Basic Lattice Cryptography: Encryption and Fiat-Shamir Signatures

Vadim Lyubashevsky

IBM Research - Zurich, Saumerstrasse 4, 8803 Rüschlikon, Switzerland

(Original Version: December 2019. Last Updated December 17, 2020)

Abstract. This tutorial introduces readers, who are already familiar with the basics of public key cryptography, to the foundational constructions of lattice-based cryptography. It describes the necessary techniques for constructing state-of-the-art encryption and signature schemes based on the hardness of the LWE and SIS problems, as well as their more efficient versions defined over polynomial rings.

Contents

1 Prelude

The beginning of lattice-based cryptography traces back to two works from the mid 1990's. The first was Ajtai's result [Ajt96] that showed that being able to solve a random instance of a certain problem was as hard as solving some believed-to-be-hard problem for *every* lattice. The second was the NTRU cryptosystem [HPS98], which constructed an efficient cryptosystem based on a new, potentially-hard distribution over lattice problems. These two results garnered some interest from the cryptographic community leading to a series of works throughout the 2000's [Mic07, MR07, Reg09, PR06, LM06, GPV08, Lyu09, LPR10] that aimed to combine the two branches of research by creating efficient (at least in an asymptotic sense) encryption, signature and identity-based encryption schemes which were based on solid theoretical foundations. Lattice-based cryptography gained further momentum in the early 2010's due to the first realization, from any assumption, of a fully-homomorphic encryption scheme [Gen09] and the increase in research aimed towards building a quantum computer that would be able to break all cryptography based on factoring and discrete log [Sho97]. The latter part of the 2010's saw lattice-based schemes go from being just asymptotically-efficient to being truly practical. By the time the NIST post-quantum cryptography standardization process started in late 2017, lattice-based schemes were among the fastest and most compact quantum-resistant primitives and even surpassed their number-theoretic counterparts in terms of efficiency.

This tutorial focuses on describing schemes from the last category and the hope is that by the end, the reader will have all the tools needed to read and understand the concrete instantiations and design decisions made in most of the encryption and signature schemes submitted to the NIST standardization process. The target audience for this manuscript are people who are already familiar with basic cryptography and have therefore seen concepts such as indistinguishability, CPA-secure encryption, zero-knowledge proofs, random oracles, the Fiat-Shamir transform, etc. in some prior context. Those familiar with these concepts most likely first saw them when studying public key cryptographic primitives based on the hardness of the discrete log or RSA problems. Keeping the discrete logarithm constructions, such as El Gamal encryption and Schnorr signatures, in mind will actually be very useful for understanding the high-level ideas of the lattice constructions.

There is a lot of lattice cryptography that this manuscript does not cover. For example, it does not say anything about more "advanced" constructions beyond encryption or signatures – a good overview of some of those may be found in [Pei16]. It also doesn't say much about cryptanalysis or the geometric aspects of lattice cryptography which are crucial to understanding some more advanced constructions. A good reference for the geometrical foundations of lattices is [MG02] and the lecture notes [Mic19]. Some recent papers related to algorithms and cryptanalysis relating to the problems upon which the lattice-based NIST standardization candidates are based are [ACD$^+$18, AM18, ADH$^+$19].

While the main reason for the interest in lattice-based encryption and signatures is their presumed resistance to quantum attacks, we do not broach this topic. The random oracle model (ROM) where, in the security proof, a concrete cryptographic hash function is replaced with a perfect random function to which the adversary only has oracle access, has an analogy in the quantum setting (QROM) where the adversary is allowed to also make quantum queries (i.e. querying the oracle on a superposition of inputs) to such a function. While security proofs in the ROM do not immediately carry over to the QROM setting, there have been many recent works bridging the two closer together (e.g. [HHK17, SXY18, KLS18, DFMS19, LZ19]) with the only remaining difference being the tightness of, or the concrete problem in, the security reduction. At the time of this writing, if one just replaces all the cryptographic functions with their counterparts that have the requisite quantum security (i.e. for 128-bit security, block ciphers should have 256-bit keys, and cryptographic hash functions should have 256-bit outputs for 2^{nd}-preimage resistance and 384-bit outputs for collision resistance), then there aren't any known improved attacks on real-life schemes proven secure in the ROM if the adversary is given the extra quantum power of querying the random oracle on a superposition of inputs.

1.1 Outline

In section 2, we present the framework for lattice-based encryption over the integers based on the hardness of the LWE problem that stems from the original work of Regev [Reg09] up to its most efficient instantiation [BCD+16]. We also discuss many small, but crucial, tricks for compressing the outputs. Some of these tricks are fairly general and will also come in handy in the constructions presented in the later sections. In Section 3, we introduce lattices and present a connection between the hardness of the LWE problem and the hardness of certain lattice problems. In Section 4, we review polynomial rings and instantiate the analogous version of the encryption scheme from Section 2, which forms the framework for the most efficient lattice-based encryption schemes. We also work out a particular detailed example showing how one would efficiently perform key generation, encryption, decryption, and calculate the decryption error probability. Finally in Section 5, we present all the needed techniques for optimized lattice-based analogues of the Schnorr signature schemes [DKL+18]. The techniques for constructing Σ-protocols also serve as a gateway to more advanced constructions involving zero-knowledge proofs.

2 Encryption

We only discuss CPA-secure encryption, as there are generic transformations that convert CPA-secure encryption schemes to CCA-secure ones as well as to key exchange protocols secure against active attackers. One should take note that the the CPA-secure schemes in this chapter can be defined in a way that gives rise to decryption errors. That is, even if the encryptor behaves honestly, he may produce a ciphertext that will not decrypt to the message he encrypted. To achieve the desired security against active attackers (i.e. CCA security), one would need to use generic transformations such as Fujisaki-Okamoto (c.f. [FO99, Den02, HHK17, SXY18]) that take this error into account. In general, having a small-enough decryption error (say, around 2^{-150}) doesn't appear to hurt the practical security of schemes.

2.1 Some Notation

All operations in this section will be performed in the ring $(\mathbb{Z}_q, +, \times)$ with the usual addition and multiplication of integers modulo q. For a set S, we write $a \leftarrow S$ to mean that a is chosen uniformly at random from the set S. For any positive integer β, we will define the set

$$[\beta] = \{-\beta, \ldots -1, 0, 1, \ldots, \beta\}. \tag{1}$$

This notation naturally extends to vectors (and matrices) by writing $[\beta]^{n\times m}$. By default, all our vectors will be column vectors. One can also indicate that a vector $\mathbf{v}$ is in $[\beta]^m$ by writing $\|\mathbf{v}\|_\infty \le \beta$.

2.2 A Motivating Example

Let's pretend for a second that for positive integers $q, n \ge m$, and $\beta \ll q$, the following two distributions are computationally indistinguishable (in the security parameter related to m):

1. $(\mathbf{A}, \mathbf{A}\mathbf{s})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ and $\mathbf{s} \leftarrow [\beta]^m$
2. $(\mathbf{A}, \mathbf{u})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$.

Our indistinguishability assumption is clearly absurd because one can use Gaussian elimination to invert $\mathbf{A}$ (or an $n \times n$ submatrix of $\mathbf{A}$) and check whether there is indeed an $\mathbf{s} \in [\beta]^m$ satisfying $\mathbf{As} = \mathbf{u}$. Please bear with us, though, because a slightly modified version of this assumption forms the foundation of most lattice cryptography. We will now use this assumption to construct a simple CPA-secure public key encryption scheme that very much resembles the discrete logarithm based El Gamal encryption scheme. The secret and public keys of the scheme are

$$\mathsf{sk} : \mathbf{s} \leftarrow [\beta]^m, \ \mathsf{pk} : (\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times m}, \mathbf{t} = \mathbf{As}). \tag{2}$$

To encrypt a message $\mu \in \mathbb{Z}_q$, the encryptor picks a random vector $\mathbf{r} \leftarrow [\beta]^m$ and outputs the ciphertext $(\mathbf{u}, v) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$ where

$$(\mathbf{u}^T = \mathbf{r}^T\mathbf{A}, v = \mathbf{r}^T\mathbf{t} + \mu). \tag{3}$$

To decrypt, one simply computes

$$\mu = v - \mathbf{u}^T\mathbf{s}. \tag{4}$$

Correctness of the scheme follows because

$$v - \mathbf{u}^T\mathbf{s} = \mathbf{r}^T\mathbf{t} + \mu - \mathbf{r}^T\mathbf{As} = \mathbf{r}^T\mathbf{As} + \mu - \mathbf{r}^T\mathbf{As} = \mu. \tag{5}$$

The security of the scheme follows by our (absurd) assumption and the hybrid argument. By the assumption, the public key $(\mathbf{A}, \mathbf{t})$ in (2) is indistinguishable from uniform. The public key matrix $\mathbf{A}' = [\mathbf{A} \mid \mathbf{t}] \in \mathbb{Z}_q^{m \times (m+1)}$ is therefore indistinguishable from uniform, and using our assumption once more we obtain that the distribution $(\mathbf{A}', \mathbf{r}^T\mathbf{A}')$ is also indistinguishable from uniform. It therefore follows that the distribution of $(\mathbf{A}, \mathbf{t}, \mathbf{u}, v)$ is indistinguishable from uniform, and therefore the scheme is CPA-secure.
Notice that we used our indistinguishability assumption twice – once for arguing that the public key looks random, and the other time to argue that the ciphertext does. Instead of choosing $\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times m}$, we could have chosen it randomly from $\mathbb{Z}_q^{n \times m}$ for $m \neq n$. If we set $m \gg n$, then one can easily show that the public key is actually statistically-close to being uniform. But then forming the ciphertext would still require us to use the absurd indistinguishability assumption. So using the assumption at least once is inescapable.

2.3 The LWE Problem

We will now make a "small" adjustment to the assumption from the previous section which will make it plausibly valid and still allow us to construct a cryptosystem following the same outline. We now define a simple version of the Learning with Errors Problem (LWE) [Reg09] upon which most of lattice cryptography rests.

Definition 1. For positive integers m, n, q, and $\beta \ll q$, the $\mathsf{LWE}_{n,m,q,\beta}$ problem asks to distinguish between the following two distributions:

1. $(\mathbf{A}, \mathbf{As} + \mathbf{e})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow [\beta]^m, \mathbf{e} \leftarrow [\beta]^n$
2. $(\mathbf{A}, \mathbf{u})$, where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^n$.

The crucial part that makes $\mathsf{LWE}_{n,m,q,\beta}$ hard is the presence of the additional "error" vector $\mathbf{e}$ which makes the simple Gaussian elimination attack inapplicable. The exact hardness of the problem depends on the parameters n, m, q, and β. The problem becomes harder as m and β/q grow. The parameter n is not known to have a large impact on the hardness of the problem except in extreme cases where n is as large as approximately $m^{2\beta+1}$, in which case one can build a distinguisher in time approximately $m^{2\beta}$ by linearization techniques [AG11]. In the constructions that we will present throughout this chapter, n will never need to be so large. Since the parameter n will not be particularly important, we will sometimes omit it when stating the hardness assumption and just write $\mathsf{LWE}_{m,q,\beta}$.
We also mention that there is nothing too special about using the uniform distribution for our secret and error terms – it just makes the presentation simpler and so we

choose to use this distribution throughout. In the original definition of LWE, the error distribution was chosen as a rounded Gaussian. This distribution was necessary for the average-case to worst-case reduction proofs [Reg09, Pei09] showing that LWE is at least as hard as some worst-case lattice problems. This restriction has since been shown unnecessary, and one can use, for example, the uniform distribution [DM13, MP13]. Some practical implementations use the binomial distribution to generate the errors (c.f. [ADPS16, BDK+18]) because in practice it is sometimes faster to generate a string of bits and add them up instead of generating a uniform element in $[\beta]$.

Something to note is that in the definition of LWE, we are not setting any conditions on the relative sizes of m, n and q. It's therefore possible that some choices will lead to trivially hard LWE instances. For example, if $n < m$ and β is large-enough, then the distribution $(\mathbf{A}, \mathbf{As} + \mathbf{e})$ could indeed be statistically-close to $(\mathbf{A}, \mathbf{u})$. And then one clearly should not be able to build an encryption scheme based on just this assumption. The reason why things will not work out will become clear when we consider correctness of the decryption. It is of course also possible to set parameters so that LWE is not a hard problem. For example, if $m = 1$, then it is not difficult to figure out whether $\mathbf{As} + \mathbf{e}$ is close to a multiple of the *vector* $\mathbf{A}$. We will discuss the hardness of the LWE problem in Section 3. Another note is that it is also possible to define LWE where the secret $\mathbf{s}$ is chosen to be uniform in $\mathbb{Z}_q^m$ (being careful to then take n large enough so that the LWE problem does not become trivially hard); and this is indeed how LWE was originally defined in [Reg09]. A simple proof [ACPS09], however, shows that taking $\mathbf{s}$ to be from the same distribution as $\mathbf{e}$ results in an essentially equally-hard problem. For applications, it is usually more efficient to take $\mathbf{s}$ to be smaller, and so we will only consider this version of the LWE problem.

2.3.1 An LWE-Based Encryption Scheme

In the rest of this section, we will present cryptosystems which stem from the original work in [Reg09] and have been improved and generalized via a series of observations in various follow-up works (c.f. [ACPS09, Pei09, LPS10, LP11, BCD+16]). We can view the scheme in this section as a modification of the encryption scheme from Section 2.2, but based on the hardness of $\mathsf{LWE}_{m,q,\beta}$ instead the clearly false assumption made there. Our first modification will be the message μ – rather than being an arbitrary element in $\mathbb{Z}_q$, it will now come from the set $\{0, 1\}$. The key generation from (2) is modified to:

$$\mathsf{sk}: \mathbf{s} \leftarrow [\beta]^m, \ \mathsf{pk}: (\mathbf{A} \leftarrow \mathbb{Z}_q^{m \times m}, \mathbf{t} = \mathbf{As} + \mathbf{e}_1), \text{ where } \mathbf{e}_1 \leftarrow [\beta]^m. \tag{6}$$

To encrypt a message $\mu \in \{0, 1\}$, the encryptor chooses $\mathbf{r}, \mathbf{e}_2 \in [\beta]^m$ and $e_3 \in [\beta]$, and outputs

$$\left(\mathbf{u}^T = \mathbf{r}^T\mathbf{A} + \mathbf{e}_2^T, v = \mathbf{r}^T\mathbf{t} + e_3 + \frac{q}{2}\mu\right). \tag{7}$$

Let's first discuss a little notation. We are working in $\mathbb{Z}_q$, but the above equation has a strange-looking term $q/2$. What we mean by this is an element in $\mathbb{Z}_q$ that's closest to the rational number $q/2$ (e.g. if $q = 13$, then $q/2 = 6$). So the division operation is not in $\mathbb{Z}_q$ – i.e. we are not multiplying by the inverse of 2. (If we will ever want to do division in $\mathbb{Z}_q$, we will instead write it as multiplication by an inverse). Also, to be precise, we really should write $\lceil q/2 \rfloor$, and such rounded terms will indeed come in often. For elegance of notation, we will usually omit specifying exactly to which of the neighboring integers we should be rounding, as this is never really important for understanding the main concepts.

Before discussing decryption, let's quickly go through the security argument to see why the scheme is based on $\mathsf{LWE}_{m,q,\beta}$. The argument is the same as in Section 2.2. The indistinguishability of the public key from the uniform distribution over $\mathbb{Z}_q^{m\times(m+1)}$ stems directly from the $\mathsf{LWE}_{m,q,\beta}$ assumption. Rewriting the public key $(\mathbf{A}, \mathbf{t})$ as a matrix $\mathbf{A}' = [\mathbf{A} \mid \mathbf{t}]$, we see that the $\mathsf{LWE}_{m,q,\beta}$ assumption again implies

that the distribution $\left(\mathbf{A}', \mathbf{r}^T\mathbf{A}' + \begin{bmatrix}\mathbf{e}_2 \\ e_3\end{bmatrix}^T\right)$ is also indistinguishable from uniform. Thus $(\mathbf{A}, \mathbf{T}, \mathbf{u}, v)$ is indistinguishable from uniform for any $\mu \in \{0,1\}$ based on $\mathsf{LWE}_{m,q,\beta}$. Note that we used the $\mathsf{LWE}_{m,q,\beta}$ assumption twice and the parameter m came into play as the number of columns of $\mathbf{A}$ in the argument showing that the public key looks random, and then as the number of rows in $\mathbf{A}$ in the argument that the ciphertext looks random. This is intuitively the reason why it makes sense to set the number of rows and columns in $\mathbf{A}$ to be equal when trying to minimize the combined size of the public key and ciphertext in public key encryption. We discuss this topic further in Section 2.4, and in Section 2.5.4 also discuss some applications, and a sightly-modified cryptosystem, in which one may not want to have the number of rows equal to the number of columns.
To decrypt, one computes $v - \mathbf{u}^T\mathbf{s}$. But rather than this cleanly giving us the message μ as in (4), we instead obtain

$$v - \mathbf{u}^T\mathbf{s} = \mathbf{r}^T(\mathbf{A}\mathbf{s} + \mathbf{e}_1) + e_3 + \frac{q}{2}\mu - \left(\mathbf{r}^T\mathbf{A} + \mathbf{e}_2^T\right)\mathbf{s} \tag{8}$$
$$= \mathbf{r}^T\mathbf{e}_1 + e_3 + \frac{q}{2}\mu - \mathbf{e}_2^T\mathbf{s} \tag{9}$$

As in (5), the $\mathbf{r}^T\mathbf{A}\mathbf{s}$ terms in the above equation cancelled out; we are however left with some combinations of "error" terms. Luckily, all these error terms have coefficients bounded by $\pm\beta$, and so the vector products $\mathbf{r}^T\mathbf{e}_1$ and $\mathbf{e}_2^T\mathbf{s}$ in (9) each consist of m terms of magnitude at most β^2 each. Therefore one can rewrite (9) as $e + \frac{q}{2}\mu$ where $e \in [2m\beta^2 + \beta]$, and so if the parameters are set such that $2m\beta^2 + \beta < q/4$ the decryptor can determine μ from looking at $v - \mathbf{u}^T\mathbf{s}$ by checking whether the preceding value is closer to 0 or to $q/2$.

2.3.2 Bounding the Total Error

The value $2m\beta^2 + \beta$ that we computed above is the upper bound on the error magnitude. Because the coefficients of $\mathbf{r}, \mathbf{s}, \mathbf{e}_1$, and $\mathbf{e}_2$ are chosen randomly in the range $[\beta]$, which is centered around 0, it is actually quite unlikely that the error will ever be so high. Due to cancellations, it will in fact be closer to $\mathcal{O}(\sqrt{m}\beta^2)$. In general, we would be fine to get a bound on the error such that with very high probability (say $1 - 2^{-150}$) the error will be below $q/4$. This would imply that the probability of a decryption error is at most 2^{-150}. Such small errors can be tolerated in applications and when using this CPA-encryption scheme as a component of other constructions (e.g. CCA-secure encryption).
There are asymptotic ways to approximately compute such bounds, but when dealing with concrete parameters and β is not too large, it is possible (using simple scripts) to get the exact value for

$$\Pr_{\mathbf{s},\mathbf{r},\mathbf{e}_1,\mathbf{e}_2 \leftarrow [\beta]^m, e_3 \leftarrow [\beta]}[\mathbf{r}^T\mathbf{e}_1 + e_3 - \mathbf{e}_2^T\mathbf{s} \in [\alpha]] \tag{10}$$

using the fact that probability distributions of sums of random variables can be modeled as products of polynomials. Suppose that A and B are random variables (possibly with different distributions) over the finite set $[\gamma]$. For all $i \in [\gamma]$, let A_i be the probability that A is equal to i. Now define the polynomial

$$A(X) = \sum_{i=-\gamma}^{\gamma} A_i X^i,$$

and the polynomial $B(X)$ in the same fashion. Let

$$C(X) = A(X) \cdot B(X) = \sum_{i=-2\gamma}^{2\gamma} C_i X^i$$

be the product of $A(X)$ and $B(X)$. One can now make a direct connection between the coefficients C_i and the probability that $A + B = i$. In particular,

$$\Pr[A + B = i] = C_i. \tag{11}$$

This implies that

$$\Pr[A + B \in [\alpha]] = \sum_{i=-\alpha}^{\alpha} C_\alpha. \tag{12}$$

This directly carries over to computing the probability in (10) by noticing that $\mathbf{r}^T\mathbf{e}_1 - \mathbf{e}_2^T\mathbf{s}$ is distributed as a sum of $2m$ *independent* random variables A, where

$$A_i = \Pr[A = i] = \Pr_{x,y \leftarrow [\beta]}[xy = i],$$

and e_3 is just a uniformly distributed random variable over $[\beta]$. So, for example, if $\beta = 2$, then

$$A_{-4} = A_4 = \frac{2}{25},\ A_{-2} = A_2 = \frac{4}{25},\ A_{-1} = A_1 = \frac{2}{25},\ A_0 = \frac{9}{25},$$

and all the other A_i are 0. If we then write

$$C(X) = \left(\frac{2}{25}X^{-4} + \frac{4}{25}X^{-2} + \frac{2}{25}X^{-1} + \frac{9}{25} + \frac{2}{25}X + \frac{4}{25}X^2 + \frac{2}{25}X^4\right)^{2m} \cdot \left(\frac{1}{5}X^{-2} + \frac{1}{5}X^{-1} + \frac{1}{5} + \frac{1}{5}X + \frac{1}{5}X^2\right),$$

then the probability in (10) is exactly $\sum_{i=-\alpha}^{\alpha} C_i$, where C_i are again the coefficients associated to X^i in $C(X)$. By setting $\alpha = q/4 - 1$, we will obtain the probability that the decryption algorithm in Section 2.3.1 will correctly decrypt.

2.4 Public Key and Ciphertext Size Trade-offs

We now know how to set the parameters m, q, β relative to one another so that the encryption scheme in Section 2.3.1 correctly decrypts with overwhelming probability. We haven't yet discussed how one should set the parameters in order for it to be secure, and we will defer this to a bit later. But we can still compute the sizes of the public key and the ciphertext in terms of the parameters q, m, β.
The public key (see (6)) consists of a random matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times m}$ and a vector $\mathbf{t} \in \mathbb{Z}_q^m$. Since $\mathbf{A}$ is completely random, there is no need to store it – if one creates $\mathbf{A}$ by choosing a 256-bit seed ρ and then defining $\mathbf{A}$ as the expansion of ρ using some cryptographic PRF (e.g. SHAKE based on the SHA-3 function), then one only needs to store the 256 bits ρ instead of the $m^2 \log q$ bits of $\mathbf{A}$. One will need to expand $\mathbf{A}$ from ρ when encrypting and decrypting, but it's usually a worthwhile trade-off to do so in lieu of storing the $\mathbf{A}$. The other part of the public key depends on the secret key and so cannot be compressed in a similar way, and thus the total public key size is $256 + m \log q$ bits. The ciphertext $(\mathbf{u}, v) \in \mathbb{Z}_q^m \times \mathbb{Z}_q$ can be represented using $(m + 1)\log q$ bits. Concrete secure parameter settings will require taking $m \approx 700$ and $q \approx 2^{13}$, and so it is somewhat inefficient to have the encryption of just one plaintext bit be so large. We will now give a generalization of the LWE encryption scheme that allows for various trade-offs between the public key and ciphertext size.
The reason that our current LWE-based encryption scheme has such a large ciphertext expansion is that $\mathbf{u}$ consists of $m \log q$ bits. To reduce the ciphertext expansion, we will give a variant of the scheme that amortizes the ciphertext part $\mathbf{u}$ among the encryption of many messages. The trade-off is that the public key will be larger.
Suppose that we would like to encrypt $N = k\ell$ bits, which we arrange into a matrix $\mathbf{M} \in \{0, 1\}^{k\times\ell}$. The key generation procedure will now be

$$\mathsf{sk}:\ \mathbf{S} \leftarrow [\beta]^{m\times\ell},\ \mathsf{pk}:\ \left(\mathbf{A} \leftarrow \mathbb{Z}_q^{m\times m}, \mathbf{T} = \mathbf{A}\mathbf{S} + \mathbf{E}_1\right),\ \text{where } \mathbf{E}_1 \leftarrow [\beta]^{m\times\ell}. \tag{13}$$

Observe that, compared to (6), the public key size is now increased to $256 + \ell m \log q$ bits. The encryption algorithm proceeds analogously where the encryptor chooses $\mathbf{R}, \mathbf{E}_2 \leftarrow [\beta]^{k\times m}$ and $\mathbf{E}_3 \leftarrow [\beta]^{k\times \ell}$, and outputs the ciphertext

$$\left(\mathbf{U} = \mathbf{RA} + \mathbf{E}_2, \mathbf{V} = \mathbf{RT} + \mathbf{E}_3 + \frac{q}{2}\mathbf{M}\right). \tag{14}$$

which is comprised of $km \log q + k\ell \log q$ bits. To obtain a trade-off between the public key and ciphertext sizes, one can vary the parameters k and ℓ, while keeping its product (the total number of bits N) fixed. To obtain the minimum combined public key and ciphertext size, one should set $k \approx \ell \approx \sqrt{N}$. In this way, encrypting N bits requires $\approx 256 + 2\sqrt{N}m \log q + N \log q$ bits, which in practice will be dominated by the $2\sqrt{N}m \log q$ term because one generally never needs to use public key encryption to encrypt more than $N = 256$ bits before switching to symmetric encryption.
The security of this cryptosystem is again directly based on $\mathsf{LWE}_{m,q,\beta}$. Notice that the public key $(\mathbf{A}, \mathbf{AS} + \mathbf{E}_1)$ can be rewritten as $(\mathbf{A}, \mathbf{As}_1 + \mathbf{e}_1, \ldots, \mathbf{As}_\ell + \mathbf{e}_\ell)$ where $\mathbf{s}_i$ and $\mathbf{e}_i$ are, respectively, the i^{th} columns of $\mathbf{S}$ and $\mathbf{E}_1$. The indistinguishability of $(\mathbf{A}, \mathbf{T})$ from uniform then directly follows from $\mathsf{LWE}_{m,q,\beta}$ using the usual hybrid argument with a loss of $\log \ell$ bits of security.[1]
Writing $\mathbf{A}' = [\mathbf{A} \mid \mathbf{T}]$, we again use the $\mathsf{LWE}_{m,q,\beta}$ assumption (and the hybrid argument) to note that the distribution $(\mathbf{A}', \mathbf{RA}' + [\mathbf{E}_1 \mid \mathbf{E}_2])$ is indistinguishable from the uniform distribution, and therefore

$$(\mathbf{U}, \mathbf{V}) = \left(\mathbf{A}', \mathbf{RA}' + [\mathbf{E}_1 \mid \mathbf{E}_2 + \mathbf{M}]\right) \tag{15}$$

is indistinguishable from uniform for any fixed message $\mathbf{M}$.
Decryption is done exactly the same way as we've been doing it for all the other schemes in this section. Given the ciphertext $(\mathbf{U}, \mathbf{V})$, the decryptor computes

$$\begin{aligned}\mathbf{V} - \mathbf{US} &= \mathbf{R}(\mathbf{AS} + \mathbf{E}_1) + \mathbf{E}_3 + \frac{q}{2}\mathbf{M} - (\mathbf{RA} + \mathbf{E}_2)\mathbf{S} \qquad (16)\\ &= \mathbf{RE}_1 + \mathbf{E}_3 + \frac{q}{2}\mathbf{M} - \mathbf{E}_2\mathbf{S}. \qquad (17)\end{aligned}$$

From above, observe that the $(i,j)^{th}$ coefficient of $\mathbf{V} - \mathbf{US}$ is equal to

$$\mathbf{r}^T\mathbf{e}_1 + e_3 + \frac{q}{2}\mu - \mathbf{e}_2^T\mathbf{s},$$

where $\mathbf{r}^T$ and $\mathbf{e}_2^T$ are the i^{th} rows of $\mathbf{R}$ and $\mathbf{E}_2$, $\mathbf{e}_1$ and $\mathbf{s}$ are the j^{th} columns of $\mathbf{E}_1$ and $\mathbf{S}$, and e_3 and μ are the $(i,j)^{th}$ positions of $\mathbf{E}_3$ and $\mathbf{M}$. Since all the vectors are in $\mathbb{Z}_q^m$ and all their coefficients were uniformly chosen from $[\beta]$, we are in the exact same situation as in (9) and so one can set the parameters m, q, β in the exact same way as before to have a small decryption error.

2.5 Some Variations and Optimizations

The scheme presented in the previous section is more of a general framework of what one would do in practice. When instantiating such a scheme with concrete parameters, there are several possible optimizations that one would want to attempt doing, which we will describe next. We should mention that it's not easy to figure out exactly which optimizations one can use, and to exactly how to set the parameters optimally, without actually trying some possibilities and seeing what security / output size one gets.

2.5.1 Reducing the Ciphertext Size by Removing the Low-Order Part

The ciphertext part $\mathbf{V}$ contributes $N \log q$ bits to the total ciphertext size. Suppose that instead of the encryptor publishing all $\log q$ bits of each coefficient of $\mathbf{V}$ as part of the ciphertext, he wanted to transmit only κ bits per coefficient. This is possible,

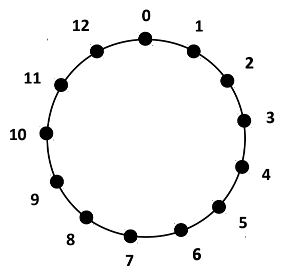

Figure 1: A representation of $\mathbb{Z}_{13}$ as points on a circle. The set $\mathcal{S} = \{\lceil i \cdot 13/4 \rfloor : 0 \le i < 4\}$ consists of the four points $0, 3, 6, 10$. $\mathcal{S}$ can be represented by 2 bits and every point in $\mathbb{Z}_{13}$ is within a distance of $\lceil 13/8 \rceil = 2$ of one of the elements of $\mathcal{S}$.

but will add a further error to the decryption equation. Visualizing the additive group $\mathbb{Z}_q$ as points on a circle (c.f. Figure 1), we want to pick a set $\mathcal{S} \subset \mathbb{Z}_q$ of size 2^κ so that the maximum distance between neighboring points in $\mathcal{S}$ (measured in the number of $\mathbb{Z}_q$ points between them) is as small as possible. Note that $q/2^\kappa$ is the smallest we can hope for, so we would like to get as close as possible to this number. One could define such a set as

$$\mathcal{S} = \{\lceil i \cdot q/2^\kappa \rfloor : 0 \le i < 2^\kappa\}. \tag{18}$$

If $2^\kappa \mid q$, then the distance between all the neighboring points is the same. Otherwise all distances are within 1 of each other, which is as good as one can hope for.

The crucial feature of the set $\mathcal{S}$ is that every $v \in \mathbb{Z}_q$ is within a distance of $\lceil q/2^{\kappa+1} \rceil$ of some element of $\mathcal{S}$. Let us define $\texttt{HIGH}_S(v)$ to be the element in S closest to v and $\texttt{LOW}_S(v)$ to be $v - \texttt{HIGH}_S(v)$. Then instead of transmitting $\mathbf{V} \in \mathbb{Z}_q^{k \times \ell}$ as part of the ciphertext, one could transmit a $\mathbf{V}' \in \mathcal{S}^{k \times \ell} = \texttt{HIGH}_S(\mathbf{V})$. Note that there exists an $\mathbf{E}' \in [q/2^{\kappa+1}]^{k \times \ell} = \texttt{LOW}_S(\mathbf{V})$ such that $\mathbf{V} = \mathbf{V}' + \mathbf{E}'$.

If the ciphertext $(\mathbf{U}, \mathbf{V}')$ is created in this fashion, then the decryption $\mathbf{V}' - \mathbf{US}$ will produce

$$\mathbf{V}' - \mathbf{US} = \mathbf{RE}_1 + \mathbf{E}_3 - \mathbf{E}' + \frac{q}{2}\mathbf{M} - \mathbf{E}_2\mathbf{S}, \tag{19}$$

where the only difference with the decryption in (17) is the presence of the $\mathbf{E}'$ term. Notice that if one sets κ to be close to $\log q$, then $\mathbf{E}'$ will have virtually no effect on the decryption error because there are other terms that are involved in an inner product which produce much larger coefficients in (19). In practice, one can usually set κ to be a small constant like 3 or 4 without the decryption error increasing too much. This implies that instead of contributing $N \log q$ bits to the ciphertext, $\mathbf{V}$ contributes just κN bits. Assuming that $\mathbf{V}$ is uniformly distributed, we can obtain the exact distribution of $\mathbf{E}'$ and then compute the decryption error probability using the exact same techniques as in Section 2.3.2.

In some cases it may also make sense to use a similar bit reduction procedure on the ciphertext part $\mathbf{U}$. Here, though, one cannot remove bits without noticeably increasing the decryption error because any error added to $\mathbf{U}$ will get multiplied by $\mathbf{S}$, and so we would get an additional $\mathbf{E}''\mathbf{S}$ term in (19), where $\mathbf{E}''$ is defined analogously to $\mathbf{E}'$. But because $\mathbf{U}$ is the dominant source of ciphertext size, reducing the number of bits in $\mathbf{U}$ even by a little bit, could make a noticeable difference as well. The trick is then to balance the decryption error and the ciphertext size using trial-and-error.

[1]As with many applications of the hybrid argument, it's not clear in this case whether there is a real security loss or it's just an artefact of the proof.

2.5.2 Learning with Rounding

By the LWE assumption, the distribution $(\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}) \in Z_q^{n\times m} \times \mathbb{Z}_q^n$ looks indistinguishable from uniform when the coefficients of $\mathbf{s}$ and $\mathbf{e}$ come from $[\beta]$. If we round each coefficient of $\mathbf{t}$ to the nearest point in some subset $\mathcal{S} \subseteq \mathbb{Z}_q$, as in Section 2.5.1, then the distribution of $(\mathbf{A}, \texttt{HIGH}_S(\mathbf{t}))$ is still indistinguishable from $(\mathbf{A}, \texttt{HIGH}_S(\mathbf{u}))$, where $\mathbf{u}$ is uniformly random. Let's now examine $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$. Since $\mathbf{e}$ has coefficients in a small subset $[\beta]$, it may turn out that it doesn't have *any* effect on the value of $\texttt{HIGH}_S(\mathbf{A}\mathbf{s} + \mathbf{e})$. In particular, when $\mathcal{S}$ is defined as in (18), then the probability (over the randomness of $\mathbf{A}, \mathbf{s}, \mathbf{e}$) that $\texttt{HIGH}_S(\mathbf{A}\mathbf{s} + \mathbf{e}) = \texttt{HIGH}_S(\mathbf{A}\mathbf{s})$ is approximately $\left(1 - \frac{\beta|\mathcal{S}|}{2q}\right)^n$. To see this, notice that if some coefficient of $\mathbf{A}\mathbf{s}$ is not within $\beta/2$ of the midpoint between two points in $\mathcal{S}$, then adding an error in $[\beta]$ will not change the point in $\mathcal{S}$ to which it gets rounded.

So whenever q is large with respect to β and $|\mathcal{S}|$, adding $\mathbf{e}$ doesn't make a difference, and so there is no reason to add it in the first place! Distinguishing the distribution $(\mathbf{A}, \texttt{HIGH}_S(\mathbf{A}\mathbf{s}+\mathbf{e}))$ from uniform is called the Learning with Rounding (LWR) problem, and it is at least as hard as LWE whenever adding $\mathbf{e}$ doesn't affect the rounded output [BPR12, AKPW13, BGM$^+$16]. Sometimes this assumption is used in constructions of encryption schemes even when the parameters do not permit a reduction from LWE (i.e. q is not large enough). In this case, it's a separate assumption and its relationship with LWE is unclear.

The advantage of making the LWR assumption is that it permits smaller parameters due to the fact that it does not add the error $\mathbf{e}$. As an illustration, let's take a look at (19). The error $\mathbf{E}_3$ was added in (15) as an LWE error, whereas the error $\mathbf{E}'$ naturally occurred due to the rounding. The distribution of $\mathbf{E}_3'$ is essentially uniform in some range (the range depends on the set S) and so, intuitively, it might be that adding $\mathbf{E}_3$ in (15) is unnecessary. Thus under the LWR assumption, the rounding is performing two functions – it adds a uniform-like error vector and it reduces the ciphertext size. Similarly, instead of adding an error to the ciphertext $\mathbf{U}$, we can simply round it to some (different) set $\mathcal{S}$ which has the effect of adding deterministic error to $\mathbf{U}$.

2.5.3 Encrypting More Bits per Slot

Our encryption scheme packed the N-bit message into a matrix $\mathbf{M} \in \{0,1\}^{k\times \ell}$. We could have instead, for example, packed it into a matrix

$$\mathbf{M} = \{0, 1, \ldots, 2^b - 1\}^{(k/\sqrt{b})\times(\ell/\sqrt{b})}.$$

For decryption to work, we would need to make two small changes to the encryption and decryption algorithms, while also adjusting the parameters. We will replace the $\frac{q}{2}\mathbf{M}$ term in the encryption algorithm with $\frac{q}{2^b}\mathbf{M}$. The part of the decryption algorithm computing $\mathbf{V} - \mathbf{U}\mathbf{S}$ will then result in the $\frac{q}{2}$ term being similarly replaced with $\frac{q}{2^b}$ in (17). This means that in order for decryption to produce the correct result, one would need to have the remaining terms be in $[q/2^{b+1}]$ rather than $[q/4]$ as before.

The trivial modification of the parameters in order for decryption to again work would involve simply increasing q by a factor of approximately 2^{b-1}. Note that this could have a positive effect of the size of the ciphertext and public key. Take, for example, the size of the public key $256 + \ell m \log q$. If we increase q by a factor of 2^{b-1}, but decrease ℓ by a factor of 2, we will obtain $256 + \frac{\ell m}{2}(\log q + b - 1)$. In other words, the size is smaller when $b - 1 < \log q$. But increasing q while keeping eveything else constant has the effect of decreasing security (as mentioned in the beginning of Section 2.3, the problem becomes harder as the ratio β/q grows), so we will need to either increase β or increase m. The way to achieve the optimal parameters is to try a few possible options.

2.5.4 "Asymmetrical" LWE Encryption

In all the versions of public key encryption that we presented thus far, the security proof used the LWE assumption to argue that the public key is computationally indistinguishable from uniform, and then used the uniformity of the public key and the LWE assumption one more time to argue that the ciphertext is computationally indistinguishable from uniform. It is sometimes, however, useful to have the public key be truly uniform. Or, similarly, have the ciphertext be truly uniform under the (computational) assumption that the public key was. In other words, for some applications we may want to apply the LWE assumption only once. We'll now explain how to construct such a cryptosystem and also give some intuition for when something like this may actually be useful in practice.

To make the public key or the ciphertext uniform, we will need to use the *leftover hash lemma.* This lemma applied to our scenario roughly states that if $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ (where q is a prime) and $\mathbf{s} \leftarrow [\beta]^m$, where $(2\beta+1)^m \gg q^n$, then the distribution of $(\mathbf{A}, \mathbf{As})$ is statistically-close to the distribution of $(\mathbf{A}, \mathbf{u})$, where $\mathbf{u} \leftarrow \mathbb{Z}_q^n$. In other words, if $\mathbf{s}$ is chosen uniformly from some set, then the size of this set should be larger than the size of the range of the function $\mathbf{As}$. With the leftover hash lemma in hand, it's fairly easy to see how one would modify either the key generation (13) or the encryption (15) procedure to ensure that either the public key or the ciphertext are random.

If we would like the public key to be uniformly random, we replace (13) with

$$\mathsf{sk}: \mathbf{S} \leftarrow [\beta']^{m\times\ell},\ \mathsf{pk}: \left(\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{T} = \mathbf{AS}\right),\ \text{where } (2\beta'+1)^m > q^n. \tag{20}$$

Note that since we're not using the LWE assumption twice, there is no longer a reason to have β in the key generation be the same as in encryption – and so we give them different names. The encryption and decryption equation can remain exactly as in (15) and (17). When setting the parameters to make sure that decryption returns the correct answer, one should pay attention to the fact that the term $\mathbf{E}_2\mathbf{S}$ in (17) is larger due to the fact that m and/or β' are larger than before, and also to the fact that the term $\mathbf{RE}_1$ in (17) is 0 because $\mathbf{E}_1$ does not exist in the key generation procedure.

The necessary modifications for the scenario where we would like to have the ciphertext be uniformly-random (after applying the LWE assumption to conclude that the public key in (13) is indistinguishable from random) uses the exact same principle. In particular, the dimension of $\mathbf{A}$ and the distribution of $\mathbf{R}$ should be such that $([\mathbf{A} \mid \mathbf{T}], \mathbf{R}[\mathbf{A} \mid \mathbf{T}]$ where $[\mathbf{A} \mid \mathbf{T}] \leftarrow \mathbb{Z}_q^{n\times m}$ are indistinguishable from $([\mathbf{A} \mid \mathbf{T}], [\mathbf{U} \mid \mathbf{V}])$, where $\mathbf{U}, \mathbf{V}$ are uniform.

Without going into much detail, we will now touch upon why one would want to have either the public key or the ciphertext be truly uniform. The two examples we provide are by no means exhaustive, but give a flavor about where and why sacrificing some efficiency may be required. The main application of a uniform public key is in the lattice construction of identity-based encryption [GPV08]. In this scenario, the public key of a user with identity $x \in \{0,1\}^*$ is $(\mathbf{A}, \mathbf{t}_x)$, where $\mathbf{t}_x = \mathcal{H}(x)$ for a cryptographic hash function (modelled as a random oracle) that maps $\{0,1\}^*$ to $\mathbb{Z}_q^n$. In other words, $\mathbf{A}$ is common to all users, while $\mathbf{t}_x$ is unique to every user and is uniformly-random. In an IBE scheme, the public key of user x should be computable by anyone. It is therefore not possible to generate it from some secret information (e.g. like the secret key) as in (6). On the other hand, there needs to exist a way to associate a secret key with the public key. The solution to this problem is for the master authority to possess a "trap-door" to $\mathbf{A}$, which allows him to create a low-norm vector $\mathbf{s}_x$ satisfying $\mathbf{As}_x = \mathbf{t}_x$. This $\mathbf{s}_x$ is then the secret decryption key of user x. Interestingly, note that the key generation is *not* done as in (20) – in particular, the secret key is created *after* the public key. This is only possible because the master authority created $\mathbf{A}$ together with a trapdoor. Despite this difference in the order of key creation, the main result of [GPV08] provides algorithms such that the distribution of $\mathbf{A}, \mathbf{s}, \mathbf{t}$ will be the same whether one chooses $\mathbf{s}$ before computing $\mathbf{t}$ or the other way around.

An application where one would want the ciphertext to be uniformly random comes up when there is a chance that something about the ciphertext randomness $\mathbf{r}$ is leaked. It was observed in [AGV09] that one could leak something about $\mathbf{r}$, and by the leftover hash lemma, the ciphertext would still remain uniformly random.

2.6 Non-Interactive Key Exchange

The encryption schemes described in this section can be trivially converted to a passively-secure *key transport* scheme in which the two parties wish to agree on a shared symmetric key (e.g. an AES key). The protocol would simply involve the first party creating a public key (as in (13)) and sending it to the second party. The second party then chooses the AES key and encrypts it as the message $\mathbf{M}$ and sends the ciphertext (as in (15)). The first party then decrypts the shared key $\mathbf{M}$.

While the above protocol is good enough for most purposes where classical key exchange (e.g. Diffie-Hellman) is used, there is a critical order to the flow of the protocol. The user sending $\mathbf{M}$ cannot send his message before receiving the public key. In classical protocols, such as Diffie-Hellman, either user can send his g^{x_i} first, or the One can create a protocol with this property from the LWE problem, but it will be a much less efficient way of exchanging keys where this arbitrary flow property isn't needed.

We will describe this simple protocol for the case of agreeing on one random bit. To agree on more bits, one would apply the same ideas as in Section 2.4. There is a public random matrix $\mathbf{A} \in \mathbb{Z}_q^{m\times m}$ that is trusted by everyone to have been honestly generated (e.g. it is expanded by SHAKE from the seed 0). Both parties choose vectors $\mathbf{s}_1, \mathbf{s}_2, \mathbf{e}_1, \mathbf{e}_2 \leftarrow [\beta]^m$. The first party sends $\mathbf{u}_1^T = \mathbf{s}_1^T\mathbf{A} + \mathbf{e}_1^T$, while the second sends $\mathbf{u}_2 = \mathbf{A}\mathbf{s}_2 + \mathbf{e}_2$. Upon receiving the message from its respective counter-party, the first party computes $\mathbf{s}_1^T\mathbf{u}_2$ and if this value is closer to $q/2$ than to 0 (i.e. it's between $q/4$ and $3q/4$), it will set the shared bit $b_1 = 1$ (otherwise it will set $b_1 = 0$). The second party computes $\mathbf{u}_1^T\mathbf{s}_2$ and sets $b_2 = 0$ or 1 using the same rule. In short, they end up with

$$\mathbf{s}_1^T\mathbf{u}_2 = \mathbf{s}_1^T\mathbf{A}\mathbf{s}_2 + \mathbf{s}_1^T\mathbf{e}_2 \tag{21}$$

$$\mathbf{u}_1^T\mathbf{s}_2 = \mathbf{s}_1^T\mathbf{A}\mathbf{s}_2 + \mathbf{e}_1^T\mathbf{s}_2. \tag{22}$$

and hope that the error terms terms $\mathbf{s}_1^T\mathbf{e}_2$ and $\mathbf{e}_1^T\mathbf{s}_2$ (which have maximum magnitude $m\beta^2$) don't cause $b_1 \neq b_2$. Notice that the probability that $b_1 \neq b_2$ is at most the probability that $\mathbf{s}_1^T\mathbf{A}\mathbf{s}_2$ falls into the "dangerous" ranges near $3q/4$ and $q/4$. In particular,

$$\Pr[b_1 \neq b_2] < \Pr\left[\mathbf{s}_1^T\mathbf{A}\mathbf{s}_2 \in \left[\frac{3q}{4} + m\beta^2, \frac{3q}{4} - m\beta^2\right] \text{ or } \left[\frac{q}{4} + m\beta^2, \frac{q}{4} - m\beta^2\right]\right]. \tag{23}$$

The probability that $\mathbf{s}_1^T\mathbf{A}\mathbf{s}_2$ is any partiular value in $\mathbb{Z}_q$ is $1/q$ and so the above probability is at most $4m\beta^2/q$. In practice, one would use the techniques from Section 2.3.2 to reduce this probability, but one would still end up with a probability of $\Omega(\beta^2\sqrt{m}/q)$ of a mismatch in b_1 and b_2. This is quite different than the situation we had with the encryption schemes where we could set parameters so that the error is exponentially small (or even 0) with respect to q. In the non-interactive key agreement, getting a small error will require setting q to be very large which has an adverse effect on the communication size.

3 Hardness of LWE and Other Lattice Problems

We will now give a geometric view of the LWE problem. While this connection is not really necessary for understanding how most cryptographic constructions work, it is crucial for understanding their security.

3.1 Lattices

At the core of the geometric interpretation of the LWE problem are objects known as *lattices.* An m-dimensional integer lattice Λ is simply a subgroup of the group $(\mathbb{Z}^m, +)$. Such a group can be described via a generating set called a basis. In particular, a lattice Λ defined by a (full-rank) basis $\mathbf{B} \in \mathbb{Z}^{m\times m}$ is

$$\Lambda = \mathcal{L}(\mathbf{B}) = \{\mathbf{v} \in \mathbb{Z}^m \ : \ \exists \mathbf{z} \in \mathbb{Z}^m \text{ s.t. } \mathbf{Bz} = \mathbf{v}\}. \tag{24}$$

In this chapter, we will just restrict ourselves to special types of lattices called *q-ary integer lattices*, as these are the ones that are used in cryptographic constructions. They also have the nice theoretical property that, asymptotically, solving some problem over random instances of these lattices is as hard as solving some problem for *any* lattice. This is the celebrated worst-case to average-case reduction line of research [Ajt96, Reg09] that formed the foundation, and spearheaded the development, of lattice-based cryptography.
For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, the q-ary lattice Λ defined by $\mathbf{A}$ is

$$\Lambda = \mathcal{L}_q^\perp(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^m \ : \ \mathbf{Av} \equiv 0 \pmod q\} \tag{25}$$

It's not hard to see that under the usual vector addition operation, the above set is a group. For reader's familiar with linear codes, the above two definitions of lattices are akin to describing a code using a generating matrix (24) or a parity-check matrix (25). Even more specifically, the lattices that we will be dealing with are

$$\Lambda = \mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{I}_n]), \tag{26}$$

where $\mathbf{I}_n$ is the $n \times n$ identity matrix. This is not much of a restriction because if the $\mathbf{A}$ in (25) contains n columns that are linearly independent over $\mathbb{Z}_q$ (without loss of generality, suppose that $\mathbf{A} = [\mathbf{A}_1 \mid \mathbf{A}_2]$ where $\mathbf{A}_2 \in \mathbb{Z}_q^{n\times n}$ is invertible), then we can write $\mathbf{A}_2^{-1}\mathbf{A} = [\mathbf{A}_2^{-1}\mathbf{A}_1 \mid \mathbf{I}]$ and $\mathcal{L}_q^\perp(\mathbf{A}) = \mathcal{L}_q^\perp(\mathbf{A}_2^{-1}\mathbf{A})$, where the latter is in the form of (26). For lattices in the form of (26), it is also easy to switch between the "generator" matrix representation in (24) and the "parity check" matrix representation in (25). It's an easy exercise to check that

$$\mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{I}_n]) = \mathcal{L}\left(\begin{bmatrix} -\mathbf{I}_m & \mathbf{0} \\ \mathbf{A} & q\mathbf{I}_n \end{bmatrix}\right). \tag{27}$$

3.1.1 The Quotient Group and Determinant

The determinant of a full-rank lattice $\Lambda \subseteq \mathbb{Z}^m$, written as $\det(\Lambda)$, is the inverse of the density of Λ in the space $\mathbb{Z}^m$. That is, if we define the set $S_r = \{\mathbf{z} \in \mathbb{Z}^m \ : \ \|\mathbf{z}\| < r\}$, then

$$\det(\Lambda) = \lim_{r\to\infty} \frac{|S_r|}{|\Lambda \cap S_r|}.$$

If $\Lambda = \mathcal{L}(\mathbf{B})$ for a full-rank matrix $\mathbf{B} \in \mathbb{Z}^{m\times m}$, then $\det(\Lambda) = \det(\mathbf{B})$, where the right-hand side is the usual matrix determinant of $\mathbf{B}$. For example, the determinant of the $(n+m)$-dimensional lattice in (27) is $\det(\Lambda) = \det(\mathbf{B}) = q^n$. Another equivalent definition of the determinant of a full-rank m-dimensional lattice Λ is the size of the quotient group $\mathbb{Z}^m/\Lambda$.
The parity-check representation of a lattice, $\Lambda = \mathcal{L}_q^\perp(\mathbf{A})$, is convenient for checking whether two vectors in $\mathbb{Z}^m$ are in the same coset of $\mathbb{Z}^m/\Lambda$. In particular, $\mathbf{z}_1$ and $\mathbf{z}_2$ are in the same coset if and only if $\mathbf{Az}_1 \equiv \mathbf{Az}_2 \pmod q$. With this observation it's easy to see that when $\Lambda = \mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{I}_n])$, there are exactly q^n cosets; which is consistent with our previous observation that the determinant of the lattice in (27) is q^n.

3.1.2 Distance to the Lattice

For an m-dimensional lattice Λ and any vector $\mathbf{r} \in \mathbb{Z}^m$ (not necessarily in Λ), the ℓ_p-norm distance from $\mathbf{r}$ to the lattice is defined as

$$\Delta_p(\mathbf{r}, \Lambda) = \min_{\mathbf{v}\in\Lambda} \|\mathbf{v} - \mathbf{r}\|_p. \tag{28}$$

Observe that for any two elements $\mathbf{r}_1$ and $\mathbf{r}_2$ belonging to the same coset of $\mathbb{Z}^m/\Lambda$, $\Delta_p(\mathbf{r}_1, \Lambda) = \Delta_p(\mathbf{r}_2, \Lambda)$, and so distances are well-defined notions for cosets as well. Therefore if $\Lambda = \mathcal{L}_q^\perp(\mathbf{A})$ and $\mathbf{t} \equiv \mathbf{A}\mathbf{z} \pmod q$ defines a coset $\mathbf{z} + \Lambda$, then we write

$$\Delta_p^C(\mathbf{t}, \Lambda) = \Delta_p(\mathbf{z}, \Lambda).$$

For convenience, we write Δ^C instead of Δ to denote that $\mathbf{t}$ is the image of the coset under $\mathbf{A}$, rather than some coset representative.
We will now prove some statements about the (non)-existence of short vectors in random lattices. Lemmas 1 and 2 show that random cosets are far from random q-ary lattices and that q-ary lattices don't have very short vectors. Lemma 3 proves a partial converse, giving a lower bound on the length of the shortest vector in any q-ary lattice.

Lemma 1. *For any q and any $\mathbf{t} \in \mathbb{Z}_q^n$ that has a coefficient t_i such that* $\gcd(t_i, q) = 1$,

$$\Pr_{\mathbf{A}\leftarrow\mathbb{Z}_q^{n\times m}}[\exists \mathbf{z} \in [\beta]^{n+m} \textit{ s.t. } [\mathbf{A} \mid \mathbf{I}_n]\mathbf{z} \equiv \mathbf{t} \pmod q] \le (2\beta + 1)^{n+m}/q^n$$

Proof. Since $\mathbf{t}$ has some coefficient relatively prime to q, it must also be the case that some coefficient of $\mathbf{z}$ must also be relatively prime to q. Without loss of generality, assume that it's the first one. Then we have that for a fixed $\mathbf{z}$,

$$\begin{aligned}\Pr_{\mathbf{A}\leftarrow\mathbb{Z}_q^{n\times m}}[[\mathbf{A} \mid \mathbf{I}_n]\mathbf{z} \equiv \mathbf{t} \pmod q] &= \Pr_{\mathbf{a}\leftarrow\mathbb{Z}_q^n, \mathbf{A}'\leftarrow\mathbb{Z}_q^{n\times(m-1)}}\left[[\mathbf{a} \mid \mathbf{A}' \mid \mathbf{I}_n]\begin{bmatrix} z_1 \\ \mathbf{z}' \end{bmatrix} \equiv \mathbf{t} \pmod q\right] \\ &= \Pr_{\mathbf{a}\leftarrow\mathbb{Z}_q^n}[\mathbf{a}z_1 \equiv \mathbf{t} - [\mathbf{A}' \mid \mathbf{I}_n]\mathbf{z}' \pmod q] \\ &= \Pr_{\mathbf{a}\leftarrow\mathbb{Z}_q^n}[\mathbf{a} \equiv z_1^{-1}(\mathbf{t} - [\mathbf{A}' \mid \mathbf{I}_n]\mathbf{z}') \pmod q] = q^{-n},\end{aligned}$$

where $z^{-1} \pmod q$ exists because we assumed that $\gcd(z_1, q) = 1$. Since there are $(2\beta + 1)^{n+m}$ possible vectors in $[\beta]^{n+m}$, the statement in the lemma follows by the union bound.

Corollary 1.

$$\Pr_{\mathbf{A}\leftarrow\mathbb{Z}_q^{n\times m}, \mathbf{t}\leftarrow\mathbb{Z}_q^n}[\Delta_\infty^C(\mathbf{t}, \Lambda) \le \beta] \le (1 - |\mathbb{Z}_q^*|/q)^n + (2\beta + 1)^{n+m}/q^n,$$

where $\Lambda = \mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{I}_n])$.

Some "popular" settings for q in lattice cryptography are prime q and q that are powers of 2. In both of these cases, the first term in the probability bound is negligible in n ($(1/q)^n$ and 2^{-n}, respectively). And so, whenever $\beta^{1+m/n} \ll q$, random cosets will be more than distance β away from Λ.
The next lemma states that the probability, over the choice of $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$, that the lattice $\mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{I}_n])$ has a short non-zero vector is small. We only prove this lemma for a prime q, as it's somewhat more messy for other choices. The proof of the lemma is virtually identical to that of Lemma 1.

Lemma 2. *For any prime q,*

$$\Pr_{\mathbf{A}\leftarrow\mathbb{Z}_q^{n\times m}}[\exists \mathbf{z} \in [\beta]^{n+m} \setminus \{\mathbf{0}\} \textit{ s.t. } [\mathbf{A} \mid \mathbf{I}_n]\mathbf{z} \equiv \mathbf{0} \pmod q] \le (2\beta + 1)^{n+m}/q^n$$

The next Lemma is a converse of the one above; it gives a lower bound on the length of an existing non-zero vector.

Lemma 3. *For any q and any $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$,*

$$\exists \mathbf{z} \in \left[q^{n/(n+m)}\right]^{n+m} \setminus \{\mathbf{0}\} \ s.t.\ [\mathbf{A} \mid \mathbf{I}_n]\mathbf{z} \equiv \mathbf{0} \pmod q$$

Proof. The proof is by the pigeonhole principle. There are more than $(q^{n/(n+m)})^{n+m} = q^n$ vectors in $\mathbb{Z}^{n+m}$ whose coefficients are between 0 and $q^{n/(n+m)}$. Since there are only q^n possibilities for the value of $\mathbf{Az} \bmod q$, there must exist two $\mathbf{z}_1, \mathbf{z}_2$ with coefficients in the aforementioned range such that $\mathbf{Az}_1 \equiv \mathbf{Az}_2 \pmod q$. Thus $\mathbf{z}_1 - \mathbf{z}_2 \in \left[q^{n/(n+m)}\right]^{n+m}$ and $\mathbf{A}(\mathbf{z}_1 - \mathbf{z}_2) \equiv \mathbf{0} \pmod q$.

Looking at the statements of Lemmas 2 and 3, we see that the boundary between there existing a vector in $\Lambda = \mathcal{L}_q^{\perp}([\mathbf{A} \mid \mathbf{I}_n])$ with coefficients in $[\beta]$ and there not being one with high probability is pretty sharp. Lemma 3 states that when $\beta = q^{n/(n+m)}$, such a vector always exists. On the other hand, if we set $\beta < \frac{1}{4}q^{n/(n+m)}$, then the probability of there being a vector in Λ with coefficients in $[\beta]$ is less than $2^{-(n+m)}$.

3.2 Finding Short Vectors in Random Lattices (the SIS Problem)

A fundamental computational question one can ask about a lattice is to find a "short" (non-zero) vector in it. When specifically tailored to the lattices we've been dealing with above, the question simply becomes to find a non-zero $\mathbf{z} \in [\beta]^{n+m}$ such that $[\mathbf{A} \mid \mathbf{I}_n]\mathbf{z} \equiv \mathbf{0} \pmod q$. Lemma 3 states that when $\beta = q^{n/(n+m)}$, such a vector surely exists, but the proof does not give us any way of finding it. As of today, all known (quantum) algorithms for finding such vectors (for uniformly random $\mathbf{A}$) take $2^{\Omega(m+n)}$ time (c.f. [AKS01, ADRS15, AS18]).
The problem does get easier as β increases. Clearly if $\beta = q/2$, then the problem is trivially solved by just setting the coefficients of $\mathbf{z}$ that are being multiplied by $\mathbf{I}_n$ to the target coefficients. For smaller value of β, one would run an algorithm that finds a vector in the lattice that is some factor larger than the smallest vector. All modern efficient (i.e. polynomial-time) algorithms for finding such short vectors are descendants of the famous LLL algorithm [LLL82] which guarantees to find a vector of length at most $2^{O(n+m)}$ times larger than that of the shortest vector in the lattice. Lemma 3 then implies that for a random $\mathbf{A}$, the LLL algorithm will find a vector $\mathbf{z} \in \left[2^{\Omega(n+m)} \cdot q^{n/(n+m)}\right]^{n+m}$ in $\mathcal{L}_q^{\perp}([\mathbf{A} \mid \mathbf{I}_n])$.
While the length of the vector the LLL algorithm is guaranteed to find is exponentially larger (in the dimension of the lattice) than the length of the shortest vector, in practice, this exponent is not too large. Until experiments were run with lattices of sufficiently high dimension (at least 100), it wasn't even clear whether the real approximation factor of LLL was exponential or just linear. As it turns out, the approximation factor is indeed exponential in the dimension, but the base of the exponent is rather small. Experiments in [GN08] showed that one can find vectors in random lattices Λ of the form (26) (of dimension $m + n$) of length approximately

$$\min\left\{q, \det(\Lambda)^{1/(n+m)} \cdot \delta^{n+m}\right\} = \min\left\{q, q^{n/(n+m)} \cdot \delta^{n+m}\right\} \tag{29}$$

where δ depends on how much time the algorithm takes. As a very rough rule of thumb, $\delta = 1.01$ is considered within reach, whereas $\delta = 1.005$ may never be achieved for lattices of high-enough dimension (e.g. more than 500).
Note that if the minesion of the lattice Λ is very large, one can just remove arbitrarily many columns from $\mathbf{A}$ and run LLL on the resulting lattice. In particular, it is optimal to have the dimension of the lattice be $\sqrt{n \log q / \log \delta}$ (see [MR08, Chapter 3]), which results in the size of the found vector being

$$\min\left\{q, 2^{2\sqrt{n \log q \log \delta}}\right\}. \tag{30}$$

The problem of finding a short vector in a random lattice as in (26) is called the SIS (Short Integer Solution) problem. It is known that solving random instances of this problem is at least as hard as solving some related problem in all lattices [Ajt96, MR07]. We will write $\mathsf{SIS}_{n,m,q,\beta}$ to be the problem of finding a vector with coefficients in $[\beta]$ when given a random lattice as in (26). Notice that by Lemma 2, the $\mathsf{SIS}_{n,m,q,\beta}$ problem is vacuously hard when $\beta \ll \frac{1}{2}q^{n/(n+m)}$ and (30) states that the problem gets easier as β grows.
Also note from (30) that once m is larger than $\sqrt{n \log q / \log \delta}$, then it has no impact on the hardness of the problem since it does not figure in the formula for the size of the found vector. This is quite similar to the situation in the $\mathsf{LWE}_{n,m,q,\beta}$ problem, where the parameter n does not matter much. And as in that case, we will just write $\mathsf{SIS}_{n,q,\beta}$.

3.3 The LWE Lattices

Let us now rephrase the $\mathsf{LWE}_{n,m,q,\beta}$ problem from Section 2.3 in the language of lattices. If we choose a random $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ and random $\mathbf{s} \leftarrow [\beta]^m, \mathbf{e} \leftarrow [\beta]^n$, and output $(\mathbf{A}, \mathbf{t} = \mathbf{As} + \mathbf{e})$, then this is the same as outputting a lattice $\Lambda = \mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{I}_n)$ for a random $\mathbf{A}$ and a coset $\mathbf{t}$ in $\mathbb{Z}_q^m/\Lambda$ such that $\Delta_\infty^C(\mathbf{t}, \Lambda) \leq \beta$. On the other hand, outputting $(\mathbf{A}, \mathbf{u})$ for a random $\mathbf{u} \leftarrow \mathbb{Z}_q^n$ is akin to outputting the lattice Λ and a random coset of $\mathbb{Z}^m/\Lambda$. The $\mathsf{LWE}_{n,m,q,\beta}$ problem can therefore be restated as trying to distinguish between cosets that are close to the lattice and random cosets.
Our encryption scheme had $m = n$ and for correctness we needed $\beta^2 = O(q/\sqrt{m})$, and therefore $\beta \ll \sqrt{q}$. From Corollary 1, this implies that random cosets will be further away than β from the lattice. Thus the $\mathsf{LWE}_{n,m,q,\beta}$ problem for parameters that make the encryption scheme work can be seen as distinguishing between cosets that are close to the lattice with those that are far away.
When $\mathbf{t} = \mathbf{As} + \mathbf{e}$, it implies that there is a vector $\mathbf{z} \in [\beta]^{n+m+1}$ in the lattice

$$\mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{t} \mid \mathbf{I}_n]). \tag{31}$$

The LLL algorithm is then guaranteed to find a vector with coefficients in $\left[\delta^{n+m+1} \cdot \beta\right]$. If this bound is less than $q^{n/(n+m+1)}$, then by Lemma 2 we are able to distinguish lattices in (31) when $\mathbf{t} = \mathbf{As}+\mathbf{e}$ from lattices in (31) when $\mathbf{t}$ is uniformly random. Just like for the algorithm in the previous section, the more time one is willing to spend on the LLL-type algorithm, the smaller δ can be. So the LWE problem gets harder the larger β is. Once β is large-enough (approximately $q^{n/(n+m)}$), the distribution of $\mathbf{t} = \mathbf{As} + \mathbf{e}$ will actually become uniformly-random and so distinguishing this distribution from uniform becomes vacuously hard.
The way the LLL-type algorithms work is by finding short vectors in lower-dimensional "projected lattices". The larger the dimension of these lattices we take, the longer the algorithm needs to run, and the smaller the δ will be. It's important to note that if we take a projected dimension that's too small, then the short vector $\begin{bmatrix} \mathbf{s} \\ -1 \\ \mathbf{e} \end{bmatrix}$ of the lattice in (31) will have *no effect* on the length of the short vector that's found, and the algorithm will act exactly as if $\mathbf{t}$ were uniform and thus find a vector of length dictated by the equation in (29).
Deducing a good approximation to the running time of an LLL-type algorithm for a particular value of δ is fairly involved (c.f. [ACD+18, ADH+19]) and is out of scope of this survey. The key concept that we need to know for constructing lattice-based primitives is that when $\mathbf{t} = \mathbf{As} + \mathbf{e}$, then the problem of finding a short vector (or even distinguishing the lattice from uniform when $\beta < q^{n/(n+m)}$) in the lattice in (31) becomes harder as β grows for $0 < \beta < q^{n/(n+m)}$, and then becomes easier as β grows for $q^{n/(n+m)} < \beta < q$. The former range of β corresponds to the LWE problem, whereas the latter range is SIS. For constructing encryption schemes, which are just based on LWE, we simply want to set β as large as possible that will still allow decryption. When building our signature scheme in Section 5, the hardness

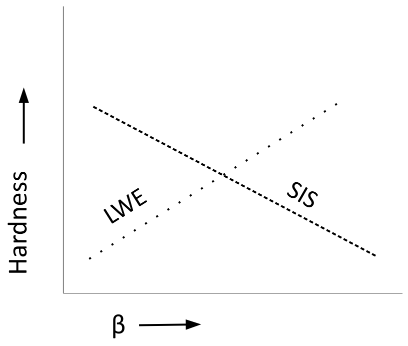

Figure 2: The hardness of $\mathsf{LWE}_{n,m,q,\beta}$ and $\mathsf{SIS}_{n,m,q,\beta}$ for fixed n, m, q, and varying β. The lines are not meant to describe the concrete hardness of these problems, but rather to illustrate that if a cryptographic scheme has its hardness based on both $\mathsf{LWE}_{n,m,q,\beta}$ and $\mathsf{SIS}_{n,m,q,\beta'}$ for $\beta' = \kappa\beta$ for $\kappa > 1$, then the optimal parameter setting is to pick β, β' such that the hardness of the two problems is equivalent.

Table 1: Approximate (conservative) hardness of the $\mathsf{LWE}_{m,q,\beta}$ and $\mathsf{SIS}_{n,q,\beta}$ problems against quantum algorithms for some parameters that resemble those used in encryption / signature schemes.

$\mathsf{LWE}_{m,q,\beta}$ Parameters			
m	β	q	Security
512	2	2^{13}	110
768	2	2^{13}	160
1024	2	2^{13}	210
768	6	2^{23}	90
1024	5	2^{23}	128
1280	3	2^{23}	160

$\mathsf{SIS}_{n,q,\beta}$ Parameters			
n	β	q	Security
1024	2^{20}	2^{23}	95
1280	2^{20}	2^{23}	125
1536	2^{20}	2^{23}	160

of the SIS problem and its relationship to LWE will become important for optimal parameter settings, as we describe below.

The lattice problem that the signature will be based on is a hybrid between LWE and SIS. Suppose that we are given a lattice as in (31) where $\mathbf{t} = \mathbf{As} + \mathbf{e}$ with $\mathbf{s}, \mathbf{e}$ having coefficients in $[\beta]$ and then asked to find a vector $\mathbf{z} \in [\beta']^{n+m+1}$, where $\beta' \geq \beta$ in this lattice. If the $\mathsf{LWE}_{n,m,q,\beta}$ problem is hard, then the lattice in (31) is indistinguishable from the lattice

$$\mathcal{L}_q^{\perp}([\mathbf{A}' \mid \mathbf{I}_n]), \tag{32}$$

where $\mathbf{A}' \leftarrow \mathbb{Z}_q^{n\times(m+1)}$, and then finding a $\mathbf{z} \in [\beta']^{n+m+1}$ in this lattice becomes the $\mathsf{SIS}_{n,m+1,q,\beta'}$ problem. So in order to make the whole problem hard, we need $\mathsf{LWE}_{n,m,q,\beta}$ and $\mathsf{SIS}_{n,m+1,q,\beta'}$ to be hard. So the optimal setting of parameters will be exactly when these two problems are equally hard. An illustration of this appears in Figure 2. In Table 1, we give some representative parameters for the $\mathsf{LWE}_{m,q,\beta}$ and $\mathsf{SIS}_{n,q,\beta}$ problems that are similar to those used in encryption and signatures schemes based on the framework in this tutorial, and their security levels.

4 Encryption Over Polynomial Rings

The main inefficiency with the LWE-based encryption scheme in Section 2.3 was that it required a fairly large ciphertext for encrypting one bit – in particular, the ciphertext expansion was linear in the security parameter. This inefficiency was somewhat mitigated by the scheme in Section 2.4 by making the ciphertext expansion only square root in the security parameter at the expense of blowing up the public key by the same factor. In this section, we will show how to get rid of this square root blow-up by considering the LWE problem not over $\mathbb{Z}_q$, but over higher degree polynomial rings.

4.1 Polynomial Rings

The polynomial ring $(\mathbb{Z}[X], +, \times)$, with an indeterminate X, consists of elements of the form $a(X) = \sum\limits_{i=0}^{\infty} a_i X^i$, for $a_i \in \mathbb{Z}$, with the usual polynomial addition and multiplication operations. For convenience, we will often omit the indeterminate X, and simply write a instead of $a(X)$. The degree of a, denoted $\deg(a)$, is the largest i for which $a_i \neq 0$. A polynomial a is *monic* if $a_{deg(a)} = 1$ and it is irreducible if it cannot be written as $a = bc$ where $b, c \in \mathbb{Z}[X]$ and $\deg(b), \deg(c) < \deg(a)$.

The encryption schemes that we saw in Section 2 involved operations over the ring $(\mathbb{Z}, +, \times)$. A generalization of this ring with which we will be working with for the remainder of the chapter is the ring $(\mathcal{R}_f, +, \times)$, where $f \in \mathbb{Z}[X]$ is a monic polynomial of degree d.[2] The elements of $\mathcal{R}_f$ are the polynomials $a = \sum\limits_{i=0}^{d-1} a_i X^i$, where $a_i \in \mathbb{Z}$. The sum of two elements in $\mathcal{R}_f$ simply involves summing the corresponding coefficients in $\mathbb{Z}$. That is,

$$a + b = \sum_{i=0}^{d-1} (a_i + b_i) X^i.$$

So the addition of polynomials in $\mathcal{R}_f$ can be seen as addition of vectors over $\mathbb{Z}^d$. Multiplication of a polynomial by an element in $\mathbb{Z}$ therefore also has the same interpretation as multiplying a vector by a constant.

Multiplication of two polynomials in $\mathcal{R}_f$ involves performing a normal polynomial multiplication followed by a reduction modulo f. Reduction modulo f means (just like for integers), the remainder after a division by f is performed. In particular, any polynomial $a \in \mathcal{R}_f$ can be uniquely written as $a = bf + r$ for $b, r \in \mathcal{R}_f$ where $\deg(r) < d$. And so $a \bmod f = r$.

To see that any a can indeed be decomposed in this fashion, we use induction. If $\deg(a) < d$, then $a = r$ (and $b = 0$) and we're done. Now suppose that all a of degree $k - 1$ can be written as $a = bf + r$ and let a' have degree k. Then $a' - a'_k f X^{k-d}$ has degree $k - 1$ and by the inductive hypothesis can be written as $bf + r$ for some b and r. We can therefore write $a' = \left(a' X^{k-d} + b\right) f + r$.

To prove that the above decomposition of a is unique, assume for contradiction that there are distinct $(b, r) \neq (b', r')$ such that $bf + r = b'f + r'$. This implies that $(b - b')f + (r - r') = 0$. Since $\deg(r - r') < d$, it must be that $r = r'$. Then we know that if $(b - b') \neq 0$, then $\deg((b - b')f) = deg(b - b') + deg(f) \neq 0$. And so $b = b'$ and we have a contradiction.

Note that the above proof of existence of b and r gives rise to a very simple algorithm for computing $a \bmod f$ – multiply f by an appropriate monomial αX^i and subtract it away from a to create a polynomial of a smaller degree and continue until getting a polynomial of degree less than d. For example if we would like to reduce $2X^3 +$

[2] In the lattice literature, this ring is often written as $\mathbb{Z}[X]/(f(X))$.

$8X^2 + 5X + 1$ modulo $f = X^2 - 2X + 1$, we write

$$\begin{aligned} 2X^3 + 8X^2 + 5X + 1 &\equiv 2(2X^2 - X) + 8X^2 + 5X + 1 \equiv 12X^2 + 3X + 1 \pmod f \\ &\equiv 12(2X-1) + 3X + 1 \equiv 27X - 11 \pmod f. \end{aligned}$$

As a simple observation, note that the usual ring $(\mathbb{Z}, +, \times)$ is a special instantiation of the ring $(\mathcal{R}_f, +, \times)$ in which the polynomial f is defined to be $f = X$ (in fact $f = X - \alpha$, for any $\alpha \in \mathbb{Z}$ is also fine).

4.1.1 Polynomials and Linear Algebra

A useful observation is that polynomial multiplication modulo f can be written as a multiplication of a matrix in $\mathbb{Z}^{d\times d}$ with a vector in $\mathbb{Z}^d$. Observe that the product $ab \bmod f$ can be written as:

$$ab \bmod f = a \cdot \left(\sum_{i=0}^{d-1} b_i X^i\right) \bmod f = \sum_{i=0}^{d-1} (aX^i \bmod f) b_i. \tag{33}$$

Since each $aX^i \bmod f$ is a polynomial of degree less than d, it can be thought of as a vector in $\mathbb{Z}^d$. The multiplication ab can therefore be seen as a linear combination (with weights b_i) of these d vectors, and thus can be represented as a matrix-vector multiplication. For example, the product

$$(2X^2 - 1)(X^2 - X + 2) \bmod X^3 - X + 1 = 5X^2 - 3X$$

can be written as

$$\begin{bmatrix} -1 & -2 & 0 \\ 0 & 1 & -2 \\ 2 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -3 \\ 5 \end{bmatrix}. \tag{34}$$

When treating polynomials $a = \sum_{i=0}^{d-1} a_i X^i \in \mathcal{R}_f$ as vectors and matrices, it will be convenient to use the following notation $\mathcal{V}_a \in \mathbb{Z}^d$ and $\mathcal{M}_a \in \mathbb{Z}^{d\times d}$:

$$\mathcal{V}_a = \begin{bmatrix} a_0 \\ a_1 \\ \cdots \\ a_{d-1} \end{bmatrix} \in \mathbb{Z}^d \text{ , and } \mathcal{M}_a = \begin{bmatrix} \mathcal{V}_a & \mathcal{V}_{aX \bmod f} & \cdots & \mathcal{V}_{aX^{d-1} \bmod f} \end{bmatrix} \in \mathbb{Z}^{d\times d} \tag{35}$$

We did not include f as part of the notation for $\mathcal{V}_a$ and $\mathcal{M}_a$, but the f should always be evident from context.
Using this notation, we can rewrite (34) as

$$\mathcal{M}_{2X^2-1} \cdot \mathcal{V}_{X^2-X+2} = \mathcal{V}_{5X^2-3X}.$$

We can also extend the above notation to matrices of polynomials. For a vector $\mathbf{a} = \begin{bmatrix} a_1 \\ \cdots \\ a_n \end{bmatrix} \in \mathcal{R}_f^n$ and matrix $\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,m} \\ \cdots & \cdots & \cdots \\ a_{n,1} & \cdots & a_{n,m} \end{bmatrix} \in \mathcal{R}_f^{n\times m}$, we define $\mathcal{V}_{\mathbf{a}}$ and $\mathcal{M}_{\mathbf{A}}$ as

$$\mathcal{V}_{\mathbf{a}} = \begin{bmatrix} \mathcal{V}_{a_1} \\ \cdots \\ \mathcal{V}_{a_n} \end{bmatrix} \in \mathbb{Z}^{dn} \text{ , and } \mathcal{M}_{\mathbf{A}} = \begin{bmatrix} \mathcal{M}_{a_{1,1}} & \cdots & \mathcal{M}_{a_{1,m}} \\ \cdots & \cdots & \cdots \\ \mathcal{M}_{a_{n,1}} & \cdots & \mathcal{M}_{a_{n,m}} \end{bmatrix} \in \mathbb{Z}^{dn\times dm}. \tag{36}$$

From the above definitions, one can check that for any $\mathbf{A} \in \mathcal{R}_f^{n\times m}$ and $\mathbf{b} \in \mathcal{R}_f^m$, we have

$$\mathcal{M}_{\mathbf{A}} \cdot \mathcal{V}_{\mathbf{b}} = \mathcal{V}_{\mathbf{Ab}} \in \mathbb{Z}^{dn} \tag{37}$$

4.1.2 Coefficient Growth

When a and b are integers, the magnitude of their product is trivial to compute – it's just the absolute value of ab. For $a, b \in \mathcal{R}_f$, bounding the magnitude of the product is a bit more involved and is heavily dependent on the polynomial f. Because multiplication $ab \in \mathcal{R}_f$ can be written as $\mathcal{M}_a\mathcal{V}_b$, a simple bound on the maximum coefficient of the product is $d\|\mathcal{M}_a\|_\infty \cdot \|\mathcal{V}_b\|_\infty$, where $\|\cdot\|_\infty$ is the absolute value of the largest coefficient. One could obtain better bounds (in the ℓ_2 norm) by computing the maximum singular value of $\mathcal{M}_a$. But either way, the effect of f on the size of the coefficients in $\mathcal{M}_a$ is crucial to bounding the product.
The best we could hope for, in terms of keeping $\mathcal{M}_a$ small, is that $\|\mathcal{M}_a\|_\infty = \|\mathcal{V}_a\|_\infty$. The only two polynomials f for which this holds are $X^d \pm 1$. For polynomials of the form $X^d \pm X^{d/2} + 1$ and $\sum_{i=0}^{d} X^i$, we have $\|\mathcal{M}_a\|_\infty \le 2\|\mathcal{V}_a\|_\infty$. Here are some example matrices $\mathcal{M}_a$ for a polynomial $a = a_0 + a_1X + a_2X^2 + a_3X^3$ for several f's of degree 4.

$$f = X^4 - 1 \longrightarrow \mathcal{M}_a = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \tag{38}$$

$$f = X^4 + 1 \longrightarrow \mathcal{M}_a = \begin{bmatrix} a_0 & -a_3 & -a_2 & -a_1 \\ a_1 & a_0 & -a_3 & -a_2 \\ a_2 & a_1 & a_0 & -a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \tag{39}$$

$$f = X^4 - X^2 + 1 \longrightarrow \mathcal{M}_a = \begin{bmatrix} a_0 & -a_3 & -a_2 & -a_1 - a_3 \\ a_1 & a_0 & -a_3 & -a_2 \\ a_2 & a_1 + a_3 & a_0 + a_2 & -a_3 + a_1 \\ a_3 & a_2 & a_1 + a_3 & a_0 + a_2 \end{bmatrix} \tag{40}$$

$$f = X^4 + X^3 + X^2 + X + 1 \longrightarrow \mathcal{M}_a = \begin{bmatrix} a_0 & -a_3 & -a_2 + a_3 & -a_1 + a_2 \\ a_1 & a_0 - a_3 & -a_2 & -a_1 + a_3 \\ a_2 & a_1 - a_3 & a_0 - a_2 & -a_1 \\ a_3 & a_2 - a_3 & a_1 - a_2 & a_0 - a_1 \end{bmatrix} \tag{41}$$

There are also polynomials f for which $\|\mathcal{M}_a\|_\infty \gg \|\mathcal{V}_a\|_\infty$. Unsurprisingly, if f itself has large coefficients, then $\|\mathcal{M}_a\|$ will as well. But there are also f with small coefficients that produce $\mathcal{M}_a$'s with exponentially larger coefficients than a – one such example is $f = X^d + 2X^{d-1} + 1$. Polynomials f that result in matrices $\mathcal{M}_a$ having much larger coefficients than a are not useful for cryptographic purposes. In general, we prefer to use polynomials that result in the ratio between $\|\mathcal{M}_a\|_\infty$ and $\|\mathcal{V}_a\|_\infty$ to be 1 or 2.

4.2 The Generalized-LWE Problem

Let us extend the notation from Section 2 and for a polynomial $a = \sum_{i=0}^{d} a_iX^i \in \mathcal{R}_f$, we write $a \leftarrow [\beta]$ to mean that all integer coefficients a_i are chosen uniformly from $[\beta]$. Similarly, for a vector $\mathbf{a} \in \mathcal{R}_f^m$, we write $\mathbf{a} \leftarrow [\beta]^m$ to be the distribution in which every coefficient of every polynomial in $\mathbf{a}$ is chosen uniformly from $[\beta]$.
We now present a version of the LWE problem that is defined over general rings $\mathcal{R}_f$, rather than just over $\mathbb{Z}$ as in Definition 1. Analogously, we will be working over the ring $\mathcal{R}_{q,f}$, which is like the ring $\mathcal{R}_f$ except that the polynomial coefficients are in $\mathbb{Z}_q$ rather than in $\mathbb{Z}$. The ring $\mathcal{R}_{q,f}$ is often written in the lattice literature as $\mathbb{Z}_q[X]/(f(X))$.

Definition 2. For positive integers $m, n, q, \beta \ll q$, and ring $\mathcal{R}_{q,f}$, the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ problem asks to distinguish between the following two distributions:

1. $(\mathbf{A}, \mathbf{As}+\mathbf{e})$, where $\mathbf{A} \leftarrow \mathcal{R}_{q,f}^{n\times m}, \mathbf{s} \leftarrow [\beta]^m, \mathbf{e} \leftarrow [\beta]^n$
2. $(\mathbf{A}, \mathbf{u})$, where $\mathbf{A} \leftarrow \mathcal{R}_{q,f}^{n\times m}$ and $\mathbf{u} \leftarrow \mathcal{R}_{q,f}^{n}$.

As before, the parameter n does not have any known effect on the hardness of the problem, unless it is large, and so we will usually just write $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{m,\beta}$. The preceding definition of the generalized LWE problem and the following cryptosystem follows the line of work (which related its security to worst-case instances of certain lattice problems) of [LPR10, BV11, LPR13b, LS15].

4.3 Generalized-LWE Encryption

The description of the encryption scheme is virtually identical to that in Section 2.3.1 with the ring $\mathbb{Z}$ being replaced with $\mathcal{R}_f$. The main advantage of the scheme will be that the message μ, being in $\mathcal{R}_f$, allows us to pack d bits into it.

$$\mathsf{sk}:\ \mathbf{s} \leftarrow [\beta]^m,\ \mathsf{pk}:\ (\mathbf{A} \leftarrow \mathcal{R}_{q,f}^{m\times m}, \mathbf{t} = \mathbf{As}+\mathbf{e}_1),\ \text{where } \mathbf{e}_1 \leftarrow [\beta]^m. \tag{42}$$

To encrypt a message $\mu \in \mathcal{R}_f$ whose coefficients are in $\{0,1\}$, the encryptor samples $\mathbf{r}, \mathbf{e}_2 \leftarrow [\beta]^m$ and $e_3 \leftarrow [\beta]$, and outputs

$$\left(\mathbf{u}^T = \mathbf{r}^T\mathbf{A} + \mathbf{e}_2^T, v = \mathbf{r}^T\mathbf{t} + e_3 + \frac{q}{2}\mu\right). \tag{43}$$

The security argument based on $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{m,\beta}$ is identical to the proof based on $\mathsf{LWE}_{m,q,\beta}$ in Section 2.3.1.
To decrypt, one computes

$$v - \mathbf{u}^T\mathbf{s} = \mathbf{r}^T(\mathbf{As}+\mathbf{e}_1) + e_3 + \frac{q}{2}\mu - \left(\mathbf{r}^T\mathbf{A} + \mathbf{e}_2^T\right)\mathbf{s} \tag{44}$$

$$= \mathbf{r}^T\mathbf{e}_1 + e_3 + \frac{q}{2}\mu - \mathbf{e}_2^T\mathbf{s} \tag{45}$$

To compute the decryption error, we can rewrite the above equation (excluding $\frac{q}{2}\mathcal{V}_\mu$) as

$$\mathcal{M}_{\mathbf{r}^T}\mathcal{V}_{\mathbf{e}_1} + \mathcal{V}_{e_3} - \mathcal{M}_{\mathbf{e}_2^T}\mathcal{V}_{\mathbf{s}}$$

and then apply the techniques in Section 2.3.2 to compute the probability that none of the d coefficients has magnitude greater than $q/4$. When $f = X^d \pm 1$, then for each of the d coefficients, computing this probability is the same as when we worked over the integers because the coefficients in every row of $\mathcal{M}_{\mathbf{r}^T}$ and $\mathcal{M}_{\mathbf{e}_2^T}$ are independent (see (38) and (39)). One then applies the union bound to bound the probability that all d decryption errors are small. For rings over other polynomials, one can still apply the techniques in Section 2.3.2 if one can rewrite the matrix-vector multiplication as a sum of independent random variables.

4.3.1 Optimizations and Efficiency

The main advantage of the Generalized-LWE scheme is that one does not need to increase the size of the public key, as in Section 2.4, in order to be able to encrypt a larger message. When f has degree d, the ring naturally supports the encryption of d bits. So as long as we set $d \geq 256$, which is the length of an AES (or any symmetric cipher) key that one would encrypt using public key encryption, we are able to use an optimally-small public key. Similarly, there is no need for packing more than one message bit per coefficient as in Section 2.5.3. The optimization in Section 2.5.1 is stil very useful and is applied in exactly the same manner as before. The Learning with Rounding problem and cryptosystem (Section 2.5.2), as well as the non-interactive key exchange (Section 2.6), are also defined analogously for the ring $\mathcal{R}_f$.

4.3.2 Security and Connection to Integer Lattices

The connection between polynomial operations in $\mathcal{R}_f$ and linear algebra over $\mathbb{Z}$ in (37) allows us to make useful connections between the lattices we saw in Section 3.1 and (short) solutions to polynomial equations involved in the $\mathcal{R}_{q,f}\text{-}\mathsf{LWE}_{n,m,\beta}$ definition in the previous section. The hardness of the LWE-based encryption scheme relied on the hardness of finding, for a random $\mathbf{A} \in \mathbb{Z}_q^{m\times m}$ and a $\mathbf{t} \in \mathbb{Z}_q^m$, integer vectors $\mathbf{s}, \mathbf{e} \in \mathbb{Z}_q^m$ with small coefficients satisfying $\mathbf{As} + \mathbf{e} = \mathbf{t}$. We saw that this was equivalent to finding a particular vector with small coefficients in the lattice $\mathcal{L}_q^\perp([\mathbf{A} \mid \mathbf{t} \mid \mathbf{I}_m])$. And we based the concrete security of the LWE scheme on the hardness of this latter problem.
The security of the more efficient encryption scheme in this section is analogously based on the hardness of finding, for a random $\mathbf{A} \in \mathcal{R}_{q,f}^{m\times m}$ and a $\mathbf{t} \in \mathcal{R}_{q,f}^m$, polynomial vectors $\mathbf{s}, \mathbf{e} \in \mathcal{R}_f^m$ with small coefficients satisfying $\mathbf{As} + \mathbf{e} = \mathbf{t}$. By (37), this is equivalent to finding integer vectors $\mathcal{V}_\mathbf{s}, \mathcal{V}_\mathbf{e} \in \mathbb{Z}^{dm}$ with small coefficients satisfying $\mathcal{M}_\mathbf{A}\mathcal{V}_\mathbf{s} + \mathcal{V}_\mathbf{e} = \mathcal{V}_\mathbf{t}$. Since this equation is over $\mathbb{Z}$, we can transform it into a problem about finding short vectors in integer lattices – that is, finding a particular vector with short coefficients in the lattice $\mathcal{L}_q^\perp([\mathcal{M}_\mathbf{A} \mid \mathcal{V}_\mathbf{t} \mid \mathbf{I}_{dm}])$.
Because the number of columns in $\mathcal{M}_\mathbf{A}$ is dm, the dimension of the lattice

$$\mathcal{L}_q^\perp([\mathcal{M}_\mathbf{A} \mid \mathcal{V}_\mathbf{t} \mid \mathbf{I}_{dm}])$$

is $d(m+1)+1$. If the algebraic structure of $\mathcal{R}_f$ does not exhibit any weaknesses (see Section 4.4.1 below), then the hardness of finding a short vector in this lattice is the same as in the $\mathsf{LWE}_{n',m',q,\beta}$ lattice in (31) with $n' = dn$ and $m' = dm$. For the $\mathcal{R}_{q,f}\text{-}\mathsf{LWE}_{n,m,\beta}$ problem, therefore, the important value is dm – the product of the degree of f in $\mathcal{R}_f$ and the number of columns of $\mathbf{A} \in \mathcal{R}_f^{n\times m}$. Similarly, for the $\mathcal{R}_{q,f}\text{-}\mathsf{SIS}_{n,m,\beta}$ problem, the important value is dn – the product of the degree of f and the number of rows in $\mathbf{A}$.

4.4 Exploiting the Algebraic Structure ...

4.4.1 ... for Attacks

Suppose that we are working over the ring $\mathcal{R}_{q,f}$ with $f = X^d - 1$. Because $X - 1$ is a factor of $X^d - 1$, there is a ring homomorphism from $\mathcal{R}_{q,f}$ to $\mathcal{R}_{q,X-1}$ which maps elements $a = \sum\limits_{i=0}^{d-1} a_iX^i \in \mathcal{R}_{q,f}$ to $a' = \sum\limits_{i=0}^{d-1} a_i \in \mathcal{R}_{q,X-1}$. Because the ring $\mathcal{R}_{q,X-1}$ is exactly the ring $\mathbb{Z}_q$ with the usual addition and multiplication modulo q, we actually have a ring homomorphism from $\mathcal{R}_{q,f}$ to $\mathbb{Z}_q$. What's particularly special about this homomorphism is that if the coefficients of a are small, then the image of a under the homomorphism is small as well (i.e. can only be a factor of d larger). This implies the following simple attack on the $\mathcal{R}_{q,f}\text{-}\mathsf{LWE}_{m,\beta}$ problem where we're given $\mathbf{A} \in \mathcal{R}_{q,f}^{n\times m}, \mathbf{t} \in \mathcal{R}_{q,f}^n$ and are asked to decide whether there exist $\mathbf{s}, \mathbf{e}$ with coefficients in $[\beta]$ satisfying $\mathbf{As} + \mathbf{e} = \mathbf{t}$ (the analogous attack on the $\mathcal{R}_{q,f}\text{-}\mathsf{SIS}_{n,m,\beta}$ problem was given in [PR06, LM06]):

Let $\mathbf{A}' \in \mathbb{Z}_q^{n\times m}, \mathbf{t}' \in \mathbb{Z}_q^n$ be images of $\mathbf{A}, \mathbf{t}$ under the homomorphism. If there really existed $\mathbf{s} \in [\beta]^m \subset \mathcal{R}_f^m$ and $\mathbf{e} \in [\beta]^n \subset R^n$ satisfying $\mathbf{As} + \mathbf{e} = \mathbf{t}$, then there exist $\mathbf{s}' \in [d\beta]^m \subset \mathbb{Z}^m, \mathbf{e}' \in [d\beta]^n \subset \mathbb{Z}^n$ satisfying $\mathbf{A}'\mathbf{e}' + \mathbf{s}' = \mathbf{t}'$. Because m and n are fairly small, one can efficiently find such a vector by looking for the shortest vector in the lattice set up as in (31). If we find such $\mathbf{s}', \mathbf{e}'$, then we claim that $\mathbf{s}, \mathbf{e}$ with coefficients in $[\beta]$ exist. If we don't find such $\mathbf{s}', \mathbf{e}'$, then we say that $\mathbf{A}, \mathbf{t}$ were random. If $\mathbf{A}, \mathbf{t}$ were uniformly random, then so are $\mathbf{A}', \mathbf{t}'$, and one can use Lemma 1 to upper-bound the probability that $\mathbf{s}', \mathbf{e}'$ with small coefficients can exist. If this upper bound is $1 - \epsilon$, then the above distinguisher has advantage at least ϵ.
The most crucial element enabling the above attack on $\mathcal{R}_{q,f}\text{-}\mathsf{LWE}_{m,\beta}$ was that there existed a homomorphism into a ring of a smaller degree which *did not increase the coefficient size* by much. If f had a factor of, for example, $X - 2$, then the

above attack would not work because the homomorphism would map an element $a = \sum_{i=0}^{d-1} a_i X^i \in \mathcal{R}_{q,f}$ to $a' = \sum_{i=0}^{d-1} a_i 2^i \in \mathbb{Z}_q$, and so a' would be in a range that is exponentially dependent on d. Interestingly, the worst-case to average-case reductions showing that solving $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{m,\beta}$ and $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m,\beta}$ implies finding short vectors in ideal / module lattices [PR06, LM06, LPR13a, LS15, PRS17] only require that f be irreducible (or at least have a large irreducible part in the case of $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m,\beta}$) over the ring $\mathbb{Z}[X]$ and not $\mathbb{Z}_q[X]$. As we will see in the next section, it is actually quite advantageous, in terms of implementation efficiency, to work over a ring $\mathcal{R}_{q,f}$ where the polynomial f has many low-degree factors in $\mathbb{Z}_q[X]$.

4.4.2 ... for Efficiency

The most computationally-involved algebraic operation in the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ encryption scheme is multiplication of polynomials in $\mathcal{R}_{q,f}$. The basic "school-book" polynomial multiplication of two polynomials of degree d requires $O(d^2)$ operations. There are better methods like Karatsuba and Toom-Cook which take approximately $O(d^{1.5})$ time. The most efficient way to perform polynomial multiplication in $\mathcal{R}_{q,f}$, which requires only $O(d \log d)$ operations over $\mathbb{Z}_q$, is via the NTT (Number Theoretic Transform), which is a special case of the FFT performed over the field $\mathbb{Z}_q^*$ rather than over the complex numbers. Recall that the FFT over the complex numbers crucially uses the fact that there always exists an element $\zeta \in \mathcal{C}$ whose order is d, where d is the degree of the product of two polynomials. The existence of such elements in $\mathbb{Z}_q^*$ is not always a given, and so f and q will need to be chosen appropriately. The NTT multiplication procedure is particularly amenable to parallelization, and is therefore extremely fast on architecture that supports AVX2 instructions [Sei18]. Another practical advantage is that this operation can be done "in place", meaning that it does not require any extra temporary storage space while performing the computation, which can be useful for constrained devices. In this section, we will describe the NTT.
If the polynomial f factors as

$$f(X) = (X - r_1) \cdots (X - r_d) \tag{46}$$

for distinct $r_i \in \mathbb{Z}_q$, then one can perform multiplication via the Chinese Remainder Theorem (CRT). For an element $a \in \mathcal{R}_{q,f}$, we define its CRT representation $\hat{a}$ as

$$\hat{a} = (a \bmod X - r_1, \ldots, a \bmod X - r_d) \in \mathbb{Z}_q^d, \tag{47}$$

and then multiplication $ab \in \mathcal{R}_{q,f}$ can be performed as follows:

1. Compute $\hat{a} = (a_1, \ldots, a_d)$ and $\hat{b} = (b_1, \ldots, b_d)$ as in (47).
2. Multiply individual components to produce $\widehat{ab} = \hat{a} \odot \hat{b} = (a_1 b_1, \ldots, a_d b_d)$.
3. Compute c such that $\hat{c} = \widehat{ab}$

Step 2 of the above procedure consists of just d multiplications over $\mathbb{Z}_q$, and so it is steps 1 and 3 that will form the bulk of the running time. The key to being able to perform these steps extremely efficiently is to make sure that the polynomial f splits in a very special way over $\mathbb{Z}_q[X]$. The most efficient polynomials f are of the form $X^d \pm 1$ when d is a power-of-2, and we will be using $X^d + 1$ in the below example, which computes steps 1 and 3 using the Number Theory Transform (NTT), which is just an analogue of the Fast Fourier Transform (FFT), but done over the field $\mathbb{Z}_q^*$ rather than $\mathbb{C}$.
For j that are powers of 2, let us define the sets $\rho^{(j)}$ to consist of elements $r \in \mathbb{Z}_q^*$ such that $r^j = -1$. In particular, these elements r are j^{th} roots of -1. By the fact that the equation $X^j + 1$ can have at most j solutions over the field $\mathbb{Z}_q^*$, we know that the size of $\rho^{(j)}$ is at most j. If we set q such that $q \equiv 1 (\bmod\, 2j)$, then the size of $\rho^{(j)}$ will be exactly j. Thus if $q \equiv 1 (\bmod\, 2d)$, we will have the factorization of $X^d + 1$ as in (46).

An important observation is that if $r \in \rho^{(j)}$, then $-r$ is as well (since $j \geq 2$ is even). Furthermore, it's clear that if r is in $\rho^{(j)}$, then $r^2 \in \rho^{(j/2)}$. By repeating this argument, $-r^2$ is also in $\rho^{(j/2)}$. So by induction, all sets $\rho^{(j)}$ for $j \geq 2$ contain $\pm r$, and r^2 is in $\rho^{(j/2)}$.
We use the above observation for computing $\hat{a}$ in the following way: the linear factors $X - r_1, \ldots, X - r_d$ for $r_i \in \rho^{(d)}$ can be rewritten as $X - r_1, X + r_1, X - r_2, X + r_2, \ldots, X - r_{d/2}, X + r_{d/2}$, and so we can write

$$\begin{aligned} X^d + 1 &= (X - r_1)(X + r_1)(X - r_2)X + r_2) \cdots (X - r_{d/2}), (X + r_{d/2}) \\ &= (X^2 - r_1^2)(X^2 - r_2^2) \cdots (X^2 - r_{d/2}^2), \end{aligned}$$

where $r_i^2 \in \rho^{(d/2)}$. By the same observation, we can go further and write

$$\begin{aligned} X^d + 1 &= (X - r_1)(X + r_1)(X - r_2)X + r_2) \cdots (X - r_{d/2}), (X + r_{d/2}) \\ &= (X^2 - r_1^2)(X^2 + r_1^2) \cdots (X^2 - r_{d/4}^2)(X^2 + r_{d/4}^2) \\ &= (X^4 - r_1^4) \cdots (X^4 - r_{d/4}^4) \\ &= (X^8 - r_1^8) \cdots (X^8 - r_{d/8}^8) \\ &\cdots \\ &= (X^{d/2} - r_1^{d/2})(X^{d/2} - r_2^{d/2}), \end{aligned}$$

where $r_2^{d/2} = -r_1^{d/2}$. Note that once we have computed

$$(a \bmod X^{d/j} - r_1^{d/j}, \ldots, a \bmod X^{d/j} - r_j^{d/j}) \tag{48}$$

for all the j elements $r_i^{d/j} \in \rho^{(j)}$, we can compute

$$(a \bmod X^{d/2j} - r_1^{d/2j}, \ldots, a \bmod X^{d/2j} - r_{2j}^{d/2j}) \tag{49}$$

for all the $2j$ elements in $\rho^{(2j)}$ by taking the value $a \bmod (X^{d/j} - r_i^{d/j})$ for each i and computing

$$a \bmod (X^{d/2j} \pm r_i^{d/2j}) = a \bmod (X^{d/j} - r_i^{d/j}) \bmod (X^{d/2j} \pm r_i^{d/2j}). \tag{50}$$

We will now see that the computation in (50) takes order of d/j steps (each step is either a multiplication or addition over $\mathbb{Z}_q$), and so doing it for each of the j elements in (48), requires on the order of d steps. Since there are $\log d$ levels, the total time to compute $\hat{a}$ is on the order of $d \log d$. We will also show that it takes essentially the same time to compute a when given $\hat{a}$ because computing (48) from (49) also takes on the order of d/j operations.
The basic operation in (50) involves taking some polynomial $a \in \mathbb{Z}_q[X]$ of degree d/j and computing

$$a_- = a \bmod X^{d/2j} - r_i^{d/2j} \text{ and } a_+ = a \bmod X^{d/2j} + r_i^{d/2j}. \tag{51}$$

These reductions are actually quite simple to perform and require just $d/2j$ multiplications by $r_i^{d/2j}$ and d/j additions. For example, if $a = a_3X^3 + a_2X^2 + a_1X + a_0$, then

$$a_- = (a_1 + a_3 r_i^{d/2j})X + (a_0 + a_2) \tag{52}$$
$$a_+ = (a_1 - a_3 r_i^{d/2j})X + (a_0 - a_2). \tag{53}$$

Note that the same multiplications by $r_i^{d/2j}$ are performed for computing a_- and a_+ and therefore can be re-used.
The main step in the computation of the inverse NTT (i.e. going from (49) to (48)) involves taking two polynomials a_- and a_+ of degree $d/2j - 1$ and computing the unique polynomial a of degree at most $d/j - 1$ satisfying (51). To see how such an a can be efficiently computed, note from (52) and (53) that $2^{-1} \cdot (a_- + a_+)$ gives us the

coefficients a_1 and a_0 of a, while $(2r_i^{d/2j})^{-1}(a_- - a_+)$ gives us the remaining coefficients a_3 and a_2. This completes the description of how multiplication is performed in the ring $\mathcal{R}_{q,X^d+1}$.
It should be pointed out that it's not necessary for q to be chosen such that (46) holds in order to have efficient multiplication. For example, even if we only have $f(X) = (X^2 - r_1)\cdots(X^2 - r_{d/2})$, and $X^2 - r_i$ are irreducible, we can still perform efficient multiplication. Instead of defining the CRT representation as in (47), we could instead define it as

$$\hat{a} = (a \bmod X^2 - r_1, \ldots, a \bmod X^2 - r_{d/2}). \tag{54}$$

Then the only difference in the NTT multiplication algorithm is that step 2 performs multiplication over the field $\mathcal{R}_{q,f}$ with $f = X^2 - r_i$. This is only slightly less efficient than multiplication over $\mathbb{Z}_q^*$, but the slight advantage is that the number of levels for steps 1 and 3 is now $\log d - 1$ instead of $\log d$. One also doesn't need the polynomial f to be $X^d + 1$. There are other (cyclotomic) polynomials that have a factorization tree very similar to $X^d + 1$. For example, for certain d and q, the polynomial $f = X^d - X^{d/2} + 1$ factors into $(X^{d/2} - r_1)(X^{d/2} - r_2)$ and then $X^{d/2} - r_i)$ factors as in the $X^d + 1$ case (i.e. $X^{d/2} - r_1 = (X^{d/4} - r')(X^{d/4} + r')$) until the irreducible field $\mathcal{R}_{q,f}$ where f has degree 3. Multiplication can be performed over such rings (c.f. [LS19]) almost as efficiently as when $f = X^d + 1$. One can perform NTT multiplication over rings $\mathcal{R}_{q,f}$ for arbitrary f by multiplying over $\mathbb{Z}_q[X]$ and then reducing modulo f. Multiplication over $\mathbb{Z}_q[X]$ can be performed by doing multiplication over $\mathcal{R}_{q,X^d+1}$ where the dimension d is chosen to be high enough so that reduction modulo $X^d + 1$ never occurs.

4.5 NTRU

The NTRU cryptosystem [HPS98] was the first truly efficient lattice-based encryption scheme, and was also the first to propose using polynomial rings – specifically $\mathcal{R}_{q,f}$ (with $f = X^d - 1$) – in lattice-based cryptography. The scheme was originally proposed as a trapdoor one-way function, which can be seen as a OW-CPA cryptosystem.[3] There is also a simple modification that allows us to construct a CPA-secure encryption scheme [SS11]. In most use cases, however, the trapdoor one-way function is sufficient since there is a black-box transformation from such primitives to CCA-secure encryption schemes (c.f. [Den02]).

4.5.1 The NTRU Trapdoor 1-way Function

In the previous sections, we worked with the decision version of the Generalized-LWE problem, but it is also very natural to define a *search* version of it. This problem could be stated as finding the e when being given $(a, as + e)$ for $a \leftarrow \mathcal{R}_{q,f}$ and $s, e \leftarrow [\beta]$. Notice that if a is invertible in $\mathcal{R}_{q,f}$, then finding e immediately also implies finding s.
The NTRU problem is very similar to the above, except that the polynomial a is not chosen at random from $\mathcal{R}_{q,f}$, but is rather the product of the integer $(2\beta + 1)$ and polynomials g_1 and g_2^{-1} where $g_i \leftarrow [\beta]$ (conditioned on g_2 being invertible) and $2\beta + 1$ being relatively prime to q (e.g. $\beta = 1$ is a popular choice). The hardness behind NTRU relies on the assumption that the search $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ problem (with $n = m = 1$) is still hard whenever a is not uniformly random, but is instead the product $(2\beta + 1)g_1 g_2^{-1}$.
The NTRU problem is formally defined as follows:

Definition 3. Let $p = 2\beta + 1$. Given $(a, as + e)$, where $a = pg_1g_2^{-1}$ for $g_1, g_2, s, e \leftarrow [\beta]$ and g_2 being invertible in $\mathcal{R}_{q,f}$ and $\mathcal{R}_{p,f}$, find e.

[3] An encryption scheme is OW-CPA secure if an attacker, in possession of the public key, cannot recover the message from a ciphertext of a randomly-chosen message.

Based on the presumed hardness of the above problem, we can construct a trapdoor one-way function family as follows: to generate a random element from the family, we choose secret invertible polynomials $g_1, g_2 \leftarrow [\beta]$ with g_2 being invertible in $\mathcal{R}_{q,f}$ and $\mathcal{R}_{p,f}$, and output the public key to be

$$a = pg_1 g_2^{-1}. \tag{55}$$

The secret key is g_2.
The one-way function mapping $s, e \leftarrow [\beta]$ to $\mathcal{R}_{q,f}$ computes

$$b = as + e \in \mathcal{R}_{q,f}. \tag{56}$$

Observe that in order to recover e, s, it is enough to recover these modulo p since there is a 1-1 correspondence between elements in $[\beta]$ and residues modulo $p = 2\beta + 1$. To recover s, e modulo p using the secret key, one first computes

$$g_2 b \bmod p = pg_1 s + g_2 e \bmod p = g_2 e \bmod p. \tag{57}$$

The first equality (in the ring $\mathcal{R}_{q,f}$) simply follows from the definition of a and b. The second equality only holds true if the preceding equation holds true not only in $\mathcal{R}_{q,f}$, but also over $\mathcal{R}_f$. Here is where we again use the fact that the coefficients of g_i, s, and e are small with respect to q, and can bound them to be less than q (either always or with very high probability). If this is the case, then we have

$$(g_2 b \bmod p) g_2^{-1} \bmod p = e, \tag{58}$$

where we multiplied by the inverse of g_2 in $\mathcal{R}_{p,f}$. Once we have $e \pmod{p}$, which is the same as $e \in [\beta]$, we can also compute

$$(b - e)a^{-1} = s. \tag{59}$$

4.5.2 Security

If we don't presume any special structure of the polynomial a, then recovering s, e in (56) is identical to the attack on the public key of the Generalized-LWE instance in which we try to find a short vector in the lattice

$$\mathcal{L}_q^{\perp}([\mathcal{M}_a \mid \mathcal{V}_b \mid \mathbf{I}_d]). \tag{60}$$

We can also try to recover the secret key g_1, g_2 (or some other short polynomials related to it) from $a = pg_1 g_2^{-1}$ by looking for a short vector in the lattice

$$\mathcal{L}_q^{\perp}([\mathcal{M}_{p^{-1}a} \mid \mathbf{I}_d]). \tag{61}$$

The lattices in (60) and (61) look very similar, with the only relevant difference being the presence of one extra vector $\mathcal{V}_b$ in (60). It was therefore somewhat surprising that in certain scenarios, where q is significantly larger than β (but not so large as to be in the space where generic lattice reduction algorithms clearly work), it was significantly easier to find short vectors in the lattice in (61) than in (60) [ABD16, CJL16, KF17]. While these attacks don't translate to attacks against NTRU paramters, they do prevent the NTRU assumption from being used in advanced primitives (e.g. FHE) that require large moduli and small noise. For this reason, schemes based on Generalized-LWE (whose security relies on essentially (60)) are used in such scenarios.

4.6 Generalized-LWE and NTRU Efficiency

4.6.1 Efficiency of the Generalized-LWE Encryption

Looking at the efficiency of key generation of the Generalized-LWE scheme (42), the most expensive operations are sampling $\mathbf{A}$ and the multiplication $\mathbf{As}$. Since $\mathbf{A}$ consists of m^2 polynomials in $\mathcal{R}_{q,f}$, representing $\mathbf{A}$ requires dm^2 elements in $\mathbb{Z}_q$. The multiplication $\mathbf{As}$ involves m^2 polynomial multiplications and m additions. If the polynomial multiplication is done via NTT as in Section 4.4.2, then the cost is actually noticeably cheaper than doing m^2 NTT multiplications that involve all three steps in Section 4.4.2. Firstly, the matrix $\mathbf{A}$ can be generated so that it's already in CRT form – so instead of generating uniformly random coefficients of the polynomial a from $\mathbb{Z}_q$, one would generate the coefficients $a_1, \ldots, a_d$ of $\hat{a}$ from $\mathbb{Z}_q$, which results in the exact same distribution.[4] Having $\mathbf{A}$ in this form immediately removes the need to perform the first step in the NTT multiplication algorithm. Secondly, we only need to perform m CRT conversions for $\mathbf{s}$ and don't need to perform any inverse CRT conversions for the product $\mathbf{As}$ since it's better to leave $\mathbf{t} = \mathbf{As}$ already in CRT form. So while we perform step 2 m^2 times (thus requiring $m^2 d$ multiplications in $\mathbb{Z}_q$), step 1 is only performed m times, while step 3 is never needed.

Let us now look at the encryption step in (43). If the encryptor does not store the matrix $\mathbf{A}$, he will need to generate $\mathbf{A}$ from some seed, just like in the key generation. After this, the multiplication $\mathbf{r}^T\mathbf{A}$ takes exactly the same time as computing $\mathbf{As}$ in the key generation step. There is also the additional computation $\mathbf{r}^T\mathbf{t}$. If we are doing multiplication using NTT, then $\mathbf{r}$ and $\mathbf{t}$ are already in CRT form ($\mathbf{r}$ from the previous multiplication by $\mathbf{A}$ and $\mathbf{t}$ because it is stored in CRT form during key generation. It is important to point out that if we are going to reduce the size of the ciphertext by removing the low-order bits as in Section 2.5.1, then one will need to first apply the inverse NTT to the result of $\mathbf{r}^T\mathbf{A}$ and $\mathbf{r}^T\mathbf{t}$ in order to perform the rounding operations on the polynomial coefficients. Furthermore, to preserve the created compactness, the ciphertext cannot be stored in CRT form.

The most expensive part of the decryption operation in (45) involves the multiplication $\mathbf{u}^T\mathbf{s}$, and so is the least expensive of the three operations. It should be pointed out, however, that in the CCA version of the encryption scheme, the decryption operation needs to re-encrypt, and thus includes the steps from the above encryption operation. In practice, if multiplication is done via NTT, then the most expensive part turns out to be the operation of generating $\mathbf{A}$ from a seed.

4.6.2 Efficiency of NTRU

The running time of creating an NTRU function (i.e. the public key a) involves sampling g_1, g_2, performing an inversion g_2^{-1}, and then multiplying by g_1. The complexity of polynomial multiplication is the same as in the Generalized-LWE encryption described above when $m = 1$. For a fair apples-to-apples comparison, we will compare NTRU to the Generalized-LWE scheme where $m = 1$. If the ring supports NTT, then the inversion step g_2^{-1} is also very simple. One just computes the CRT representation of g_2 and inverts all the CRT coefficients in $\mathbb{Z}_q^*$, which is the CRT representation of g_2^{-1}. If the ring does not support NTT, then inversion (or polynomial division) can be significantly more expensive (around an order of magnitude for practical parameters, c.f. [HRSS17]) than multiplication. The main observation here is that the NTRU public key is just a, whereas the Generalized-LWE public key would consist of a, t. As previously discussed, one does not need to store the polynomial a, but just a small seed from which to expand it, and so the the vast majority of the public key description is in t. And so the NTRU and Generalized-LWE public keys have essentially the same size. The main advantage of NTRU will be the fact that one will not need to expand a from a seed when doing encryption (and decryption in the CCA construction), which is a significant computational saving.

The function evaluation step in (56) exactly corresponds to the computation $\mathbf{r}^T\mathbf{A} + \mathbf{e}$ in the Generalized-LWE encryption procedure. The function inversion / decryption step in (57) first involves multiplying b by g_2^{-1} in $\mathcal{R}_{q,f}$, and then the result by g_2^{-1} in $\mathcal{R}_{p,f}$. The polynomial g_2^{-1} was already needed in the key generation step, so the first multiplication is just a multiplication in $\mathcal{R}_{q,f}$. The second part involves inverting g_2 in $\mathcal{R}_{p,f}$. We would either have to do this now, or precompute it during the key generation step. The interesting part is that we cannot use NTT for this inversion

[4]Note that the same method cannot be applied to generating short polynomials $s \in [\beta]$ because there is no *independent* distribution over its CRT coefficients that leads to such a distribution.

because the prime p (generally 3) is too small to support NTT – i.e. it is not possible to have d distinct $r_i \in \mathbb{Z}_p^*$ such that $f(X) = (X - r_1)\cdots(X - r_d)$. A way to avoid needing to do this inversion is by creating a g_2 that is congruent to $1 \pmod{p}$. In other words, $g_2 = 1 + p \cdot g_2'$ where $g_2' \leftarrow [\beta]$. The downside of this approach is that g_2 is now larger, which means that the coefficients of $pg_1s + g_2e$ in (57) are larger, and so q has to be larger so that this value does not get reduced modulo q. Still, the fastest lattice-based encryption scheme (or, for that matter, any currently known encryption scheme) would be an instantiation of NTRU over rings that support NTT and with $g_2 \equiv 1 \pmod{p}$ [LS19].

4.7 A Concrete Example of an Encryption Scheme

In this section, we will put everything together by giving an example of an $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{m,\beta}$ encryption scheme with $f = X^{256} + 1$, $q = 3329$, $n = m = 3$, and $\beta = 1$. This set of parameters is similar to those in the CPA version of the CRYSTALS-Kyber encryption scheme in the NIST standardization process [BDK+18, ABD+19] that has over 128 bits of classical (and quantum) security.[5] The polynomial f factors as

$$X^{256} + 1 = (X^2 - r_1)\cdots(X^2 - r_{128}) \bmod 3329 \tag{62}$$

and thus each element $a \in \mathcal{R}_{q,f}$ has a CRT representation $\hat{a}$ as in (54).
The secret key consists of

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \text{ and } \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} \leftarrow \mathcal{R}_{q,f}^3$$

where $s_i, e_i \leftarrow [\beta]$.
The first part of the public key is a seed $\rho \in \{0,1\}^{256}$ whose expansion via some PRNG (such as SHAKE) represents the matrix

$$\hat{\mathbf{A}} = \begin{bmatrix} \hat{a}_{11} & \hat{a}_{12} & \hat{a}_{13} \\ \hat{a}_{21} & \hat{a}_{22} & \hat{a}_{23} \\ \hat{a}_{31} & \hat{a}_{32} & \hat{a}_{33} \end{bmatrix},$$

where each $\hat{a}_{ij}$ is the CRT representation, as in (54), of some polynomial $a_{ij} \in \mathcal{R}_{q,f}$. As we mentioned in Section 4.4.2, if we do our multiplications using NTT, it makes sense to have the uniformly random $\mathbf{A}$ to already be in CRT form to avoid doing conversions.
The second part of the public key is

$$\hat{\mathbf{t}} = \begin{bmatrix} \hat{t}_1 \\ \hat{t}_2 \\ \hat{t}_3 \end{bmatrix} = \widehat{\mathbf{As} + \mathbf{e}},$$

which is the CRT representation of $\mathbf{t} = \mathbf{As} + \mathbf{e} \in \mathcal{R}_{q,f}^3$. Since we already have $\mathbf{A}$ in CRT representation, the way to compute $\hat{\mathbf{t}}$ is by first putting $\mathbf{s}, \mathbf{e}$ into CRT form and then computing

$$\hat{\mathbf{A}}\hat{\mathbf{s}} + \hat{\mathbf{e}} = \begin{bmatrix} \hat{a}_{11} \odot \hat{s}_1 + \hat{a}_{12} \odot \hat{s}_2 + \hat{a}_{13} \odot \hat{s}_3 + \hat{e}_1 \\ \hat{a}_{21} \odot \hat{s}_1 + \hat{a}_{22} \odot \hat{s}_2 + \hat{a}_{23} \odot \hat{s}_3 + \hat{e}_2 \\ \hat{a}_{31} \odot \hat{s}_1 + \hat{a}_{32} \odot \hat{s}_2 + \hat{a}_{33} \odot \hat{s}_3 + \hat{e}_3 \end{bmatrix},$$

where we recall that $\odot$ means that two CRT representations are multiplied component-wise. The total size of the public key $\rho, \hat{\mathbf{t}}$ is thus $256 + 3 \cdot 256 \cdot 12$ bits.
To encrypt a message μ as in (43), the encryptor generates $\mathbf{r}, \mathbf{e}_2$, and e_3 as in (43), converts $\mathbf{r}$ into CRT form and computes $\widehat{\mathbf{r}^T\mathbf{A}} = \hat{\mathbf{r}}^T\hat{\mathbf{A}}$ and $\mathbf{r}^T\mathbf{t} = \hat{\mathbf{r}}^T\hat{\mathbf{t}}$ as above

[5] The main difference is that in Kyber, the secret / error coefficients are not chosen uniformly from [1], but from a binomial distribution which is generated as $(a_1 + a_2 - a_3 - a_4)$ where each $a_i \leftarrow \{0,1\}$ and has a weight on $0, \pm1, \pm2$ of $6/16, 4/16$, and $1/16$.

and then applies the inverse NTT to obtain $\mathbf{r}^T\mathbf{A}$ and $\mathbf{r}^T\mathbf{t}$ in the usual coefficient representation. He then adds the errors $\mathbf{e}_2$ and e_3 and the message μ to create $\mathbf{u} \in \mathcal{R}_{q,f}^3$ and $v \in \mathcal{R}_{q,f}$ as in (43). Unlike with the public key, we are not leaving the ciphertext in CRT form because we will be compressing it via rounding as in Section 2.5.1.

There are two reasons why the techniques in Section 2.5.1 require the coefficient, rather than the CRT, representation. The first is that compression works by "chopping off" the low-order bits in the coefficient representation, so we cannot be in CRT domain for that. And secondly, the compressed ciphertexts require fewer bits to represent in the coefficient representation, but not in the CRT one.

We will now compress $\mathbf{u}$ and v using the techniques in Section 2.5.1. We will only keep 3 bits of each coefficient of v and thus define the set $\mathcal{S}_v$ to consist of 2^3 points equidistantly distributed in $\mathbb{Z}_q$. We transmit only $\mathtt{HIGH}_{\mathcal{S}_v}(v)$ (which requires 3 bits per coefficient – so $3 \cdot 256$ in total). If we keep 10 bits of each coefficient of each polynomial of $\mathbf{u}$, then we define $\mathcal{S}_u$ to consist of 2^{10} points (almost) equidistantly distributed in $\mathbb{Z}_q$. As in above, we transmit only $\mathtt{HIGH}_{\mathcal{S}_u}(\mathbf{u})$ (which requires 10 bits per coefficient – so $3 \cdot 10 \cdot 256$). The ciphertext size is thus $(3 + 3 \cdot 10) \cdot 256$ bits.

Decryption proceeds as in (45), except the decryptor does not have the full ciphertext $(\mathbf{u}, v)$, but rather just $(\mathbf{u}', v') = (\mathtt{HIGH}_{\mathcal{S}_u}(\mathbf{u}), \mathtt{HIGH}_{\mathcal{S}_v}(v))$. So he computes $v' - \mathbf{u}'\mathbf{s}$ by putting $\mathbf{u}', \mathbf{s}$ into CRT representation[6] and compute $\mathbf{u}'^T\mathbf{s}$ in coefficient representation by computing $\hat{\mathbf{u}}'^T\hat{\mathbf{s}}$ and then performing an inverse NTT, and then subtracting it from v'.

4.7.1 Decryption Error Calculation

Using the facts that $\mathbf{u} = \mathtt{HIGH}_{\mathcal{S}_u}(\mathbf{u}) + \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})$ where $\mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u}) \in [q/2^{11}]^3$ and $v = \mathtt{HIGH}_{\mathcal{S}_v}(v) + \mathtt{LOW}_{\mathcal{S}_v}(v)$ where $\mathtt{LOW}_{\mathcal{S}_v}(v) \in [q/2^4]$, we can rewrite the decryption equation $v' - \mathbf{u}'^T\mathbf{s}$ as

$$\begin{aligned} v' - \mathbf{u}'^T\mathbf{s} &= (v - \mathtt{LOW}_{\mathcal{S}_v}(v)) - (\mathbf{u} - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u}))^T\mathbf{s} \\ &= \mathbf{r}^T\mathbf{e}_1 + e_3 + \frac{q}{2}\mu - \mathtt{LOW}_{\mathcal{S}_v}(v) - (\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u}))^T\,\mathbf{s} \end{aligned}$$

The probability of getting a decryption error is the probability that any coefficient in the polynomial

$$\mathbf{r}^T\mathbf{e}_1 + e_3 - \mathtt{LOW}_{\mathcal{S}_v}(v) - (\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u}))^T\,\mathbf{s} \tag{63}$$

is greater in magnitude than $q/4$. Something to note is that the coefficients of the above polynomial will not be independent, so we will be computing the probability that some particular polynomial has magnitude greater than $q/4$ and then applying the union bound.

To use the techniques in Section 2.3.2 for computing the probability distribution of the coefficients in (63), we need to write each coefficient as a sum of independently distributed random variables. For example, the coefficients of the polynomial $\mathbf{r}^T\mathbf{e}_1$ are the coefficients of the matrix-vector product $\mathcal{M}_{\mathbf{r}^T} \cdot \mathcal{V}_{\mathbf{e}_1}$ over $\mathbb{Z}_q$. Each row of $\mathcal{M}_{\mathbf{r}^T}$ consists of $256 \cdot 3$ independently-distributed integers from [1], and $\mathcal{V}_{\mathbf{e}_1}$ similarly consists of the same number of independent elements with the same distribution. Thus the product of the first row of $\mathcal{M}_{\mathbf{r}^T}$ with $\mathcal{V}_{\mathbf{e}_1}$ is a sum of $256 \cdot 3$ independent random variables with the distribution in Table 2. And so the probability distribution of any coefficient of $\mathbf{r}^T\mathbf{e}_1$ is defined by the polynomial $\left((2/9)X^{-1} + 5/9 + (2/9)X\right)^{768}$.

We now move on to the next terms in (63). The distribution of e_3 is simply represented by the polynomial $(1/3)X^{-1}+1/3+(1/3)X$. We now need to compute the distribution of $\mathtt{LOW}_{\mathcal{S}_v}(v)$. It's not hard to see that if v is random, then $\mathtt{LOW}_{\mathcal{S}_v}(v)$ will also be uniform in $\left[\lceil q/2^4 \rfloor\right]$, except possibly having a slightly lower probability on the element

[6]The decryptor may want to save some time and already have $\hat{\mathbf{s}}$ in CRT form from the key generation phase. But this requires more storage since $\mathbf{s}$ is made up of small coefficients but $\hat{\mathbf{s}}$ isn't. If space is an issue, the decryptor doesn't even need to keep $\mathbf{s}$, but can just keep the seed he used to expand to create $\mathbf{s}$.

Table 2: Distribution of $ab \in \mathbb{Z}$, where $a, b \leftarrow [1]$. This is used to compute the distribution of each coefficient of $\mathbf{r}^T\mathbf{e}_1$.

-1	0	1
$2/9$	$5/9$	$2/9$

Table 3: Distribution of $a - b \in \mathbb{Z}$ where $a \leftarrow [1]$ and $b \leftarrow \{-2, -1, 0, 1\}$. This is used to compute the distribution of the coefficients of $\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})$ given the pessimistic simplification that each coefficient of $\mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})$ is uniformly $-1, 0, 1$, or 2.

-2	-1	0	1	2	3
$1/12$	$2/12$	$3/12$	$3/12$	$2/12$	$1/12$

(or both elements) with magnitude $\lceil q/2^4 \rfloor$. So pessimistically, for the decryption error, we will just assume that it has a uniform distribution over $\left[\lceil q/2^4 \rfloor\right]$. Concretely, it can be represented by the polynomial $\sum\limits_{i=-208}^{208} (1/417)X^i$.

Computing the distribution of each coefficient of $\mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})$ is done similarly. There are 2^{10} elements in the set $\mathcal{S}_u$ and thus there are either 3 or 4 spaces between 2 elements. The difference between every point in $\mathbb{Z}_q$ and an element in $\mathcal{S}_u$ is thus either $-2, -1, 0$, or 1 (we could replace -2 with 2, but it will not matter because we will eventually multiply by $\mathbf{s}$ which is symmetric around 0). The probability of $-1, 0$, and 1 is the same, while the probability of -2 is smaller. But for a simpler analysis (which will end up increasing the decryption error), let's just assume that each has a $1/4$ probability. Table 3 then has the probability distribution of each coefficient of $\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})$.

Now we can compute the distribution of each coefficient of $(\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u}))^T\mathbf{s}$ in the same way we computed the distribution of $\mathbf{r}^T\mathbf{e}_1$ above. In particular, the product of each coefficient of $\mathcal{M}_{\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})}$ with each coefficient of $\mathcal{V}_{\mathbf{s}}$ is computed in Table 4, and thus each coefficient of the product $\mathcal{M}_{\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u})} \cdot \mathcal{V}_{\mathbf{s}}$ is a sum of $256 \cdot 3$ random variables having this distribution.

Combining everything together, we compute the polynomial representing the probability distribution of each coefficient of (63) over $\mathbb{Z}$ as

$$\begin{aligned}
P(X) = \sum_{i \in \mathbb{Z}} p_i Z^i &= \left(\frac{2}{9}X^{-1} + \frac{5}{9} + \frac{2}{9}X\right)^{768} \\
&\cdot \left(\frac{1}{3}X^{-1} + \frac{1}{3} + \frac{1}{3}X\right) \\
&\cdot \left(\sum_{j=-208}^{208} \frac{1}{417}X^j\right) \\
&\cdot \left(\frac{1}{12}X^{-2} + \frac{2}{12}X^{-1} + \frac{3}{12} + \frac{3}{12}X + \frac{2}{12}X^2 + \frac{1}{12}X^3\right)^{768}
\end{aligned}$$

Table 4: Distribution of $ab \in \mathbb{Z}$ where a has the distribution in Table 3 and $b \leftarrow [1]$. This is used to compute the distribution of each of the coefficients of $(\mathbf{e}_2 - \mathtt{LOW}_{\mathcal{S}_u}(\mathbf{u}))^T\mathbf{s}$.

-3	-2	-1	0	1	2	3
$1/36$	$3/36$	$5/36$	$18/36$	$5/36$	$3/36$	$1/36$

```
\p 200;
default(parisize,1200000000); \\set precision and stacksize
pola=(2/9)*X^-1+5/9+(2/9)*X^1;
polb=(1/36)*X^-3+(3/36)*X^-2+(5/36)*X^-1+(18/36)
+(1/36)*X^3+(3/36)*X^2+(5/36)*X^1;
polv=0; for(n=-208,208,polv=polv+(1/417)*X^n);
pole=(1/3)*X^-1+1/3+(1/3)*X^1;
pp=(pola^768)*(polb^768)*polv*pole;
s=0; for(n=-832,832,s=s+polcoeff(pp,n));
ans=log(1-s)/log(2) \\ans=-203
```

Figure 3: PARI code for computing the probability of decryption errors. The `polb` line should not be entered as two separate lines into the PARI-gp shell.

and then compute the sum

$$\sum_{i=-832}^{832} p_i \tag{64}$$

to compute the lower bound on the probability that a coefficient of (63) is between $-q/4$ and $q/4$. Note that it's a lower bound because the polynomial $P(X)$ has terms of degree between -3281 and 3281, while the only possible coefficients are modulo 3329. So modulo $q = 3329$, the probability of, for example, 0 is $p_0 + p_{3329} + p_{-3329}$. But the values of the latter two terms is extremely small, and so the value of (64) is quite tight. To compute the probability of error, $1 - (64)$, one would use a script. Figure 3 has an example of such a computation in PARI [The18], which gives the probability of approxiamtely 2^{-203}. Using the union bound on all the 768 coefficients of (63), we obtain that the probability of all the coefficients being less than $q/4$ in magnitude is at least $1 - 2^{193}$.

5 Digital Signatures from Σ-Protocols

In this section, we will give a construction of a lattice-based digital signature scheme whose high-level structure is similar to the Schnorr signature scheme [Sch89] based on the hardness of the discrete logarithm problem. In a Schnorr digital signature scheme, the public key consists of two elements g, h in a finite field and the signature is a non-interactive Zero-Knowledge Proof of Knowledge (ZKPoK) of an exponent x such that $g^x = h$. The non-interactive proof is obtained in two steps. First, we construct a 3-move interactive Σ-protocol that's an honest-verifier ZKPoK of an x satisfying $g^x = y$. And secondly, we transform the interactive proof into a non-interactive one using the Fiat-Shamir transform.

5.1 The Goal for the ZKPoK

We would like to now follow a similar roadmap for constructing a signature scheme from the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ and $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m,\beta}$ problems. We can start with a $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ instance $(\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e})$, where $\mathbf{A} \leftarrow \mathcal{R}_{q,f}^{n\times m}$, $\mathbf{s} \leftarrow [\beta]^m, \mathbf{e} \leftarrow [\beta]^n$, make $\mathbf{A}, \mathbf{t}$ the public key and hope to be able to create a ZKPoK of $\mathbf{s}, \mathbf{e}$ in the appropriate range. Creating such a proof turns out to be significantly less efficient than in the discrete logarithm setting. The reason is that in addition to proving that the $\mathbf{s}, \mathbf{e}$ satisfy $\mathbf{A}\mathbf{s} + \mathbf{e} = \mathbf{t}$, we also need to prove that the coefficients of $\mathbf{s}, \mathbf{e}$ fall into a particular range (ideally $[\beta]$, but $[\bar{\beta}]$ for some $\bar{\beta}$ a little larger than β would also be OK). Anyone familiar with "range proofs" will instantly recognize that these proofs are often significantly more involved than proofs that simply prove knowledge of an element in the group (or ring) satisfying a certain relation.

The most efficient signatures that stem from Σ-protocols go around the need for proving knowledge of small $\mathbf{s}_1, \mathbf{s}_2$ satisfying

$$\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 = \mathbf{t}, \tag{65}$$

and instead prove knowledge of a relaxed solution to the equation. In particular, we will be giving protocols that allow a prover in possession of $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$ satisfying the above equation to prove knowledge of $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2$ in a somewhat larger interval than $[\beta]$, and another element $\bar{c}$ with small coefficients satisfying

$$\mathbf{A}\bar{\mathbf{s}}_1 + \bar{\mathbf{s}}_2 = \bar{c}\mathbf{t}. \tag{66}$$

5.1.1 The Challenge Space.

The size of the coefficients in $\bar{c}$ (as well as $\bar{\mathbf{s}}_1$ and $\bar{\mathbf{s}}_2$) will be dependent on the challenge space of the ZKPoK. Since we would like these to be small, we want to define our challenge space so that it consists of polynomials with small norms.
If we're working over the ring $\mathcal{R}_f$ where the degree of f is d, then define η to be the smallest integer such that $2^\eta \cdot \binom{d}{\eta} > 2^{256}$ (we assume that d is large enough such that such an η exists). Then we define the challenge set $\mathcal{C} \subset \mathcal{R}_f$ as

$$\mathcal{C} = \{c \in [1], \|c\|_1 = \eta\}, \tag{67}$$

and the set of all (non-zero) differences as

$$\bar{\mathcal{C}} = \{\bar{c} = c_1 - c_2, \text{ for } c_1 \neq c_2 \in \mathcal{C}\}. \tag{68}$$

So $\mathcal{C}$ consists of all polynomials in $\mathcal{R}_f$ with exactly η non-zero coefficients taken from the set $\{-1, 1\}$. By the definition of η, the size of $\mathcal{C}$ is greater than 2^{256}. Note that we could have defined $\mathcal{C}$ to also include polynomials with fewer than η non-zero coefficients, but this would increase the complexity of sampling a random element in $\mathcal{C}$ while not increasing its size by much.

5.2 The Basic Σ-Protocol

We now describe the basic scheme, given in Figure 4, which is a variant of the one presented in [Lyu09]. A distinctive feature of this protocol is that it does not have perfect completeness. In order to keep the coefficients of the output small, it will be necessary to perform a rejection sampling step in the last round of the Σ-protocol to make sure that the distribution is independent of the secret. In all the protocols here, the rejection sampling step will be quite simple – just checking whether all the coefficients are in a certain range. One can perform slightly more complicated rejection sampling steps, which require sampling and rejecting according to a discrete Gaussian distribution, which results in slightly smaller outputs [Lyu12, DDLL13]. The main downside of these latter algorithms is that the rejection sampling step is more complex and a slightly incorrect implementation may end up leaking the secret key. It may be therefore more preferable to have a simpler algorithm for such a widely-used primitive, like a digital signature.
The main consequence of the rejection sampling step is that the running time of the signing algorithm will be a random variable (but independent of $\mathbf{s}_1, \mathbf{s}_2$), rather than fixed. Other than this, a reader familiar with Schnorr-type proofs will find a lot of similarities in this protocol. The first stage of the protocol consists of creating the masking variables $\mathbf{y}_1$ and $\mathbf{y}_2$, the second step is the challenge, and the last step consists of adding the mask to the product of the challenge and the secret. A rejection sampling step is then performed, and if the prover sends $\perp$, he will need to restart the protocol.
The transformation of this protocol to a signature scheme will use the usual Fiat-Shamir transform, in which the challenge is created as a hash of the message and the first message of the prover. For verification, it is crucial that the verifier is able to recover the first message of the prover from the signature. In other words, the

Private information: $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$
Public information: $\mathbf{A} \in \mathcal{R}_{q,f}^{n\times m}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \in \mathcal{R}_{q,f}^n$

$$\begin{array}{lcl}
\underline{\text{Prover}} & & \underline{\text{Verifier}} \\
\mathbf{y}_1 \leftarrow [\gamma + \bar{\beta}]^m & & \\
\mathbf{y}_2 \leftarrow [\gamma + \bar{\beta}]^n, & & \\
\omega := \mathcal{H}(\mathbf{A}\mathbf{y}_1 + \mathbf{y}_2) & & \\
 & \xrightarrow{\omega} & \\
 & & c \leftarrow \mathcal{C} \\
 & \xleftarrow{c} & \\
\mathbf{z}_1 := c\mathbf{s}_1 + \mathbf{y}_1 & & \\
\mathbf{z}_2 := c\mathbf{s}_2 + \mathbf{y}_2 & & \\
\text{if } \mathbf{z}_1 \notin [\bar{\beta}]^m \text{ or } \mathbf{z}_2 \notin [\bar{\beta}]^n & & \\
\quad \text{then } (\mathbf{z}_1, \mathbf{z}_2) := \perp & & \\
 & \xrightarrow{(\mathbf{z}_1, \mathbf{z}_2)} & \\
 & & \text{Accept iff } \mathbf{z}_1 \in [\bar{\beta}]^m \text{ and } \mathbf{z}_2 \in [\bar{\beta}]^n \\
 & & \quad \text{and } \mathcal{H}(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}) = \omega
\end{array}$$

Figure 4: The basic Zero-Knowledge Proof System in which the prover knows $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$ satisfying (65) and gives a ZKPoK of knowledge of $\bar{\mathbf{s}}_1 \in [2\bar{\beta}]^m$, $\bar{\mathbf{s}}_2 \in [2\bar{\beta}]^n$, and a $\bar{c} \in \bar{\mathcal{C}}$ satisfying (66). The value γ is defined in Lemma 4, and the value of $\bar{\beta}$ affects the completeness of the protocol (i.e. the probability that $\perp$ is not sent) as specified in Lemma 4. The function $\mathcal{H}$ is a cryptographic hash function, modeled as a random oracle, mapping onto the space $\{0,1\}^{256}$.

first message in the Σ-protocol should be a deterministic function of the last two. While nothing particularly interesting happens with respect to this requirement in the basic protocol in Figure 4, the more efficient protocols in Figures 5 and 6 will require some additional checks / messages in order to make sure that this is the case. For this reason, we present our identification schemes such that the first message of the prover is already hashed. This is also a common trick for keeping the size of the output small in an interactive scheme – instead of sending a possibly long first message, the prover can send a short commitment to it.

5.2.1 Honest Verifier Zero-Knowledge

We will now prove that the protocol in Figure 4 is HVZK. That is, we will show how to create valid transcripts without knowledge of the secret $\mathbf{s}_1, \mathbf{s}_2$. The key to the proof is the below lemma which shows that for all $\mathbf{s}_1, \mathbf{s}_2$ with bounded coefficients, the probability of $\perp$ is the same (and computes this value), and furthermore shows that the distribution of $\mathbf{z}_1, \mathbf{z}_2$ are independent of $\mathbf{s}_1, \mathbf{s}_2$.

Lemma 4. *If $\gamma \in \mathbb{Z}^+$ is such for all $s \in [\beta], c \in \mathcal{C}$, $cs \in [\gamma]$, then for all $\mathbf{s}_i, c$,*

$$\Pr_{\mathbf{y}_1,\mathbf{y}_2}[(\mathbf{z}_1,\mathbf{z}_2) \neq \perp] = \left(\frac{2\bar{\beta}+1}{2(\bar{\beta}+\gamma)+1}\right)^{d(m+n)} \tag{69}$$

$$\forall \mathbf{z}_1' \in [\beta]^m, \mathbf{z}_2' \in [\beta]^n,\ \Pr_{\mathbf{y}_1,\mathbf{y}_2}\left[(\mathbf{z}_1,\mathbf{z}_2) = (\mathbf{z}_1',\mathbf{z}_2') \mid (\mathbf{z}_1,\mathbf{z}_2) \neq \perp\right] = \left(\frac{1}{2\bar{\beta}+1}\right)^{d(m+n)} \tag{70}$$

Proof. Consider $\begin{bmatrix}\mathbf{z}_1\\ \mathbf{z}_2\end{bmatrix} = \begin{bmatrix}c\mathbf{s}_1\\ c\mathbf{s}_2\end{bmatrix} + \begin{bmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{bmatrix}$ as a sum of integer vectors

$$\mathbf{z} = \begin{bmatrix}\mathcal{V}_{\mathbf{z}_1}\\ \mathcal{V}_{\mathbf{z}_2}\end{bmatrix} = \begin{bmatrix}\mathcal{V}_{c\mathbf{s}_1}\\ \mathcal{V}_{c\mathbf{s}_2}\end{bmatrix} + \begin{bmatrix}\mathcal{V}_{\mathbf{y}_1}\\ \mathcal{V}_{\mathbf{y}_2}\end{bmatrix} \in \mathbb{Z}^{d(n+m)}.$$

The probability that the i^{th} coefficient (for any i) of $\mathbf{z}$ is a particular coefficient $\nu_z \in [\bar{\beta}]$ is exactly $\frac{1}{2\bar{\beta}+1}$. The reason is the following: suppose that the i^{th} coefficient in $\begin{bmatrix}\mathcal{V}_{c\mathbf{s}_1}\\ \mathcal{V}_{c\mathbf{s}_2}\end{bmatrix}$ is ν_s, then i^{th} coefficient ν_y of the vector $\begin{bmatrix}\mathcal{V}_{\mathbf{y}_1}\\ \mathcal{V}_{\mathbf{y}_2}\end{bmatrix}$ will need to be exactly $\nu_z - \nu_s$. Notice that $\nu_z \in [\bar{\beta}]$ and $\nu_s \in [\gamma]$ implies that $\nu_z - \nu_s \in [\bar{\beta}+\gamma]$, which is exactly the range that the coefficient ν_y gets selected from. Therefore the probability that ν_y will be this value is exactly $\frac{1}{2(\bar{\beta}+\gamma)+1}$. Thus

$$\forall \mathbf{z}_1' \in [\bar{\beta}]^m, \mathbf{z}_2' \in [\bar{\beta}]^n, \Pr_{\mathbf{y}_1,\mathbf{y}_2}\left[(\mathbf{z}_1,\mathbf{z}_2) = (\mathbf{z}_1',\mathbf{z}_2')\right] = \left(\frac{1}{2(\bar{\beta}+\gamma)+1}\right)^{d(m+n)}. \quad (71)$$

And since there are $(2\bar{\beta}+1)^{d(m+n)}$ possible valid $(\mathbf{z}_1',\mathbf{z}_2') \in [\beta]^m \times [\beta]^n$ that could be sent, we obtain the claim in the first part of the lemma (i.e. (69)).
To obtain the second part of the lemma, we observe that (70) = (71)/(69).

The approximate probability of the prover not sending $\perp$ is

$$\left(\frac{2\bar{\beta}+1}{2(\bar{\beta}+\gamma)+1}\right)^{d(m+n)} > \left(\frac{\bar{\beta}}{\bar{\beta}+\gamma}\right) = \left(1+\frac{\gamma}{\bar{\beta}}\right)^{-d(m+n)} \approx e^{-\gamma d(m+n)/\bar{\beta}}, \quad (72)$$

and so setting $\bar{\beta} = -\gamma d(m+n)$ would result in the protocol requiring an expected number of e repetitions before a non-bot value is sent. One could of course set $\bar{\beta}$ to be smaller at the cost of a higher number of repetitions.
We now use Lemma 4 to show how to simulate a transcript with the correct probability. To simulate a random oracle query $\mathcal{H}(\mathbf{w})$, the simulator checks whether $\mathbf{w}$ was already assigned and, if not, chooses a uniformly random value $\omega \leftarrow \{0,1\}^{256}$ and programs $\mathcal{H}(\mathbf{w}) = \omega$.
To simulate a valid transcript, the simulator flips a coin, and with probability $1-(69)$, he outputs $(\omega \leftarrow \{0,1\}^{256}, c \leftarrow \mathcal{C}, \perp)$. Otherwise, with probability (69), he chooses a random $\mathbf{z}_1 \leftarrow [\bar{\beta}]^m$, $\mathbf{z}_2 \leftarrow [\bar{\beta}]^n$, $c \leftarrow \mathcal{C}$, $\omega \leftarrow \{0,1\}^{256}$, programs $\omega = \mathcal{H}(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t})$ (or if in the very unlikely event that the value of $\mathcal{H}(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t})$ was already set, he sets ω to this value), and outputs $(\omega, c, \mathbf{z}_1, \mathbf{z}_2)$.
In the case that the simulator does not send $\perp$, the distribution perfectly simulates the honest transcript because by Lemma 4, the values of $\mathbf{z}_1, \mathbf{z}_2$ are uniformly random conditioned on them not being $\perp$. In the case that the simulator outputs $\perp$, because the entropy of the $\mathbf{y}_i$ that force a $\perp$ in the real protocol is high, one can show that it's unlikely for $\mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$ to hit any particular point, and so it is valid (except with small probability) to output a completely random value ω every time. This completes the proof that the protocol in Figure 4 is HVZK.
Before continuing, we would like to make the important remark that the probability with which the honest prover will send $\perp$ (and thus will have to repeat the protocol) is *independent* of the secret $\mathbf{s}_1, \mathbf{s}_2$. This is important in real-world applications where any dependence of the running time on the secret would lead to side-channel attacks where the adversary attempts to deduce some information about the secret by observing the running time of the prover. The protocol in Figure 4 is therefore immune to this simple form of attack.

5.2.2 Proof of Knowledge

To show that the protocol is a PoK, we use the usual rewinding argument in which the prover sends ω and then successfully replies to two challenges c, c' with $(\mathbf{z}_1, \mathbf{z}_2)$

Private information: $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$
Public information: $\mathbf{A} \in \mathcal{R}_{q,f}^{n\times m}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2 \in \mathcal{R}_{q,f}^n$

Prover — Verifier

$\mathbf{y} \leftarrow [\gamma + \bar{\beta}]^m$
$\omega := \mathcal{H}(\mathtt{HIGH}_S(\mathbf{A}\mathbf{y}))$

$\xrightarrow{\omega}$

$c \leftarrow \mathcal{C}$

$\xleftarrow{c}$

$\mathbf{z} := c\mathbf{s}_1 + \mathbf{y}$
if $\mathbf{z} \notin [\bar{\beta}]^m$ or $\mathtt{LOW}_S(\mathbf{A}\mathbf{y} - c\mathbf{s}_2) \notin [\delta_S - \gamma]^n$
then $\mathbf{z} := \perp$

$\xrightarrow{\mathbf{z}}$

Accept iff $\mathbf{z} \in [\bar{\beta}]^m$
and $\mathcal{H}(\mathtt{HIGH}_S(\mathbf{A}\mathbf{z} - c\mathbf{t})) = \omega$

Figure 5: Basic Zero-Knowledge Proof System with a smaller output. The set $\mathcal{S} \in \mathbb{Z}_q$ has size 2^κ and the function $\mathtt{HIGH}_S, \mathtt{LOW}_S$, and the constant δ_S are defined as in the text of Section 5.3. The prover, who knows $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$ satisfying (65), produces a ZKPoK of $\bar{\mathbf{s}}_1 \in [2\bar{\beta}]^m$, $\bar{\mathbf{s}}_2 \in [q/2^\kappa]^n$, and a $\bar{c} \in \bar{\mathcal{C}}$ satisfying (66). The value γ is defined in Lemma 4, and the value of $\bar{\beta}$ affects the completeness of the protocol (i.e. the probability that $\perp$ is not sent) as specified in (77). The function $\mathcal{H}$ is a cryptographic hash function, modelled as a random oracle, mapping onto the space $\{0,1\}^{256}$.

and $(\mathbf{z}_1', \mathbf{z}_2')$. If we can extract such two transcripts $(\omega, c, \mathbf{z}_1, \mathbf{z}_2)$ and $(\omega, c', \mathbf{z}_1', \mathbf{z}_2')$ satisfying the verification equation, then (unless one can find a collision in $\mathcal{H}$) it must be that $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{A}\mathbf{z}_1' + \mathbf{z}_2' - c'\mathbf{t}$. The preceding simplifies to $\mathbf{A}(\mathbf{z}_1 - \mathbf{z}_1') + (\mathbf{z}_2 - \mathbf{z}_2') = (c - c')\mathbf{t}$, which is exactly the statement in (66).

5.3 Reducing the Proof Size

In this section we will show how to reduce the size of the proof by essentially removing the need to send $\mathbf{z}_2$ in Figure 4. The intuition is that to prove knowledge of (66), it's enough to output a proof corresponding to the $\bar{\mathbf{s}}_1$ such that $\mathbf{A}\bar{\mathbf{s}}_1 \approx \bar{c}\mathbf{t}$. Thus one does not, in principle, need to send the value corresponding to $\mathbf{z}_2$. One needs to be careful, though, to change the protocol so that it still remains zero-knowledge. The idea for not sending $\mathbf{z}_2$ appeared in [GLP12, BG14], and the protocol we present in Figure 5 is due to [BG14].

The idea, and execution, of not sending $\mathbf{z}_2$ is somewhat similar in spirit to the bit-dropping idea for shortening the ciphertext in Section 2.5.1. As in that section, suppose that we pick a set $\mathcal{S} \subset \mathbb{Z}_q$ of size 2^κ so that the distance between any two elements in this set is $\approx q/2^\kappa$ (see (18)). Let us also recall the notation from that section with which we can uniquely represent any $w \in \mathbb{Z}_q$ as $w = \mathtt{HIGH}_S(w) + \mathtt{LOW}_S(w)$, where $\mathtt{HIGH}_s(w) \in S$ and $\mathtt{LOW}_S(w) = w - \mathtt{HIGH}_S(w) \in [q/2^{\kappa+1}]$. This notation can be naturally extended to vectors over $\mathcal{R}_{q,f}$ by applying this decomposition to each integer coefficient of each polynomial.

Also, define δ_S to be the largest integer such that for all 2^κ elements $s_i \in \mathcal{S}$, the sets $s_i + [\delta_S]$ are all disjoint. If we pick the points in $\mathcal{S}$ such that they are equidistant from each other on the circle representing $\mathbb{Z}_q$, then δ_S will be approximately $q/2^{\kappa+1}$, which is also the maximum value, over all $w \in \mathbb{Z}_q$, of $\mathtt{LOW}_S(w)$. For the rest of the section, we will assume that $\mathcal{S}$ is picked in such a manner. An important simple observation is that for all positive $\gamma < \delta_S$,

$$\mathtt{LOW}_S(w) \in [\delta_S - \gamma] \text{ and } s \in [\gamma] \Longrightarrow \mathtt{HIGH}_S(w) = \mathtt{HIGH}_S(w + s) \tag{73}$$

With the above notation, consider the protocol in Figure 5. We will show that it is a proof of knowledge of $\bar{\mathbf{s}}_1 \in [2\bar{\beta}]^m, \bar{\mathbf{s}}_2 \in [q/2^\kappa]^n, \bar{c} \in \bar{\mathcal{C}}$ satisfying (66).

5.3.1 Correctness

For correctness (in the case that $\mathbf{z} \neq \perp$), we need to show that $\mathtt{HIGH}_S(\mathbf{Ay}) = \mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t})$. If we write

$$\mathbf{Az} - c\mathbf{t} = \mathbf{A}(c\mathbf{s}_1 + \mathbf{y}) - c(\mathbf{As}_1 + \mathbf{s}_2) = \mathbf{Ay} - c\mathbf{s}_2, \tag{74}$$

we know by one of the Prover's conditions for not sending $\perp$ that $\mathtt{LOW}_S(\mathbf{Ay} - c\mathbf{s}_2) \in [\delta_S - \gamma]^n$. The latter implies, by the observation in (73), that $\mathtt{HIGH}_S(\mathbf{Ay}) = \mathtt{HIGH}_S(\mathbf{Ay} - c\mathbf{s}_2)$, and therefore $\mathtt{HIGH}_S(\mathbf{Ay}) = \mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t})$.

5.3.2 Zero-Knowledge

The key to showing zero-knowledge is the observation that the simulator only needs to know the probability with which $\mathbf{z} = c\mathbf{s}_1 + \mathbf{y}$ causes a $\perp$, and the distribution of $\mathbf{z}$ given that it does not cause a $\perp$. The other condition that causes a $\perp$ can then be checked by computing using $\mathbf{Az} - c\mathbf{t}$ (see (74)).
Notice that the simulator does not know how to check whether $\mathtt{LOW}_S(\mathbf{Ay} - c\mathbf{s}_2) \in [\delta_S - \gamma]^n$ in the case that $\mathbf{z} \notin [\bar{\beta}]^m$ (because in this case the distribution of $\mathbf{z}$ depends on $\mathbf{s}_1$); but this is not important because such a $\mathbf{z}$ will cause $\perp$ anyway.
From the proof of Lemma 4, we know that the probability that $\mathbf{z}$ causes a $\perp$ is exactly

$$1 - \left(\frac{2\bar{\beta} + 1}{2(\bar{\beta} + \gamma) + 1}\right)^{dm}, \tag{75}$$

and conditioned on $\mathbf{z} \in [\bar{\beta}]^m$, its distribution is uniform.
The simulation, therefore, proceeds as follows: the simulator tosses a coin and with probability (75), sends $(\omega \leftarrow \{0,1\}^{256}, c \leftarrow \mathcal{C}, \perp)$. Otherwise, he picks $\mathbf{z} \leftarrow [\bar{\beta}]^m, c \leftarrow \mathcal{C}$ and checks whether $\mathtt{LOW}_S(\mathbf{Az} - c\mathbf{t}) \in [\delta_S - \gamma]^n$. If it is, then it sends $\mathbf{z}$, and sends $\perp$ otherwise.

5.3.3 Computing the probability of $\perp$

From (75), we know that $\Pr_{\mathbf{y}}\left[\mathbf{z} \in [\bar{\beta}]^m\right] = \left(\frac{2\bar{\beta}+1}{2(\bar{\beta}+\gamma)+1}\right)^{dm} \approx e^{-\gamma dm/\bar{\beta}}$ (see (72)). To compute the probability that $\mathtt{LOW}_S(\mathbf{Ay} - c\mathbf{s}_2) \in [\delta_S - \gamma]^n$, we make the heuristic assumption that the distribution of $\mathbf{Ay} - c\mathbf{s}_2$ is uniform in $\mathcal{R}_{q,f}^n$, and thus $\mathtt{LOW}_S(\mathbf{Ay} - c\mathbf{s}_2)$ is uniformly distributed in $[\delta_S]^n$. Therefore the probability that a random element in $[\delta_S]^n$ is inside $[\delta_S - \gamma]^n$ is

$$\left(\frac{2(\delta_S - \gamma) + 1}{2\delta_S + 1}\right)^{dn} > \left(1 - \frac{\gamma}{\delta_S}\right)^{dn} \approx e^{-\gamma dn/\delta_S}. \tag{76}$$

Combining the two probabilities, we obtain that

$$\Pr_{\mathbf{y}}[\mathbf{z} \neq \perp] \approx e^{-\gamma d(m/\bar{\beta} + n/\delta_S)}. \tag{77}$$

Note that a larger $\bar{\beta}$ and a larger δ_S increase the correctness probability of the protocol. But as we will see below, the larger these values are, the larger the coefficients in the extracted $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2$ satisfying (66) will be.

Private information: $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$
Public information: $\mathbf{A} \in \mathcal{R}_{q,f}^{n\times m}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ (not used by the verifier), $\mathbf{t}_1 = \mathtt{HIGH}_{\mathcal{T}}(\mathbf{t})$

Prover		Verifier
$\mathbf{y} \leftarrow [\gamma + \bar{\beta}]^m$		
$\omega := \mathcal{H}(\mathtt{HIGH}_{\mathcal{S}}(\mathbf{A}\mathbf{y}))$		
	$\xrightarrow{\omega}$	
		$c \leftarrow \mathcal{C}$
	$\xleftarrow{c}$	
$\mathbf{z} := c\mathbf{s}_1 + \mathbf{y}$		
if $\mathbf{z} \notin [\bar{\beta}]^m$ or $\mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2) \notin [\delta_{\mathcal{S}} - \gamma]^n$		
then $(\mathbf{z}, \mathbf{h}) := \perp$		
if $\mathbf{z} \neq \perp$, then $\mathbf{h} := \mathtt{HINT}(\mathbf{A}\mathbf{z} - c\mathbf{t}_1, c\mathbf{t}_0)$		
	$\xrightarrow{(\mathbf{z},\mathbf{h})}$	
		Accept iff $\mathbf{z} \in [\bar{\beta}]^m$
		and $\mathcal{H}(\mathtt{USEHINT}(\mathbf{A}\mathbf{z} - c\mathbf{t}_1, \mathbf{h})) = \omega$

Figure 6: The Zero-Knowledge Proof System with a smaller proof and a smaller public Key. The set $\mathcal{S} \in \mathbb{Z}_q$ has size 2^κ and the function $\mathtt{HIGH}_{\mathcal{S}}, \mathtt{LOW}_{\mathcal{S}}$, and the constant $\delta_{\mathcal{S}} \approx q/2^{\kappa+1}$ are defined as in the text of Section 5.3. The set $\mathcal{T}$ has size 2^ℓ and we require that with high probability (over the choice of c and $\mathbf{t}$), $c \cdot \mathtt{LOW}_{\mathcal{T}}(\mathbf{t}) \in [\delta_{\mathcal{S}}]^n$. The functions HINT and USEHINT are defined as in the text of Section 5.4. The prover, who knows $\mathbf{s}_1 \in [\beta]^m, \mathbf{s}_2 \in [\beta]^n$ satisfying (65), produces a ZKPoK of $\bar{\mathbf{s}}_1 \in [2\bar{\beta}]^m$, $\bar{\mathbf{s}}_2 \in [q/2^{\kappa-1}]^n$, and a $\bar{c} \in \bar{\mathcal{C}}$ satisfying (81). The value γ is defined in Lemma 4, and the value of $\bar{\beta}$ affects the completeness of the protocol (i.e. the probability that $\perp$ is not sent) as specified in (77). The function $\mathcal{H}$ is a cryptographic hash function, modelled as a random oracle, mapping onto the space $\{0,1\}^{256}$.

5.3.4 Proof of Knowledge

Via rewinding, the extractor can obtain two transcripts $(\omega, c, \mathbf{z})$ and $(\omega, c', \mathbf{z}')$ that satisfy the verification equations. Unless one can find a collision in $\mathcal{H}$, it must be that $\mathtt{HIGH}_{\mathcal{S}}(\mathbf{A}\mathbf{z} - c\mathbf{t}) = \mathtt{HIGH}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t})$. By definition, we have

$$\mathbf{A}\mathbf{z} - c\mathbf{t} = \mathtt{HIGH}_{\mathcal{S}}(\mathbf{A}\mathbf{z} - c\mathbf{t}) + \mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z} - c\mathbf{t})$$
$$\mathbf{A}\mathbf{z}' - c'\mathbf{t} = \mathtt{HIGH}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t}) + \mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t})$$

where $\mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z} - c\mathbf{t}), \mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z} - c\mathbf{t}) \in [q/2^{\kappa+1}]^n \approx [\delta_{\mathcal{S}}]^n$. Subtracting the two above equations, we obtain

$$\mathbf{A}(\mathbf{z} - \mathbf{z}') - (c - c')\mathbf{t} = \mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t}) - \mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t}) \in [q/2^\kappa]^n, \qquad (78)$$

which is equivalent to (66) when we set $\bar{\mathbf{s}}_2$ to $\mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t}) - \mathtt{LOW}_{\mathcal{S}}(\mathbf{A}\mathbf{z}' - c'\mathbf{t})$.

5.4 Reducing the Public Key Size

Similarly to the intuition for removing the need to transmit $\mathbf{z}_2$ from the protocol in Figure 4 in the previous section due to the fact that $\mathbf{A}\mathbf{z}_1 \approx c\mathbf{t}$, we also notice that if we write $\mathbf{t} = \mathtt{HIGH}_{\mathcal{T}}(\mathbf{t}) + \mathtt{LOW}_{\mathcal{T}}(\mathbf{t})$ (where $\mathcal{T} \subset \mathbb{Z}_q$), then $\mathbf{A}\mathbf{z}_1 \approx c(\mathtt{HIGH}_{\mathcal{T}}(\mathbf{t}))$. In other words, the verifier does not need to know the low-order bits of $\mathbf{t}$ in order to *approximately* verify the verification equation.
By not making $\mathtt{LOW}_{\mathcal{T}}(\mathbf{t})$ a part of the public key, we again run into a similar problem as in the previous section in that without some adjustments to the protocol, the

verifier will not be able to compute the same value inside $\mathcal{H}$ as the prover. Notice that in the equation (74), the verifier did not need to know $\mathbf{s}_2$ because he knew $\mathbf{t}$. But now, he doesn't even know $\mathbf{t}$! We will now describe the techniques needed to make the proof work and present the protocol, from [DKL+18], in Figure 6.

Similarly to the way in which we defined the set $S \subset \mathbb{Z}_q$ in the previous section, we define a set $\mathcal{T} \subset \mathbb{Z}_q$ that consists of 2^ℓ elements each a distance of approximately $q/2^\ell$ apart. Instead of outputting the entire vector $\mathbf{t} \in \mathcal{R}_{q,f}^n$ as part of the public key, we will decompose it as $\mathbf{t} = \mathbf{t}_1 + \mathbf{t}_0$, where $\mathbf{t}_1 = \mathtt{HIGH}_{\mathcal{T}}(\mathbf{t})$ and $\mathbf{t}_0 = \mathtt{LOW}_{\mathcal{T}}(\mathbf{t})$. The public key will only consist of $\mathbf{t}_1$, which requires $nd\ell$ bits to represent instead of $nd \log q$ required to represent the entire $\mathbf{t}$.

The place where the verifies used $\mathbf{t}$ in the protocol in Figure 5 was in the computation of $\mathbf{Az} - c\mathbf{t}$. If the verifier only has $\mathbf{t}_1$, then he can compute $\mathbf{Az} - c\mathbf{t}_1 = \mathbf{Az} - c\mathbf{t} + c\mathbf{t}_0$. In order for verification to work out, we would need

$$\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t}) = \mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t} + c\mathbf{t}_0). \tag{79}$$

At this point, the natural question is why not resolve this issue by applying the same technique as in the previous section? That is, we can make sure that $c\mathbf{t}_0 < \gamma'$ for some γ' and then use the observation in (73) to force the above equation to always hold. The problem with this approach is that it's wasteful and unlikely to result in a useful reduction in size. The reason we upper-bounded the coefficients of $c\mathbf{s}_2$ (over all possible secrets $\mathbf{s}_2$) by γ was that we needed to keep $\mathbf{s}_2$ a secret. On the other hand, we do not need to keep $\mathbf{t}_0$ a secret – we simply want it to be unnecessary for verification. So it's perfectly fine if the verifier is able to compute something that strongly depends on $\mathbf{t}_0$ – i.e. $\mathbf{Az} - c\mathbf{t} + c\mathbf{t}_0$. Our only goal here is that a verifier who knows $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t} + c\mathbf{t}_0)$ to be able to derive $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t})$.

The main observation is that if $c\mathbf{t}_0 \in [\delta_S]^n$, then a verifier who can compute $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t} + c\mathbf{t}_0)$ would only require one bit of extra information (per coefficient) to determine $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t})$. In particular, if some integer point v is between points s_i and s_{i+1} in the set $\mathcal{S}$ on the circle representing $\mathbb{Z}_q$, and we add to it an integer point $v' \in [\delta_S]$, then the closest point to $v + v'$ in S can only be s_i or s_{i+1} (i.e. $\mathtt{HIGH}_S(v + v') = s_i$ or s_{i+1}). The prover, who knows, v and v' can provide this one bit of information to the verifier. In other words, he can tell the verifier whether $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t}) = \mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t} + c\mathbf{t}_0)$ or not. Either way, the verifier can now determine $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t})$. Using this technique, one can significantly reduce the size of the public key at the cost of adding an extra dn bits to the signature.

As notation, we will write $\mathtt{HINT}(\mathbf{Az} - c\mathbf{t}_1, c\mathbf{t}_0) = \mathbf{h} \in \{0,1\}^{dn}$ to be the vector of hints necessary to recover $\mathtt{HIGH}_S(\mathbf{Az} - c\mathbf{t})$ from $\mathbf{Az} - c\mathbf{t}_1$. We'll write $\mathtt{USEHINT}(\mathbf{Az} - c\mathbf{t}_1, \mathbf{h})$ to be this recovery procedure. In particular, $\mathtt{USEHINT}(\mathbf{Az} - c\mathbf{t}_1, \mathbf{h})$ computes $\mathbf{v} = \mathbf{Az} - c\mathbf{t}_1$ and then for each integer coefficient of $\mathbf{v}$ (which lies between s_i and s_{i+1} in $\mathcal{S}$), it uses the corresponding bit of $\mathbf{h}$ to either output the closer s_i (if the hint bit is 0) or the one that is further away (if the hint bit is 1). In practice, it'll usually turn out that the majority of the time the hint bit will be 0, and so the hint vector can be more efficiently represented by just enumerating the few positions where the hint bit should be 1.

With this notation, the algorithm with compressed signatures and public keys is presented in Figure 6. It will be important for the security proof to note that for any vector $\mathbf{v} \in \mathcal{R}_{q,f}^n$ and a hint vector $\mathbf{h} \in \{0,1\}^{dn}$, we have

$$\mathbf{v} - \mathtt{USEHINT}(\mathbf{v}, \mathbf{h}) \in [q/2^\kappa]^n. \tag{80}$$

The above is directly implied by the $\mathtt{USEHINT}$ procedure and the fact that the distance between two neighboring points in $\mathcal{S}$ is $q/2^\kappa$.

The statement that is proved by the protocol in Figure 6 is the knowledge of $\bar{\mathbf{s}}_1 \in [2\bar{\beta}]^m$, $\bar{\mathbf{s}}_2 \in [q/2^{\kappa-1}]^n$, and $\bar{c} \in \bar{\mathcal{C}}$ satisfying

$$\mathbf{A}\bar{\mathbf{s}}_1 + \bar{\mathbf{s}}_2 = \bar{c}\mathbf{t}_1. \tag{81}$$

If we would like to relate this to the original $\mathbf{t}$ and satisfy (66), we can substitute $\mathbf{t}_1 = \mathbf{t} - \mathbf{t}_0$ and rewrite the above equation as $\mathbf{A}\bar{\mathbf{s}}_1 + (\bar{\mathbf{s}}_2 + \bar{c}\mathbf{t}_0) = \bar{c}\mathbf{t}$ and so the length of the vector $\bar{\mathbf{s}}_2$ gets increased by the maximum possible value (over all $\bar{c} \in \bar{\mathcal{C}}$) of $\bar{c}\mathbf{t}_0$.

5.4.1 Correctness and Zero-Knowledge

An important point that is worth repeating is that while the verifier does not need the value of $\mathbf{t}_0 = \mathtt{LOW}_{\mathcal{T}}(\mathbf{t})$ for verification, one *should not* consider the value $\mathbf{t}_0$ to be secret because outputting $\mathbf{h}$ leaks some information about $\mathbf{t}_0$. The way to think about $\mathbf{t}$ for the proofs is that the verifier knows the entire $\mathbf{t}$, but only uses $\mathbf{t}_1$ in verification. The zero-knowledge property of the scheme immediately follows from the zero-knowledge property of the scheme in Figure 5, which we already established in Section 5.3.2, because the only difference in the prover's output is the construction of the hint $\mathbf{h}$, which can be done knowing $\mathbf{z}$ and $\mathbf{t}$.
The correctness of the scheme follows whenever $c\mathbf{t}_0 \in [\delta_S]^n$. Note that it doesn't effect the security of the scheme if $\mathbf{t}_0$ is chosen such that $c\mathbf{t}_0$ is sometimes not in $[\delta_S]^n$ – though this will require additional restarts on the part of the prover.

5.4.2 Proof of Knowledge

Via rewinding, one obtains the transcripts $(\omega, c, (\mathbf{z}, \mathbf{h}))$ and $(\omega, c', (\mathbf{z}', \mathbf{h}'))$. Unless the prover can find a collision in $\mathcal{H}$, it must be that

$$\mathtt{USEHINT}(\mathbf{A}\mathbf{z} - c\mathbf{t}_1, \mathbf{h}) = \mathtt{USEHINT}(\mathbf{A}\mathbf{z}' - c'\mathbf{t}_1, \mathbf{h}'). \tag{82}$$

From (80), we know that

$$\begin{aligned}\mathbf{A}\mathbf{z} - c\mathbf{t}_1 - \mathtt{USEHINT}(\mathbf{A}\mathbf{z} - c\mathbf{t}_1, \mathbf{h}) &\in [q/2^\kappa]^n \\ \mathbf{A}\mathbf{z}' - c'\mathbf{t}_1 - \mathtt{USEHINT}(\mathbf{A}\mathbf{z}' - c'\mathbf{t}_1, \mathbf{h}) &\in [q/2^\kappa]^n,\end{aligned}$$

which together with (82) implies that

$$\mathbf{A}(\mathbf{z} - \mathbf{z}') - (c - c')\mathbf{t}_1 \in [q/2^{\kappa-1}]^n, \tag{83}$$

which in turn directly implies knowledge of $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2$, and $\bar{c}$ as in (81).

5.5 Digital Signature

The signing procedure in Figure 7 is the Fiat-Shamir transform of the protocol in Figure 6, where the secret keys $\mathbf{s}_1, \mathbf{s}_2$ are chosen uniformly at random from their respective domains. A minor point that we draw attention to is that a true Fiat-Shamir transform of the protocol in Figure 6 would have $c := \mathcal{H}(\mathcal{H}(\mathtt{HIGH}_S(\mathbf{Ay}), \mu)$, but we removed the inner $\mathcal{H}$ call because it's unnecessary to nest $\mathcal{H}$'s when they're modelled as random oracles. Another point to notice is that because the protocol is no longer interactive, there is never a need for the prover to send $\perp$ – he can simply keep restarting the protocol until the rejection sampling step is successful.
The correctness and the zero-knowledge properties of the protocol follow directly from these respective properties of the interactive protocol in Figure 6. The security proof of the signature scheme proceeds in two steps. In the first step, we use the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ assumption to argue that $(\mathbf{A}, \mathbf{t})$ looks uniformly random. Then, via the generic properties of the Fiat-Shamir transform, we know that one can extract from a successful signer the same thing as from a successful prover in Section 5.4.2 – $\bar{\mathbf{s}}_1, \bar{\mathbf{s}}_2$ satisfying (81). The security of the scheme therefore relies on the hardness of the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ and the $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m+1,\beta}$ problems. Recall that the hardness of the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ problem depends on m, whereas the hardness of the $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m+1,\beta}$ problem depends on n. For this reason, it could make sense (see Section 5.6 to set the parameters such that $m \neq n$.
An interesting observation is that the protocol is a proof of knowledge of (81), but it is possible to set parameters such that *for a random* $\mathbf{t}$ no solution to (81) with

Private information: $\mathbf{s}_1 \leftarrow [\beta]^m, \mathbf{s}_2 \leftarrow [\beta]^n$
Public information: $\mathbf{A} \in \mathcal{R}_{q,f}^{n\times m}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ (not used by the verifier), $\mathbf{t}_1 = \mathtt{HIGH}_T(\mathbf{t})$

<u>Signer</u> <u>Verifier</u>

$\mathbf{y} \leftarrow [\gamma + \bar{\beta}]^m$
$c := \mathcal{H}(\mathtt{HIGH}_S(\mathbf{A}\mathbf{y}), \mu) \in \mathcal{C}$
$\mathbf{z} := c\mathbf{s}_1 + \mathbf{y}$
if $\mathbf{z} \notin [\bar{\beta}]^m$ or $\mathtt{LOW}_S(\mathbf{A}\mathbf{y} - c\mathbf{s}_2) \notin [\delta_S - \gamma]^n$
then RESTART
$\mathbf{h} := \mathtt{HINT}(\mathbf{A}\mathbf{z} - c\mathbf{t}_1, c\mathbf{t}_0)$

$$\xrightarrow{(\mathbf{z},\mathbf{h})}$$

Accept iff $\mathbf{z} \in [\bar{\beta}]^m$
and $\mathcal{H}(\mathtt{USEHINT}(\mathbf{A}\mathbf{z} - c\mathbf{t}_1, \mathbf{h})) = \omega$

Figure 7: Digital Signature Scheme obtained as a result of applying the Fiat-Shamir transform to the protocol in Figure 6 that signs a message (digest) μ.

the $\bar{\mathbf{s}}_i$ in the prescribed ranges exists – in other words the $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m,\beta}$ problem is vacuously hard, and so the security of the signature scheme is only based on $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$. Furthermore, in this scenario, the proof can be altered so that rewinding is unnecessary – and one only needs to prove that it is highly unlikely that even an all-powerful prover is able to respond to more than one challenge $c \in \mathcal{C}$ [AFLT12, ABB+17, KLS18]. This latter probability is at most the probability that (81) can be satisfied with some $\bar{\mathbf{s}}_i$ in their respective sets.

The lack of rewinding allows the proof to be tighter (the Fiat-Shamir transform of a Σ-protocol that uses rewinding in the extraction loses a factor of the number of queries) and allows for more standard (and tighter) assumptions when proving security in the quantum random oracle model. The major downside of a proof in which parameters have to be set so that a solution to (81) doesn't exist is that one needs to have the secret keys $\mathbf{s}_i$ be significantly smaller with respect to the modulus q. This makes the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ problem easier and subsequently requires us to increase the dimension of the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ problem (by increasing d or the dimension of the matrix) and results in a larger signature and public key size. The most efficient setting of the parameters is when the $\mathcal{R}_{q,f}$-$\mathsf{LWE}_{n,m,\beta}$ and the $\mathcal{R}_{q,f}$-$\mathsf{SIS}_{n,m,\beta}$ problems end up being of the same hardness.

5.6 A Concrete Example of a Signature Scheme

We now give a sample instantiation for a digital signature scheme very much resembling [DKL+18], which is (conservatively) estimated to have 128-bits of security. We do not discuss how one would go about efficiently implementing the scheme, since it uses the same main ideas as the encryption scheme implementation in Section 4.7.

We will be working over the ring $\mathcal{R}_{q,f}$ for $f = X^{256} + 1$ and $q \approx 2^{23}$. We will select q to be a prime that allows for efficient NTT (i.e. $q \equiv 1 \pmod{512}$) as in Section 4.4.2. The challenge set $\mathcal{C}$ consists of polynomials with $0, \pm 1$ coefficients with exactly 196 $0's$ (and thus 60 non-zeros) and the secrets $\mathbf{s}_1, \mathbf{s}_2$ have coefficients randomly chosen from $[\beta]$ for $\beta = 5$. Since the product of $\mathbf{s}_1$ (similarly $\mathbf{s}_2$) and $c \in \mathcal{C}$ can be written as $\mathcal{M}_{\mathbf{s}_1} \cdot \mathcal{V}_c$, we see that each coefficient of the resulting vector is a sum of 60 integers uniformly distributed in $[\beta]$. Representing this probability distribution by the coefficients of the polynomial $P(X) = \left(\sum_{i=-5}^{5} \frac{1}{11} X^i\right)^{60}$ as in Section 2.3.2, we

Table 5: Sample parameters for a digital signature scheme very similar to CRYSTALS-Dilithium [DKL+18].

q	$\approx 2^{23}$ s.t. $q \equiv 1 \pmod{512}$
$f(X)$	$X^{256}+1$
β	5
(n,m)	$(5,4)$
$\mathcal{C}$	$c \in [1]$ with 60 $\pm 1's$ and 196 $0's$
γ	275
$\bar{\beta}$	$(q-1)/16$
$\mathcal{S}$	$\{i \cdot (q-1)/16 \mid 0 \le i \le 15\}$
$\delta_{\mathcal{S}}$	$(q-1)/32+1$
$\mathcal{T}$	$\{\lceil i \cdot q/2^9 \rfloor \mid 0 \le i < 2^9\}$

obtain $\sum_{i=-275}^{275} P_i > 1 - 2^{137}$, where the P_i are the coefficients of $P(X)$ corresponding to X^i, and so

$$\Pr_{\mathbf{s}_1 \leftarrow [\beta]^4}[\mathbf{s}_1 c \in [275]^4] > 1 - 256 \cdot 4 \cdot 2^{-137} > 1 - 2^{-127}.$$

The same calculation holds for $\mathbf{s}_2$, so we can set $\gamma = 275$.
The set $\mathcal{S}$ and $\mathcal{T}$ are then defined as in Table 5. The definition of set $\mathcal{S}$ implies that every point in $\mathbb{Z}_q$ is at most $(q-1)/32+1$ away from any element in $\mathcal{S}$. Similarly, the definition of $\mathcal{T}$ implies that all the coefficients of $\mathbf{t}_0$ are in $[2^{13}]$. One can now check that $c\mathbf{t}_0$ has coefficients in $[\delta_{\mathcal{S}}]$ with high probability and so the scheme will be correct (with high probability). To make the scheme always correct, the prover should additionally check that $c\mathbf{t}_0 \in [\delta_{\mathcal{S}}]^5$, but we ignored this part in the description of the scheme.
The probability that a restart does not occur is computed from (77) as approximately

$$e^{-275 \cdot 256 \cdot (4 \cdot 16/(q-1) + 5 \cdot 32/(q-1))} \approx 0.15,$$

which means that one needs, on average, about 6.5 signing attempts before a signature is produced. This makes the signing procedure noticeably slower than verification. Still, optimized implementations of the signing procedure runs in well under a millisecond on a standard personal computer.
The public key consists of a 256-bit seed ρ that is used in the expansion of the public matrix $\mathbf{A}$ and the vector $\mathbf{t}_1 = \mathtt{HIGH}_{\mathcal{S}}(\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2)$. Since the set $\mathbf{S}$ can be described with 9 bits, and there are $5 \cdot 256$ integer elements in $\mathbf{t}_1$, the description of $\mathbf{t}_1$ requires $5 \cdot 256 \cdot 9$ bits.
The signature consists of the vector $\mathbf{z}_1$, the challenge c, and the hint vector $\mathbf{h}$. Since $\mathbf{z}_1 \in [\bar{\beta}]^4$, its representation requires 20 bits per coefficient, for a total of $256 \cdot 4 \cdot 20$ bits. The challenge c consists of approximately 256 bits, while the hint vector $\mathbf{h}$ is a binary vector of dimension $256 \cdot 5$, and so requires at most that many bits to represent. As we mentioned earlier, in practice the vector $\mathbf{h}$ may be quite sparse and could require fewer bits to represent.

References

[ABB+17] Erdem Alkim, Nina Bindel, Johannes A. Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting TESLA in the quantum random oracle model. In *PQCrypto*, volume 10346 of *Lecture Notes in Computer Science*, pages 143–162. Springer, 2017.

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In *CRYPTO (1)*, volume 9814 of *Lecture Notes in Computer Science*, pages 153–178. Springer, 2016.

[ABD$^+$19] Roberto Avanzi, Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: Algorithm specifications and supporting documentation, 2019.

[ACD$^+$18] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer. Estimate all the {LWE, NTRU} schemes! In *SCN*, volume 11035 of *Lecture Notes in Computer Science*, pages 351–367. Springer, 2018.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, pages 595–618, 2009.

[ADH$^+$19] Martin R. Albrecht, Léo Ducas, Gottfried Herold, Elena Kirshanova, Eamonn W. Postlethwaite, and Marc Stevens. The general sieve kernel and new records in lattice reduction. In *EUROCRYPT (2)*, volume 11477 of *Lecture Notes in Computer Science*, pages 717–746. Springer, 2019.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In *USENIX Security Symposium*, pages 327–343. USENIX Association, 2016.

[ADRS15] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. Solving the shortest vector problem in 2^n time using discrete gaussian sampling: Extended abstract. In *STOC*, pages 733–742. ACM, 2015.

[AFLT12] Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. Tightly-secure signatures from lossy identification schemes. In *EUROCRYPT*, pages 572–590, 2012.

[AG11] Sanjeev Arora and Rong Ge. New algorithms for learning in presence of errors. In *ICALP (1)*, pages 403–415, 2011.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–495. Springer, 2009.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In *CRYPTO (1)*, volume 8042 of *Lecture Notes in Computer Science*, pages 57–74. Springer, 2013.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610. ACM, 2001.

[AM18] Divesh Aggarwal and Priyanka Mukhopadhyay. Improved algorithms for the shortest vector problem and the closest vector problem in the infinity norm. In *ISAAC*, volume 123 of *LIPIcs*, pages 35:1–35:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[AS18] Divesh Aggarwal and Noah Stephens-Davidowitz. Just take the average! an embarrassingly simple 2ˆn-time algorithm for SVP (and CVP). In *SOSA@SODA*, volume 61 of *OASICS*, pages 12:1–12:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

[BCD$^+$16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In *ACM Conference on Computer and Communications Security*, pages 1006–1018. ACM, 2016.

[BDK$^+$18] Joppe W. Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS - kyber: A cca-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy, EuroS&P*, pages 353–367, 2018.

[BG14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In *CT-RSA*, pages 28–47, 2014.

[BGM$^+$16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In *TCC (A1)*, volume 9562 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2016.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for ntru problems and cryptanalysis of the ggh multilinear map without a low-level encoding of zero. *LMS Journal of Computation and Mathematics*, 19(A):255–266, 2016.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *CRYPTO (1)*, pages 40–56, 2013.

[Den02] Alexander W. Dent. A designer's guide to kems. *IACR Cryptology ePrint Archive*, 2002. http://eprint.iacr.org/2002/174.

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the fiat-shamir transformation in the quantum random-oracle model. In *CRYPTO (2)*, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383. Springer, 2019.

[DKL$^+$18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):238–268, 2018.

[DM13] Nico Döttling and Jörn Müller-Quade. Lossy codes and a new variant of the learning-with-errors problem. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2013.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO*, pages 537–554, 1999.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES*, pages 530–547, 2012.

[GN08] Nicolas Gama and Phong Q. Nguyen. Predicting lattice reduction. In *EUROCRYPT*, pages 31–51, 2008.

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

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *TCC*, pages 341–371, 2017.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[HRSS17] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. High-speed key encapsulation from NTRU. In *CHES*, pages 232–252, 2017.

[KF17] Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 3–26, 2017.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586. Springer, 2018.

[LLL82] Arjen Lenstra, Hendrik Lenstra Jr., and Laszlo Lovasz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, (261):513–534, 1982.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155, 2006.

[LP11] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA*, pages 319–339, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.

[LPR13a] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43, 2013. Preliminary version appeared in EUROCRYPT 2010.

[LPR13b] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT*, pages 35–54, 2013.

[LPS10] Vadim Lyubashevsky, Adriana Palacio, and Gil Segev. Public-key cryptographic primitives provably as secure as subset sum. In *TCC*, pages 382–400, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015.

[LS19] Vadim Lyubashevsky and Gregor Seiler. NTTRU: truly fast NTRU using NTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):180–201, 2019.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, pages 598–616, 2009.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, pages 738–755, 2012.

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum fiat-shamir. In *CRYPTO (2)*, volume 11693 of *Lecture Notes in Computer Science*, pages 326–355. Springer, 2019.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems - a cryptograhic perspective*, volume 671 of *The Kluwer international series in engineering and computer science*. Springer, 2002.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.

[Mic19] Daniele Micciancio. Lattices: Algorithms and applications, 2019. `http://cseweb.ucsd.edu/classes/fa17/cse206A-a/`.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of sis and lwe with small parameters. In *CRYPTO (1)*, pages 21–39, 2013.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[MR08] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In Daniel J. Bernstein, Johannes Buchmann, and Erik Dahmen, editors, *Chapter in Post-quantum Cryptography*, pages 147–191. Springer, 2008.

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

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, pages 145–166, 2006.

[PRS17] Chris Peikert, Oded Regev, and Noah Stephens-Davidowitz. Pseudorandomness of ring-lwe for any ring and modulus. In *STOC*, pages 461–473. ACM, 2017.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[Sei18] Gregor Seiler. Faster AVX2 optimized NTT multiplication for ring-lwe lattice cryptography. *IACR Cryptology ePrint Archive*, 2018:39, 2018. http://eprint.iacr.org/2018/039.

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

[SS11] Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, pages 27–47, 2011.

[SXY18] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. Tightly-secure key-encapsulation mechanism in the quantum random oracle model. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 520–551. Springer, 2018.

[The18] The PARI Group, Univ. Bordeaux. *PARI/GP version* `2.9.1`, 2018. available from `http://pari.math.u-bordeaux.fr/`.