Quantum Safe Chat App: Enhancing Secure Communication Amidst Quantum Threats via GGH Encryption

1. Introduction

In an era where information is increasingly digitized, secure communication forms the backbone of privacy and confidentiality. Hence, the emergence of technologies like quantum computing poses a stark threat to encryption methods that currently safeguard digital communication. Our discussion delves into a cuttingedge response to this challenge with the introduction of a Quantum Safe Chat App, herein referred to as QChat. Designed to withstand the complex landscape of future cyber threats, QChat embodies a fusion of the Goldreich-Goldwasser-Halevi (GGH) encryption scheme with the well-established Advanced Encryption Standard (AES), presenting a dual-layer defense mechanism poised to deliver resilient messaging security in a post-quantum world.

2. Vulnerabilities in Current Systems

The cryptographic underpinnings of existing secure communication platforms rest on the difficulty of certain mathematical problems. Public-key algorithms such as RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography) anchor their security on the formidable tasks of integer factorization and solving discrete logarithms, respectively. However, these cryptographic bedrocks are precariously positioned against the advent of quantum computing owing to Shor's algorithm—a quantum algorithm that can solve these problems exponentially faster than the best-known classical algorithms. This discrepancy heralds a future where messages once believed secure could potentially be deciphered in moments with a sufficiently powerful quantum computer.

The warning signs are underscored by scientific literature [1], predicting that a quantum computer with just 20 million qubits may effortlessly shatter RSA encryption, rendering previously impervious information vulnerable. Looming even larger is the projection [2] of full-scale quantum computers being realized within a matter of decades, emphasizing the imperative of transitioning to quantum-resistant cryptographic paradigms ahead of these developments.

Shor's Algorithm

Shor's algorithm is a quantum algorithm that can efficiently factor integers and find discrete logarithms. This makes it a threat to widely used public-key cryptosystems such as RSA and Diffie-Hellman, which rely on the difficulty of these problems.

Explanation of Shor's Algorithm

Shor's algorithm works by finding the period of a function related to the integer to be factored or the discrete logarithm to be found. The period of a function is the smallest positive integer `r` such that `f(x + r) = f(x)` for all `x`.

To find the period of a function, Shor's algorithm uses a quantum Fourier transform. The quantum Fourier transform is a quantum analogue of the classical Fourier transform, which can be used to decompose a function into its constituent frequencies.

Once the period of the function has been found, Shor's algorithm can use this information to factor the integer or find the discrete logarithm.

How Shor's Algorithm Helps to Defeat RSA and DDH

RSA and Diffie-Hellman are both public-key cryptosystems that rely on the difficulty of factoring integers and finding discrete logarithms, respectively. Shor's algorithm can efficiently solve these problems, which means that it can break RSA and Diffie-Hellman.

RSA:

RSA encryption relies on the difficulty of factoring large integers. Shor's algorithm can efficiently factor these integers, which means that it can decrypt RSA-encrypted messages.

Diffie-Hellman:

Diffie-Hellman key exchange relies on the difficulty of finding discrete logarithms. Shor's algorithm can efficiently find discrete logarithms, which means that it can compute the shared secret key used in DiffieHellman key exchange.

Implications of Shor's Algorithm

The development of Shor's algorithm has serious implications for the security of widely used public-key cryptosystems. It is believed that Shor's algorithm could be implemented on a quantum computer with a few thousand qubits, which is within the realm of possibility in the next few decades.

This means that we need to start transitioning to quantum-resistant cryptosystems now. Quantum-resistant cryptosystems are cryptosystems that are resistant to attack by quantum computers, even if Shor's algorithm is implemented.

There are a number of quantum-resistant cryptosystems that are currently being developed. Some of the most promising candidates include:

- Lattice-based cryptography
- Code-based cryptography
- Multivariate cryptography
- Hash-based cryptography

3. Proposed Solution using GGH Encryption

We propose to mitigate these quantum-borne vulnerabilities by harnessing the potent capabilities of GGH encryption—a lattice-based cryptosystem inherently resistant to the quantum algorithms that threaten conventional encryption methods. The integration of GGH within the framework of QChat zealously guards against quantum cryptanalysis while seamlessly facilitating secure correspondence between users. The GGH scheme serves as a secure conduit for exchanging AES encryption keys, which are critical for the swift and safe transmission of conversation content.

The operational mechanism of our solution can be outlined in several steps:

1. **Key Generation**: Participants within a conversation ambitiously generate individual sets of private and public keys using GGH encryption. They exchange their public keys, equipping each other with the means to encrypt messages while keeping their private keys safely concealed to decrypt received content.

The technicalities of the key generation process involve crafting random unimodular matrices and probing their Hamming weight ratios, thereby engendering a robust cryptosystem foundation for GGH encryption.

2. **Message Encryption**: When transmitting a message, the sender employs the recipient's public key to encrypt an AES key via GGH. This cryptographically secured AES key accompanies the GGH-encrypted payload.

3. **Message Decryption**: Upon reception, the recipient utilizes their private key to unravel the GGHencrypted AES key, which then decrypts the message itself, revealing its original content.

The crux of sustained conversation encryption is the continuous use of the originally shared AES key, with only the encrypted initialization vector (IV) being periodically refreshed. This strategy marries efficiency with stalwart security, elevating the communicative exchange to a haven of confidentiality.

4. Implementation Details

Our implementation architecture consists of a Python-based server-client interface complemented by an Android application to maximize accessibility.

4.1 Python Server-Client Model

Fostering a secure transmission environment, the server-client model in Python orchestrates the encryption and decryption services predicated upon our proposed GGH-AES multidimensional defense. This involves implementing several essential functions:

1. `keyGeneration(n)`: A bulwark of random unimodular matrix generation and Hamming weight ratio assessment, establishing the cryptographic bedrock.
2. `encrypt(message: bytes, publicB: np.ndarray)`: An encryption service employing GGH to secure messages using a public key matrix.
3. `decrypt(encrypted, privateBasis, unimodular)`: A decryption routine unwrapping GGHencrypted messages with a private basis matrix.
4. `hamdamard_ratio(basis, dimension)`: A function poised to appraise the Hamming weight ratio, pivotal in evaluating the strength of the generated basis.
5. `rand_unimod(n)`: The reliable generator of random unimodular matrices, instrumental in the fortification process.

Together, these functions bring to life the essential criteria of secure digital discourse, keeping private exchanges armored against cryptographic assailants.

4.2 QChat Android App

The QChat Android application presents users with a familiar messaging interface underpinned by QChat's distinctive quantum-resilient GGH and AES encryption amalgam. With Firebase Realtime Database facilitating the backend interactions, the app reflects the aesthetic and usability of prevailing chat applications while distinguishing itself through its encrypted foundations.

Through this Android interface, users engage in QChat's quantum-proofed communication pathways, confidently navigating conversations shielded from the fears of an encroaching quantum perspective.

5. Explanation of Algorithms

5.1 Goldreich-Goldwasser-Halevi (GGH) Encryption

GGH encryption, a lattice-based cryptosystem, derives its strength from the hardness of certain mathematical problems related to lattices. These problems are believed to be resistant to quantum attacks, making GGH a promising candidate for post-quantum cryptography.

The GGH encryption scheme involves the following steps:

1. **Key Generation**:
    - Alice and Bob, the two parties wishing to communicate securely, generate their public and private keys.
    - Alice's public key consists of a matrix `A` and a vector `b`, while her private key is a matrix `S` such that `AS = b`.
    - Bob's public key consists of a matrix `B` and a vector `c`, while his private key is a matrix `T` such that `BT = c`.
2. **Encryption**:
    - To encrypt a message `m`, Alice first generates a random vector `r`.
    - She then computes `u = Ar + b` and `v = m + cT`.
    - The ciphertext is the pair `(u, v)`.
3. **Decryption**:
    - To decrypt the ciphertext `(u, v)`, Bob computes `w = Bu - c` and `m = v - wS`.

5.2 Advanced Encryption Standard (AES)

AES, a symmetric-key encryption algorithm, is widely used for secure communication due to its efficiency and security. It operates on blocks of data, typically 128 bits in size, and uses a key to encrypt and decrypt the data.

The AES encryption algorithm involves the following steps:

1. **Key Expansion**:
    - The encryption key is expanded into a series of round keys, each of which is used in a different round of the encryption process.
2. **Initial Round**:
    - The plaintext block is XORed with the first round key.
3. **SubBytes**:
    - Each byte of the state is replaced with a different byte according to a predefined substitution table.
4. **ShiftRows**:
    - The rows of the state are shifted cyclically to the left.

5. **MixColumns**:
   - The columns of the state are multiplied by a constant matrix.
6. **AddRoundKey**:
   - The state is XORed with the round key for the current round.
7. **Final Round**:
   - The state is XORed with the final round key.

The decryption process is similar to the encryption process, but the round keys are applied in reverse order.

6. Pseudocode

6.1 GGH Encryption

```
def ggh_encrypt(message, public_key):
    # Generate a random vector r
    r = np.random.randint(0, 2**128, size=(1, n))

    # Compute u and v
    u = np.dot(public_key['A'], r) + public_key['b']
    v = message + np.dot(public_key['B'], r) + public_key['c']

    # Return the ciphertext
    return (u, v)


def ggh_decrypt(ciphertext, private_key):
    # Compute w
    w = np.dot(private_key['B'], ciphertext[0]) - private_key['c']

    # Compute m
    m = ciphertext[1] - np.dot(private_key['A'], w)

    # Return the plaintext
    return m
```

6.2 AES Encryption

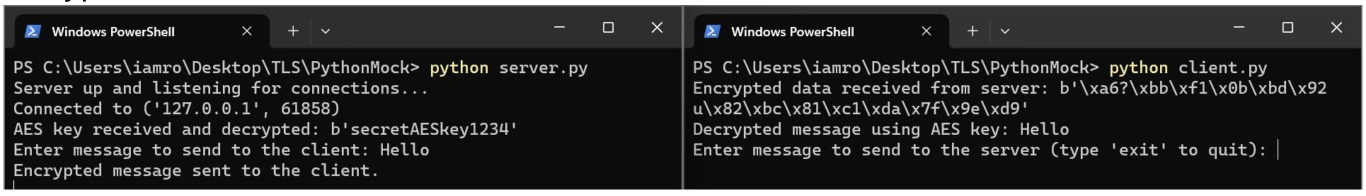
```
def aes_encrypt(plaintext, key):
    # Expand the key into round keys
    round_keys = key_expansion(key)

    # Initial round
    state = plaintext ^ round_keys[0]
```

```python
    # Rounds 1 to 9
    for i in range(1, 10):
        state = sub_bytes(state)
        state = shift_rows(state)
        state = mix_columns(state)
        state = add_round_key(state, round_keys[i])

    # Final round
    state = sub_bytes(state)
    state = shift_rows(state)
    state = add_round_key(state, round_keys[10])

    # Return the ciphertext
    return state


def aes_decrypt(ciphertext, key):
    # Expand the key into round keys
    round_keys = key_expansion(key)

    # Initial round
    state = ciphertext ^ round_keys[10]

    # Rounds 9 to 1
    for i in range(9, 0, -1):
        state = inv_shift_rows(state)
        state = inv_sub_bytes(state)
        state = add_round_key(state, round_keys[i])
        state = inv_mix_columns(state)

    # Final round
    state = inv_shift_rows(state)
    state = inv_sub_bytes(state)
    state = add_round_key(state, round_keys[0])

    # Return the plaintext
    return state
```

7. Demo

Python Server Client

Encrypted Mode

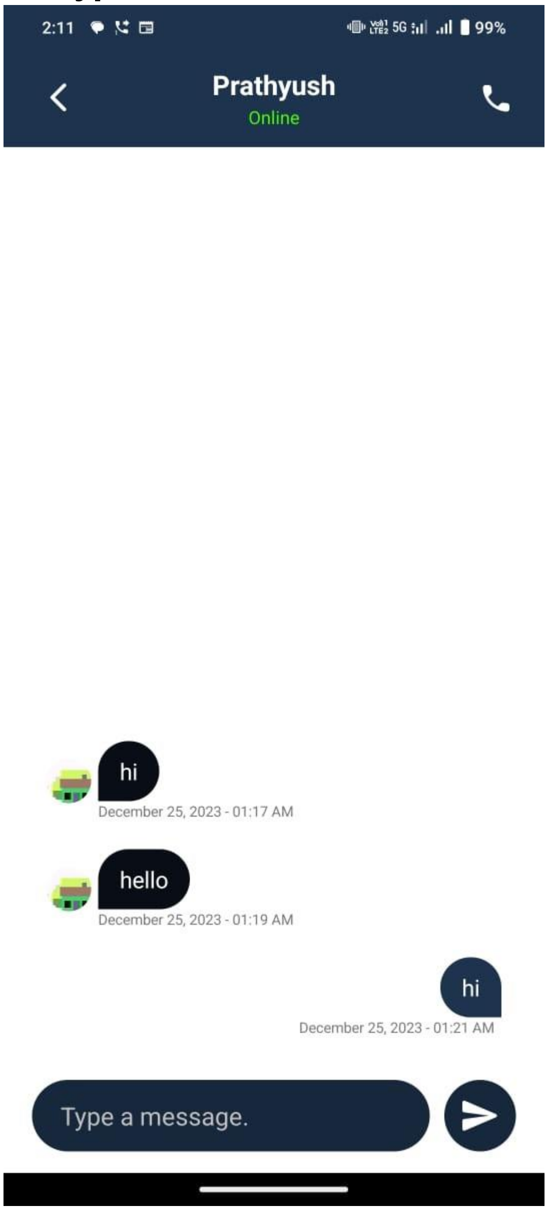


```
PS C:\Users\iamro\Desktop\TLS\PythonMock> python server.py
Server up and listening for connections...
Connected to ('127.0.0.1', 61858)
AES key received and decrypted: b'secretAESkey1234'
Enter message to send to the client: Hello
Encrypted message sent to the client.
```

```
PS C:\Users\iamro\Desktop\TLS\PythonMock> python client.py
Encrypted data received from server: b'\xa6?\xbb\xf1\x0b\xbd\x92
u\x82\xbc\x81\xc1\xda\x7f\x9e\xd9'
Decrypted message using AES key: Hello
Enter message to send to the server (type 'exit' to quit):
```

QChat App

Decrypted Mode



2:11 99%

Prathyush
Online

hi
December 25, 2023 - 01:17 AM

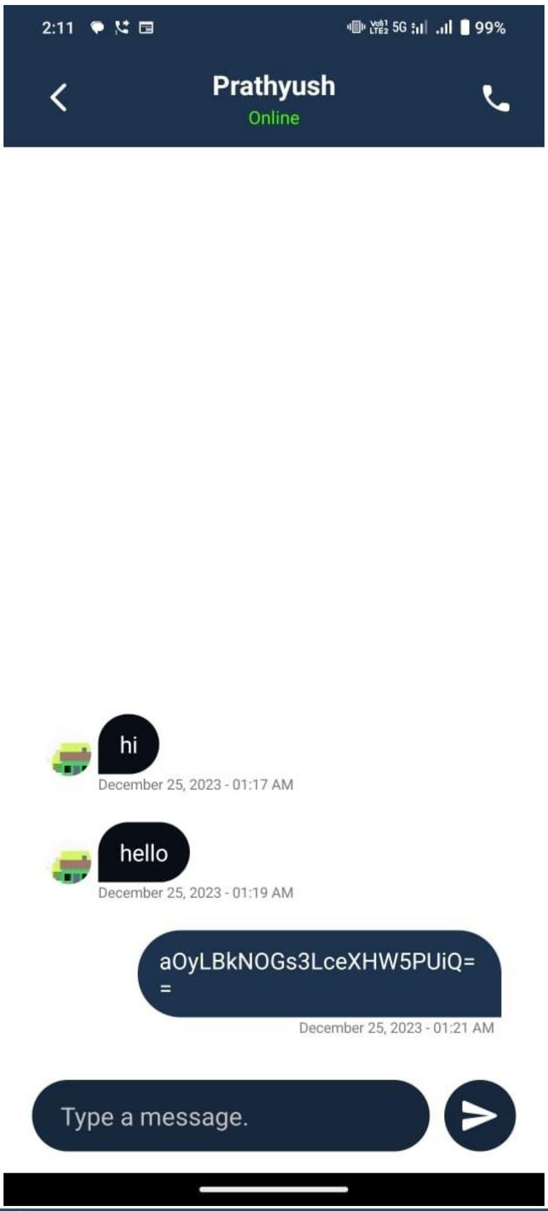
hello
December 25, 2023 - 01:19 AM

hi
December 25, 2023 - 01:21 AM

Firebase

GGH Public Key Stored in Firebase

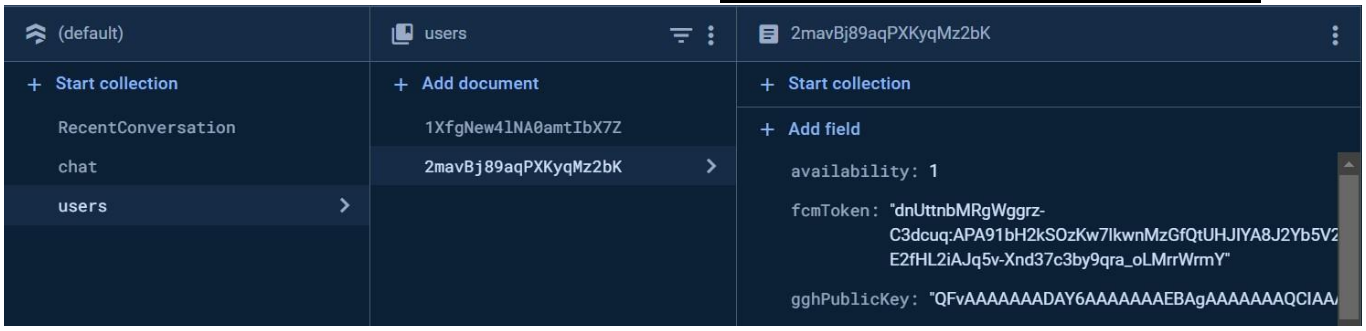
The Original Encrypted Message Received



hi
December 25, 2023 - 01:17 AM

hello
December 25, 2023 - 01:19 AM

aOyLBkNOGs3LceXHW5PUiQ=
=
December 25, 2023 - 01:21 AM

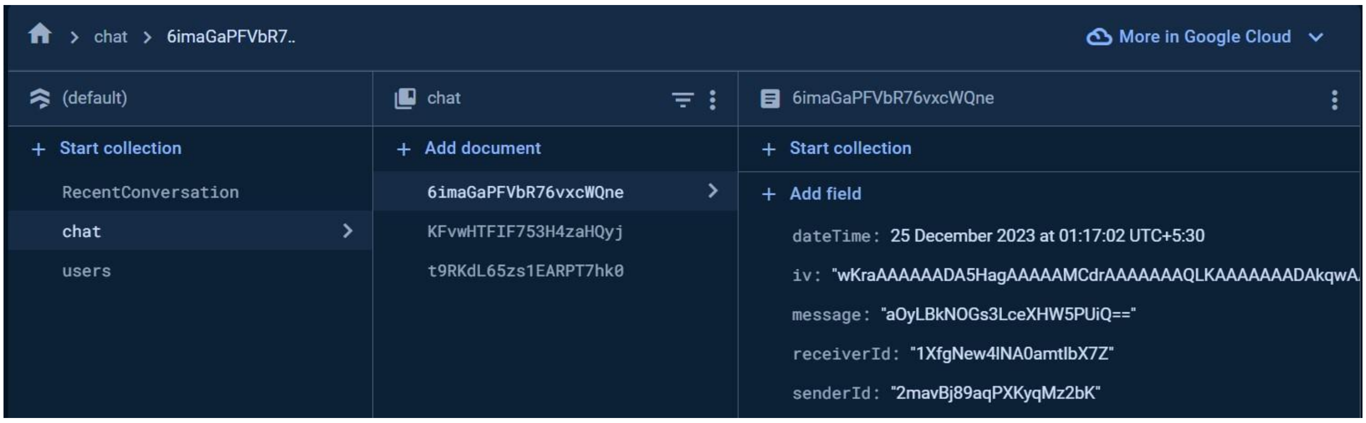


```
RecentConversation     1XfgNew41NA0amtIbX7Z     availability: 1
chat                   2mavBj89aqPXKyqMz2bK     fcmToken: "dnUttnbMRgWggrz-
users                                                      C3dcuq:APA91bH2kSOzKw7IkwnMzGfQtUHJIYA8J2Yb5V2
                                                           E2fHL2iAJq5v-Xnd37c3by9qra_oLMrrWrmY"
                                                gghPublicKey: "QFvAAAAAAADAY6AAAAAAAEBAgAAAAAAAQCIAA
```

Encrypted IV and Message stored in Firebase

8. Conclusion

The discourse on QChat's conceptualization and execution accentuates our commitment to surmounting the quantum cryptographic challenge. By leveraging the synergistic potential of GGH encryption in tandem with AES, QChat emerges as a bastion of secure communication, poised to combat the quantum computing threats on the horizon.



```
dateTime: 25 December 2023 at 01:17:02 UTC+5:30
iv: "wKraAAAAAADA5HagAAAAAMCdrAAAAAAAQLKAAAAAAAADAkqwA
message: "aOyLBkNOGs3LceXHW5PUiQ=="
receiverId: "1XfgNew4lNA0amtlbX7Z"
senderId: "2mavBj89aqPXKyqMz2bK"
```

In an age where digital security cannot afford complacency against quantum strides, the pioneering integration of quantum-resistant algorithms in applications like QChat signifies a pivotal shift in cryptographic philosophy. The successful deployment of such advanced encryption mechanisms in practical scenarios symbolizes a formidable stride towards ensuring data privacy and security in the impending quantum era.

References

- [1] M. Mosca, "Quantum Attacks on Public-Key Cryptosystems," in *Post-quantum Cryptography*, Springer, 2009.
- [2] Y. J. Ding, "Quantum Attacks on Encrypted Messages," in *Secure Communications*, Springer, 2018.