Hybrid Cryptography: Diffie-Hellman X25519 and Quantum-Resistant Kyber Key Exchange.

Author: Christian Ghantous
Course/Unit: Advanced Algorithms (SIT320)
University: Deakin
Submission Date: 28th September 2024

Introduction

The Internet of Things (IoT) is rapidly expanding, with a projected revenue of $934 billion by 2033 (Statista, 2024). This growth increases vulnerabilities to cyberattacks, particularly in sectors like automotive, healthcare and transportation.

While the X25519 Diffie-Hellman key exchange is widely used to secure communications, it remains vulnerable to future quantum-based attacks. To counter these threats, post-quantum cryptography, such as the Kyber key encapsulation mechanism (KEM), is essential.

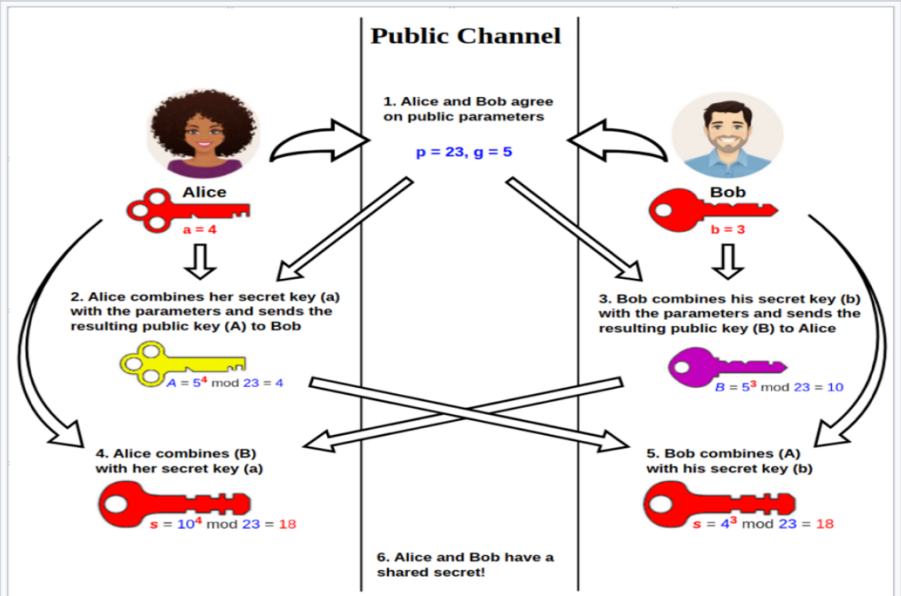
A hybrid approach, combining X25519 and Kyber, can secure communications during the transition to quantum-resistant systems. Additionally, post-quantum digital signatures like Dilithium ensure authenticity, protecting against man-in-the-middle attacks by verifying public keys.

This paper proposes a hybrid key exchange combining X25519 with Kyber and Dilithium signatures to protect IoT communications from both classical and quantum threats.

Diffie-Hellman key exchange

The Diffie-Hellman key exchange uses algebraic methods in symmetric cryptography to securely generate a shared secret key over a public channel without directly transmitting the key. This shared key is then used for encrypted communication between two parties (Gillis, 2022).

How the Key exchange works (Figure 1. Wikipedia, 2019)



With Diffie–Hellman key exchange, two parties arrive at a common secret key, without passing the common secret key across the public channel.

While the original Diffie-Hellman algorithm relies on modular arithmetic with large prime numbers, the Curve25519 version employs elliptic-curve cryptography (ECC), offering improved security and efficiency.

Diffie-Hellman key exchange using Curve25519 (X25519)

Curve25519 uses elliptic-curve algebra, specifically the **Montgomery curve,** to efficiently generate public and private keys, offering **128-bit security** for fast and secure key exchanges. The curve is defined by the equation:

$$y^2 = x^3 + 486662x^2 + x$$

Over the prime field $2^{255} - 19$, with the base point x=9 generating a cyclic subgroup. Co-factor of 8, meaning the number of elements in the subgroup is 1/8 that of the elliptic curve group. Using a prime order subgroup preventing Pohlig-Hellman algorithm attacks (Wikipedia, 2019).

In **X25519**, the X-coordinate represents a point on the elliptic curve, while the Z-coordinate is used in intermediate calculations to avoid slow and insecure division operations. By using XZ coordinates with the **Montgomery ladder**, X25519 enables efficient and secure key exchanges (Wikipedia, 2019).

This design improves performance by only using the X-coordinate, making it immune to timing attacks and allowing any **32-byte string** as a valid public key without needing to verify if it belongs to the curve, simplifying implementation and improving security.

Code Implementation of Diffie-Hellman Key Exchange (X25519) using the Cryptography Library (Cryptography.io, Latest)

```python
from cryptography.hazmat.primitives.asymmetric import x25519

# Generate private keys for Alice and Bob
alice_private_key = x25519.X25519PrivateKey.generate()
bob_private_key = x25519.X25519PrivateKey.generate()

# Generate public keys for Alice and Bob
alice_public_key = alice_private_key.public_key()
bob_public_key = bob_private_key.public_key()

# Exchange public keys and compute shared secrets
alice_shared_key = alice_private_key.exchange(bob_public_key)
bob_shared_key = bob_private_key.exchange(alice_public_key)

# Verify that the shared keys are equal
assert alice_shared_key == bob_shared_key

print("Shared secret established successfully.")
```

```
Shared secret established successfully.
```

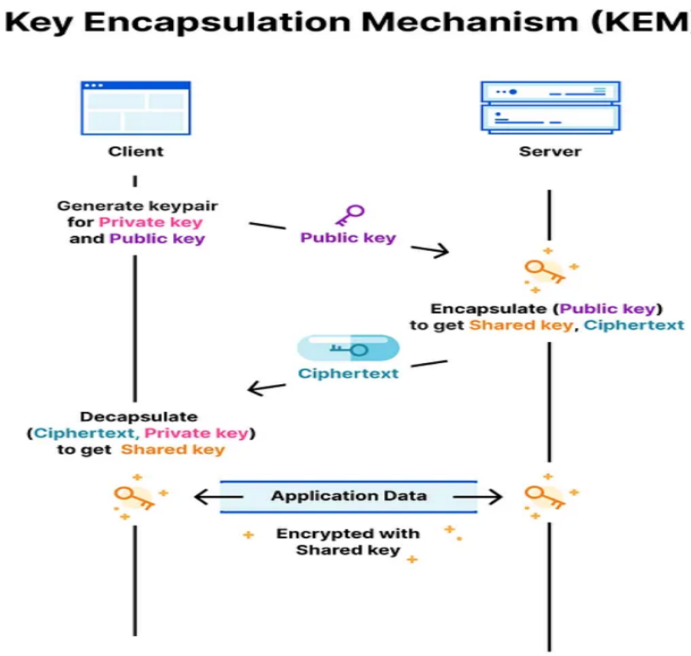
This code demonstrates a Diffie-Hellman key exchange using the X25519 elliptic curve algorithm. First, Alice and Bob each generate their own private and public keys. They then exchange their public keys and use them, along with their private keys, to compute a shared secret independently. The shared secret will be identical for both Alice and Bob, ensuring secure communication. Finally, the code verifies that both shared secrets are the same and prints a confirmation message.

X25519, by itself, does not provide authentication it only facilitates the secure exchange of a shared secret. Without authentication, it cannot verify the identity of the parties involved, leaving it vulnerable to man-in-the-middle attacks. To strengthen this key exchange method and make it future-proof, I integrate the **quantum-resistant Kyber algorithm** with **X25519**, using **Dilithium** for digital signatures to maintain integrity.

Quantum-Resistant Kyber Algorithm.

Kyber is a post-quantum cryptographic algorithm designed to be secure against attacks from quantum computers. It's a **Key-Encapsulation Mechanism (KEM),** which allows two parties to securely exchange a symmetric key using asymmetric cryptography (Pathum, 2024).

Key-Encapsulation Mechanism (KEM) Works (Pathum, 2024)



The client generates a key pair, a private key (kept secret) and a public key (shared with the server). The server then uses the client's public key to encapsulate a shared secret (symmetric key), producing a ciphertext. The ciphertext is sent back to the client, who uses their private key to decapsulate the message and retrieve the shared symmetric key. Both parties now share this key, which can be used to securely encrypt and decrypt data.

Kyber's security is based on the **Learning with Errors (LWE)** problem, which introduces small random errors into linear equations. These errors make it extremely difficult for both classical and quantum computers to solve the system and recover the original secret, ensuring the security of the key encapsulation and decapsulation process (Pathum, 2024).

Code implementation Post-Quantum Key Exchange Using Kyber1024 Algorithm Using the OQS Library

```
# Post-Quantum Key Exchange using Kyber
# Alice generates a key pair and gets her public key
alice_kem = oqs.KeyEncapsulation("Kyber1024")
alice_public_pq = alice_kem.generate_keypair()

# Bob encapsulates a secret to Alice's public key
bob_kem = oqs.KeyEncapsulation("Kyber1024")
ciphertext, bob_shared_pq = bob_kem.encap_secret(alice_public_pq)

# Alice decapsulates the ciphertext to get the shared secret
alice_shared_pq = alice_kem.decap_secret(ciphertext)
```

1. Alice generates a key pair using the Kyber1024 algorithm. This produces:
    - A public key (alice_public_pq), which is shared with Bob.

- A private key, which Alice keeps secret.

2. Bob uses Alice's public key to perform key encapsulation. This process generates:
   - A ciphertext, which Bob sends to Alice.
   - A shared secret (bob_shared_pq), which Bob keeps.
3. Alice receives the ciphertext from Bob. She then:
   - Decapsulates the ciphertext using her private key.
   - Retrieves the shared secret (alice_shared_pq), which is identical to Bob's shared secret.

Dilithium Signature Algorithm

A Dilithium signature is a cryptographic scheme that relies on the hardness of lattice-based problems, specifically the **Shortest Vector Problem (SVP)** and **Learning with Errors (LWE)**. It generates digital signatures using lattice-based algorithms, with security grounded in the computational difficulty of finding short vectors in high-dimensional lattices (SVP) and solving noisy linear equations (LWE) (Schwabe, 2021).

These problems are extremely hard to solve, even for quantum computers, making Dilithium secure against both classical and quantum attacks. Using uniform sampling, Dilithium creates efficient signatures that are resistant to forgery or manipulation, ensuring future-proof security.

Code implementation of The Dilithium Signature Algorithm Using the OQS Library

```
# Alice signs her post-quantum public key
alice_sig = oqs.Signature("Dilithium5")
alice_public_sig = alice_sig.generate_keypair()
signature = alice_sig.sign(alice_public_pq)

# Bob verifies Alice's signature
bob_sig = oqs.Signature("Dilithium5")
if bob_sig.verify(alice_public_pq, signature, alice_public_sig):
    print("Alice's public key is authentic.")
else:
    print("Failed to verify Alice's public key.")
```

Alice generates a key pair using the Dilithium5 signature algorithm and then signs her post-quantum public key, producing a digital signature. Bob receives Alice's public key, her signature key, and the signature. Bob uses this information to verify that the signature is valid, ensuring that Alice's public key is authentic and hasn't been tampered with. If successful, Bob can trust the public key for secure cryptographic operations.

Hybrid Key Exchange: Combining X25519 with Quantum-Resistant Kyber and Dilithium

I combine X25519 Diffie-Hellman with the quantum-resistant Kyber algorithm to deliver future-proof security. To safeguard the integrity of the exchanged public keys, Dilithium digital signatures are used, ensuring authenticity and defending against man-in-the-middle attacks. By integrating these methods, the hybrid exchange secures current communications while preparing for emerging quantum-based threats.

Full Code Implementation using the Cryptography and OQS library.

```python
from cryptography.hazmat.primitives.asymmetric import x25519
from cryptography.hazmat.primitives.kdf.hkdf import HKDF
from cryptography.hazmat.primitives import hashes
import oqs

# Classical X25519 Key Exchange
alice_private_key = x25519.X25519PrivateKey.generate()
bob_private_key = x25519.X25519PrivateKey.generate()

alice_public_key = alice_private_key.public_key()
bob_public_key = bob_private_key.public_key()

alice_shared_classical = alice_private_key.exchange(bob_public_key)
bob_shared_classical = bob_private_key.exchange(alice_public_key)

# Post-Quantum Key Exchange using Kyber
# Alice generates a key pair and gets her public key
alice_kem = oqs.KeyEncapsulation("Kyber1024")
alice_public_pq = alice_kem.generate_keypair()

# Alice signs her post-quantum public key
alice_sig = oqs.Signature("Dilithium5")
alice_public_sig = alice_sig.generate_keypair()
signature = alice_sig.sign(alice_public_pq)

# Alice sends alice_public_pq, alice_public_sig, and signature to Bob

# Bob verifies Alice's signature
bob_sig = oqs.Signature("Dilithium5")
if bob_sig.verify(alice_public_pq, signature, alice_public_sig):
    print("Alice's public key is authentic.")
else:
    print("Failed to verify Alice's public key.")

# Bob encapsulates a secret to Alice's public key
bob_kem = oqs.KeyEncapsulation("Kyber1024")
ciphertext, bob_shared_pq = bob_kem.encap_secret(alice_public_pq)

# Alice decapsulates the ciphertext to get the shared secret
alice_shared_pq = alice_kem.decap_secret(ciphertext)

# Verify that the PQ shared secrets are equal
assert alice_shared_pq == bob_shared_pq

# Combine the shared secrets
combined_secret = alice_shared_classical + alice_shared_pq

# Derive the final shared key using HKDF
derived_key = HKDF(
    algorithm=hashes.SHA256(),
    length=32,
    salt=None,
    info=b'hybrid key exchange',
).derive(combined_secret)

# Both Alice and Bob now share 'derived_key'
print("Hybrid shared secret established successfully.")
```

```
Alice's public key is authentic.
Hybrid shared secret established successfully.
```

Conclusion

This hybrid cryptographic approach combines the strengths of both X25519 Diffie-Hellman key exchange and quantum-resistant Kyber to ensure long-term security against evolving cyber threats. By integrating Dilithium digital signatures, the integrity and authenticity of the exchanged public keys are preserved, providing strong protection against man-in-the-middle attacks. This ensures secure communications not only for current cryptographic standards but also for future quantum-resistant environments. As the Internet of Things (IoT) and other digital infrastructures continue to grow, adopting hybrid cryptographic solutions like this will be essential for safeguarding sensitive information in an increasingly quantum-ready world.

References

Worldwide IoT revenue 2033 | Statista. (2024). Statista. https://www.statista.com/statistics/1194709/iot-revenue-worldwide/#:~:text=IoT%20global%20annual%20revenue%202020%2D2033&text=The%20global%20Internet%20of%20Things

Gillis, A. S. (2022, October). *What is Diffie-Hellman Key Exchange? - Definition from WhatIs.com*. SearchSecurity. https://www.techtarget.com/searchsecurity/definition/Diffie-Hellman-key-exchange

Wikipedia Contributors. (2019, December 10). *Diffie–Hellman key exchange*. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

X25519 key exchange — Cryptography 39.0.0.dev1 documentation. (n.d.). Cryptography.io. https://cryptography.io/en/latest/hazmat/primitives/asymmetric/x25519/

Pathum, U. (2024, Jan 05). *CRYSTALS Kyber: The Key to Post-Quantum Encryption*. Medium. https://medium.com/@hwupathum/crystals-kyber-the-key-to-post-quantum-encryption-3154b305e7bd

Schwabe, P. (2021, Feb 16). *Dilithium*. Pq-Crystals.org. https://pq-crystals.org/dilithium/