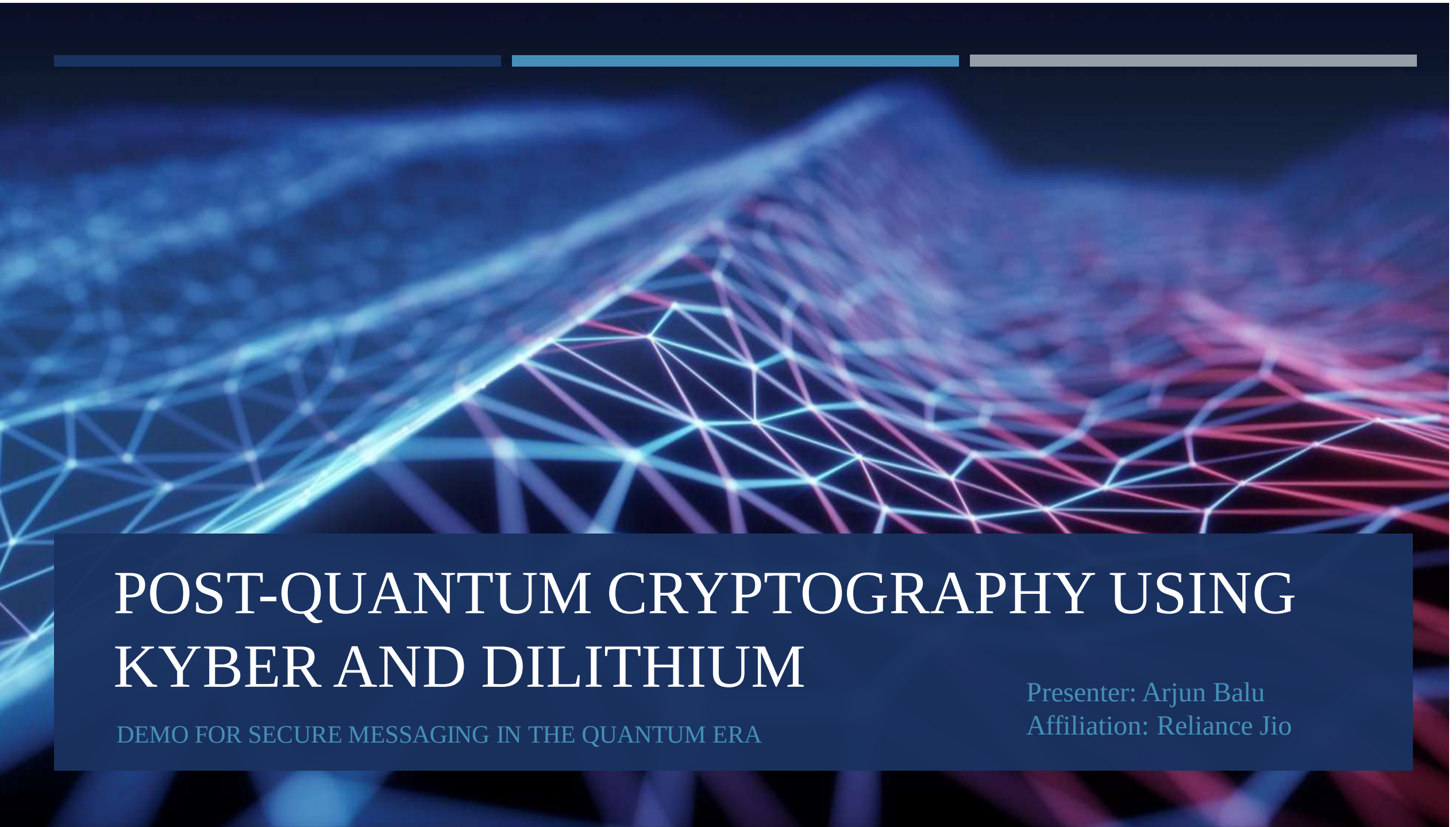


POST-QUANTUM CRYPTOGRAPHY USING KYBER AND DILITHIUM

DEMO FOR SECURE MESSAGING IN THE QUANTUM ERA

Presenter: Arjun Balu
Affiliation: Reliance Jio

WHY POST-QUANTUM CRYPTOGRAPHY?

- 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.

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:

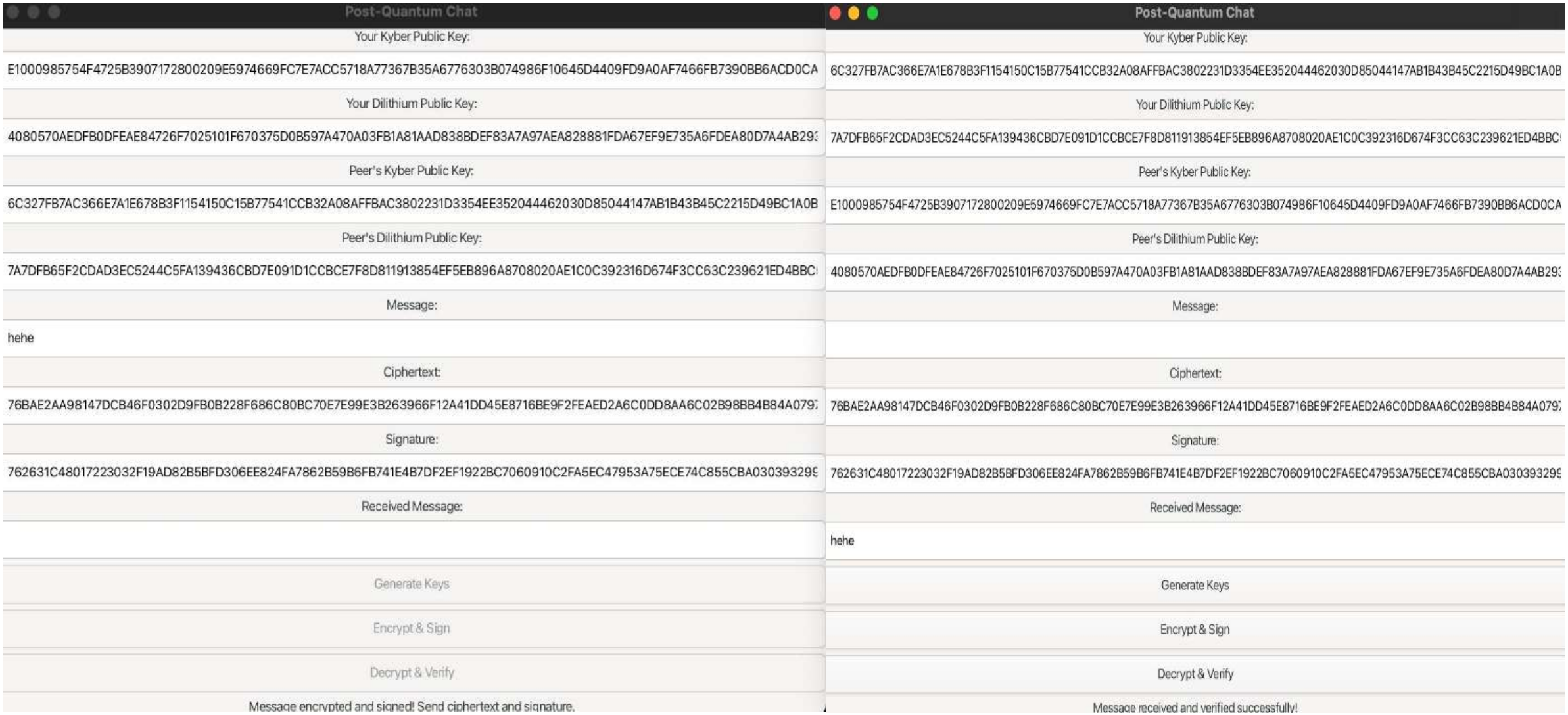
1. Key Generation: Generate public/secret key pair.
2. Encapsulation: Encrypt a random secret using the peer's public key to produce ciphertext.
3. Decapsulation: Decrypt the ciphertext using the secret key to recover the shared secret.

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



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.