Effects of PQC on Wireguard protocol

Qiguang Wang

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Future Network System)

Supervisor: Stephen Farrell

August 2023

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Qiguang Wang

August 23, 2023

Effects of PQC on Wireguard protocol

Qiguang Wang, Master of Science in Computer Science

University of Dublin, Trinity College, 2023

Supervisor: Stephen Farrell

This paper introduces KEM-WireGuard, a post quantum secure instantiation. This study is based on the KEMTLS protocol (ACM CCS 2020), which uses key-encapsulation mechanisms (KEMs) instead of post quantum signatures to authenticate communication parties. It brings both the TLS and the traditional post-quantum TLS into a more efficient protocol. Notably, this variant does not merely put focus on few aspects of security such as authentication and forward secrecy, as commonly pursued by earlier research at designing post-quantum protocols. Instead, it also offer bilateral post-quantum authentication with moderate cost of computation and communication. To accomplish this, we substitute the existing Elliptic Curve Diffie-Hellman key exchange with a new asymmetric crypto primitive, namely KEM. This thesis explores various combinations of different KEMs to alter existing WireGuard implementations. We propose one generic construction and two incremental modifications, providing different level of security against quantum threats.

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, Stephen Farrell, for his exceptional guidance, mentorship, and strongc support throughout the course of this research project. His insightful advice, constructive feedback, and dedication to my academic growth have been instrumental in shaping the direction and quality of this work.

Thanks to Stephen's profound expertise, and unwavering support, the whole process of the research becomes smoother. His encouragement to explore innovative approaches and his patient guidance makes a valuable contribution.

I am deeply appreciative of the time and effort Stephen has put in my research. His commitment to my success and his role as a mentor have greatly influenced my development as a researcher and scholar.

Lastly, I would like to express my heartfelt thanks to my family and friends for their support, understanding, and encouragement throughout this work. Their belief in me has been a constant source of motivation.

In conclusion, I am truly grateful for the guidance and mentorship of Stephen Farrell, which has been pivotal in shaping the outcome of this research.

Qiguang Wang

University of Dublin, Trinity College
August 2023

Contents

List of Tables

List of Figures

Chapter 1

Introduction

WireGuard Donenfeld (2017) is an innovative secure network protocol that functions as a virtual network interface within the Linux kernel in most cases. Its primary goal is to serve as a drop-in replacement for similar secure network protocol including IPsec and OpenVPN, providing enhanced security, better performance and improved friendliness. The secure tunnel is established by binding the peer's public key with its corresponding source address. The design of WireGuard is directly deriving from the Noise framework, bearing in mind the problem of pervasive surveillance and active tampering. Also, to get over the bandwidth and computational cost, WireGuard streamlines the handshake process, utilizing a single round trip to achieve security of both perfect forward secrecy (PFS) Diffie et al. (1992) Günther (1990) and key-compromise impersonation (K-CI) resistance Boyd and Mathuria (2003) Just and Vaudenay (1996) Blake-Wilson et al. (1997) But, all these security properties only holds in the classical computer model. With the arrival of the quantum computers, it is necessary to redefine of WireGuard Diffie-Hellman (DH) style handshake to use key encapsulation mechanisms (KEMs) Cramer and Shoup (2001) for authentication, rather than pre-shared static key as a transitional method.

1.1 Migrating to post-quantum WireGuard

To achieve higher security levels against quantum adversaries without losing the same security level against the classical adversaries, the original WireGuard protocol needs to be revised. The main focus of this research is on amending the handshake part of the WireGuard.

1.2 Motivation

1.2.1 WireGuard

WireGuard provides a very simple and novel interface for setting up secure channels. All the cryptography primitives are cutting edge: Curve25519 Bernstein (2006), ChaCha20 Bernstein (2008), and Poly1305 Bernstein (2005) and the complexity, as well as the sheer amount of code is ignorable. Unlike TLS, which offers backward compatibility and cryptography agility. The WireGuard hard-codes all the cryptography primitive choices. It's one of the reasons why WireGuard has gained a reputation for performance and reliability.

In the meantime, these primitives provide unparalleled degree of performance in the first place, introduce fewer dependencies and less complexity to manage which will make it easier to ensure that the new design is secure. Furthermore, Its simplicity and minimalist design philosophy aligns well with the current state of post-quantum cryptography. Many post-quantum schemes are still being analyzing and optimized, and are often more computationally intensive or require larger key sizes than their classical counterparts.

Given the diverse range of performance trade-offs of post-quantum algorithms, exploring alternatives to the TLS-like ECDH combined with signature seems a sound way of improvement such as the KEMTLS Schwabe et al. (2020) protocol and its state-of-art variant KEMTLS-PDK Schwabe et al. (2021) which is the abbreviation for KEMTLS with pre-distributed keys. Both replace the ECDH and signature with KEMs.

1.2.2 Post-Quantum Cryptography

In recent years, there has been a considerable amount of research on quantum computers which can solve specific mathematical problems that are difficult for classical computers by exploiting quantum phenomena. In the foreseeable future, the large-scale quantum computer probably will bring unprecedented speed and power in computing to reality. However, this quantum edge also poses serious problems which now is so-called *quantum apocalypse* - "store now, decrypt later". One of such threats lies in the domain of public key cryptography. When the time comes, the quantum computers could potentially break many of the cryptographic systems, which were designed and secured under the premises of classical computing. It's crucial that we adopt a proactive approach and remain one step ahead of the potential risks. We can not afford to idle until the quantum computers begin dismantling today's public-key cryptography is running. Therefore, the goal of this project is to scrutinize the vulnerable part of the WireGuard and propose a new WireGuard which is post-quantum safe while still keeps the virtue of original WireGuard protocol.

The necessity for quantum safe cryptography stems from the capability of Shor's algorithm's ability to efficiently resolve problems related to factorization and discrete logarithm which are the security backbones of RSA Rivest et al. (1978), ECC Miller (1986) and ECDSA American National Standards Institute, Inc. (2005) that rely on the *IFP* (integer factorization problem), the *DLP* (discrete logarithms problem) and the *ECDLP* (elliptic-curve discrete logarithm problem) respectively. By using Shor (1994)'s algorithm, a k-bit (in binary) number can be factored in the time of $O(k^3)$ using a quantum computer with $5k + 1$ qubits. As the NIST standardization process is ongoing, this standardization doesn't assert the supremacy of one suggestion over another. Given that the NIST standardization process has already chosen the Selected Algorithms 2022 Schwabe et al. (2022), the most secure transition method will be integrate the standardized Post-Quantum Cryptography algorithms instead of waiting for national bodies to finalize PQC algorithms in which case where the risk associated with quantum broken cryptography is not acceptable.

1.3 Goals

The major goal of this thesis is to add post-quantum cryptography into WireGuard, aiming to uphold perfect forward secrecy (PFS), mutual authenticity, and adaptive security. Appelbaum et al. (2019) proposed an efficient approach, applying to the WireGuard protocol that can achieve transitional post-quantum security to the WireGuard via utilizing optional pre-shared key (PSK) value. This key, calculated independently of the key agreement protocol, could be used to initialize a post quantum(PQ) key exchange. In order to address this issue, it is necessary to integrate PQ key agreements directly into the WireGuard protocol.

While addressing these primary goals, the secondary objectives revolve around maintaining optimal performance and clean interface. Thus, this paper strives to keep a balance between advancing WireGuard's security level to confront quantum threats, without compromising its hallmark simplicity and performance.

1.4 Contribution

In this thesis, we introduce KEM-WireGuard, a generic post-quantum augmentation to the WireGuard protocol. This innovation not only improves the security design to keep secret against quantum attacks, as done in previous attempts at transitioning to post-quantum security, but also ambitiously aims for comprehensive post-quantum security,

encompassing authentication as well. We also present the KEM-based instantiation of post-quantum WireGuard using Golang native cryptographic library - CIRCL (Cloudflare Interoperable Reusable Cryptographic Library). We focus on a coherent and cohesive implementation while considering the unique characteristics of different KEMs. We employ various combinations of KEMs to maximize the utilization of their unique performance and key size characteristics, with the ultimate goal of achieving optimal bandwidth and performance.

Our approach involves benchmarking and comparing various instantiations of the post-quantum WireGuard variants with previous implementations of PQ-WireGuard and the original WireGuard. Consequently, it should :

- Attain all the security characteristics of WireGuard while also being resilient against attacks by a large-scale quantum computer
- Reach a firm decision on the cryptographic primitives explicitly, thereby eliminating the need for an cryptographic negotiation.
- Reduce the on-wire format size as well as the memory consumption.

The original WireGuard heavily relies on the ECDH key agreement, which is not a simple task to replace with PQ schemes. It's important to note that while the application of post-quantum algorithms in the WireGuard protocol may increase security margin, it's crucial to carefully evaluate their implementation and performance to ensure they meet the necessary standards for secure communication. Given that SIDH/SIKE Galbraith and Vercauteren (2017) can be compromised in classical polynomial time, it's definitely left out of the consideration. The only feasible post-quantum key exchange is CSIDH Moriya et al. (2020), but its security is relatively new and unproven. So we decided to follow the round-3-submissions Albrecht et al. (2020) and round-4-submissions Aragon et al. (2022), modifying the WireGuard protocol to use interactive key-encapsulation mechanisms (KEMs) exclusively.

Chapter 2

Background

2.1 Quantum Algorithms

2.1.1 Shor's Algorithm

Shor's algorithm, introduced by mathematician Peter Shor in 1994, is a quantum algorithm which can efficiently solve the factoring problem (get p and q from $n = p \cdot q$) and the discrete logarithm problem (get e from $x = g^e \mod p$). It has the potential to render all widely deployed key agreement and digital signature schemes, which are both critical to the security of the Internet, obsolete, including RSA and elliptic curve cryptography. The major application of Shor's algorithm lies in its ability to find the prime factors of large numbers, a task that is exceptionally difficult for classical computers to perform. Shor's algorithm, when run on a sufficiently powerful quantum computer, can factorize these large numbers exponentially faster than the best known classical algorithm.

Consequences of Shor's algorithm:

- Quantum order-finding algorithm can be implemented in $O(n^3)$ quantum gate steps ($k = \log n$), **bounded-error quantum polynomial time** (BQP).
- Factoring is solvable in quantum polynomial time.
- Modified Shor can also solve discrete logarithm problem, which means it totally breaks discrete log-based and elliptic curve cryptography.

Therefore, the underlying basis of many modern public key cryptographic systems, including RSA, DSA, and elliptic-curve cryptography (ECC), rests on the difficulty of the integer factorization or the discrete logarithm problem, will be dead in the presence of a large-scale quantum computer, thus providing a significant impetus for the development of post-quantum cryptography.

2.1.2 Grover's Algorithm

Grover's algorithm, devised by Grover (1996), is another significant quantum algorithm. It uses amplitude amplification to efficiently perform unstructured search to find the unique input for a black box function that leads to the corresponding output. Meanwhile, While a classical computer would need $O(n)$ steps to complete the search, Grover offers a quadratic speedup over classical algorithms for similar search problems. In cryptographic terms, Grover's algorithm can effectively halve the key length of symmetric cryptographic systems, reducing, for example, the security level of a 256-bit key to that of a 128-bit key in classical terms.

The application of Grover's algorithm could seriously jeopardize symmetric key algorithms by significantly speeding up exhaustive key search attacks or brute force attacks, effectively halving the cryptographic key length. For an n-bit symmetric cryptographic algorithm, there are 2^n possible keys. With the current computing platforms, the key range of 2^{128} for a 128-bit AES algorithm is virtually uncrackable. However, with the advent of quantum platforms and the implementation of Grover's algorithm, the security level of AES's 128-bit key size would effectively be reduced to an insecure 64-bit equivalent key length. Fortunately, most symmetric key algorithms provide additional key lengths, enabling applications to shift to the more secure versions.

Hash algorithms are also likely to be impacted by Grover's algorithm, given their nature of producing a fixed-sized output from inputs of arbitrary size. The enhanced pace of Grover's algorithm could potentially speed up collision attacks, which involve identifying two distinct inputs that yield the same output. The emergence of quantum-based platforms also poses a challenge for hash algorithms. However, hash functions such as SHA-256, which yields a 256-bit output, and SHA-512, with a 512-bit output, appear to remain resistant to quantum attacks due to their longer output lengths.

However, unlike Shor's algorithm, Grover's algorithm doesn't pose an existential threat to all symmetric cryptographic systems. The serial processing requirement of Grover's algorithm limits its impact on the symmetric cryptography. Additionally, the threat can be mitigated by simply doubling the key length, a change that would impose minimal overhead on modern computational systems but exponentially increases the time required by Grover's algorithm to find a match.

In conclusion, the edge of Grover's algorithm indeed stresses the need for transitioning to post quantum cryptographic systems that can withstand quantum adversaries, but it has less devastating effect comparing to Shor's algorithm.

2.2 Related work

WireGuard's most important security component is the authenticated key exchange, a critical part that determines the overall security of the protocol. Authenticated key exchange (AKE) is a cryptographic protocol that allows two entities to securely derive a shared secret over an insecure channel and give assurance that both entitles is the intended one. An famous type of AKE is the ECDH based key exchange, which is favored for its flexibility and ability to offer security attributes such as forward secrecy. However, many practical AKE protocols rely on a combination of (EC)DH and digital signatures, including WireGuard.

As we will move into the post-quantum era soon, relying on the security of the Diffie-Hellman assumption will become unacceptable. Thus, it is crucial to transition towards post-quantum cryptographic (PQC) alternatives that are believed to resist quantum computer attacks. The research community has put great effort into this area. NIST's Post-Quantum Cryptography Standardization ahs split the public-key cryptosystems into public key encryption and key establishment algorithms since round 2 Submissions. The result of this process is Selected Algorithms 2022: CRYSTALS-KYBER Schwabe et al. (2022) for public key encryption plus Round 4 Submission. Considering the mathematical categories of the remained algorithms, they all fall within three mathematical assumptions:

Hash-based solutions apply its hash to transform input into an diffused, variable size one. Its one-way property as they are computationally hard to be computed in reverse is what signature schemes require.

Lattice-based schemes are based on a set of defined basis vectors. their security relies on the difficulty of solving the Shortest Vector Problem (SVP). While calculating, noise is inserted to make it hard to recover the secret.

There are essentially two viable options to migrate to post quantum cryptography schemes: The first option is to migrate to hybrid implementations that use a combination of pre-quantum and post-quantum schemes. Schemes of this kind enable backwards compatibility while preventing potential weaknesses in post-quantum algorithms from causing attack surfaces. The second option is directly deploying the bleeding-edge post-quantum schemes which is exact what this research is doing.

2.2.1 Transitional post-quantum constructions

A popular approach to transitional post-quantum security is to combine ECDH with a post-quantum primitive, forming a hybrid quantum-classical cryptography. This approach provides the benefits of the well-studied classical security while also providing a layer of post-quantum security via the use of a post-quantum algorithm. In Bindel et al.

(2019a) established the models for hybrid authenticated key exchange protocols. Also, Google and Cloudflare conducted the CECPQ2 experiment that intended to help evaluate the combination of X25519 Bernstein (2006) and a post-quantum key-agreement based on NTRU-HRSS-KEM Schanck et al. (2017) a plugin for the TLS key-agreement part. This experiment implemented two hybrid quantum-classical key exchanges: CECPQ2 (lattice-based NTRU-HRSS + X25519) and CECPQ2b (isogeny-based SIKE + X25519), integrated them into their TLS stack and deployed on their servers and in Chrome Canary clients. The post-quantum components are KEM-based algorithms and the final shared secrets are concatenate and combine with HKDF(HMAC-based Extract-and-Expand Key Derivation Function) Krawczyk (2010a). In Crockett et al. (2019), the authors prototyped and inspected different design of combining classical and post-quantum algorithms.

2.2.2 KEM-based authenticated key exchange constructions

Typically, the underlining algebraic structures of some PQC can provide a Public Key Encryption (PKE), which can then be converted into a Key Encapsulation Mechanism (KEM) through a standard transformation construction. However, a particular characteristic of these algebraic structures used in many post-quantum PKEs and KEMs lead to malleable ciphertexts, expose them to chosen-ciphertext attacks (CCA) Cramer and Shoup (1998). To counter the effect, the most common way is to apply the Fujisaki-Okamoto (FO) transform Fujisaki and Okamoto (1999) or its variants and upgrade OW-CPA security to IND-CCA(2) security. In Hövelmanns et al. (2018) proposed a modification of Fujisaki-Okamoto AKE that can turn any deterministic PKE into FO-like IND-CCA secure KEMs in the quantum random oracle model. And Bindel et al. (2019b) tighten the bound and extended their result to the case of non-deterministic PKEs or PKEs feature decryption failure. There are excellent examples show that KEM is a drop-in replacement alternative to build the ability of key agreement without the traditional signature schemes and Diffie-Hellman key agreement schemes. In Schwabe et al. (2020) the authors introduce KEMTLS, a novel approach to implement handshake protocol of the TLS 1.3 that make full of IND-CCA-secure KEMs as a new way of sever/client authentication instead of signatures. Hülsing et al. (2021a) present PQ-WireGuard that filled the emptiness in the area of post-quantum authentication.

2.3 Cryptographic building blocks

2.3.1 ECDH

Vaudenay (2005) proposed a remarkable solution for establishing shared secret keys. It can work over any group as long as the normal discrete logarithm problem (DLP) is hard in the given groups. Furthermore, a much more efficient key exchange method ECDH was proposed. The Elliptic Curve Diffie-Hellman (ECDH) is a variant of the Diffie Hellman key exchange over groups based on elliptic curves which differs from the previous DH in their cyclic groups. The security of the DH key exchange relies on the difficulty of the DLP in $\mathbb{Z}_p^*$ (the multiplicative group modulo prime p). On the other hand, the security of ECDH relies on the difficulty of Elliptic curve discrete logarithm problem (ECDLP) which is believed to be harder than DLP. The ECDH is then widely deployed due to its smaller key size, fast and usually constant-time multiplication and sound security assumption.

1. During setup phase, A and B agree on the public parameters of elliptic curve group with a fixed prime p and point $\mathbf{P}$ on this curve.
2. A generates a random secret α and B generates a random secret β.
3. A computes its public key Q as $Q \leftarrow \alpha \cdot \mathbf{P} \pmod{\mathbf{p}}$ and sends B Q .
4. B computes its public key R as $R \leftarrow \beta \cdot \mathbf{P} \pmod{\mathbf{p}}$ and sends A R .
5. Both can derive a shared secret $K = \alpha \cdot R = \alpha \cdot \beta \cdot P = \beta \cdot Q$.

An eavesdropper can only see the tuple of $p, P, \alpha \cdot \mathbf{P} \pmod{\mathbf{p}}, \beta \cdot \mathbf{P} \pmod{\mathbf{p}}$, and would need to break the Elliptic Curve Diffie-Hellman problem to learn the shared secret. However, an naive implementation without authentication leaves this protocol subject to a man-in-the-middle attack.

2.3.2 Noise Protocol Framework

Noise Protocol Framework (Noise) Ho et al. (2022) is a framework for building cryptographic secure key agreement. Each variant of the framework provides key exchange patterns with a subtle difference, due to the way they achieve authentication and forward secrecy. Unlike concrete implementations like TLS and IPSec, it provides a template for developers who want to write their own security protocols. Although the latest TLS 1.3 has removed some obsolete algorithms and has a more modern architecture, it still requires a certificate authority (CA) to issue server and client certificates, which means it is inevitable to involve the complexity of Public key infrastructure (PKI).

Noise provides a set of tools and patterns for designing and implementing secure communication protocols, including IKpsk2 which the WireGuard handshake is based on. By using the Noise framework, it is possible to design a KEM-based WireGuard to addresses the limitations of the current DH-based key exchange. Here are the necessary notations of Noise:

Handshake pattern names are designated by a single character that represents how to distribute the sender's long-term static key:

- I ← Static key for sender **I**mmediately sent to receiver
- K ← Static key for sender **K**nown to receiver
- 1 ← Which message is the authentication DH deferred to
- *psk* ← Both parties have a identical pre-shared symmetric key

In canonical form, the left-most entity will be assumed as the initiator and the right-most entity will be regarded as the responder. Each entity needs to keep track of these following variables:

- **s, e**: The **local** entity's static and ephemeral key pairs
- **rs, re**: The **remote** entity's static and ephemeral **public** keys.
- **h**: A handshake hash that hashes all the handshake data
- **ck**: A chaining key that hashes all previous DH authentication data.

2.3.3 WireGuard handshake

WireGuard's handshake can be represented by the IKpsk2 pattern shown in Figure 2.1, defined by the Noise Protocol Framework.

IKpsk2:
← s
...
→ e, es, s, ss
← e, ee, se, psk

Figure 2.1: Noise notation of IKpsk2

According to the Nose framework, "IKpsk2" can interpreted be as follows:

- **I**: Static key for initiator will be sent to receiver in plaintext in the first message.

- **K**: The static public key is pre-distributed to initiator before handshake start.
- **psk2**: The pre-shared symmetric key will be feed into chaining key after the second message.

In Figure 2.1, we give an overview of the generic IKpsk2 handshake pattern, omitting all the specific details of massages and calculations. Before any connection, both entity already have the other side's pre-distributed public key and their own private key. The combination of the direction indicated by the arrow and the (right/left most) single character indicates the type of DH values.

The initiator starts by sending its ephemeral and static public key along with the corresponding DH authentications to the responder. Then, the responder will also perform the corresponding DH and send the result back to the initiator. At the end, both can finish the handshake by calculating the last DH which is the input key material for two symmetric keys. These is so-called 3-way DH. In a Noise key exchange, both parties keep updating the chaining key and hash during the handshake. They update the chaining key after each shared secret is established, using a key derivation function (KDF) that takes the shared secret and old chaining key as inputs.

Now that we've covered the necessary background information, we can move on to examining the WireGuard's handshake. In WireGuard scenario, it shares the same handshake pattern. WireGuard is just a instantiation with extra defence methods to reduce the attack surface.

In the initial handshake message of WireGuard, the following information will be sent:

- ephemeral: The initiator's temporary public key generated for this handshake (not encrypted, used for ECDH)
- static: The initiator's encrypted static long-term public key
- timestamp: The encrypted timestamp of the current time

In the response handshake message of WireGuard, the following information will be sent:

- ephemeral: The responder's temporary public key generated for this handshake (not encrypted, used for ECDH)

2.3.4 Key encapsulation mechanisms

A KEM is an asymmetric post quantum cryptographic primitive that enables two parties A and B to agree on a shared secret ss in a key space $\mathbb{K}$. It has the following operations:

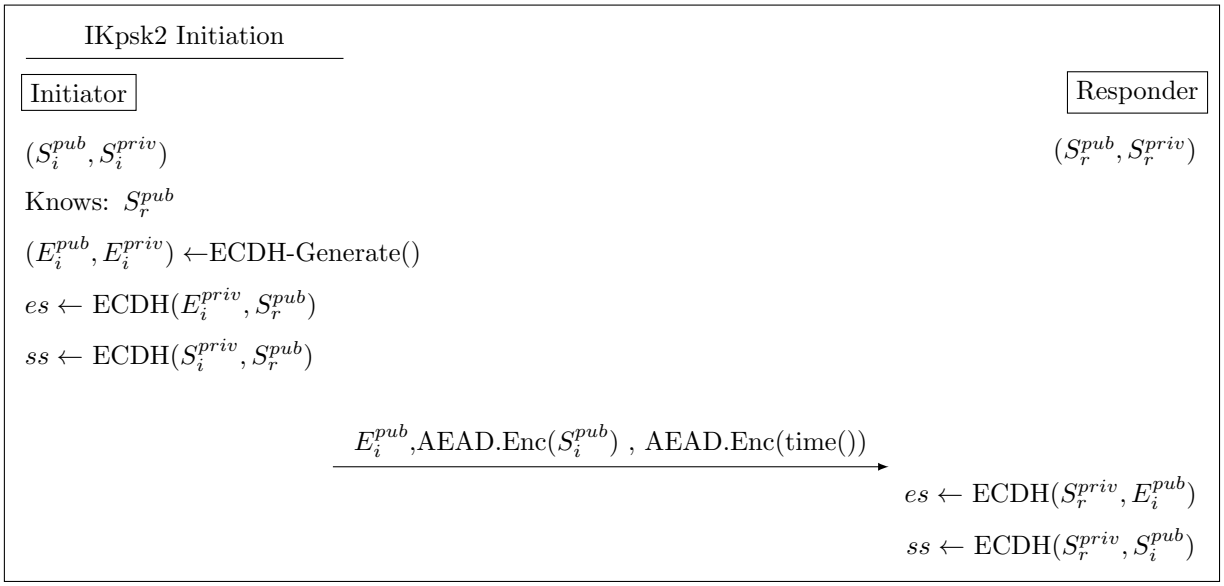

Figure 2.2: Initiation message of WireGuard

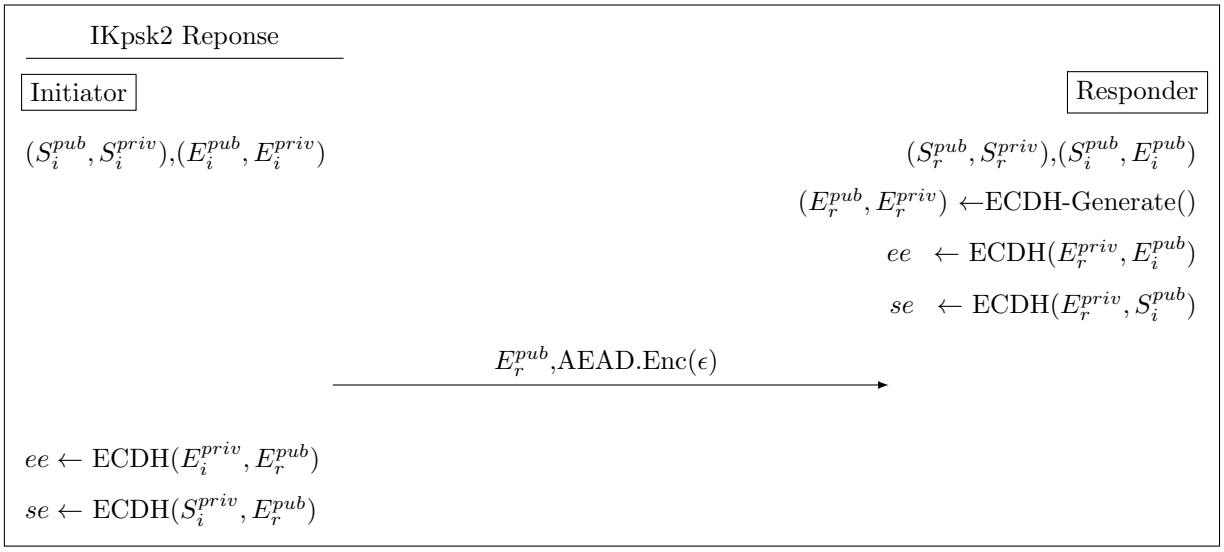

Figure 2.3: Response message of WireGuard

- **Key generation**: KEM.Keygen() probabilistically generates a public and private keypair (pk, sk)
- Encapsulation: KEM.Encapsulate(pk) probabilistically generates a fresh shared secret and ciphertext/encapsulation (ss, ct) against a given public key;
- Decapsulation: KEM.Decapsulate(ct, sk) decapsulates to the same shared secret ss against a given private key.

All KEM primitives evaluated here are round 4 submissions of the NIST Post-Quantum Cryptography project.

Algorithm	Family	Status	Advantage	Drawback
CRYSTALS-Kyber	Lattice	NIST winner	Secure, Fastest	
BIKE	Code	NIST round 4	Secure	
Classic McEliece	Code	NIST round 4	Secure	Large keys, Slow
HQC	Code	NIST round 4	Secure, Fast	
SIKE	Isogeny	NIST round 4	Insecure	Broken

Figure 2.4: Overivew of KEMs

- Classic McEliece Albrecht et al. (2022): It is a IND-CCA2 KEM designed with conservative security margin. It is constructed from a one-way public key encryption(pke).

  The first code-based pke was introduced in 1978 by McEliece McEliece (1978).It utilizes binary Goppa code. A ciphertext is a codeword with random errors. The private is the redundancy to recover plaintext from the encryption.

  This type of KEMs has a serious drawback: its public/private key size is too large to be transmitted on the wire.

  For McEliece, they can fall into two camps: systematic form (non-f) and semi-systematic form (f). They are distinct from each other due to the way of generating the keys. The non-f variant is more efficient at generating the key but has a larger failure rate and probably need more tries to re-generate. The semi-systematic form one has a lower failure rate, but is slightly slower than the systematic form.

- CRYSTALS-Kyber: The "Cryptographic Suite for Algebraic Lattices" (CRYSTALS) encompasses two cryptographic primitives: Kyber, an IND-CCA2 KEM scheme; It is based on hard problems over module lattices, are designed to be secure in face of quantum computers.

  Module lattices can be thought of as lattices that lie between the ones used in the definitions of the Learning with errors problem (LWE) Regev (2005), and those used for the Ring-LWE problem Lyubashevsky et al. (2012). If the ring underlying the module has a sufficiently high degree such as 256-bit, then these lattices inherit all the efficiency of the ones used in the Ring-LWE problem

  Kyber is the Selected Algorithm 2022. The submission provides three variants aiming at different security levels. Specifically, Kyber-512 roughly achieves security Level I, Kyber-768 roughly achieves security Level III, and Kyber-1024 achieves security Level-V.

- BIKE: BIKE Vasseur (2021) is a code-based key encapsulation mechanism based

on QC-MDPC (Quasi-Cyclic Moderate Density Parity-Check) codes and have been the selected as the alternative KEM to Kyber.

- HQC: HQC (Hamming Quasi-Cyclic) Aguilar Melchor et al. (2022) is a also code-based KEM scheme designed to provide security against attacks by both classical and quantum computers.

2.3.5 Pseudo-random functions

A Pseudorandom Function (PRF) Goldreich et al. (1986) is a function defined over 2-tuple (K, X) that (ideally) is indistinguishable from be randomly selected function from the set of all functions with the same domain and value set. There should be efficient algorithms to calculate $F : K \times X \to Y$ The PRF is secure iff a random function in $Funs[X, Y]$ is indistinguishable from a random function in S_F

$$\begin{cases} Funs[X,Y] : \text{the set of } \textbf{all} \text{ functions from X to Y } (Size|Y|^{|X|}) \\ S_F = F(k,\cdot) \text{ s.t. } k \in K \ (Size|K|) \end{cases}$$

2.3.6 BLAKE2

BLAKE2b and BLAKE2s are cryptographic hash proposed by Aumasson et al. (2013). It is derived from the SHA-3 proposal BLAKE by the team. It has two families, each optimized for specific platforms. For example, Blake2b is optimized for 64-bit platforms and provides a higher security margin, while Blake2s is optimized for smaller CPU architectures with a emphasis on memory footprint.

Internally, BLAKE2b is a message digest using 64-bit words and produce output range from 1 to 64 bytes. BLAKE2s is a message digest using 32-bit words and produce output range from 1 to 32 bytes. BLAKE2 is a versatile cryptographic hash function that offers a variety of algorithms tailed for different requirements. Here's a brief overview of each algorithm:

1. Keyed hashing (MAC or PRF): It is a variant of keyed hash functiom: It is initiated with a secret value and then feed the data to produce a MAC. This MAC tag provides both integrity and authentication. Wireguard uses it as underlying hash used to construct Key KDF and MAC function.

2. Incremental tree-hashing: It has the ability of incremental hashing, where the hash value is updated continuously as new data is mixed into its internal state.

3. Combinations: BLAKE2 allows for combinations of the above algorithms, enabling users to choose the best approach for their specific use case. For example, users might want to use a keyed hash with a salt for added security, or combine incremental hashing with a MAC for efficient authentication and integrity checking.

During the handshake, the BLAKE2s is used with fixed 32 bytes output. It is not only used for the continuous hash, but also as PRF in key derivation based on HMAC-based Key Derivation Function (HKDF) Krawczyk (2010b). HKDF is a key-derivation function built on top of Hash-based Message Authentication Code (HMAC) Bellare et al. (1996). On the contrary, the chaining hash employs hash directly. This implies that the standard collision attack with a complexity of $O(2^{n/2})$ remains. It's important to note that the security of the chaining hash value does not require the collision resistance.

2.3.7 AEAD

Authenticated encryption with associated data (AEAD) designed by Black (2005) is a symmetric encryption that performs secret key encryption and authentication. Prior to the existence of AEAD schemes, ensuring the integrity/authentication and confidentiality of plaintext were two distinct design objectives.

Authenticated encryption is a cryptographic primitive that combines both privacy (confidentiality) and integrity into a single primitive. It ensures that the data being encrypted by AE schemes is both encrypted and authenticated, meaning that not only the content is indistinguishable from attackers, but also any tampering or unauthorized modification can be detected. This is achieved by using symmetric encryption and adding authentication tags to the encrypted data, allowing the anyone with the secret key to verify both the authenticity and the integrity of the data. Associated Data (AD) refers to information that cannot be encrypted itself but still requires protection for its integrity. In cryptography, particularly in the context of AEAD schemes, AD is additional data that is included alongside the plaintext before encryption. This data is not encrypted but is included in the computation of the authentication tag.

All data sent over a WireGuard is encrypted by AEAD under the session key and the chaining hash used as AD. Although the chaining hash value is not sent on the wire, it ties all data produced in different phrase of the handshake to the according ciphertext.

$$\begin{aligned} (pk, sk) &\leftarrow\!\!\$\ KEM.Keygen() \\ (ss, ct) &\leftarrow\!\!\$\ KEM.Encapsulate(pk) \\ ss &\leftarrow\ KEM.Decapsulate(ct, sk) \end{aligned}$$

Figure 2.5: AE scheme

- **ChaCha20-Poly1305** as described in RFC 8439 Nir and Langley (2018) is an AEAD, that combines the ChaCha20 stream cipher with the Poly1305 Bernstein (2005) message authentication code. Its usage in IETF protocols is standardized in RFC 8439 Nir and Langley (2018). It has fast software performance, and without hardware acceleration, is usually faster than AES-GCM.

- **XChaCha20Poly1305** is an extended 192-bit nonce variant of the ChaCha20Poly1305 construction, using XChaCha20instead of ChaCha20. XChaCha20Poly1305 has a much greater limit on the number of messages and message size than AES128GCM, but when it does fail (very unlikely) it also leaks key material. When choosing nonce at random, the XChaCha20Poly1305 construction allows for better security than Chacha20ploy1305.

Chapter 3

Approach

In this section, we present a high-level overview of the generic protocol design in Figure 3.3 . The objective is to provide a comprehensive understanding of the protocol's design, while intentionally omitting the specific instantiations involving various KEMs. This overview serves as a foundation upon which the subsequent chapters will delve into the instantiation details and practical implementations of the protocol.

We build our KEM-WireGuard protocol layer by layer starting from replacing the ECDH with KEMs. Each resulted layer will achieve a certain security objectives listed below.

Like in KEMTLS(PDK), the initiator encapsulates the shared secret against the responder's long-term KEM public key ,producing a ciphertext. Since the initiator already got the other side's public key, it can execute the encapsulation before the handshake and send encapsulated ciphertext in the initiation message. We mix this first shared secret ss_S into the KDF and chaining hash calculation. As a consequence, the only entity who can retrieve the plaintext message encrypted under the key derived from ss_S is the responder who has knowledge of ss_S, which means initiator has the assurance that this KDF derived key is implicitly authenticated.

Like the 3-way DH of WireGuard, we assume the handshake is a interactive connection. In static non-interactive DH pattern, each party can perform DH against the other's long-term public key before the connection ever happen. Thus enabling them to directly start the secure connection seemingly to have pre-shared key. However, the only pre-distributed one is the public key instead of the encapsulation which is encapsulated/encrypted each time the initiator run the KEM.Encapsulate(). And the responder needs to decapsulate/decrypt the encapsulation to acquire the shared key inside. That the restrict brought by KEM. So it is impossible to simply "DH" the secret key of initiator and part of responder's public key to derive a shared secret.

3.1 Security Properties

KEM-WireGuard inherits most security properties from WireGuard. Its primary goal is to establish a secure channel between communicating peers. Specifically, a secure channel should guarantee at least the following properties

Secrecy : To protect the secrecy of handshake, key encapsulations using the keypairs (sk_S , pk_S), (sk_e , pk_e) and (sk_C , pk_C), to provide secrecy. Their respective ciphertexts are called ct_S, ct_e, and ct_C. The resulting shared keys are ss_S, ss_e and ss_C. After a successful handshake, only the communication parties *could* derive the session key or shared secret. Meanwhile, the data sent over the channel should only be accessible by the communication parties. Although a single key encapsulation scheme is sufficient to provide secrecy. We combine two different KEMs to achieve optimal performance and security. To protect the secrecy of data at rest (except the encapsulation and authentication tag), we apply the AEAD to all the application data.

Integrity : Both the entities can be aware of any unexpected modification to the data sent over the channel made by third parties.

Authentication : If the the other entity is authenticated, the initiator of authentication request **should only** talk to the **particularly identified** one he ought to talking to Since the integrity and secrecy is assured by the combination of public key encryption and secret key encryption, and the session key derived by public key cryptography is the most important component of the subsequent secret key encryption, we should put most effort into how we apply the public key encryption for authentication. Typically, there are two categories of authentications:

- Implicit authentication: In most cases, key authentication is referred to as "implicit key authentication" because it does not involve explicit validation of the key itself, but rather relies on the security properties of the underlying cryptographic primitives and the secrecy of the key to ensure its authenticity. It is a property of key agreement protocol that only the other intended entity may be able to get the session key. This property can be either unilateral (authenticated for one entity) or mutual (authenticated for both entities).

- To explain the difference between implicit authentication and the other one, the notion "key confirmation" will be introduced. It's a similar property to implicit authentication. It means one entity can be sure that some other entities do get hold

of the session key. It does not require "only identified entities" to have the key, it is just rather a guarantee of "having the key".

- Explicit authentication: It requires both implicit authentication and key confirmation properties. It means the entity being explicitly authenticated not only is the intended one who is able to acquire the session key but also actually have the key. In the WireGuard scenario, it does not explicitly authenticate all the communication data which restricts its security in regard to eCK-PFS model Cremers and Feltz (2012). Therefore , we proposes a more efficient and safer way to leverage the KEM to achieve mutual explicit authentication, in a similar manner to the kemtls and kemtlspdk and achieve both post-quantum authentication and eCK-PFS. The research also instantiates with a variety of post-quantum secure KEMs based on fundamentally different mathematical assumptions.

3.2 Migrate to KEM-based WireGuard

For achieving forward secrecy and explicit authentication, we follow the security model of kemtls. This model captures three levels of forward secrecy for stage keys, while, at the same time,captures the implicit authentication.

3.2.1 Naïve Design

In this design, we simply attempt to replace the ECDH with the KEMs and only achieve very limited security level: None of the party has long-term key pair at first. The initiator starts by generating a fresh KEM key pair and sent the encapsulation to the responder. The responder will decapsulate it lately and acquire the shared secret. After applying the KDF on the shared secret, both party can start communication with data encrypted with the derived key. There is neither guarantee of authentication nor guarantees against active attackers. This handshake pattern shown in Figure 3.1 closely resembles the NNhfs pattern defined by the KEM-based Hybrid Forward Secrecy Noise Framework.

NNkem:
$\rightarrow$ e1
$\leftarrow$ ekem1

Figure 3.1: Noise notation of NNkem

3.2.2 The static-static KEM

In this design, we attempt to build an equivalent to the static to static ECDH-like key agreement since WireGuard has two pair of long-term static key pairs. For this, we use a handshake pattern that similarly resembles the KKhfs pattern defined by the KEM-based Hybrid Forward Secrecy Noise Framework but with the ECDH computation removed. This handshake pattern is shown in Figure 3.2.

KKkem:
$\rightarrow$ s1
$\leftarrow$ s2
...
$\rightarrow$ skem2
$\leftarrow$ skem1

Figure 3.2: Noise notation of KKkem

3.2.3 Forward Secrecy

Forward secrecy level 1 Forward secrecy level 1 guarantees that the passive adversaries at this protocol level has no access to the derive shared secret, even when the adversaries somehow get one party's long-term secret key (In Forward secrecy model, we have to assume not all keys are compromised, otherwise we have nothing to use to defend). However, at this level/stage, the ss_S itself can not provide implicit authentication yet.

So we require another pair of KEM keys **Ephemeral**: (sk_e , pk_e) freshly generated by the initiator and send the pk_e as plaintext to the responder at the initiation message. When the responder receive pk_e , it immediately encapsulate against pk_e and send encapsulation ct_e back. The initiator can the decapsulate against its sk_e and obtain ss_e. After this interaction, both parties perform KDF with the ss_e and ss_S . Any passive adversaries want to reveal the this secret at this phrase will need to get both sk_S and sk_e. Furthermore, since our protocol inherits the **chaining** hash and KDF from WireGuard, the subsequent communication using derived key obtain the forward secrecy level 1.

Forward secrecy level 2 Forward secrecy level 2 guarantees that the passive adversaries at this protocol level has no access to the derive shared secret at forward secrecy level 1 or has no access to either party's long-term secret key at any stage. The second case produces implicit authentication for responder (responder has sk_S tied to pk_S) This is achieved when responder performs KDF with (ss_S , ss_e) .

Full forward Secrecy Full forward Secrecy guarantees that the passive adversaries at this protocol level has no access to the derive shared secret at forward secrecy level 1 or has no access to either party's long-term secret key before the stage finishes. To achieve implicit authentication for the initiator, we require the third KEM key pairs **Static**: (sk_C, pk_C) generated by the initiator and distribute the pk_C as plaintext to the responder at the setup stage. In the response message, the responder can perform encapsulation, and send ct_C to the initiator. When the initiator receive ct_C , it immediately decapsulate against sk_C and obtain ss_C. After this interaction, we achieve forward Secrecy for client. To achieve forward Secrecy for the responder, the responder need to **accept** the key derived with ($ss_S||ss_e||ss_C$). So we use KDF with ($ss_S||ss_e||ss_C$) to derive last four-tuple symmetric keys and lately encrypt application data under these keys. Note that the **accept** here can be just a constant string like "client confirmation" but has to be authenticated by MAC of the final derived key.

Explicit authentication This security objective is obtained when one party has the proof that the other party is explicitly authenticated. It improves upon the implicit authentication a little more by (the one needs to be explicitly authenticated) providing explicit authentication of keys (signature in TLS/MAC in our KEM-WireGuard). This mutual authentication is also achieved when the 4-tuple symmetric keys derived with ($ss_S||ss_e||ss_C$) and both parties uses MAC to authenticate their data with these keys.

Anonymity This security objective guarantees that the passive adversaries can never tie the recorded data back to according initiator. Although in general, the **ephemeral** indicates this key pair should be **erased** after the session finished, it doesn't add great overhead to protocol. So we decide to obtain anonymity at the end. In forward secrecy level 1, we send the ephemeral public key as plaintext in the air which can leak the relation between the application data and the ephemeral public if sk_e compromises. So we require the pk_e encrypted with the ss_S at the first time.

3.3 The Symbolic Proof

Our Tamarin model of KEM-WireGuard is an extension of the Meier et al. (2013) . The original model did not provide lemmas for claimed replay resistance and Denial of Service (DoS) mitigation. Hülsing et al. (2021b) bridged this gap by introduce proofs for these specific properties. Moreover, this model extend the previous PFS to full PFS.

Through the verification, KEM-WireGuard indeed inherits the security propertes from the original one, based upon the premise that long-term KEM schemes should be CCA-

secure Cramer and Shoup (1998).

For KEM-WireGuard, We model this approach in model direction in our code repository.

3.4 Overview of the Design

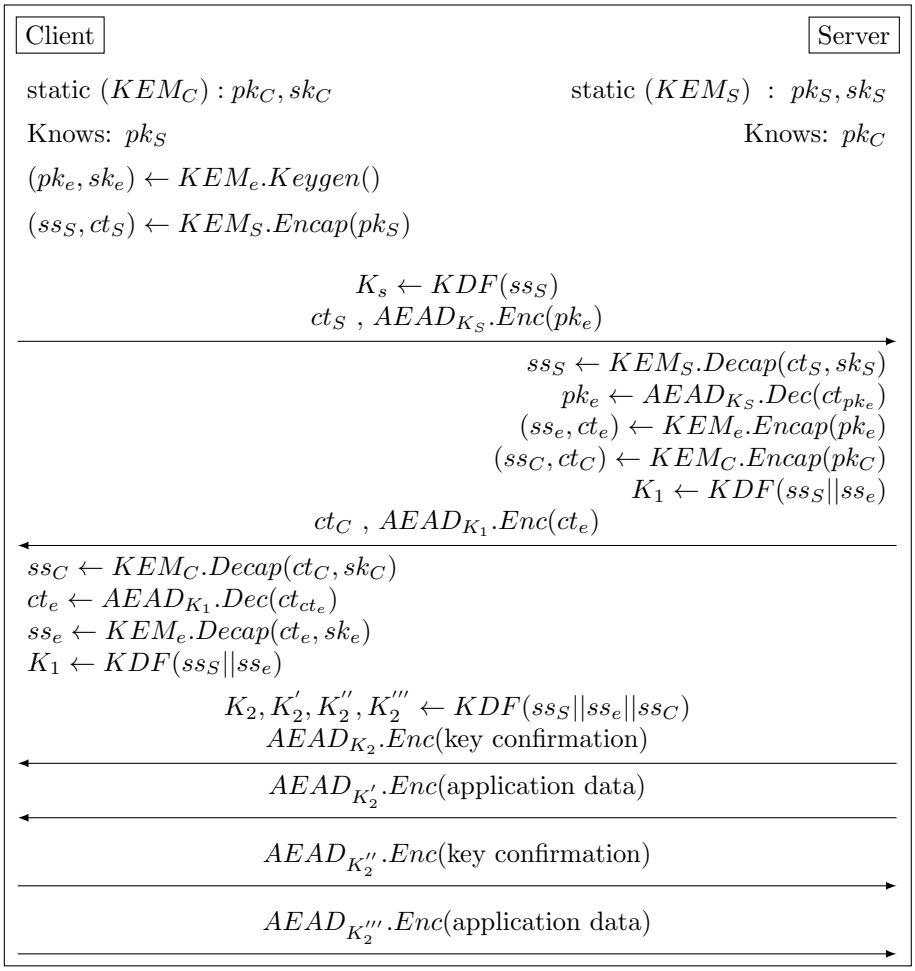

Figure 3.3: Generic KEM-WireGuard design

Chapter 4

Implementation

KEM-WireGuard draws considerable inspiration from KEMTLSPDK, reflecting its fundamental construction. Figure 3.3 provides overview of the generic protocol. The KEM-WireGuard uses two distinct KEMs serving different purposes: KEM_e for ephemeral key agreement and (KEM_S , KEM_C) for secrecy and authentication. It is possible to instantiate generic KEM-WireGuard with a single KEM scheme for both KEM_e and (KEM_S , KEM_C), as we do in our general one, or combine various schemes, as demonstrated lately, to achieve optimal performance and security. For instance, a long-term KEM can utilize an KEM scheme that features a slow key generation but a fast encap/decap and ephemeral KEM can utilize an KEM scheme that features small public key size. Moreover, any of these KEMs could be combined with classical cryptographic scheme such as ECDH, formulating transitional post-quantum constructions.

Because the key distribution is out-of-band, we handle this part in the configuration file. The administers will generate long-term key pairs for each endpoint and distribute them to each endpoint's configuration file manually. During the running time, the KEM-WireGuard will read the long-term public key of the endpoints and its own long-term private key from its configuration file.

4.1 Prototype Implementation

We integrate single KEM: Kyber512 for both ephemeral key exchange and long-term authentication KEMs in the prototype implementation.

4.1.1 Initiation message

The initiator initiates the KEM-WireGuard handshake:

1. Generate its ephemeral KEM public keypair (sk_e , pk_e)

2. Encapsulate against responder's long-term public key pk_S to obtain (ss_S , ct_S)
3. Derive session key K_S with ss_S using KDF
4. Encrypt pk_e with AEAD under key: K_S
5. Send ct_S and encrypted pk_e to the responder

The code provided in listing 4.1 serves as a demonstration of the initiation message structure.

```
type MessageInitiation struct {
Type          uint32
Sender        uint32
Ephemeral     [NoisePublicKeySize+chacha20poly1305.Overhead]byte
//Server Static Encapsulation
CipherTextS   [kyber512.CiphertextSize]byte
Static        [blake2s.Size+chacha20poly1305.Overhead]byte
Timestamp     [tai64n.TimestampSize+chacha20poly1305.Overhead]byte
MAC1          [blake2s.Size128]byte
MAC2          [blake2s.Size128]byte
}
```

Listing 4.1: Packet structure of initiation message

The following Golang code provided in listing 4.2 demonstrates the packet size for an initiation message based on the defined structure.

```
MessageInitiationSize=2*4+kyber512.PublicKeySize
+chacha20poly1305.Overhead+kyber512.CiphertextSize
+blake2s.Size+chacha20poly1305.Overhead
+tai64n.TimestampSize+chacha20poly1305.Overhead
+2*blake2s.Size128
```

Listing 4.2: Packet size (byte) of initiation message

4.1.2 Response message

The responder receive the initiation message and responds:

1. Decapsulate the encapsulation ct_S against its long term private key sk_S to obtain ss_S

2. Derive session key K_S with ss_S using KDF
3. Decrypt the encrypted pk_e with AEAD under key: K_S
4. Encapsulate against initiator's ephemeral public key pk_e to obtain (ss_e , ct_e)
5. Encapsulate against initiator's long-term public key pk_C to obtain (ss_C , ct_C)
6. Derive session key K_1 with ($ss_S||ss_e$) using KDF
7. Send ct_C and encrypted ct_e to the responder
8. Derive 4-tuple of session keys (K_2, K_2', K_2'', K_2''') with ($ss_S||ss_e||ss_C$) using KDF
9. Send key confirmation and application data encrypted with (K_2, K_2') respectively

The code provided in listing 4.3 serves as a demonstration of the initiation message structure.

```
type MessageResponse struct {
Type         uint32
Sender       uint32
Receiver     uint32
//Ephemeral Encapsulation
CipherTextE [kyber512.CiphertextSize]byte
//Client Static Encapsulation
CipherTextC [kyber512.CiphertextSize
+chacha20poly1305.Overhead]byte
MAC1         [blake2s.Size128]byte
MAC2         [blake2s.Size128]byte
}
```

Listing 4.3: Packet structure of response message

The following Golang code provided in listing 4.2 demonstrates the packet size for an initiation message based on the defined structure.

```
MessageResponseSize=3*4+2*kyber512.CiphertextSize
+chacha20poly1305.Overhead+2*blake2s.Size128
```

Listing 4.4: Packet size (byte) of response message

4.1.3 Consume Response message

The initiator receive the response message and performs the following:

1. Decapsulate the encapsulation ct_C against its long term private key sk_C to obtain ss_C
2. Derive session key K_1 with ss_C using KDF
3. Decrypt the encrypted ct_e with AEAD under key: K_1
4. Derive 4-tuple of session keys $(K_2, K_2', K_2'', K_2''')$ with $(ss_S||ss_e||ss_C)$ using KDF
5. Send key confirmation and application data encrypted with (K_2'', K_2''') respectively

Chapter 5

Evaluation

Since the primary focus of interest in our research is achieving the optimal performance while remain the post-quantum security level, this chapter will demonstrate the comprehensive analysis of various KEM schemes in terms of their size attributes and performance metrics, followed by an evaluation of KEM-WireGuard instantiated with Kyber512, Classic McEliece 348864f, Bike-L1, and HQC-128. comparing the prototype implementations to the user-space wireguard-go. According to Approach chapter, modifications were only made to the handshake code. As a result, the primary task will be the evaluating the handshake process, along with the transmission of initiation and response messages.

5.1 Experiment

5.2 Size metrics of KEM primitives

In Table 5.1, we analyses the sizing metrics for various KEM primitives, expressed in terms of the size of their respective public keys, private keys, and encapsulation size. We include the original cryptographic primitive x25519 into the measurement, even though it isn't a KEM. This is because the x25519 is structured in a manner akin to KEM in the original WireGuard design. The metrics are sorted based on the sum of public key size + $3 \times$ encapsulation size). This sorting aims to comprehensively display the overall impact of the KEMs on KEM-WireGuard because the full round handshake of KEM-WireGuard requires three encapsulations and a transmission of a single ephemeral public key.

Primitive	PublicKeySize	PrivateKeySize	Encapsulation	pub+3∗Encap
X25519	32	32	32	128
Kyber512	800	1632	768	3104
BIKE-L1	1541	5223	1573	6260
HQC-128	2249	2289	4481	15692
McEliece348864	261120	6492	96	261408

Table 5.1: Size metrics of the NIST-PQC level I primitives

5.3 Time metrics of KEM primitives

In Table 5.2, we analyses the time metrics for various KEM primitives in terms of the runtime of their respective *Key Generation*, *Encapsulation*, and *Decapsulation*. The number of KEM operations for the full round handshake of KEM-WireGuard at the respective side of initiator and responder is also shown at Table 5.3. So the metrics are sorted based on the orders of magnitude of their overall runtime.

Runtime	Key Generation	Encapsulation	Decapsulation
kyber512	0.02	0.01	0.01
HQC-128	0.03	0.05	0.09
BIKE-L1	0.13	0.03	0.5
mceliece348864	51.88	0.35	19.70

Table 5.2: Time metrics of the NIST-PQC level I primitives (in millisecond)

As we can see from the Table 5.2, the Kyber512 is the best option if the performance is the determining factor while designing the protocol. It has considerably outperformed the other PQ schemes in any algorithm aspect

	Key Generation	Encapsulation	Decapsulation
Initiator	1	1	2
Responder	0	2	1

Table 5.3: Respective ops number of initiator and responder

5.4 Message Sizes

Integrate KEM into the WireGuard introduces significant overheads due to the large public/private key and encapsulation of post quantum KEM. All these data involves in

the transmission of initiation and response messages. For PQ KEM, these data becomes quite overwhelming time lag for network transmissions, ranging from a few hundred bytes to several hundred Kilobytes in some extreme cases. Consequently, there is a possibility of packets exceeding the Ethernet's maximum transmission unit (MTU) of 1500 Bytes and cause the re-transmission of these packets which is not what we want for a 2-round trip network protocol.

Given that both WireGuard and our protocol relies on UDP, segmentation isn't provided at this layer. When large packet exceeding the Ethernet's MTU is transmitted over UDP, fragmentation could occur at the IP layer. Meanwhile, it is even worse that, as in practice, the IP layer permits routers to silently drop oversized packets. To circumvent packet drop or fragmentation, a common way is to split before sending packet into IP layer and split in advance. That is not the best practice for a protocol running on top of UDP because any packet loss will lead to the failure of handshake. The failure correlates strongly with the output of KEM primitives. Hence, the choice of KEM primitives with compact **public key sizes** becomes crucial to avoid unwanted overhead, such as extended transmission times. Consequently, KEM primitives characterized by excessively large encapsulation are excluded from our decision list. For instance, this pertains to all code-based cryptosystems, which necessitate multiple Kilobytes for a single public key. Considering our pre-distributed case, all the long-term public keys are distributed at the setup time leave their transmission time out of the equation. So KEM scheme like Classic McEliece can also be evaluated.

5.5 Results

In Table 5.4, we illustrates the final benchmark results of various instantiations, in terms of server-side runtime, client-side runtime, and the total runtime of respective handshake part. These data provides an extensive overview of the outcomes derived from different KEM instantiations. These metrics offer valuable insights into the performance dynamics of each KEM-WireGuard instantiation that use the selected KEM primitives. It also helps us to choose from diverse KEM scheme combinations for integration into KEM-WireGuard.

The server-side runtime refers to the time taken by the server to execute the necessary KEM operations during the handshake process. It covers ephemeral key generation, static encapsulation/decapsulation, and other related computations on the server side.

On the other hand, the client-side runtime refers to the time taken by the server to execute the necessary KEM operations during its handshake process. This includes encapsulation/decapsulation, and other relevant computations on the client side.

The total runtime represents the total time taken for both the server-side and client-side handshake. It offers a holistic view of the overall efficiency of various KEM-WireGuard instantiations.

By analyzing these runtime metrics across the different KEM primitives: Kyber512, Classic McEliece348864, Bike-L1, and HQC-128, we can conclude the comparative performance of each instantiation in terms of cryptographic operations and overall efficiency. This evaluation helps us decide the KEM choice regarding the most suitable KEM-WireGuard instantiation for optimal performance, taking into account factors such as runtime performance, security requirements, and bandwidth usage.

Protocol	Initiation size (bytes)	Response size (bytes)	Time in microseconds		
			Server	Client	Total
WireGuard (original)	148	92	0.36	0.47	1.37
KEM-WireGuard (kyber512 only/time-optimal)	1700	1596	0.21	0.34	0.57
KEM-WireGuard (HQC-128)	6862	9022	0.49	0.61	0.84
KEM-WireGuard (BIKE-L1)	3246	3206	1.46	3.65	4.60
KEM-WireGuard (mcliece348864f only)	261348	252	20.82	91.47	133.36
KEM-WireGuard (M&K/size-optimal)	1028	252	20.40	20.95	41.47

Table 5.4: Runtime for the NIST-PQC level I primitives

The Cloudflare CIRCL library does not provide the primitive of BIKE-L1, HQC-128 and Classic McEliece. The Classic McEliece is adapted from forked version of CIRCL. The other two primitive are adapted from liboqs-go. It offers a Go wrapper for the Open Quantum Safe's liboqs library, which is a C library for quantum-resistant cryptographic algorithms. For memory safety reason, we prefer the native go implementation of cryptography primitives to Go wrapper of another C library.

5.6 Benchmark

In this section, we illustrate how we perform the evaluation the KEM-WireGuard handshake which mainly focus on benchmarking performance metrics to assess the efficiency and effectiveness of the protocols in real-world scenarios. In 5.6, we show the code snippet about benchmarking the total time of the whole handshake including both side.

```
func BenchmarkHandshake(b *testing.B) {
for i := 0; i < b.N; i++ {
        b.StopTimer()
        dev1 := randBDevice(b)
        dev2 := randBDevice(b)

        peer1, _ := dev2.NewPeer(dev1.staticIdentity.privateKey.publicKey())
        peer2, _ := dev1.NewPeer(dev2.staticIdentity.privateKey.publicKey())
        peer1.Start()
        peer2.Start()
        b.StartTimer()

        msg1, _ := dev1.CreateMessageInitiation(peer2)
        packet := make([]byte, 0, MessageInitiationSize)
        writer := bytes.NewBuffer(packet)
        binary.Write(writer, binary.LittleEndian, msg1)
        dev2.ConsumeMessageInitiation(msg1)

        msg2, _ := dev2.CreateMessageResponse(peer1)

        dev1.ConsumeMessageResponse(msg2)
        b.StopTimer()
        dev1.Close()
        dev2.Close()
        b.StartTimer()
}
}
```

In 5.6, we show the code snippet about benchmarking the run time of the handshake starting from the the time when the server receive the initiation message end with the time when the server creates the response message.

```
func BenchmarkHandshakeServer(b *testing.B) {
for i := 0; i < b.N; i++ {
        b.StopTimer()
        dev1 := randBDevice(b)
        dev2 := randBDevice(b)

        peer1, _ := dev2.NewPeer(dev1.staticIdentity.privateKey.publicKey())
        peer2, _ := dev1.NewPeer(dev2.staticIdentity.privateKey.publicKey())
        peer1.Start()
        peer2.Start()

        msg1, _ := dev1.CreateMessageInitiation(peer2)
```

```
13
14          packet := make([]byte, 0, MessageInitiationSize)
15          writer := bytes.NewBuffer(packet)
16          binary.Write(writer, binary.LittleEndian, msg1)
17          b.StartTimer()
18          dev2.ConsumeMessageInitiation(msg1)
19          msg2, _ := dev2.CreateMessageResponse(peer1)
20          b.StopTimer()
21          dev1.ConsumeMessageResponse(msg2)
22          dev1.Close()
23          dev2.Close()
24          b.StartTimer()
25  }
26  }
```

As shown in the code 5.6, this benchmark records from the time when the client sends the initiation message and end with the time when the client consumes the response message.

```
1   func BenchmarkHandshakeClient(b *testing.B) {
2   for i := 0; i < b.N; i++ {
3           b.StopTimer()
4           dev1 := randBDevice(b)
5           dev2 := randBDevice(b)
6           peer1, _ := dev2.NewPeer(dev1.staticIdentity.privateKey.publicKey())
7           peer2, _ := dev1.NewPeer(dev2.staticIdentity.privateKey.publicKey())
8           peer1.Start()
9           peer2.Start()
10          b.StartTimer()
11          msg1, _ := dev1.CreateMessageInitiation(peer2)
12          b.StopTimer()
13
14          packet := make([]byte, 0, MessageInitiationSize)
15          writer := bytes.NewBuffer(packet)
16          binary.Write(writer, binary.LittleEndian, msg1)
17
18          dev2.ConsumeMessageInitiation(msg1)
19          msg2, _ := dev2.CreateMessageResponse(peer1)
20          b.StartTimer()
21          dev1.ConsumeMessageResponse(msg2)
22          b.StopTimer()
23
24          dev1.Close()
25          dev2.Close()
```

```
26 }
27 }
```

As shown in the table, single KEM/Kyber512-only is the time optimal version among all the instantiations. It is even faster compared to the original wireguard-go implementation in all the runtime metrics. The only drawback comes from its Initiation message: it exceed the MTU size thus sufferers from fragmentation. On the other hand, it features a relatively small bandwidth usage compared to other PQ KEM schemes. The M&K stands for combination of **M**cliece348864 as the long-term KEM scheme and **K**yber512 the ephemeral key exchange KEM since the Classic McEliece offer a unparalleled small encapsulation This is at least an order of magnitude lower than any other KEM schemes. So this 5.1 is our final result of the KEM-WireGuard design

5.7 Summary

Our study has demonstrated the feasibility of incorporating Post-Quantum KEM schemes in WireGuard, particularly when the long-term public keys are pre-distributed in advance. Notably, WireGuard serves as an ideal starting point for such experiment

However, current drawback lies in the inevitable performance overhead associated with underlining KEMs. It open ups to new avenues to improve the performance and efficiency of the KEM implementations by leveraging CPU instructions such as Advanced Vector Extensions 2 (AVX2). By harnessing these specialized instructions, which are designed to handle parallel calculation efficiently, we can significantly accelerate cryptographic operations and reduce the overall execution time.

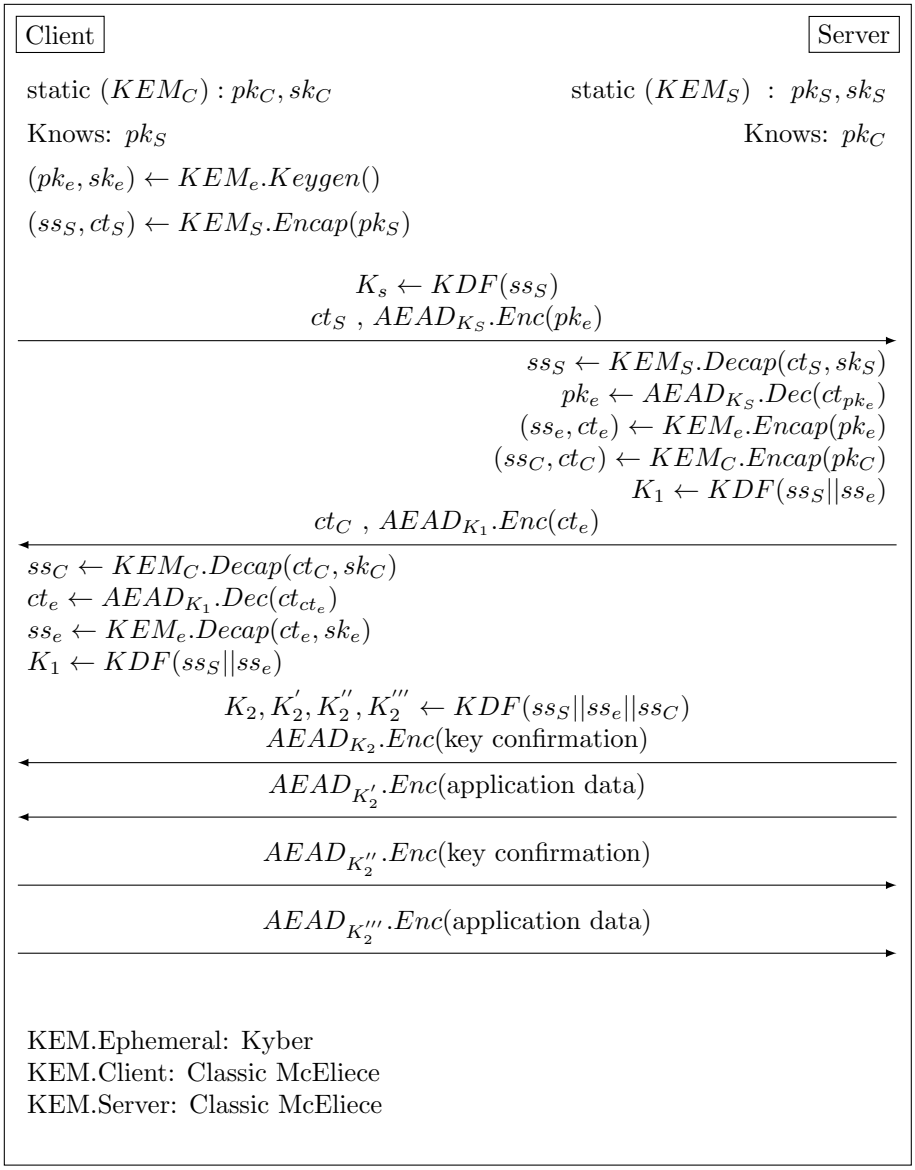

Figure 5.1: On-wire KEM-WireGuard design

Chapter 6

Conclusions & Future Work

In this paper, we presented the evaluation of KEM-WireGuard with pre-distributed long-term public KEM keys. The results show that it is possible that KEM may be the more efficient option for the post-quantum future of TLS.

Firstly, We build our KEM-WireGuard protocol layer by layer starting from replacing the ECDH with KEMs. By aligning each layer with KEM-WireGuard's security objectives, we achieved a system that not only meet the common needs of a secure tunnel but also prevent potential vulnerabilities and provide extra security property. Our design process sought to strike a balance between security, efficiency, and practicality, ensuring that the resulting protocol would be at least as the same security level as any other instantiations.

That generic design laid the foundation for the following experimentation and analysis. Building on this, we instantiated KEM-WireGuard with a diverse range of PQ KEM schemes, thereby extending our evaluation scope to cover the final round submission of NIST PQC. Through rigorous evaluation, we tested and compared these instantiations, not only among themselves but also against the original user-space implementation of wireguard-go. This evaluation provided us with well qualified insights into the performance, efficiency, and security metrics of our various instantiations.

By systematically assessing these instantiations, we have obtained valuable insights into their strengths and limitations. This evaluation has enabled us to draw meaningful comparisons, thereby identifying the most suitable instantiation that aligns with our security and outperforms others. It is important to note that wireguard-go, as in benchmark, played a pivotal role in setting a baseline against our instantiations.

6.1 Future Work

To optimize performance, future work could be leveraging advanced CPU instructions, such as AVX2 and AESNI. This optimization strategy has the potential to significantly speedup the execution of cryptographic primitives, thereby improving the overall efficiency of the protocol. We cloud also explore the implementation of KEM-WireGuard within the kernel space. By shifting the code base to the kernel level, it becomes possible to harness the benefits of tighter integration with the operating system, possibly resulting in reduced latency and in-memory copy for secure communication. Further improvements can be made by providing an abstract layer to facilitate versatility of KEM schemes selection. This abstract layer serves as a interface that allows users and developers to seamlessly switch between various KEM schemes, tailed to their specific security requirements and technological needs.

Bibliography

Aguilar Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Persichetti, E., Zémor, G., Bos, J., Dion, A., Lacan, J., Robert, J.-M., and Veron, P. (2022). HQC. Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

Albrecht, M. R., Bernstein, D. J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K. G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C. J., Tomlinson, M., and Wang, W. (2020). Classic McEliece. Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

Albrecht, M. R., Bernstein, D. J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K. G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C. J., Tomlinson, M., and Wang, W. (2022). Classic McEliece. Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions.

American National Standards Institute, Inc. (2005). ANSI X9.62 public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA).

Appelbaum, J., Martindale, C., and Wu, P. (2019). Tiny WireGuard tweak. pages 3–20.

Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Gueron, S., Guneysu, T., Aguilar Melchor, C., Misoczki, R., Persichetti, E., Sendrier, N., Tillich, J.-P., Zémor, G., Vasseur, V., Ghosh, S., and Richter-Brokmann, J. (2022). BIKE. Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions.

Aumasson, J.-P., Neves, S., Wilcox-O'Hearn, Z., and Winnerlein, C. (2013). BLAKE2: Simpler, smaller, fast as MD5. pages 119–135.

Bellare, M., Canetti, R., and Krawczyk, H. (1996). Keying hash functions for message authentication. pages 1–15.

Bernstein, D. J. (2005). The poly1305-aes message-authentication code. In Gilbert, H. and Handschuh, H., editors, *Fast Software Encryption*, pages 32–49, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bernstein, D. J. (2006). Curve25519: New Diffie-Hellman speed records. pages 207–228.

Bernstein, D. J. (2008). *The Salsa20 Family of Stream Ciphers*, pages 84–97. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., and Stebila, D. (2019a). Hybrid key encapsulation mechanisms and authenticated key exchange. pages 206–226.

Bindel, N., Hamburg, M., Hövelmanns, K., Hülsing, A., and Persichetti, E. (2019b). Tighter proofs of CCA security in the quantum random oracle model. Cryptology ePrint Archive, Report 2019/590. https://eprint.iacr.org/2019/590.

Black, J. (2005). *Authenticated encryption*, pages 11–21. Springer US, Boston, MA.

Blake-Wilson, S., Johnson, D., and Menezes, A. (1997). Key agreement protocols and their security analysis. pages 30–45.

Boyd, C. and Mathuria, A. (2003). *Protocols for Authentication and Key Establishment*.

Cramer, R. and Shoup, V. (1998). A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. pages 13–25.

Cramer, R. and Shoup, V. (2001). Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. Cryptology ePrint Archive, Report 2001/108. https://eprint.iacr.org/2001/108.

Cremers, C. J. F. and Feltz, M. (2012). Beyond eCK: Perfect forward secrecy under actor compromise and ephemeral-key reveal. pages 734–751.

Crockett, E., Paquin, C., and Stebila, D. (2019). Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Cryptology ePrint Archive, Report 2019/858. https://eprint.iacr.org/2019/858.

Diffie, W., van Oorschot, P. C., and Wiener, M. J. (1992). Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125.

Donenfeld, J. A. (2017). WireGuard: Next generation kernel network tunnel.

Fujisaki, E. and Okamoto, T. (1999). How to enhance the security of public-key encryption at minimum cost. pages 53–68.

Galbraith, S. D. and Vercauteren, F. (2017). Computational problems in supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2017/774. https://eprint.iacr.org/2017/774.

Goldreich, O., Goldwasser, S., and Micali, S. (1986). How to construct random functions. *Journal of the ACM*, 33(4):792–807.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. pages 212–219.

Günther, C. G. (1990). An identity-based key-exchange protocol. pages 29–37.

Ho, S., Protzenko, J., Bichhawat, A., and Bhargavan, K. (2022). Noise: A library of verified high-performance secure channel protocol implementations. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 107–124.

Hövelmanns, K., Kiltz, E., Schäge, S., and Unruh, D. (2018). Generic authenticated key exchange in the quantum random oracle model. Cryptology ePrint Archive, Report 2018/928. https://eprint.iacr.org/2018/928.

Hülsing, A., Ning, K.-C., Schwabe, P., Weber, F., and Zimmermann, P. R. (2021a). Post-quantum WireGuard. pages 304–321.

Hülsing, A., Ning, K.-C., Schwabe, P., Weber, F., and Zimmermann, P. R. (2021b). Post-quantum wireguard. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 304–321. IEEE.

Just, M. and Vaudenay, S. (1996). Authenticated multi-party key agreement. pages 36–49.

Krawczyk, H. (2010a). Cryptographic extraction and key derivation: The HKDF scheme. Cryptology ePrint Archive, Report 2010/264. https://eprint.iacr.org/2010/264.

Krawczyk, H. (2010b). Cryptographic extraction and key derivation: The HKDF scheme. pages 631–648.

Lyubashevsky, V., Peikert, C., and Regev, O. (2012). On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230. https://eprint.iacr.org/2012/230.

McEliece, R. J. (1978). A public-key cryptosystem based on algebraic coding theory. The deep space network progress report 42-44, Jet Propulsion Laboratory, California Institute of Technology. https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF.

Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The tamarin prover for the symbolic analysis of security protocols. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25*, pages 696–701. Springer.

Miller, V. S. (1986). Use of elliptic curves in cryptography. pages 417–426.

Moriya, T., Onuki, H., and Takagi, T. (2020). How to construct CSIDH on Edwards curves. pages 512–537.

Nir, Y. and Langley, A. (2018). ChaCha20 and Poly1305 for IETF Protocols. RFC 8439.

Regev, O. (2005). On lattices, learning with errors, random linear codes, and cryptography. pages 84–93.

Rivest, R. L., Shamir, A., and Adleman, L. M. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126.

Schanck, J. M., Hulsing, A., Rijneveld, J., and Schwabe, P. (2017). NTRU-HRSS-KEM. Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

Schwabe, P., Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Seiler, G., Stehlé, D., and Ding, J. (2022). CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

Schwabe, P., Stebila, D., and Wiggers, T. (2020). Post-quantum TLS without handshake signatures. pages 1461–1480.

Schwabe, P., Stebila, D., and Wiggers, T. (2021). More efficient post-quantum KEMTLS with pre-distributed public keys. Cryptology ePrint Archive, Report 2021/779. https://eprint.iacr.org/2021/779.

Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. pages 124–134.

Vasseur, V. (2021). QC-MDPC codes DFR and the IND-CCA security of BIKE. Cryptology ePrint Archive, Report 2021/1458. https://eprint.iacr.org/2021/1458.

Vaudenay, S. (2005). Secure communications over insecure channels based on short authenticated strings. pages 309–326.

Appendix

.1 AI declaration

In the process of preparing thesis, The AI tools were used for covering relevant material as much as possible in case of mine ignorance or lack of knowledge.Neither the code nor any analysis involes the use of AI.