Google Summer of Code 2024

MERL
Accelerating Engineering Innovation

KyberLite - Baby Kyber Accelerator

A scaled down implementation of NIST Finalist Post Quantum Cryptography algorithm, KYBER.

Mentors:

Dr. Farhan Ahmed Karim

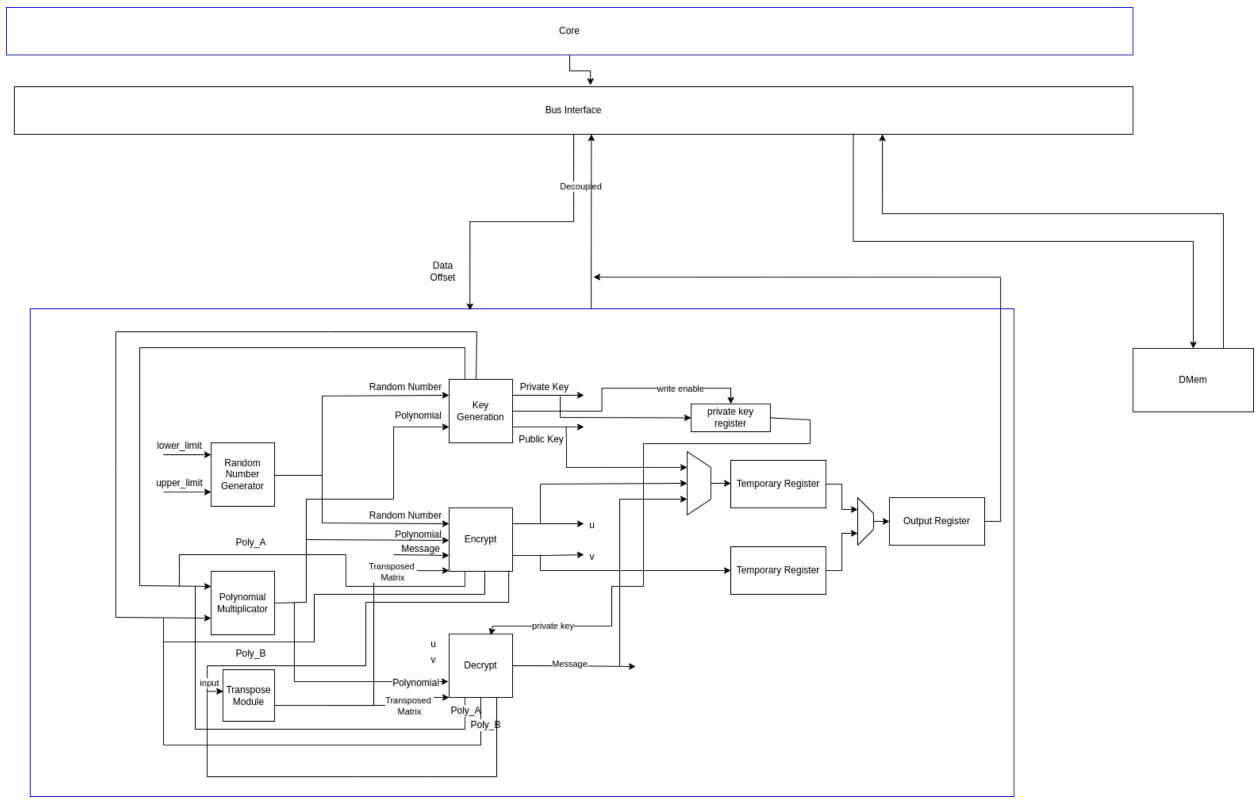
Shahzaib Kashif

Mentee:

Hamna Mohiuddin

KyberLite - Baby Kyber Accelerator

Architecture



Introduction

In the rapidly evolving landscape of cryptography, the advent of quantum computing poses a significant threat to traditional encryption methods. Algorithms like RSA and elliptic curve cryptography, which have long been the cornerstone of secure communications, are vulnerable to the computational power that quantum computers could one day wield.

NIST has finalized the selection of CRYSTALS-Kyber as a post-quantum key encapsulation mechanism (KEM) to secure communications against potential quantum computing threats. Kyber, a lattice-based cryptographic algorithm, is designed for general encryption purposes, including securing public-facing websites, by ensuring the establishment of secure keys for symmetric encryption even in the presence of quantum computers. Alongside Kyber, NIST also selected three other signature schemes—CRYSTALS-Dilithium, FALCON, and SPHINCS+—to standardize these algorithms.

Baby Kyber is a simplified version of Kyber, created to make the core concepts easier to understand. By using smaller security parameters, modulus of $q = 17$ and a polynomial modulus of $f = x^4 + 1$, and straightforward arithmetic, Baby Kyber retains the essential features of CRYSTALS-Kyber algorithm while being more accessible for learning and experimentation.

Key Generation

Key generation in the Baby Kyber cryptosystem involves producing a secure public and secret key pair, where the secret key consists of small polynomials and the public key includes a matrix of random polynomials and a vector derived from matrix multiplication.

Secret Key Generation

The secret key, denoted as s, is composed of two polynomials with small coefficients. These coefficients, which are randomly generated within a specific range, are directly assigned to the secret key output array from the input secret_key array.

Public Key Generation

The public key has two primary components:
A matrix of random polynomials **A**, generated with random coefficients taken modulo a prime number **q**.
A vector of polynomials **t**, obtained through matrix multiplication of **A** with the secret key **s** and the addition of an error vector **e**.

$$t = A * s + e$$

The matrix **A** is passed into the module, and each polynomial in **A** is multiplied with the corresponding polynomials in the secret key **s**. This multiplication is handled by the **PolynomialMatrixMultiplication** module, producing intermediate polynomial results stored in **poly_out0**, **poly_out1**, **poly_out2**, and **poly_out3**.

The results are summed to create intermediate values, which are then added to the error vector **e** to generate the final vector **t**. The final values are reduced modulo **q** to ensure they remain within the specified range, forming the second part of the public key.

Encryption

Encryption in KyberLite relies on generating a ciphertext from a given message using the public key, random polynomial vectors, and error polynomials. The process involves multiple steps, including polynomial matrix multiplication, modular reduction, and error vector addition. Below is an overview of the encryption procedure and its implementation.

Message Polynomial

The message is first converted into a polynomial based on its binary representation. Each bit of the message is treated as a coefficient in the polynomial. For example, the binary representation of the number 11 is 1011, leading to the polynomial $\mathbf{m_b}$.

$$mb = x^3 + x + 1$$

This message polynomial $\boldsymbol{m_b}$ is then scaled by multiplying it with $\lfloor q/2 \rceil$, where $\boldsymbol{q}$ is a predefined modulus. This scaling is necessary to ensure that the polynomial coefficients are sufficiently large, a requirement for proper decryption. For instance, with

$$q = 17$$
$$\lfloor q/2 \rceil = 9$$

The scaled polynomial $m = 9x^3 + 9x + 9$.

Multiplication

The encryption procedure involves several polynomial matrix multiplications. The public key comprises a matrix **A** and a vector **t**, which are multiplied by the random polynomial vector
r to compute the intermediate values **u** and **v**. The polynomials **r, e1** and **e2** are randomly generated, small polynomials used to add noise to the ciphertext.

Polynomial matrix multiplication is handled using the PolynomialMatrixMultiplication submodule. This module performs the multiplication of the transposed public key matrix with the random polynomial vector **r**, producing intermediate polynomial vectors **poly_out0** through **poly_out5**.

Modular Reduction and Error Vector Addition

The results from the polynomial matrix multiplications are then reduced modulo **q.** This step ensures that the polynomial coefficients stay within a manageable range, crucial for both encryption and subsequent decryption. Error vectors **e1** and **e2** are then added to the results. The addition of these vectors introduces randomness, enhancing the security of the encryption.

The final ciphertext consists of the vectors **u** and **v**, where **u** is derived from the sum of the polynomial multiplication results and **e1**, and **v** incorporates the scaled message polynomial
m with **e2**.

$$u = A^T * r + e1$$

$$v = t^T * r + e2 + m$$

Decryption

The decryption process in KyberLite involves reconstructing the original binary message from the given ciphertext using the secret key. The key insight behind this decryption mechanism is to calculate a noisy polynomial **m$_n$** and then use specific criteria to determine the original binary message bits.

The Decrypt module takes as input the secret key and ciphertext, both structured as polynomial arrays. The goal is to compute the noisy polynomial mn, reduce it modulo **q** (which is set to 17 in this implementation), and then recover the original binary message **m$_b$** by analyzing the coefficients of **m$_n$**.
The steps involve,

Multiplication

The ciphertext and secret key are input into two instances of a PolynomialMatrixMultiplication module, which computes the dot product between the ciphertext's polynomials and the secret key's polynomials.
The outputs from these multiplications (**poly_out0** and **poly_out1**) represent the products of the **secret key** with different parts of the **ciphertext**.

Calculation of Noisy Polynomial m_n

The computed polynomial products are subtracted from the corresponding ciphertext value to yield the noisy result mn. This step is crucial as it combines all components to form the polynomial that approximates the original scaled message.

$$mn = v - s^T * u$$

Since the computation may result in negative values, an adjustment is made to ensure all coefficients of mn are non-negative and within the range [0, q].

Rounding Coefficients

The coefficients are analyzed and rounded based on their proximity to either
$\lfloor q/2 \rceil$ (which is 9) or 0. This step effectively determines whether the corresponding binary bit in mb should be 1 or 0.

Binary Message Recovery

The rounded coefficients are then converted into the binary message $\mathbf{m_b}$, where a coefficient of 9 corresponds to a bit value of 1, and any other value corresponds to 0.
Finally, the binary message $\mathbf{m_b}$ is converted into its decimal equivalent, representing the plaintext message.

Polynomial Multiplication

The PolynomialMatrixMultiplication module is designed to perform the core operation of multiplying two polynomials, which is essential in the Kyber encryption and decryption process. Here's a breakdown of the implementation:

Initialization of Temporary Result

The **temp_result** array is initialized to zero for each coefficient. This array stores intermediate results during the polynomial multiplication process.

First Loop (Multiplication and Addition)

The first nested loop iterates through the coefficients of the two input polynomials (polynomial1 and polynomial2). For each pair of coefficients, it calculates the product and adds it to the corresponding position in the temp_result array.
This loop handles the normal multiplication where coefficients are multiplied and added to their respective positions in the resulting polynomial.

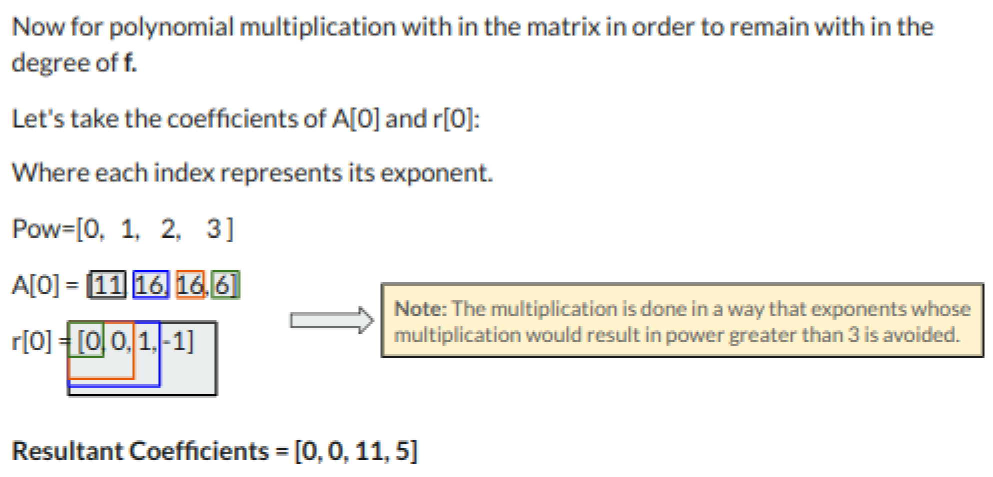
Second Loop (Subtraction for High-Order Terms)

The second nested loop is responsible for handling the modular arithmetic inherent in the polynomial multiplication process. Specifically, it subtracts the high-order terms (those that exceed the polynomial degree of 3) and appropriately wraps them around to ensure that the result remains within the polynomial's degree.
This operation ensures the result adheres to the constraints of a polynomial of degree 3, consistent with the Baby Kyber specification.

Modulo Operation

After computing the intermediate results, a modulo operation is performed on each coefficient in temp_result to ensure that all values are within the range of the modulus $q = 17$.

Explanation



Now for polynomial multiplication with in the matrix in order to remain with in the degree of **f.**

Let's take the coefficients of A[0] and r[0]:

Where each index represents its exponent.

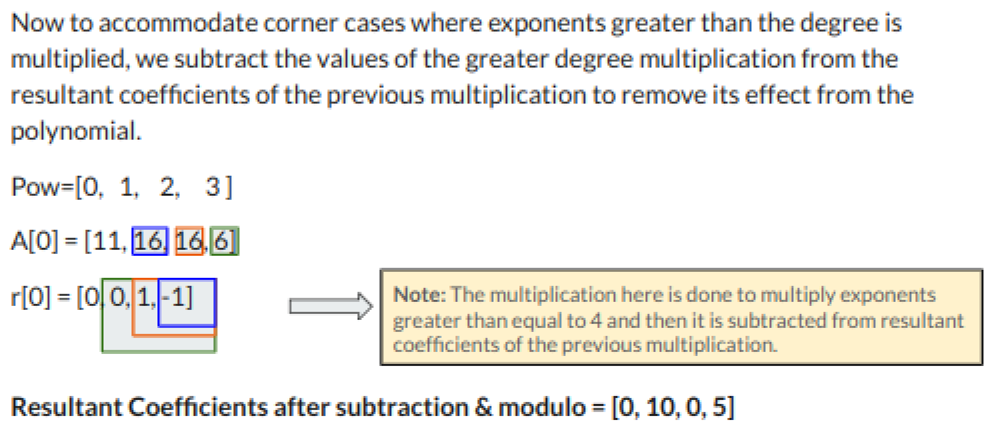
Pow=[0, 1, 2, 3]

A[0] = [11, 16, 16, 6]

r[0] = [0, 0, 1, -1]

Note: The multiplication is done in a way that exponents whose multiplication would result in power greater than 3 is avoided.

Resultant Coefficients = [0, 0, 11, 5]



Now to accommodate corner cases where exponents greater than the degree is multiplied, we subtract the values of the greater degree multiplication from the resultant coefficients of the previous multiplication to remove its effect from the polynomial.

Pow=[0, 1, 2, 3]

A[0] = [11, 16, 16, 6]

r[0] = [0, 0, 1, -1]

Note: The multiplication here is done to multiply exponents greater than equal to 4 and then it is subtracted from resultant coefficients of the previous multiplication.

Resultant Coefficients after subtraction & modulo = [0, 10, 0, 5]

Conclusion

In conclusion, Baby Kyber offers a simplified yet effective introduction to the world of post-quantum cryptography. By demystifying the complex operations behind the Kyber algorithm, it provides a valuable learning tool for those looking to understand and experiment with the cryptographic techniques.

Repository: KyberLite - Baby Kyber Accelerator

[Ppt] Baby Kyber by Hamna: KyberLite

References

[1] How Does Kyber Work?

[2] https://github.com/vishwahaha/Baby-Kyber

[3] https://asecuritysite.com/blog/Baby-Kyber-Part-2--Encryption-and-Decryption-505654d06efa.htm

[4] https://www.linkedin.com/pulse/from-baby-teenage-kyber-marjan-sterjev/

[5] https://github.com/QUB-ARM-STM32/Baby-Kyber