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

Rikke Bendlin[1,⋆], Sara Krehbiel[2], and Chris Peikert[2,⋆⋆]

[1] Department of Computer Science, Aarhus University
[2] School of Computer Science, Georgia Institute of Technology

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 robust against up to $\ell/2$ semi-honest or $\ell/3$ malicious parties (out of ℓ total). 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.

1 Introduction

A *threshold* cryptographic scheme [18] is one that allows any quorum of h out of ℓ trustees to jointly perform some privileged operation(s), but remains correct and secure even if up to some $t < h$ of the parties behave 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.

⋆ 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.

⋆⋆ 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.

M. Jacobson et al. (Eds.): ACNS 2013, LNCS 7954, pp. 218–236, 2013.

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) a minimal amount of interaction—i.e., 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 [13] version of the CCA-secure Cramer-Shoup cryptosystem [17], and Shoup's [35] 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 [34] 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) [22], has resulted in secure lattice-based hash-and-sign signatures and (hierarchical) identity-based encryption schemes [15,1], along with many more applications (e.g., [23,10,9,2]). 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) [3,6], 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 [22]. 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 [30,25]. 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 [25], (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 [22,15,1,25] 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 the full version of this paper we show how this is (straightforwardly) done for the simplest of these applications, namely, the GPV signature and IBE schemes [22]; 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 [4,26] for digital signatures, and Learning With Errors [32] for identity-based encryption). We work in a version of the universal composability (UC) framework [12], specialized to the threshold setting, and as a result also get strong security guarantees for protocols under arbitrary composition.
- They work for an optimal threshold of $h = t + 1$ for semi-honest adversaries, and $h = 2t+1$ for active (malicious) adversaries. (Recall that h is the number of honest parties needed to successfully execute the protocol, and the robustness threshold t is an upper bound on the number of dishonest 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).

Regarding the final item, the trusted setup model is the one used by Canetti and Goldwasser [13] 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 [25], and the parallel and offline nature of recent standalone Gaussian sampling algorithms [30,25].[1] A key technical challenge is that the security of the

[1] In particular, it appears very difficult to implement, in a noninteractive threshold fashion, iterative sampling algorithms like those from [24,22] which use the classical trapdoor notion of a short basis.

sampling algorithms from [30,25] 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 1 for further details.

Open problems. In addition to simple, non-interactive protocols for discrete Gaussian sampling with trusted setup, the full version of this paper provides protocols that avoid both trusted setup and online interaction. These protocols are designed as follows: first, we give efficient protocols that use (offline) access to a functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$, which produces shares of Gaussian-distributed values over the integers $\mathbb{Z}$ (see Section 4 and the full version for details). Then, 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 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 [29,25], without going through a generic IBE-to-CCA transformation [8].

Related work in threshold lattice cryptography. A few works have considered lattice cryptography in the threshold setting. For encryption schemes, Bendlin and Damgård [7] gave a threshold version of Regev's CPA-secure encryption scheme based on the learning with errors (LWE) problem [32]. Related work by Myers *et al.* [27] described threshold decryption for fully homomorphic cryptosystems. Xie *et al.* [36] gave a threshold CCA-secure encryption scheme from any lossy trapdoor function (and hence from lattices/LWE [31]), though its public key and encryption runtime grow at least linearly with the number of trustees. For signatures, Feng *et al.* [21] 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 [28]. Cayrel *et al.* [16] 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

[2] We note that it is possible to give a threshold protocol using a combination of Gaussian sampling and trapdoor delegation [15,25], but it is obviously no simpler than Gaussian sampling alone.

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 recall 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 [25], present our functionalities for these algorithms in the threshold setting, and show how these functionalities can be implemented efficiently and noninteractively using trusted setup. We additionally provide a functionality and protocol for trapdoor delegation. In Section 4 we remove the trusted setup assumption and show how to implement the key generation functionality. Due to space restrictions, we refer to the full version for implementations of the Gaussian sampling functionalities using offline interaction instead of trusted setup. The full version also details a simple example application of our protocols, namely, a threshold version of the GPV signature scheme [22] realizing the threshold signature functionality of [5].

2 Preliminaries

We denote the reals by $\mathbb{R}$ and the integers by $\mathbb{Z}$. For a positive integer ℓ, we let $[\ell] = \{1, \ldots, \ell\}$. A symmetric real matrix Σ is *positive definite*, written $\Sigma > \mathbf{0}$, if $\mathbf{z}^t \Sigma \mathbf{z} > 0$ for all nonzero $\mathbf{z}$. Positive definiteness defines a partial ordering on real matrices: we say that $\mathbf{X} > \mathbf{Y}$ if $\mathbf{X} - \mathbf{Y} > \mathbf{0}$. We say that $\mathbf{X}$ is a *square root* of a positive definite matrix Σ, written $\mathbf{X} = \sqrt{\Sigma}$, if $\mathbf{X}\mathbf{X}^t = \Sigma$. 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 Lattices and 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 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)

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

We define the Gaussian function $\rho\colon \mathbb{R}^m \to (0, 1]$ as $\rho(\mathbf{x}) = \exp(-\pi\langle \mathbf{x}, \mathbf{x}\rangle) = \exp(-\pi\|\mathbf{x}\|^2)$. Generalizing to any nonsingular $\mathbf{B} \in \mathbb{R}^{m \times m}$, we define the Gaussian function with parameter $\mathbf{B}$ as

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

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 $\rho_{\sqrt{\Sigma}}$, and refer to Σ as its covariance matrix. For a lattice coset $\Lambda + \mathbf{c}$ and covariance matrix $\Sigma > \mathbf{0}$, the *discrete Gaussian distribution* $D_{\Lambda+\mathbf{c},\sqrt{\Sigma}}$ is defined to assign probability proportional to $\rho_{\sqrt{\Sigma}}(\mathbf{x})$ to each $\mathbf{x} \in \Lambda + \mathbf{c}$, and zero elsewhere. That is, $D_{\Lambda+\mathbf{c},\sqrt{\Sigma}}(\mathbf{x}) := \rho_{\sqrt{\Sigma}}(\mathbf{x})/\rho_{\sqrt{\Sigma}}(\Lambda + \mathbf{c})$. A discrete Gaussian is said to be *spherical* with parameter $s > 0$ if its covariance matrix is $s^2\mathbf{I}$.

In some of our proofs we use the notion of the *smoothing parameter* $\eta_\epsilon(\Lambda)$ of a lattice Λ [26], generalized to arbitrary covariances. For reasons associated with the smoothing parameter, throughout the paper we often attach a factor $\omega_n = \omega_n(n) = \omega(\sqrt{\log n})$ to Gaussian parameters $\sqrt{\Sigma}$ (or ω_n^2 to covariance matrices Σ), which represents an arbitrary fixed function that grows asymptotically faster than $\sqrt{\log n}$. In exposition we usually omit reference to these factors, but we always retain them where needed in formal expressions. The full version gives further background on lattices and Gaussians.

2.2 The GPV Schemes

As mentioned in the introduction, the two non-trivial algorithmic steps of many lattice-based cryptographic schemes are generating a lattice $\Lambda = \Lambda^\perp(\mathbf{A})$ together with a strong trapdoor $\mathbf{R}$, and sampling from discrete Gaussian distributions over a given coset of Λ. In Section 3, we give functionalities and protocols for these tasks in the threshold setting.

Here we briefly recall the well-known GPV signature scheme from [22], which uses these operations ($\mathsf{GenTrap}$ and $\mathsf{SampleD}$), and serves as an immediate application of the present work. The scheme is parametrized by a security parameter n, modulus q, and message space $\mathcal{M}$, and it uses a hash function $H\colon \mathcal{M} \to \mathbb{Z}_q^n$ which is modeled as a random oracle. At a high level, $\mathsf{GenTrap}(n, q, m)$ (for sufficiently large m) generates a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ with distribution statistically close to uniform, 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 parameter s) over the lattice coset $\Lambda_{\mathbf{u}}^\perp(\mathbf{A})$. 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 signature σ. Otherwise, let $\mathbf{x} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{R}, H(\mu), s)$, store (μ, σ), and output signature $\sigma = \mathbf{x}$.
- $\mathsf{Verify}(vk, \mu, \sigma = \mathbf{x})$: If $\mathbf{A}\mathbf{x} = H(\mathbf{m})$ and $\mathbf{x}$ is sufficiently short, then accept; otherwise, reject.

See [22] for the proof of (strong) unforgeability under worst-case lattice assumptions. Another immediate application of the present work is the identity-based encryption (IBE) scheme of [22], where vk and sk above are the master public and secret keys, respectively, and signatures on identities are the secret keys for individual identities.

2.3 Secret Sharing

In this work we need to distribute secret lattice vectors among ℓ players so that any sufficiently large number of players can reconstruct the secret, but no group of $t < \ell$ or

fewer players collectively get any 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 [33]. There is a rich theory of secret sharing for arbitrary additive groups and modules, e.g., [19,20]. We refer to the full version of this paper for secret sharing details, and simply note here that a variant of the Shamir secret sharing scheme has the desired properties.

Our notation is as follows: Let G be any finite abelian (additive) group. We denote player i's share of some value $v \in G$ by $[\![v]\!]^i$, and the tuple of all such shares by $[\![v]\!]$.

2.4 UC Framework

We frame our results in the Universal Composability (UC) framework [12,11]. 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 1. *We say that 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* $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \overset{s}{\approx} \mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$.

What makes this definition so strong and useful is the general *composition theorem* [12], which (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.

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 t-limited adversaries, which may adaptively corrupt a bounded number t of the parties over the entire lifetime of a protocol. Corruptions can occur before or after any invoked protocol/functionality command, but not during its execution. 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.

In the case of semi-honest corruptions, namely when corrupted parties reveal their protocol traffic to the adversary but always execute the protocol faithfully, we prove security for $t < |\mathcal{P}|/2$ and $h = t + 1$. In the case of malicious corruptions, namely when corrupted parties send messages on behalf of the adversary that are not necessarily consistent with the protocol, we prove security for $t < |\mathcal{P}|/3$ and $h = 2t + 1$. These parameters come directly from the secrecy and robustness guarantees of the secret sharing scheme described in Section 2.3.

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 those like hashing suggested in [13], which deals with similar synchronization issues.

3 Threshold KeyGen, Gaussian Sampling, and 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, we describe threshold variants of the GPV signature and IBE schemes [22] in the full version. Other signature and (H)IBE schemes (e.g., [15,1,25]) can be adapted similarly (where delegation is needed for HIBE).

In Section 3.1 we recall the recent standalone (non-threshold) key generation and discrete Gaussian sampling algorithms of [25], 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. We also define two lower-level "helper" functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$, and show how they can be realized noninteractively using either trusted setup or offline precomputation. In Section 3.3 we give an efficient noninteractive protocol that realizes $\mathcal{F}_{\text{GS}}$ using access to $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$. In Section 3.4 we give a functionality and protocol for trapdoor delegation.

Since key generation tends to be rare in applications, $\mathcal{F}_{\text{KG}}$ can be realized using trusted setup; alternatively, later in Section 4 we realize $\mathcal{F}_{\text{KG}}$ without trusted setup using some lower-level functionalities described there. We additionally realize $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$ with these and other lower-level functionalities in the full version of the paper.

3.1 Trapdoors and Standalone Algorithms

We recall the notion of a (strong) lattice trapdoor and associated algorithms recently introduced by Micciancio and Peikert [25]; 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 $\mathbf{G} := \mathbf{I}_n \otimes \mathbf{g}^t \in \mathbb{Z}_q^{n \times nk}$, the direct sum of n copies of $\mathbf{g}^t$. 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 $\Lambda^\perp_{\mathbf{u}}(\mathbf{G})$ (with parameter $s \geq s_{\mathbf{g}} \cdot \omega_n$) can be done by concatenating n independent samples over appropriate cosets of $\Lambda^\perp(\mathbf{g}^t)$.

Definition 2 ([25]). *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 $\mathbf{A}_{\mathbf{H}} := \mathbf{A} - [\mathbf{0} \mid \mathbf{HG}]$ with tag $\mathbf{H}^* - \mathbf{H} \in \mathbb{Z}_q^{n \times n}$.

The key-generation algorithm of [25] 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 [25] is an instance of the "convolution" approach from [30]. 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 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 $\mathbf{x} = \mathbf{p} + \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\mathbf{z} \in \Lambda^\perp_{\mathbf{u}}(\mathbf{A}_{\mathbf{H}})$, 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 $D_{\Lambda^\perp_{\mathbf{u}}(\mathbf{A}_{\mathbf{H}}), s \cdot \omega_n}$, 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 [28].

3.2 Functionalities for Threshold Sampling

Ideal functionalities for threshold key generation and discrete Gaussian sampling are specified in Figure 1 and Figure 2, respectively; they internally execute the standalone algorithms described above.

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.

Fig. 1. Key generation functionality

To realize $\mathcal{F}_{\text{KG}}$ in the trusted setup model (as used in [13]) we can simply let the trusted party play the role of $\mathcal{F}_{\text{KG}}$, because key generation is a one-time setup. Without trusted setup, we give in Section 4 a simple and efficient protocol that realizes $\mathcal{F}_{\text{KG}}$ using a simple integer-sampling functionality $\mathcal{F}_{\text{Samp}\mathbb{Z}}$. This in turn can be realized using general-purpose multiparty computation tools.

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 [25] 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.

Fig. 2. Gaussian sampling functionality

We realize $\mathcal{F}_{\text{GS}}$ in Section 3.3. For modularity, the following subsections first define two lower-level functionalities $\mathcal{F}_{\text{Perturb}}$ and $\mathcal{F}_{\text{Correct}}$ (Figures 3 and 4), which respectively generate the perturbation and syndrome-correction components of the standalone sampling algorithm. We describe how these helper functionalities can be realized efficiently and noninteractively using trusted setup, and the full version of this paper realizes them without trusted setup. 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 so that the trusted setup/offline precomputation phases of our protocols can prepare sufficient randomness to support noninteractive online phases. (If the bound B is reached, then the parties can just initialize new copies of $\mathcal{F}_{\text{GS}}$, $\mathcal{F}_{\text{Perturb}}$, $\mathcal{F}_{\text{Correct}}$.)

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 it (and any realization) can be invoked offline, before the result is needed. With trusted setup, the functionality can be realized trivially by just precomputing and distributing (shares of) B samples in the initialization phase, which the parties then draw from in the online phase. Without trusted setup $\mathcal{F}_{\text{Perturb}}$ can be realized relatively efficiently using $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ and some standard low-level MPC functionalities (for multiplication and blinding).

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 online 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 1 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 the lattice coset to which it belongs. (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 $\mathsf{perturb}$ 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 received shares.
- Send $(\mathsf{perturb}, sid, [\![\mathbf{p}]\!]^i, \bar{\mathbf{w}}, \mathbf{w})$ to each party i in $\mathcal{P}$.

Fig. 3. Perturbation functionality

Lemma 1. *Let $\bar{\mathbf{A}} \in \mathbb{Z}_q^{n \times \bar{m}}$ be uniformly random for $\bar{m} = m - nk \geq n \lg q + \omega(\log n)$, and let*

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

(where $\oplus$ denotes the direct sum). 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* $\mathbf{u} = \left[\begin{smallmatrix}\bar{\mathbf{w}} \\ \mathbf{w}\end{smallmatrix}\right] = \mathbf{B}\mathbf{p} \in \mathbb{Z}_q^{2n}$ *is* $\mathrm{negl}(n)$*-far from uniform, and the conditional distribution of* $\mathbf{p}$ *given* $\mathbf{u}$ *is* $D_{\Lambda^\perp_{\mathbf{u}}(\mathbf{B}), \sqrt{\Sigma_{\mathbf{p}}}}$.

Proof. By [25, 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 [25], we have $\eta_{\epsilon'}(\Lambda^\perp(\mathbf{G})) \leq \sqrt{5} \cdot \omega_n$. 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

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

Syndrome Correction. Our syndrome correction 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 $\Lambda^\perp_{\mathbf{v}}(\mathbf{A})$, where $\mathbf{v}$ depends on the desired syndrome $\mathbf{u}$, the functionality must be invoked online. As indicated in the overview, the standalone algorithm samples $\mathbf{z} \leftarrow \Lambda^\perp_{\mathbf{v}}(\mathbf{G})$ 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 $\mathbf{v}$ itself is not revealed in the standalone algorithm, but in our setting $\mathbf{v}$ is determined solely by public information like the tags $\mathbf{H}^*, \mathbf{H}$ and the syndromes $\bar{\mathbf{w}}, \mathbf{w}$ of the perturbation $\mathbf{p}$.)

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 $\mathsf{correct}$ 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 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 $\mathcal{P}$.

Fig. 4. Syndrome correction functionality

Realizing $\mathcal{F}_{\text{Correct}}$ with a noninteractive protocol relies crucially on the *parallel* and *offline* nature of the corresponding step of sampling a coset of $\Lambda^\perp(\mathbf{G})$ in the algorithm of [25]. In particular, we use the fact that without knowing $\mathbf{v}$ in advance, that algorithm can precompute *partial* samples for each of the $q = \mathrm{poly}(n)$ 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 secret-sharing homomorphism, the shares $[\![\mathbf{y}]\!]^i$ recombine to some $\mathbf{y} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right]\mathbf{z} \in \mathbb{Z}^m$ for some Gaussian-distributed $\mathbf{z}$ of parameter $s_{\mathbf{g}} \cdot \omega_n$, such that $\mathbf{A}_{\mathbf{H}} \cdot \mathbf{y} = (\mathbf{H}^* - \mathbf{H}) \cdot \mathbf{v} \in \mathbb{Z}_q^n$.

The full version gives an efficient protocol for $\mathcal{F}_{\text{Correct}}$, without trusted setup. It populates the local queues $Q^i_{j,v}$ in the offline phase in a distributed manner, using the shares of $\mathbf{R}$ together with access to $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ and standard share-blinding $\mathcal{F}_{\text{Blind}}$ and multiplication $\mathcal{F}_{\text{Mult}}$ functionalities. In short, it samples (shares of) the values $\mathbf{z}_{j,v}$ from the coset $\Lambda^\perp_v(\mathbf{g}^t)$ using $\mathcal{F}_{\text{Samp}\mathbb{Z}}$, the homomorphic properties of secret sharing, and $\mathcal{F}_{\text{Blind}}$. Then using $\mathcal{F}_{\text{Mult}}$ it computes (shares of) $\mathbf{y}_{j,v} = \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right](\mathbf{e}_j \otimes \mathbf{z}_{j,v})$.

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 realizing $\mathcal{F}_{\text{GS}}$ using these helper functionalities, 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 $\mathcal{F}_{\text{GS}}$, $\mathcal{F}_{\text{Perturb}}$, and $\mathcal{F}_{\text{Correct}}$ 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 of those shares. Moreover, such checks would be unnecessary in the usual 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 arguments $\bar{\mathbf{A}}, z$ corresponding to a statistically secure instantiation of the trapdoor generator from [25]. Similarly, $\mathcal{F}_{\text{GS}}$ (and

[3] This issue is not limited to our setting, and can arise anytime the key-generation and secret-key operations of a threshold scheme are put into separate functionalities. We note that using "joint state" [14] does not appear to resolve the issue, because it only allows multiple instances of the *same* protocol to securely share some joint state.

$\mathcal{F}_{\text{DelTrap}}$) 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 $\mathcal{F}_{\text{Correct}}$ are not intended for direct use by applications, but for proving the security of their realizations we also require that they be initialized using a prior output of $\mathcal{F}_{\text{KG}}$.) These restrictions are all described more formally in the full version of the paper.

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. As long as an application protocol obeys the above conditions in its use of $\mathcal{F}_{\text{KG}}$ and $\mathcal{F}_{\text{GS}}$ (and $\mathcal{F}_{\text{DelTrap}}$), the UC framework's composition theorem will still hold for the application itself, *without* any restriction on the 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 1, and proved via the simulator $\mathcal{S}_{\text{GS}}$ in Figure 6.

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. In other words, in the fully realized sampling protocol, where the helper functionalities are replaced by their respective realizations, 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.

Theorem 1. *Protocol π_{GS} statistically realizes $\mathcal{F}_{GS}$ in the $(\mathcal{F}_{Perturb}, \mathcal{F}_{Correct})$-hybrid model for t-limited environments in $\mathcal{Z}$.*

Proof (sketch). Essentially, the simulator $\mathcal{S}_{\text{GS}}$ in Figure 6 just maintains consistent sharings of $\mathbf{p} = \mathbf{0}$ and $\mathbf{y} = \mathbf{x}$ for each call to sample, and 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

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 $\mathbf{x} = \mathbf{p} + \mathbf{y}$.
- Output $(\mathsf{sample}, sid, \mathbf{x})$.

Fig. 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.

Fig. 6. Simulator for π_{GS}

on any fixed $\bar{\mathbf{w}}, \mathbf{w}$) of the final output $\mathbf{x}$ in both worlds. For item (1), as proved in Lemma 1, 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 $\Lambda_{\mathbf{u}}^{\perp}(\mathbf{B})$, 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 $D_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}_{\mathbf{H}}), s}$, 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 almost word-for-word identical to that of the "convolution lemma" from [25], 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 fixed coset of $\Lambda^{\perp}(\mathbf{B})$, instead of over $\mathbb{Z}^m$ as in the standalone algorithm. Fortunately, Lemma 1 says that $\sqrt{\Sigma_{\mathbf{p}}} \geq 2\eta_{\epsilon}(\Lambda^{\perp}(\mathbf{B}))$, and this is enough to adapt the proof from [25] 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

Here we sketch a straightforward use of the above protocols to do distributed trapdoor delegation, which is used in hierarchical IBE schemes. Due to space restrictions, we leave the formal definition of a trapdoor delegation functionality, protocol, and proof of security to the full version of the paper.

The functionality $\mathcal{F}_{\text{DelTrap}}$ corresponds to the algorithm $\mathsf{DelTrap}$ in [25] for delegating a lattice trapdoor. That algorithm works as follows: given a trapdoor $\mathbf{R}$ for some $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and an extension $\mathbf{A}' = [\mathbf{A}_{\mathbf{H}} | \mathbf{A}_1] \in \mathbb{Z}_q^{n \times (m+nk)}$ (where $\mathbf{A}_{\mathbf{H}} = \mathbf{A} - [\mathbf{0} \mid \mathbf{HG}]$ as always) and tag $\mathbf{H}' \in \mathbb{Z}_q^{n \times n}$, it outputs 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 $\mathbf{R}$). It does this simply by sampling Gaussian columns of $\mathbf{R}'$ to satisfy the relation $\mathbf{A}_{\mathbf{H}} \cdot \mathbf{R}' = \mathbf{H}' \cdot \mathbf{G} - \mathbf{A}_1$. In the threshold setting, where the parties have a sharing of the trapdoor $\mathbf{R}$, a distributed protocol for this process is trivial in the $\mathcal{F}_{\text{GS}}$-hybrid model: the parties simply use $\mathcal{F}_{\text{GS}}$ to sample the columns of $\mathbf{R}'$, using the public columns of $\mathbf{H}' \cdot \mathbf{G} - \mathbf{A}_1$ as the desired syndromes.

4 Key Generation without Trusted Setup

Here we show how to implement the key-generation functionality $\mathcal{F}_{\text{KG}}$ without any trusted setup, instead using access to two low-level functionalities $\mathcal{F}_{\text{Blind}}$ and $\mathcal{F}_{\text{Samp}\mathbb{Z}}$. Informally, $\mathcal{F}_{\text{Blind}}$ takes shares of some value and returns to each party a fresh sharing of the same value, and $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ distributes shares of a discrete Gaussian-distributed value over the integer lattice $\mathbb{Z}$ (or equivalently, $\mathbb{Z}^{h \times w}$ for some $h, w \geq 1$). The full definitions of these functionalities, which we use for realizing other functionalities without trusted setup, are given in the full version of the paper along with descriptions of interactive protocols realizing them offline. A simplified version of $\mathcal{F}_{\text{Samp}\mathbb{Z}}$ that is sufficient for realizing $\mathcal{F}_{\text{KG}}$ is given in Figure 7.

The protocol π_{KG} realizing $\mathcal{F}_{\text{KG}}$ in the $(\mathcal{F}_{\text{Samp}\mathbb{Z}}, \mathcal{F}_{\text{Blind}})$-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 get shares

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.

Fig. 7. Integer sampling functionality

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 not anything more about the honest parties' shares $[\![\mathbf{R}]\!]^i$ themselves. The formal protocol π_{KG} is given in Figure 8.

Protocol π_{KG} in the $(\mathcal{F}_{\text{Samp}\mathbb{Z}}, \mathcal{F}_{\text{Blind}})$-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 $(\mathsf{gen}, sid, \mathbf{A} = [\bar{\mathbf{A}} \mid \mathbf{H}^* \cdot \mathbf{G} + \mathbf{A}_1], [\![\mathbf{R}]\!]^i)$.

Fig. 8. Key generation protocol

The announced (blinded) shares $-\bar{\mathbf{A}}[\![\mathbf{R}]\!]^i$ form a uniformly random (and independent of the honest parties' outputs $[\![\mathbf{R}]\!]^i$) sharing of $\mathbf{A}_1 = -\bar{\mathbf{A}}\mathbf{R}$. This is the heart of the security analysis; a simulator for demonstrating security is given in the full version.

Theorem 2. *Protocol π_{KG} statistically realizes $\mathcal{F}_{KG}$ in the $(\mathcal{F}_{Samp\mathbb{Z}}, \mathcal{F}_{Blind})$-hybrid model for t-limited environments in $\mathscr{Z}$.*

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. EUROCRYPT, pp. 553–572. Springer, Heidelberg (2010)
2. Agrawal, S., Freeman, D.M., Vaikuntanathan, V.: Functional encryption for inner product predicates from learning with errors. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 21–40. Springer, Heidelberg (2011)
3. Ajtai, M.: Generating hard instances of the short basis problem. In: Wiedermann, J., Van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 1–9. Springer, Heidelberg (1999)
4. Ajtai, M.: Generating hard instances of lattice problems. Quaderni di Matematica 13, 1–32 (2004); Preliminary version in STOC 1996

5. Almansa, J.F., Damgård, I.B., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 593–611. Springer, Heidelberg (2006)
6. Alwen, J., Peikert, C.: Generating shorter bases for hard random lattices. Theory of Computing Systems 48(3), 535–553 (2011); Preliminary version in STACS 2009
7. Bendlin, R., Damgård, I.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 201–218. Springer, Heidelberg (2010)
8. Boneh, D., Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. SIAM J. Comput. 36(5), 1301–1328 (2007)
9. Boneh, D., Freeman, D.M.: Homomorphic signatures for polynomial functions. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 149–168. Springer, Heidelberg (2011)
10. Boneh, D., Freeman, D.M.: Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In: Public Key Cryptography, pp. 1–16 (2011)
11. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067 (2000), `http://eprint.iacr.org/`
12. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145 (2001)
13. Canetti, R., Goldwasser, S.: An efficient *threshold* public key cryptosystem secure against adaptive chosen ciphertext attack. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 90–106. Springer, Heidelberg (1999)
14. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Heidelberg (2003)
15. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (2010)
16. Cayrel, P.-L., Lindner, R., Rückert, M., Silva, R.: A lattice-based threshold ring signature scheme. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. LATINCRYPT, pp. 255–272. Springer, Heidelberg (2010)
17. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
18. Desmedt, Y.G., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (1990)
19. Desmedt, Y., Frankel, Y.: Perfect homomorphic zero-knowledge threshold schemes over any finite abelian group. SIAM J. Discrete Math. 7(4), 667–679 (1994)
20. Fehr, S.: Span programs over rings and how to share a secret from a module. Master's thesis, ETH Zurich, Institute for Theoretical Computer Science (1998)
21. Feng, T., Gao, Y., Ma, J.: Changeable threshold signature scheme based on lattice theory. In: International Conference on E-Business and E-Government, pp. 1311–1315 (2010)
22. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: STOC, pp. 197–206 (2008)
23. Dov Gordon, S., Katz, J., Vaikuntanathan, V.: A group signature scheme from lattice assumptions. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 395–412. Springer, Heidelberg (2010)
24. Klein, P.N.: Finding the closest lattice vector when it's unusually close. In: SODA, pp. 937–941 (2000)
25. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 700–718. Springer, Heidelberg (2012)

26. Micciancio, D., Regev, O.: Worst-case to average-case reductions based on Gaussian measures. SIAM J. Comput. 37(1), 267–302 (2007); Preliminary version in FOCS 2004
27. Myers, S., Sergi, M., Shelat, A.: Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive, Report 2011/454 (2011), http://eprint.iacr.org/
28. Nguyen, P.Q., Regev, O.: Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. J. Cryptology 22(2), 139–160 (2009); Preliminary version in Eurocrypt 2006
29. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: STOC, pp. 333–342 (2009)
30. Peikert, C.: An efficient and parallel gaussian sampler for lattices. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 80–97. Springer, Heidelberg (2010)
31. Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: STOC, pp. 187–196 (2008)
32. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. J. ACM 56(6), 1–40 (2009); Preliminary version in STOC 2005
33. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
34. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
35. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (2000)
36. Xie, X., Xue, R., Zhang, R.: Efficient threshold encryption from lossy trapdoor functions. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 163–178. Springer, Heidelberg (2011)