_i^*)$, let $\mathbf{v}_i$ be the i^{th} column of $\mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N)$, and parse $\mathsf{pk}_i^* = \mathbf{P}_i^*$.
- For each index $j \in [M] \setminus \{j_i^*\}_{i \in C}$ set $\mathbf{w}_j = \mathbf{B}\mathbf{r}_{\mathbf{w}_j}$. The challenger programs

$$[H_1(\xi^*, (1,1)) \mid \cdots \mid H_1(\xi^*, (1, \hat{\ell} m))] := \mathbf{A}$$
$$[H_1(\xi^*, (3,1)) \mid \cdots \mid H_1(\xi^*, (3, m))] := \mathbf{C}_0,$$

  and $H_1(\xi^*, (2, j)) := \mathbf{w}_j$ for each $j \in [M] \setminus \{j_i^*\}_{i \in C}$.
- For each $i \in C$ set $\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi^*, i))$ and compute

$$\mathbf{w}_{j_i^*} = \mathbf{m}_i - \mathbf{P}_i^* \mathbf{u}_i + \mathbf{C}_0 \mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{w}_j.$$

  The challenger then programs $H_1(\xi^*, (2, j_i^*)) := \mathbf{w}_{j_i^*}$.
- For each $i \in [N] \setminus C$, let $c \in [\mathsf{ctr}]$ be the first counter where $\mathsf{D}_{\mathsf{key}}[c] = (\mathbf{P}_i^*, \mathbf{T}_i^*)$ for some trapdoor $\mathbf{T}_i^*$. Then, the challenger sets

$$\mathbf{d}_i = \mathbf{m}_i + \mathbf{C}_0 \mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j \in S_i} \mathbf{w}_j,$$

  computes $\mathbf{u}_i \leftarrow \mathsf{SamplePre}\big(\mathbf{P}_i^*, \mathbf{T}_i^*, \mathbf{d}_i, \sigma_{\mathsf{agg}}\big)$, sets $\mathbf{u}_i = \mathbf{T}_i^* \mathbf{G}_n^{-1}(\mathbf{d}_i)$ if $\|\mathbf{u}_i\| > \sqrt{\lambda}\sigma_{\mathsf{agg}}$, and programs $H_3(\xi^*, i) := \mathsf{DGS.Explain}(1^\lambda, \mathbf{u}_i, \sigma_{\mathsf{agg}})$.

* **Key-generation queries:** The challenger increments ctr = ctr+1, samples $(\mathbf{P}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$, sets $\mathsf{pk} = \mathbf{P}$, adds $\mathsf{ctr} \mapsto (\mathsf{pk}, \mathbf{T})$ to $\mathsf{D}_{\mathsf{key}}$, and replies with $(\mathsf{ctr}, \mathsf{pk})$.
* **Hint-generation queries:** When $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ for $c \le \mathsf{ctr}$, the challenger parses $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ where $\hat{\mathsf{ct}}_{\mathsf{main}} = (\hat{\boldsymbol{\Psi}}, \hat{\mathbf{c}}_1^{\mathsf{T}}, \hat{\mathbf{c}}_2^{\mathsf{T}}, \hat{\mathbf{c}}_3^{\mathsf{T}}, \hat{c}_1', \ldots, \hat{c}_M')$. Then, the challenger checks that

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

  If not, the challenger responds with $\perp$. If the proof verifies, the challenger computes

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

  If $\|\tilde{\mathbf{e}}\| > \sqrt{\lambda}\sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^{\mathsf{T}} \neq \tilde{\mathbf{s}}^{\mathsf{T}}\mathbf{B} + \tilde{\mathbf{e}}^{\mathsf{T}}$, then the challenger halts with output 0. Otherwise, the challenger looks up $(\mathsf{pk}, \mathbf{T}) = \mathsf{D}_{\mathsf{key}}[c]$, parses $\mathsf{pk} = \mathbf{P}$, samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z}, \sigma_{\mathsf{ht}}, \sqrt{\lambda}\sigma_{\mathsf{ht}}}$, and replies with $\tilde{\mathbf{s}}^{\mathsf{T}}\mathbf{P} + \hat{\mathbf{e}}_{\mathsf{ht}}^{\mathsf{T}}$.

– **Challenge phase:** After the adversary is finished making pre-challenge queries, it outputs a collusion bound 1^Q with $Q \le N$. The challenger checks that $|C| \le Q$ and halts with output 0 otherwise. Then, for each slot $i \in [N]$, the adversary specifies a tuple $(c_i^*, f_i^*, \mathsf{pk}_i^*)$ where $f_i^* \in \mathcal{F}_\tau$, $c_i^* = \perp$ for all $i \in C$, and $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$ for all $i \notin C$. For each $i \in C$, the adversary additionally provides (sk_i^*, r_i) such that $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) = \mathsf{KeyGen}(\mathsf{pp}; r_i)$. For each $i \in [N]$, the challenger then checks the following:

  * If $c_i^* \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up $(\mathsf{pk}', \mathbf{T}') = \mathsf{D}_{\mathsf{key}}[c_i^*]$ and halts with output 0 if $\mathsf{pk}_i^* \neq \mathsf{pk}'$.
  * If $c_i^* = \perp$, the challenger halts with output 0 if $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) \neq \mathsf{KeyGen}(\mathsf{pp}; r_i)$.

If the checks pass, the challenger sets

$$\xi^*_{\mathsf{chal}} = (\mathsf{pp}, Q, (\mathsf{pk}^*_1, \ldots, \mathsf{pk}^*_N), (f^*_1, \ldots, f^*_N)).$$

The challenger checks that ξ^*_{chal} is contained in $\mathsf{D}_{\mathsf{agg}}$, and halts with output 0 if not. If the check passes, then the challenger parses $\mathsf{sk}^*_i = \mathbf{K}^*_i \in \{0,1\}^{4m^5 \times m}$ for all $i \in C$, samples $e_{\mathbf{w}_1}, \ldots, e_{\mathbf{w}_M} \leftarrow \bar{D}_{\mathbb{Z}, \sigma_{\mathsf{smudge}}, \sqrt{\lambda}\sigma_{\mathsf{smudge}}}$, and sets

$$\begin{aligned}
\mathbf{c}_1^\mathsf{T} &= \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T} = \mathbf{y}^\mathsf{T} \\
\mathbf{c}_2^\mathsf{T} &= \mathbf{y}^\mathsf{T}\mathbf{R}_\mathsf{C} \\
\mathbf{c}_3^\mathsf{T} &= \mathbf{y}^\mathsf{T}\mathbf{R}_\mathsf{A} \\
c'_j &= \mathbf{y}^\mathsf{T}\mathbf{r}_{\mathbf{w}_j} + e_{\mathbf{w}_j} \qquad \forall j \in [M] \setminus \{j^*_i\}_{i \in C},
\end{aligned}$$

where the components multiplying $\mathbf{y}^\mathsf{T}$ are retrieved from $\mathsf{D}_{\mathsf{agg}}[\xi^*_{\mathsf{chal}}]$. For each $i \in C$, the challenger computes $\underline{\psi}_{f'_i} = \boldsymbol{\eta}^\mathsf{T}_{n+1}\mathsf{EvalF}(\boldsymbol{\Psi}, f'_i)$ and $\mathbf{h}_{\mathbf{A}, \hat{f}^*_i, \boldsymbol{\varphi}} = \mathsf{EvalFX}(\mathbf{A}, \hat{f}^*_i, \boldsymbol{\varphi})$. It then sets $\mathbf{z}_i \in \mathbb{Z}^{4m^5}$ to be the i^{th} column of $\mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ and computes the remaining ciphertext components as

$$c'_{j^*_i} = \mathbf{y}^\mathsf{T}\Big(\mathbf{z}_i - \mathbf{K}^*_i\mathbf{u}_i + \mathbf{R}_\mathsf{C}\mathbf{v}_i - \mathbf{R}_\mathsf{A}\mathbf{h}_{\mathbf{A}, \hat{f}^*_i, \boldsymbol{\varphi}} - \textstyle\sum_{j \in S_i \setminus \{j^*_i\}} \mathbf{r}_{\mathbf{w}_j}\Big) + f^*_i(\mathbf{x}) \cdot \lfloor q/2 \rfloor - \underline{\psi}_{f'_i} + e_{\mathbf{w}_{j^*_i}}.$$

Finally, the challenger sets $\mathsf{ct}^*_{\mathsf{main}} = (\boldsymbol{\Psi}, \mathbf{c}_1^\mathsf{T}, \mathbf{c}_2^\mathsf{T}, \mathbf{c}_3^\mathsf{T}, c'_1, \ldots, c'_M)$ and computes a simulated proof

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

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

- **Post-challenge query phase:** The challenger answers random-oracle and hint-generation queries as in the pre-challenge phase, subject to the restriction that $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$ for each hint-generation query.
- **Output phase:** At the end of the experiment, $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment (and 0 if $\mathcal{A}$ aborts without outputting a bit).

- Hyb_{16}: Same as Hyb_{15} except the challenger samples the vector $\mathbf{y} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{4m^5}$ uniformly at random, instead of setting $\mathbf{y}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T}$.

- Hyb_{17}: Same as Hyb_{16} except the challenger samples $\boldsymbol{\Psi} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{(n+1)\times \ell m}$ uniformly at random. This corresponds to the static experiment from Definition 3.1 corresponding to the case where $b = 1$ (i.e., the challenger answers the adversary's queries using the simulator algorithm).

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

Lemma 3.14. *Suppose Π_{NIZK} satisfies zero knowledge. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $\left|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks zero knowledge of Π_{NIZK} with the same advantage ε. Algorithm $\mathcal{B}$ works as follows:

- **Setup phase:** Algorithm $\mathcal{B}$ receives $\mathsf{crs}_{\mathsf{NIZK}}$ from the zero-knowledge challenger. It samples $\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}$ and constructs $\mathbf{B}$ and $\mathsf{pp}_{\mathsf{com}}$ exactly as in Hyb_0 and Hyb_1. Algorithm $\mathcal{B}$ gives $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ to $\mathcal{A}$. Finally, algorithm $\mathcal{B}$ initializes a counter $\mathsf{ctr} = 0$ and an empty dictionary $\mathsf{D}_{\mathsf{key}}$.

- **Pre-challenge query phase:** Algorithm $\mathcal{B}$ answers the random-oracle, key-generation, and hint-generation queries according to the specification in Hyb_0 and Hyb_1.

- **Challenge phase:** Algorithm $\mathcal{B}$ implements the challenge phase exactly as described in Hyb_0 and Hyb_1. After constructing $\mathsf{ct}^*_{\mathsf{main}}$, algorithm $\mathcal{B}$ submits the query $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e}))$ to the zero-knowledge challenger to obtain π^*_{ct}. Algorithm $\mathcal{B}$ gives $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{main}}, \pi^*_{\mathsf{ct}})$ to $\mathcal{A}$.
- **Post-challenge query phase:** Algorithm $\mathcal{B}$ responds to post-challenge random-oracle queries and hint-generation queries exactly as in the pre-challenge phase (by following the exact procedure in Hyb_0 and Hyb_1).
- **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which algorithm $\mathcal{B}$ also outputs.

First, we argue that algorithm $\mathcal{B}$ only queries the NIZK challenger on a valid statement-witness pair. By design, algorithm $\mathcal{B}$ makes a single query $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e}))$ to the NIZK challenger, where $\mathsf{ct}^*_{\mathsf{main}} = (\boldsymbol{\Psi}, \mathbf{c}_1^\intercal, \mathbf{c}_2^\intercal, \mathbf{c}_3^\intercal, c'_1, \ldots, c'_M)$. By definition, C_{ct} outputs 1 if $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$ and $\|\mathbf{e}\| \le \sqrt{\lambda}\sigma_{\mathsf{LWE}}$. By construction, algorithm $\mathcal{B}$ sets $\mathbf{c}_1^\intercal = \mathbf{s}^\intercal\mathbf{B} + \mathbf{e}^\intercal$, and moreover, algorithm $\mathcal{B}$ samples $\mathbf{e}$ such that $\|\mathbf{e}\| \le \sqrt{\lambda}\sigma_{\mathsf{LWE}}$. Thus,

$$C_{\mathsf{ct}}((\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), (\mathbf{s}, \mathbf{e})) = 1.$$

We now consider the two possible distributions of $\mathsf{crs}_{\mathsf{NIZK}}$ and the proof π^*_{ct}:

- If the challenger samples $\mathsf{crs}_{\mathsf{NIZK}} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$ and constructs π^*_{ct} as

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

  then algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_0. In this case, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1]$.
- If the challenger samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$ and constructs π^*_{ct} as

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

  then algorithm $\mathcal{B}$ perfectly simulates an execution of Hyb_1. Thus, algorithm $\mathcal{B}$ outputs 1 with probability $\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

We conclude that $\mathcal{B}$ breaks zero knowledge with the same advantage ε, as required. □

Lemma 3.15. *Suppose Π_{NIZK} is simulation extractable. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. By design, hybrids Hyb_1 and Hyb_2 are identical experiments, except in Hyb_2, the challenger additionally runs the extractor when answering hint-generation queries and halts with output 0 if extraction fails. Let Bad be the event that, in an execution of Hyb_1, algorithm $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ on a ciphertext $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ such that the following two conditions hold:

- $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1$; and
- $\|\tilde{\mathbf{e}}\| > \sqrt{\lambda}\sigma_{\mathsf{LWE}}$ or $\hat{\mathbf{c}}_1^\intercal \neq \tilde{\mathbf{s}}^\intercal\mathbf{B} + \tilde{\mathbf{e}}^\intercal$, where $(\tilde{\mathbf{s}}, \tilde{\mathbf{e}}) \leftarrow \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}})$.

The second condition is equivalent to

$$C_{\mathsf{ct}}((\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), (\tilde{\mathbf{s}}, \tilde{\mathbf{e}})) = 0.$$

Observe that if Bad does not happen, then the adversary's view in Hyb_1 and Hyb_2 is identically distributed. Thus, $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right| \le \Pr[\mathsf{Bad}]$. It thus suffices to bound $\Pr[\mathsf{Bad}]$. To do so, let Q' be a bound on the number of hint-generation queries algorithm $\mathcal{A}$ makes. Since $\mathcal{A}$ is efficient, $Q' = \mathsf{poly}(\lambda)$. Suppose also that $\Pr[\mathsf{Bad}] \ge \varepsilon$ for some non-negligible ε. We now use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ for the simulation-extractability game with the same advantage ε:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{B}$ receives $\mathsf{crs}_{\mathsf{NIZK}}$ from the challenger. Then, algorithm $\mathcal{B}$ samples $\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}$ and constructs $\mathbf{B}$ and $\mathsf{pp}_{\mathsf{com}}$ exactly as in Hyb_1. It gives $\mathsf{pp} = (1^\lambda, 1^\tau, N, \mathbf{B}, \mathsf{crs}_{\mathsf{NIZK}}, \mathsf{pp}_{\mathsf{com}})$ to $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an empty dictionary $\mathsf{D}_{\mathsf{key}}$. Algorithm $\mathcal{B}$ also samples an index $j \xleftarrow{\mathrm{R}} [Q']$.
- **Pre-challenge query phase:** Algorithm $\mathcal{B}$ responds to the queries as follows:
  - **Random-oracle queries and key-generation queries:** Algorithm $\mathcal{B}$ answers these queries using the same procedure as described in Hyb_1. In particular, none of these operations require knowledge of the simulation trapdoor $\mathsf{td}_{\mathsf{NIZK}}$.
  - **Hint-generation queries:** When $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ with $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$, algorithm $\mathcal{B}$ first checks if this is the j^{th} hint-generation query that algorithm $\mathcal{A}$ has made. If so, algorithm $\mathcal{B}$ halts with output $(C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}})$. Otherwise, algorithm $\mathcal{B}$ responds using the same procedure as described in Hyb_1 (which, again, does not require knowledge of the simulation trapdoor $\mathsf{td}_{\mathsf{NIZK}}$).
- **Challenge phase:** Algorithm $\mathcal{B}$ implements the challenge phase exactly as in Hyb_1. Specifically, after algorithm $\mathcal{B}$ constructs $\mathsf{ct}^*_{\mathsf{main}}$, it queries the simulation oracle on $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}))$ to obtain the proof π^*_{ct}. It then gives $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{main}}, \pi^*_{\mathsf{ct}})$ to $\mathcal{A}$.
- **Post-challenge query phase:** Algorithm $\mathcal{B}$ handles the post-challenge queries in the same manner as the pre-challenge queries.
- **Output phase:** If algorithm $\mathcal{B}$ has not yet halted (i.e., the adversary $\mathcal{A}$ makes fewer than j hint-generation queries), then it outputs $\perp$.

By construction, the simulation-extractability challenger samples $(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.TrapSetup}(1^\lambda)$ and $\pi^*_{\mathsf{ct}} \leftarrow \mathsf{NIZK.Sim}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}))$. This perfectly coincides with the distribution of $\mathsf{crs}_{\mathsf{NIZK}}$ and π^*_{ct} in Hyb_1. This means algorithm $\mathcal{B}$ perfectly simulates the view of $\mathcal{A}$ in an execution of Hyb_1. Thus, with probability at least ε, adversary $\mathcal{A}$ will make a hint-generation query $(\hat{\mathsf{ct}}, c)$ where $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$ such that

- $\mathsf{NIZK.Verify}(\mathsf{crs}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}}) = 1$; and
- $C_{\mathsf{ct}}((\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), (\tilde{\mathbf{s}}, \tilde{\mathbf{e}})) = 0$ where $(\tilde{\mathbf{s}}, \tilde{\mathbf{e}}) \leftarrow \mathsf{NIZK.Extract}(\mathsf{td}_{\mathsf{NIZK}}, C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}})$.

Let $j^* \in [Q']$ be the index of this hint-generation query. Since algorithm $\mathcal{B}$ samples $j \xleftarrow{\mathrm{R}} [Q']$, with probability $1/Q'$, it will be the case that $j = j^*$. In this case, algorithm $\mathcal{B}$ is successful provided that the tuple $(C_{\mathsf{ct}}, (\mathbf{B}, \hat{\mathsf{ct}}_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}), \hat{\pi}_{\mathsf{ct}})$ associated with the j^{th} hint-generation query $(\hat{\mathsf{ct}}, c)$ was not added to $\mathcal{Q}$ by an earlier query. We consider two cases:

- Suppose the j^{th} hint-generation query is a pre-challenge query. In this case, algorithm $\mathcal{B}$ has not made *any* simulation queries to the simulation-extractable challenger. Thus, algorithm $\mathcal{B}$ wins the simulation-extractability game.
- Suppose the j^{th} hint-generation query is a post-challenge query, which means $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$. In this case, algorithm $\mathcal{B}$ has made a single simulation query to the challenger on $(C_{\mathsf{ct}}, (\mathbf{B}, \mathsf{ct}^*_{\mathsf{main}}, \sqrt{\lambda}\sigma_{\mathsf{LWE}}))$ and received π^*_{ct} as the output. Since $\hat{\mathsf{ct}} \neq \mathsf{ct}^*$, $\hat{\mathsf{ct}} = (\hat{\mathsf{ct}}_{\mathsf{main}}, \hat{\pi}_{\mathsf{ct}})$, and $\mathsf{ct}^* = (\mathsf{ct}^*_{\mathsf{main}}, \pi^*_{\mathsf{ct}})$, this means that either $\hat{\mathsf{ct}}_{\mathsf{main}} \neq \mathsf{ct}^*_{\mathsf{main}}$ or $\hat{\pi}_{\mathsf{ct}} \neq \pi^*_{\mathsf{ct}}$. Once more, this is a valid output and algorithm $\mathcal{B}$ wins the simulation-extractability game.

We conclude that algorithm $\mathcal{B}$ wins the simulation-extractability game with advantage ε/Q', which is non-negligible since ε is assumed to be non-negligible and $Q' = \mathsf{poly}(\lambda)$. □

Lemma 3.16. *Suppose* $\sigma_{\mathsf{ht}} \geq 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} \cdot \lambda^{\omega(1)}$. *Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. Hybrids Hyb_2 and Hyb_3 are identical except in how the challenger computes its reply to a hint-generation query $(\hat{\mathsf{ct}}, c)$. In both experiments, the challenger verifies the proof $\hat{\pi}_{\mathsf{ct}}$, extracts $(\tilde{\mathbf{s}}, \tilde{\mathbf{e}})$ from $\hat{\pi}_{\mathsf{ct}}$, and halts with output 0 unless $\hat{\mathbf{c}}_1^{\mathsf{T}} = \tilde{\mathbf{s}}^{\mathsf{T}}\mathbf{B} + \tilde{\mathbf{e}}^{\mathsf{T}}$ and $\|\tilde{\mathbf{e}}\| \leq \sqrt{\lambda}\sigma_{\mathsf{LWE}}$. Let $\mathbf{P} = \mathbf{BK} \in \mathbb{Z}_q^{n \times m}$ denote the public key of user c, where $\mathbf{K} \in \{0,1\}^{4m^5 \times m}$ is the matrix sampled in the corresponding key-generation query. Consider how the challenger responds to the hint-generation query in the two experiments:

- In Hyb_2, the challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{ht}},\sqrt{\lambda}\sigma_{\mathsf{ht}}}$ and replies with

$$\hat{\mathbf{c}}_1^\intercal \mathbf{K} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal = \tilde{\mathbf{s}}^\intercal \mathbf{B}\mathbf{K} + \tilde{\mathbf{e}}^\intercal \mathbf{K} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal = \tilde{\mathbf{s}}^\intercal \mathbf{P} + \tilde{\mathbf{e}}^\intercal \mathbf{K} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal.$$

- In Hyb_3, the challenger samples $\hat{\mathbf{e}}_{\mathsf{ht}} \leftarrow \bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{ht}},\sqrt{\lambda}\sigma_{\mathsf{ht}}}$ and replies with $\tilde{\mathbf{s}}^\intercal \mathbf{P} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$.

Since $\mathbf{K} \in \{0,1\}^{4m^5 \times m}$, this means $\|\tilde{\mathbf{e}}^\intercal \mathbf{K}\| \le 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}}$. By Lemma 2.13 and a standard hybrid argument, the statistical distance between the distributions of $\tilde{\mathbf{e}}^\intercal \mathbf{K} + \hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$ and $\hat{\mathbf{e}}_{\mathsf{ht}}^\intercal$ is $\mathsf{negl}(\lambda)$ as long as $\sigma_{\mathsf{ht}} \ge 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} \cdot \lambda^{\omega(1)}$. Since the adversary makes at most $\mathsf{poly}(\lambda)$ hint-generation queries, the claim follows by a standard hybrid argument. □

Lemma 3.17. *Suppose $m \ge 2n \log q$, q is prime, and $\sigma_{\mathsf{pp}} \ge \log m$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Hybrids Hyb_3 and Hyb_4 are identical except in how the challenger samples the keys for the honest parties:

- In Hyb_3, the challenger samples $\mathbf{K} \xleftarrow{\mathrm{R}} \{0,1\}^{4m^5 \times m}$ and sets $\mathbf{P} = \mathbf{B}\mathbf{K}$. The matrix $\mathbf{K}$ is not used anywhere else in this experiment.
- In Hyb_4, the challenger samples $\mathbf{P} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$.

The challenger in both experiments sets $\mathbf{B} = \mathbf{W}_{\mathsf{com}}(\mathbf{I}_{2m^2} \otimes \mathbf{R}_{\mathsf{com}}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\intercal \otimes \mathbf{G}_n$ with $\mathbf{W}_{\mathsf{com}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times 2m^3}$ and $\mathbf{R}_{\mathsf{com}} \leftarrow \bar{D}^{m \times 2m^3}_{\mathbb{Z},\sigma_{\mathsf{pp}},\sqrt{\lambda}\sigma_{\mathsf{pp}}}$. Thus, the lemma holds by Lemma 2.18 and a hybrid argument (over the adversary's key-generation queries). □

Lemma 3.18. *Suppose $n \ge \lambda$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Hybrids Hyb_4 and Hyb_5 are identical except for the additional abort event introduced in Hyb_5. We argue that this event changes the output with only negligible probability. The only instance where the output can differ is if there exists some $i \in [N] \setminus C$ such that pk_i^* is not contained in $\mathsf{D}_{\mathsf{key}}$ when ξ^*_{chal} first appears as the first component of a random oracle query. The experiment will output 0 if there exists some $i \in [N] \setminus C$ such that pk_i^* is not contained in $\mathsf{D}_{\mathsf{key}}$ at the challenge phase, so this event only occurs if a public key in ξ^*_{chal} is sampled *after* ξ^*_{chal} first appears in a random oracle query. In Hyb_4, the honestly-generated public keys are uniform over $\mathbb{Z}_q^{n \times m}$, so the probability that the challenger samples a public key that matches pk_i^* is at most $q^{-nm} = \mathsf{negl}(\lambda)$. Since the adversary can make at most a polynomial number of key-generation queries, we can appeal to a union bound and conclude that the probability the challenger samples pk_i^* after ξ^*_{chal} first appears in a random oracle query is $\mathsf{negl}(\lambda)$. Thus, the differing abort event occurs with negligible probability, as desired. □

Lemma 3.19. *For all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_6(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Hybrids Hyb_5 and Hyb_6 are identical except for the additional abort condition in Hyb_6, which occurs when, for some corrupted slot $i \in C$, there does not exist an index $j_i^* \in S_i \setminus \bigcup_{i' \in C \setminus \{i\}} S_{i'}$, where $S_i = \mathsf{CFF.Expand}(\mathsf{pp}_{\mathsf{CFF}}, i)$. Take any aggregation-valid random oracle query with prefix ξ^* and let Q be the collusion bound associated with ξ^*. By Condition (ii), we have $|C| \le Q$, so the challenger runs CFF.Expand with a collusion bound that is at least the size of the corrupted set $C \subseteq [N]$. Moreover, by Condition (i), the values $H_2(\xi^*, 1), \ldots, H_2(\xi^*, \rho_{\mathsf{CFF}})$ are uniform and unqueried at the time of this query, so the public parameters $\mathsf{pp}_{\mathsf{CFF}} = H_2(\xi^*, 1) \| \cdots \| H_2(\xi^*, \rho_{\mathsf{CFF}})$ are distributed independently of C for each ξ^*. Thus, by Corollary 2.10,

$$\Pr\left[\exists i \in C : S_i \subseteq \bigcup_{i' \in C \setminus \{i\}} S_{i'}\right] \le \varepsilon = 2^{-\log^2(\lambda)} = \mathsf{negl}(\lambda).$$

Taking a union bound over the at most Q_{ro} aggregation-valid random oracle queries, with overwhelming probability, every $i \in C$ admits an index $j_i^* \in S_i$ with $j_i^* \notin S_{i'}$ for all $i' \in C \setminus \{i\}$ on every such query. In this case the behavior in Hyb_6 is identical to that in Hyb_5, proving the lemma. □

Lemma 3.20. *Suppose the discrete Gaussian sampler is ρ-explainable (Definition 2.20) and $\sigma_{\mathsf{agg}} = \mathsf{poly}(\lambda, d)$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

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

Proof. First, we note that algorithm $\mathcal{A}$ specifies the function-family parameter 1^τ in unary, so $\tau = \mathsf{poly}(\lambda)$. Correspondingly, this means $\sigma_{\mathsf{agg}} = \mathsf{poly}(\lambda, d(\tau))$ is bounded by $\mathsf{poly}(\lambda)$, as required by Theorem 2.21. Next, we note that hybrids Hyb_6 and Hyb_7 are identical except in how the challenger samples the vector $\mathbf{u}_i$ and the values $H_3(\xi^*, i)$ for $i \in [N] \setminus C$ when responding to an aggregation-valid random oracle query with prefix ξ^*.

- In Hyb_6, $H_3(\xi^*, i)$ is uniform and $\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi^*, i))$.
- In Hyb_7, the challenger samples $\mathbf{u}_i \leftarrow \bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{agg}},\sqrt{\lambda}\sigma_{\mathsf{agg}}}$, and sets $H_3(\xi^*, i) \leftarrow \mathsf{DGS.Explain}(1^\lambda, \mathbf{u}_i, \sigma_{\mathsf{agg}})$.

Fix an honest slot i. By the explainability of the sampler (Definition 2.20), the pair $(\mathbf{u}_i, H_3(\xi^*, i))$ in Hyb_6 is statistically close to the distribution $(\mathbf{u}_i, \mathsf{DGS.Explain}(1^\lambda, \mathbf{u}_i, \sigma_{\mathsf{agg}}))$ for $\mathbf{u}_i \leftarrow \bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{agg}},\sqrt{\lambda}\sigma_{\mathsf{agg}}}$. Since N and Q_{ro} are polynomially bounded, the claim follows by a hybrid argument over the at most N honest slots and Q_{ro} aggregation-valid random oracle queries. □

Lemma 3.21. *Suppose $\sigma_{\mathsf{agg}} \geq \log m$, $m \geq 2n \log q$, and q is prime. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_7(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. The only difference between the hybrids is the distribution of $\mathbf{u}_i$ for $i \in [N] \setminus C$ when an aggregation-valid query is made:

- In Hyb_7, the challenger samples $\mathbf{u}_i \leftarrow \bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{agg}},\sqrt{\lambda}\sigma_{\mathsf{agg}}}$.
- In Hyb_8, the challenger samples $\mathbf{d}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and $\mathbf{u}_i \leftarrow (\mathbf{P}_i^*)^{-1}_{\sigma_{\mathsf{agg}}}(\mathbf{d}_i)$.

First, $\bar{D}^m_{\mathbb{Z},\sigma_{\mathsf{agg}},\sqrt{\lambda}\sigma_{\mathsf{agg}}}$ and $D^m_{\mathbb{Z},\sigma_{\mathsf{agg}}}$ are within $\mathsf{negl}(\lambda)$ statistical distance by Lemma 2.11. The resulting distributions for $\mathbf{u}_i$ are then statistically indistinguishable by Lemma 2.12 since $\mathbf{P}_i^*$ is sampled uniformly. Thus, the lemma follows by a hybrid argument over at most N honest slots and Q_{ro} aggregation-valid random oracle queries. □

Lemma 3.22. *Suppose $m \geq 3n \log q$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_8(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Hybrids Hyb_8 and Hyb_9 are identical except in how the challenger samples the keys for the honest parties:

- In Hyb_8, the challenger samples $\mathbf{P} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^{n \times m}$.
- In Hyb_9, the challenger samples $(\mathbf{P}, \mathbf{T}) \leftarrow \mathsf{TrapGen}(1^n, q, m)$. The trapdoor $\mathbf{T}$ is not used anywhere in this experiment.

By the trapdoor-distribution property of Lemma 2.14, the above distributions are statistically indistinguishable. The claim now follows by a hybrid argument over the adversary's key-generation queries. □

Lemma 3.23. *Suppose $m \geq 2n \log q$ and $\sigma_{\mathsf{agg}} > \log m$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_9(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{10}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Since $\mathbf{P}_i^*\mathbf{T}_i^*\mathbf{G}_n^{-1}(\mathbf{d}_i) = \mathbf{d}_i$, the only difference between the experiments is the resetting of $\mathbf{u}_i$ for $i \in [N] \setminus C$, which occurs only when $\|\mathbf{u}_i\| > \sqrt{\lambda}\sigma_{\mathsf{agg}}$ for $\mathbf{u}_i \leftarrow (\mathbf{P}_i^*)^{-1}_{\sigma_{\mathsf{agg}}}(\mathbf{d}_i)$. Since the distribution of $\mathbf{P}_i^*$ is statistically close to uniform, Lemma 2.11 implies that this happens with probability at most $m \cdot 2^{-\lambda} + \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$. The claim now follows by a union bound over the at most N honest slots and Q_{ro} aggregation-valid random oracle queries. □

Lemma 3.24. *Suppose $m \geq 3n \log q$ and $\sigma_{\mathsf{agg}} \geq m \log n$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_{10}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{11}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. Hybrids Hyb_{10} and Hyb_{11} are identical except in how the challenger samples the preimage $\mathbf{u}_i$ of the target $\mathbf{d}_i$ for each honest slot $i \in [N] \setminus C$ when an aggregation-valid random oracle query is made:

- In Hyb_{10}, the challenger samples $\mathbf{u}_i \leftarrow (\mathbf{P}_i^*)^{-1}_{\sigma_{\mathsf{agg}}}(\mathbf{d}_i)$.
- In Hyb_{11}, the challenger samples $\mathbf{u}_i \leftarrow \mathsf{SamplePre}(\mathbf{P}_i^*, \mathbf{T}_i^*, \mathbf{d}_i, \sigma_{\mathsf{agg}})$, where $\mathbf{T}_i^*$ is the gadget trapdoor generated alongside $\mathbf{P}_i^*$ in Hyb_{10}.

Since $\mathbf{P}_i^*\mathbf{T}_i^* = \mathbf{G}_n$ and $\sigma_{\mathsf{agg}} \geq m \log n = m\|\mathbf{T}_i^*\| \log n$, the preimage-distribution property of Lemma 2.14 implies that the two distributions of $\mathbf{u}_i$ are statistically close. Taking a union bound over the at most N honest slots and Q_{ro} aggregation-valid queries, the claim follows. □

Lemma 3.25. *For all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_{11}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{12}(\mathcal{A}) = 1]$.*

Proof. Hybrids Hyb_{11} and Hyb_{12} are identical except in how the challenger generates the value $\mathbf{w}_{j_i^*} = H_1(\xi^*, (2, j_i^*))$ for each corrupted slot $i \in C$ when answering aggregation-valid random oracle queries with prefix ξ^*.

- In Hyb_{11}, the value $H_1(\xi^*, (2, j_i^*))$ is the uniform value recorded in the random-oracle table.
- In Hyb_{12}, the challenger samples $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, sets $\mathbf{w}_{j_i^*} = \mathbf{m}_i - \mathbf{P}_i^*\mathbf{u}_i + \mathbf{C}_0\mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{w}_j$, and programs $H_1(\xi^*, (2, j_i^*)) \coloneqq \mathbf{w}_{j_i^*}$.

By definition, an aggregation-valid random oracle query is the first query with prefix ξ^*, so the adversary has not queried $H_1(\xi^*, (2, j_i^*))$ before the challenger programs it. By the abort condition introduced in Hyb_6, the indices $\{j_i^*\}_{i \in C}$ are distinct, so the challenger programs one value for each j_i^*. Since $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ is uniform and independent of every other quantity, the programmed value $\mathbf{w}_{j_i^*}$ is uniform and independent in Hyb_{12}, exactly as in Hyb_{11}. The two experiments are therefore identically distributed. □

Lemma 3.26. *For all $\lambda \in \mathbb{N}$, $\Pr[\mathsf{Hyb}_{12}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{13}(\mathcal{A}) = 1]$.*

Proof. Hybrids Hyb_{12} and Hyb_{13} are identical except in how the challenger samples the preimage target $\mathbf{d}_i$ for the honest slots $i \in [N] \setminus C$ on aggregation-valid random oracle queries.

- In Hyb_{12}, the challenger samples $\mathbf{d}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ (as in Hyb_8).
- In Hyb_{13}, the challenger samples $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and sets $\mathbf{d}_i = \mathbf{m}_i + \mathbf{C}_0\mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \sum_{j \in S_i} \mathbf{w}_j$.

In both experiments $\mathbf{u}_i$ is sampled as a preimage of $\mathbf{d}_i$ in the same way, so it suffices to compare the distribution of $\mathbf{d}_i$. Since $\mathbf{m}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ is uniform and independent of the remaining quantities, the vector $\mathbf{d}_i$ is uniform and independent in Hyb_{13}, exactly as in Hyb_{12}. The two experiments are therefore identically distributed. □

Lemma 3.27. *Suppose $m \geq 2n \log q$, q is prime, and $\sigma_{\mathsf{pp}} \geq \log m$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_{13}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{14}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. First, hybrids Hyb_{13} and Hyb_{14} differ in *when* the challenger samples the components $\mathbf{s}, \mathbf{e}, \mathbf{R}_{\boldsymbol{\Psi}}$, and $\boldsymbol{\Psi}$: in Hyb_{13}, these are sampled in the challenge phase, whereas in Hyb_{14}, they are sampled in the setup phase. This reordering does not change the distribution of the experiment since these components are sampled independently of every other quantity in the experiment (and the challenge input $\mathbf{x}$ is fixed at the beginning of the input-selective security game). Otherwise, the two experiments are identical except that the matrix $\mathbf{A}$, the matrix $\mathbf{C}_0$, and the non-special vectors $\mathbf{w}_j$ ($j \in [M] \setminus \{j_i^*\}_{i \in C}$) are programmed when an aggregation-valid random oracle query is made:

- In Hyb_{13}, the values $\mathbf{A}$, $\mathbf{C}_0$, and $\mathbf{w}_j$ are the uniform values recorded in the random-oracle table.
- In Hyb_{14}, the challenger samples binary $\mathbf{R}_{\mathbf{A}}, \mathbf{R}_{\mathbf{C}}, \mathbf{r}_{\mathbf{w}_j}$ and sets $\mathbf{A} = \mathbf{B}\mathbf{R}_{\mathbf{A}} + \boldsymbol{\varphi}^{\mathsf{T}} \otimes \mathbf{G}_n$, $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_{\mathbf{C}} - \mathbf{C}_1$, and $\mathbf{w}_j = \mathbf{B}\mathbf{r}_{\mathbf{w}_j}$ (programming the corresponding random-oracle values).

Let $\mathbf{K} = [\mathbf{R}_{\mathbf{A}} \mid \mathbf{R}_{\mathbf{C}} \mid (\mathbf{r}_{\mathbf{w}_j})_{j\in[M]\setminus\{j_i^*\}_{i\in C}}] \in \{0,1\}^{4m^5\times k}$ be the concatenation of these binary matrices, where

$$k = \hat{\ell}m + m + \left|[M] \setminus \left\{j_i^*\right\}_{i\in C}\right| \le (n+1)\ell m^2 \lceil \log q\rceil + m + \tilde{O}(Q),$$

using $\hat{\ell} = (n+1)\ell m \lceil \log q\rceil$ and $M = \tilde{O}(Q)$. Since n, m, $\log q$, and ℓ are polynomially bounded in λ, τ, and $\log N$, and since the adversary outputs 1^τ, N, and 1^Q where $Q \le N$, we conclude that $k = \mathsf{poly}(\lambda)$. The shifts $\boldsymbol{\varphi}^\mathsf{T} \otimes \mathbf{G}_n$ and $\mathbf{C}_1$ are determined by quantities independent of $\mathbf{K}$. Moreover, in Hyb_{14}, we have $\mathbf{C} = \mathbf{C}_0 + \mathbf{C}_1 = \mathbf{B}\mathbf{R}_{\mathbf{C}}$ and $\mathbf{A} - \boldsymbol{\varphi}^\mathsf{T} \otimes \mathbf{G}_n = \mathbf{B}\mathbf{R}_{\mathbf{A}}$, so the challenge ciphertext components can be written as

$$\begin{aligned}
\mathbf{c}_2^\mathsf{T} &= \mathbf{s}^\mathsf{T}\mathbf{C} + \mathbf{e}^\mathsf{T}\mathbf{R}_{\mathbf{C}} = \mathbf{s}^\mathsf{T}(\mathbf{B}\mathbf{R}_{\mathbf{C}}) + \mathbf{e}^\mathsf{T}\mathbf{R}_{\mathbf{C}} \\
\mathbf{c}_3^\mathsf{T} &= \mathbf{s}^\mathsf{T}(\mathbf{A} - \boldsymbol{\varphi}^\mathsf{T} \otimes \mathbf{G}_n) + \mathbf{e}^\mathsf{T}\mathbf{R}_{\mathbf{A}} = \mathbf{s}^\mathsf{T}(\mathbf{B}\mathbf{R}_{\mathbf{A}}) + \mathbf{e}^\mathsf{T}\mathbf{R}_{\mathbf{A}} \\
c'_j &= \mathbf{s}^\mathsf{T}\mathbf{w}_j + e_{\mathbf{w}_j} = \mathbf{s}^\mathsf{T}(\mathbf{B}\mathbf{r}_{\mathbf{w}_j}) + e_{\mathbf{w}_j} \quad \text{for all } j \in [M] \setminus \left\{j_i^*\right\}_{i\in C}.
\end{aligned}$$

Each of these terms can be simulated from $\mathbf{B}\mathbf{K}$ and the leakage $\mathbf{e}^\mathsf{T}\mathbf{K}$, together with quantities that are independent of $\mathbf{K}$ (namely $\mathbf{s}$, $\mathbf{e}$, and the smudging noise $e_{\mathbf{w}_j}$). The same is true of the remaining components $c'_{j_i^*}$ for $i \in C$, of the vectors $\mathbf{d}_i$ the challenger computes for the honest slots, and of the values $\mathbf{A}$, $\mathbf{C}_0$, and $\mathbf{w}_j$ the challenger programs into the random oracle. Thus, the adversary's entire view is a function of $\mathbf{B}\mathbf{K}$, the leakage $\mathbf{e}^\mathsf{T}\mathbf{K}$, and quantities independent of $\mathbf{K}$. By Lemma 2.18, the distribution of $(\mathbf{B}, \mathbf{B}\mathbf{K}, \mathbf{e}^\mathsf{T}\mathbf{K})$ is statistically close to that of $(\mathbf{B}, \mathbf{U}, \mathbf{e}^\mathsf{T}\mathbf{K})$ for uniform $\mathbf{U}$. Replacing $\mathbf{B}\mathbf{K}$ with $\mathbf{U}$ transforms the distribution in Hyb_{14} into the distribution in Hyb_{13}. Thus the two are statistically close and the claim follows by a hybrid argument over at most Q_{ro} aggregation-valid queries. □

Lemma 3.28. *Suppose* $\sigma_{\mathsf{smudge}} \ge \lambda^{\omega(1)} \cdot \beta_{\mathsf{sm}}$, *where* $\beta_{\mathsf{sm}} = \tilde{O}(Q\ell) \cdot m^{O(d\cdot\mathsf{polylog}(m))} \cdot \mathsf{poly}(s) \cdot \sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}}\sigma_{\mathsf{agg}}$. *Then there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$\left|\Pr[\mathsf{Hyb}_{14}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{15}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Hybrids Hyb_{14} and Hyb_{15} are identical except for the components c'_j for $j \in [M]$.

- Consider $c'_{j_i^*}$ for $i \in C$. In Hyb_{14}, we have $c'_{j_i^*} = \mathbf{s}^\mathsf{T}\mathbf{w}_{j_i^*} + e_{\mathbf{w}_{j_i^*}}$, where

$$\mathbf{w}_{j_i^*} = \mathbf{m}_i - \mathbf{P}_i^*\mathbf{u}_i + \mathbf{C}_0\mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \sum_{j\in S_i\setminus\{j_i^*\}} \mathbf{w}_j.$$

  Let $\mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}} = \mathsf{EvalFX}(\mathbf{A}, \hat{f}_i^*, \boldsymbol{\varphi})$. By Theorem 2.15, we have

$$(\mathbf{A} - \boldsymbol{\varphi}^\mathsf{T} \otimes \mathbf{G}_n)\mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}} = \mathbf{a}_{\hat{f}_i^*} - \hat{f}_i^*(\boldsymbol{\varphi}).$$

  Similarly, by Theorem 2.19, the commitment $\mathbf{C}_1 = \mathsf{Com}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ and the opening $\mathbf{Z} = \mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ satisfy $\mathbf{C}_1\mathbf{V}_N = \mathbf{M} - \mathbf{B}\mathbf{Z}$. This means $\mathbf{C}_1\mathbf{v}_i = \mathbf{m}_i - \mathbf{B}\mathbf{z}_i$. Finally, in Hyb_{14}, we have $\mathbf{P}_i^* = \mathbf{B}\mathbf{K}_i^*$, $\mathbf{A} - \boldsymbol{\varphi}^\mathsf{T} \otimes \mathbf{G}_n = \mathbf{B}\mathbf{R}_{\mathbf{A}}$, $\mathbf{C}_0 = \mathbf{B}\mathbf{R}_{\mathbf{C}} - \mathbf{C}_1$, and $\mathbf{w}_j = \mathbf{B}\mathbf{r}_{\mathbf{w}_j}$ for all $j \in [M] \setminus \left\{j_i^*\right\}_{i\in C}$. Substituting these relations into the above expression for $c'_{j_i^*}$, we obtain

$$\begin{aligned}
c'_{j_i^*} &= \mathbf{s}^\mathsf{T}\Big(\mathbf{m}_i - \mathbf{P}_i^*\mathbf{u}_i + \mathbf{C}_0\mathbf{v}_i - \mathbf{a}_{\hat{f}_i^*} - \textstyle\sum_{j\in S_i\setminus\{j_i^*\}} \mathbf{w}_j\Big) + e_{\mathbf{w}_{j_i^*}} \\
&= \mathbf{s}^\mathsf{T}\Big(\mathbf{m}_i - \mathbf{B}\mathbf{K}_i^*\mathbf{u}_i + (\mathbf{B}\mathbf{R}_{\mathbf{C}} - \mathbf{C}_1)\mathbf{v}_i - \big(\mathbf{B}\mathbf{R}_{\mathbf{A}}\mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}} + \hat{f}_i^*(\boldsymbol{\varphi})\big) - \textstyle\sum_{j\in S_i\setminus\{j_i^*\}} \mathbf{B}\mathbf{r}_{\mathbf{w}_j}\Big) + e_{\mathbf{w}_{j_i^*}} \\
&= \mathbf{s}^\mathsf{T}\Big(\mathbf{m}_i - \mathbf{B}\mathbf{K}_i^*\mathbf{u}_i + \mathbf{B}\mathbf{R}_{\mathbf{C}}\mathbf{v}_i - \mathbf{m}_i + \mathbf{B}\mathbf{z}_i - \mathbf{B}\mathbf{R}_{\mathbf{A}}\mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}} - \hat{f}_i^*(\boldsymbol{\varphi}) - \textstyle\sum_{j\in S_i\setminus\{j_i^*\}} \mathbf{B}\mathbf{r}_{\mathbf{w}_j}\Big) + e_{\mathbf{w}_{j_i^*}} \\
&= \mathbf{s}^\mathsf{T}\mathbf{B}\Big(\mathbf{z}_i - \mathbf{K}_i^*\mathbf{u}_i + \mathbf{R}_{\mathbf{C}}\mathbf{v}_i - \mathbf{R}_{\mathbf{A}}\mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}} - \textstyle\sum_{j\in S_i\setminus\{j_i^*\}} \mathbf{r}_{\mathbf{w}_j}\Big) - \mathbf{s}^\mathsf{T}\hat{f}_i^*(\boldsymbol{\varphi}) + e_{\mathbf{w}_{j_i^*}} \\
&= \mathbf{s}^\mathsf{T}\mathbf{B}\mathbf{t}_i - \mathbf{s}^\mathsf{T}\hat{f}_i^*(\boldsymbol{\varphi}) + e_{\mathbf{w}_{j_i^*}},
\end{aligned} \tag{3.10}$$

where we write

$$\mathbf{t}_i \coloneqq \mathbf{z}_i - \mathbf{K}_i^* \mathbf{u}_i + \mathbf{R}_{\mathsf{C}} \mathbf{v}_i - \mathbf{R}_{\mathsf{A}} \mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}} - \textstyle\sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{r}_{\mathbf{w}_j}.$$

In Hyb_{15}, the challenger instead sets

$$c'_{j_i^*} = (\mathbf{s}^\intercal \mathbf{B} + \mathbf{e}^\intercal)\mathbf{t}_i + f_i^*(\mathbf{x}) \cdot \lfloor q/2 \rfloor - \underline{\psi}_{f_i'} + e_{\mathbf{w}_{j_i^*}}. \tag{3.11}$$

As in the proof of Theorem 3.12, Theorem 2.15 gives

$$\underline{\psi}_{f_i'} = \mathbf{s}^\intercal \hat{f}_i^*(\boldsymbol{\varphi}) + f_i^*(\mathbf{x}) \cdot \lfloor q/2 \rfloor + \mathbf{e}^\intercal \mathbf{R}_{\boldsymbol{\Psi}} \mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}},$$

where $\mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}} = \mathsf{EvalFX}(\boldsymbol{\Psi}, f_i', \mathbf{x})$. Substituting this expression for $\underline{\psi}_{f_i'}$ into Eq. (3.11), the value of $c'_{j_i^*}$ in Hyb_{15} can be written as

$$\begin{aligned} c'_{j_i^*} &= \mathbf{s}^\intercal \mathbf{B} \mathbf{t}_i + \mathbf{e}^\intercal \mathbf{t}_i + f_i^*(\mathbf{x}) \cdot \lfloor q/2 \rfloor - \left(\mathbf{s}^\intercal \hat{f}_i^*(\boldsymbol{\varphi}) + f_i^*(\mathbf{x}) \cdot \lfloor q/2 \rfloor + \mathbf{e}^\intercal \mathbf{R}_{\boldsymbol{\Psi}} \mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}}\right) + e_{\mathbf{w}_{j_i^*}} \\ &= \mathbf{s}^\intercal \mathbf{B} \mathbf{t}_i - \mathbf{s}^\intercal \hat{f}_i^*(\boldsymbol{\varphi}) + \mathbf{e}^\intercal \left(\mathbf{t}_i - \mathbf{R}_{\boldsymbol{\Psi}} \mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}}\right) + e_{\mathbf{w}_{j_i^*}}. \end{aligned} \tag{3.12}$$

Comparing Eqs. (3.10) and (3.12), we see that the two expressions are identical except that the noise term $e_{\mathbf{w}_{j_i^*}}$ in Hyb_{15} is shifted by the additive term

$$\Delta_i = \mathbf{e}^\intercal \left(\mathbf{t}_i - \mathbf{R}_{\boldsymbol{\Psi}} \mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}}\right) \in \mathbb{Z}.$$

We now bound $|\Delta_i|$ by expanding $\mathbf{t}_i$ and bounding each term individually. Recall that $\mathbf{e} \in \mathbb{Z}^{4m^5}$ satisfies $\|\mathbf{e}\| \le \sqrt{\lambda}\sigma_{\mathsf{LWE}}$, and that $\mathbf{K}_i^*, \mathbf{R}_{\mathsf{C}} \in \{0,1\}^{4m^5 \times m}$, $\mathbf{R}_{\mathsf{A}} \in \{0,1\}^{4m^5 \times \hat{\ell} m}$, $\mathbf{R}_{\boldsymbol{\Psi}} \in \{0,1\}^{4m^5 \times \ell m}$, and $\mathbf{r}_{\mathbf{w}_j} \in \{0,1\}^{4m^5}$. In particular, $\left\|\mathbf{e}^\intercal \mathbf{K}_i^*\right\|, \|\mathbf{e}^\intercal \mathbf{R}_{\mathsf{C}}\|, \|\mathbf{e}^\intercal \mathbf{R}_{\mathsf{A}}\|, \|\mathbf{e}^\intercal \mathbf{R}_{\boldsymbol{\Psi}}\| \le 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}}$. We now consider each term in $\mathbf{t}_i$:

- Since $\mathbf{z}_i$ is the i^{th} column of $\mathsf{Open}(\mathsf{pp}_{\mathsf{com}}, \mathbf{M})$ and $\|\mathbf{R}_{\mathsf{com}}\| \le \sqrt{\lambda}\sigma_{\mathsf{pp}}$, we appeal to Theorem 2.19 and conclude that $\|\mathbf{z}_i\| \le O(\sqrt{\lambda}\sigma_{\mathsf{pp}} m^7 \log q \log N)$, so

$$\left|\mathbf{e}^\intercal \mathbf{z}_i\right| \le 4m^5 \cdot \sqrt{\lambda}\sigma_{\mathsf{LWE}} \cdot O(\sqrt{\lambda}\sigma_{\mathsf{pp}} m^7 \log q \log N) = \lambda \cdot \sigma_{\mathsf{LWE}}\sigma_{\mathsf{pp}} \cdot \log q \log N \cdot m^{O(1)}.$$

- Since $\mathbf{u}_i = \mathsf{DGS.Sample}(1^\lambda, 1^m, \sigma_{\mathsf{agg}}; H_3(\xi_{\mathsf{chal}}^*, i))$, we have $\|\mathbf{u}_i\| \le \sqrt{\lambda}\sigma_{\mathsf{agg}}$ by Definition 2.20, so

$$\left|\mathbf{e}^\intercal \mathbf{K}_i^* \mathbf{u}_i\right| \le m \cdot 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} \cdot \sqrt{\lambda}\sigma_{\mathsf{agg}} = \lambda \cdot \sigma_{\mathsf{LWE}}\sigma_{\mathsf{agg}} \cdot m^{O(1)}.$$

- Since $\mathbf{v}_i$ is the i^{th} column of $\mathsf{Ver}(\mathsf{pp}_{\mathsf{com}}, N)$, Theorem 2.19 gives $\|\mathbf{v}_i\| \le O(\sqrt{\lambda}\sigma_{\mathsf{pp}} m^4 \log q)$, so

$$\left|\mathbf{e}^\intercal \mathbf{R}_{\mathsf{C}} \mathbf{v}_i\right| \le m \cdot 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} \cdot O(\sqrt{\lambda}\sigma_{\mathsf{pp}} m^4 \log q) = \lambda \cdot \sigma_{\mathsf{LWE}}\sigma_{\mathsf{pp}} \cdot \log q \cdot m^{O(1)}.$$

- As in the proof of Theorem 3.12, Theorem 2.15 gives $\|\mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}}\|, \|\mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}}\| \le m^{O(d \cdot \mathsf{polylog}(m))} \cdot \mathsf{poly}(s)$. Since $\hat{\ell} = (n+1)\ell m \lceil \log q \rceil$ and $n, \log q \le m$, this means

$$\left|\mathbf{e}^\intercal \mathbf{R}_{\mathsf{A}} \mathbf{h}_{\mathbf{A},\hat{f}_i^*,\boldsymbol{\varphi}}\right|, \left|\mathbf{e}^\intercal \mathbf{R}_{\boldsymbol{\Psi}} \mathbf{h}_{\boldsymbol{\Psi},f_i',\mathbf{x}}\right| \le \sqrt{\lambda} \cdot \sigma_{\mathsf{LWE}}\ell \cdot m^{O(d \cdot \mathsf{polylog}(m))} \cdot \mathsf{poly}(s).$$

- Finally, since $S_i \subseteq [M]$ where $M = \tilde{O}(Q)$ by Corollary 2.10,

$$\left|\textstyle\sum_{j \in S_i \setminus \{j_i^*\}} \mathbf{e}^\intercal \mathbf{r}_{\mathbf{w}_j}\right| \le M \cdot 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} = \tilde{O}(Q) \cdot \sqrt{\lambda} \cdot \sigma_{\mathsf{LWE}} \cdot m^{O(1)}.$$

Since each of the width parameters $\sigma_{\mathsf{LWE}}, \sigma_{\mathsf{pp}}, \sigma_{\mathsf{agg}}$ is at least 1, and using the fact that $\lambda \le m$ and $\log q \le m$, we can sum up the above bounds to conclude that

$$|\Delta_i| \le \tilde{O}(Q\ell) \cdot m^{O(d \cdot \mathsf{polylog}(m))} \cdot \mathsf{poly}(s) \cdot \sigma_{\mathsf{pp}}\sigma_{\mathsf{LWE}}\sigma_{\mathsf{agg}} = \beta_{\mathsf{sm}}.$$

Since $e_{\mathbf{w}_{j_i^*}} \leftarrow \bar{D}_{\mathbb{Z},\sigma_{\mathsf{smudge}},\sqrt{\lambda}\sigma_{\mathsf{smudge}}}$ with $\sigma_{\mathsf{smudge}} \ge \lambda^{\omega(1)} \cdot \beta_{\mathsf{sm}}$, we appeal to Lemma 2.13 to conclude that the distributions $e_{\mathbf{w}_{j_i^*}}$ and $e_{\mathbf{w}_{j_i^*}} + \Delta_i$ are statistically close. Thus, the distributions of $c'_{j_i^*}$ are statistically indistinguishable between the two experiments.

- Consider $j \in [M] \setminus \{j_i^*\}_{i \in C}$. In this case, $\mathbf{w}_j = \mathbf{B}\mathbf{r}_{\mathbf{w}_j}$. Thus, in Hyb_{14},
$$c'_j = \mathbf{s}^\mathsf{T}\mathbf{w}_j + e_{\mathbf{w}_j} = \mathbf{s}^\mathsf{T}\mathbf{B}\mathbf{r}_{\mathbf{w}_j} + e_{\mathbf{w}_j}.$$
In Hyb_{15}, we have
$$c'_j = (\mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T})\mathbf{r}_{\mathbf{w}_j} + e_{\mathbf{w}_j}.$$
Again, these are identical expressions except that the noise term $e_{\mathbf{w}_j}$ in Hyb_{14} is shifted by the additive term $\mathbf{e}^\mathsf{T}\mathbf{r}_{\mathbf{w}_j}$ in Hyb_{15}. Since $\mathbf{r}_{\mathbf{w}_j} \in \{0,1\}^{4m^5}$ and $\|\mathbf{e}\| \le \sqrt{\lambda}\sigma_{\mathsf{LWE}}$, we have $|\mathbf{e}^\mathsf{T}\mathbf{r}_{\mathbf{w}_j}| \le 4m^5\sqrt{\lambda}\sigma_{\mathsf{LWE}} \le \beta_{\mathsf{sm}}$. Appealing to Lemma 2.13 again, the distribution of c'_j is statistically indistinguishable in the two experiments.

Since $M = \mathsf{poly}(\lambda)$, the claim now follows by a hybrid argument over the M components $c'_1, \ldots, c'_M$. □

Lemma 3.29. *Suppose the $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE assumption holds with lattice parameters $(n, m, q, \sigma_{\mathsf{LWE}})$. Then, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*
$$\left|\Pr[\mathsf{Hyb}_{15}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{16}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. The two hybrids differ in the distribution of $\mathbf{y}^\mathsf{T}$. Suppose $\left|\Pr[\mathsf{Hyb}_{15}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{16}(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE:

- At the start of the game, $\mathcal{B}$ gets a challenge $(\mathbf{W}, \mathbf{R}, \mathbf{v}^\mathsf{T})$ from the decomposed LWE challenger, where $\mathbf{W} \in \mathbb{Z}_q^{n \times 2m^3}$ and $\mathbf{R} \in \mathbb{Z}^{m \times 2m^3}$, and sets $\mathbf{W}_{\mathsf{com}} = \mathbf{W}$, $\mathbf{R}_{\mathsf{com}} = \mathbf{R}$, and $\mathbf{y}^\mathsf{T} = \mathbf{v}^\mathsf{T} \in \mathbb{Z}_q^{4m^5}$. Algorithm $\mathcal{B}$ then runs the setup phase as in Hyb_{15}, except it constructs
$$\mathbf{B} = \mathbf{W}_{\mathsf{com}}(\mathbf{I}_{2m^2} \otimes \mathbf{R}_{\mathsf{com}}) + \mathsf{vec}(\mathbf{I}_{2m^2})^\mathsf{T} \otimes \mathbf{G}_n \in \mathbb{Z}_q^{n \times 4m^5} \quad \text{and} \quad \mathsf{pp}_{\mathsf{com}} = (\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}, \mathbf{B})$$
from the challenge (rather than sampling $\mathbf{W}_{\mathsf{com}}$ and $\mathbf{R}_{\mathsf{com}}$ itself), and it does *not* sample $\mathbf{s}$ and $\mathbf{e}$. It uses $\mathbf{y}^\mathsf{T}$ in place of $\mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T}$ when constructing $\mathbf{\Psi}$.
- Algorithm $\mathcal{B}$ runs the rest of the game exactly as described in Hyb_{15} (which only requires knowledge of $\mathbf{y}^\mathsf{T}$).

If $\mathbf{y}^\mathsf{T} = \mathbf{s}^\mathsf{T}\mathbf{B} + \mathbf{e}^\mathsf{T}$, algorithm $\mathcal{B}$ perfectly simulates Hyb_{15}, and if $\mathbf{y}^\mathsf{T}$ is uniform, algorithm $\mathcal{B}$ perfectly simulates Hyb_{16}. Thus, algorithm $\mathcal{B}$ breaks $(2m^2, \sigma_{\mathsf{pp}})$-decomposed LWE with advantage ε. □

Lemma 3.30. *Suppose $m \ge 2n \log q$, q is prime, and $\sigma_{\mathsf{pp}} \ge \log m$. Then there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{Hyb}_{16}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{17}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. The difference between the hybrids is the distribution of $\mathbf{\Psi}$:

- In Hyb_{16}, the challenger samples $\mathbf{R}_{\mathbf{\Psi}} \xleftarrow{\mathsf{R}} \{0,1\}^{4m^5 \times \ell m}$ and sets $\mathbf{\Psi} = \begin{bmatrix} \mathbf{B} \\ \mathbf{y}^\mathsf{T} \end{bmatrix} \mathbf{R}_{\mathbf{\Psi}} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1}$.
- In Hyb_{17}, the challenger samples $\mathbf{\Psi} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^{(n+1) \times \ell m}$.

We show that the distributions are indistinguishable in two steps:

- First, we apply Lemma 2.18 with the fixed vector $\mathbf{y}$ and $\mathbf{K} = \mathbf{R}_{\mathbf{\Psi}} \in \{0,1\}^{4m^5 \times \ell m}$. Then, $(\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}, \mathbf{B}\mathbf{R}_{\mathbf{\Psi}}, \mathbf{y}^\mathsf{T}\mathbf{R}_{\mathbf{\Psi}})$ is statistically close to $(\mathbf{W}_{\mathsf{com}}, \mathbf{R}_{\mathsf{com}}, \mathbf{U}, \mathbf{y}^\mathsf{T}\mathbf{R}_{\mathbf{\Psi}})$ where $\mathbf{U} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^{n \times \ell m}$. This means the following distributions are statistically indistinguishable:
$$\begin{bmatrix} \mathbf{B} \\ \mathbf{y}^\mathsf{T} \end{bmatrix} \mathbf{R}_{\mathbf{\Psi}} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1} \quad \text{and} \quad \begin{bmatrix} \mathbf{U} \\ \mathbf{y}^\mathsf{T}\mathbf{R}_{\mathbf{\Psi}} \end{bmatrix} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1}.$$
- Next, since $\mathbf{y} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^{4m^5}$ is uniform and $4m^5 \ge 2 \log q$, we can apply Lemma 2.17 to conclude that the distribution of $(\mathbf{y}, \mathbf{y}^\mathsf{T}\mathbf{R}_{\mathbf{\Psi}})$ is statistically close to $(\mathbf{y}, \mathbf{u}^\mathsf{T})$ where $\mathbf{u} \xleftarrow{\mathsf{R}} \mathbb{Z}_q^{\ell m}$. This means the following distributions are statistically indistinguishable:
$$\begin{bmatrix} \mathbf{B} \\ \mathbf{y}^\mathsf{T} \end{bmatrix} \mathbf{R}_{\mathbf{\Psi}} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1} \quad \text{and} \quad \begin{bmatrix} \mathbf{U} \\ \mathbf{u}^\mathsf{T} \end{bmatrix} + \mathbf{x}^\mathsf{T} \otimes \mathbf{G}_{n+1}.$$
Since $\mathbf{U}$ and $\mathbf{u}$ are uniform and independent of all other quantities, the latter distribution is precisely Hyb_{17}.

The claim now holds by a hybrid argument. □

Completing the proof. By Lemmas 3.14 to 3.30, we conclude that

$$|\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{17}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda),$$

and security follows. □

Remark 3.31 (Reducing Size of Master Public Key)**.** The master public key $\mathsf{mpk} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{w}_1, \ldots, \mathbf{w}_M)$ in Construction 3.11 contains $M = \tilde{O}(Q)$ vectors $\mathbf{w}_1, \ldots, \mathbf{w}_M \in \mathbb{Z}_q^n$. Thus, the size of the master public key scales with the collusion bound Q. We can remove this dependence using an extra layer of indirection. Let $H_0 \colon \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function (which we also model as a random oracle). Then, we modify the Aggregate algorithm to first compute a short digest $\xi' = H_0(\xi)$ of the aggregation input ξ from Eq. (3.4), and then derive every random-oracle-generated component using ξ' as the prefix rather than ξ. In particular, the Aggregate algorithm now sets $\mathbf{w}_j = H_1(\xi', (2, j))$ for each $j \in [M]$. Likewise, it derives $\mathbf{A}$, $\mathsf{pp}_{\mathsf{CFF}}$, $\mathbf{C}_0$, and $\mathbf{u}_i$ using ξ'. Since the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$ and the matrix $\mathbf{A}$ are now recomputable from ξ' alone, the aggregation algorithm instead outputs $\mathsf{mpk} = (\mathsf{crs}_{\mathsf{NIZK}}, \mathbf{B}, \mathbf{C}, \xi')$. This reduces the size of the master public key to $\mathsf{poly}(\lambda, d, \log \ell, \log s, \log N)$, which is independent of the collusion bound Q and depends only polylogarithmically on the input length ℓ. Similarly, we can reduce the size of the helper decryption keys by setting $\mathsf{hsk}_i = (\xi', \mathbf{v}_i, \mathbf{z}_i, f_i)$. The encryption and recovery algorithms can compute the matrix $\mathbf{A}$, the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_M$, and the cover-free set parameters $\mathsf{pp}_{\mathsf{CFF}}$ from ξ' and then proceed as before. Finally, the ciphertext is unchanged. It still contains the M components $c'_1, \ldots, c'_M$, so the size of the ciphertext still scales with the collusion bound Q (and this is unavoidable due to Remark 3.8). This also means that the running time of the encryption algorithm scales with the collusion bound (and formally, we would augment the encryption algorithm to take the collusion bound 1^Q as an explicit input).

The security analysis carries over with one change. In the proof of Theorem 3.13, the challenger programs the values of H_1, H_2, H_3 at ξ^* on each aggregation-valid random oracle query with prefix ξ^*. In the modified scheme, whenever the adversary queries H_0 on such an input ξ^*, the challenger samples a fresh $\xi' \xleftarrow{R} \{0,1\}^\lambda$ and responds with ξ'. Then the challenger programs the values of $H_1(\xi', \cdot)$, $H_2(\xi', \cdot)$, and $H_3(\xi', \cdot)$, exactly as before. Since each ξ' is uniform and independent of the adversary's view at the time it is sampled, the probability that the adversary queries H_1, H_2, H_3 on an input whose prefix is ξ' is at most $Q_{\mathsf{ro}}^2 / 2^\lambda = \mathsf{negl}(\lambda)$, where $Q_{\mathsf{ro}} = \mathsf{poly}(\lambda)$ is again a bound on the number of random oracle queries the adversary makes. The rest of the analysis now follows similarly.

Remark 3.32 (Functions with Multi-Bit Output)**.** We can extend Construction 3.11 to support functions $f \colon \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{out}}}$ with ℓ_{out} output bits by instantiating the scheme with $N\ell_{\mathsf{out}}$ slots and collusion bound $Q\ell_{\mathsf{out}}$. Namely, each user now registers ℓ_{out} slots, where the k^{th} slot of user i is associated with the function $f_i^{(k)}$ that computes the k^{th} output bit of f_i. The user recovers $f_i(\mathbf{x})$ by decrypting each of its ℓ_{out} slots. Since an adversary that corrupts a user obtains all ℓ_{out} of its slots, the collusion bound scales accordingly, and security follows immediately from Theorem 3.13. We note that a user can reuse a single key-pair $(\mathsf{pk}, \mathsf{sk})$ for all of its slots. Since GetHint depends only on the user's secret key and the ciphertext (and not on the slot index or the function), a single decryption hint suffices for all ℓ_{out} slots.

Parameter instantiation. Let λ be a security parameter, $\varepsilon \in (0,1)$ be a constant, and $\mathcal{F} = \{\mathcal{F}_\tau\}_{\tau \in \mathbb{N}}$ be a family of functions $f \colon \{0,1\}^\ell \to \{0,1\}^{\ell_{\mathsf{out}}}$ on inputs of length $\ell = \ell(\tau)$ and outputs of length $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\tau)$, and which can be computed by Boolean circuits of depth at most $d = d(\tau)$ and size at most $s = s(\tau)$. Let N be the number of slots and $Q \le N$ be the collusion bound. We instantiate the parameters in Construction 3.11 (with the extension from Remark 3.32) as follows so as to satisfy Theorems 3.12 and 3.13:

- We set the lattice dimension $n = \lambda \cdot d^{1/\varepsilon} \cdot \mathsf{polylog}(\lambda, d, \ell, s, N)$, where the polylog factor is chosen so that $n^\varepsilon \ge \log q$.
- We take $m = \Theta(n \log q)$ where $m \ge 3n \log q$. In particular, $n \ge \lambda$ and $m > 2(n+1)\log q$.
- We set the Gaussian width parameters to be

$$\sigma_{\mathsf{pp}} = \lceil \log m \rceil + 1, \quad \sigma_{\mathsf{agg}} = 2m \log n, \quad \sigma_{\mathsf{LWE}} = \mathsf{poly}(n), \quad \sigma_{\mathsf{ht}} = 2^{\log^2 \lambda} \cdot m^5 \sqrt{\lambda} \sigma_{\mathsf{LWE}},$$

  and $\sigma_{\mathsf{smudge}} = 2^{\log^2 \lambda} \cdot \tilde{O}(N\ell) \cdot m^{O(d \cdot \mathsf{polylog}(m))} \cdot \mathsf{poly}(s) \cdot \sigma_{\mathsf{pp}} \sigma_{\mathsf{LWE}} \sigma_{\mathsf{agg}}$. Taking $\lambda^{\omega(1)} = 2^{\log^2 \lambda}$, these satisfy the requirements $\sigma_{\mathsf{pp}} > \log m$, $\sigma_{\mathsf{agg}} > m \log n$, $\sigma_{\mathsf{agg}} = \mathsf{poly}(\lambda, d)$, $\sigma_{\mathsf{ht}} \ge 4m^5 \sqrt{\lambda} \sigma_{\mathsf{LWE}} \cdot \lambda^{\omega(1)}$, and $\sigma_{\mathsf{smudge}} \ge \lambda^{\omega(1)} \cdot \beta_{\mathsf{sm}}$

from Theorem 3.13. Note here that we use the fact that the extension from Remark 3.32 instantiates Construction 3.11 with $N\ell_{\mathsf{out}}$ slots and collusion bound $Q\ell_{\mathsf{out}}$, and since $s \geq \ell_{\mathsf{out}}$, the extra factor of ℓ_{out} can be absorbed into $\mathsf{poly}(s)$.

- We choose the modulus q to be a prime satisfying

$$\begin{aligned} q &> 4 \cdot \tilde{O}(N\ell_{\mathsf{out}}) \cdot m^{O(d \cdot \mathsf{polylog}(m))} \cdot \mathsf{poly}(s) \cdot \ell \cdot \sigma_{\mathsf{LWE}} \sigma_{\mathsf{pp}} \sigma_{\mathsf{agg}} \sigma_{\mathsf{smudge}} \sigma_{\mathsf{ht}} \\ &= 2^{\mathsf{polylog}(\lambda)} \cdot N\ell \cdot m^{O(d \cdot \mathsf{polylog}(m))} \cdot \mathsf{poly}(s), \end{aligned}$$

  so as to satisfy Theorem 3.12.

Taken together, this means $\log m = \tilde{O}(1)$, $\log q = \tilde{O}(d)$, and $m = \lambda \cdot \tilde{O}\left(d^{1+1/\varepsilon}\right)$. By construction, $\log q \leq n^{\varepsilon}$, so $q = 2^{\tilde{O}(d)} \leq 2^{n^{\varepsilon}}$, and correspondingly, we rely on the decomposed LWE assumption with a sub-exponential modulus-to-noise ratio. With this setting of parameters, the ciphertext $\mathsf{ct} = (\mathsf{ct}_{\mathsf{main}}, \pi_{\mathsf{ct}})$ has size

$$\underbrace{M \lceil \log q \rceil}_{c'_1, \ldots, c'_M} + \underbrace{\left((n+1)\ell m + 4m^5 + m + \hat{\ell} m\right) \lceil \log q \rceil}_{\boldsymbol{\Psi}, \mathbf{c}_1^{\mathsf{T}}, \mathbf{c}_2^{\mathsf{T}}, \mathbf{c}_3^{\mathsf{T}}} + \underbrace{\mathsf{poly}(\lambda, d)}_{\pi_{\mathsf{ct}}} = Q\ell_{\mathsf{out}} \cdot \tilde{O}(d) + \ell \cdot \mathsf{poly}(\lambda, d, \log N),$$

where we use the fact that $M = \tilde{O}(Q\ell_{\mathsf{out}})$, $\hat{\ell} = (n+1)\ell m \lceil \log q \rceil$, and that the proof π_{ct} has size $|(\mathbf{s}, \mathbf{e})| + \mathsf{poly}(\lambda, d) = \mathsf{poly}(\lambda, d)$ by Fact 2.2. Based on these parameter choices, we can appeal to Theorems 3.12 and 3.13 and Fact 2.2, and Remarks 3.7, 3.31 and 3.32 to obtain the following corollary:

Corollary 3.33 (Slotted Succinct Bounded-Collusion Registered FE)**.** *Let λ be a security parameter, N be the number of slots, and $Q \leq N$ be a collusion bound. Let $\mathcal{F}$ be a function family with input length $\ell = \ell(\lambda)$ and output length $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$ that can be computed by a Boolean circuit of depth at most $d = d(\lambda)$ and size at most $s = s(\lambda)$. Then, under the decomposed LWE assumption with a sub-exponential modulus-to-noise ratio, there exists an input-selective static CCA-secure succinct bounded-collusion registered FE scheme for $\mathcal{F}$ in the random oracle model. The scheme supports up to N users and satisfies the following properties:*

- **Public parameters, user keys, and decryption hints:** *The public parameters* pp*, each individual user's public key* pk *and secret key* sk*, and the decryption hints* ht *all have size* $\mathsf{poly}(\lambda, d, \log \ell, \log s, \log N)$.
- **Master public key:** *The size of the master public key* mpk *is* $\mathsf{poly}(\lambda, d, \log \ell, \log s, \log N)$*. Note that this is independent of the collusion bound Q (see Remark 3.31).*
- **Helper decryption keys:** *The size of each user's helper decryption key is* $\ell_{\mathsf{out}} \cdot \mathsf{poly}(\lambda, d, \log \ell, \log s, \log N)$*; here, we do not count the description length of the function associated with the helper decryption key. Note that this is independent of the collusion bound Q (see Remark 3.31).*
- **Ciphertext:** *A ciphertext* ct *has size* $Q\ell_{\mathsf{out}} \cdot \tilde{O}(d) + \ell \cdot \mathsf{poly}(\lambda, d, \log N)$.

Moreover, the scheme has a transparent setup.

4 Silent Threshold Encryption

In this section, we compile our registered FE scheme from Section 3 into a silent threshold encryption scheme with ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$, where T is the threshold and N is the number of users. More generally, we construct a bounded-collusion distributed monotone-policy encryption scheme (that supports key aggregation) from any computational secret sharing scheme.

4.1 Secret Sharing

We recall the notion of a secret sharing scheme [Sha79, BL88]. In the standard information-theoretic setting, security requires that shares for any unauthorized set reveal no information about the message. We also consider the more general computational setting [Yao89, VNS+03] where we only require the unauthorized shares to computationally hide the message.

Definition 4.1 (Monotone Access Policy)**.** A *monotone access policy* on N parties is a monotone Boolean function $P\colon \{0,1\}^N \to \{0,1\}$ (i.e., if $P(\beta_1,\ldots,\beta_N)=1$ and $\beta'_i \geq \beta_i$ for all $i \in [N]$, then $P(\beta'_1,\ldots,\beta'_N)=1$). For a set $S \subseteq [N]$, we define

$$P(S) := P(\beta_1,\ldots,\beta_N) \quad \text{where} \quad \beta_i = 1 \text{ if } i \in S \text{ and } \beta_i = 0 \text{ otherwise.}$$

We say that a set $S \subseteq [N]$ is *authorized* for P if $P(S)=1$ and *unauthorized* otherwise.

Definition 4.2 (Secret Sharing)**.** Let $\mathcal{P}$ be a family of monotone access policies. A secret sharing scheme for $\mathcal{P}$ with per-party share size $\ell = \ell(\lambda)$ and randomness length $\rho_{\mathsf{SS}} = \rho_{\mathsf{SS}}(\lambda)$ is a pair of efficient algorithms $\Pi_{\mathsf{SS}} = (\mathsf{Share}, \mathsf{Reconst})$ with the following syntax:

- $\mathsf{Share}(P, i, \mu; r) \to (\sigma_i)$: On input a policy $P \in \mathcal{P}$ (over N parties), a party index $i \in [N]$, a message $\mu \in \{0,1\}$, and randomness $r \in \{0,1\}^{\rho_{\mathsf{SS}}}$, the deterministic sharing algorithm outputs the i^{th} share $\sigma_i \in \{0,1\}^{\ell}$.
- $\mathsf{Reconst}(P, T, \{(i,\sigma_i)\}_{i\in T}) \to \mu$: On input a policy $P \in \mathcal{P}$ (over N parties), a set $T \subseteq [N]$, and shares σ_i for $i \in T$, the reconstruction algorithm outputs a message $\mu \in \{0,1\}$.

We require the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, policies $P \in \mathcal{P}$ (over N parties), messages $\mu \in \{0,1\}$, and authorized sets $T \subseteq [N]$ where $P(T)=1$,

$$\Pr\left[\mathsf{Reconst}(P, T, \{(i,\sigma_i)\}_{i\in T}) = \mu : \begin{array}{c} r \xleftarrow{R} \{0,1\}^{\rho_{\mathsf{SS}}} \\ \forall i \in T : \sigma_i = \mathsf{Share}(P, i, \mu; r) \end{array}\right] = 1.$$

- **Non-adaptive security:** For an adversary $\mathcal{A}$ and a bit $b \in \{0,1\}$, we define the non-adaptive security game as follows:
  - **Setup:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ (over N parties) and a set of parties $T \subseteq [N]$.
  - **Challenge phase:** The challenger checks that $P(T)=0$ and halts with output 0 if not. Otherwise, the challenger samples $r \xleftarrow{R} \{0,1\}^{\rho_{\mathsf{SS}}}$, computes $\sigma_i = \mathsf{Share}(P, i, b; r)$ for each $i \in T$, and gives $\{\sigma_i\}_{i\in T}$ to $\mathcal{A}$.
  - **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.

  We say that Π_{SS} is secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b'=1 \mid b=0] - \Pr[b'=1 \mid b=1]| = \mathsf{negl}(\lambda) \tag{4.1}$$

  in the secret-sharing security game defined above. If Eq. (4.1) holds for all (possibly unbounded) adversaries $\mathcal{A}$, we say the scheme is *statistically secure*. If the distinguishing advantage is 0, we say the scheme is *perfectly secure*.

Additionally, we define the *share circuit complexity* of Π_{SS} to be the size of the smallest Boolean circuit family $\mathcal{F}$ of circuits $C_{P,j}(\mu, r) = \mathsf{Share}(P, j, \mu; r)$ that output the j^{th} share (as a function of the message and randomness) for a policy $P \in \mathcal{P}$ and an index j.

Classic secret sharing schemes like Shamir's scheme [Sha79] for threshold policies, or the linear secret sharing schemes for monotone Boolean formulas from [BL88, LW11] satisfy perfect security. More recently, Applebaum et al. [ABI+23] constructed *succinct* computational secret sharing schemes where the per-party share size ℓ is sublinear in the length of the policy description.

Remark 4.3 (Randomness Length)**.** In Definition 4.2, we assume that the randomness length ρ_{SS} is a fixed function of λ and in particular, independent of the number of parties N. We note that this is without loss of generality, since one can always use a pseudorandom generator (PRG) to expand a $\mathsf{poly}(\lambda)$-size seed into however many bits of randomness are required by the sharing algorithm. Computational security of the resulting scheme would then follow by security of the PRG.

Instantiation for threshold policies. For threshold policies, we can combine Shamir secret sharing [Sha79] with a pseudorandom function (PRF) that can be computed by polylogarithmic-depth Boolean circuits (e.g., the lattice-based scheme from [BPR12]) to obtain a computational threshold secret sharing scheme with the following properties:

Fact 4.4 (Computational Threshold Secret Sharing)**.** Let λ be a security parameter and $\mathcal{P} = \{P_{N,T}\}$ where $P_{N,T}$ is the threshold policy with N users and threshold $T \le N$ (i.e., $P_{N,T}(S) = 1$ if $|S| \ge T$). Then, under the LWE assumption (with a super-polynomial modulus-to-noise ratio), there exists a computationally-secure secret sharing scheme $\Pi_{\mathsf{SS}} = (\mathsf{Share}, \mathsf{Reconst})$ for $\mathcal{P}$ with the following properties:

- **Per-party share size:** The shares output by Share for a policy $P_{N,T} \in \mathcal{P}$ have size $\ell = O(\log N)$.
- **Randomness length:** The randomness length for Share is $\rho_{\mathsf{SS}} = \mathsf{poly}(\lambda)$.
- **Share circuit complexity:** The algorithm $\mathsf{Share}(P_{N,T}, \cdot, \cdot)$ can be computed by a Boolean circuit of size $T \cdot \mathsf{poly}(\lambda, \log N)$ and depth $\mathsf{poly}(\log \lambda, \log T)$.

4.2 Bounded-Collusion Distributed Monotone-Policy Encryption

In this section, we define the notion of distributed monotone-policy encryption with one-round distributed decryption, following [DJWW25, CW26a, AGY26a]. Our definition differs from [DJWW25, CW26a, AGY26a] in that additionally we parameterize the scheme by a collusion bound Q and only require security against adversaries that corrupt at most Q parties. Similar to [AGY26a], we also include a deterministic aggregation procedure (a requirement in the setting of silent threshold encryption [GKPW24]).

Definition 4.5 (Distributed Monotone-Policy Encryption)**.** Let λ be a security parameter, Q be a collusion bound, and $\mathcal{P}$ be a family of monotone access policies. A distributed monotone-policy encryption scheme with policy space $\mathcal{P}$ and collusion bound Q is a tuple of efficient algorithms $\Pi_{\mathsf{MPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, N) \to \mathsf{pp}$: On input the security parameter $\lambda \in \mathbb{N}$ and the number of parties $N \in \mathbb{N}$ (in binary), the setup algorithm outputs the public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{sk})$: On input the public parameters pp, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}) \to b$: On input the public parameters pp and a public key pk, the validity-checking algorithm outputs a bit $b \in \{0, 1\}$.
- $\mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), P) \to (\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N)$: On input a collusion bound $Q \in \mathbb{N}$, the public parameters pp, a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, and a policy $P \in \mathcal{P}$, the aggregation algorithm outputs a master public key mpk and helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_N$. This algorithm is deterministic.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mu) \to \mathsf{ct}$: On input the master public key mpk and a message $\mu \in \{0, 1\}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{GetHint}(\mathsf{sk}_i, \mathsf{ct}) \to \mathsf{ht}_i$: On input a secret key sk_i and a ciphertext ct, the hint-generation algorithm outputs a decryption hint ht_i.
- $\mathsf{Decrypt}(P, \{(i, \mathsf{ht}_i, \mathsf{hsk}_i)\}_{i \in T}, \mathsf{ct}) \to \mu$: On input the policy $P \in \mathcal{P}$, a list of hints and helper decryption keys $(\mathsf{ht}_i, \mathsf{hsk}_i)$ for $i \in T \subseteq [N]$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \{0, 1\}$.

Moreover, Π_{MPE} should satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, all $N \in \mathbb{N}$, all collusion bounds $Q \in \mathbb{N}$, all policies $P \in \mathcal{P}$ (over N parties), all sets $T \subseteq [N]$ where $P(T) = 1$, all messages $\mu \in \{0,1\}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, N)$, all $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for $i \in T$, and all public keys pk_i for $i \in [N] \setminus T$,

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

  where we have $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), P)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mu)$, and $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{sk}_i, \mathsf{ct})$ for all $i \in T$.

- **Completeness:** For all security parameters $\lambda \in \mathbb{N}$, all $N \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, N)$, and all $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$, we have $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}) = 1$.

- **Security:** For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the security game as follows:

  - **Setup:** At the beginning of the game, the adversary outputs the number of parties N and the challenger samples $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, N)$ 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 $\mathsf{ctr}' \leq \mathsf{ctr}$, and the challenger replies with $\mathsf{sk}_{\mathsf{ctr}'}$. The challenger adds ctr' to C.
    - **Hint-generation query:** In a hint-generation query, the adversary specifies a ciphertext ct and a counter value $\mathsf{ctr}' \leq \mathsf{ctr}$. The challenger replies with $\mathsf{GetHint}(\mathsf{sk}_{\mathsf{ctr}'}, \mathsf{ct})$.
  - **Challenge phase:** In the challenge phase, the adversary specifies a challenge policy $P \in \mathcal{P}$ on N users and a collusion bound 1^Q such that $Q \leq N$. In addition, for each $i \in [N]$, algorithm $\mathcal{A}$ either specifies a public key pk_i or a counter value ctr_i. For each $i \in [N]$ where $\mathcal{A}$ specifies a counter value $\mathsf{ctr}_i \leq \mathsf{ctr}$, the challenger sets $\mathsf{pk}_i = \mathsf{pk}_{\mathsf{ctr}_i}$. The challenger computes $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), P)$ and replies with $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, b)$.
  - **Post-challenge query phase:** The adversary can continue to make corruption and hint-generation queries. If the adversary asks for a hint on the challenge ciphertext ct^*, the counter value ctr' is added to C.
  - **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.

  For each $i \in [N]$, let $\beta_i = 1$ if the adversary chose the key itself or a corrupted counter value $\mathsf{ctr}_i \in C$ during the challenge phase, and $\beta_i = 0$ otherwise. We say that $\mathcal{A}$ is *admissible* if $P(\beta_1, \ldots, \beta_N) = 0$, $\sum_{i \in [N]} \beta_i \leq Q$, and for each $i \in [N]$ where $\mathcal{A}$ specifies a public key pk_i, we have $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i) = 1$. We say that Π_{MPE} is *secure* if for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda) \tag{4.2}$$

  in the security game defined above.

- **Succinctness:** There exists a polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $\lambda \in \mathbb{N}$, all $N \in \mathbb{N}$, all collusion bounds $Q \in \mathbb{N}$, all public parameters pp in the support of $\mathsf{Setup}(1^\lambda, N)$, all policies $P \in \mathcal{P}$ (over N parties), all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, and setting $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), P)$, we have that $|\mathsf{mpk}|, |\mathsf{hsk}_i| = \mathsf{poly}(\lambda, Q, \log N)$ for all $i \in [N]$.

Definition 4.6 (Static Security)**.** Let Π_{MPE} be a distributed monotone-policy encryption scheme with collusion bound Q. We define the *static security* game for Π_{MPE} by restricting the adversary in the security game from Definition 4.5 as follows:

- At the beginning of the setup phase, the adversary commits to the set of corrupted parties $C \subseteq [N]$.
- The adversary cannot make any corruption queries in either phase.
- In the post-challenge query phase, the adversary cannot make a hint-generation query on the challenge ciphertext ct^*.
- During the challenge phase, the adversary is required to specify a public key pk_i of its own choosing for all $i \in C$ and a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$ for all $i \notin C$.

Since there are no corruption queries in the static security game, the values $\beta_1, \ldots, \beta_N$ from Definition 4.5 satisfy $\beta_i = 1$ if and only if $i \in C$. Thus, in this setting, an adversary is *admissible* if C is unauthorized (i.e., $P(C) = 0$), $|C| \le Q$, and $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i) = 1$ for all $i \in C$. We say that Π_{MPE} satisfies *static security* if for all efficient and admissible adversaries $\mathcal{A}$, Eq. (4.2) holds in the static security game.

4.3 Silent Threshold Encryption from Succinct Bounded-Collusion Registered FE

In this section, we show how to construct silent threshold encryption with ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$ from slotted succinct bounded-collusion registered FE, where T is the threshold and N is the number of users. Specifically, we first show how to construct bounded-collusion distributed monotone-policy encryption for any policy with a computational secret sharing scheme and then instantiate the computational secret sharing scheme for threshold policies using Shamir secret sharing [Sha79] with randomness derived from a low-depth PRF.

Construction 4.7 (Distributed Monotone-Policy Encryption)**.** Let λ be a security parameter, N be the number of parties, and Q be a collusion bound. We define the following:

- Let $\mathcal{P}$ be a family of monotone access policies over N parties. Let $\Pi_{\mathsf{SS}} = (\mathsf{SS.Share}, \mathsf{SS.Reconst})$ be a computational secret sharing scheme for $\mathcal{P}$ with per-party share size $\ell = \ell(\lambda)$ and share randomness length $\rho_{\mathsf{SS}} = \rho_{\mathsf{SS}}(\lambda)$.
- Let $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ be the function family where $\mathcal{F}_\lambda$ contains the share-generation circuits $C_{P,j} \colon \{0,1\}^{\rho_{\mathsf{SS}}(\lambda)+1} \to \{0,1\}^{\ell(\lambda)}$ for all policies $P \in \mathcal{P}$ over N parties and all $j \in [N]$, where
$$C_{P,j}(\mu, r) := \mathsf{SS.Share}(P, j, \mu; r).$$
- Let $\Pi_{\mathsf{RFE}} = (\mathsf{RFE.Setup}, \mathsf{RFE.KeyGen}, \mathsf{RFE.IsValid}, \mathsf{RFE.Aggregate}, \mathsf{RFE.Encrypt}, \mathsf{RFE.GetHint}, \mathsf{RFE.Recover})$ be a slotted succinct bounded-collusion registered FE scheme (Definition 3.1) for the function family $\mathcal{F}$. Since each share circuit has ℓ-bit outputs, we require Π_{RFE} to support functions with ℓ-bit outputs (see Remark 3.32).

We construct a distributed monotone-policy encryption scheme $\Pi_{\mathsf{MPE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{GetHint}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda, N)$: Output $\mathsf{RFE.Setup}(1^\lambda, 1^\lambda, N)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Output $\mathsf{RFE.KeyGen}(\mathsf{pp})$.
- $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk})$: Output $\mathsf{RFE.IsValid}(\mathsf{pp}, \mathsf{pk})$.
- $\mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), P)$: Output $\mathsf{RFE.Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), (C_{P,1}, \ldots, C_{P,N}))$.
- $\mathsf{Encrypt}(\mathsf{mpk}, \mu)$: Sample $r \xleftarrow{R} \{0,1\}^{\rho_{\mathsf{SS}}}$ and output $\mathsf{RFE.Encrypt}(\mathsf{mpk}, (\mu, r))$.
- $\mathsf{GetHint}(\mathsf{sk}_i, \mathsf{ct})$: Output $\mathsf{RFE.GetHint}(\mathsf{sk}_i, \mathsf{ct})$.
- $\mathsf{Decrypt}(P, \{(i, \mathsf{ht}_i, \mathsf{hsk}_i)\}_{i \in T}, \mathsf{ct})$: For each $i \in T$, compute the shares $\sigma_i = \mathsf{RFE.Recover}(\mathsf{ht}_i, \mathsf{hsk}_i, \mathsf{ct})$. Output $\mathsf{SS.Reconst}(P, T, \{(i, \sigma_i)\}_{i \in T})$.

Theorem 4.8 (Correctness)**.** *Suppose* Π_{RFE} *and* Π_{SS} *are correct. Then, Construction 4.7 is correct.*

Proof. Take any $\lambda \in \mathbb{N}$, any $N \in \mathbb{N}$, any collusion bound $Q \le N$, any policy $P \in \mathcal{P}$ over N parties, any set $T \subseteq [N]$ where $P(T) = 1$, any message $\mu \in \{0,1\}$, any pp in the support of $\mathsf{Setup}(1^\lambda, N)$, any $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{pp})$ for $i \in T$, and any choice of pk_i for $i \notin T$. Let

$$(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1, \ldots, \mathsf{pk}_N), P)$$

and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, \mu)$. For each $i \in T$, let $\mathsf{ht}_i \leftarrow \mathsf{GetHint}(\mathsf{sk}_i, \mathsf{ct})$. Now, consider the output of

$$\mathsf{Decrypt}(P, \{(i, \mathsf{ht}_i, \mathsf{hsk}_i)\}_{i \in T}, \mathsf{ct}).$$

By correctness of Π_{RFE}, we have

$$\sigma_i = \mathsf{RFE.Recover}(\mathsf{ht}_i, \mathsf{hsk}_i, \mathsf{ct}) = \mathsf{SS.Share}(P, i, \mu; r)$$

where $r \in \{0,1\}^{\rho_{\mathsf{SS}}}$ is the randomness sampled by Encrypt. By correctness of Π_{SS}, we have

$$\mathsf{Decrypt}(P, \{(i, \mathsf{ht}_i, \mathsf{hsk}_i)\}_{i \in T}, \mathsf{ct}) = \mathsf{SS.Reconst}(P, T, \{(i, \sigma_i)\}_{i \in T}) = \mu. \qquad \square$$

Theorem 4.9 (Static Security)**.** *Suppose Π_{RFE} satisfies input-selective static CCA security and Π_{SS} is secure. Then, Construction 4.7 is statically secure.*

Proof. Take any efficient adversary $\mathcal{A}$ for the static security game. We 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^λ, the adversary outputs the number of parties N and the set of corrupted parties $C \subseteq [N]$. The challenger gives $\mathsf{pp} \leftarrow \mathsf{RFE.Setup}(1^\lambda, 1^\lambda, N)$ 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 $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{RFE.KeyGen}(\mathsf{pp})$, and replies with $\mathsf{pk}_{\mathsf{ctr}}$.
  - **Pre-challenge hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ for $c \le \mathsf{ctr}$, the challenger replies with $\mathsf{RFE.GetHint}(\mathsf{sk}_c, \hat{\mathsf{ct}})$.
  - **Challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ over N parties, a collusion bound 1^Q where $Q \le N$ and $|C| \le Q$. Algorithm $\mathcal{A}$ specifies a public key pk_i^* for $i \in C$ and a counter value ctr_i for each $i \in [N] \setminus C$. For each $i \in [N] \setminus C$, the challenger sets $\mathsf{pk}_i^* = \mathsf{pk}_{\mathsf{ctr}_i}$. For each $i \in C$, the challenger checks that $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i^*) = 1$ and halts with output 0 if not. Otherwise, the challenger computes

    $$(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_N) = \mathsf{RFE.Aggregate}(1^Q, \mathsf{pp}, (\mathsf{pk}_1^*, \ldots, \mathsf{pk}_N^*), (C_{P,1}, \ldots, C_{P,N})).$$

    The challenger then samples $r \xleftarrow{R} \{0,1\}^{\rho_{\mathsf{SS}}}$ and gives $\mathsf{ct}^* \leftarrow \mathsf{RFE.Encrypt}(\mathsf{mpk}, (b, r))$ to $\mathcal{A}$.
  - **Post-challenge hint-generation queries:** Algorithm $\mathcal{A}$ can continue to make hint-generation queries on ciphertexts $(\hat{\mathsf{ct}}, c)$ 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.
- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$ except the challenger uses the RFE simulator $\mathcal{S} = (\mathcal{S}_{\mathsf{pp}}, \mathcal{S}_{\mathsf{kg}}, \mathcal{S}_{\mathsf{ct}}, \mathcal{S}_{\mathsf{ht}})$ instead of the honest RFE algorithms. Here, $\mathcal{S}$ is the simulator for Π_{RFE} from Definition 3.10. Specifically, the challenger now does the following:
  - **Setup phase:** The challenger now samples $(\mathsf{pp}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{pp}}(1^\lambda, 1^\lambda, N, C)$ and gives pp to $\mathcal{A}$.
  - **Key-generation queries:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger increments $\mathsf{ctr} = \mathsf{ctr} + 1$, samples $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{kg}}(\mathsf{st}_{\mathcal{S}})$, and replies with $\mathsf{pk}_{\mathsf{ctr}}$.

- **Pre-challenge hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ for $c \le \mathsf{ctr}$, the challenger computes $(\mathsf{ht}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{ht}}(\mathsf{st}_{\mathcal{S}}, c, \hat{\mathsf{ct}})$ and replies with ht.
- **Challenge phase:** The challenger checks that $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i^*) = 1$ for each $i \in C$ and halts with output 0 if not (exactly as in $\mathsf{Hyb}_0^{(b)}$). Otherwise, it samples $r \xleftarrow{R} \{0,1\}^{\rho_{\mathsf{SS}}}$ and defines the following quantities for each $i \in [N]$:
  - If $i \in C$, the challenger sets $\mathsf{ctr}_i = \bot$ and $y_i = C_{P,i}(b, r) = \mathsf{SS.Share}(P, i, b; r)$.
  - If $i \in [N] \setminus C$, then ctr_i is the counter value $\mathcal{A}$ specified for slot i, and the challenger sets $y_i = \bot$.

  The challenger then computes $(\mathsf{ct}^*, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{ct}}(\mathsf{st}_{\mathcal{S}}, 1^Q, \{(\mathsf{ctr}_i, C_{P,i}, \mathsf{pk}_i^*, y_i)\}_{i \in [N]})$ and gives ct^* to $\mathcal{A}$.
- **Post-challenge hint-generation queries:** The challenger answers these queries exactly like the pre-challenge hint-generation queries (i.e., using $\mathcal{S}_{\mathsf{ht}}$).
- **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.

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

Lemma 4.10. *Suppose Π_{RFE} satisfies input-selective static CCA security. Then, there exists a negligible function $\mathsf{negl}(\lambda)$ 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. Suppose $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| = \varepsilon$ for some $b \in \{0,1\}$ and some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks input-selective static CCA security of Π_{RFE}:

- **Setup phase:** Algorithm $\mathcal{B}$ samples $r \xleftarrow{R} \{0,1\}^{\rho_{\mathsf{SS}}}$ and gives $(1^\lambda, 1^\lambda, N, C, \mathbf{x})$, where $\mathbf{x} = (b, r)$, to the challenger to get the public parameters pp. Algorithm $\mathcal{B}$ gives pp to $\mathcal{A}$.
- **Key-generation queries:** When algorithm $\mathcal{A}$ makes a key-generation query, algorithm $\mathcal{B}$ makes a key-generation query to the challenger to get $\mathsf{pk}_{\mathsf{ctr}}$, which it forwards to $\mathcal{A}$.
- **Hint-generation queries:** When algorithm $\mathcal{A}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$, algorithm $\mathcal{B}$ makes a hint-generation query $(\hat{\mathsf{ct}}, c)$ to the challenger to get ht, which it forwards to $\mathcal{A}$.
- **Challenge phase:** When algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ over N parties, a collusion bound 1^Q where $Q \le N$, a public key pk_i^* (which sets $\mathsf{ctr}_i = \bot$) for $i \in C$, and a counter value ctr_i for each $i \in [N] \setminus C$, algorithm $\mathcal{B}$ outputs 0 if $|C| > Q$. For each $i \in [N]$ where $\mathcal{A}$ specifies a counter value $\mathsf{ctr}_i \le \mathsf{ctr}$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i^* = \mathsf{pk}_{\mathsf{ctr}_i}$. Algorithm $\mathcal{B}$ gives $(1^Q, \{(\mathsf{ctr}_i, C_{P,i}, \mathsf{pk}_i^*)\}_{i \in [N]})$ to the challenger to get ct^*, which it forwards to $\mathcal{A}$. Note that the registered FE challenger checks that $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i^*) = 1$ for each $i \in C$ (and halts with output 0 if not), which matches the behavior in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$.
- **Output phase:** At the end of the game, if algorithm $\mathcal{A}$ outputs $b' \in \{0,1\}$, algorithm $\mathcal{B}$ also outputs b'.

By admissibility of $\mathcal{A}$, algorithm $\mathcal{B}$ is admissible. If the registered FE challenger uses the real algorithms to construct each component, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_0^{(b)}$. If the registered FE challenger constructs the components using the simulator, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_1^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks input-selective static CCA security with the same advantage ε. □

Lemma 4.11. *Suppose Π_{SS} is secure. Then, there exists a negligible function $\mathsf{negl}(\lambda)$ such that for all $b \in \{0,1\}$ and all $\lambda \in \mathbb{N}$, $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.*

Proof. Suppose $|\Pr[\mathsf{Hyb}_1^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(1)}(\mathcal{A}) = 1]| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient adversary $\mathcal{B}$ that breaks security of Π_{SS}:

- **Setup and query phases:** Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on input 1^λ. Algorithm $\mathcal{A}$ outputs the number of parties N and the set of corrupted parties $C \subseteq [N]$. Algorithm $\mathcal{B}$ executes the remainder of the setup and query phases exactly as described in $\mathsf{Hyb}_1^{(b)}$ since these phases do not depend on b (or the secret sharing scheme).
- **Challenge phase:** Algorithm $\mathcal{A}$ outputs a policy $P \in \mathcal{P}$ over N parties, a collusion bound 1^Q where $Q \leq N$ and $|C| \leq Q$, a public key pk_i^* for each $i \in C$, and a counter value ctr_i for each $i \in [N] \setminus C$. For each $i \in [N] \setminus C$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i^* = \mathsf{pk}_{\mathsf{ctr}_i}$, and for each $i \in C$, it sets $\mathsf{ctr}_i = \bot$. Next, algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{pp}, \mathsf{pk}_i^*) = 1$ for each $i \in C$ and halts with output 0 if not. Otherwise, algorithm $\mathcal{B}$ submits the policy P together with the set C to the secret sharing challenger, which replies with shares $\{y_i\}_{i \in C}$. For each $i \in [N] \setminus C$, algorithm $\mathcal{B}$ sets $y_i = \bot$. Finally, algorithm $\mathcal{B}$ constructs the challenge ciphertext by computing

$$(\mathsf{ct}^*, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_{\mathsf{ct}}\left(\mathsf{st}_{\mathcal{S}}, 1^Q, \left\{(\mathsf{ctr}_i, C_{P,i}, \mathsf{pk}_i^*, y_i)\right\}_{i \in [N]}\right)$$

  and gives ct^* to $\mathcal{A}$.
- **Output phase:** At the end of the game, if algorithm $\mathcal{A}$ outputs $b' \in \{0,1\}$, $\mathcal{B}$ also outputs b'.

Since $\mathcal{A}$ is admissible, we have $P(C) = 0$. In this case, if the secret sharing challenger shares 0, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_1^{(0)}$, and if the secret sharing challenger shares 1, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_1^{(1)}$. Thus, algorithm $\mathcal{B}$ breaks secret sharing security with the same advantage ε. □

Security now follows via Lemmas 4.10 and 4.11 and a hybrid argument. □

Instantiations. Combining Construction 4.7 with Theorems 4.8 and 4.9 and instantiating the underlying registered FE scheme with Corollary 3.33, we obtain the following corollary:

Corollary 4.12 (Distributed Monotone-Policy Encryption with Bounded Corruptions)**.** *Let λ be a security parameter, $Q = Q(\lambda)$ be a collusion bound, and $\mathcal{P}$ be a policy family over N parties with a secret sharing scheme with per-party share size $\ell = \ell(\lambda)$ and randomness length $\rho_{\mathsf{SS}} = \rho_{\mathsf{SS}}(\lambda)$. Suppose the share-generation algorithm can be computed by a Boolean circuit with depth $d = d(\lambda)$ and size $s = s(\lambda)$. 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 $\mathcal{P}$ with collusion bound Q in the random oracle model. The scheme satisfies the following properties:*

- **Public parameters, user keys, and decryption hints:** *The public parameters* pp*, each individual user's public key* pk *and secret key* sk*, and the decryption hints* ht *all have size* $\mathsf{poly}(\lambda, d, \log \rho_{\mathsf{SS}}, \log s, \log N)$.
- **Aggregated public key:** *The size of the aggregated public key* mpk *is* $\mathsf{poly}(\lambda, d, \log \rho_{\mathsf{SS}}, \log s, \log N)$.
- **Helper decryption keys:** *The size of each user's helper decryption key is* $\ell \cdot \mathsf{poly}(\lambda, d, \log \rho_{\mathsf{SS}}, \log s, \log N)$.
- **Ciphertext:** *A ciphertext* ct *has size* $Q\ell \cdot \tilde{O}(d) + \rho_{\mathsf{SS}} \cdot \mathsf{poly}(\lambda, d, \log N)$.

Moreover, the scheme has a transparent setup.

Finally, we instantiate Corollary 4.12 with the threshold secret sharing scheme from Fact 4.4 and the collusion bound $Q = T - 1$. In this case, the per-party share size is $\ell = O(\log N)$, the randomness length is $\rho_{\mathsf{SS}} = \mathsf{poly}(\lambda)$, and the share-generation circuit has depth $d = \mathsf{poly}(\log \lambda, \log T) = \tilde{O}(1)$ and size $s = T \cdot \mathsf{poly}(\lambda, \log N)$. This yields the following corollary:

Corollary 4.13 (Silent Threshold Encryption)**.** *Let λ be a security parameter, N be the number of users, and T be a threshold. Then, under the decomposed LWE assumption with a sub-exponential modulus-to-noise ratio, there exists a statically-secure silent threshold encryption scheme in the random oracle model with the following properties:*

- **Public parameters, user keys, and decryption hints:** *The public parameters* pp*, each individual user's public key* pk *and secret key* sk*, and the decryption hints* ht *all have size* $\mathsf{poly}(\lambda, \log N)$.

- **Aggregated public key and helper decryption keys:** *The aggregated public key* mpk *and each user's helper decryption key have size* $\mathsf{poly}(\lambda, \log N)$.
- **Ciphertext:** *A ciphertext* ct *has size* $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$.

Moreover, the scheme has a transparent setup.

Remark 4.14 (Supporting Dynamic Thresholds)**.** Following a similar compiler from [BLM+24], we can use a generic power-of-two approach to support *dynamic* thresholds with only logarithmic overhead. For completeness, we briefly describe the approach here. First, we say a silent threshold encryption scheme supports dynamic thresholds if the threshold T is provided at encryption time rather than aggregation time. At a high level, we run $n = \lceil \log N \rceil$ instances of Construction 4.7, where the i^{th} instance is instantiated with collusion bound 2^i and supports all thresholds $T \leq 2^i$:

- Since the actual threshold T is only determined at *encryption* time, it is now provided as an input to the encryption algorithm. Thus, we modify the function associated with user j to be the circuit

$$C_j(\mu, T, r) := \mathsf{SS.Share}(P_{N,T}, j, \mu; r).$$

  Again, by Fact 4.4, such a function can be computed by a Boolean circuit of size $2^i \cdot \mathsf{poly}(\lambda, \log N)$ and depth $\mathsf{poly}(\log \lambda, \log N)$ in the i^{th} instance (which only needs to support thresholds $T \leq 2^i$).
- The aggregation algorithm now aggregates the users' public keys in each of the n instances, where the i^{th} instance is aggregated with collusion bound 2^i. The master public key consists of all n aggregated master public keys and each user's helper decryption key now consists of the n helper keys for the n instances.
- To encrypt with respect to a threshold T, the encrypter encrypts the triple (μ, T, r) using the master public key of the i^{th} instance, where $i = \lceil \log T \rceil$. The threshold T is also included in the clear, so that the decrypters know which instance to use and how to reconstruct. Decryption is unchanged. Each user recovers its share of μ using its helper key for the i^{th} instance and then reconstructs the message from the shares.

Security follows from the fact that the i^{th} instance is secure against up to $2^i \geq T$ colluding users. The transformation increases the size of the master public key and the size of each user's helper decryption key by a factor of $n = O(\log N)$, and the size of the ciphertext by a factor of at most 2 (from needing to use $2^{\lceil \log T \rceil}$ as the collusion bound instead of T). The parameter sizes from Corollary 4.13 are therefore unaffected.

Acknowledgments

We thank Russell Lai for the pointer to Shamir secret sharing over cyclotomic rings with small interpolation coefficients. 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.

Statement on AI use. All of the ideas in this paper were developed without the use of AI. After developing the main ideas, we used Claude to provide initial drafts of the technical overview and some of the proofs as well as for subsequent copy-editing passes to correct spelling and grammar errors. The authors read over the initial drafts written by Claude and made extensive edits to improve the exposition. The authors verified the correctness and originality of all of the content in the paper.

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.

[ABV+12] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy IBE) from lattices. In *PKC*, 2012.

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

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO*, 2013.

[AGY26a] 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.

[AGY26b] Abtin Afshar, Rishab Goyal, and Saikumar Yadugiri. Silent distributed cryptography for DNFs and threshold policies from lattices. *IACR Cryptol. ePrint Arch.*, 2026.

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *Comput. Complex.*, 15(2), 2006.

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

[AL21] Martin R. Albrecht and Russell W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. In *CRYPTO*, 2021.

[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.

[AMYY25] Shweta Agrawal, Anuja Modi, Anshu Yadav, and Shota Yamada. Zeroizing attacks against evasive and circular evasive LWE. In *TCC*, 2025.

[ANP23] Benny Applebaum, Oded Nir, and Benny Pinkas. How to recover a secret with o(n) additions. In *CRYPTO*, 2023.

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In *TCC*, 2017.

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In *TCC*, 2019.

[AWY20] Shweta Agrawal, Daniel Wichs, and Shota Yamada. Optimal broadcast encryption from LWE and pairings in the standard model. In *TCC*, 2020.

[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.

[BCF+25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyan Qu, and Mingyuan Wang. BEAST-MEV: batched threshold encryption with silent setup for MEV prevention. *IACR Cryptol. ePrint Arch.*, 2025. Available at https://eprint.iacr.org/2025/1419.pdf.

[BÇM21] Marshall Ball, Alper Çakan, and Tal Malkin. Linear threshold secret-sharing with binary reconstruction. In *Conference on Information-Theoretic Cryptography*, 2021.

[BCTW16] Zvika Brakerski, David Cash, Rotem Tsabary, and Hoeteck Wee. Targeted homomorphic attribute-based encryption. In *TCC*, 2016.

[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.

[BDJ+25] Pedro Branco, Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, Spencer Peters, and Vinod Vaikuntanathan. Pseudorandom obfuscation and applications. In *CRYPTO*, 2025.

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

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, 2018.

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

[BLM+24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor tracing without trusted authority from registered functional encryption. In *ASIACRYPT*, 2024.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, 2012.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*, 1993.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private constrained PRFs (and more) from LWE. In *TCC*, 2017.

[BÜW24] Chris Brzuska, Akin Ünal, and Ivy K. Y. Woo. Evasive LWE assumptions: Definitions, classes, and counterexamples. In *ASIACRYPT*, 2024.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 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.

[CLW25] Valerio Cini, Russell W. F. Lai, and Ivy K. Y. Woo. Pilvi: Lattice threshold PKE with small decryption shares and improved security. In *ASIACRYPT*, 2025.

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

[CW26a] Jeffrey Champion and David J. Wu. Distributed monotone-policy encryption for DNFs from lattices. In *EUROCRYPT*, 2026.

[CW26b] Jeffrey Champion and David J. Wu. Optimal distributed monotone-policy encryption for DNFs and more from lattices. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/1464.pdf.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *STOC*, 1994.

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

[Des87] Yvo Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO*, 1987.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, 1989.

[DJM+25] Nico Döttling, Abhishek Jain, Giulio Malavolta, Surya Mathialagan, and Vinod Vaikuntanathan. Simple and general counterexamples for private-coin evasive LWE. In *CRYPTO*, 2025.

[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.

[DPY24] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE beyond predicates: (attribute-based) linear functions and more. In *ASIACRYPT*, 2024.

[DR82] Arkadii Georgievich D'yachkov and Vladimir Vasil'evich Rykov. Bounds on the length of disjunctive codes. *Problemy Peredachi Informatsii*, 18(3), 1982.

[EFF85] Paul Erdős, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel J. Math*, 51(1-2), 1985.

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In *ASIACRYPT*, 2023.

[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.

[Fra89] Yair Frankel. A practical protocol for large group oriented networks. In *EUROCRYPT*, 1989.

[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.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GJM+24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *IEEE S&P*, 2024.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, 2013.

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

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In *CRYPTO*, 2024.

[Gol20] Oded Goldreich. On (Valiant's) polynomial-size monotone formula for majority. In *Computational Complexity and Property Testing - On the Interplay Between Randomness and Computation*, volume 12050 of *Lecture Notes in Computer Science*. Springer, 2020.

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

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, 2015.

[GWWW26] Junqing Gong, Brent Waters, Hoeteck Wee, and David J. Wu. Threshold batched identity-based encryption from pairings in the plain model. In *EUROCRYPT*, 2026.

[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. Distributed monotone policy encryption with stronger security for DNFs and threshold policies from lattices. *IACR Cryptol. ePrint Arch.*, 2026. Available at https://eprint.iacr.org/2026/1609.pdf.

[GY26c] Rishab Goyal and Saikumar Yadugiri. Equivocal broadcast encryption: Adaptively-secure optimal distributed broadcast encryption from lattices. In *CRYPTO*, 2026.

[HHY26] Tzu-Hsiang Huang, Wei-Hsiang Hung, and Shota Yamada. A note on obfuscation-based attacks on private-coin evasive LWE. *IACR Commun. Cryptol.*, 2(4), 2026.

[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.

[HJL25] Yao-Ching Hsieh, Aayush Jain, and Huijia Lin. Lattice-based post-quantum iO from circular security with random opening assumption. In *CRYPTO*, 2025.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

[HSW25] Mathias Hall-Andersen, Mark Simkin, and Benedikt Wagner. Silent threshold encryption with one-shot adaptive security. *IACR Cryptol. ePrint Arch.*, 2025. Available at https://eprint.iacr.org/2025/1384.pdf.

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *FOCS*, 2000.

[KLNO24] Michael Klooß, Russell W. F. Lai, Ngoc Khanh Nguyen, and Michal Osadnik. RoK, paper, SISsors toolkit for lattice-based succinct arguments - (extended abstract). In *ASIACRYPT*, 2024.

[KS64] William H. Kautz and Richard C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theory*, 10(4), 1964.

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

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

[LWW25] George Lu, Brent Waters, and David J. Wu. Multi-authority registered attribute-based encryption. In *EUROCRYPT*, 2025.

[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.*, 2010:556, 2010. Available at https://eprint.iacr.org/2010/556.pdf.

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

[PST25] Tapas Pal, Robert Schädlich, and Erkan Tairi. Registered functional encryption for pseudorandom functionalities from lattices: Registered ABE for unbounded depth circuits and Turing machines, and more. *IACR Cryptol. ePrint Arch.*, 2025. Available at https://eprint.iacr.org/2025/967.pdf.

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

[RSY21] Leonid Reyzin, Adam D. Smith, and Sophia Yakoubov. Turning HATE into LOVE: compact homomorphic ad hoc threshold encryption for scalable MPC. In *Cyber Security Cryptography and Machine Learning*, 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.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM CCS*, 2010.

[SWW26] Roy Stracovsky, Brent Waters, and David J. Wu. Pairing-based registered ABE for Boolean formulas with a linear-size CRS. In *CRYPTO*, 2026.

[Val84] Leslie G. Valiant. Short monotone formulae for the majority function. *J. Algorithms*, 5(3), 1984.

[VNS+03] Vinod Vaikuntanathan, Arvind Narayanan, K. Srinathan, C. Pandu Rangan, and Kwangjo Kim. On the power of computational secret sharing. In *INDOCRYPT*, 2003.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and Null-IO from evasive LWE. In *ASIACRYPT*, 2022.

[WC81] Mark N. Wegman and Larry Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3), 1981.

[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.

[WQZD10] 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.

[WWW22] Brent Waters, Hoeteck Wee, and David J. Wu. Multi-authority ABE from lattices without random oracles. In *TCC*, 2022.

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

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

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

[ZCHZ24] Yijian Zhang, Jie Chen, Debiao He, and Yuqing Zhang. Bounded collusion-resistant registered functional encryption for circuits. In *ASIACRYPT*, 2024.

[ZLZ+24] Ziqi Zhu, Jiangtao Li, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered functional encryptions from pairings. In *EUROCRYPT*, 2024.

A An Alternative Approach to Build Registered FE

In Section 4, we showed how to obtain silent threshold encryption with ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$, where T is the threshold, N is the number of users, and λ is the security parameter. The main ingredient we used is a slotted succinct bounded-collusion registered FE (cf. Section 3). In this section, we describe two alternative approaches for building slotted bounded-collusion registered FE for general circuits:

- The first approach relies on a distributed broadcast encryption scheme [CW24, CHW25, AMR25, WW25, GY26c, GY26a, AMR26, CW26b] and has ciphertext size that is polynomial in the circuit size. This construction follows the template from [ZCHZ24], except we additionally sketch how to support adversarially-chosen public keys.
- The second approach is based on registered attribute-based encryption (ABE) [CHW25, WW25] and follows the approach of Goldwasser et al. [GKP+13]. This yields a scheme with succinct ciphertexts (which, like Corollary 3.33, only scale with the depth of the circuit).

In conjunction with our compiler in Section 4, both approaches yield a silent threshold encryption scheme with ciphertext size $\mathsf{poly}(\lambda, T)$ from the decomposed LWE assumption in the random oracle model. Thus, both schemes still achieve non-trivial succinctness for sufficiently small thresholds $T = N^{\varepsilon}$ (for a sufficiently small constant $\varepsilon < 1$). Since these alternative constructions achieve worse succinctness than the construction from Corollary 4.13, we provide only a high-level sketch of these approaches here. Throughout this section, we consider the simulation-based security notion from Definition 3.1, restricted to the static setting of Definition 3.3 where the adversary does not make any corruption queries. As in the main body, the adversary may still choose the public keys in the corrupted slots itself.

Overview of the approach. The construction consists of two steps. First, we construct a registered *single-key* FE scheme. Then following [GVW15, AR17, AV19], we show how to compile the single-key scheme into a Q-bounded-collusion registered FE scheme, where Q is the collusion bound. The first step can be realized by adapting the single-key FE construction of Sahai and Seyalioglu [SS10], which combines garbled circuits with public-key encryption, to the registered setting. Alternatively, a more succinct instantiation can be obtained by adapting the construction of succinct FE from attribute-based encryption, leveled homomorphic encryption, and garbled circuits from the work of Goldwasser et al. [GKP+13]. We start by describing the bootstrapping step.

The [GVW12] compiler. We consider the approach of Gorbunov, Vaikuntanathan, and Wee [GVW12], who described a generic approach to transform a single-key FE scheme into a bounded-collusion FE scheme. The work of [ZCHZ24] also showed how to combine a distributed broadcast encryption scheme with the [GVW15] compiler to construct a bounded-collusion registered FE scheme for general circuits. However, their approach did not consider adversarially-chosen keys, which is critical for the application to silent threshold encryption. Here, we sketch a version that handles adversarially-chosen keys. At a high level, the compiler runs M independent instances of a single-key FE scheme. Namely, the public key for the scheme consists of $(\mathsf{mpk}_1, \ldots, \mathsf{mpk}_M)$, where mpk_i is the master public key for the i^{th} scheme. Let Q denote the collusion bound and let $M = \mathsf{poly}(Q)$ be sufficiently large (e.g., $M = \Omega(Q^5)$). The scheme has the following structure:

- In the centralized setting, to generate a secret key for a function f, the central authority first selects a random subset $\Gamma \subseteq [M]$. The authority then generates a secret key for f in each FE instance indexed by Γ.
- To encrypt an input $\mathbf{x}$, the encrypter first constructs Shamir secret shares $\mathbf{x}_1, \ldots, \mathbf{x}_M$ of $\mathbf{x}$ and then for each $i \in [M]$, encrypts $\mathbf{x}_i$ under mpk_i. The ciphertext consists of the M encrypted shares.
- The basic version of the compiler supports evaluating functions in NC^1.[7] Specifically, suppose a user has a key for a bounded-degree polynomial f. For each $i \in \Gamma$, the user can decrypt to learn $f(\mathbf{x}_i)$. By setting the parameters accordingly, given $f(\mathbf{x}_i)$ for all $i \in \Gamma$, the user can interpolate the polynomial and learn $f(\mathbf{x})$. Reconstruction relies on the fact that one can homomorphically evaluate low-degree polynomial on Shamir secret-shared values. For the particular application to silent threshold encryption, we only need FE for *linear* functions because the share-generating function in vanilla Shamir secret sharing is a linear function in the secret sharing randomness.
- Suppose an adversary obtains Q secret keys associated with the subsets $\Gamma_1, \ldots, \Gamma_Q$. Whenever an FE instance i falls into two or more of these subsets (i.e., any index $i \in \Gamma_j \cap \Gamma_k$ for some $j \neq k$), then multiple keys have been issued for instance i and the security guarantee of the underlying single-key FE scheme can no longer be invoked. However, if an instance i belongs to exactly one subset, then we can invoke single-key security of the i^{th} FE scheme.
- The key observation in [GVW15] is that if the subsets $\Gamma_1, \ldots, \Gamma_Q$ are chosen at random, then one can set the parameters so that with high probability, the number of indices that fall into more than one set is small. As a result, only a small fraction of the underlying single-key FE instances will be "bad" (i.e., the user obtains more than one secret key for the particular instance). Then, by combining the security of the underlying secret sharing scheme with the security of the remaining single-key FE instances, the work of [GVW12] proves simulation security against up to Q colluding key holders.

Adapting [GVW12] to the registered setting. We now adapt the above paradigm to the registered setting. Instead of a central authority that issues the secret keys, we now have M registration buckets, where the i^{th} bucket corresponds to the i^{th} instance of the single-key registered FE scheme, where each bucket is associated with its own (independently-sampled) CRS. The scheme has the following structure:

- **Key generation.** A user first samples a random seed γ and derives a subset $\Gamma \subseteq [M]$ from γ. Then, for each instance $i \in \Gamma$, the user samples a fresh public/secret key-pair for the i^{th} single-key registered FE scheme. Its public key consists of the seed γ together with the public keys for the registered FE instances indexed by Γ.
- **Aggregation.** To aggregate a collection of public keys and their associated functions, the aggregator first recovers the subset Γ associated with each user from its seed γ. Then, for each $i \in [M]$, the aggregator takes the public keys registered in the i^{th} bucket (i.e., the keys of the users whose subset contains i) together with their associated functions, and aggregates them using the i^{th} single-key registered FE scheme to obtain a master public key mpk_i. The aggregated public key is the collection $(\mathsf{mpk}_1, \ldots, \mathsf{mpk}_M)$. Each user's helper decryption key consists of the helper decryptions key associated with the instances indexed by Γ.

[7] The work of [GVW12] then shows how bootstrap an FE scheme for NC^1 into one that supports general polynomial-size circuits by additionally relying on a low-depth PRG and randomized encodings [Yao86, IK00, AIK06].

- **Encryption and decryption.** These proceed exactly as in the centralized setting. The encrypter secret-shares the input $\mathbf{x}$ into M shares and encrypts the i^{th} share with respect to mpk_i, while a user with function f decrypts to learn $f(\mathbf{x}_i)$ for each $i \in \Gamma$ and then interpolates to obtain $f(\mathbf{x})$.

The main challenge is that, unlike in the centralized setting, the subsets Γ are now chosen by the users, who may be malicious. For instance, the corrupted users could choose highly-correlated subsets so as to maximize the overlap $\bigcup_{j \neq k}(\Gamma_j \cap \Gamma_k)$ among their subsets $\Gamma_1, \ldots, \Gamma_Q$. If the overlap is too big, then too many of the single-key instances are "bad" and the security argument from [GVW12] no longer applies.

To fix this problem, we can have the users derive Γ from the seed γ using a random oracle (i.e., set $\Gamma = H(\gamma)$). A malicious user can still query the random oracle on many candidate seeds and register using the most favorable ones. Thus, unlike in the centralized setting, the subsets associated with the corrupted users are no longer independently distributed. However, the analysis of [GVW12] establishes a stronger guarantee: for *any* fixed collection of Q random subsets, the probability that their overlap exceeds the permitted bound is exponentially small. For an adversary that makes at most Q_{ro} random-oracle queries, there are at most $\binom{Q_{\text{ro}}}{Q}$ collections of subsets it can choose to register. By choosing the parameters appropriately, we can still appeal to a union bound to show that with overwhelming probability, *every* collection the adversary could have registered satisfies the required bound. In this case, the combinatorial guarantee underlying the analysis of [GVW12] continues to hold even when the subsets are chosen adversarially.

Cover-freeness and randomization. The description above omits a few technical components that are also present in the compiler from [GVW12]. First, the encrypter not only secret-shares the input $\mathbf{x}$, it also encrypts secret shares of 0 in M' additional slots. Second, a key for a function f is not a key for f itself, but for a derived function $G_{f,\Delta}$, where $\Delta \subseteq [M']$ is a subset of the dummy slots (the precise definition of $G_{f,\Delta}$ is not important for this overview). Since the dummy slots contain shares of 0, they do not affect correctness, and in the security analysis, they provide the simulator with a fresh random value for each key it has to simulate. For this to work, the analysis requires that the subsets $\Delta_1, \ldots, \Delta_Q$ associated with the adversary's keys be cover-free (Definition 2.3).

In the registered setting, the subsets Δ are again chosen by the users, so a malicious user could try to bias its choice so as to violate cover-freeness. This is handled exactly as before: each subset Δ is derived by applying the random oracle to a user-chosen seed. The analysis of [GVW12] shows that any fixed collection of Q random subsets fails to be cover-free with exponentially small probability, so a union bound over the $\binom{Q_{\text{ro}}}{Q}$ collections the adversary can assemble from its random-oracle queries again yields a negligible failure probability. As such, both of the combinatorial properties underlying the analysis of [GVW12] (the small overlap of the subsets $\Gamma_1, \ldots, \Gamma_Q$ and the cover-freeness of the subsets $\Delta_1, \ldots, \Delta_Q$) hold in the registered setting, and the security argument carries over essentially unchanged.

Registered single-key FE from [SS10, ZCHZ24]. It remains to construct a registered *single-key* FE scheme. We first describe the approach from [ZCHZ24] which follows the template of Sahai and Seyalioglu [SS10]. The work of [SS10] combines garbled circuits with public-key encryption to obtain a single-key FE scheme. To build a registered FE scheme, the work of [ZCHZ24] replaces the public-key encryption scheme with a distributed broadcast encryption scheme that supports key aggregation [CW24, CHW25, AMR25, WW25, GY26c, GY26a, AMR26, CW26b]. In this setting, there is a common reference string, and each user independently samples their own public/secret key-pair. Then, given a collection of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ and a set $S \subseteq [N]$, there is a public and deterministic aggregation algorithm that compresses the public keys of the users in S into a short master public key mpk_S of size $\mathsf{poly}(\lambda)$. In particular, the size of mpk_S is independent of the number of users in S. Anyone can encrypt a message with respect to mpk_S, and any user $i \in S$ can decrypt using its own secret key sk_i (together with a helper decryption key output by the aggregation algorithm). Security requires that the ciphertext hide the message from any collection of users outside S, even if those users choose their public keys maliciously.

We now describe the template from [SS10, ZCHZ24]. Suppose the functions we support have descriptions of length ℓ_{cir}, and write $f[i]$ for the i^{th} bit of the description of the function f. The scheme has the following structure:

- **Registration buckets.** For each position $i \in [\ell_{\text{cir}}]$ and each bit $b \in \{0,1\}$, we create a registration bucket. Each of the $2\ell_{\text{cir}}$ buckets is an instance of a distributed broadcast encryption scheme (again, with its own CRS).
- **Key generation.** A user samples a public/secret key-pair for each bucket and publishes the collection $\mathsf{pk} = \{\mathsf{pk}^{i,b}\}_{i \in [\ell_{\text{cir}}], b \in \{0,1\}}$ as its public key.

- **Aggregation.** Suppose users with functions $f_1, \ldots, f_N$ have registered the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, where $\mathsf{pk}_j = \left\{\mathsf{pk}_j^{i,b}\right\}_{i,b}$. For each $i \in [\ell_{\mathsf{cir}}]$ and $b \in \{0,1\}$, the aggregator aggregates the public keys of the users whose function description agrees with b at position i. Namely, it aggregates the keys $\left\{\mathsf{pk}_j^{i,b}\right\}_{j : f_j[i]=b}$ to obtain a master public key $\mathsf{mpk}_{i,b}$ for bucket (i,b). The aggregated public key is the collection $\mathsf{mpk} = \{\mathsf{mpk}_{i,b}\}_{i \in [\ell_{\mathsf{cir}}], b \in \{0,1\}}$. In particular, the key that a user with function f registers only contributes to the ℓ_{cir} buckets indexed by $(i, f[i])$.

- **Encryption.** To encrypt an input $\mathbf{x}$, the encrypter garbles the universal circuit $U_{\mathbf{x}}$ that takes as input the description of a function f and outputs $f(\mathbf{x})$. Let $\{\mathsf{lab}_{i,b}\}_{i \in [\ell_{\mathsf{cir}}], b \in \{0,1\}}$ denote the garbled input labels. For each pair (i,b), the encrypter encrypts the label $\mathsf{lab}_{i,b}$ with respect to $\mathsf{mpk}_{i,b}$.

- **Decryption.** A user with function f can decrypt exactly the buckets its public key contributed to, and thus, recovers the labels $\{\mathsf{lab}_{i,f[i]}\}_{i \in [\ell_{\mathsf{cir}}]}$. These constitute a garbled input encoding of the description of f, so the user can evaluate the garbled circuit to obtain $U_{\mathbf{x}}(f) = f(\mathbf{x})$.

Security follows the same argument as in [SS10, ZCHZ24]. Namely, security of the distributed broadcast encryption scheme ensures that a user does not learn the labels associated with the buckets its public key did not contribute to, while privacy of the garbled circuit ensures that the labels it does learn reveal nothing more than $f(\mathbf{x})$. We note that this scheme is inherently single-key: if two users with different functions $f \neq f'$ both decrypt, then at any position i where the descriptions differ, they respectively recover $\mathsf{lab}_{i,f[i]}$ and $\mathsf{lab}_{i,f'[i]}$, and together, they hold both labels for an input wire of the garbled circuit. To support bounded collusions, we thus need to apply the above variant of [GVW12] to the basic registered single-key FE scheme. We now bound the ciphertext size for this scheme.

- Suppose the functions we support can be computed by Boolean circuits of size s. Then, $\ell_{\mathsf{cir}} = \tilde{O}(s)$. A ciphertext for the single-key scheme consists of a garbling of the universal circuit $U_{\mathbf{x}}$, which has size $\tilde{O}(s) \cdot \mathsf{poly}(\lambda)$, together with $2\ell_{\mathsf{cir}}$ distributed broadcast encryption ciphertexts of size $\mathsf{poly}(\lambda)$ each. Thus, a ciphertext for the single-key scheme already has size $\tilde{O}(s) \cdot \mathsf{poly}(\lambda)$. Since a ciphertext for the bounded-collusion scheme consists of a single-key ciphertext for each of the $M = \mathsf{poly}(Q)$ instances, the ciphertext size of the resulting Q-bounded-collusion registered FE scheme is $\mathsf{poly}(Q) \cdot \tilde{O}(s) \cdot \mathsf{poly}(\lambda)$. For instance, if $M = O(Q^5)$, the resulting ciphertext size is $Q^5 \cdot \tilde{O}(s) \cdot \mathsf{poly}(\lambda)$. In contrast with Corollary 3.33, the ciphertext size here scales with the *size* of the functions rather than just their depth.

- In the context of silent threshold encryption, suppose we instantiate Construction 4.7 with vanilla Shamir secret sharing.[8] Then the sharing randomness consists of the $T-1$ coefficients of the sharing polynomial and $\rho_{\mathsf{SS}} = \tilde{O}(T)$. Taking the collusion bound to be $Q = T - 1$, each user is associated with a share function on inputs of length $1 + \rho_{\mathsf{SS}} = \tilde{O}(T)$ which can be computed by a Boolean circuit of size $s = \tilde{O}(T)$. Substituting into the above, the ciphertext size is $\mathsf{poly}(T, \lambda)$. If $M = O(Q^5)$, this becomes $T^6 \cdot \mathsf{poly}(\lambda)$. In fact, even if we could set $M = \tilde{O}(Q)$, this would still lead to a ciphertext of size $T^2 \cdot \mathsf{poly}(\lambda)$. In contrast, Corollary 4.13 achieves ciphertext size $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$.

Succinct registered single-key FE from [GKP+13]. As in the scheme of [SS10], the ciphertext in the above construction grows with the size of the supported functions. In the application to silent threshold encryption, the function that computes a share of the message has size that scales with the threshold T, and the ciphertext size inherits this dependence. To avoid this, we can instead use a *succinct* registered single-key FE scheme, where the ciphertext size only scales with the *depth* of the supported functions. Such a scheme can be obtained by adapting the succinct FE scheme from Goldwasser et al. [GKP+13], which combines ABE, leveled homomorphic encryption (LHE), and garbled circuits. To obtain the registered variant, we simply replace the ABE scheme with a registered ABE scheme:

- **Registration buckets.** Let ℓ_{lhe} denote the bit length of an LHE ciphertext (encrypting a single bit). As before, we create a bucket for each position $i \in [\ell_{\mathsf{lhe}}]$ and each bit $b \in \{0,1\}$, except now each of the $2\ell_{\mathsf{lhe}}$ buckets is an instance of registered ABE.

[8] We could also use the PRF-based sharing algorithm from Fact 4.4. However, since the ciphertext size for this scheme depends on the size of the functions being computed, and using a low-depth PRF to derive the randomness does not decrease the size of the share-generating function, this optimization does not change the bottom line.

- **Key generation.** For each bucket (i, b), a user registering the function f generates a registered ABE key for the predicate $f'_{i,b}$ that takes as input an LHE ciphertext $\hat{\mathbf{x}}$, homomorphically evaluates f on $\hat{\mathbf{x}}$ to obtain an evaluated ciphertext $\widehat{f(\mathbf{x})}$, and outputs 1 if and only if the i^{th} bit of $\widehat{f(\mathbf{x})}$ is b.
- **Encryption.** To encrypt an input $\mathbf{x}$, the encrypter samples an LHE key, computes an LHE encryption $\hat{\mathbf{x}}$ of $\mathbf{x}$, and garbles the LHE decryption circuit (with the LHE secret key hardwired). Let $\{\mathsf{lab}_{i,b}\}_{i\in[\ell_{\mathsf{lhe}}],b\in\{0,1\}}$ denote the garbled input labels. For each pair (i, b), the encrypter encrypts the label $\mathsf{lab}_{i,b}$ using the registered ABE instance for bucket (i, b) with respect to the attribute $\hat{\mathbf{x}}$. The ciphertext also includes $\hat{\mathbf{x}}$.
- **Decryption.** For each position i, exactly one of the predicates $f'_{i,0}$ and $f'_{i,1}$ is satisfied by the attribute $\hat{\mathbf{x}}$. Thus, a user with function f recovers exactly the labels $\left\{\mathsf{lab}_{i,\widehat{f(\mathbf{x})}[i]}\right\}_{i\in[\ell_{\mathsf{lhe}}]}$, where $\widehat{f(\mathbf{x})}[i]$ denotes the i^{th} bit of $\widehat{f(\mathbf{x})}$. These constitute a garbled input encoding of $\widehat{f(\mathbf{x})}$, so the user can evaluate the garbled LHE decryption circuit to obtain $f(\mathbf{x})$.

Security follows the same argument as in [GKP+13]. Namely, security of the registered ABE scheme ensures that a user only learns the labels associated with the bits of $\widehat{f(\mathbf{x})}$, privacy of the garbled circuit ensures that these labels reveal nothing more than the output of the LHE decryption circuit, and semantic security of LHE ensures that $\hat{\mathbf{x}}$ hides the input $\mathbf{x}$. We note that if the underlying registered ABE scheme is only input-selectively secure, then both the resulting registered single-key FE scheme and the bounded-collusion registered FE scheme we obtain from it are input-selectively secure. We now bound the ciphertext size for this scheme.

- Suppose the functions we support are defined on inputs of length ℓ and can be computed by Boolean circuits of depth d, and suppose the registered ABE scheme has succinct ciphertexts (i.e., of size $\mathsf{poly}(\lambda, d')$, where d' is a bound on the depth of the supported predicates). Since the predicate $f'_{i,b}$ homomorphically evaluates a function of depth d, we can take $d' = \mathsf{poly}(\lambda, d)$. A ciphertext for the single-key FE scheme now consists of an LHE encryption of the input, which has size $\ell \cdot \mathsf{poly}(\lambda, d)$, a garbling of the LHE decryption circuit, which has size $\mathsf{poly}(\lambda, d)$, and $2\ell_{\mathsf{lhe}}$ registered ABE ciphertexts of size $\mathsf{poly}(\lambda, d')$ each. Thus, a ciphertext for the single-key registered FE scheme has size $\ell \cdot \mathsf{poly}(\lambda, d)$, which is *independent* of the size of the supported functions. As before, a ciphertext for the bounded-collusion scheme consists of a single-key ciphertext for each of the $M = \mathsf{poly}(Q)$ instances, so the ciphertext size of the resulting Q-bounded-collusion registered FE scheme is $\mathsf{poly}(Q) \cdot \ell \cdot \mathsf{poly}(\lambda, d)$. For instance, if $M = O(Q^5)$, the resulting ciphertext size is $Q^5 \cdot \ell \cdot \mathsf{poly}(\lambda, d)$. Contrast this with Corollary 3.33, where the ciphertext size is $Q \cdot \tilde{O}(d) + \ell \cdot \mathsf{poly}(\lambda, d)$.
- In the context of silent threshold encryption, suppose we instantiate Construction 4.7 with the PRF-based sharing algorithm from Fact 4.4 (instantiated with a PRF in NC^1). In this case, the sharing randomness is a PRF key of length $\rho_{\mathsf{SS}} = \mathsf{poly}(\lambda)$. Taking the collusion bound to be $Q = T - 1$, each user is now associated with a share function on inputs of length $\ell = 1 + \rho_{\mathsf{SS}} = \mathsf{poly}(\lambda)$ and of depth $d = \tilde{O}(1)$. Substituting into the above, the ciphertext size is $\mathsf{poly}(Q) \cdot \mathsf{poly}(\lambda)$. If $M = O(Q^5)$, this becomes $T^5 \cdot \mathsf{poly}(\lambda)$. Optimistically, suppose we could take $M = \tilde{O}(Q)$; note that we are *not* claiming such an instantiation is feasible for succinct registered FE. Even in this setting, the resulting ciphertext would have size $\tilde{O}(T) \cdot \mathsf{poly}(\lambda)$. Once again, this is worse than the $\tilde{O}(T) + \mathsf{poly}(\lambda, \log N)$ ciphertext size we obtain via Corollary 4.13.