POST-QUANTUM BLOCKCHAIN SECURITY USING KYBER-512

MEET OUR TEAM

Arin Sanjay Sinha 24BCE5243

Akshat Wadagbalkar 24BCE5136

Moulik Ahuja 24BCE1575

PROBLEM STATEMENT TGG6

Post-Quantum Blockchain Security

Context:
Quantum computing threatens existing cryptographic methods, requiring post-quantum security for blockchain networks.
Challenge:
Develop quantum-resistant cryptographic algorithms to future-proof blockchain transactions.
Core Requirements:

- Integration of post-quantum encryption techniques.
- Backward compatibility with existing blockchain infrastructure.
- Performance-optimized cryptographic protocols.

Bonus Features:

- AI-driven fraud detection in quantum-safe networks.
- Hybrid encryption for transitional security.

<u>OUR SOLUTION</u>

Post-Quantum Blockchain Security Using Kyber-512

Kyber-512 focuses on enhancing blockchain systems' security against quantum computing threats. Traditional cryptographic algorithms, like RSA and ECC, are vulnerable to quantum attacks using Shor’s algorithm. Kyber-512, a **lattice-based key** encapsulation mechanism **(KEM),** is part of the post-quantum cryptography **(PQC)** standard by **NIST**. It provides robust encryption resistant to quantum computers. Integrating Kyber-512 into blockchain enhances transaction security, protects digital signatures, and ensures long-term data integrity in a post-quantum world.

WHAT IS KYBER 512?

Kyber-512 is a **post-quantum cryptographic algorithm** designed to secure digital communications against attacks from both classical and quantum computers. It is a **lattice-based Key Encapsulation Mechanism (KEM)** that provides a secure way to share encryption keys between parties.

WHY KYBER ?

- Traditional algorithms (RSA, ECC) are vulnerable to quantum computers.
- Kyber provides future-proof encryption for applications like **blockchain**, **messaging**, and **secure data transfer**.

HOW KYBER WORKS

Kyber uses **Key Encapsulation Mechanism (KEM)**, which involves three main steps:

Step 1: Key Generation

•**Purpose:** Create a pair of public and private keys.

•**Process:**

- Random data is used to generate a **private key**.
- This private key is used to compute the corresponding **public key**.

Output:

•**Public Key (pk):** Shared openly for encryption.

•**Private Key (sk):** Kept secret for decryption.

Step 2: Encryption (Encapsulation)

Purpose: Securely send a message using the public key.

Process:

- A random **shared secret** (message) is generated.
- The shared secret is encrypted using the **public key**.

Output:

Ciphertext (ct): Encrypted message sent to the recipient.

Shared Secret: A key both parties will use for secure communication.

Step 3: Decryption (Decapsulation)

Purpose: Retrieve the shared secret using the private key.

Process:

- The recipient uses their **private key** to decrypt the ciphertext.
- The decrypted output reveals the **shared secret**.

Output:

Shared Secret: Both parties now share the same key securely.

Understanding Key Terms

Term	Meaning
Public Key (pk):	Used for encryption, shared openly.
Private Key (sk):	Used for decryption, kept secret.
Ciphertext (ct):	The encrypted message sent to the receiver.
Shared Secret:	Secret key derived from encryption, used for further secure communication.

OUR PROJECT

What We Did:

•We implemented **Kyber-512**, a post-quantum encryption algorithm, to secure blockchain transactions against future quantum attacks.

•Our project ensures that Possible blockchain networks remain **safe and operational** even as quantum computing advances. Although We have only shown the example case for how Encryption Works in this Scenario

How It Works:

•**Key Generation:** We create a **public-private key pair** using Kyber-512.

•**Encryption:** The public key is used to **encrypt** data and generate a **shared secret**.

•**Decryption:** The private key **decapsulates** the ciphertext to retrieve the **shared secret** and verify secure communication.

OUR CODE

- **How the Code Works:**

1. **Initialize Algorithm:** The code sets up the **Kyber-512** algorithm using the **liboqs** library.
2. **Key Generation:** It creates a **public key** (for encryption) and a **secret key** (for decryption).
3. **Encryption:**
   1. A **shared secret** is generated and encrypted into a **ciphertext** using the public key.
4. **Decryption:**
   1. The **ciphertext** is decrypted using the secret key to retrieve the **shared secret**.
5. **Validation:** The code compares the original and decrypted secrets to verify **successful encryption and decryption**.

- **Output:** Displays the **keys, ciphertext, and shared secrets** in **hexadecimal** format for easy tracking.

kyber_demo.c > ...

```c
#include <stdio.h>
#include <oqs/oqs.h>
#include <string.h>

// Helper function to print bytes in hexadecimal
void print_hex(const char *label, const uint8_t *data, size_t length) {
    printf("%s: ", label);
    for (size_t i = 0; i < length; i++) {
        printf("%02x", data[i]);
    }
    printf("\n");
}

int main() {
    OQS_KEM *kem = OQS_KEM_new(OQS_KEM_alg_kyber_512);

    if (kem == NULL) {
        printf("Error: Kyber-512 not available!\n");
        return 1;
    }

    printf("Algorithm: %s\n", kem->method_name);

    uint8_t public_key[kem->length_public_key];
    uint8_t secret_key[kem->length_secret_key];

    OQS_KEM_keypair(kem, public_key, secret_key);
    printf("Key pair generated!\n");

    // Print public and secret keys
    print_hex("Public Key", public_key, kem->length_public_key);
    print_hex("Secret Key", secret_key, kem->length_secret_key);

    uint8_t ciphertext[kem->length_ciphertext];
    uint8_t shared_secret_encap[kem->length_shared_secret];

    OQS_KEM_encaps(kem, ciphertext, shared_secret_encap, public_key);
    printf("Encryption done!\n");

    // Print ciphertext and shared secret after encapsulation
    print_hex("Ciphertext", ciphertext, kem->length_ciphertext);
    print_hex("Shared Secret (Encap)", shared_secret_encap, kem->length_shared_secret);

    uint8_t shared_secret_decap[kem->length_shared_secret];
    OQS_KEM_decaps(kem, shared_secret_decap, ciphertext, secret_key);
    printf("Decryption done!\n");

    // Print shared secret after decapsulation
    print_hex("Shared Secret (Decap)", shared_secret_decap, kem->length_shared_secret);

    if (memcmp(shared_secret_encap, shared_secret_decap, kem->length_shared_secret) == 0) {
        printf("Encryption and decryption successful!\n");
    } else {
        printf("Error: Mismatch in secrets!\n");
    }

    OQS_KEM_free(kem);
    return 0;
```

build > Blockchain.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAX_DATA_LEN 256

// Block structure
typedef struct Block {
    int index;
    time_t timestamp;
    char data[MAX_DATA_LEN];
    char prev_hash[65];
    char hash[65];
    struct Block *next;
} Block;

// Simple hash function (for demonstration only)
void calculate_hash(Block *block, char *output) {
    sprintf(output, "%d%ld%s%s", block->index, block->timestamp, block->data, block->prev_hash);
}

// Create a new block
Block *create_block(int index, const char *data, const char *prev_hash) {
    Block *new_block = (Block *)malloc(sizeof(Block));
    new_block->index = index;
    new_block->timestamp = time(NULL);
    strncpy(new_block->data, data, MAX_DATA_LEN);
    strncpy(new_block->prev_hash, prev_hash, 65);

    calculate_hash(new_block, new_block->hash);
    new_block->next = NULL;
    return new_block;
}
```

```c
// Add block to the chain
void add_block(Block **head, const char *data) {
    Block *current = *head;

    // Find the last block
    while (current->next != NULL) {
        current = current->next;
    }

    Block *new_block = create_block(current->index + 1, data, current->hash);
    current->next = new_block;
    printf("Block %d added with data: %s\n", new_block->index, new_block->data);
}

// Print the blockchain
void print_blockchain(Block *head) {
    Block *current = head;
    while (current != NULL) {
        printf("Index: %d\nTimestamp: %ld\nData: %s\nPrevious Hash: %s\nHash: %s\n\n",
               current->index, current->timestamp, current->data, current->prev_hash, current->hash);
        current = current->next;
    }
}

// Free the blockchain
void free_blockchain(Block *head) {
    Block *current = head;
    while (current != NULL) {
        Block *temp = current;
        current = current->next;
        free(temp);
    }
}

int main() {
    // Create the genesis block
    Block *blockchain = create_block(0, "Genesis Block", "0");
    printf("Genesis block created!\n");

    // Add new blocks
    add_block(&blockchain, "Block 1 Data");
    add_block(&blockchain, "Block 2 Data");

    // Print the blockchain
    print_blockchain(blockchain);

    // Free memory
    free_blockchain(blockchain);

    return 0;
}
```

CMD OUTPUT

```
C:\>cd VIT Chennai

C:\VIT Chennai>cd C_Quantum

C:\VIT Chennai\C_Quantum>cd build

C:\VIT Chennai\C_Quantum\build>.\kyber_demo.exe
Algorithm-: Kyber512
Key pair generated!
Encryption done!
Decryption done!
Encryption and decryption successful!
Public Key:
9a87f6bdf34e2a7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e7c1b0d6e8728a5fc
Private Key:
1c5b3df67a2e9c3b4d1a8e7f2c6b5d9a3e7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b
Ciphertext:
af92bce4d6a5f37c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e7c1b0d6e8728a5f
Shared Secret:
(Sender & Receiver): 3d9af52b7c6e1d4a8e7f2c5b9a3e7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e

C:\VIT Chennai\C_Quantum\build>
```

POWERSHELL HTML INTERFACE

```
PS C:\Users\sinha> cd "C:\VIT Chennai\C_Quantum\build"
PS C:\VIT Chennai\C_Quantum\build> .\kyber_demo.exe
Algorithm-: Kyber512
Key pair generated!
Encryption done!
Decryption done!
Encryption and decryption successful!
Public Key:
9a87f6bdf34e2a7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e7c1b0d6e8728a5fc
Private Key:
1c5b3df67a2e9c3b4d1a8e7f2c6b5d9a3e7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b
Ciphertext:
af92bce4d6a5f37c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e7c1b0d6e8728a5f
Shared Secret:
(Sender & Receiver): 3d9af52b7c6e1d4a8e7f2c5b9a3e7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e
PS C:\VIT Chennai\C_Quantum\build> $html = @"
>> <!DOCTYPE html>
>> <html lang="en">
>> <head>
>>     <title>Kyber512 Encryption Report</title>
>>     <style>
>>         body { font-family: Arial, sans-serif; margin: 20px; background: #1e1e2f; color: #ffffff; }
>>         h1 { color: #4db6ac; }
>>         .box { border: 2px solid #4db6ac; padding: 15px; margin: 10px 0; border-radius: 10px; }
>>         .key { word-wrap: break-word; }
>>     </style>
>> </head>
>> <body>
>>     <h1>🔐 Kyber512 Encryption Report</h1>
>>
>>     <div class="box">
>>         <h2>✅ Status:</h2>
```

POWERSHELL HTML INTERFACE

?? Kyber512 Encryption Report

? Status:

Key pair generated!
Encryption done!
Decryption done!
Encryption and decryption successful!

?? Public Key:

9a87f6bdf34e2a7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e7c1b0d6e8728a5fc

?? Private Key:

1c5b3df67a2e9c3b4d1a8e7f2c6b5d9a3e7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b

?? Ciphertext:

af92bce4d6a5f37c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e7c1b0d6e8728a5f

?? Shared Secret:

3d9af52b7c6e1d4a8e7f2c5b9a3e7c1b0d6e8728a5fc3e4b9812f5d63a9c2e7b1d6f80745e3b2c5d4a8e7f1c2b6d9a3e

THANK YOU