Cryptographie avancée - TP 8
ZZ3 F5 - Réseaux et Sécurité Informatique
Signature post-quantique : CRYSTALS-Kyber/Dilithium

The Cryptographic Suite for Algebraic Lattices *CRYSTALS* is among the NIST standards for post-quantum cryptographic toolkits. CRYSTALS encompasses two cryptographic primitives: Kyber, an IND-CCA2-secure *key-encapsulation mechanism* (think about it as an encryption system); and Dilithium, a strongly EUF-CMA-secure *digital signature algorithm*. Both algorithms are based on hard problems over module lattices, are designed to withstand attacks by large quantum computers.

https://pq-crystals.org/dilithium/

The Dilithium signature

We begin by examining the characteristics of the *Dilithium signature*. The original publication of the Dilithium signature scheme dates back to 2018 and can be accessed at the following URL: `https://eprint.iacr.org/2017/633.pdf`. Additionally, the cryptographic suite's website gathers a wealth of information regarding this signature scheme.

1. Based on the NIST call for proposal[1] and the website dedicated to the signature scheme (cf logo), find the size of the signature with security assumed equivalent to AES-128. Compare it to the size of a signature based on the RSA signature scheme.

2. Locate the reference implementation on the Dilithium website and execute its different versions. Compare the results with those obtained with the OpenSSL library for RSA signatures.

Baby Kyber

Kyber is a post-quantum public-key encryption system. Its main use case is to establish keys of symmetric-key systems in higher-level protocols like TLS, Signal or OpenPGP. It is a post-quantum system because Kyber is specifically designed to be secure even in the presence of quantum computers.

To give an intuition about Kyber we will implement a down-scaled version of Kyber, that we will call Baby Kyber. Baby Kyber is equivalent to "regular" Kyber, except that the security parameters are smaller, making everything much more readable. Also, "regular" Kyber uses compression for ciphertexts, which we will omit here as it is not relevant for the underlying crypto system.

We define $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n+1)$ as the set of polynomials of degree at most $n-1$ with coefficient in $\mathbb{Z}_q$ (*i.e.* we work modulus X^n+1 for the polynomials and modulus q for its coefficients). For our example we will take $n = 4$ and $q = 17$.

Key Generation: The private key of a Kyber key pair consists of polynomials with small coefficients, i.e. it is a vector of "small" polynomials. In Baby Kyber each private key contains two polynomials. In our example we'll use the private key:

$$s = (-x^3 - x^2 + x, -x^3 - x)$$

[1] https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

Generating this private key is straightforward. We essentially use random small coefficients in $\{-\eta_1, \ldots, \eta_1\}$, for $\eta_1 = 3$.

A Kyber public key consists of two elements. A matrix of random polynomials A and a vector of polynomials t. Generation of the matrix is fairly simple, we just generate random coefficients and take them modulo q. For our example we'll use:

$$A = \begin{pmatrix} 6x^3 + 16x^2 + 16x + 11 & 9x^3 + 4x^2 + 6x + 3 \\ 5x^3 + 3x^2 + 10x + 1 & 6x^3 + x^2 + 9x + 15 \end{pmatrix}$$

To calculate t we need an additional error vector e. This error vector also consists of polynomials with small coefficients, exactly like the private key. In our example we'll use the error vector:

$$e = (x^2, x^2 - x)$$

Now we can calculate t by matrix multiplication and additon:

$$t = As + e$$

This makes t a vector of polynomials, just like s and e. In our example we end up with $t = (16x^3 + 15x^2 + 7, 10x^3 + 12x^2 + 11x + 6)$.

We obtain the secret key s and the public key (A, t).

Encryption: The encryption procedure uses an error and a randomizer polynomial vector e_1 and r. These polynomial vectors are freshly generated for every encryption. Additionally, we need an error polynomial e_2. The polynomials are small, just like the ones in s. We'll use:

$$r = (-x^3 + x^2, x^3 + x^2 - 1) \quad ; \quad e_1 = (x^2 + x, x^2) \quad ; \quad e_2 = -x^3 - x^2$$

Now, to encrypt a message, we have to turn it into a polynomial. We do so by using the message's binary representation. Every bit of the message is used as a coefficient. In our example, we want to encrypt the number 11, $(11)_{10} = (1011)_2$, therefore:

$$m_b = 1x^3 + 0x^2 + 1x^1 + 1 = x^3 + x + 1$$

Before encryption we have to scale this polynomial. We upscale m_b by multiplying it with $\lfloor q/2 \rceil$ ($\lfloor \cdot \rceil$ corresponds to closest integer with ties being rounded up), *i.e.* the integer closest to $q/2$. This is done because the polynomial's coefficients need to be large. In the decryption part we'll see why this is necessary. In our example, $\lfloor q/2 \rceil = 9$. Our final ready-to-be-encrypted message therefore is:

$$m = \lfloor q/2 \rceil \times m_b = 9x^3 + 9x + 9$$

We encrypt m using the public key (A, t). The encryption procedure calculates two values:

$$u = A^T r + e_1 = (11x^3 + 11x^2 + 10x + 3, 4x^3 + 4x^2 + 13x + 11)$$
$$v = t^T r + e_2 + m = 7x^3 + 6x^2 + 8x + 15$$

Kyber ciphtertexts consist of those two values: (u, v). A polynomial vector u and the polynomial v.

Decryption: Given the private key s and a ciphertext (u, v),first, we compute a noisy result $m_n = v - s^T u$.

This result is noisy, because the computation actually does not yield the original message m. By looking at the equations we can see that:

$$m_n = e^T r + e_2 + m + s^T e_1$$

Now it becomes apparent why we needed to scale mm by making its coefficients large. If you recall, all other terms in the equation were chosen to be small. So the coefficients of m_n are either close to$\lfloor q/2 \rceil$ implying that the original binary coefficient of m_b was 1 or close to 0 implying the original binary coefficient was 0.

In our example we have $m_n = 7x^3 + 14x^2 + 7x + 5$. We can recover the original scaled message m by going through the coefficients of m_n and check if they are closer to 9 or 0 (or equivalently q).

The recovered plaintext therefore is: 11.

1. Implement Baby Kyber.

Scaling Baby Kyber The Kyber public key encryption systems works just like Baby Kyber, just with bigger parameters and compression. For example, Kyber1024, its stronger version, works with the parameters:

$n = 256$**:** maximum degree of the used polynomial.
$k = 4$**:** number of polynomials per vector.
$q = 3329$**:** modulus for numbers.
$\eta_1 = 2$ **:** control how big coefficients of "small" polynomials can be.

It also uses some other parameters we may ignore.

2. Scaled up your implementation of Baby Kyber to Kyber1024.
3. Use the modes presented in one of the previous lectures to encrypte message of arbitrary length. Do not use the ECB mode !
4. Based on a theoretical approach, compute the expension factor between a plaintext and a ciphertext. Compare it to what we obtained for RSA.