yptosystem is not secure in dimension 3, since even e large enough to make an exhaustive search impractical, there are efficient algorithms to find good bases in low dimension. In dimension 2, an algorithm for finding a good basis dates back to Gauss. A powerful generalization to arbitrary dimension, known as the LLL algorithm, is covered in Sect. 7.13.

Remark 7.37. We observe that GGH is an example of a probabilistic cryptosystem (see Sect. 3.10), since a single plaintext leads to many different ciphertexts due to the choice of the random perturbation $\boldsymbol{r}$. This leads to a potential danger if Bob sends the same message twice using different random perturbations, or sends different messages using the same random perturbation. One possible solution is to choose the random perturbation $\boldsymbol{r}$ deterministically by applying a hash function (Sect. 8.1) to the plaintext $\boldsymbol{m}$, but this causes other security issues. See Exercises 7.20 and 7.21 for a further discussion.

Remark 7.38. An alternative version of GGH reverses the roles of $\boldsymbol{m}$ and $\boldsymbol{r}$, so the ciphertext has the form $\boldsymbol{e} = \boldsymbol{r}W + \boldsymbol{m}$. Alice finds $\boldsymbol{r}W$ by computing the lattice vector closest to $\boldsymbol{e}$, and then she recovers the plaintext as $\boldsymbol{m} = \boldsymbol{e} - \boldsymbol{r}W$.

7.9 Convolution Polynomial Rings

In this section we describe the special sort of polynomial quotient rings that are used by the NTRU public key cryptosystem, which is the topic of Sects. 7.10 and 7.11. The reader who is unfamiliar with basic ring theory should read Sect. 2.10 before continuing.

Definition. Fix a positive integer N. The *ring of convolution polynomials (of rank N)* is the quotient ring

$$R = \frac{\mathbb{Z}[x]}{(x^N - 1)}.$$

Similarly, the *ring of convolution polynomials (modulo q)* is the quotient ring

$$R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^N - 1)}.$$

Proposition 2.50 tells us that every element of R or R_q has a unique representative of the form

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1}$$

with the coefficients in $\mathbb{Z}$ or $\mathbb{Z}/q\mathbb{Z}$, respectively. We observe that it is easier to do computations in the rings R and R_q than it is in more general polynomial quotient rings, because the polynomial $x^N - 1$ has such a simple form. The

point is that when we mod out by $x^N - 1$, we are simply requiring x^N to equal 1. So any time x^N appears, we replace it by 1. For example, if we have a term x^k, then we write $k = iN + j$ with $0 \le j < N$ and set

$$x^k = x^{iN+j} = (x^N)^i \cdot x^j = 1^i \cdot x^j = x^j.$$

In brief, the exponents on the powers of x may be reduced modulo N.

It is often convenient to identify a polynomial

$$\boldsymbol{a}(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{N-1} x^{N-1} \in R$$

with its vector of coefficients

$$(a_0, a_1, a_2, \ldots, a_{N-1}) \in \mathbb{Z}^N,$$

and similarly with polynomials in R_q. Addition of polynomials corresponds to the usual addition of vectors,

$$\boldsymbol{a}(x) + \boldsymbol{b}(x) \longleftrightarrow (a_0 + b_0, a_1 + b_1, a_2 + b_2, \ldots, a_{N-1} + b_{N-1}).$$

The rule for multiplication in R is a bit more complicated. We write $\star$ for multiplication in R and R_q, to distinguish it from standard multiplication of polynomials.

Proposition 7.39. *The product of two polynomials $\boldsymbol{a}(x), \boldsymbol{b}(x) \in R$ is given by the formula*

$$\boldsymbol{a}(x) \star \boldsymbol{b}(x) = \boldsymbol{c}(x) \quad \text{with} \quad c_k = \sum_{i+j \equiv k \pmod N} a_i b_{k-i}, \tag{7.27}$$

where the sum defining c_k is over all i and j between 0 and $N-1$ satisfying the condition $i + j \equiv k \pmod N$. The product of two polynomials $\boldsymbol{a}(x), \boldsymbol{b}(x) \in R_q$ is given by the same formula, except that the value of c_k is reduced modulo q.

Proof. We first compute the usual polynomial product of $\boldsymbol{a}(x)$ and $\boldsymbol{b}(x)$, after which we use the relation $x^N = 1$ to combine the terms. Thus

$$\begin{aligned}
\boldsymbol{a}(x) \star \boldsymbol{b}(x) &= \left(\sum_{i=0}^{N-1} a_i x^i\right) \star \left(\sum_{j=0}^{N-1} b_j x^j\right) \\
&= \sum_{k=0}^{2N-2} \left(\sum_{i+j=k} a_i b_j\right) x^k \\
&= \sum_{k=0}^{N-1} \left(\sum_{i+j=k} a_i b_j\right) x^k + \sum_{k=N}^{2N-2} \left(\sum_{i+j=k} a_i b_j\right) x^{k-N}
\end{aligned}$$

$$= \sum_{k=0}^{N-1} \left(\sum_{i+j=k} a_i b_j \right) x^k + \sum_{k=0}^{N-2} \left(\sum_{i+j=k+N} a_i b_j \right) x^k$$
$$= \sum_{k=0}^{N-1} \left(\sum_{i+j \equiv k \pmod N} a_i b_j \right) x^k. \qquad \square$$

Example 7.40. We illustrate multiplication in the convolution rings R and R_q with an example. We take $N = 5$ and let $\boldsymbol{a}(x), \boldsymbol{b}(x) \in R$ be the polynomials

$$\boldsymbol{a}(x) = 1 - 2x + 4x^3 - x^4 \qquad \text{and} \qquad \boldsymbol{b}(x) = 3 + 4x - 2x^2 + 5x^3 + 2x^4.$$

Then

$$\begin{aligned} \boldsymbol{a}(x) \star \boldsymbol{b}(x) &= 3 - 2x - 10x^2 + 21x^3 + 5x^4 - 16x^5 + 22x^6 + 3x^7 - 2x^8 \\ &= 3 - 2x - 10x^2 + 21x^3 + 5x^4 - 16 + 22x + 3x^2 - 2x^3 \\ &= -13 + 20x - 7x^2 + 19x^3 + 5x^4 \quad \text{in } R = \mathbb{Z}[x]/(x^5 - 1). \end{aligned}$$

If we work instead in the ring R_{11}, then we reduce the coefficients modulo 11 to obtain

$$\boldsymbol{a}(x) \star \boldsymbol{b}(x) = 9 + 9x + 4x^2 + 8x^3 + 5x^4 \quad \text{in } R_{11} = (\mathbb{Z}/11\mathbb{Z})[x]/(x^5 - 1).$$

Remark 7.41. The *convolution product* of two vectors is given by

$$(a_0, a_1, a_2, \ldots, a_{N-1}) \star (b_0, b_1, b_2, \ldots, b_{N-1}) = (c_0, c_1, c_2, \ldots, c_{N-1}),$$

where the c_k are defined by (7.27). We use $\star$ interchangeably to denote convolution multiplication in the rings R and R_q and the convolution product of vectors.

There is a natural map from R to R_q in which we simply reduce the coefficients of a polynomial modulo q. This reduction modulo q map satisfies

$$\big(\boldsymbol{a}(x) + \boldsymbol{b}(x)\big) \bmod q = \big(\boldsymbol{a}(x) \bmod q\big) + \big(\boldsymbol{b}(x) \bmod q\big), \tag{7.28}$$
$$\big(\boldsymbol{a}(x) \star \boldsymbol{b}(x)\big) \bmod q = \big(\boldsymbol{a}(x) \bmod q\big) \star \big(\boldsymbol{b}(x) \bmod q\big). \tag{7.29}$$

(In mathematical terminology, the map $R \to R_q$ is a ring homomorphism.)

It is often convenient to have a consistent way of going in the other direction. Among the many ways of lifting, we choose the following.

Definition. Let $\boldsymbol{a}(x) \in R_q$. The *center-lift of* $\boldsymbol{a}(x)$ *to* R is the unique polynomial $\boldsymbol{a}'(x) \in R$ satisfying

$$\boldsymbol{a}'(x) \bmod q = \boldsymbol{a}(x)$$

whose coefficients are chosen in the interval

$$-\frac{q}{2} < a'_i \le \frac{q}{2}.$$

For example, if $q = 2$, then the center-lift of $\boldsymbol{a}(x)$ is a binary polynomial.

Remark 7.42. It is important to observe that the lifting map does not satisfy the analogs of (7.28) and (7.29). In other words, the sum or product of the lifts need *not* be equal to the lift of the sum or product.

Example 7.43. Let $N = 5$ and $q = 7$, and consider the polynomial

$$\boldsymbol{a}(x) = 5 + 3x - 6x^2 + 2x^3 + 4x^4 \in R_7.$$

The coefficients of the center-lift of $\boldsymbol{a}(x)$ are chosen from $\{-3, -2, \ldots, 2, 3\}$, so

$$\text{Center-lift of } \boldsymbol{a}(x) = -2 + 3x + x^2 + 2x^3 - 3x^4 \in R.$$

Similarly, the lift of $\boldsymbol{b}(x) = 3 + 5x^2 - 6x^3 + 3x^4$ is $3 - 2x^2 + x^3 + 3x^4$. Notice that

$$(\text{Lift of } \boldsymbol{a}) \star (\text{Lift of } \boldsymbol{b}) = 20x + 10x^2 - 11x^3 - 14x^4$$

and

$$(\text{Lift of } \boldsymbol{a} \star \boldsymbol{b}) = -x + 3x^2 + 3x^3$$

are not equal to one another, although they are congruent modulo 7.

Example 7.44. Very few polynomials in R have multiplicative inverses, but the situation is quite different in R_q. For example, let $N = 5$ and $q = 2$. Then the polynomial $1 + x + x^4$ has an inverse in R_2, since in R_2 we have

$$(1 + x + x^4) \star (1 + x^2 + x^3) = 1 + x + x^2 + 2x^3 + 2x^4 + x^6 + x^7 = 1.$$

(Since $N = 5$, we have $x^6 = x$ and $x^7 = x^2$.) When q is a prime, the extended Euclidean algorithm for polynomials (Proposition 2.46) tells us which polynomials are units and how to compute their inverses in R_q.

Proposition 7.45. *Let q be prime. Then $\boldsymbol{a}(x) \in R_q$ has a multiplicative inverse if and only if*

$$\gcd\big(\boldsymbol{a}(x), x^N - 1\big) = 1 \quad \text{in } (\mathbb{Z}/q\mathbb{Z})[x]. \tag{7.30}$$

If (7.30) is true, then the inverse $\boldsymbol{a}(x)^{-1} \in R_q$ can be computed using the extended Euclidean algorithm (Proposition 2.46) to find polynomials $\boldsymbol{u}(x), \boldsymbol{v}(x) \in (\mathbb{Z}/q\mathbb{Z})[x]$ satisfying

$$\boldsymbol{a}(x)\boldsymbol{u}(x) + (x^N - 1)\boldsymbol{v}(x) = 1.$$

Then $\boldsymbol{a}(x)^{-1} = \boldsymbol{u}(x)$ in R_q.

Proof. Proposition 2.46 says that we can find polynomials $\boldsymbol{u}(x)$ and $\boldsymbol{v}(x)$ in the polynomial ring $(\mathbb{Z}/q\mathbb{Z})[x]$ satisfying

$$\boldsymbol{a}(x)\boldsymbol{u}(x) + (x^N - 1)\boldsymbol{v}(x) = \gcd\big(\boldsymbol{a}(x), x^N - 1\big).$$

If the gcd is equal to 1, then reducing modulo $x^N - 1$ yields $\boldsymbol{a}(x) \star \boldsymbol{u}(x) = 1$ in R_q. Conversely, if $\boldsymbol{a}(x)$ is a unit in R_q, then we can find a polynomial $\boldsymbol{u}(x)$ such that $\boldsymbol{a}(x) \star \boldsymbol{u}(x) = 1$ in R_q. By definition of R_q, this means that

$$\boldsymbol{a}(x)\boldsymbol{u}(x) \equiv 1 \pmod{(x^N - 1)},$$

so by definition of congruences, there is a polynomial $\boldsymbol{v}(x)$ satisfying

$$\boldsymbol{a}(x)\boldsymbol{u}(x) - 1 = (x^N - 1)\boldsymbol{v}(x) \quad \text{in } (\mathbb{Z}/q\mathbb{Z})[x]. \qquad \square$$

Example 7.46. We let $N = 5$ and $q = 2$ and give the full details for computing $(1 + x + x^4)^{-1}$ in R_2. First we use the Euclidean algorithm to compute the greatest common divisor of $1 + x + x^4$ and $1 - x^5$ in $(\mathbb{Z}/2\mathbb{Z})[x]$. (Note that since we are working modulo 2, we have $1 - x^5 = 1 + x^5$.) Thus

$$\begin{aligned} x^5 + 1 &= x \cdot (x^4 + x + 1) + (x^2 + x + 1), \\ x^4 + x + 1 &= (x^2 + x)(x^2 + x + 1) + 1. \end{aligned}$$

So the gcd is equal to 1, and using the usual substitution method yields

$$\begin{aligned} 1 &= (x^4 + x + 1) + (x^2 + x)(x^2 + x + 1) \\ &= (x^4 + x + 1) + (x^5 + 1 + x(x^4 + x + 1)) \\ &= (x^4 + x + 1)(x^3 + x^2 + 1) + (x^5 + 1)(x^2 + x). \end{aligned}$$

Hence

$$(1 + x + x^4)^{-1} = 1 + x^2 + x^3 \quad \text{in } R_2.$$

(See Exercise 1.12 for an efficient computer algorithm and Fig. 1.3 for the "magic box method" to compute $\boldsymbol{a}(x)^{-1}$ in R_q.)

Remark 7.47. The ring R_q makes perfect sense regardless of whether q is prime, and indeed there are situations in which it can be advantageous to take q composite, for example $q = 2^k$. In general, if q is a power of a prime p, then in order to compute the inverse of $\boldsymbol{a}(x)$ in R_q, one first computes the inverse in R_p, then "lifts" this value to an inverse in R_{p^2}, and then lifts to an inverse in R_{p^4}, and so on. (See Exercise 7.27.) Similarly, if $q = q_1 q_2 \cdots q_r$, where each $q_i = p_i^{k_i}$ is a prime power, one first computes inverses in R_{q_i} and then combines the inverses using the Chinese remainder theorem.

7.10 The NTRU Public Key Cryptosystem

Cryptosystems based on the difficulty of integer factorization or the discrete logarithm problem are group-based cryptosystems, because the underlying hard problem involves only one operation. For RSA, Diffie–Hellman, and Elgamal, the group is the group of units modulo m for some modulus m that

may be prime or composite, and the group operation is multiplication modulo m. For ECC, the group is the set of points on an elliptic curve modulo p and the group operation is elliptic curve addition.

Rings are algebraic objects that have two operations, addition and multiplication, which are connected via the distributive law. In this section we describe NTRUEncrypt, the NTRU public key cryptosystem. NTRUEncrypt is most naturally described using convolution polynomial rings, but the underlying hard mathematical problem can also be interpreted as SVP or CVP in a lattice. We discuss the connection with lattices in Sect. 7.11.

7.10.1 NTRUEncrypt

In this section we describe NTRUEncrypt, the NTRU (pronounced *en-trū*) public key cryptosystem. We begin by fixing an integer $N \ge 1$ and two moduli p and q, and we let R, R_p, and R_q be the convolution polynomial rings

$$R = \frac{\mathbb{Z}[x]}{(x^N - 1)}, \qquad R_p = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^N - 1)}, \qquad R_q = \frac{(\mathbb{Z}/q\mathbb{Z})[x]}{(x^N - 1)},$$

described in Sect. 7.9. As usual, we may view a polynomial $\boldsymbol{a}(x) \in R$ as an element of R_p or R_q by reducing its coefficients modulo p or q. In the other direction, we use center-lifts to move elements from R_p or R_q to R. We make various assumptions on the parameters N, p and q, in particular we require that N be prime and that $\gcd(N, q) = \gcd(p, q) = 1$. (The reasons for these assumptions are explained in Exercises 7.32 and 7.37.)

We need one more piece of notation before describing NTRUEncrypt.

Definition. For any positive integers d_1 and d_2, we let

$$\mathcal{T}(d_1, d_2) = \left\{ \boldsymbol{a}(x) \in R : \begin{array}{l} \boldsymbol{a}(x) \text{ has } d_1 \text{ coefficients equal to } 1, \\ \boldsymbol{a}(x) \text{ has } d_2 \text{ coefficients equal to } -1, \\ \boldsymbol{a}(x) \text{ has all other coefficients equal to } 0 \end{array} \right\}.$$

Polynomials in $\mathcal{T}(d_1, d_2)$ are called *ternary* (or *trinary*) *polynomials*. They are analogous to *binary polynomials*, which have only 0's and 1's as coefficients.

We are now ready to describe NTRUEncrypt. Alice (or some trusted authority) chooses public parameters (N, p, q, d) satisfying the guidelines described earlier (or see Table 7.4). Alice's private key consists of two randomly chosen polynomials

$$\boldsymbol{f}(x) \in \mathcal{T}(d+1, d) \quad \text{and} \quad \boldsymbol{g}(x) \in \mathcal{T}(d, d). \tag{7.31}$$

Alice computes the inverses

$$\boldsymbol{F}_q(x) = \boldsymbol{f}(x)^{-1} \text{ in } R_q \quad \text{and} \quad \boldsymbol{F}_p(x) = \boldsymbol{f}(x)^{-1} \text{ in } R_p. \tag{7.32}$$

(If either inverse fails to exist, she discards this $\boldsymbol{f}(x)$ and chooses a new one. We mention that Alice chooses $\boldsymbol{f}(x)$ in $\mathcal{T}(d+1,d)$, rather than in $\mathcal{T}(d,d)$, because elements in $\mathcal{T}(d,d)$ never have inverses in R_q; see Exercise 7.24.)

Alice next computes

$$\boldsymbol{h}(x) = \boldsymbol{F}_q(x) \star \boldsymbol{g}(x) \quad \text{in } R_q. \tag{7.33}$$

The polynomial $\boldsymbol{h}(x)$ is Alice's public key. Her private key, which she'll need to decrypt messages, is the pair $\big(\boldsymbol{f}(x), \boldsymbol{F}_p(x)\big)$. Alternatively, Alice can just store $\boldsymbol{f}(x)$ and recompute $\boldsymbol{F}_p(x)$ when she needs it.

Bob's plaintext is a polynomial $\boldsymbol{m}(x) \in R$ whose coefficients satisfy $-\frac{1}{2}p < m_i \le \frac{1}{2}p$, i.e., the plaintext $\boldsymbol{m}$ is a polynomial in R that is the center-lift of a polynomial in R_p. Bob chooses a random polynomial (a random element) $\boldsymbol{r}(x) \in \mathcal{T}(d,d)$ and computes[7]

$$\boldsymbol{e}(x) \equiv p\boldsymbol{h}(x) \star \boldsymbol{r}(x) + \boldsymbol{m}(x) \pmod q. \tag{7.34}$$

Bob's ciphertext $\boldsymbol{e}(x)$ is in the ring R_q.

On receiving Bob's ciphertext, Alice starts the decryption process by computing

$$\boldsymbol{a}(x) \equiv \boldsymbol{f}(x) \star \boldsymbol{e}(x) \pmod q. \tag{7.35}$$

She then center lifts $\boldsymbol{a}(x)$ to an element of R and does a mod p computation,

$$\boldsymbol{b}(x) \equiv \boldsymbol{F}_p(x) \star \boldsymbol{a}(x) \pmod p. \tag{7.36}$$

Assuming that the parameters have been chosen properly, we now verify that the polynomial $\boldsymbol{b}(x)$ is equal to the plaintext $\boldsymbol{m}(x)$.

NTRUEncrypt, the NTRU public key cryptosystem, is summarized in Table 7.4.

Proposition 7.48. *If the NTRUEncrypt parameters (N,p,q,d) are chosen to satisfy*

$$q > (6d+1)\,p, \tag{7.37}$$

then the polynomial $\boldsymbol{b}(x)$ computed by Alice in (7.36) is equal to Bob's plaintext $\boldsymbol{m}(x)$.

Proof. We first determine more precisely the shape of Alice's preliminary calculation of $\boldsymbol{a}(x)$. Thus

$$\begin{aligned}
\boldsymbol{a}(x) &\equiv \boldsymbol{f}(x) \star \boldsymbol{e}(x) \pmod q && \text{from (7.35)},\\
&\equiv \boldsymbol{f}(x) \star \big(p\boldsymbol{h}(x) \star \boldsymbol{r}(x) + \boldsymbol{m}(x)\big) \pmod q && \text{from (7.34)},\\
&\equiv p\boldsymbol{f}(x) \star \boldsymbol{F}_q(x) \star \boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x) \pmod q && \text{from (7.33)}.\\
&\equiv p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x) \pmod q && \text{from (7.32)}.
\end{aligned}$$

[7]Note that when we write a congruence of polynomials modulo q, we really mean that the computation is being done in R_q.

Public parameter creation	
A trusted party chooses public parameters (N,p,q,d) with N and p prime, $\gcd(p,q) = \gcd(N,q) = 1$, and $q > (6d+1)p$.	
Alice	**Bob**
Key creation	
Choose private $\boldsymbol{f} \in \mathcal{T}(d+1,d)$ that is invertible in R_q and R_p. Choose private $\boldsymbol{g} \in \mathcal{T}(d,d)$. Compute $\boldsymbol{F}_q$, the inverse of $\boldsymbol{f}$ in R_q. Compute $\boldsymbol{F}_p$, the inverse of $\boldsymbol{f}$ in R_p. Publish the public key $\boldsymbol{h} = \boldsymbol{F}_q \star \boldsymbol{g}$.	(mod q)
Encryption	
	Choose plaintext $\boldsymbol{m} \in R_p$. Choose a random $\boldsymbol{r} \in \mathcal{T}(d,d)$. Use Alice's public key $\boldsymbol{h}$ to compute $\boldsymbol{e} \equiv p\boldsymbol{r} \star \boldsymbol{h} + \boldsymbol{m} \pmod q$. Send ciphertext $\boldsymbol{e}$ to Alice.
Decryption	
Compute $\boldsymbol{a} = \boldsymbol{f} \star \boldsymbol{e} \equiv p\boldsymbol{g} \star \boldsymbol{r} + \boldsymbol{f} \star \boldsymbol{m} \pmod q$. Center-lift to $\boldsymbol{a} \in R$ and compute $\boldsymbol{m} \equiv \boldsymbol{F}_p \star \boldsymbol{a} \pmod p$.	

Table 7.4: NTRUEncryt: the NTRU public key cryptosystem

Consider the polynomial

$$p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x), \tag{7.38}$$

computed exactly in R, rather than modulo q. We need to bound its largest possible coefficient. The polynomials $\boldsymbol{g}(x)$ and $\boldsymbol{r}(x)$ are in $\mathcal{T}(d,d)$, so if, in the convolution product $\boldsymbol{g}(x) \star \boldsymbol{r}(x)$, all of their 1's match up and all of their -1's match up, the largest possible coefficient of $\boldsymbol{g}(x) \star \boldsymbol{r}(x)$ is $2d$. Similarly, $\boldsymbol{f}(x) \in \mathcal{T}(d+1,d)$ and the coefficients of $\boldsymbol{m}(x)$ are between $-\frac{1}{2}p$ and $\frac{1}{2}p$, so the largest possible coefficient of $\boldsymbol{f}(x) \star \boldsymbol{m}(x)$ is $(2d+1)\cdot\frac{1}{2}p$. So even if the largest coefficient of $\boldsymbol{g}(x) \star \boldsymbol{r}(x)$ happens to coincide with the largest coefficient of $\boldsymbol{r}(x) \star \boldsymbol{m}(x)$, the largest coefficient of (7.38) has magnitude at most

$$p \cdot 2d + (2d+1) \cdot \frac{1}{2}p = \left(3d + \frac{1}{2}\right) p.$$

Thus our assumption (7.37) ensures that every coefficient of (7.38) has magnitude strictly smaller than $\frac{1}{2}q$. Hence when Alice computes $\boldsymbol{a}(x)$ modulo q

(i.e., in R_q) and then lifts it to R, she recovers the exact value (7.38). In other words,

$$\boldsymbol{a}(x) = p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x) \tag{7.39}$$

exactly in R, and not merely modulo q.

The rest is easy. Alice multiplies $\boldsymbol{a}(x)$ by $\boldsymbol{F}_p(x)$, the inverse of $\boldsymbol{f}(x)$ modulo p, and reduces the result modulo p to obtain

$$\begin{aligned}
\boldsymbol{b}(x) &\equiv \boldsymbol{F}_p(x) \star \boldsymbol{a}(x) \pmod{p} && \text{from (7.36),}\\
&\equiv \boldsymbol{F}_p(x) \star \bigl(p\boldsymbol{g}(x) \star \boldsymbol{r}(x) + \boldsymbol{f}(x) \star \boldsymbol{m}(x)\bigr) \pmod{p} && \text{from (7.39),}\\
&\equiv \boldsymbol{F}_p(x) \star \boldsymbol{f}(x) \star \boldsymbol{m}(x) \pmod{p} && \text{reducing mod } p,\\
&\equiv \boldsymbol{m}(x) \pmod{p}. && \text{from (7.32).}
\end{aligned}$$

Hence $\boldsymbol{b}(x)$ and $\boldsymbol{m}(x)$ are the same modulo p. □

Remark 7.49. The condition $q > (6d+1)p$ in Proposition 7.48 ensures that decryption never fails. However, an examination of the proof shows that decryption is likely to succeed even for considerably smaller values of q, since it is highly unlikely that the positive and negative coefficients of $\boldsymbol{g}(x)$ and $\boldsymbol{r}(x)$ will exactly line up, and similarly for $\boldsymbol{f}(x)$ and $\boldsymbol{m}(x)$. So for additional efficiency and to reduce the size of the public key, it may be advantageous to choose a smaller value of q. It then becomes a delicate problem to estimate the probability of decryption failure. It is important that the probability of decryption failure be very small (e.g., smaller than 2^{-80}), since decryption failures have the potential to reveal private key information to an attacker.

Remark 7.50. Notice that NTRUEncrypt is an example of a probabilistic cryptosystem (Sect. 3.10), since a single plaintext $\boldsymbol{m}(x)$ has many different encryptions $p\boldsymbol{h}(x)\star\boldsymbol{r}(x)+\boldsymbol{m}(x)$ corresponding to different choices of the random element $\boldsymbol{r}(x)$. As is common for such systems, cf. Remark 7.37 for GGH, it is a bad idea for Bob to send the same message twice using different random elements, just as it is inadvisable for Bob to use the same random element to send two different plaintexts; see Exercise 7.34. Various ways of ameliorating this danger for GGH, which also apply *mutatis mutandis* to NTRUEncrypt, are described in Exercises 7.20 and 7.21.

Remark 7.51. The polynomial $\boldsymbol{f}(x) \in \mathcal{T}(d+1,d)$ has small coefficients, but the coefficients of its inverse $\boldsymbol{F}_q(x) \in R_q$ tend to be randomly and uniformly distributed modulo q. (This is not a theorem, but it is an experimentally observed fact.) For example, let $N = 11$ and $q = 73$ and take a random polynomial

$$\boldsymbol{f}(x) = x^{10} + x^8 - x^3 + x^2 - 1 \in \mathcal{T}(3,2).$$

Then $\boldsymbol{f}(x)$ is invertible in R_q, and its inverse

$$\boldsymbol{F}_q(x) = 22x^{10}+33x^9+15x^8+33x^7-10x^6+36x^5-33x^4-30x^3+12x^2-32x+28$$

has random-looking coefficients. Similarly, in practice the coefficients of the public key and the ciphertext,

$$\boldsymbol{h}(x) \equiv \boldsymbol{F}_q(x) \star \boldsymbol{g}(x) \pmod{q} \quad \text{and} \quad \boldsymbol{e}(x) \equiv p\boldsymbol{r}(x) \star \boldsymbol{h}(x) + \boldsymbol{m}(x) \pmod{q},$$

also appear to be randomly distributed modulo q.

Remark 7.52. As noted in Sect. 7.7, a motivation for using lattice-based cryptosystems is their high speed compared to discrete logarithm and factorization-based cryptosystems. How fast is NTRUEncrypt? The most time consuming part of encryption and decryption is the convolution product. In general, a convolution product $\boldsymbol{a}\star\boldsymbol{b}$ requires N^2 multiplications, since each coefficient is essentially the dot product of two vectors. However, the convolution products required by NTRUEncrypt have the form $\boldsymbol{r}\star\boldsymbol{h}$, $\boldsymbol{f}\star\boldsymbol{e}$, and $\boldsymbol{F}_p\star\boldsymbol{a}$, where $\boldsymbol{r}$, $\boldsymbol{f}$, and $\boldsymbol{F}_p$ are ternary polynomials. Thus these convolution products can be computed without any multiplications; they each require approximately $\frac{2}{3}N^2$ additions and subtractions. (If d is smaller than $N/3$, the first two require only $\frac{2}{3}dN$ additions and subtractions.) Thus NTRUEncrypt encryption and decryption take $\mathcal{O}(N^2)$ steps, where each step is extremely fast.

Example 7.53. We present a small numerical example of NTRUEncrypt with public parameters

$$(N,p,q,d) = (7,3,41,2).$$

We have

$$41 = q > (6d+1)p = 39,$$

so Proposition 7.48 ensures that decryption will work. Alice chooses

$$\boldsymbol{f}(x) = x^6-x^4+x^3+x^2-1 \in \mathcal{T}(3,2) \quad \text{and} \quad \boldsymbol{g}(x) = x^6+x^4-x^2-x \in \mathcal{T}(2,2).$$

She computes the inverses

$$\begin{aligned}
\boldsymbol{F}_q(x) &= \boldsymbol{f}(x)^{-1} \bmod q = 8x^6 + 26x^5 + 31x^4 + 21x^3 + 40x^2 + 2x + 37 \in R_q,\\
\boldsymbol{F}_p(x) &= \boldsymbol{f}(x)^{-1} \bmod p = x^6 + 2x^5 + x^3 + x^2 + x + 1 \in R_p.
\end{aligned}$$

She stores $\bigl(\boldsymbol{f}(x), \boldsymbol{F}_p(x)\bigr)$ as her private key and computes and publishes her public key

$$\boldsymbol{h}(x) = \boldsymbol{F}_q(x) \star \boldsymbol{g}(x) = 20x^6 + 40x^5 + 2x^4 + 38x^3 + 8x^2 + 26x + 30 \in R_q.$$

Bob decides to send Alice the message

$$\boldsymbol{m}(x) = -x^5 + x^3 + x^2 - x + 1$$

using the random element

$$\boldsymbol{r}(x) = x^6 - x^5 + x - 1.$$

Bob computes and sends to Alice the ciphertext

$$e(x) \equiv pr(x)\star h(x)+m(x) \equiv 31x^6+19x^5+4x^4+2x^3+40x^2+3x+25 \pmod q.$$

Alice's decryption of Bob's message proceeds smoothly. First she computes

$$f(x)\star e(x) \equiv x^6+10x^5+33x^4+40x^3+40x^2+x+40 \pmod q. \quad (7.40)$$

She then center-lifts (7.40) modulo q to obtain

$$a(x) = x^6+10x^5-8x^4-x^3-x^2+x-1 \in R.$$

Finally, she reduces $a(x)$ modulo p and computes

$$F_p(x)\star a(x) \equiv 2x^5+x^3+x^2+2x+1 \pmod p. \quad (7.41)$$

Center-lifting (7.41) modulo p retrieves Bob's plaintext $m(x) = -x^5+x^3+x^2-x+1$.

7.10.2 Mathematical Problems for NTRUEncrypt

As noted in Remark 7.51, the coefficients of the public key $h(x)$ appear to be random integers modulo q, but there is a hidden relationship

$$f(x)\star h(x) \equiv g(x) \pmod q, \quad (7.42)$$

where $f(x)$ and $g(x)$ have very small coefficients. Thus breaking NTRUEncrypt by finding the private key comes down to solving the following problem:

The NTRU Key Recovery Problem
Given $h(x)$, find ternary polynomials $f(x)$ and $g(x)$ satisfying $f(x)\star h(x) \equiv g(x) \pmod q$.

Remark 7.54. The solution to the NTRU key recovery problem is not unique, because if $\big(f(x),g(x)\big)$ is one solution, then $\big(x^k\star f(x), x^k\star g(x)\big)$ is also a solution for every $0 \le k < N$. The polynomial $x^k\star f(x)$ is called a *rotation of* $f(x)$ because the coefficients have been cyclically rotated k positions. Rotations act as private decryption keys in the sense that decryption with $x^k\star f(x)$ yields the rotated plaintext $x^k\star m(x)$.

More generally, any pair of polynomials $\big(f(x),g(x)\big)$ with sufficiently small coefficients and satisfying (7.42) serves as an NTRU decryption key. For example, if $f(x)$ is the original decryption key and if $\theta(x)$ has tiny coefficients, then $\theta(x)\star f(x)$ may also work as a decryption key.

Remark 7.55. Why would one expect the NTRU key recovery problem to be a hard mathematical problem? A first necessary requirement is that the problem not be practically solvable by a brute-force or collision search. We discuss such searches later in this section. More importantly, in Sect. 7.11.2 we prove that solving the NTRU key recovery problem is (almost certainly) equivalent to solving SVP in a certain class of lattices. This relates the NTRU problem to a well-studied problem, albeit for a special collection of lattices. The use of lattice reduction is currently the best known method to recover an NTRU private key from the public key. Is lattice reduction the best possible method? Just as with integer factorization and the various discrete logarithm problems underlying other cryptosystems, no one knows for certain whether faster algorithms exist. So the only way to judge the difficulty of the NTRU key recovery problem is to note that it has been well studied by the mathematical and cryptographic community. Then a quantitative estimate of the difficulty of solving the problem is obtained by applying the fastest algorithm currently known.

How hard is Eve's task if she tries a brute-force search of all possible private keys? Note that Eve can determine whether she has found the private key $f(x)$ by verifying that $f(x)\star h(x) \pmod q$ is a ternary polynomial. (In all likelihood, the only polynomials with this property are the rotations of $f(x)$, but if Eve happens to find another ternary polynomial with this property, it will serve as a decryption key.)

So we need to compute the size of the set of ternary polynomials. In general, we can specify an element of $\mathcal{T}(d_1,d_2)$ by first choosing d_1 coefficients to be 1 and then choosing d_2 of the remaining $N-d$ coefficients to be -1. Hence

$$\#\mathcal{T}(d_1,d_2) = \binom{N}{d_1}\binom{N-d_1}{d_2} = \frac{N!}{d_1!\,d_2!\,(N-d_1-d_2)!}. \quad (7.43)$$

We remark that this number is maximized if d_1 and d_2 are both approximately $N/3$.

For a brute-force search, Eve must try each polynomial in $\mathcal{T}(d+1,d)$ until she finds a decryption key, but note that all of the rotations of $f(x)$ are decryption keys, so there are N winning choices. Hence it will take Eve approximately $\#\mathcal{T}(d+1,d)/N$ tries to find some rotation of $f(x)$.

Example 7.56. We consider the set of NTRUEncrypt parameters

$$(N,p,q,d) = (251,3,257,83).$$

(This set does not satisfy the $q > (6d+1)p$ requirement, so there may be a rare decryption failure; see Remark 7.49.) Eve expects to check approximately

$$\frac{\mathcal{T}(84,83)}{251} = \frac{1}{251}\binom{251}{84}\binom{167}{83} \approx 2^{381.6}$$

polynomials before finding a decryption key.

Remark 7.57. Not surprisingly, if Eve has a sufficient amount of storage, she can use a collision algorithm to search for the private key. (This was

first observed by Andrew Odlyzko.) We describe the basic idea. Eve searches through pairs of ternary polynomials

$$f_1(x) = \sum_{0\le i<N/2} a_i x^i \qquad \text{and} \qquad f_2(x) = \sum_{N/2\le i<N} a_i x^i$$

having the property that $f_1(x) + f_2(x) \in \mathcal{T}(d+1,d)$. She computes

$$f_1(x) \star h(x) \pmod q \qquad \text{and} \qquad -f_2(x) \star h(x) \pmod q$$

and puts them into bins depending on their coefficients. The bins are set up so that when a polynomial from each list lands in the same bin, the quantity

$$\big(f_1(x) + f_2(x)\big) \star h(x) \pmod q$$

has small coefficients, and hence $f_1(x) + f_2(x)$ is a decryption key. For further details, see [101].

The net effect of the collision algorithm is, as usual, to more or less take the square root of the number of steps required to find a key, so the collision-search security is approximately the square root of (7.43). Returning to Example 7.56, a collision search takes on the order of $\sqrt{2^{381.6}} \approx 2^{190.8}$ steps.

In general, if we maximize the size of $\mathcal{T}(d+1,d)$ by setting $d \approx N/3$, then we can use Stirling's formula (Proposition 7.29) to estimate

$$\#\mathcal{T}(d+1,d) \approx \frac{N!}{((N/3)!)^3} \approx \left(\frac{N}{e}\right)^N \cdot \left(\left(\frac{N}{3e}\right)^{N/3}\right)^{-3} \approx 3^N.$$

So a collision search in this case take $\mathcal{O}(3^{N/2}/\sqrt{N})$ steps.

Remark 7.58. We claimed earlier that $f(x)$ and its rotations are probably the only decryption keys in $\mathcal{T}(d+1,d)$. To see why this is true, we ask for the probability that some random $f(x) \in \mathcal{T}(d+1,d)$ has the property that

$$f(x) \star h(x) \pmod q \quad \text{is a ternary polynomial.} \tag{7.44}$$

Treating the coefficients of (7.44) as independent[8] random variables that are uniformly distributed modulo q, the probability that any particular coefficient is ternary is $3/q$, and hence the probability that every coefficient is ternary is approximately $(3/q)^N$. Hence

$$\begin{pmatrix}\text{Expected number of decryption keys in } \mathcal{T}(d+1,d)\end{pmatrix} \approx \Pr\begin{pmatrix}f(x)\in\mathcal{T}(d+1,d)\\ \text{is a decryption key}\end{pmatrix} \times \#\mathcal{T}(d+1,d)$$
$$= \left(\frac{3}{q}\right)^N \binom{N}{d+1}\binom{N-d-1}{d}.$$

[8]The coefficients of $f(x) \star h(x) \pmod q$ are not entirely independent, but they are sufficiently independent for this to be a good approximation.

Returning to Example 7.56, we see that the expected number of decryption keys in $\mathcal{T}(84,83)$ for $N = 251$ and $q = 257$ is

$$\left(\frac{3}{257}\right)^{251}\binom{251}{84}\binom{167}{83} \approx 2^{-1222.02}. \tag{7.45}$$

Of course, if $h(x)$ is an NTRUEncrypt public key, then there do exist decryption keys, since we built the decryption key $f(x)$ into the construction of $h(x)$. But the probability calculation (7.45) makes it unlikely that there are any additional decryption keys beyond $f(x)$ and its rotations.

7.11 NTRUEncrypt as a Lattice Cryptosystem

In this section we explain how NTRU key recovery can be formulated as a shortest vector problem in a certain special sort of lattice. Exercise 7.36 sketches a similar description of NTRU plaintext recovery as a closest vector problem.

7.11.1 The NTRU Lattice

Let

$$h(x) = h_0 + h_1 x + \cdots + h_{N-1}x^{N-1}$$

be an NTRUEncrypt public key. The *NTRU lattice* L_h^{NTRU} associated to $h(x)$ is the $2N$-dimensional lattice spanned by the rows of the matrix

$$M_h^{\text{NTRU}} = \left(\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{N-1}\\ 0 & 1 & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{N-2}\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1 & h_1 & h_2 & \cdots & h_0\\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0\\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right).$$

Notice that M_h^{NTRU} is composed of four N-by-N blocks:

Upper left block = Identity matrix,
Lower left block = Zero matrix,
Lower right block = q times the identity matrix,
Upper right block = Cyclical permutations of the coefficients of $h(x)$.