



POST-QUANTUM CRYPTOGRAPHY USING KYBER AND DILITHIUM

DEMO FOR SECURE MESSAGING IN THE QUANTUM ERA

Presenter: Arjun Balu
Affiliation: Reliance Jio

- Emerging Threat of Quantum Computers:

Shor's algorithm can break classical RSA/ECC.

- Future-Proof Security:

Protect data against adversaries with quantum capabilities.

- NIST PQC Standardization:

Kyber (KEM) and Dilithium (Signature) are leading candidates/standards.

WHY POST-QUANTUM CRYPTOGRAPHY?

INTRODUCTION TO THE PROJECT



What is Post-Quantum Cryptography (PQC)?

- Cryptographic algorithms resistant to attacks from quantum computers.
- Standardization led by NIST (Kyber, Dilithium, etc.).

Project Goal:

- Demonstrate a simple chat application that uses post-quantum primitives for:
- Key Encapsulation (Kyber)
- Digital Signatures (Dilithium)

CRYPTOGRAPHIC COMPONENTS

Kyber (Key Encapsulation Mechanism)

- Purpose: Securely exchange a shared secret key between two parties.
- Key Steps:
 1. Key Generation: Generate public/secret key pair.
 2. Encapsulation: Encrypt a random secret using the peer's public key to produce ciphertext.
 3. Decapsulation: Decrypt the ciphertext using the secret key to recover the shared secret.

Dilithium (Digital Signature)

- Purpose: Verify the authenticity and integrity of a message.
- Key Steps:
 1. Key Generation: Generate public/secret key pair.
 2. Signing: Sign the message using the secret key.
 3. Verification: Verify the signature using the public key.

APP INTERFACE

Post-Quantum Chat	Post-Quantum Chat
Your Kyber Public Key:	Your Kyber Public Key:
E1000985754F4725B3907172800209E5974669FC7E7ACC5718A77367B35A6776303B074986F10645D4409FD9A0AF7466FB7390BB6ACD0CA	6C327FB7AC366E7A1E678B3F1154150C15B77541CCB32A08AFFBAC3802231D3354EE352044462030D85044147AB1B43B45C2215D49BC1A0B
Your Dilithium Public Key:	Your Dilithium Public Key:
4080570AEDFB0DFAE84726F7025101F670375D0B597A470A03FB1A81AAD838BDEF83A7A97AEA828881FDA67EF9E735A6FDEA80D7A4AB29C	7A7DFB65F2CDAD3EC5244C5FA139436CBD7E091D1CCBCE7F8D811913854EF5EB896A8708020AE1C0C392316D674F3CC63C239621ED4BBC
Peer's Kyber Public Key:	Peer's Kyber Public Key:
6C327FB7AC366E7A1E678B3F1154150C15B77541CCB32A08AFFBAC3802231D3354EE352044462030D85044147AB1B43B45C2215D49BC1A0B	E1000985754F4725B3907172800209E5974669FC7E7ACC5718A77367B35A6776303B074986F10645D4409FD9A0AF7466FB7390BB6ACD0CA
Peer's Dilithium Public Key:	Peer's Dilithium Public Key:
7A7DFB65F2CDAD3EC5244C5FA139436CBD7E091D1CCBCE7F8D811913854EF5EB896A8708020AE1C0C392316D674F3CC63C239621ED4BBC	4080570AEDFB0DFAE84726F7025101F670375D0B597A470A03FB1A81AAD838BDEF83A7A97AEA828881FDA67EF9E735A6FDEA80D7A4AB29C
Message:	Message:
hehe	
Ciphertext:	Ciphertext:
76BAE2AA98147DCB46F0302D9FB0B228F686C80BC70E7E99E3B263966F12A41DD45E8716BE9F2FEAED2A6C0DD8AA6C02B98BB4B84A079	76BAE2AA98147DCB46F0302D9FB0B228F686C80BC70E7E99E3B263966F12A41DD45E8716BE9F2FEAED2A6C0DD8AA6C02B98BB4B84A079
Signature:	Signature:
762631C48017223032F19AD82B5BFD306EE824FA7862B59B6FB741E4B7DF2EF1922BC7060910C2FA5EC47953A75ECE74C855CBA03039329C	762631C48017223032F19AD82B5BFD306EE824FA7862B59B6FB741E4B7DF2EF1922BC7060910C2FA5EC47953A75ECE74C855CBA03039329C
Received Message:	Received Message:
	hehe
Generate Keys	Generate Keys
Encrypt & Sign	Encrypt & Sign
Decrypt & Verify	Decrypt & Verify
Message encrypted and signed! Send ciphertext and signature.	Message received and verified successfully!

HIGH-LEVEL ARCHITECTURE

1. Key Generation (Local):

- Generate Kyber public/secret keys.
- Generate Dilithium public/secret keys.

2. Exchange Public Keys:

- Send your Kyber & Dilithium public keys to the peer.
- Receive the peer's Kyber & Dilithium public keys.

3. Sending a Message:

- Encapsulate a shared secret using the peer's Kyber public key.
- Encrypt your plaintext message with the shared secret.
- Sign the plaintext message with your Dilithium secret key.
- Send ciphertext + signature to the peer.

4. Receiving a Message:

- Decapsulate the shared secret with your Kyber secret key.
- Decrypt the ciphertext using the shared secret.
- Verify the signature using the peer's Dilithium public key.

CODE WALKTHROUGH (MAIN STEPS)

■ generate_keys()

1. Kyber:

- `OQS_KEM *kem = OQS_KEM_new(OQS_KEM_alg_kyber_1024);`
- `OQS_KEM_keypair(...)` generates Kyber public/secret keys.

2. Dilithium:

- `OQS_SIG *sig = OQS_SIG_new(OQS_SIG_alg_dilithium_2);`
- `OQS_SIG_keypair(...)` generates Dilithium public/secret keys.
- GUI Updates:
- Display generated public keys (hex-encoded) in the GTK entries.

■ send_message()

1. Read Message from the message_entry.

2. Encapsulate:

- `OQS_KEM_encaps(...)` with the peer's Kyber public key → produces `kyber_ciphertext` & `kyber_shared_secret`.

3. Encrypt:

- Use a simple XOR with the `kyber_shared_secret` to encrypt the message.

4. Sign:

- `OQS_SIG_sign(...)` with your Dilithium secret key to produce `dilithium_signature`.

5. Combine & Display:

- Construct a single “full ciphertext” (Kyber ciphertext + message length + encrypted message).
- Convert everything to hex and display in `ciphertext_entry` and `signature_entry`.

■ receive_message()

1. Parse the combined ciphertext:

- Extract Kyber ciphertext, message length, and encrypted message.

2. Decapsulate:

- OQS_KEM_decaps(...) with your Kyber secret key → recovers the shared secret.

3. Decrypt:

- XOR with the recovered shared secret to get the plaintext.

4. Verify:

- OQS_SIG_verify(...) with the peer's Dilithium public key to check the signature.

5. Display:

- If verification succeeds, show the plaintext in received_message_entry.

OVERVIEW

1. USES GTK 4 FOR THE INTERFACE.
2. WIDGETS:
 - TEXT ENTRIES FOR KEYS, MESSAGE, CIPHERTEXT, SIGNATURE.
 - BUTTONS: GENERATE KEYS, ENCRYPT & SIGN, DECRYPT & VERIFY.
3. WORKFLOW:
 - THE USER CLICKS EACH BUTTON IN SEQUENCE TO GENERATE KEYS, SEND, OR RECEIVE

GUI WITH GTK



Demo Flow

1. Generate Keys → Observe local keys in the UI.
2. Copy & Paste local keys to the peer, and peer's keys back into the local UI.
3. Write a Message → Click Encrypt & Sign.
4. Send ciphertext + signature to the peer.
5. Peer clicks Decrypt & Verify → sees the original message if everything is correct.