Ryan Christopher

CS695 – Milestone 2

An Analysis of “Quantum-Safe integration of TLS in SDN networks”

Table of Contents:

Introduction

"Quantum Safe integration of TLS in SDN networks" by Buruaga, Mendex, Brito, and Martin introduces a ciphersuite for the TLS 1.3 protocol that uses a hybridization of quantum-resistant and classical cryptographic methods. The goal of this paper was to design and test the ciphersuite combining both styles of cryptography to produce a means of secure communication in an environment where adversaries possess classical and quantum machines.

The main contribution of the paper

The main contribution of the paper is the introduction of a new ciphersuite to be employed by TLS 1.3. Buruaga et al. propose the combination of the Diffie Hellman Key Exchange, Quantum Key Distribution, ML-KEM (formerly Kyber – a key encapsulation mechanism “related to the computational difficulty of the Module Learning with Errors problem” [2]), AES-256 in GCM, and SHA384. As such, the proposed ciphersuite name is:

DHKE_QKD_PQC_TLS_AES_256_GCM_SHA_384

This proposed implementation of TLS within an SDN network uses the three shared secret generation methods (DHKE, QKD, and PQC) along with a Key Derivation Function (KDF) in order to generate the cipher between two groups in the SDN.

The motivation that inspired the work

This paper is a direct result from recent notable advancements in the field of quantum computing and the subsequent requirements for post-quantum or quantum-resistant cryptography standardization and implementation. Buruaga et al. argue that "it is imperative to start encrypting the information right now, to mitigate the risk that a

future quantum computer could decrypt data while it remains sensitive." Multiple organizations have publicly announced their quantum chips, joining the often called "race to 1 million qubits," including IBM’s Condor chip, Google’s Willow chip, and Microsoft’s Majorana 1 chip.

As organizations and governments race to develop quantum chips that have higher qubit counts and improved error correction, there is speculation to the specific point that our existing cryptographic methods will be no longer secure. Shor's algorithm, developed in 1994, led to the discovery that quantum systems are particularly capable of factoring numbers and solving the discrete logarithm problem, the "two main staples on which current public key cryptography is based" (Buruaga et al.).

Originally, in 2012, it was estimated that a system with 1 billion qubits could break RSA-2048. However, as their paper suggests, Craig Gidney's and Martin Ekera's "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" published in 2021 argues "the upper end of the estimate of how many qubits will be needed to factor 2048 bit RSA integers has dropped nearly two orders of magnitude; from a billion to twenty million.” For reference IBM's Condor contains 1,121 qubits, Google's Willow contains 105 qubits (with notable improvements in error correction), and Microsoft's Majorana 1 contains 8 topological qubits with the plans to scale the number of qubits very quickly.

These advancements are causing a growing number of researchers and scientist to test post-quantum cryptographic methods in the assumption that the near future will require replacements to existing public key cryptographic methods.

How do authors collect the data, any challenges?

Buruaga et al. conducted a performance evaluation of the proposed quantum-safe SDN-enabled TLS using the ciphersuite previously described. There was a focus on the latency, efficiency, and impact on the security on the system. The data collected measured the time to complete individual steps performed throughout the TLS handshake, and comparing the times to classical methods. To ensure randomness of their tests, Buruaga et al. collected "more than 1,500 instances for each topology with approximately 150 instances recorded for each combination of key exchange" that occurred over a two-week period to capture diverse network states.

A noticeable challenge lies with the QKD component of the system. QKD is introduced as a substitution for DHKE, where "the functionality of the discrete log is performed by a physical process based on the properties of quantum physics" (Buruaga et al). While this method is immune to computational attacks, there must be QKD link to allow this part of the handshake to work. In the performance evaluation, there was testing done on both a direct connection and with the use of an intermediate QKD node. Building an infrastructure to support this system could be time consuming and costly, leading to a potential challenge of this exact implementation. Additionally, Buruaga et al. make an argument that the Key Provisioning System should be well-designed as "the underlying QKD network can introduce significant variability due to fluctuations in the quantum channel."

What metrics are used by authors to analyze the data and why such metrics?

In the results, the time to complete various steps of the TLS handshake is captured, as well as important computational information such as the size of the key, level of NIST

security, and security bits. By using these metrics, we are able to guage a number of important factors when considering the implementation to TLS. The size of the key is important to consider as larger keys require more storage and processing power, where the level of NIST security and security bits show the strength corresponding to the different cryptographic methods that the ciphersuite is composed of. These metrics aid in providing the computational toll versus achieved security in the performance evaluation.

A brief experiment

I spent some time researching the methods used in this paper, and after some time searching I came across the Rust Crypto github repository where a collection of crates for various cryptographic functions are maintained. Among them are both ML-KEM and AES-GCM-256. I first wrote a function to encrypt and decrypt using the standard RSA method. While not the complete picture, I added another function that partially accomplishes the hybridization of classical and PQC methods the authors suggest for cryptographic security moving forward. Then, I passed a text file of random quotes to encrypt and decrypt as test data and compared the results in time required to process each step.

```rust
fn rsa(message: String) -> String {
    let mut results: String = "".to_owned();
    // timestamps for total_time and stopwatch
    let total_time = Instant::now();
    let mut stopwatch = Instant::now();

    // instance of random number generator
    let mut rng = rand::thread_rng();

    // set bit size (therefore key size will be 256 bytes)
    let bits = 2048;

    // generate private and public keys
    let priv_key = RsaPrivateKey::new(&mut rng, bits).expect("failed to generate a key");
    let pub_key = RsaPublicKey::from(&priv_key);
    results.push_str(&format!("Key generation time - {:?}\n", stopwatch.elapsed()));
    stopwatch = Instant::now();

    // encrypt message
    let data = message.as_bytes();
    let enc_data = pub_key.encrypt(&mut rng, Pkcs1v15Encrypt, &data[..]).expect("failed to encrypt");
    results.push_str(&format!("Encryption time - {:?}\n", stopwatch.elapsed()));
    stopwatch = Instant::now();

    // decrypt message
    let dec_data = priv_key.decrypt(Pkcs1v15Encrypt, &enc_data).expect("failed to decrypt");
    results.push_str(&format!("Decryption time - {:?}\n", stopwatch.elapsed()));

    // verify initial message and decrypted message are the same
    assert_eq!(&data[..], &dec_data[..]);
    results.push_str(&format!("Total time - {:?}\n", total_time.elapsed()));

    results
}
```

```rust
fn kyber(message: String) -> String {
    let mut results: String = "".to_owned();
    // timestamps for total_time and stopwatch
    let total_time = Instant::now();
    let mut stopwatch = Instant::now();

    // instance of random number generator
    let mut rng = rand::thread_rng();

    // encapsulate = public key, decapsulate = private key
    let (dk, ek) = MlKem768::generate(&mut rng);
    results.push_str(&format!("Key generation time - {:?}\n", stopwatch.elapsed()));
    stopwatch = Instant::now();

    // encapsulate shared key to the holder of the decapsulation key, receive the shared secret 'k_send'
    // and the encapsulated form 'ct'
    let (ct, k_send) = ek.encapsulate(&mut rng).unwrap();
    results.push_str(&format!("Encapsulation time - {:?}\n", stopwatch.elapsed()));
    stopwatch = Instant::now();

    // decapsulate the shared key
    let k_recv = dk.decapsulate(&ct).unwrap();
    results.push_str(&format!("Decapsulation time - {:?}\n", stopwatch.elapsed()));

    // verify shared keys are identical
    assert_eq!(k_send, k_recv);
    stopwatch = Instant::now();

    // take symmetric key generated from kyber, pass as GenericArray
    let key = Key::<Aes256Gcm>::from_slice(&k_send);

    // create cipher from key
    let cipher = Aes256Gcm::new(&key);
    results.push_str(&format!("Key and cipher generation time - {:?}\n", stopwatch.elapsed()));
    stopwatch = Instant::now();

    // initialize nonce/IV to be used in combination with encryption key
    let nonce = Aes256Gcm::generate_nonce(&mut OsRng);

    // encrypt message
    let ciphertext = cipher.encrypt(&nonce, message.as_bytes().as_ref()).unwrap();
    results.push_str(&format!("Encryption time - {:?}\n", stopwatch.elapsed()));
    stopwatch = Instant::now();

    // decrypt message
    let plaintext = cipher.decrypt(&nonce, ciphertext.as_ref()).unwrap();
    results.push_str(&format!("Decryption time - {:?}\n", stopwatch.elapsed()));

    // verify initial message and decrypted message are the same
    assert_eq!(&plaintext, message.as_bytes());
    results.push_str(&format!("Total time - {:?}\n", total_time.elapsed()));

    results
}
```

The script I wrote uses ML-KEM (formerly Kyber) to create a public and private keys which are used to encapsulate a shared key between two users. Once the shared key is established, the messages are encrypted via AES-GCM-256 using the ML-KEM key generated, and then decrypted using the same key. Similar to the results of Buruaga et al., the hybrid approach, while appearing to have more steps and use more complicated methods, is far more efficient in key generation time (and therefore overall time) compared to an existing asymmetric key method like RSA.

ML-KEM/Kyber

```
Key generation time - 1.8897ms
Encapsulation time - 1.9382ms
Decapsulation time - 2.5266ms
Key and cipher generation time - 3.9µs
Encryption time - 16µs
Decryption time - 10.9µs
Total time - 6.3937ms
```

RSA

```
Key generation time - 1.2694284s
Encryption time - 2.2582ms
Decryption time - 21.2579ms
Total time - 1.2929529s
```

Significant findings, insights, and conclusions

Initially the implementation of post quantum cryptography into existing systems was seen as impractical due to it's computational weight. The results from this paper propose implementations that do not take as heavy of a computational toll as once thought, as algorithms like ML-KEM and ML-DSA are improved upon and tested more comprehensively. In particular, the operation time of PQC (159.82 microseconds) and QKD (234.28 microseconds) compared to the existing operation time of Diffie-Hellman (460.38 microseconds) show that the hybridization of the TLS protocol incurs a minimal penalty in performance.

While a minimal penalty in performance is gained, a substantial level of security is gained in the SDN networks described in this paper and has the capability to protect a network environment against classical and quantum attacks.

In their closing remarks, Buruaga et al. stated “this finding underscores the viability of hybridization as a practical approach to transitioning current security infrastructures toward quantum-safe ones, protecting data transfers even in the long term."

References

[1] Buruaga, Jaime S., et al. "Quantum-Safe integration of TLS in SDN networks." *arXiv preprint arXiv:2502.17202* (2025).

[2] --, “Module-Lattice-Based Key-Encapsulation Mechanism Standard,” U.S. Department of Commerce, Washington, D.C., Tech. Rep. Federal Information Processing Standards Publications (FIPS PUBS) [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203

[3] Gidney, Craig, and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits." *Quantum* 5 (2021): 433.