QUASIKOM Developer Manual

Johann Großschädl

johann.groszschaedl@gmx.net

Abstract: Security and privacy concerns pose serious obstacles to the further evolution and expansion of the Internet of Things (IoT). Standardized security protocols like Transport Layer Security (TLS) and its datagram variant DTLS can enable secure (i.e. encrypted and authenticated) communication between two devices over an insecure network. However, all currently-used public-key cryptosystems, in particular RSA and elliptic-curve schemes, will become insecure in the dawning era of quantum computing since they can be easily broken with a large-scale quantum computer. In the coming years, the whole public-key infrastructure of the IoT needs to be extended to support post-quantum cryptosystems, which introduces a tremendous research and development effort. The netidee project QUASIKOM contributed to this effort through an efficient implementation of the NTRU cryptosystem for resource-constrained IoT devices and its integration into the TinyDTLS protocol stack. This manual summarizes the work carried out in the course of the QUASIKOM project and consists of three main sections. In the first section, the NTRU public-key cryptosystem and the underlying arithmic operations are explained. Then, in the second section, the low-level functions for the field arithmetic and the hash function SHA-2 are described in more detail and their API is specified. Finally, the third part gives and overview of the DTLS protocol and briefly describes how TinyDTLS can be compiled and executed in a Linux-based development environment.

1 Overview of NTRU

In this section we first set some basic notation and then give a brief description of the key generation, encryption, and decryption operations.

Notation and Parameters.

Before NTRUEncrypt or any other NTRU-based cryptosystem can actually be used, the two involved parties need to agree on a common set of domain parameters. Such domain parameters always include the triple (N, p, q), which defines the underlying algebraic structures. N is the so-called *degree parameter* and typically taken to be a prime between 400 and 800. Currently, $N = 439$ is recommended to achieve a security level of 128 bits, whereas in the past $N = 397$ was deemed sufficient for this level. The degree parameter determines the quotient ring $\mathcal{R} = \mathbb{Z}[x]/(x^N - 1)$ in which all arithmetic operations take place. Every element of $\mathcal{R}$ has a unique representative in the form of a polynomial in x of degree $N - 1$ with coefficients from $\mathbb{Z}$. Since the "modulus polynomial" that defines $\mathcal{R}$ is simply $x^N - 1$, the multiplication of two polynomials from $\mathcal{R}$ corresponds to the cyclic convolution of their coefficients[1], which can be performed very efficiently compared to the multiplication in more general polynomial quotient rings.

[1] Thus, the quotient ring $\mathcal{R} = \mathbb{Z}[x]/(x^N - 1)$ is often referred to as ring of convolution polynomials (or convolution polynomial ring) in the literature.

The parameters p and q are called *small modulus* and *large modulus*, respectively. Common choices for q are a power of two, which is mainly motivated by the need for fast arithmetic modulo q. The most recent parameters sets in version of the specification use $q = 2^{11} = 2048$ across all security levels, but in the past smaller values like $2^7 = 128$ (for 80-bit security) or $2^8 = 256$ (for higher security levels) were recommended. On the other hand, the small modulus p is either a positive integer or a polynomial and must be relatively prime to q in the ring $\mathcal{R}$. When both p and q are integers, this requirement translates into $\gcd(p, q) = 1$. Small values of p (in relation to q) decrease the probability of decryption failures and reduce the storage requirements for the private key. The parameters given in the specification fix the value of p to 3; alternative choices used in the past include $p = 2$ (which requires q to be prime and the polynomial $p = x + 2$.

As we will show below, the multiplications in $\mathcal{R}$ executed during encryption and decryption involve a reduction of all coefficients modulo q so that the final result is an element of the quotient ring $\mathcal{R}_q = (\mathbb{Z}/q\mathbb{Z})[x]/(x^N - 1)$. This ring is subset of the ring $\mathcal{R}$ and any polynomial $a(x) \in \mathcal{R}$ can be simply "reduced" to become an element of $\mathcal{R}_q$ by reducing all its $N - 1$ coefficients modulo q. The equivalent operation in the other direction, used to "lift" an element $a(x)$ from $\mathcal{R}_q$ to $\mathcal{R}$, is a *center-lift* and returns the unique polynomial $a'(x) \in \mathcal{R}$ satisfying $a'(x) \bmod q = a(x)$ whose coefficients a'_i lie in $[-q/2, q/2 - 1]$. Public keys in NTRUEncrypt are elements of $\mathcal{R}_q$, while private keys are "small" polynomials (i.e. polynomials with coefficients of roughly the same magnitude as p). In this paper, we only consider private keys of the form $f(x) = 1 + pF(x)$, where $F(x)$ is a *ternary polynomial* with coefficients in $\{-1, 0, 1\}$. We define $\mathcal{T}$ as the set of all such ternary polynomials having a degree of $N - 1$. Furthermore, we define $\mathcal{T}(d_1, d_2)$ for positive integers d_1 and d_2 as the subset of $\mathcal{T}$ that contains all ternary polynomials with d_1 coefficients equal to 1, d_2 coefficients equal to -1, and the rest equal to 0. Plaintexts (after padding and formatting) are represented as elements of $\mathcal{T}$, whereas ciphertexts are elements of $\mathcal{R}_q$.

The literature mentions a few additional parameters, including d_f, d_g, and d_r, which specify the number of ± 1 coefficients of ternary polynomials that are generated randomly during the key generation and encryption, respectively. To simplify matters, we assume in this paper $d = d_f = d_g = d_r$ and we call d the *weight parameter*. The parameter sets given in the specification fix the weight parameter to $d = \lfloor N/3 \rceil$ in order to maximize the size of the key space.

Key Generation.

The key generation procedure gets a quadruple of the form (N, p, q, d) that meets all guidelines mentioned above as input. In order to obtain a key pair for NTRUEncrypt, the following steps shall be carried out.

1. Generate a (pseudo)random ternary polynomial $F(x) \in \mathcal{T}(d, d)$.
2. Compute $f(x) = 1 + pF(x)$.
3. Compute the inverse $f(x)^{-1} \bmod q$. If $f(x)$ has no inverse mod q then go to Step 1.
4. Generate a (pseudo)random ternary polynomial $g(x) \in \mathcal{T}(d + 1, d)$.
5. Check whether $g(x)$ is invertible modulo q. If it is not then go to Step 4.
6. Compute $h(x) = f(x)^{-1} \star g(x) \bmod q$.

7. Output $f(x)$ as private key and $h(x)$ as public key.

As already stated before, we only consider private keys of the special form $f(x) = 1 + pF(x)$ where $F(x)$ is a ternary polynomial in this paper. Such keys allow for a number of optimizations, which are not possible when $f(x)$ is a small polynomial of more general form. In our case (i.e. $p = 3$), the coefficients f_i of $f(x)$ are in $\{-3, 0, 3\}$ for $1 \leq i \leq N-1$, while $f_0 \in \{-2, 1, 4\}$, and consequently $f(x) \equiv 1 \bmod p$. However, an arbitrary polynomial does not have this property and requires the key generation function to be implemented as described in the first NTRU paper from 1998. In this original NTRU proposal, $f(x)$ is a ring element that is invertible modulo q and modulo p. The key generation involves two inversions of $f(x)$, one modulo q as specified above (Step 3) and one modulo p, whereby the result of the latter is then used as operand of a convolution in the decryption. This extra inversion and convolution has a significant impact on the performance of key generation and decryption, respectively. Fortunately, it is possible to avoid both when $f(x) \equiv 1 \bmod p$ since, in this case, the inverse of $f(x)$ modulo p is simply 1.

The inverse of $f(x)$ modulo q (Step 3) can be found using a variant of the Euclidean algorithm, provided it exists. However, when the domain parameters are properly chosen, the probability that both $f(x)$ and $g(x)$ are invertible modulo q is very high; consequently, the steps 1 to 5 will be executed only once in most runs of the key generation function. Note that the polynomial $g(x)$ is intentionally chosen from $\mathcal{T}(d+1, d)$ and not $\mathcal{T}(d, d)$ since the elements of the latter set never have inverses in $\mathcal{R}_q$. The multiplication in Step 6 represents the computation of a *convolution product*, which consists of a polynomial multiplication modulo $x^N - 1$ (yielding a polynomial of degree $N-1$ as result), followed by a reduction of the coefficients modulo q. We use the star symbol $\star$ for this operation to distinguish it from a conventional polynomial multiplication that returns a product of degree $2N - 2$.

Encryption.

The encryption function gets besides the domain parameters the message M to be encrypted and the public key $h(x)$ of the receiving party as input. Like any other public-key encryption scheme, NTRUEncrypt requires the message to be properly padded and formatted to prevent certain attacks. The standardized NTRUEncrypt scheme defines several supporting functions to transfer a message M into a ternary polynomial $m(x) \in \mathcal{T}$ that can serve as plaintext in the encryption operation outlined below. We will not explain these functions further since this paper focuses on efficient polynomial arithmetic. In order to obtain the ciphertext $c(x)$ of a message M to be encrypted under the public key $h(x)$, the following steps shall be carried out.

1. Represent the message M as a ternary polynomial $m(x) \in \mathcal{T}$.
2. Generate a (pseudo)random blinding polynomial $r(x) \in \mathcal{T}(d, d)$.
3. Compute $c(x) = ph(x) \star r(x) + m(x) \bmod q$.
4. Output $c(x)$ as ciphertext.

Note that, in practice, the polynomial $m(x)$ representing the (padded and formatted) plaintext needs to have a certain minimal number coefficients equal to 1, -1, and 0. The encryption is a randomized process because it involves a so-called *blinding polynomial*

$r(x)$ that is chosen uniformly at random from the set $\mathcal{T}(d,d)$[2]. In terms of arithmetic operations (Step 3), the encryption mainly consists of the computation of the convolution product of $h(x)$ and $r(x)$, followed by a multiplication of all $N-1$ coefficients by $p=3$. Then, the plaintext polynomial $m(x)$ is added to the product and the coefficients are reduced modulo q. The ciphertext $c(x)$ is an element of $\mathcal{R}_q$.

Decryption.

The receiving party needs the private key $f(x)$ corresponding to the public key used for encryption to decrypt the ciphertext $c(x)$. In order to recover the message M, the following steps shall be carried out.

1. Compute $a(x) = c(x) \star f(x) \bmod q = pc(x) \star F(x) + c(x) \bmod q$.
2. Compute $a'(x) = \text{center-lift}(a(x))$.
3. Compute $b(x) = a'(x) \bmod p$.
4. Compute $m(x) = \text{center-lift}(b(x))$.
5. Convert $m(x)$ to message M and output it.

The decryption process is deterministic, but depending on the parameterization, it can happen that the obtained plaintext is not completely correct (we will discuss decryption failures in more detail below). Step 1 contains the main arithmetic operation of decryption, namely the computation of the convolution product of the ciphertext $c(x)$ and the private key $f(x)$. A simple substitution of $c(x)$ by $ph(x) \star r(x) + m(x) \equiv pf(x)^{-1} \star g(x) \star r(x) + m(x) \bmod q$ combined with the fact that $f(x)^{-1} \star f(x) \equiv 1 \bmod q$ leads to the following equation.

$$a(x) \equiv c(x) \star f(x) \equiv pg(x) \star r(x) + m(x) \star f(x) \bmod q \tag{1}$$

In order to explain why (and how) the decryption works, let us first ignore the reduction modulo q and assume Equation (1) is an exact equality in $\mathcal{R}$ instead of a congruence relation. Under this assumption, it is fairly straightforward to see that $a(x) \equiv m(x) \bmod p$ because all coefficients of the term $pg(x) \star r(x)$ are multiples of p, which means they vanish modulo p, and $f(x) \equiv 1 \bmod p$ due to the special form of $f(x)$. Now we have to analyze under which conditions the assumption made before is true so that Equation (1) holds in $\mathcal{R}$ and not just in $\mathcal{R}_q$. For this purpose, it is very important to recall that all four involved polynomials have small coefficients. Concretely, the coefficients of $g(x)$, $m(x)$, and $r(x)$ lie in the interval $[-1,1]$ and those of $f(x)$ in $[-3,4]$. When both $g(x)$ and $r(x)$ have d coefficients equal to 1 and also d coefficients equal to -1, then the largest possible coefficient of the convolution product $g(x) \star r(x)$ is $2d$. The maximum value of the coefficients of $m(x) \star f(x)$ is very similar. For well-chosen parameters, there is a very high probability (or even certainty) that all coefficients of $pg(x) \star r(x) + m(x) \star f(x)$ lie in the interval $[-q/2, q/2-1]$. If this is the case, a reduction modulo q and subsequent center-lift (Step 2) does not change the actual value of any of these coefficients, which means the decryption process will succeed in recovering the plaintext polynomial $m(x)$ by reducing $a'(x)$ modulo p as described above. The final step is a center-lift to obtain $m(x)$ with coefficients in $\{-1,0,1\}$.

[2]The standardized NTRUEncrypt scheme generates $r(x)$ by hashing the message together with an object-ID, a random salt, and a part of the public key $h(x)$.

A decryption failure occurs when a validly encrypted plaintext is not recovered correctly, which is extremely unlikely with well-chosen parameters. Before going into detail, let us recap that, in order for the decryption to succeed, the center-lift of the convolution product $a(x) = c(x) \star f(x) \bmod q$ needs to equal $pg(x) \star r(x) + m(x) \star f(x)$ exactly and not just modulo q. As noted above, this is the case when all coefficients of $pg(x) \star r(x) + m(x) \star f(x)$ lie in the interval $[-q/2, q/2 - 1]$. On the other hand, if one or more coefficients are outside this interval, the decryption fails as these coefficients can only be recovered modulo q and not with their original magnitude. This is because the out-of-range coordinates of $pg(x) \star r(x) + m(x) \star f(x)$ appear "wrapped around" modulo q in the polynomial $a'(x)$ obtained in Step 2, whereas all coordinates of magnitude less than $q/2$ appear unchanged in $a'(x)$. Any wrap error in $a'(x)$ propagates to the polynomial $m(x)$ computed in Step 4, which means the affected coefficients do not match with those of the original plaintext polynomial used for encryption since $\gcd(q, p) = 1$. More concretely, wrapped coefficients are off by a multiple of q modulo p.

Decryption failures threaten the security of NTRUEncrypt because they can leak information about the secret key $f(x)$ to an attacker. However, when using state-of-the-art parameter sets, the probability of a decryption failure is vanishingly low. For example, the parameter set `ees439ep1`, which is assumed to provide a security level of (at least) 128 bits, features a decryption failure rate of 2^{-195}, i.e. one out of 2^{195} ciphertexts does not get decrypted correctly. On the other hand, the parameter set `ees743ep1` (targeting 256 bits of security) comes with a slightly higher probability for a decryption failure to occur, namely 2^{-112}. However, it is possible to entirely prevent decryption failures by choosing the parameter set to satisfy $q > (6d+1)p$. Indeed, decryption never fails with the `ees743ep1` parameters when q is increased from 2048 to 4096. Unfortunately, increasing q reduces the resistance of NTRUEncrypt against lattice reduction attacks and increases the size of ciphertexts and public keys. This explains why the parameter sets aim for a compromise: instead of completely eliminating decryption failures, the parameters were generated to have a non-zero (but very small) failure probability in order to make them more "implementation-friendly." In the case of `ees439ep1`, an attacker would need to carry out 2^{195} encryptions before she can expect to find a message $m(x)$ and blinding polynomial $r(x)$ that triggers a decryption failure, which is much more costly than other forms of attack. The situation is different for the `ees743ep1` parameter set since the 2^{112} encryption operations needed to find a pair $m(x), r(x)$ that gets decrypted incorrectly are clearly below the (nominal) cost of breaking a cryptosystem aimed at 256 bits of security. However, the standardized version of NTRUEncrypt allows the decrypting party to recover $r(x)$ from $c(x)$ and re-compute the ciphertext as in Step 3 of the encryption operation[3], which makes it possible to detect a failure by simply comparing the re-computed ciphertext with the received one.

2 Low-Level Functions and API

The two most performance-critical components of NTRU are the polynomial arithmetic and the hash function SHA-2, which is used to generate the blinding polynomial $r(x)$. These components were implemented in both C and Assembly language for 8-bit AVR

[3]In the standardized NTRUEncrypt scheme, the plaintext polynomial $m(x)$ consists of the actual message and a random padding $b(x)$ called *salt*. The blinding polynomial $r(x)$ is generated by hashing $m(x)$ together with $b(x)$, an object-ID, and a part of the decrypting party's public key $h(x)$. Upon receipt of the ciphertext, the decrypting party recovers the plaintext $m(x)$ and extracts $b(x)$, which allows her to compute $r(x)$ in exactly the same way as the encrypting party did.

QUASIKOM DevelopBT..sUcei5@al

RSA and elliptic-curve schemes,will become insecure in th
computing since they can be easily broken with a large-scale quantum computer. In

The parameters p and q are called *small modulus* and *large modulus*, respectively. Common choices for q

A decryption failure occurs when a validly encrypted plaintext is not recovered correctly, which is extremely unlikely with well-chosen parameters. Before going into detail, let us recap that, in order for the decryption to succeed, the center-lift of the convolution product $a(x) = c(x) \star f(x) \bmod q$ needs to equal $pg(x) \star r(x) + m(x) \star f(x)$ exactly and not just modulo q. As noted above, this is the case when all coe□cients of $pg(x) \star r(x) + m(x) \star f(x)$ lietheterval $[q/2; q/2-1]$ .thehand,or