How to Share a Lattice Trapdoor: Threshold Protocols for Signatures and (H)IBE

Rikke Bendlin* Sara Krehbiel† Chris Peikert‡

Abstract

We develop secure *threshold* protocols for two important operations in lattice cryptography, namely, generating a hard lattice Λ together with a "strong" trapdoor, and sampling from a discrete Gaussian distribution over a desired coset of Λ using the trapdoor. These are the central operations of many cryptographic schemes: for example, they are exactly the key-generation and signing operations (respectively) for the GPV signature scheme, and they are the public parameter generation and private key extraction operations (respectively) for the GPV IBE. We also provide a protocol for trapdoor delegation, which is used in lattice-based hierarchical IBE schemes. Our work therefore directly transfers all these systems to the threshold setting.

Our protocols provide information-theoretic (i.e., statistical) security against adaptive corruptions in the UC framework, and they are private and robust against an optimal number of semi-honest or malicious parties. Our Gaussian sampling protocol is both noninteractive and efficient, assuming either a trusted setup phase (e.g., performed as part of key generation) or a sufficient amount of interactive but offline precomputation, which can be performed before the inputs to the sampling phase are known.

*Department of Computer Science, Aarhus University. Email: rikkeb@cs.au.dk. Supported by the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council). Part of this work was performed while visiting the Georgia Institute of Technology.

†School of Computer Science, College of Computing, Georgia Institute of Technology. Email: sarak@gatech.edu

‡School of Computer Science, College of Computing, Georgia Institute of Technology. Email: cpeikert@cc.gatech.edu. Supported by the Alfred P. Sloan Foundation and the National Science Foundation under CAREER Award CCF-1054495. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

1 Introduction

A *threshold* cryptographic scheme [DF89] is one that allows any quorum of h out of ℓ trustees to jointly perform some privileged operation(s) by following a specified protocol, and remains correct and secure even if up to some $t < h$ of the parties deviate from the protocol adversarially. For example, in a threshold signature scheme any h trustees can sign an agreed-upon message, and no t malicious players (who may even pool their knowledge and coordinate their actions) can prevent the signature from being produced, nor forge a valid signature on a new message. Similarly, a threshold encryption scheme requires at least h trustees to decrypt a ciphertext. Threshold cryptography is very useful for both distributing trust and increasing robustness in systems that perform high-value operations, such as certificate authorities (CAs) or private-key generators in identity-based encryption (IBE) systems.

Desirable efficiency properties in a threshold system include: (1) efficient local computation by the trustees; (2) low interaction—e.g., one broadcast message from each party—when performing the privileged operations; and (3) key sizes and public operations that are independent of the number of trustees. For example, while it might require several parties to *sign* a message, it is best if the signature can be *verified* without even being aware that it was produced in a distributed manner.

Over the years many elegant and rather efficient threshold systems have been developed. To name just a few representative works, there are simple variants of the ElGamal cryptosystem, Canetti and Goldwasser's [CG99] version of the CCA-secure Cramer-Shoup cryptosystem [CS98], and Shoup's [Sho00] version of the RSA signature scheme. These systems, along with almost all others in the literature, are based on number-theoretic problems related to either integer factorization or the discrete logarithm problem in cyclic groups. As is now well-known, Shor's algorithm [Sho97] would unfortunately render all these schemes insecure in a "post-quantum" world with large-scale quantum computers.

Lattice-based cryptography. Recently, *lattices* have been recognized as a viable foundation for quantum-resistant cryptography, and the past few years have seen the rapid growth of many rich lattice-based systems. A fruitful line of research, starting from the work of Gentry, Peikert and Vaikuntanathan (GPV) [GPV08], has resulted in secure lattice-based hash-and-sign signatures and (hierarchical) identity-based encryption schemes [CHKP10, ABB10], along with many more applications (e.g., [GKV10, BF11b, BF11a, AFV11]). All these schemes rely at heart on two nontrivial algorithms: the key-generation algorithm produces a lattice Λ together with a certain kind of "strong" trapdoor (e.g., a short basis of Λ) [Ajt99, AP09], while the signing/key-extraction algorithms use the trapdoor to randomly sample a short vector from a *discrete Gaussian distribution* over a certain coset $\Lambda + \mathbf{c}$, which is determined by the message or identity [GPV08]. Initially, both tasks were rather complicated algorithmically, and in particular the Gaussian sampling algorithm involved several adaptive iterations, so it was unclear whether either task could be efficiently and securely distributed among several parties. Recently, however, both key generation and Gaussian sampling have been simplified and made more efficient and parallel [Pei10, MP12]. This is the starting point for our work.

Our results. We give threshold protocols for the main nontrivial operations in lattice-based signature and (H)IBE schemes, namely: (1) generating a lattice Λ together with a strong trapdoor of the kind recently proposed in [MP12], (2) sampling from a discrete Gaussian distribution over a desired coset of Λ, and (3) delegating a trapdoor for a higher-dimensional extension of Λ. Since these are the only secret-key operations used in the signature and (H)IBE schemes of [GPV08, CHKP10, ABB10, MP12] and several other related works, our protocols can be plugged directly into all those schemes to distribute the signing algorithms and the (H)IBE private-key generators. In Section 4 we show how this is (straightforwardly) done for the simplest of these applications, namely, the GPV signature scheme [GPV08]; the GPV IBE scheme and other applications work similarly.

Our protocols have several desirable properties:

- They provide *information-theoretic* (i.e., statistical) security for *adaptive* corruptions. By information-theoretic security, we mean that the security of the key-generation and sampling protocols *themselves* relies on no computational assumption—instead, the application alone determines the assumption (usually, the Short Integer Solution assumption [Ajt96, MR04] for digital signatures, and Learning With Errors [Reg05] for identity-based encryption). We work in a version of the universal composability (UC) framework [Can01], specialized to the threshold setting, and as a result also get strong security guarantees for protocols under arbitrary composition.
- They are secure for an optimal threshold of semi-honest or active (malicious) parties, which is determined by the precise communication model and setup assumption. For example, we can tolerate $h - 1$ semi-honest parties assuming trusted setup (see below), or $t = h - 1$ malicious parties in a model with both broadcast and private channels, using the verifiable secret sharing scheme of [RB89]. (Recall that h is the number of (semi-)honest parties the protocol requires to execute successfully, and the robustness threshold t is an upper bound on the number of malicious parties.)
- The public key and trapdoor "quality" (i.e., the width of the discrete Gaussian that can be sampled using the trapdoor; smaller width means higher quality) are essentially the same as in the standalone setting. In particular, their sizes are independent of the number of trustees; the individual shares of the trapdoor are the same size as the trapdoor itself; and the protocols work for the same lattice parameters as in the standalone setting, up to small constant factors.
- They have *noninteractive* and very efficient *online* phases (corresponding to the signing or key-extraction operations), assuming either (1) a setup phase in which certain shares are distributed by a trusted party (e.g., as part of key generation), or (2) the parties themselves perform a sufficient amount of interactive precomputation in an offline phase (without relying on any trusted party). We provide protocols for these two settings in Section 3 and Appendix A, respectively.

Regarding the final item, the trusted setup model is the one used by Canetti and Goldwasser [CG99] for constructing threshold chosen ciphertext-secure threshold cryptosystems: as part of the key-generation process, a trusted party also distributes shares of some appropriately distributed secrets to the parties, which they can later use to perform an *a priori* bounded number of noninteractive threshold operations. Or, in lieu of a trusted party, the players can perform some interactive precomputation (offline, before the desired coset is known) to generate the needed randomness. The downside is that this precomputation is somewhat expensive, since the only solution we have for one important step (namely, sampling shares of a Gaussian-distributed value over $\mathbb{Z}$) is to use somewhat generic information-theoretic multiparty computation tools. On the plus side, the circuit for this sampling task is rather shallow, with depth just slightly super-constant $\omega(1)$, so the round complexity of the precomputation is not very high. We emphasize that the expensive precomputation is executed offline, before the applications decides which lattice cosets will be sampled from, and that the online protocols remain efficient and non-interactive.

Our protocols rely on the very simple form of the new type of strong trapdoor recently proposed in [MP12], and the parallel and offline nature of recent standalone Gaussian sampling algorithms [Pei10, MP12].[1] A key technical challenge is that the security of the sampling algorithms from [Pei10, MP12] crucially relies on the secrecy of some intermediate random variables known as "perturbations." However, in order to obtain a noninteractive protocol we need the parties to publicly reveal certain information about these perturbations. Fortunately, we can show that the leaked information is indeed simulatable, and so security is unharmed. See Section 3 and in particular Lemma 3.2 for further details.

[1] In particular, it appears very difficult to implement, in a noninteractive threshold fashion, iterative sampling algorithms like those from [Kle00, GPV08] which use the classical trapdoor notion of a short basis.

Open problems. In addition to simple, non-interactive protocols for discrete Gaussian sampling with trusted setup, in Appendix A we give efficient protocols for discrete Gaussian sampling that avoid both trusted setup and online interaction by using (offline) access to a functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$, which produces shares of Gaussian-distributed values over the integers $\mathbb{Z}$ (see Appendix A.1 for details). We show how to instantiate $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ using a (somewhat inefficient) interactive protocol using generic MPC techniques. It remains an interesting open problem to design discrete Gaussian sampling protocols without trusted setup whose *offline precomputation* is efficient and/or non-interactive as well. An efficient realization of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ would yield such a solution, but there may be other routes as well.

Another intriguing problem is to give a simple and noninteractive threshold protocol for inverting the LWE function $g_{\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{s}^t\mathbf{A} + \mathbf{e}^t \bmod q$ (for short error vector $\mathbf{e}$) using a shared trapdoor. We find it surprising that, while in the standalone setting this inversion task is conceptually and algorithmically much simpler than Gaussian sampling, we have not yet been able to find a simple threshold protocol for it.[2] Such a protocol could, for example, be useful for obtaining threshold analogues of the chosen ciphertext-secure cryptosystems from [Pei09, MP12], without going through a generic IBE-to-CCA transformation [BCHK07].

Related work in threshold lattice cryptography. A few works have considered lattice cryptography in the threshold setting. For encryption schemes, Bendlin and Damgård [BD10] gave a threshold version of Regev's CPA-secure encryption scheme based on the learning with errors (LWE) problem [Reg05]. Related work by Myers *et al.* [MSs11] described threshold decryption for fully homomorphic cryptosystems. Xie *et al.* [XXZ11] gave a threshold CCA-secure encryption scheme from any lossy trapdoor function (and hence from lattices/LWE [PW08]), though its public key and encryption runtime grow at least linearly with the number of trustees. For signatures, Feng *et al.* [FGM10] gave a threshold signature scheme where signing proceeds sequentially through each trustee, making the scheme highly interactive; also, the scheme is based on NTRUSign, which has been broken [NR06]. Cayrel *et al.* [CLRS10] gave a lattice-based threshold *ring* signature scheme, in which at least t trustees are needed to create an *anonymous* signature. In that system, each trustee has its own public key, and verification time grows linearly with the number of trustees. In summary, lattice-based threshold schemes to date have either been concerned with distributing the *decryption* operation in public-key cryptosystems, and/or have lacked key efficiency properties typically asked of threshold systems (which our protocols do enjoy). Also, other important applications such as (H)IBE have yet to be realized in a threshold manner.

Organization. The remainder of the paper is organized as follows. In Section 2 we overview the relevant background on lattices, secret sharing, and the UC framework. In Section 3 we review the standalone key-generation and discrete Gaussian sampling algorithms of [MP12], present our functionalities for these algorithms in the threshold setting, and show how these functionalities can be implemented efficiently and noninteractively using trusted setup. At the end of Section 3 we additionally provide a functionality and protocol for trapdoor delegation. Finally, in Section 4 we detail a simple example application of our protocols, namely, a threshold version of the GPV signature scheme [GPV08] realizing the threshold signature functionality of [ADN06]. In the appendix, we remove the trusted setup assumption and show how to instead use offline interaction to implement all our functionalities.

2 Preliminaries

We denote the reals by $\mathbb{R}$ and the integers by $\mathbb{Z}$. For a positive integer ℓ, we let $[\ell] = \{1, \ldots, \ell\}$.

[2] We note that it is possible to give a threshold protocol using a combination of Gaussian sampling and trapdoor delegation [CHKP10, MP12], but it is obviously no simpler than Gaussian sampling alone.

A square symmetric real matrix Σ is *positive definite*, written $\Sigma > \mathbf{0}$, if $\mathbf{x}^t \Sigma \mathbf{x} > 0$ for all nonzero $\mathbf{x}$. Positive definiteness defines a partial ordering on symmetric matrices: we say that $\Sigma_1 > \Sigma_2$ if $(\Sigma_1 - \Sigma_2) > \mathbf{0}$. For any nonsingular matrix $\mathbf{B} \in \mathbb{R}^{n \times n}$, the symmetric matrix $\Sigma = \mathbf{B}\mathbf{B}^t$ is positive definite. We say that $\mathbf{B}$ is a *square root* of $\Sigma > \mathbf{0}$, written $\mathbf{B} = \sqrt{\Sigma}$, if $\mathbf{B}\mathbf{B}^t = \Sigma$. Every $\Sigma > \mathbf{0}$ has a square root; moreover, the square root is unique up to right-multiplication by an orthogonal matrix, i.e., $\mathbf{B}' = \sqrt{\Sigma}$ if and only if $\mathbf{B}' = \mathbf{B}\mathbf{Q}$ for some orthogonal matrix $\mathbf{Q}$. A square root can be computed efficiently using, e.g., the Cholesky decomposition. The largest singular value (also called spectral norm or operator norm) of a real matrix $\mathbf{X}$ is defined as $s_1(\mathbf{X}) = \max_{\mathbf{u} \neq \mathbf{0}} \|\mathbf{X}\mathbf{u}\| / \|\mathbf{u}\|$. For convenience, we sometime write a scalar s to mean the scaled identity matrix $s\mathbf{I}$, whose dimension will be clear from context.

2.1 Continuous Gaussians

The n-dimensional Gaussian function $\rho\colon \mathbb{R}^n \to (0, 1]$ is defined as

$$\rho(\mathbf{x}) \stackrel{\Delta}{=} \exp(-\pi \cdot \|\mathbf{x}\|^2) = \exp(-\pi \cdot \langle \mathbf{x}, \mathbf{x} \rangle).$$

Applying a linear transformation given by a nonsingular real matrix $\mathbf{B}$ yields the Gaussian function

$$\rho_{\mathbf{B}}(\mathbf{x}) := \rho(\mathbf{B}^{-1}\mathbf{x}) = \exp\left(-\pi \cdot \left\langle \mathbf{B}^{-1}\mathbf{x}, \mathbf{B}^{-1}\mathbf{x} \right\rangle\right) = \exp\left(-\pi \cdot \mathbf{x}^t \Sigma^{-1} \mathbf{x}\right),$$

where $\Sigma = \mathbf{B}\mathbf{B}^t > \mathbf{0}$. Because $\rho_{\mathbf{B}}$ is distinguished only up to Σ, we usually refer to it as $\rho_{\sqrt{\Sigma}}$.

Normalizing $\rho_{\sqrt{\Sigma}}$ by its total measure $\int_{\mathbb{R}^n} \rho_{\sqrt{\Sigma}}(\mathbf{x})\, d\mathbf{x} = \sqrt{\det \Sigma}$ over $\mathbb{R}^n$, we obtain the probability distribution function of the (continuous) *Gaussian distribution* $D_{\sqrt{\Sigma}}$. It is easy to check that a random variable $\mathbf{x}$ having distribution $D_{\sqrt{\Sigma}}$ can be written as $\sqrt{\Sigma} \cdot \mathbf{z}$, where $\mathbf{z}$ has spherical Gaussian distribution D_1. Therefore, the random variable $\mathbf{x}$ has *covariance*

$$\mathop{\mathrm{E}}_{\mathbf{x} \sim D_{\sqrt{\Sigma}}} \left[\mathbf{x} \cdot \mathbf{x}^t\right] = \sqrt{\Sigma} \cdot \mathop{\mathrm{E}}_{\mathbf{z} \sim D_1} \left[\mathbf{z} \cdot \mathbf{z}^t\right] \cdot \sqrt{\Sigma}^t = \sqrt{\Sigma} \cdot \frac{\mathbf{I}}{2\pi} \cdot \sqrt{\Sigma}^t = \frac{\Sigma}{2\pi},$$

by linearity of expectation. (The $\mathbf{I}/(2\pi)$ covariance of $\mathbf{z} \sim D_1$ arises from the independence of its entries, which are each distributed as D_1 in one dimension, and therefore have variance $1/(2\pi)$.) For convenience, in this paper we implicitly scale all covariance matrices by a 2π factor, and refer to Σ as the covariance matrix of $D_{\sqrt{\Sigma}}$.

2.2 Lattices and Discrete Gaussians

A *lattice* Λ is a discrete additive subgroup of $\mathbb{R}^m$ for some $m \geq 0$. In this work we are only concerned with full-rank integer lattices, which are additive subgroups of $\mathbb{Z}^m$ with finite index. Most recent cryptographic applications use a particular family of so-called *q-ary* integer lattices, which contain $q\mathbb{Z}^m$ as a sublattice for some integer q, which in this work will always be bounded by $\mathrm{poly}(n)$. For positive integers n and q, let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be arbitrary, and define the full-rank m-dimensional q-ary lattice

$$\Lambda^{\perp}(\mathbf{A}) = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \bmod q\}.$$

For any $\mathbf{u} \in \mathbb{Z}_q^n$ admitting an integral solution $\mathbf{x} \in \mathbb{Z}^m$ to $\mathbf{A}\mathbf{x} = \mathbf{u} \bmod q$, define the coset (or shifted lattice)

$$\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}) = \Lambda^{\perp}(\mathbf{A}) + \mathbf{x} = \{\mathbf{z} \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{u} \bmod q\}.$$

Note that for $n, m, q \leq 2$ and $m > Cn \log q$ for some fixed constant $C > 1$, the columns of a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ generate all of $\mathbb{Z}_q^n$ with all but $\mathrm{negl}(n)$ probability.

Let $\Lambda \subset \mathbb{R}^m$ be a lattice, let $\mathbf{c} \in \mathbb{R}^m$, and let $\Sigma > \mathbf{0}$ be a positive definite matrix. The *discrete Gaussian distribution* $D_{\Lambda+\mathbf{c},\sqrt{\Sigma}}$ is simply the Gaussian distribution $D_{\sqrt{\Sigma}}$ restricted so that its support is the coset $\Lambda + \mathbf{c}$. That is, for all $\mathbf{x} \in \Lambda + \mathbf{c}$,

$$D_{\Lambda+\mathbf{c},\sqrt{\Sigma}}(\mathbf{x}) = \frac{\rho_{\sqrt{\Sigma}}(\mathbf{x})}{\rho_{\sqrt{\Sigma}}(\Lambda + \mathbf{c})} \propto \rho_{\sqrt{\Sigma}}(\mathbf{x}).$$

A discrete Gaussian is said to be *spherical* with parameter $s > 0$ if its covariance matrix is $s^2\mathbf{I}$.

We recall an important definition and some useful properties of discrete Gaussian distributions on lattices. For $\epsilon > 0$, the *smoothing parameter* [MR04] $\eta_\epsilon(\Lambda)$ of a lattice Λ is defined as the smallest $s > 0$ such that $\rho_{1/s}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \epsilon$, where Λ^* is the dual lattice (whose precise definition we will not need here). Here we generalize the smoothing parameter to non-spherical Gaussians; note that this definition is consistent with the partial ordering on positive definite matrices.

Definition 2.1 (Smoothing parameter). *Let $\Sigma > \mathbf{0}$ be any positive definite matrix. We say that $\sqrt{\Sigma} \geq \eta_\epsilon(\Lambda)$ if $\eta_\epsilon(\sqrt{\Sigma}^{-1} \cdot \Lambda) \leq 1$, or equivalently, if $\rho_{\sqrt{\Sigma^{-1}}}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \epsilon$.*

The following lemma is a slight generalization of [Reg05, Claim 3.8] (see also [MR04, Lemma 4.1]) to non-spherical Gaussians, obtained by applying a linear transformation to the Gaussian function and lattice. Informally, it says that every coset of Λ has essentially the same mass under $\rho_{\sqrt{\Sigma}}$, when $\sqrt{\Sigma}$ exceeds the smoothing parameter of Λ. The corollary then follows by a routine calculation.

Lemma 2.2. *For any m-dimensional lattice Λ, real $\epsilon > 0$, $r \geq \eta_\epsilon(\Lambda)$, and $\mathbf{c} \in \mathbb{R}^m$, we have $\rho_{\sqrt{\Sigma}}(\Lambda + \mathbf{c}) \in [1 \pm \epsilon] \cdot Z_{\Lambda,r}$, where $Z_{\Lambda,r}$ depends only on Λ and r (not $\mathbf{c}$).*

Corollary 2.3. *Let $\Lambda' \subseteq \Lambda$ be full-rank lattices, and let $\sqrt{\Sigma} \geq \eta_\epsilon(\Lambda')$ for some $\epsilon > 0$. For $\mathbf{x} \leftarrow D_{\Lambda,\sqrt{\Sigma}}$, the marginal distribution of $\mathbf{c} = \mathbf{x} \bmod \Lambda'$ is within statistical distance $\epsilon/2$ from uniform over Λ/Λ', and the conditional distribution of $\mathbf{x}$ given $\mathbf{c}$ is $D_{\Lambda'+\mathbf{c},\sqrt{\Sigma}}$.*

The following special case of [MP12, Lemma 2.4] says that for uniformly random $\mathbf{A}$ and appropriate parameters, the lattice $\Lambda^\perp(\mathbf{A})$ has small smoothing parameter with very high probability.

Lemma 2.4. *Let $n, m, q \geq 2$ be positive integers and $C > 1$ be a fixed constant such that $m > Cn \log q$, and let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be uniformly random. For any fixed $\omega_n = \omega(\sqrt{\log n})$ there exists some $\epsilon = \mathrm{negl}(n)$ such that $\eta_\epsilon(\Lambda^\perp(\mathbf{A})) \leq \omega_n$ except with probability $2^{-\Omega(n)}$.*

Finally, we need the "convolution lemma" of [Pei10, Theorem 3.1].

Lemma 2.5. *Let $\Sigma_1, \Sigma_2 > \mathbf{0}$ be positive definite matrices, with $\Sigma = \Sigma_1 + \Sigma_2 > \mathbf{0}$ and $\Sigma^{-1} = \Sigma_1^{-1} + \Sigma_2^{-1} > \mathbf{0}$. Let Λ_1, Λ_2 be lattices such that $\sqrt{\Sigma_1} \geq \eta_\epsilon(\Lambda_1)$ and $\sqrt{\Sigma_2} \geq \eta_\epsilon(\Lambda_2)$ for some positive $\epsilon \leq 1/2$, and let $\mathbf{c}_1, \mathbf{c}_2 \in \mathbb{R}^m$ be arbitrary. In the following experiment:*

$$\textit{choose } \mathbf{x}_2 \leftarrow D_{\Lambda_2+\mathbf{c}_2,\sqrt{\Sigma_2}}\textit{, then choose } \mathbf{x}_1 \leftarrow \mathbf{x}_2 + D_{\Lambda_1+\mathbf{c}_1-\mathbf{x}_2,\sqrt{\Sigma_1}},$$

the marginal distribution of $\mathbf{x}_1$ is within statistical distance 8ϵ of $D_{\Lambda_1+\mathbf{c}_1,\sqrt{\Sigma}}$.

Throughout the paper we often attach a factor $\omega_n = \omega_n(n) = \omega(\sqrt{\log n})$, which represents an arbitrary fixed function that grows asymptotically faster than $\sqrt{\log n}$, to Gaussian parameters $\sqrt{\Sigma}$ (or ω_n^2 to covariance matrices Σ). In exposition we usually omit reference to these factors, but we always retain them where needed in formal expressions.

2.3 Secret Sharing for Additive Groups

In this work we will need to distribute secret lattice points among multiple players, so that any sufficiently large number of players is able to reconstruct the points, but smaller subsets collectively get no information about the secret. Because a lattice Λ is an *infinite* additive group (and in particular is not a field), it is not immediately amenable to standard secret-sharing techniques like those of [Sha79]. Fortunately, for our purposes it will suffice to share elements of a suitable *finite* quotient group $G = \Lambda/\bar{\Lambda}$, where the sublattice $\bar{\Lambda} \subseteq \Lambda$ is "sparse" enough in Λ that an element of G identifies an element of Λ with enough specificity for our applications. There is a rich theory of secret sharing for arbitrary additive groups and modules, e.g., [DF94, Feh98]. Here we recall the relevant material in enough generality for our purposes.

Let G be a finite abelian (additive) group with identity element 0. The *exponent* of G, denoted $e(G)$, is the smallest positive integer m such that $mg = g + g + \cdots + g = 0$ for every $g \in G$. We have that G is a module over the ring $R = \mathbb{Z}_{e(g)}$, which gives the following form of Shamir's $(t+1)$-out-of-ℓ secret-sharing scheme [Sha79]. Let $t < \ell$ be positive integers, where t denotes a bound on the number of corrupt players out of ℓ total. Suppose that G is an R-module for some ring R which has efficiently computable operations and $\ell + 1$ known elements $U = \{r_0 = 0, r_1, \ldots, r_\ell\} \subseteq R$ such that $r_i - r_j$ is invertible in R (i.e., a unit) for every $i \neq j$. For example, in our protocols we will have $e(G) = q^d$ for some public integers $q \geq 2$ and $d \geq 1$, so we can take $R = \mathbb{Z}_{q^d}$ and $r_i = i \bmod q^d$, as long as ℓ is smaller than every prime divisor of q. When this condition does not hold, we can use an extension ring instead, as described below.

To share a value $g \in G$, one chooses a formal polynomial $f(X) = \sum_{j=0}^{t} f_j X^j \in G[X]$ of degree at most t, where $f_0 = g$ and the $f_i \in G$ for $i \geq 1$ are uniformly random and independent. Player $i \in [\ell]$ is publicly associated with the value $r_i \in R$, and gets the share $s_i = f(r_i) = \sum_{j=0}^{t-1} r_i^j f_j \in G$. Usually we let f be implicit, denoting the ith player's share as $[\![g]\!]^i$ and the tuple of all shares by $[\![g]\!]$. Note that the product group G^k is also an additive group with exponent $e(G)$, so we can share vectors or matrices with entries in G as above, using the same ring $\mathbb{Z}_{e(G)}$. (Equivalently, this is just an independent entry-wise sharing.)

The above scheme has several important properties (whose proofs are straightforward; see, e.g., [Feh98]):

- It is *ideal*: the shares $s_i \in G$ belong to the same set as the shared value $g \in G$.
- It is *perfectly secret*: for any shared value $g \in G$, any tuple of up to t shares s_i is distributed uniformly.
- It is *perfectly correct* and *robust*: any $t+1$ shares s_i of g (along with their corresponding evaluation points r_i) can be used to efficiently recover $f(X)$, and hence $g = f_0 = f(0)$, by interpolation.

  Moreover, given at least $3t+1$ values s'_i (along with the corresponding evaluation points r_i), where at least $2t+1$ are correct shares $s'_i = f(r_i)$ of g and the remaining t may be arbitrary, one can efficiently recover $f(X)$ and hence $g = f_0$ using, e.g., the Welch-Berlekamp algorithm for unambiguous decoding of Reed-Solomon codes. (The algorithm is usually described for codes defined over finite fields, but its proof of correctness goes through without modification in our setting.)
- It is *homomorphic*: if $g, g' \in G$ have respective shares $s_i = f(r_i), s'_i = f'(r_i)$ for $i \in [\ell]$, then $s_i + s'_i = (f + f')(r_i)$ and $rs_i = (rf)(r_i)$ are respective shares of $g + g'$ and rg for any $r \in R$. Moreover, let $G' \subseteq G$ be a subgroup; then $\bar{s}_i = s_i \bmod G'$ are shares of $\bar{g} = g \bmod G'$, via the polynomial $\bar{f}(X) = f(X) \bmod G'[X]$. Additionally, if $g \in G'$, then $s_i - \bar{s}_i \in G'$ are shares of g.

Secret sharing with extension rings. The above scheme works when the number of parties ℓ is less than every prime divisor of $e(G)$. When this is not the case (e.g., when using $q = 2^k$, which is a convenient choice for the trapdoor construction described in Section 3.1), we can instead share elements from the vector group G^k, which is a module over a certain *extension ring* of $\mathbb{Z}_{e(G)}$ that has a suitable set U of size p^k, where p is the smallest prime divisor of $e(G)$. By choosing $k \geq \log_p(\ell + 1)$, we can share elements of G among ℓ players using shares in G^k, or even amortize the sharing of up to k elements in G at a time.

In brief, we use the extension ring $R = \mathbb{Z}_{e(G)}[X]/F(X)$ for any monic degree-k polynomial $F(X) = \sum_{i=0}^{k} F_i X^i \in \mathbb{Z}_{e(G)}[X]$ that is irreducible modulo every prime dividing $e(G)$. Then it can be verified that G^k is an R-module, where multiplication $R \times G^k \to G^k$ is defined by the rule $X \cdot (g_0, \ldots, g_{k-1}) = (0, g_0, \ldots, g_{k-2}) - (F_0 \cdot g_{k-1}, \ldots, F_{k-1} \cdot g_{k-1})$. An element of R is a unit if and only if it is nonzero modulo every prime integer divisor of $e(G)$, so letting p be the smallest such divisor, the polynomial residues in R with coefficients in $\{0, \ldots, p-1\}$ give us p^k elements $r_i \in R$ such that $r_i - r_j$ is a unit for all $i \neq j$, as needed. See, for example, [DF94] or [Feh98, Chapter 3] for full details.

Verifiable secret sharing. To recover a shared value, our protocols instruct honest parties to broadcast their respective shares and then reconstruct the value from the announced shares. As mentioned above, the Welch-Berlekamp algorithm efficiently reconstructs the shared value given at least $2t + 1$ correct shares and up to t incorrect ones (which may come from malicious parties), which means we can tolerate any $t < \ell/3$ malicious parties. Assuming appropriate communication channels, it is possible to improve this threshold to any $t < \ell/2$ malicious parties by using a verifiable secret sharing (VSS) protocol, e.g., the one of [RB89]. The share-distribution and reconstruction steps of our protocols can be straightforwardly modified to use VSS, but we omit these modifications for simplicity of exposition.

2.4 UC Framework

We frame our results in the Universal Composability (UC) framework [Can00, Can01]. In the UC framework, security is defined by considering a probabilistic polynomial-time (PPT) machine $\mathcal{Z}$, called the environment. In coordination with an adversary that may corrupt some of the players, $\mathcal{Z}$ chooses inputs and observes the outputs of a protocol executed in one of two worlds: a "real" world in which the parties interact with each other in some specified protocol π while a dummy adversary $\mathcal{A}$ (controlled by $\mathcal{Z}$) corrupts players and controls their interactions with honest players, and an "ideal" world in which the players interact directly with a *functionality* $\mathcal{F}$, while a simulator $\mathcal{S}$ (communicating with $\mathcal{Z}$) corrupts players and controls their interactions with $\mathcal{F}$. The views of the environment in these executions are respectively denoted $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$, and the protocol is said to realize the functionality if these two views are indistinguishable. In this work we are concerned solely with statistical indistinguishability (which is stronger than the computational analogue), denoted by the relation $\overset{s}{\approx}$.

Definition 2.6. *A protocol* π statistically realizes *a functionality* $\mathcal{F}$ *(or alternatively, is a UC-secure implementation of* $\mathcal{F}$*) if for any probabilistic polynomial-time (PPT) adversary* $\mathcal{A}$*, there exists a PPT simulator* $\mathcal{S}$ *such that for all PPT environments* $\mathcal{Z}$*, we have* $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{s}{\approx} \mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$.

The *universal composition theorem* [Can01] informally states that any UC-secure protocol remains secure under concurrent general composition. This allows for the modular design of functionalities and protocols which can be composed to produce secure higher-level protocols. Our functionalities implicitly use standard conventions like delayed public and private outputs, corruptions, etc, which are addressed in detail in [Can00, Can01].

UC framework for threshold protocols. We consider a specialized case of the UC framework that is appropriate for modeling threshold protocols. All of our functionalities are called with a session ID of the form $sid = (\mathcal{P}, sid')$, where $\mathcal{P}$ is a set of ℓ parties representing the individual trustees in the threshold protocol. We prove security against adversaries that may adaptively corrupt a certain bounded number of the parties over the entire lifetime of a protocol, and consider both the semi-honest case (in which corrupted parties still execute the protocol faithfully) and the malicious case. At the time of corruption, the entire

view of the player to that point (and beyond) is revealed to the adversary; in particular, we do not assume secure erasures. For robustness, we additionally require that when the environment issues a command to a functionality/protocol, it always does so for at least h honest parties in the same round.

Many of our protocols require the parties to maintain and use consistent local states, corresponding to certain shared random variables that are consumed by the protocols. We note that synchronizing their local states may be nontrivial, if not every party is involved with executing every command. For this reason we assume some mechanism for coordinating local state, such as hashing as suggested in [CG99], which deals with similar synchronization issues.

3 Threshold Key Generation, Gaussian Sampling, and Trapdoor Delegation

In this section, we present UC functionalities and protocols for generating a lattice with a shared trapdoor, for sampling from a coset of that lattice, and for securely delegating a trapdoor of a higher-dimensional extension of the lattice. As an example application of these functionalities, in Section 4 we describe threshold variants of the GPV signature and IBE schemes of [GPV08]; other signature and (H)IBE schemes (e.g., [CHKP10, ABB10, MP12]) can be adapted similarly (where delegation is needed for HIBE).

We start in Section 3.1 by recalling the recent standalone (non-threshold) key generation and discrete Gaussian sampling algorithms of [MP12], which form the basis of our protocols. In Section 3.2 we present the two main functionalities $\mathcal{F}_{\text{KG}}$ (key generation) and $\mathcal{F}_{\text{GS}}$ (Gaussian sampling) corresponding to the standalone algorithms.

- Since key generation tends to be rare in applications, $\mathcal{F}_{\text{KG}}$ can be realized using trusted setup; alternatively, in Appendix A we realize $\mathcal{F}_{\text{KG}}$ via an interactive protocol without trusted setup.
- To realize $\mathcal{F}_{\text{GS}}$, we define two lower-level "helper" functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$, and give in Section 3.3 an efficient noninteractive protocol that realizes $\mathcal{F}_{\text{GS}}$ using access to them. Section 3.2 realizes the helper functionalities noninteractively using trusted setup, and Appendix A realizes them using offline precomputation (instead of trusted setup).
- Finally, in Section 3.4 we give a functionality $\mathcal{F}_{\text{DelTrap}}$ and protocol for trapdoor delegation.

3.1 Trapdoors and Standalone Algorithms

We recall the notion of a (strong) lattice trapdoor and associated algorithms recently introduced by Micciancio and Peikert [MP12] (see that paper for full details and proofs). Let n and q be positive integers and $k = \lceil \lg q \rceil$. Define the "gadget" vector $\mathbf{g} = (1, 2, 4, \ldots, 2^{k-1}) \in \mathbb{Z}_q^k$ and matrix

$$\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^t = \begin{bmatrix} \cdots \mathbf{g}^t \cdots & & & \\ & \cdots \mathbf{g}^t \cdots & & \\ & & \ddots & \\ & & & \cdots \mathbf{g}^t \cdots \end{bmatrix} \in \mathbb{Z}_q^{n \times nk}.$$

The k-dimensional lattice $\Lambda^\perp(\mathbf{g}^t) \subset \mathbb{Z}^k$, and hence also the nk-dimensional lattice $\Lambda^\perp(\mathbf{G})$, has smoothing parameter bounded by $s_{\mathbf{g}} \cdot \omega_n$, where $s_{\mathbf{g}} \leq \sqrt{5}$ is a known constant. There are efficient algorithms that, given any desired syndrome $u \in \mathbb{Z}_q$, sample from a discrete Gaussian distribution over the coset $\Lambda_u^\perp(\mathbf{g}^t)$ for any given parameter $s \geq s_{\mathbf{g}} \cdot \omega_n$. Since $\Lambda^\perp(\mathbf{G}) \subset \mathbb{Z}^{nk}$ is the direct sum of n copies of $\Lambda^\perp(\mathbf{g}^t)$, discrete Gaussian sampling over a desired coset $\Lambda_{\mathbf{u}}^\perp(\mathbf{G})$ (with parameter $s \geq s_{\mathbf{g}} \cdot \omega_n$) can be accomplished by concatenating n independent samples over appropriate cosets of $\Lambda^\perp(\mathbf{g}^t)$.

Definition 3.1 ([MP12]). *Let $m \geq nk$ be an integer and define $\bar{m} = m - nk$. For $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we say that $\mathbf{R} \in \mathbb{Z}_q^{\bar{m} \times nk}$ is a* trapdoor *for $\mathbf{A}$ with* tag $\mathbf{H}^* \in \mathbb{Z}_q^{n \times n}$ *if $\mathbf{A} \left[\begin{smallmatrix} \mathbf{R} \\ \mathbf{I} \end{smallmatrix}\right] = \mathbf{H}^* \cdot \mathbf{G}$. The* quality *of the trapdoor is defined to be the spectral norm $s_1(\mathbf{R})$.*

Note that $\mathbf{H}^*$ is uniquely determined and efficiently computable from $\mathbf{R}$, because $\mathbf{G}$ contains the n-by-n identity as a submatrix. Note also that if $\mathbf{R}$ is a trapdoor for $\mathbf{A}$ with tag $\mathbf{H}^*$, then it is also a trapdoor for $\mathbf{A_H} := \mathbf{A} - [\mathbf{0} \mid \mathbf{HG}]$ with tag $\mathbf{H}^* - \mathbf{H} \in \mathbb{Z}_q^{n \times n}$.

The key-generation algorithm of [MP12] produces a parity-check matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $\mathbf{R}$ having desired tag $\mathbf{H}^*$. It does so by choosing (or being given) a uniformly random $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and a random $\mathbf{R} \in \mathbb{Z}^{\bar{m} \times nk}$ having small $s_1(\mathbf{R})$, and outputs $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}^* \cdot \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$. For sufficiently large $m \geq Cn \lg q$ (where C is a universal constant) and appropriate distribution of $\mathbf{R}$, the output matrix $\mathbf{A}$ is uniformly random, up to $\mathrm{negl}(n)$ statistical distance.

The discrete Gaussian sampling algorithm of [MP12] is an instance of the "convolution" approach from [Pei10]. It works in two phases:

1. In the *offline* "perturbation" phase, it takes as input a parity-check matrix $\mathbf{A}$, a trapdoor $\mathbf{R}$ for $\mathbf{A}$ with some tag $\mathbf{H}^* \in \mathbb{Z}_q^{n \times n}$, and a Gaussian parameter $s \geq Cs_1(\mathbf{R})$ (where C is some universal constant). It chooses one or more Gaussian perturbation vectors $\mathbf{p} \in \mathbb{Z}^m$ (one for each future call to the online sampling step) having non-spherical covariance $\Sigma_{\mathbf{p}}$ that depends only on s and the trapdoor $\mathbf{R}$.
2. In the *online* "syndrome correction" phase, it is given a syndrome $\mathbf{u} \in \mathbb{Z}_q^n$ and a tag $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$. As long as $\mathbf{H}^* - \mathbf{H} \in \mathbb{Z}_q^{n \times n}$ is invertible, it chooses $\mathbf{z} \in \mathbb{Z}^{nk}$ having Gaussian distribution with parameter $s_{\mathbf{g}} \cdot \omega_n$ over an appropriate coset of $\Lambda^{\perp}(\mathbf{G})$, and outputs $\mathbf{x} = \mathbf{p} + \left[\begin{smallmatrix} \mathbf{R} \\ \mathbf{I} \end{smallmatrix}\right] \mathbf{z} \in \Lambda_{\mathbf{u}}^{\perp}(\mathbf{A_H})$, where $\mathbf{p}$ is a fresh perturbation from the offline step.

Informally, the perturbation covariance $\Sigma_{\mathbf{p}}$ of $\mathbf{p}$ is carefully designed to cancel out the trapdoor-revealing covariance of $\mathbf{y} = \left[\begin{smallmatrix} \mathbf{R} \\ \mathbf{I} \end{smallmatrix}\right] \mathbf{z}$, so that their sum has a (public) spherical Gaussian distribution. More formally, the output $\mathbf{x}$ has distribution within $\mathrm{negl}(n)$ statistical distance of $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A_H}), s \cdot \omega_n}$, and in particular does not reveal any information about the trapdoor $\mathbf{R}$ (aside from an upper bound s on $s_1(\mathbf{R})$, which is public).

We emphasize that for security, it is essential that none of the intermediate values $\mathbf{p}$, $\mathbf{z}$ or $\mathbf{y} = \left[\begin{smallmatrix} \mathbf{R} \\ \mathbf{I} \end{smallmatrix}\right] \mathbf{z}$ be revealed, otherwise they could be correlated with $\mathbf{x}$ to leak information about the trapdoor $\mathbf{R}$ that could lead to an attack like the one given in [NR06].

3.2 Functionalities for Threshold Sampling

Here we present ideal functionalities corresponding to the above two algorithms. The key-generation and Gaussian sampling functionalities $\mathcal{F}_{\text{KG}}$ and $\mathcal{F}_{\text{GS}}$ are specified in Figure 1 and Figure 2, respectively; they internally execute the standalone algorithms described above.

To realize $\mathcal{F}_{\text{KG}}$, in the trusted setup model (as used in [CG99]) we can simply let the trusted party play the role of $\mathcal{F}_{\text{KG}}$, because key generation is a one-time setup. To realize $\mathcal{F}_{\text{GS}}$, for the purpose of modularity we define two lower-level functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$ (Figures 3 and 4), which generate the perturbation and syndrome-correction components, respectively, as in the standalone sampling algorithm. The $\mathcal{F}_{\text{GS}}$, $\mathcal{F}_{\text{Perturb}}$, and $\mathcal{F}_{\text{Correct}}$ functionalities are all initialized with a bound B on the number of Gaussian samples that they will produce in their lifetimes. This is because the trusted setup (or offline precomputation) phases of our protocols need to prepare sufficient randomness so that the online phases can be noninteractive. (If the bound B is reached, then the parties can just initialize new copies of $\mathcal{F}_{\text{GS}}$, $\mathcal{F}_{\text{Perturb}}$, and $\mathcal{F}_{\text{Correct}}$ using the same arguments from $\mathcal{F}_{\text{KG}}$.)

Functionality $\mathcal{F}_{\text{KG}}$

Generate: Upon receiving $(\mathsf{gen}, sid, \bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}, \mathbf{H}^* \in \mathbb{Z}_q^{n \times n}, z)$ from at least h honest parties in $\mathcal{P}$:

- Choose $\mathbf{R} \leftarrow D_{\mathbb{Z}, z \cdot \omega_n}^{\bar{m} \times nk}$ and compute a sharing $[\![\mathbf{R}]\!]$ over $\mathbb{Z}_q$. Let $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}^* \cdot \mathbf{G} - \bar{\mathbf{A}}\mathbf{R}]$.
- Send $(\mathsf{gen}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i)$ to each party i in $\mathcal{P}$, and $(\mathsf{gen}, sid, \mathbf{A}, \mathbf{H}^*, z)$ to the adversary.

Figure 1: Key generation functionality

Functionality $\mathcal{F}_{\text{GS}}$

Initialize: Upon receiving $(\mathsf{init}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i, \mathbf{H}^*, s, B)$ from at least h honest parties i in $\mathcal{P}$:

- Reconstruct $\mathbf{R}$ and store sid, $\mathbf{A}$, $\mathbf{R}$, $\mathbf{H}^*$, s, and B.
- Send (init, sid) to each party in $\mathcal{P}$, and $(\mathsf{init}, sid, \mathbf{A}, \mathbf{H}^*, s, B)$ to the adversary.

Sample: Upon receiving $(\mathsf{sample}, sid, \mathbf{H} \in \mathbb{Z}_q^{n \times n}, \mathbf{u} \in \mathbb{Z}_q^n)$ from at least h honest parties in $\mathcal{P}$, if $\mathbf{H}^* - \mathbf{H} \in \mathbb{Z}_q^{n \times n}$ is invertible and fewer than B calls to sample have already been made:

- Sample $\mathbf{x} \leftarrow D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}_{\mathbf{H}}), s \cdot \omega_n}$ using the algorithm from [MP12] with trapdoor $\mathbf{R}$.
- Send $(\mathsf{sample}, sid, \mathbf{x})$ to all parties in $\mathcal{P}$, and $(\mathsf{sample}, sid, \mathbf{H}, \mathbf{u}, \mathbf{x})$ to the adversary.

Figure 2: Gaussian sampling functionality

We next describe the helper functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$, and describe how they can be realized efficiently and noninteractively using trusted setup.

3.2.1 Perturbation

Our perturbation functionality $\mathcal{F}_{\text{Perturb}}$ (Figure 3) corresponds to the offline perturbation phase of the standalone sampling algorithm. The $\mathsf{perturb}$ command does not take any inputs, so its results can be precomputed offline, before the command is actually invoked. The possibility of precomputation introduces one subtlety in the definition of the functionality, however. Notice that the functionality asks the adversary for share values $[\![\mathbf{p}]\!]^i$ for the corrupted parties, then generates shares for the honest parties that are consistent with those shares; clearly this does not affect the secrecy of $\mathbf{p}$. This formulation is needed for proving security of a protocol that precomputes shares of perturbations before any real calls to $\mathsf{perturb}$ are made (we give such a protocol in Appendix A.4). It allows the simulator to choose shares on its own when simulating the precomputation, and ensures that the functionality later distributes shares that are consistent with the simulation. Observe that with trusted setup, $\mathcal{F}_{\text{Perturb}}$ can be trivially realized by just precomputing and distributing shares of B samples in the initialization phase, which the parties then consume in the online phase.

Note that $\mathcal{F}_{\text{Perturb}}$ distributes shares $[\![\mathbf{p}]\!]^i$ of a perturbation $\mathbf{p}$ to the players, which themselves do not reveal any information about $\mathbf{p}$ to the adversary, just as in the standalone Gaussian sampling algorithm. However, in order for the perturbation to be useful in the later syndrome-correction phase, the parties will need to know (and so $\mathcal{F}_{\text{Perturb}}$ reveals) some partial information about $\mathbf{p}$, namely, the syndromes $\bar{\mathbf{w}} = [\bar{\mathbf{A}} \mid -\bar{\mathbf{A}}\mathbf{R}] \cdot \mathbf{p} \in \mathbb{Z}_q^n$ and $\mathbf{w} = [\mathbf{0} \mid \mathbf{G}] \cdot \mathbf{p} \in \mathbb{Z}_q^n$. This is the main significant difference with the standalone setting, in which these same syndromes are calculated internally but never revealed. Informally, Lemma 3.2 below shows that the syndromes are uniformly random (up to negligible error), and hence can be simulated without knowing $\mathbf{p}$. Furthermore, $\mathbf{p}$ will still be a usable perturbation even after $\bar{\mathbf{w}}, \mathbf{w}$ are revealed, because it has an appropriate (non-spherical) Gaussian parameter which sufficiently exceeds the smoothing parameter of an appropriate lattice. This fact will be used later in the proof of security for our $\mathcal{F}_{\text{GS}}$ realization.

Functionality $\mathcal{F}_{\text{Perturb}}$

Initialize: Upon receiving $(\mathsf{init}, sid, \mathbf{A}_{-\mathbf{H}^*} = [\bar{\mathbf{A}} \mid -\bar{\mathbf{A}}\mathbf{R}], [\![\mathbf{R}]\!]^i, s, B)$ from at least h honest parties i in $\mathcal{P}$:

- Reconstruct $\mathbf{R}$ to compute covariance matrix $\Sigma_{\mathbf{p}} = s^2 - s_{\mathbf{g}}^2 \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] \left[\begin{smallmatrix}\mathbf{R}^t & \mathbf{I}\end{smallmatrix}\right]$ and store sid, $\mathbf{A}_{-\mathbf{H}^*}$, and $\Sigma_{\mathbf{p}}$.
- Send (init, sid) to all parties in $\mathcal{P}$ and $(\mathsf{init}, sid, \mathbf{A}_{-\mathbf{H}^*}, s, B)$ to the adversary.

Perturb: Upon receiving $(\mathsf{perturb}, sid)$ from at least h honest parties in $\mathcal{P}$, if fewer than B calls to perturb have already been made:

- Choose $\mathbf{p} \leftarrow D_{\mathbb{Z}^m, \sqrt{\Sigma_{\mathbf{p}}} \cdot \omega_n}$.
- Compute $\bar{\mathbf{w}} = \mathbf{A}_{-\mathbf{H}^*} \cdot \mathbf{p} \in \mathbb{Z}_q^n$ and $\mathbf{w} = [\mathbf{0} \mid \mathbf{G}] \cdot \mathbf{p} \in \mathbb{Z}_q^n$.
- Send $(\mathsf{perturb}, sid, \bar{\mathbf{w}}, \mathbf{w})$ to the adversary, and receive back shares $[\![\mathbf{p}]\!]^i \in \mathbb{Z}_q^m$ for each currently corrupted party i in $\mathcal{P}$.
- Generate a uniformly random sharing $[\![\mathbf{p}]\!]$ consistent with the shares received in the previous step.
- Send $(\mathsf{perturb}, sid, [\![\mathbf{p}]\!]^i, \bar{\mathbf{w}}, \mathbf{w})$ to each party i in $\mathcal{P}$.

Figure 3: Perturbation functionality

Lemma 3.2. *Let $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ be uniformly random for $\bar{m} = m - nk \geq n \lg q + \omega(\log n)$, let*

$$\mathbf{B} = \begin{bmatrix} \bar{\mathbf{A}} & -\bar{\mathbf{A}}\mathbf{R} \\ & \mathbf{G} \end{bmatrix} = (\bar{\mathbf{A}} \oplus \mathbf{G}) \left[\begin{smallmatrix}\mathbf{I} & -\mathbf{R}\\ & \mathbf{I}\end{smallmatrix}\right] \in \mathbb{Z}_q^{2n \times (\bar{m}+nk)}$$

(where $\oplus$ denotes the direct sum), and let $\Lambda = \Lambda^\perp(\mathbf{B})$. Then with all but $\mathrm{negl}(n)$ probability over the choice of $\bar{\mathbf{A}}$, we have $\eta_\epsilon(\Lambda^\perp(\mathbf{B})) \leq \sqrt{5}(s_1(\mathbf{R}) + 1) \cdot \omega_n$ for some $\epsilon = \mathrm{negl}(n)$.

In particular, for $\mathbf{p} \leftarrow D_{\mathbb{Z}^m, \sqrt{\Sigma_{\mathbf{p}}}}$ where $\sqrt{\Sigma_{\mathbf{p}}} \geq 6(s_1(\mathbf{R}) + 1) \cdot \omega_n \geq 2\eta_\epsilon(\Lambda^\perp(\mathbf{B}))$, the syndrome $\mathbf{u} = (\bar{\mathbf{w}}, \mathbf{w}) = \mathbf{B}\mathbf{p} \in \mathbb{Z}_q^{2n}$ is $\mathrm{negl}(n)$-far from uniform, and the conditional distribution of $\mathbf{p}$ given $\mathbf{u}$ is $D_{\Lambda_{\mathbf{u}}^\perp(\mathbf{B}), \sqrt{\Sigma_{\mathbf{p}}}}$.

Proof. By Lemma 2.4, we have $\eta_{\epsilon'}(\Lambda^\perp(\bar{\mathbf{A}})) \leq 2 \cdot \omega_n$ (with overwhelming probability) for some $\epsilon' = \mathrm{negl}(n)$. Also as shown in [MP12], we have $\eta_{\epsilon'}(\Lambda^\perp(\mathbf{G})) \leq \sqrt{5} \cdot \omega_n$ (see Section 3.1). This implies that

$$\eta_\epsilon(\Lambda^\perp(\bar{\mathbf{A}} \oplus \mathbf{G})) \leq \sqrt{5} \cdot \omega_n$$

where $(1 + \epsilon) = (1 + \epsilon')^2$, and in particular $\epsilon = \mathrm{negl}(n)$.

Since $\mathbf{T} = \left[\begin{smallmatrix}\mathbf{I} & -\mathbf{R}\\ & \mathbf{I}\end{smallmatrix}\right]$ is unimodular with inverse $\mathbf{T}^{-1} = \left[\begin{smallmatrix}\mathbf{I} & \mathbf{R}\\ & \mathbf{I}\end{smallmatrix}\right]$, it is easy to verify that $\Lambda^\perp(\mathbf{B}) = \mathbf{T}^{-1} \cdot \Lambda^\perp(\bar{\mathbf{A}} \oplus \mathbf{G})$, and hence

$$\eta_\epsilon(\Lambda^\perp(\mathbf{B})) \leq s_1(\mathbf{T}^{-1}) \cdot \eta_\epsilon(\Lambda^\perp(\bar{\mathbf{A}} \oplus \mathbf{G})) \leq \sqrt{5}(s_1(\mathbf{R}) + 1) \cdot \omega_n. \qquad \square$$

3.2.2 Syndrome Correction

Our functionality $\mathcal{F}_{\text{Correct}}$ (Figure 4) corresponds to the syndrome-correction step of the standalone sampling algorithm. Because its output $\mathbf{y}$ must lie in a certain coset $\Lambda_{\mathbf{v}}^\perp(\mathbf{A})$, where $\mathbf{v}$ depends on the desired final syndrome $\mathbf{u}$, the functionality must be invoked online. As indicated in the overview, the standalone algorithm samples $\mathbf{z} \leftarrow \Lambda_{\mathbf{v}}^\perp(\mathbf{G})$ and defines $\mathbf{y} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] \mathbf{z}$. The functionality does the same, but outputs only shares of $\mathbf{y}$ to their respective owners. This ensures that no information about $\mathbf{y}$ is revealed to the adversary. (Note that the input syndrome $\mathbf{v}$ itself is not revealed in the standalone algorithm, but in our setting $\mathbf{v}$ is determined

Functionality $\mathcal{F}_{\text{Correct}}$

Initialize: Upon receiving $(\mathsf{init}, sid, [\![\mathbf{R}]\!]^i, B)$ from at least h honest parties i in $\mathcal{P}$:

- Reconstruct $\mathbf{R}$ and store sid, $\mathbf{R}$, and B.
- Send (init, sid) to all parties in $\mathcal{P}$ and (init, sid, B) to the adversary.

Correct: Upon receiving $(\mathsf{correct}, sid, \mathbf{v})$ from at least h honest parties in $\mathcal{P}$, if fewer than B calls to correct have already been made:

- Sample $\mathbf{z} \leftarrow D_{\Lambda^{\perp}_{\mathbf{v}}(\mathbf{G}), s_{\mathbf{g}} \cdot \omega_n}$ and compute $\mathbf{y} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] \mathbf{z}$.
- Send $(\mathsf{correct}, sid, \mathbf{v})$ to the adversary, receive shares $[\![\mathbf{y}]\!]^i \in \mathbb{Z}_q^m$ for each currently corrupted party i, and generate a uniformly random sharing $[\![\mathbf{y}]\!]$ consistent with these shares.
- Send $(\mathsf{correct}, sid, [\![\mathbf{y}]\!]^i)$ to each party i in $\mathcal{P}$.

Figure 4: Syndrome correction functionality

entirely by public information, so it is known to the adversary.) Also, just like $\mathcal{F}_{\text{Perturb}}$, the functionality asks the simulator for shares for the corrupted parties, to make precomputation simulatable.

Realizing $\mathcal{F}_{\text{Correct}}$ with a noninteractive protocol relies crucially on the *parallel* and *offline* nature of the corresponding step in the standalone algorithm. In particular, we use the fact that without knowing $\mathbf{v}$ in advance, the algorithm can precompute *partial* samples for each of the $q = \text{poly}(n)$ scalar values $v \in \mathbb{Z}_q$, and then linearly combine n such partial samples to answer a query for a full syndrome $\mathbf{v} \in \mathbb{Z}_q^n$.

In the trusted setup model, the protocol realizing $\mathcal{F}_{\text{Correct}}$ is as follows.

1. In the offline phase, a trusted party uses the trapdoor $\mathbf{R}$ (with tag $\mathbf{H}^*$) to distribute shares as follows. For each $j \in [n]$ and $v \in \mathbb{Z}_q$, the party initializes queues $Q^i_{j,v}$ for each party i, does the following B times, and then gives each of the resulting queues $Q^i_{j,v}$ to party i.

   - Sample $\mathbf{z}_{j,v} \leftarrow D_{\Lambda^{\perp}_{v}(\mathbf{g}^t), s_{\mathbf{g}} \cdot \omega_n}$.
   - Compute $\mathbf{y}_{j,v} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] (\mathbf{e}_j \otimes \mathbf{z}_{j,v})$, where $\mathbf{e}_j \in \mathbb{Z}^n$ denotes the jth standard basis vector. Note that
   $$\mathbf{A}_{\mathbf{H}} \cdot \mathbf{y}_{j,v} = (\mathbf{H}^* - \mathbf{H})\mathbf{G} \cdot (\mathbf{e}_j \otimes \mathbf{z}_{j,v}) = (\mathbf{H}^* - \mathbf{H})(v \cdot \mathbf{e}_j),$$
   where as always, $\mathbf{A}_{\mathbf{H}} = \mathbf{A} - [\mathbf{0} \mid \mathbf{HG}]$ for any $\mathbf{H} \in \mathbb{Z}_q^{n \times n}$.
   - Generate a sharing for $\mathbf{y}_{j,v}$, and add $[\![\mathbf{y}_{j,v}]\!]^i$ to queue $Q^i_{j,v}$ for each party $i \in \mathcal{P}$.

2. In the online phase, upon receiving $(\mathsf{correct}, sid, \mathbf{v})$, each party i dequeues an entry $[\![\mathbf{y}_{j,v_j}]\!]^i$ from Q_{j,v_j} for each $j \in [n]$, and locally outputs $[\![\mathbf{y}]\!]^i = \sum_{j \in [n]} [\![\mathbf{y}_{j,v_j}]\!]^i$. Note that by linearity and the homomorphic properties of secret sharing, the shares $[\![\mathbf{y}]\!]^i$ recombine to $\mathbf{y} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] \mathbf{z} \in \mathbb{Z}^m$ for some Gaussian-distributed $\mathbf{z}$ of parameter $s_{\mathbf{g}} \cdot \omega_n$, such that $\mathbf{A}_{\mathbf{H}} \cdot \mathbf{y} = (\mathbf{H}^* - \mathbf{H}) \cdot \mathbf{v} \in \mathbb{Z}_q^n$.

Without trusted setup, we give in Appendix A.3 an efficient protocol for $\mathcal{F}_{\text{Correct}}$ that operates in a similar way, populating the local queues $Q^i_{j,v}$ during the offline phase in a distributed manner using standard share-blinding and multiplication functionalities, among others.

3.2.3 Legal Uses of the Functionalities

Putting the key-generation and Gaussian sampling operations into separate functionalities $\mathcal{F}_{\text{KG}}$ and $\mathcal{F}_{\text{GS}}$ (and $\mathcal{F}_{\text{DelTrap}}$ for delegation, in Section 3.4 below), and realizing $\mathcal{F}_{\text{GS}}$ using the helper functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$, aids modularity and simplifies the analysis of our protocols. However, as a side effect it also raises a

technical issue in the UC framework, since environments can in general provide functionalities with arbitrary inputs, even on behalf of honest users. The issue is that the functionalities are all designed to be initialized with some *common, valid* state—namely, shares of a trapdoor $\mathbf{R}$ for a matrix $\mathbf{A}$ as produced by $\mathcal{F}_{\text{KG}}$ on valid inputs—but it might be expensive or impossible for the corresponding protocols to check the consistency and validity those shares. Moreover, such checks would be unnecessary in the typical case where an application protocol, such as a threshold signature scheme, initializes the functionalities as intended.[3]

Therefore, we prove UC security for a restricted class of environments $\mathscr{Z}$ that always initialize our functionalities with valid arguments. In particular, environments in $\mathscr{Z}$ can instruct parties to instantiate $\mathcal{F}_{\text{KG}}$ only with appropriate arguments $\bar{\mathbf{A}}, z$. Similarly, $\mathcal{F}_{\text{GS}}$ (and $\mathcal{F}_{\text{DelTrap}}$) can be initialized only with a matrix $\mathbf{A}$, tag $\mathbf{H}^*$, and shares of a trapdoor $\mathbf{R}$ matching those of a prior call to the gen command of $\mathcal{F}_{\text{KG}}$, and with a sufficiently large Gaussian parameter $s \geq Cs_1 \cdot \omega_n$, where s_1 is a high-probability upper bound on $s_1(\mathbf{R})$ for the trapdoor $\mathbf{R}$ generated by $\mathcal{F}_{\text{KG}}$. The functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$ (which in any case are not intended for direct use by applications) also must be initialized using a prior output of $\mathcal{F}_{\text{KG}}$.

More formally, an environment $\mathcal{Z}$ is said to be in the class $\mathscr{Z}$ if it satisfies the following conditions specific to our functionalities:

- When $\mathcal{Z}$ instructs honest parties to run the gen command of $\mathcal{F}_{\text{KG}}$, the input matrix $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ and parameter z must correspond with a statistically secure instantiation of the trapdoor generator from [MP12]. Concretely, $\bar{\mathbf{A}}$ must be statistically close to uniformly random, and the Gaussian parameter z and dimension $\bar{m}$ must jointly be sufficiently large. As shown in [MP12], one valid instantiation is to let $z \geq 2\omega_n$ and $\bar{m} \geq Cn \lg q$ for any fixed constant $C > 1$.

- When $\mathcal{Z}$ instructs honest parties to run the init commands of $\mathcal{F}_{\text{GS}}$, $\mathcal{F}_{\text{Perturb}}$, or $\mathcal{F}_{\text{Correct}}$, the matrix $\mathbf{A}$, tag $\mathbf{H}^*$, and shares $[\![\mathbf{R}]\!]^i$ provided as the parties' inputs must match those of a prior call to $\mathcal{F}_{\text{KG}}$.gen. In addition, these init commands must all use the same Gaussian parameter s, which must be sufficiently large relative to the Gaussian parameter z and dimension $\bar{m}$ used in that call to $\mathcal{F}_{\text{KG}}$.gen. Specifically, we require $s \geq Cz(\sqrt{\bar{m}} + \sqrt{n \log q}) \cdot \omega_n$ for a certain universal constant C. By the results of [MP12], this guarantees that with overwhelming probability over the choice of $\mathbf{R}$, we have $s \geq C' s_1(\mathbf{R}) \cdot \omega_n$ for some universal constant C', which ensures that our π_{GS} protocol produces the proper distribution.

We emphasize that these restrictions on the environment are not actually limiting in any meaningful way, since our functionalities are only intended to serve as subroutines in higher-level applications, e.g. threshold signatures and (H)IBE. When designing a protocol ϕ that uses these functionalities (see, e.g., Section 4) one simply needs to ensure that ϕ does so in a manner consistent with the above conditions. Then composing ϕ with our protocols π_{KG}, π_{GS}, π_{Perturb}, and π_{Correct} (which we prove secure against environments in $\mathscr{Z}$) will yield a secure protocol against *any* t-limited environment.

3.3 Gaussian Sampling Protocol

Figure 5 defines a protocol π_{GS} that realizes the Gaussian sampling functionality $\mathcal{F}_{\text{GS}}$ in the $(\mathcal{F}_{\text{Perturb}}, \mathcal{F}_{\text{Correct}})$-hybrid model. Its sample command simply makes one call to each of the main commands of $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$, adjusting the requested syndrome as necessary to ensure that the syndrome of the final output is the desired one. (This is done exactly as in the standalone algorithm.) The shares of the perturbation $\mathbf{p}$ and syndrome-correction term $\mathbf{y}$ are then added locally and announced, allowing the players to reconstruct the final output $\mathbf{x} = \mathbf{p} + \mathbf{y}$. The security of π_{GS} is formalized in Theorem 3.3, and proved via the simulator $\mathcal{S}_{\text{GS}}$ in Figure 6.

[3] This issue is not limited to our setting, and can arise any time the key-generation and secret-key operations of a threshold scheme are put into separate functionalities. We note that using "joint state" [CR03] does not appear to resolve the issue, because it only allows multiple instances of the *same* protocol to securely share some joint state.

An essential point is that given the helper functionalities, the protocol π_{GS} is completely *noninteractive*, i.e., no messages are exchanged among the parties, except when broadcasting their shares of the final output. Similarly, recall that our realizations of $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$ are also noninteractive, either when using trusted setup or offline precomputation (see Appendix A). In other words, in the fully realized sampling protocol, the parties can sample from any desired coset using only local computation, plus one broadcast of the final output shares. We emphasize that this kind of noninteractivity is nontrivial, because the number of possible cosets is exponentially large.

Protocol π_{GS} in the ($\mathcal{F}_{\text{Perturb}}$, $\mathcal{F}_{\text{Correct}}$)-hybrid model

Initialize: On input $(\mathsf{init}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i, \mathbf{H}^*, s, B)$, party i stores $\mathbf{H}^*$, calls $\mathcal{F}_{\text{Perturb}}(\mathsf{init}, sid, \mathbf{A}_{-\mathbf{H}^*}, [\![\mathbf{R}]\!]^i, s, B)$ and $\mathcal{F}_{\text{Correct}}(\mathsf{init}, sid, [\![\mathbf{R}]\!]^i, B)$, and outputs (init, sid).

Sample: On input $(\mathsf{sample}, sid, \mathbf{H}, \mathbf{u})$, if $\mathbf{H}^* - \mathbf{H} \in \mathbb{Z}_q^{n \times n}$ is invertible, and if fewer than B calls to sample have already been made, then party i does:

- Call $\mathcal{F}_{\text{Perturb}}(\mathsf{perturb}, sid)$ and receive $(\mathsf{perturb}, sid, [\![\mathbf{p}]\!]^i, \bar{\mathbf{w}}, \mathbf{w})$.
- Compute $\mathbf{v} = (\mathbf{H}^* - \mathbf{H})^{-1}(\mathbf{u} - \bar{\mathbf{w}}) - \mathbf{w} \in \mathbb{Z}_q^n$.
- Call $\mathcal{F}_{\text{Correct}}(\mathsf{correct}, sid, \mathbf{v})$ and receive $(\mathsf{correct}, sid, [\![\mathbf{y}]\!]^i)$.
- Broadcast $[\![\mathbf{x}]\!]^i = [\![\mathbf{p}]\!]^i + [\![\mathbf{y}]\!]^i$ and reconstruct $\mathbf{x} = \mathbf{p} + \mathbf{y}$ from the announced shares.
- Output $(\mathsf{sample}, sid, \mathbf{x})$.

Figure 5: Gaussian sampling protocol

Simulator $\mathcal{S}_{\text{GS}}$

Initialize: Upon receiving $(\mathsf{init}, sid, \mathbf{A}, \mathbf{H}^*, s, B)$ from $\mathcal{F}_{\text{GS}}$, reveal to $\mathcal{Z}$ (init, sid) as outputs of both $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$ to each currently corrupted party and any party that is corrupted in the future.

Sample: Upon receiving $(\mathsf{sample}, sid, \mathbf{H}, \mathbf{u}, \mathbf{x})$ from $\mathcal{F}_{\text{GS}}$:

- Choose uniform and independent $\bar{\mathbf{w}}, \mathbf{w} \in \mathbb{Z}_q^n$ and compute $\mathbf{v} = (\mathbf{H}^* - \mathbf{H})^{-1}(\mathbf{u} - \bar{\mathbf{w}}) - \mathbf{w} \in \mathbb{Z}_q^n$.
- On behalf of $\mathcal{F}_{\text{Perturb}}$, send $(\mathsf{perturb}, sid, \bar{\mathbf{w}}, \mathbf{w})$ to $\mathcal{Z}$ and receive back shares $[\![\mathbf{p}]\!]^i$ for each currently corrupted party i in $\mathcal{P}$. Generate a uniformly random sharing $[\![\mathbf{p}]\!]$ of $\mathbf{p} = \mathbf{0}$ consistent with these shares. Send $(\mathsf{perturb}, sid, [\![\mathbf{p}]\!]^i, \bar{\mathbf{w}}, \mathbf{w})$ to each corrupted party i in $\mathcal{P}$ on behalf of $\mathcal{F}_{\text{Perturb}}$.
- On behalf of $\mathcal{F}_{\text{Correct}}$, send $(\mathsf{correct}, sid, \mathbf{v})$ to $\mathcal{Z}$ and receive back shares $[\![\mathbf{y}]\!]^i$ for each currently corrupted party i in $\mathcal{P}$. Generate a uniformly random sharing $[\![\mathbf{y}]\!]$ of $\mathbf{y} = \mathbf{x}$ consistent with these shares. Send $(\mathsf{correct}, sid, [\![\mathbf{y}]\!]^i)$ to each corrupted party i in $\mathcal{P}$ on behalf of $\mathcal{F}_{\text{Correct}}$.
- Broadcast $[\![\mathbf{x}]\!]^i = [\![\mathbf{p}]\!]^i + [\![\mathbf{y}]\!]^i$ on behalf of each honest party i.

Corruption: When $\mathcal{Z}$ requests to corrupt party i, for each previous call to sample, reveal the corresponding messages $(\mathsf{perturb}, sid, [\![\mathbf{p}]\!]^i, \bar{\mathbf{w}}, \mathbf{w})$ and $(\mathsf{correct}, sid, [\![\mathbf{y}]\!]^i)$ to party i on behalf of $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$, respectively.

Figure 6: Simulator for π_{GS}

Theorem 3.3. *Protocol π_{GS} statistically realizes $\mathcal{F}_{GS}$ in the ($\mathcal{F}_{Perturb}$, $\mathcal{F}_{Correct}$)-hybrid model for t-limited environments in $\mathscr{Z}$.*

Proof sketch. The simulator $\mathcal{S}_{\text{GS}}$ in Figure 6 maintains consistent sharings of $\mathbf{p} = \mathbf{0}$ and $\mathbf{y} = \mathbf{x}$ for each call to sample, and it releases player i's shares of these values (on behalf of $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$) upon corruption

of player i. The fact that $\mathbf{p}$ and $\mathbf{y}$ in $\mathcal{S}_{\text{GS}}$ are from incorrect distributions is not detectable (even statistically) by the environment $\mathcal{Z}$, because it sees at most t shares of each, and the shares are consistent with announced shares of $\mathbf{x} = \mathbf{p} + \mathbf{y}$.

The only other significant issues relate to (1) the syndromes $\bar{\mathbf{w}}, \mathbf{w}$ output publicly by $\mathcal{F}_{\text{Perturb}}$ in the $(\mathcal{F}_{\text{Perturb}}, \mathcal{F}_{\text{Correct}})$-hybrid world, versus the simulator's choices of those values on behalf of $\mathcal{F}_{\text{Perturb}}$ in the ideal world; and (2) the distribution (conditioned on any fixed $\bar{\mathbf{w}}, \mathbf{w}$) of the final output $\mathbf{x}$ in both worlds. For item (1), as proved in Lemma 3.2, in the hybrid world the syndromes $\bar{\mathbf{w}}, \mathbf{w}$ are jointly uniform and independent (up to negligible statistical distance) over the choice of $\mathbf{p}$ by $\mathcal{F}_{\text{Perturb}}$, just as they are when produced by the simulator. Moreover, conditioned on any fixed values of $\bar{\mathbf{w}}, \mathbf{w}$, the distribution of $\mathbf{p}$ in the hybrid world is a discrete Gaussian with covariance $\Sigma_{\mathbf{p}}$ over a certain lattice coset $\Lambda^{\perp}_{\mathbf{u}}(\mathbf{B})$, and the actual value of $\mathbf{p}$ from this distribution is perfectly hidden by the secret-sharing scheme.

For item (2), the above facts imply that in the hybrid world, $\mathbf{x} = \mathbf{p} + \mathbf{y}$ has spherical discrete Gaussian distribution $D_{\Lambda^{\perp}_{\mathbf{u}}(\mathbf{A_H}),s}$, just as the output $\mathbf{x}$ of $\mathcal{F}_{\text{GS}}$ does in the ideal world (up to negligible statistical error in both cases). The proof is essentially identical to that of the "convolution lemma" from [MP12], which guarantees the correctness of the standalone sampling algorithm (as run by $\mathcal{F}_{\text{GS}}$ in the ideal world). The only slight difference is that in the hybrid world, $\mathbf{p}$'s distribution (conditioned on any fixed values of $\bar{\mathbf{w}}, \mathbf{w}$) is a discrete Gaussian with parameter $\sqrt{\Sigma_{\mathbf{p}}}$ over a coset of $\Lambda^{\perp}(\mathbf{B})$, instead of over $\mathbb{Z}^m$ as in the standalone algorithm. Fortunately, Lemma 3.2 says that $\sqrt{\Sigma_{\mathbf{p}}} \geq 2\eta_{\epsilon}(\Lambda^{\perp}(\mathbf{B}))$, and this is enough to adapt the proof from [MP12] to the different distribution of $\mathbf{p}$.

Finally, by the homomorphic properties of secret sharing, the shares $[\![\mathbf{p}]\!]^i + [\![\mathbf{y}]\!]^i$ announced by the honest parties are jointly distributed exactly as a fresh sharing of $\mathbf{x}$ as produced by the simulator. We conclude that the hybrid and real views are statistically indistinguishable, as desired. □

3.4 Trapdoor Delegation

The trapdoor delegation functionality $\mathcal{F}_{\text{DelTrap}}$ given in Figure 7 corresponds to the algorithm $\mathsf{DelTrap}$ for delegating a lattice trapdoor in [MP12], which is used in hierarchical IBE schemes. The functionality is initialized with shares of a trapdoor $\mathbf{R}$ for some $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$. For an extended matrix $\mathbf{A}' = [\mathbf{A_H} | \mathbf{A}_1] \in \mathbb{Z}_q^{n \times (m+nk)}$ (where $\mathbf{A_H} = \mathbf{A} - [\mathbf{0} \mid \mathbf{HG}]$) and tag $\mathbf{H}' \in \mathbb{Z}_q^{n \times n}$, $\mathcal{F}_{\text{DelTrap}}$ outputs shares of a trapdoor $\mathbf{R}'$ for $\mathbf{A}'$ with tag $\mathbf{H}'$, where the distribution of $\mathbf{R}'$ is Gaussian and in particular is independent of $\mathbf{R}$.

A realization of $\mathcal{F}_{\text{DelTrap}}$ in the $\mathcal{F}_{\text{GS}}$-hybrid model is given by π_{DelTrap} in Figure 8. It is entirely straightforward, since the standalone algorithm from [MP12] just draws several Gaussian samples over appropriate (publicly computable) cosets of $\Lambda^{\perp}(\mathbf{A})$, so we omit the proof of security.

4 Threshold Signatures and IBE

Here we apply our protocols in a straightforward manner to give threshold versions of the signature and identity-based encryption schemes from [GPV08]. Other signature and (H)IBE schemes that use key-generation and Gaussian sampling as "black boxes" can be similarly adapted to the threshold setting.

The GPV schemes. For security parameter n, modulus q and message space $\mathcal{M}$, the GPV signature scheme uses a hash function $H \colon \mathcal{M} \to \mathbb{Z}_q^n$, which is modeled as a random oracle, and two algorithms $\mathsf{GenTrap}$ and $\mathsf{SampleD}$. At a high level, $\mathsf{GenTrap}(n, q, m)$ generates a nearly uniform matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a trapdoor $\mathbf{R}$. Using these, $\mathsf{SampleD}(\mathbf{A}, \mathbf{R}, \mathbf{u}, s)$ generates a Gaussian sample (for any sufficiently large

Functionality $\mathcal{F}_{\text{DelTrap}}$

Initialize: Upon receiving $(\mathsf{init}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i, \mathbf{H}^*, s, B)$ from at least h honest parties in $\mathcal{P}$:

- Reconstruct trapdoor $\mathbf{R}$ and its invertible tag $\mathbf{H}$ for $\mathbf{A}$, and store sid, $\mathbf{A}$, $\mathbf{R}$, $\mathbf{H}^*$ and s.
- Send (init, sid) to each party in $\mathcal{P}$, and $(\mathsf{init}, sid, \mathbf{A}, s, B)$ to the adversary.

Delegate: Upon receiving $(\mathsf{delegate}, sid, \mathbf{H}, \mathbf{A}_1, \mathbf{H}')$ from at least h honest parties in $\mathcal{P}$:

- If $\mathbf{H}^* - \mathbf{H} \in \mathbb{Z}_q^{n \times n}$ is invertible, using the Gaussian sampling algorithm from [MP12] with trapdoor $\mathbf{R}$, sample each column of $\mathbf{R}'$ independently from a discrete Gaussian with parameter s over the appropriate coset of $\Lambda^{\perp}(\mathbf{A}_{\mathbf{H}})$, so that $\mathbf{A}_{\mathbf{H}} \cdot \mathbf{R}' = \mathbf{H}' \cdot \mathbf{G} - \mathbf{A}_1$.
- Compute a sharing $[\![\mathbf{R}']\!]$ over $\mathbb{Z}_q$, and send $(\mathsf{delegate}, sid, [\![\mathbf{R}']\!]^i)$ to each party i, and $(\mathsf{delegate}, sid, \mathbf{H}, \mathbf{A}_1, \mathbf{H}')$ to the adversary.

Figure 7: Functionality for delegating a lattice trapdoor

Protocol π_{DelTrap} in the $\mathcal{F}_{\text{GS}}$-hybrid model

Initialize: On input $(\mathsf{init}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i, \mathbf{H}^*, s, B)$, call $\mathcal{F}_{\text{GS}}(\mathsf{init}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i, \mathbf{H}^*, s, Bnk)$.

Delegate: On input $(\mathsf{delegate}, sid, \mathbf{H}, \mathbf{A}_1, \mathbf{H}')$, party i does the following:

- For each $j = 1, \ldots, nk$, call $\mathcal{F}_{\text{GS}}(\mathsf{sample}, sid, \mathbf{H}, \mathbf{u})$ where $\mathbf{u}$ is the jth column of $\mathbf{H}'\mathbf{G} - \mathbf{A}_1$; receive $(\mathsf{sample}, sid, \mathbf{r}'_j)$ from $\mathcal{F}_{\text{GS}}$ and let $\mathbf{r}'_j$ be the jth column of $\mathbf{R}'$.
- Output $(\mathsf{sample}, sid, \mathbf{R}')$.

Figure 8: Protocol for delegating a lattice trapdoor

parameter s) over the lattice coset $\Lambda^{\perp}_{\mathbf{u}}(\mathbf{A})$. Ignoring the exact selection of parameters, the stateful version of the signature scheme consists of the following three algorithms:

- $\mathsf{KeyGen}(1^n)$: Let $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{GenTrap}(n, q, m)$ and output verification key $vk = \mathbf{A}$ and signing key $sk = \mathbf{R}$.
- $\mathsf{Sign}(sk, \mu \in \mathcal{M})$: If (μ, σ) is already in local storage, output the signature σ. Otherwise, let $\mathbf{x} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{R}, H(\mu), s)$ and store (μ, σ). Output the signature $\sigma = \mathbf{x}$.
- $\mathsf{Verify}(vk, \mu, \sigma = \mathbf{x})$: If $\mathbf{Ax} = H(\mathbf{m})$ and $\mathbf{x}$ is sufficiently short, then accept; otherwise, reject.

See [GPV08] for the proof of (strong) unforgeability under worst-case lattice assumptions.

In the GPV identity-based encryption scheme, the setup algorithm is the same as KeyGen above, and the master public and secret keys are simply the verification and signing keys above. The secret key for an individual identity is a signature on that identity. Since we are concerned only with thresholdizing the signing and key-extraction algorithms, the details of the encryption and decryption algorithms are unchanged and irrelevant here, so we need only give threshold version of KeyGen and Sign.

Thresholdizing. In order to obtain a threshold signature scheme, KeyGen and Sign must be done in a distributed way, so that the signing key $sk = \mathbf{R}$ is distributed among the participating parties and a valid signature σ can only be produced by a quorum of participating parties. In Figure 9 we recall (from [ADN06]) a formal functionality for threshold signatures. (Recall that $\mathcal{P}$ is the set of trustees, or parties authorized to receive shares of the signing key.)

Functionality $\mathcal{F}_{\text{TSig}}$

Generate: Upon receiving (gen, sid, B) from at least h honest parties in $\mathcal{P}$, send (gen, sid, B) to the adversary, receive and record verification key v, and send (gen, sid, v) to each party in $\mathcal{P}$.

Sign: Upon receiving (sign, sid, m) from at least h honest parties in $\mathcal{P}$, if fewer than B calls to sign have already been made:

- Send (sign, sid, m) to the adversary and receive signature σ.
- If there is no record of $(m, \sigma, v, 0)$, record $(m, \sigma, v, 1)$ and send $(\mathsf{sign}, sid, m, \sigma)$ to each party in $\mathcal{P}$.

Verify: Upon receiving $(\mathsf{ver}, sid, m, \sigma, v')$ from any party $p \in \mathcal{P}$:

- Send $(\mathsf{ver}, sid, m, \sigma, v')$ to the adversary and receive $(\mathsf{ver}, sid, m, \phi)$.
- If $v' = v$ and $(m, \sigma, v, 1)$ is recorded, then send $(\mathsf{ver}, sid, m, 1)$ to p.
- If $v' = v$ and there is no recorded $(m, \sigma', v, 1)$, then record $(m, \sigma, v, 0)$ and send $(\mathsf{ver}, sid, m, 0)$ to p.
- If some $(m, \sigma, v', 1)$ is recorded, then send $(\mathsf{ver}, sid, m, 1)$ to p.
- If some $(m, \sigma, v', 0)$ is recorded, then send $(\mathsf{ver}, sid, m, 0)$ to p.
- Otherwise, record (m, σ, v', ϕ) and send $(\mathsf{ver}, sid, m, \sigma, \phi)$ to p.

Figure 9: Threshold signature functionality

To construct a protocol for threshold GPV signatures we need threshold analogues of GenTrap and SampleD; these are the functionalities $\mathcal{F}_{\text{KG}}$ and $\mathcal{F}_{\text{GS}}$ (from Section 3), respectively. $\mathcal{F}_{\text{KG}}$ produces $\mathbf{A}$ as usual, but each party i receives a share $[\![\mathbf{R}]\!]^i$ of the trapdoor. To produce a signature, each party i in a quorum of signers simply calls $\mathcal{F}_{\text{GS}}.\mathsf{sample}$ with his share $[\![\mathbf{R}]\!]^i$, and this allows them to collectively produce a signature σ.

In Figure 10 we present a protocol for threshold GPV signatures in the $(\mathcal{F}_{\text{KG}}, \mathcal{F}_{\text{GS}})$-hybrid model. Note that it obeys all the constraints on the usage of $\mathcal{F}_{\text{KG}}$ and $\mathcal{F}_{\text{GS}}$ described in Section 3.2.3. Its security is easily proved using the correspondence between $\mathcal{F}_{\text{KG}}$ and KeyGen, and $\mathcal{F}_{\text{GS}}$ and Sign, so we state Theorem 4.1 without proof.

Theorem 4.1. *The protocol $\pi_{ThreshGPV}$ securely realizes $\mathcal{F}_{TSig}$, assuming the unforgeability of the GPV signature scheme (with the same parameters) under chosen-message attacks.*

References

[ABB10] S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572. 2010.

[ADN06] J. F. Almansa, I. Damgård, and J. B. Nielsen. Simplified threshold RSA with adaptive and proactive security. In *EUROCRYPT*, pages 593–611. 2006.

[AFV11] S. Agrawal, D. M. Freeman, and V. Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*. 2011. To appear.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[Ajt99] M. Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9. 1999.

Protocol $\pi_{\text{ThreshGPV}}$ in the $(\mathcal{F}_{\text{KG}}, \mathcal{F}_{\text{GS}})$-hybrid model

Generate: On input (gen, sid, B):

- The parties run an information-theoretically secure coin-flipping protocol to choose a uniformly random $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$, and let $\mathbf{H}^* = \mathbf{I}_n$ (the n-by-n identity matrix).
- Each party i calls $\mathcal{F}_{\text{KG}}(\mathsf{gen}, sid, \bar{\mathbf{A}}, \mathbf{H}^*, z = 2 \cdot \omega_n)$ and receives $(\mathsf{gen}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i)$.
- Each party i calls $\mathcal{F}_{\text{GS}}(\mathsf{init}, sid, \mathbf{A}, [\![\mathbf{R}]\!]^i, \mathbf{H}^* = \mathbf{I}_n, s = Cz \cdot O(\sqrt{n \log q}) \cdot \omega_n, B)$ and outputs $(\mathsf{gen}, sid, \mathbf{A})$.

Sign: On input (sign, sid, m), if fewer than B calls to sign have already been made, then party i does:

- First check if (sid, m, σ) is already in local storage. If so, output $(\mathsf{sign}, sid, m, \sigma)$.
- Otherwise, compute $\mathbf{u} = H(m)$, call $\mathcal{F}_{\text{GS}}(\mathsf{sample}, sid, \mathbf{H} = \mathbf{0}, \mathbf{u})$, and receive $(\mathsf{sample}, sid, \bar{\mathbf{x}} \in \mathbb{Z}_q^m)$. Interpret $\bar{\mathbf{x}}$ as the unique integer vector $\mathbf{x} \in \mathbb{Z}^m$ with entries in $[-q/2, q/2)$, store $(sid, m, \sigma = \mathbf{x})$ locally, and output $(\mathsf{sign}, sid, m, \sigma)$.

Verify: Upon receiving $(\mathsf{ver}, sid, m, \sigma = \mathbf{x}, v')$:

- Each party i checks that $v' = \mathbf{A}$, that $\mathbf{A}\mathbf{x} = H(m)$, and that $\|\mathbf{x}\| \leq s\sqrt{m}$. If so, then party i outputs $(\mathsf{ver}, sid, m, 1)$, otherwise it outputs $(\mathsf{ver}, sid, m, 0)$.

Figure 10: Threshold GPV Signatures

[ALR11] G. Asharov, Y. Lindell, and T. Rabin. Perfectly-secure multiplication for any $< n/3$. In *CRYPTO*, pages 240–258. 2011.

[AP09] J. Alwen and C. Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, April 2011. Preliminary version in STACS 2009.

[BCHK07] D. Boneh, R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.

[BD10] R. Bendlin and I. Damgård. Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In *TCC*, pages 201–218. 2010.

[BF11a] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168. 2011.

[BF11b] D. Boneh and D. M. Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *Public Key Cryptography*, pages 1–16. 2011.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10. 1988.

[Can00] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. `http://eprint.iacr.org/`.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. 2001.

[CG99] R. Canetti and S. Goldwasser. An efficient *threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In *EUROCRYPT*, pages 90–106. 1999.

[CHKP10] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552. 2010.

[CLRS10] P.-L. Cayrel, R. Lindner, M. Rückert, and R. Silva. A lattice-based threshold ring signature scheme. In *Proceedings of the First international conference on Progress in cryptology: cryptology and information security in Latin America*, LATINCRYPT'10, pages 255–272. Springer-Verlag, Berlin, Heidelberg, 2010. ISBN 3-642-14711-9, 978-3-642-14711-1.

[CR03] R. Canetti and T. Rabin. Universal composition with joint state. In *CRYPTO*, pages 265–281. 2003.

[CS98] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO*, pages 13–25. 1998.

[DF89] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO*, pages 307–315. 1989.

[DF94] Y. Desmedt and Y. Frankel. Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM J. Discrete Math.*, 7(4):667–679, 1994.

[Feh98] S. Fehr. *Span Programs over Rings and How to Share a Secret from a Module*. Master's thesis, ETH Zurich, Institute for Theoretical Computer Science, 1998.

[FGM10] T. Feng, Y. Gao, and J. Ma. Changeable threshold signature scheme based on lattice theory. *International Conference on E-Business and E-Government*, 0:1311–1315, 2010. doi:http://doi.ieeecomputersociety.org/10.1109/ICEE.2010.335.

[GKV10] S. D. Gordon, J. Katz, and V. Vaikuntanathan. A group signature scheme from lattice assumptions. In *ASIACRYPT*, pages 395–412. 2010.

[GPV08] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206. 2008.

[Kle00] P. N. Klein. Finding the closest lattice vector when it's unusually close. In *SODA*, pages 937–941. 2000.

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

[MR04] D. Micciancio and O. Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[MSs11] S. Myers, M. Sergi, and a. shelat. Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive, Report 2011/454, 2011. `http://eprint.iacr.org/`.

[NR06] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. *J. Cryptology*, 22(2):139–160, 2009. Preliminary version in Eurocrypt 2006.

[Pei09] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342. 2009.

[Pei10] C. Peikert. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO*, pages 80–97. 2010.

[PW08] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In *STOC*, pages 187–196. 2008.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *STOC*, pages 73–85. 1989.

[Reg05] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005.

[Sha79] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

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

[Sho00] V. Shoup. Practical threshold signatures. In *EUROCRYPT*, pages 207–220. 2000.

[XXZ11] X. Xie, R. Xue, and R. Zhang. Efficient threshold encryption from lossy trapdoor functions. In *PQCrypto*, pages 163–178. 2011.

A Protocols Without Trusted Setup

Here we show how to realize threshold key generation $\mathcal{F}_{\text{KG}}$ and discrete Gaussian sampling $\mathcal{F}_{\text{GS}}$ without relying on any trusted setup. Given the protocol in Section 3.3, for $\mathcal{F}_{\text{GS}}$ it suffices to realize $\mathcal{F}_{\text{Correct}}$ and $\mathcal{F}_{\text{Perturb}}$. The rest of the section is organized as follows:

- In Section A.1 we formalize three low-level utility functionalities $\mathcal{F}_{\text{Blind}}$, $\mathcal{F}_{\text{Mult}}$, and $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ used by several of our protocols, and we describe how these functionalities are realized.
- In Section A.2 we give a protocol realizing $\mathcal{F}_{\text{KG}}$ using the utility functionalities. This simple protocol and security analysis are representative of the techniques we use (in more complex ways) in later protocols as well.
- In Section A.3 we give a realization of $\mathcal{F}_{\text{Correct}}$ that uses an additional utility functionality $\mathcal{F}_{\text{Gadget}}$, which we define and realize there.
- Finally, in Section A.4 we realize $\mathcal{F}_{\text{Perturb}}$ using a simple extension of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$, which we also realize there.

A.1 Utility Functionalities

We first present the low-level utility functionalities.

Blinding. The blinding functionality $\mathcal{F}_{\text{Blind}}$ (Figure 11) simply accepts shares of some value over an arbitrary additive group G, and distributes fresh shares of the same value. Our later protocols will use blinding and the homomorphic properties of secret sharing to reveal the values of shared secrets modulo lattices, and nothing more.

Realizations of $\mathcal{F}_{\text{Blind}}$ in various communication models are standard. For example, to realize it against semi-honest corruptions with private channels is very simple: simply add sufficiently many player-generated sharings of 0 to the original shares. For malicious corruptions one can use, e.g., subprotocols of the BGW [BGW88, ALR11] or RB [RB89] protocols, which run in a constant number of rounds. We leave the

Functionality $\mathcal{F}_{\text{Blind}}$

Blind: Upon receiving $(\mathsf{blind}, sid, [\![x]\!]^i \in G)$ from at least h honest parties in $\mathcal{P}$:

- Reconstruct x and generate a fresh sharing $[\![y]\!]$ (over G) of $y = x$.
- Send $(\mathsf{blind}, sid, [\![y]\!]^i)$ to each party i in $\mathcal{P}$, and (blind, sid, G) to the adversary.

Figure 11: Blinding functionality

implementation of $\mathcal{F}_{\text{Blind}}$ unspecified and simply work in the $\mathcal{F}_{\text{Blind}}$-hybrid model where needed. We remark that our protocols use $\mathcal{F}_{\text{Blind}}$ only during initialization, so the interaction required to implement it is limited to the offline phase.

Multiplication. The multiplication functionality $\mathcal{F}_{\text{Mult}}$ (Figure 12) takes shares of two values x, y in a ring $\mathbb{Z}_{q^d}$ and returns fresh shares of their product $x \cdot y$ (modulo q^d) to the respective parties. By the homomorphic properties of secret sharing, this generalizes immediately (via local computation alone) to products of vectors and/or matrices $\mathbf{X}, \mathbf{Y}$, so we write the functionality to support this more general capability. To realize $\mathcal{F}_{\text{Mult}}$ one can use any statistically secure protocol, such as the constant-round protocols of [BGW88, RB89, ALR11]; we leave this choice unspecified and simply work in the $\mathcal{F}_{\text{Mult}}$-hybrid model where needed.

Functionality $\mathcal{F}_{\text{Mult}}$

Multiply: Upon receiving $(\mathsf{mult}, sid, [\![\mathbf{X}]\!]^i \in \mathbb{Z}_{q^d}^{h\times\ell}, [\![\mathbf{Y}]\!]^i \in \mathbb{Z}_{q^d}^{\ell\times w})$ from at least h honest parties i in $\mathcal{P}$:

- Reconstruct $\mathbf{X}$, $\mathbf{Y}$ from the shares $[\![\mathbf{X}]\!]^i$, $[\![\mathbf{Y}]\!]^i$, respectively.
- Generate a fresh sharing $[\![\mathbf{Z}]\!]$ of $\mathbf{Z} = \mathbf{X} \cdot \mathbf{Y} \in \mathbb{Z}_{q^d}^{h\times w}$.
- Send $(\mathsf{mult}, sid, [\![\mathbf{Z}]\!]^i)$ to each party i in $\mathcal{P}$, and $(\mathsf{mult}, sid, h \times \ell, \ell \times w, d)$ to the adversary.

Figure 12: Multiplication functionality

Sampling integers. Several of our protocols rely on a low-level functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ (Figure 13) for sampling discrete Gaussians over the integers $\mathbb{Z}$. At a high level, the sample command produces shares of a discrete Gaussian variable $x \in \mathbb{Z}$ with a given parameter, where the sharing is over the additive group $\mathbb{Z}_{q^d}$, (i.e., with d digits of precision), and distributes these shares $[\![x]\!]^i$ to the respective parties. Later on in Section A.4.1 we will extend $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ with some additional commands, but for now we only need the sample command.

Functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$

Sample: Upon receiving $(\mathsf{sample}, sid, h \times w, z, d)$ from at least h honest parties in $\mathcal{P}$:

- Sample $\mathbf{X} \leftarrow D_{\mathbb{Z}, z\cdot\omega_n}^{h\times w}$ and generate a fresh sharing $[\![\mathbf{X}]\!]$ over $\mathbb{Z}_{q^d}$.
- Send $(\mathsf{sample}, sid, [\![\mathbf{X}]\!]^i)$ to each party i in $\mathcal{P}$ and $(\mathsf{sample}, sid, h \times w, z, d)$ to the adversary.

Figure 13: Simplified integer sampling functionality

Because we do not know of any highly efficient algorithms for sampling discrete Gaussians, our realization uses the general "inverse transform" sampling algorithm, and implements it securely using multiparty

computation tools. Recall that inverse sampling involves a close approximation of the cumulative distribution function. To sample, one chooses a uniformly random $u \in [0, 1)$ and looks up the corresponding output value in the table. An arithmetic circuit implementing this algorithm can be written as an AND of several interval tests on the input u, so the depth of the circuit is roughly the precision (number of digits) of the entries in the lookup table, and the width of the circuit is roughly the number of entries in the table. (Other trade-offs between depth and width are possible as well.)

Importantly, we can implement the inverse sampling method for discrete Gaussians using a table of size proportional to $q = \mathrm{poly}(n)$ for very large parameters z, even though the distribution has support size proportional to z. This is because for $z \in [q^j, q^{j+1})$, the discrete Gaussian of parameter z can be decomposed using Lemma 2.5 as a convolution of j discrete Gaussians over $q^j\mathbb{Z}, q^{j-1}\mathbb{Z}, \ldots, \mathbb{Z}$ having respective parameters roughly $z, z/q, \ldots, z/q^j$. (Note that each parameter can be chosen to be larger than the smoothing parameter of the respective lattice.) Each of these distributions is highly concentrated on only $\mathrm{poly}(n)$ outputs.

Finally, we emphasize that our higher-level protocols use $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ only in their key-generation or initialization phases, and only with fixed, public Gaussian parameters, so any inefficiencies in a realization of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ are limited to the offline phase.

A.2 Realizing $\mathcal{F}_{\text{KG}}$

The protocol π_{KG} (Figure 14) realizing $\mathcal{F}_{\text{KG}}$ in the $(\mathcal{F}_{\text{Blind}}, \mathcal{F}_{\text{Samp}\mathbb{Z}})$-hybrid model is straightforward, given the homomorphic properties of the secret-sharing scheme and the simple operation of the standalone trapdoor generator, which just multiplies a public uniform matrix $\bar{\mathbf{A}}$ with a secret Gaussian-distributed matrix $\mathbf{R}$. The parties first get shares of a Gaussian-distributed trapdoor $\mathbf{R}$ using $\mathcal{F}_{\text{Samp}\mathbb{Z}}$, then announce *blinded* shares of $\mathbf{A}_1 = -\bar{\mathbf{A}}\mathbf{R} \bmod q$ and reconstruct $\mathbf{A}_1$ to determine the public key $\mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{A}_1]$. The blinding is needed so that the announced shares reveal only $\mathbf{A}_1$, and nothing more about the honest parties' shares $[\![\mathbf{R}]\!]^i$ themselves.

Protocol π_{KG} in the $(\mathcal{F}_{\text{Blind}}, \mathcal{F}_{\text{Samp}\mathbb{Z}})$-hybrid model

Generate: On input $(\mathsf{gen}, sid, \bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}, \mathbf{H}^* \in \mathbb{Z}_q^{n \times n}, z)$, party i does:

- Call $\mathcal{F}_{\text{Samp}\mathbb{Z}}(\mathsf{sample}, sid, \bar{m} \times nk, z, 1)$ and receive $(\mathsf{sample}, sid, [\![\mathbf{R}]\!]^i)$.
- Call $\mathcal{F}_{\text{Blind}}(\mathsf{blind}, sid, -\bar{\mathbf{A}}[\![\mathbf{R}]\!]^i)$ and receive $(\mathsf{blind}, sid, [\![\mathbf{A}_1]\!]^i)$.
- Broadcast $[\![\mathbf{A}_1]\!]^i$ and reconstruct $\mathbf{A}_1 = -\bar{\mathbf{A}}\mathbf{R}$ from the announced shares.
- Output $(\mathsf{gen}, sid, \mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}^* \cdot \mathbf{G} + \mathbf{A}_1], [\![\mathbf{R}]\!]^i)$.

Figure 14: Key generation protocol

A simulator $\mathcal{S}_{\text{KG}}$ for demonstrating the security of π_{KG} is provided in Figure 15. Essentially, security boils down to the fact that the announced blinded shares $-\bar{\mathbf{A}}[\![\mathbf{R}]\!]^i$ in the protocol π_{KG} form a uniformly random, and independent of the honest parties' outputs $[\![\mathbf{R}]\!]^i$, sharing of $\mathbf{A}_1 = -\bar{\mathbf{A}}\mathbf{R}$, which is exactly what $\mathcal{S}_{\text{KG}}$ constructs to simulate the broadcast messages. A full proof is a straightforward application of this observation, and of the privacy, robustness, and homomorphic properties of the secret-sharing scheme, so we omit it.

Theorem A.1. *Protocol π_{KG} statistically realizes $\mathcal{F}_{KG}$ in the $(\mathcal{F}_{Blind}, \mathcal{F}_{Samp\mathbb{Z}})$-hybrid model for t-limited environments in $\mathscr{Z}$.*

Simulator $\mathcal{S}_{\text{KG}}$

Generate: Upon receiving $(\mathsf{gen}, sid, \mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}^* \cdot \mathbf{G} + \mathbf{A}_1], \mathbf{H}^*, z)$ from $\mathcal{F}_{\text{KG}}$:

- Generate a fresh sharing of $\mathbf{A}_1$ over $\mathbb{Z}_q$.
- For all currently corrupted parties i, and whenever $\mathcal{Z}$ later requests to corrupt a party i, receive the share $[\![\mathbf{R}]\!]^i$ from $\mathcal{F}_{\text{KG}}$, and reveal to $\mathcal{Z}$ the following functionality outputs to party i:
  - $(\mathsf{sample}, sid, [\![\mathbf{R}]\!]^i)$ on behalf of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$;
  - $(\mathsf{blind}, sid, [\![\mathbf{A}_1]\!]^i)$ on behalf of $\mathcal{F}_{\text{Blind}}$.
- Broadcast $[\![\mathbf{A}_1]\!]^i$ on behalf of each honest party i.

Figure 15: Simulator for π_{KG}

A.3 Realizing $\mathcal{F}_{\text{Correct}}$

Recall that $\mathcal{F}_{\text{Correct}}$ samples and distributes shares of a (non-spherical Gaussian) vector $\mathbf{y}$ from a desired coset of $\Lambda^\perp(\mathbf{A})$. Section 3.2.2 describes how to realize $\mathcal{F}_{\text{Correct}}$ with trusted setup, partly by precomputing shares of samples from each coset of $\Lambda^\perp(\mathbf{g}^t)$. We next describe how these shares can be obtained from a utility functionality called $\mathcal{F}_{\text{Gadget}}$ (Figure 16), and how it can easily be realized.

A.3.1 Gadget Functionality

The functionality $\mathcal{F}_{\text{Gadget}}$ (Figure 16) relates to the special gadget vector $\mathbf{g}$ and lattice $\Lambda^\perp(\mathbf{g}^t)$, as defined in [MP12] and reviewed in Section 3.1. Recall that we have a fixed public vector $\mathbf{g}^t = [1, 2, 4, \ldots, 2^{k-1}] \in \mathbb{Z}_q^{1\times k}$ for $k = \lceil \lg q \rceil$, which defines a full-rank lattice $\Lambda^\perp(\mathbf{g}^t) \subset \mathbb{Z}^k$ of determinant q whose smoothing parameter is bounded by $s_{\mathbf{g}} \cdot \omega_n$, where $s_{\mathbf{g}} \le \sqrt{5}$ is a known constant. The functionality generates shares of a discrete Gaussian over the coset $\Lambda_v^\perp(\mathbf{g}^t)$ for any desired $v \in \mathbb{Z}_q$, by running any of the efficient algorithms described in [MP12].

Functionality $\mathcal{F}_{\text{Gadget}}$

Sample coset: Upon receiving $(\mathsf{cosetsample}, sid, v \in \mathbb{Z}_q)$ from at least h honest parties in $\mathcal{P}$:

- Sample $\mathbf{z} \leftarrow D_{\Lambda_v^\perp(\mathbf{g}^t), s_{\mathbf{g}}\cdot\omega_n}$ and generate a uniformly random sharing $[\![\mathbf{z}]\!]$.
- Send $(\mathsf{cosetsample}, sid, [\![\mathbf{z}]\!]^i)$ to each party i in $\mathcal{P}$, and send $(\mathsf{cosetsample}, sid, v)$ to the adversary.

Figure 16: Functionality for operations related to the gadget lattice $\Lambda^\perp(\mathbf{g}^t)$

Realizing $\mathcal{F}_{\text{Gadget}}$ is straightforward in the $(\mathcal{F}_{\text{Samp}\mathbb{Z}}, \mathcal{F}_{\text{Blind}})$-hybrid model, using the homomorphic properties of secret sharing: essentially, the parties request shares of a Gaussian-distributed $\mathbf{z} \in \mathbb{Z}^k$ from $\mathcal{F}_{\text{Samp}\mathbb{Z}}$, then broadcast blinded shares of the syndrome $u = \langle \mathbf{g}, \mathbf{z} \rangle \bmod q$ and recover u, repeating until $u = v$. (The blinding is needed so that nothing more than the syndrome is revealed about $\mathbf{z}$.) Implemented naively as in Figure 17, the expected number of trials (which may be performed in parallel) is almost exactly $q = \text{poly}(n)$, because the syndrome u is negligibly far from uniform since $\mathbf{z}$'s Gaussian parameter is at least the smoothing parameter of $\Lambda^\perp(\mathbf{g}^t)$. Alternatively, shares of samples having the wrong syndrome can be stored away and used as needed later on. Note that in any case, $\mathcal{F}_{\text{Gadget}}$ is only ever called in the offline phase of π_{Correct} (Figure 19), so efficiency is not a top priority here.

A simulator $\mathcal{S}_{\text{Gadget}}$ for demonstrating the security of our protocol is provided in Figure 18. For a t-limited

Protocol π_{Gadget} in the $(\mathcal{F}_{\text{Samp}\mathbb{Z}}, \mathcal{F}_{\text{Blind}})$-hybrid model

Sample coset: On input $(\mathsf{cosetsample}, sid, v \in \mathbb{Z}_q)$, party i does:

- Call $\mathcal{F}_{\text{Samp}\mathbb{Z}}(\mathsf{sample}, sid, k \times 1, s_{\mathbf{g}}, 1)$ and receive $(\mathsf{sample}, sid, [\![\mathbf{z}]\!]^i)$.
- Call $\mathcal{F}_{\text{Blind}}(\mathsf{blind}, sid, \langle \mathbf{g}, [\![\mathbf{z}]\!]^i \rangle \bmod q)$ and receive $(\mathsf{blind}, sid, [\![u]\!]^i)$.
- Broadcast $[\![u]\!]^i$ and reconstruct $u = \langle \mathbf{g}, \mathbf{z} \rangle$ from the broadcast shares.
- If $u = v$, output $(\mathsf{cosetsample}, sid, [\![\mathbf{z}]\!]^i)$. Otherwise, repeat.

Figure 17: Protocol for gadget operations

Simulator $\mathcal{S}_{\text{Gadget}}$

Sample coset: Upon receiving $(\mathsf{cosetsample}, sid, v)$ from $\mathcal{F}_{\text{Gadget}}$:

- Choose a uniformly random $u \in \mathbb{Z}_q$, and generate fresh sharings of u and of $\mathbf{z} = \mathbf{0} \in \mathbb{Z}_q^k$.
- For each currently corrupted party i, reveal to $\mathcal{Z}$ the following functionality outputs to party i:
  - $(\mathsf{sample}, sid, [\![\mathbf{z}]\!]^i)$ on behalf of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$;
  - $(\mathsf{blind}, sid, [\![u]\!]^i)$ on behalf of $\mathcal{F}_{\text{Blind}}$.
- Broadcast $[\![u]\!]^i$ on behalf of all honest parties i.
- Unless $u = v$, repeat.

Corruption: When $\mathcal{Z}$ requests to corrupt party i, for each previous call to $\mathsf{cosetsample}$, reveal the corresponding messages $(\mathsf{sample}, sid, [\![\mathbf{z}]\!]^i)$ and $(\mathsf{blind}, sid, [\![u]\!]^i)$ to party i on behalf of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ and $\mathcal{F}_{\text{Blind}}$, respectively.

Figure 18: Simulator for π_{Gadget}

adversary, the value of $\mathbf{z} = \mathbf{0}$ and honest parties' shares remain information theoretically hidden. The announced (blinded) shares of u in the protocol π_{Gadget} form a uniformly random (and independent of the honest parties' outputs $[\![\mathbf{z}]\!]^i$) sharing of the (nearly) uniformly random syndrome $u = \langle \mathbf{g}, \mathbf{z} \rangle$, which is exactly what $\mathcal{S}_{\text{Gadget}}$ constructs to simulate the broadcast messages. A full proof is a straightforward application of these observations and of the privacy, robustness, and homomorphic properties of secret sharing.

A.3.2 Protocol and Security Analysis

The protocol π_{Correct} in the $(\mathcal{F}_{\text{Mult}}, \mathcal{F}_{\text{Gadget}})$-hybrid model is defined formally in Figure 19. In the initialization step, for each $j \in [n]$ and $v \in \mathbb{Z}_q$ the parties populate each of their local queues $Q_{j,v}$ with at least B entries, in the following way: each party i uses $\mathcal{F}_{\text{Gadget}}$, its shares of the trapdoor $\mathbf{R}$, and $\mathcal{F}_{\text{Mult}}$ to obtain a share of $\mathbf{y}_{j,v} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right](\mathbf{e}_j \otimes \mathbf{z}_{j,v})$ for Gaussian-distributed $\mathbf{z}_{j,v} \in \Lambda_v^\perp(\mathbf{g}^t)$, and places the share in a queue $Q_{j,v}$. (Regarding the arguments to the call to $\mathcal{F}_{\text{Mult}}$, note that $\left[\begin{smallmatrix}[\![\mathbf{R}]\!]^i\\ \mathbf{I}\end{smallmatrix}\right]$ is a valid ith share of $\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]$, via a constant sharing polynomial for $\mathbf{I}$, and $\mathbf{e}_{j,v} \otimes [\![\mathbf{z}_j]\!]^i$ is similarly a valid ith share of $\mathbf{e}_j \otimes \mathbf{z}_{j,v}$.) To later answer a correct request for syndrome $\mathbf{v} \in \mathbb{Z}_q^n$, each party just draws a share from each of $Q_{1,v_1}, \ldots, Q_{n,v_n}$ and sums these shares. By the homomorphic properties of secret sharing, this yields a share of $\mathbf{y} = \sum_{j \in [n]} \mathbf{y}_{j,v_j} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\mathbf{z}$ for Gaussian $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_n) \in \Lambda_{\mathbf{v}}^\perp(\mathbf{G})$, as desired.

Theorem A.2. *Protocol π_{Correct} statistically realizes $\mathcal{F}_{\text{Correct}}$ in the $(\mathcal{F}_{\text{Mult}}, \mathcal{F}_{\text{Gadget}})$-hybrid model for t-limited environments in $\mathscr{Z}$.*

Protocol π_{Correct} in the $(\mathcal{F}_{\text{Mult}}, \mathcal{F}_{\text{Gadget}})$-hybrid model

Initialize: On input $(\mathsf{init}, sid, [\![\mathbf{R}]\!]^i, B)$, party i does:

- Locally store $(sid, [\![\mathbf{R}]\!]^i)$ and initialize local queues $Q_{j,v}$ for each $j \in [n]$ and $v \in \mathbb{Z}_q$.
- For each $j \in [n]$, while there exists some $v \in \mathbb{Z}_q$ such that $Q_{j,v}$ has fewer than B entries:
  - Call $\mathcal{F}_{\text{Gadget}}(\mathsf{cosetsample}, sid, v)$ and receive $(\mathsf{cosetsample}, sid, [\![\mathbf{z}_{j,v}]\!]^i)$.
  - Call $\mathcal{F}_{\text{Mult}}(\mathsf{mult}, sid, \begin{bmatrix} [\![\mathbf{R}]\!]^i \\ \mathbf{I} \end{bmatrix}, \mathbf{e}_j \otimes [\![\mathbf{z}_{j,v}]\!]^i)$ and receive $(\mathsf{mult}, sid, [\![\mathbf{y}_{j,v}]\!]^i)$, where $\mathbf{y}_{j,v} = \begin{bmatrix} \mathbf{R} \\ \mathbf{I} \end{bmatrix} (\mathbf{e}_j \otimes \mathbf{z}_{j,v})$.
  - Place $[\![\mathbf{y}_{j,v}]\!]^i$ in local queue $Q_{j,v}$.
- Output (init, sid).

Correct: On input $(\mathsf{correct}, sid, \mathbf{v})$, if fewer than B calls to correct have already been made, party i does:

- For each $j \in [n]$, dequeue an entry $[\![\mathbf{y}_{j,v}]\!]^i$ from Q_{j,v_j}.
- Locally compute $[\![\mathbf{y}]\!]^i = \sum_{j \in [n]} [\![\mathbf{y}_{j,v}]\!]^i$.
- Output $(\mathsf{correct}, sid, [\![\mathbf{y}]\!]^i)$.

Figure 19: Syndrome correction protocol

Simulator $\mathcal{S}_{\text{Correct}}$

Initialize: Upon receiving (init, sid, B) from $\mathcal{F}_{\text{Correct}}$:

- Initialize empty lists $Q_{j,v}$ for each $j \in [n]$ and $v \in \mathbb{Z}_q$.
- For each $j \in [n]$, while there exists some $v \in \mathbb{Z}_q$ such that $Q_{j,v}$ has fewer than B unused entries:
  - Generate a fresh sharing $[\![\mathbf{z}_{j,v}]\!]$ of $\mathbf{z}_{j,v} = \mathbf{0} \in \mathbb{Z}_q^k$, and send $(\mathsf{cosetsample}, sid, [\![\mathbf{z}_{j,v}]\!]^i)$ on behalf of $\mathcal{F}_{\text{Gadget}}$ to each currently corrupted party i in $\mathcal{P}$.
  - Generate a fresh sharing $[\![\mathbf{y}_{j,v}]\!]$ for $\mathbf{y}_{j,v} = \mathbf{0} \in \mathbb{Z}_q^m$, and send $(\mathsf{mult}, sid, [\![\mathbf{y}_{j,v}]\!]^i)$ on behalf of $\mathcal{F}_{\text{Mult}}$ to each currently corrupted i in $\mathcal{P}$.
  - Store $([\![\mathbf{z}_{j,v}]\!], [\![\mathbf{y}_{j,v}]\!])$ as an unused entry at the end of list $Q_{j,v}$.

Correct: Upon receiving $(\mathsf{correct}, sid, \mathbf{v})$ from $\mathcal{F}_{\text{Correct}}$:

- For each $j \in [n]$, look up the next unused entry $([\![\mathbf{z}_{j,v_j}]\!], [\![\mathbf{y}_{j,v_j}]\!])$ from Q_{j,v_j}, and mark it as used for this call to correct. For each currently corrupted party i in $\mathcal{P}$, send $[\![\mathbf{y}]\!]^i = \sum_{j \in [n]} [\![\mathbf{y}_{j,v}]\!]^i$ to $\mathcal{F}_{\text{Correct}}$ as the desired share for party i.

Corruption: When $\mathcal{Z}$ requests to corrupt party i,

- Receive party i's share $[\![\mathbf{y}]\!]^i$ for each previous call of the form $(\mathsf{correct}, sid, \mathbf{v})$. Look up the n corresponding (used) entries $([\![\mathbf{z}_{j,v_j}]\!], [\![\mathbf{y}_{j,v_j}]\!])$ in Q_{j,v_j}, and update the value $[\![\mathbf{y}_{n,v_n}]\!]^i$ so that $[\![\mathbf{y}]\!]^i = \sum_{j \in [n]} [\![\mathbf{y}_{j,v_j}]\!]^i$.
- For all entries $([\![\mathbf{z}_{j,v}]\!], [\![\mathbf{y}_{j,v}]\!])$, both used and unused, in each list $Q_{j,v}$, reveal to $\mathcal{Z}$ the messages $(\mathsf{cosetsample}, sid, [\![\mathbf{z}_{j,v}]\!]^i)$ and $(\mathsf{mult}, sid, [\![\mathbf{y}_{j,v}]\!]^i)$ to party i on behalf of $\mathcal{F}_{\text{Gadget}}$ and $\mathcal{F}_{\text{Mult}}$, respectively.

Figure 20: Simulator for π_{Correct}

Proof sketch. A simulator $\mathcal{S}_{\text{Correct}}$ for demonstrating the security of π_{Correct} is provided in Figure 20. The only subtlety lies in the fact that outputs from helper functionalities during precomputation must be simulated before knowing which parties will be corrupted when the corresponding correct calls are made later on. As mentioned in Section 3.2.2, this is why we designed $\mathcal{F}_{\text{Correct}}$ to ask the adversary for shares for the corrupted

parties: the simulator generates its own shares when simulating the precomputation, and provides them to the functionality upon request. Then security boils down to the fact that even though $\mathbf{y}_{j,v_j} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right](\mathbf{e}_j \otimes \mathbf{z}_{j,v_j})$, all shares of $\mathbf{z}_{j,v_j}$ (that the adversary sees) are uniform and independent of the corresponding shares of $\mathbf{y}_{j,v_j}$ since $\mathcal{F}_{\text{Mult}}$ blinds its output, so the queuing strategy employed by π_{Correct} indeed produces shares of $\mathbf{y}$ with the desired distribution. □

A.4 Realizing $\mathcal{F}_{\text{Perturb}}$

Recall that $\mathcal{F}_{\text{Perturb}}$ (Figure 3) distributed shares of of perturbations $\mathbf{p}$ drawn from the discrete Gaussian distribution $D_{\mathbb{Z}^m,\sqrt{\Sigma_{\mathbf{p}}}}$, where the covariance $\Sigma_{\mathbf{p}}$ depends on the trapdoor $\mathbf{R}$. In the standalone setting, this is straightforward: first generate a continuous Gaussian $\mathbf{p}' \in \mathbb{R}^m$ with covariance $\Sigma_{\mathbf{p}} - \mathbf{I} \cdot \omega_n^2$, then randomly round each coordinate of $\mathbf{p}'$ to a nearby integer (see [Pei10] for details). In the threshold setting, generating a good perturbation seems quite a bit more difficult, because neither $\mathbf{p}$ nor its covariance $\Sigma_{\mathbf{p}}$ can be revealed, since they leak the trapdoor $\mathbf{R}$. Fortunately, we can give a distributed protocol that emulates the standalone rounding procedure up to sufficient precision.

A.4.1 Extending $\mathcal{F}_{\text{Samp}\mathbb{Z}}$

We first extend the functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ (originally defined in Section A.1) with two additional commands, cosetsample and rround, which support randomized rounding of a shared value $x \in q^{-j}\mathbb{Z}$ to the integers $\mathbb{Z}$. Note that while cosetsample and rround are defined for (scalings of) the integer lattice $\mathbb{Z}$, the commands immediately generalize to vectors and matrices, component-wise. (This is simply because spherical Gaussians over cosets of $\mathbb{Z}^{h\times w}$ are just product distributions of Gaussians over cosets of $\mathbb{Z}$.) This extended functionality is defined formally in Figure 21.

Functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$

Sample: Upon receiving $(\mathsf{sample}, sid, h \times w, z, d)$ from at least h honest parties in $\mathcal{P}$:

- Sample $\mathbf{X} \leftarrow D^{h\times w}_{\mathbb{Z},z\cdot\omega_n}$ and generate a fresh sharing $[\![\mathbf{X}]\!]$ over $\mathbb{Z}_{q^d}$.
- Send $(\mathsf{sample}, sid, [\![\mathbf{X}]\!]^i)$ to each party i in $\mathcal{P}$ and $(\mathsf{sample}, sid, h \times w, z, d)$ to the adversary.

Sample coset: Upon receiving $(\mathsf{cosetsample}, sid, v \in q^{-j}\mathbb{Z}/q^{-j+1}\mathbb{Z}, z \geq q^{-j+1})$ from at least h honest parties in $\mathcal{P}$:

- Sample $x \leftarrow D_{q^{-j+1}\mathbb{Z}+v,z\cdot\omega_n}$ and let $c = x \bmod \mathbb{Z}$.
- Generate a fresh sharing $[\![x]\!]$ over $q^{-j}\mathbb{Z}/q\mathbb{Z}$.
- Send $(\mathsf{cosetsample}, sid, [\![x]\!]^i, c)$ to each party i in $\mathcal{P}$, and $(\mathsf{cosetsample}, sid, c, z)$ to the adversary.

Randomized round: Upon receiving $(\mathsf{rround}, sid, [\![x]\!]^i \in q^{-d}\mathbb{Z}/q\mathbb{Z})$ from at least h honest parties in $\mathcal{P}$:

- Reconstruct x and let $c = x \bmod \mathbb{Z}$. Sample an integer $z \leftarrow x + D_{\mathbb{Z}-c,d\cdot\omega_n}$.
- Generate a fresh sharing $[\![z]\!]$ over $\mathbb{Z}_q$, and send $(\mathsf{rround}, sid, [\![z]\!]^i, c)$ to each party i in $\mathcal{P}$, and $(\mathsf{rround}, sid, c)$ to the adversary.

Figure 21: Full integer sampling functionality (which replaces Figure 13)

A protocol $\pi_{\text{Samp}\mathbb{Z}}$ realizing $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ in the $\mathcal{F}_{\text{Blind}}$-hybrid model is given in Figure 22. We elaborate informally on the implementation of the two additional commands, noting that implementation of the sample command was discussed in Section A.1. We omit a formal security proof for $\pi_{\text{Samp}\mathbb{Z}}$, but we remark that while

the protocol is somewhat more complicated than the key-generation protocol of Section A.2, its security is based straightforwardly on the same main observations, plus Lemma 2.5.

Coset sampling. The cosetsample command (which exists mainly to support the randomized-rounding command, described next) generates a discrete Gaussian variable x with given parameter $z \geq q^{-j+1}$ over the (possibly very dense) lattice $q^{-j}\mathbb{Z}$, such that x's least significant base-q digit is a specified v. It then distributes shares $[\![x]\!]^i$ to the respective parties, along with the value $c = x \bmod \mathbb{Z}$, which also goes to the adversary.

Naïvely implementing cosetsample in our protocol is very simple: the parties just use sample to generate a Gaussian value $q^j x \in \mathbb{Z}$ with parameter $q^j z \cdot \omega_n$, shared over $\mathbb{Z}_{q^{j+1}}$, then reveal their blinded shares $x \bmod \mathbb{Z}$ to reconstruct c, repeating until the least-significant digit is v. Because $z \geq q^{-j+1}$ is large enough, x's least significant digit is nearly uniform by Corollary 2.3, and the expected number of trials is almost exactly $q = \mathrm{poly}(n)$. Of course, this procedure throws away many samples; a more efficient implementation would precompute many trials and store the results according to least-significant digit, so that the online phase of the command becomes just a noninteractive table lookup. For simplicity, we formally define only the naive implementation.

Randomized rounding. The rround command takes shares of a value $x \in q^{-d}\mathbb{Z}$ (represented modulo $q\mathbb{Z}$) and rounds it to an integer $z \in \mathbb{Z}$ using Gaussian rounding. It returns shares $[\![z]\!]^i$ to the respective parties, along with the coset $c = x \bmod \mathbb{Z}$ of the original input, which also goes to the adversary. In our protocol, the parties broadcast their blinded shares of $c = x \bmod \mathbb{Z}$ to reconstruct c, then use d calls to cosetsample to round x one digit at a time, from least- to most-significant digit. Note that each call to cosetsample alters the more-significant digits of $x \bmod \mathbb{Z}$, but these changes are public.

A.4.2 Protocol and Security Analysis

In brief, our perturbation protocol starts by generating a sharing of a sufficiently high-precision approximation $\mathbf{P} \approx \sqrt{\Sigma_{\mathbf{p}}}$ with some d digits of precision in its fractional part (i.e., the entries of $\mathbf{P}$ are in $q^{-d}\mathbb{Z}$). The sharing of $\mathbf{P}$ can be precomputed as part of the key-generation phase, using general multiparty computation. To generate a perturbation vector $\mathbf{p}$, the protocol first generates a sharing of a high-precision Gaussian random variable $\mathbf{p}' \in q^{-d}\mathbb{Z}^m$ having covariance $\mathbf{P}\mathbf{P}^t \approx \Sigma_{\mathbf{p}}$. It does this by invoking $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ to generate shares of a Gaussian-distributed $\mathbf{z} \in \mathbb{Z}^m$, and then invoking $\mathcal{F}_{\text{Mult}}$ to get a fresh sharing of $\mathbf{p}' = \mathbf{P}\mathbf{z}$. The parties then randomize-round their shared $\mathbf{p}' \in q^{-d}\mathbb{Z}^m$ to a shared final perturbation vector $\mathbf{p} \in \mathbb{Z}^m$, using the rround command of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$. (Recall from A.4.1 that this command reveals $\mathbf{c} = \mathbf{p}' \bmod \mathbb{Z}^m$ publicly.) Finally, using the secret-sharing homomorphisms the parties also reconstruct the two syndromes $\bar{\mathbf{w}}$ and $\bar{\mathbf{w}}$. (Recall that these are eventually needed by the full Gaussian sampling protocol π_{GS}.) Note that once the sharing of $\mathbf{P}$ is computed once and for all, the only trapdoor-dependent work is the relatively efficient call to $\mathcal{F}_{\text{Mult}}$.

For analyzing security, the essence of the argument is that the public residue $\mathbf{c} = \mathbf{p}' \bmod \mathbb{Z}^m$ returned by $\mathcal{F}_{\text{Samp}\mathbb{Z}}$.rround is (nearly) uniformly random, and hence simulatable without knowing $\mathbf{p}'$, because $\mathbf{p}'$ is drawn from a Gaussian whose parameter exceeds the smoothing parameter of $\mathbb{Z}^m$. All the remaining functionalities simply return independent and properly blinded shares of intermediate values to their respective owners, and so are trivial to simulate. Therefore, we omit a formal simulator and security proof for π_{Perturb}, which are tedious (though straightforward given the above intuition).

Protocol $\pi_{\text{Samp}\mathbb{Z}}$ in the $\mathcal{F}_{\text{Blind}}$-hybrid model

Sample: On input $(\mathsf{sample}, sid, h \times w, z, d)$, party i does:

- With the other parties, run an inverse sampling protocol (see A.1 for elaboration) to generate private output $[\![\mathbf{X}]\!]^i$, where $\mathbf{X} \leftarrow D^{h\times w}_{\mathbb{Z},z\cdot\omega_n}$.
- Output $(\mathsf{sample}, sid, [\![\mathbf{X}]\!]^i)$.

Sample coset: On input $(\mathsf{cosetsample}, sid, v \in q^{-j}\mathbb{Z}/q^{-j+1}\mathbb{Z}, z \geq q^{-j+1})$, party i does:

- Call $(\mathsf{sample}, sid, 1 \times 1, q^j z, j+1)$ and receive $(\mathsf{sample}, sid, q^j [\![x]\!]^i)$ with $x \in q^{-j}\mathbb{Z}$.
- Call $\mathcal{F}_{\text{Blind}}(\mathsf{blind}, sid, [\![x]\!]^i \bmod \mathbb{Z})$ and receive $(\mathsf{blind}, sid, [\![c]\!]^i)$.
- Announce $[\![c]\!]^i$ and reconstruct c from other parties' announced shares.
- If $c \bmod q^{-j+1} = v$, output $(\mathsf{cosetsample}, sid, [\![x]\!]^i)$. Otherwise, repeat.

Randomized round: On input $(\mathsf{rround}, sid, [\![x]\!]^i \in q^{-d}\mathbb{Z}/q\mathbb{Z})$, party i does:

- Call $\mathcal{F}_{\text{Blind}}(\mathsf{blind}, sid, [\![x]\!]^i \bmod \mathbb{Z})$ and receive a fresh share $[\![c]\!]^i$ of $c = x \bmod \mathbb{Z}$. Broadcast $[\![c]\!]^i$.
- Reconstruct $c \in q^{-d}\mathbb{Z}/\mathbb{Z}$ from the announced shares.
- Let $v = c$ and $[\![z]\!]^i = [\![x]\!]^i$. For $j = d, \ldots, 1$:
  - Call $(\mathsf{cosetsample}, sid, -v \bmod q^{-j+1}\mathbb{Z}, \sqrt{d})$ as a subroutine, and receive back $(\mathsf{cosetsample}, sid, [\![x']\!]^i \in q^{-j}\mathbb{Z}/q\mathbb{Z}, c' \in q^{-j}\mathbb{Z}/\mathbb{Z})$.
  - Let $v \leftarrow v + c' \in q^{-j+1}\mathbb{Z}/\mathbb{Z}$ and $[\![z]\!]^i \leftarrow [\![z]\!]^i + [\![x']\!]^i \in q^{-j}\mathbb{Z}/q\mathbb{Z}$, which is an ith share of $z + x' \in q^{-j+1}\mathbb{Z}/q\mathbb{Z}$.
  - Truncate $[\![z]\!]^i$ to lie in $q^{-j+1}\mathbb{Z}/q\mathbb{Z}$ (without changing the underlying shared value) as described in Section 2.3, i.e., let $[\![z]\!]^i \leftarrow [\![z]\!]^i - ([\![z]\!]^i \bmod q^{-j+1})$.
- Output $(\mathsf{rround}, sid, [\![z]\!]^i, c)$.

Figure 22: Integer sampling protocol

Protocol π_{Perturb} in the $(\mathcal{F}_{\text{Blind}}, \mathcal{F}_{\text{Mult}}, \mathcal{F}_{\text{Samp}\mathbb{Z}})$-hybrid model

In what follows, define the quotient ring $G_d = q^{-d}\mathbb{Z}/q\mathbb{Z}$.

Initialize: On input $(\mathsf{init}, sid, \mathbf{A}_{-\mathbf{H}^*}, [\![\mathbf{R}]\!]^i, s, B)$, party i does:

- With the other parties, run a statistically secure multiparty computation protocol to compute (as a private output) $[\![\mathbf{P}]\!]^i$ shared over G_d, where $\Sigma_{\mathbf{p}} = s^2 - s_{\mathbf{g}}^2 \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\left[\begin{smallmatrix}\mathbf{R}^t & \mathbf{I}\end{smallmatrix}\right]$, and $\mathbf{P} \approx \sqrt{\Sigma_{\mathbf{p}} - d^2 \cdot \omega_n^2}$
- Locally store sid, $\mathbf{A}_{-\mathbf{H}^*}$, and $[\![\mathbf{P}]\!]^i$, and initialize a local queue Q.
- While Q has fewer than B entries:
  - Call $\mathcal{F}_{\text{Samp}\mathbb{Z}}(\mathsf{sample}, sid, m \times 1, 1, d+1)$ and receive $(\mathsf{sample}, sid, [\![\mathbf{z}]\!]^i)$ for some $\mathbf{z} \leftarrow D^m_{\mathbb{Z},\omega_n}$ that is shared over $\mathbb{Z}_{q^{d+1}}$.
  - Call $\mathcal{F}_{\text{Mult}}(\mathsf{mult}, sid, q^d \cdot [\![\mathbf{P}]\!]^i, [\![\mathbf{z}]\!]^i)$ and receive $(\mathsf{mult}, sid, q^d \cdot [\![\mathbf{p}']\!]^i)$ where $\mathbf{p}' = \mathbf{P}\mathbf{z} \in G_d^m$. (Above we are multiplying and dividing shares by q^d simply to compute an isomorphism between G_d and $\mathbb{Z}_{q^{d+1}}$, because $\mathcal{F}_{\text{Mult}}$ expects to receive and return shares over the latter ring.)
  - Call $\mathcal{F}_{\text{Samp}\mathbb{Z}}(\mathsf{rround}, sid, [\![\mathbf{p}']\!]^i)$ and receive $(\mathsf{rround}, sid, [\![\mathbf{p}]\!]^i \in \mathbb{Z}_q^m, \mathbf{c} = \mathbf{p}' \bmod \mathbb{Z}^m)$, where $\mathbf{p}$ has distribution $\mathbf{p} + D_{\mathbb{Z}^m - \mathbf{p}', d\cdot\omega_n} \in \mathbb{Z}^m$.
  - Call $\mathcal{F}_{\text{Blind}}(\mathsf{blind}, sid, [\mathbf{0} \mid \mathbf{G}] \cdot [\![\mathbf{p}]\!]^i)$ and $\mathcal{F}_{\text{Blind}}(\mathsf{blind}, sid, \mathbf{A}_{-\mathbf{H}^*} \cdot [\![\mathbf{p}]\!]^i)$ and receive back $[\![\mathbf{w}]\!]^i$ and $[\![\bar{\mathbf{w}}]\!]^i$, respectively. Broadcast these shares.
  - Reconstruct $\mathbf{w}$ and $\bar{\mathbf{w}}$ from the announced shares, and put $([\![\mathbf{p}]\!]^i, \bar{\mathbf{w}}, \mathbf{w})$ in local queue Q.
- Output (init, sid).

Perturb: On input $(\mathsf{perturb}, sid)$, if fewer than B calls to $\mathsf{perturb}$ have already been made, party i does:

- Dequeue $([\![\mathbf{p}]\!]^i, \mathbf{w}, \bar{\mathbf{w}})$ from local queue Q.
- Output $(\mathsf{perturb}, sid, [\![\mathbf{p}]\!]^i, \mathbf{w}, \bar{\mathbf{w}})$.

Figure 23: Perturbation protocol