Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Post-Quantum Cryptography in WireGuard VPN

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Quentin M. Kniep
geboren am:
geboren in:

Gutachter/innen: Prof. Dr. Jens-Peter Redlich
Prof. Dr. Björn Scheuermann

eingereicht am: ..... verteidigt am: .....

Abstract

WireGuard is a new and promising VPN software. It relies on cryptographic primitives which are not post-quantum safe. This critically undermines the promise of forward secrecy because it makes all traffic vulnerable to future attacks with quantum computers. This thesis considers ways of modifying current WireGuard implementations. Three increments of modification are proposed, giving different levels of security against quantum adversaries. Performance impacts of these are shown to be moderate.

Contents

Acronyms

AEAD authenticated encryption with associated data

DH Diffie-Hellman key exchange

DLP discrete logarithm problem

DoS denial-of-service

ECDH elliptic-curve DH

KDF key derivation function

KEM key encapsulation mechanism

MAC message authentication code

MITM man-in-the-middle

MTU maximum transmission unit

PFS perfect forward secrecy

PQ post-quantum

PRF pseudo-random function

PSK pre-shared key

VPN virtual private network

WG WireGuard

1. Introduction

In a few years' time, most current VPN traffic may be decrypted.

Virtual private network (VPN) software provides a security layer above the IP layer and puts the VPN server in between you and your destination. It is used, for example, for accessing important computer infrastructure from an insecure network. In the **tunnel**, which is the connection between you and the VPN server, everything can be secured including your identity, the actual data, and its destination. If the VPN server is on the same trusted network as your destination, this is enough to be safe.

There is a wide array of VPN software available. Well-known implementations are OpenVPN, IPsec, and WireGuard, where the currently developmental WireGuard is the focus of this thesis.

1.1. Motivation

Post-Quantum Cryptography

We need to replace many of our currently used cryptographic methods with so called post-quantum (PQ) cryptography. The reason lies in developments regarding quantum computers. Most significant is Shor's algorithm, which can be used to efficiently solve both the prime factorization problem and the discrete logarithm problem (DLP). [32] These two problems are the mathematical foundation of many current cryptographic primitives, especially the extremely prevalent RSA asymmetric cipher and Diffie-Hellman key exchange (DH). Anyone who can solve these could attack almost any key agreement protocol currently in use. PQ cryptography then only means that the cryptographic primitives do not rely on these mathematical problems being hard to solve. Instead, these primitives base their security on problems for which no efficient quantum algorithm is known.

Then again, the largest quantum computer currently has 72 qubits. [21] This is a long way to go, until quantum computers can break any significant key length. Still, it is important to take preparations now, to protect the traffic currently being sent from

future attackers. Also, we do not know how long it will actually take until quantum computers are capable to break current security measures. “It could be decades, but nobody can say for sure.” John Preskill said in an LA Times interview. [25] Furthermore, from experience it is known that it takes a lot of time to change standards in security. Therefore, standards need to be established well before it becomes crucial to use them.

Current methods of perfect forward secrecy (PFS), which is defense against the threat of an attacker who collects and stores all traffic until they are able to decrypt it, are based on classical cryptography. Any messages sent today under such schemes are thus vulnerable to being recorded and later decrypted with a quantum computer.

Authentication methods in current key exchange schemes are based on classical asymmetric cryptographic primitives as well. As soon as we have adversaries with strong quantum computers, they are thus in a position where they can actively man-in-the-middle (MITM) any key exchange, which relies on those classical authentication methods. The threat is real but much less urgent than forward secrecy because a MITM attack is not possible retroactively. So, until the first quantum computer which can break current key lengths is actually built, relying on classical cryptography for authentication is enough.

WireGuard

WireGuard (WG) is a relatively new VPN software, built with the intention to focus on being fast while maintaining a small code base. Having less code is generally considered a good quality for security software because it makes code review easier. One way WireGuard achieves such a small code base is to be very cryptographically opinionated, i.e. it only supports a few state-of-the-art cryptographic primitives. The primitives have also been selected to be conservative choices in terms of security, while still being on the faster side.

Focusing on WireGuard over more established VPN software such as OpenVPN or IPsec might seem questionable. Especially regarding security critical software, it is reasonable to use tools which have reached a certain level of maturity. However, especially in this case it is less about making a good choice for now, and more about deciding what will be a good choice in the near future. WireGuard looks extremely promising, because of its great performance. It may well become one of the top contenders in the VPN software space, once it underwent a little more scrutiny.

Further details about WG and how it works can be found in the WireGuard section of chapter 2.

1.2. Goals

Main goal of this thesis is extending WireGuard to support PQ cryptography for PFS, identity hiding, and possibly even security against active attacks. In the available version of WG there is only the ability to use a pre-shared key (PSK). This key needs to be calculated separately, and can be the result of some PQ key exchange protocol. Only does this not give any of the security properties listed above against quantum computers. This feature merely provides basic encryption on the tunnel. Once the PSK leaks to an attacker with a quantum computer, they could with reasonable effort decrypt all traffic ecnrypted under session keys derived from that PSK. In order to resolve this problem, PQ key agreements will be added directly into the WireGuard protocol.

Another goal that becomes relevant considering how new and unproven PQ cryptography is, is that the security should not rely solely on PQ methods.

Secondary goals are good performance and usability, which should be maintained as well as possible, while still achieving the primary goals.

1.3. Structure

The rest of the thesis will be structured as follows: In chapter 2, there is a review of the attempts at PQ VPN software already available. It also introduces algorithms, definitions, protocols, and cryptographic primitives. Chapter 3 gives an overview of the proposed changes to WireGuard. After that, in chapter 4 there will be a brief discussion of implementation decisions. Then in chapter 5, there will be a critical analysis on whether the solution discussed in the two previous chapters solves the goals set in the beginning. We will especially look at the viability when compared to WireGuard without PQ cryptography, focusing mainly on metrics like average time to perform the handshake. Finally, the last chapter will summarize the findings, and give an outlook on what still can be done.

2. Fundamentals and Related Work

2.1. Quantum Algorithms

Shor

The problem with the current state of cryptography is that, in 1996 already, Shor devised a quantum computer algorithm [32], which can be used to solve the prime factorization problem and the DLP efficiently, i.e. in polynomial time. More specifically, it has a time complexity of $\mathcal{O}((\log N)^3)$, where N is the number to be factored or the prime defining the field for the discrete logarithm.

Once large quantum computers exist, it could be used to launch attacks with polynomial cost against RSA, ECDSA, DH, elliptic-curve DH (ECDH), ElGamal, and so on. Generally, against all cryptographic primitives which can be reduced to either prime factorization or the DLP. Therefore, it breaks today's standards in asymmetric cryptography and key exchange.

Grover

There is also an algorithm targeting symmetric cryptography and hash functions. Also in 1996, Grover developed a quantum computer algorithm [17], which can find the input for an arbitrary function giving a specific output in only $\mathcal{O}(\sqrt{N})$ calls to that function, where N is the size of the function's domain. This is a quadratic improvement over the classical case, for which the worst-case runtime trivially is $\mathcal{O}(N)$.

Brassard et al. built upon Grover's algorithm to give a quantum algorithm for finding hash collisions. [10] Their results improved the theoretical minimum complexity to $\mathcal{O}(2^{N/3})$, from the previous $\mathcal{O}(2^{N/2})$ obtained through the birthday paradox.

Although not breaking classical methods for symmetric cryptography and hashing, a reduction in the attack complexity of this magnitude would force developers to adapt key lengths. At first glance, doubling of symmetric encryption key lengths, and 50% bigger state for collision-resistant hash functions, might therefore seem warranted.

On the other hand, there are doubts whether the theoretically better complexity translates into practically improved attack capabilities. Bernstein for example offers reasonable doubt to the claim, that the quantum algorithms are better than classical methods for finding hash collisions. [6] His argument is, that the better parallelizability of Pollard's rho method [28], as presented by Oorschot and Wiener [26], makes it more cost-efficient than using quantum computers. Meaning, the best collision attack on an N bit hash function still has cost $\mathcal{O}(2^{N/2})$ in practice. To get the speedup of Grover's algorithm, long calculations have to run in serial execution. There is a proof by Zalka [33], showing that running it on M machines gives a maximum $\sqrt{M}$ speedup. And in all general security models we assume attackers to run their algorithms highly parallelized. Therefore, the actual impact of Grover's results is questionable.

In summary, increasing key lengths may or may not be necessary, but at least not nearly as drastically as assumed at first glance.

2.2. Previous Systems

There are already attempts at bringing PQ cryptography into VPN software. The projects that exist so far have some significant disadvantages, when compared to the possibilities of combining WG with a complete PQ handshake. In the following section two of the most prominent projects will be discussed. One of which only uses the static-PSK method that is supported by WireGuard directly. Thus, it does not provide PFS or identity hiding against quantum computers. The other project uses the more bloated OpenVPN, as opposed to WireGuard. Therefore, it has all the disadvantages in speed and code size that WG tries to improve upon.

In contrast, this thesis proposes a way of fixing both these disadvantages, by combining the WireGuard software with a real PQ key exchange. It should thus provide all the advantages of WireGuard over, for example, OpenVPN, as well as more complete PQ security than the PSK feature in WG.

Mullvad

Mullvad is one of the first VPN providers to try to provide ready-to-use PQ cryptography using WireGuard. They make use of the pre-shared key (PSK) feature that WireGuard offers. First performing a PQ key exchange, the output of it is then used as the PSK for the WG handshake. In there, it is simply used as additional key material when deriving the keys. This solution will never be able to provide PFS this way because

the PSK is static.

Any time in the future, an attacker with a quantum computer, who learned the PSK at some point, can break any key exchange that used it. This attack is passive, i.e. it even goes for previously recorded exchanges. They need a quantum computer to break the discrete logarithm, and therefore classic PFS. Using Shor's algorithm on their quantum computer to solve the DLP, they can derive the private ECDH key of either party from the public one. With the private ECDH key and the PSK, they have everything one of the parties had during the exchange. Therefore, they can derive the encryption keys in the same manner as the legitimate party did. The attacker has thus successfully broken the key exchange only by learning the static PSK.

Conclusively, Mullvad's approach at PQ does not provide PFS against a quantum adversary. In other words, all traffic sent over such a tunnel, is only secure against a quantum adversary for as long as the static-PSK stays safe. Furthermore, both parties need to have the PSK (by definition), and neither may lose it.

Microsoft

Microsoft goes another route and uses the more dated OpenVPN as foundation. This decision brings with it all the advantages and disadvantages of OpenVPN compared to WireGuard. Namely, they get the advantage that the code base they fork from OpenVPN is thoroughly tested and analyzed, whereas WireGuard is rather new on a security software timescale. This advantage is rather short-lived though as it shrinks the more WG is tested. Especially since WG is expected to be more easily auditable, because of its way smaller code base. On the other hand they also inherit the large code base and, compared to WG, higher ping and lower data throughput.

Microsoft's solution uses the Open Quantum Safe project's fork of OpenSSL, which implements PQ cryptographic primitives in the TLS handshake. According to TLS specification though it does only use ECDH for ephemeral keys. Therefore, there is no perfect forward secrecy against quantum adversaries. Fundamentally the same attack that was described for Mullvad's approach applies here. Identity hiding for the client's public key may be achieved though.

2.3. Noise Protocol Framework

Noise [27] is a suite of cryptographic key exchange protocols. Each of the protocols provide slightly different properties, especially regarding authentication and forward secrecy. All of them are based on DH/ECDH key agreements. Noise also defines a way of writing these protocols down concisely. To give an example, this is the way IKpsk2, the protocol WG's handshake is based on, is written in the notation established in [27]:

```
<- s
...
-> e, es, s, ss
<- e, ee, se, psk
```

The initiator of the handshake is on the left-hand side, the responder on the right-hand side. Accordingly, the arrow directions indicate the direction of communication. Single letters indicate public keys being sent, 's' stands for a static public key, 'e' for an ephemeral public key. Thus, the first line indicates: Before any of the actual handshake takes place, the initiator already knows the responder's static public key. In practice, this usually happens through user setup, as with SSH keys. Two letters represent a DH calculation taking place, the first letter indicating which of the initiator's key pairs to use, and the second letter indicating the same for the responder. These have to happen on both sides of the exchange because it is implied that the shared secret is added into a chaining key.

The **chaining key** is a fundamental concept of the Noise family of key exchanges. During the entire key exchange, both peers each maintain a continuous hash value and a chaining key. Any data transmitted or received in a packet is added into the hash value. After any single shared secret is established, it is added as key material into the chaining key through a key derivation function (KDF). In Noise protocols this always happens through a DH computation, unless it is the PSK that is used at some point of the protocol. Specifically, the new chaining key is the result of the KDF called on the shared secret, using the old chaining key as the key. Any time an outgoing message needs to be encrypted or an incoming message decrypted, a KDF with double the output length is used instead. One part is then used as the new chaining key, and the other part is the key for encryption/decryption. In the end, the chaining key is used for deriving two symmetric keys, one for each direction of communication.

2.4. WireGuard

WireGuard is a VPN software that tries to become the new standard. Jason A. Donenfeld, the main developer, criticizes current standards, especially OpenVPN and IPsec. One main point of criticism is bloatedness, in terms of code size as well as the number of options and cryptographic primitives supported. This leads to worse auditability and more room for weak security through wrong configuration, Donenfeld argues. Other areas where he tries to improve upon current solutions are ease of use and performance, e.g. ping and throughput.

Authenticated Encryption with Associated Data

Authenticated encryption with associated data (AEAD) is a method that performs symmetric encryption and authentication. It is either a dedicated construction for authenticated encryption or a generic one, from a combination of block cipher and message authentication code (MAC) function. The Associated data is a MAC value that proofs authenticity and binds the ciphertext to a context, for example through a nonce.

All data sent over a WireGuard tunnel is first put through AEAD using the session encryption key. Also, AEAD is already used during the handshake protocol.

Handshake Protocol

WireGuard's handshake resembles the IKpsk2 handshake, defined by the Noise Protocol Framework. [15] For legibility, updating of the continuous hashes and the chaining keys have been omitted from the protocol diagrams. Not further mentioned either will be two MAC values, which are used for preventing denial-of-service (DoS) attacks, and a message header, which are all part of the initiation and response messages.

Before the key exchange starts, we assume both parties have generated static key pairs and the static public key of the responder is known to the initiator. Before the first message is sent, the initiator generates an ephemeral key pair, i.e. matching private and public ECDH keys. They also calculate the first two ECDH secrets using both their own private keys and the known public key of the responder.

WireGuard's first key exchange message, the **initiation message**, then contains the initiator's ephemeral public key in plaintext, the initiator's public static key under AEAD with the key resulting from the first ECDH secret, and a timestamp in TAI64N [4] format under AEAD with the key resulting from both ECDH secrets. This

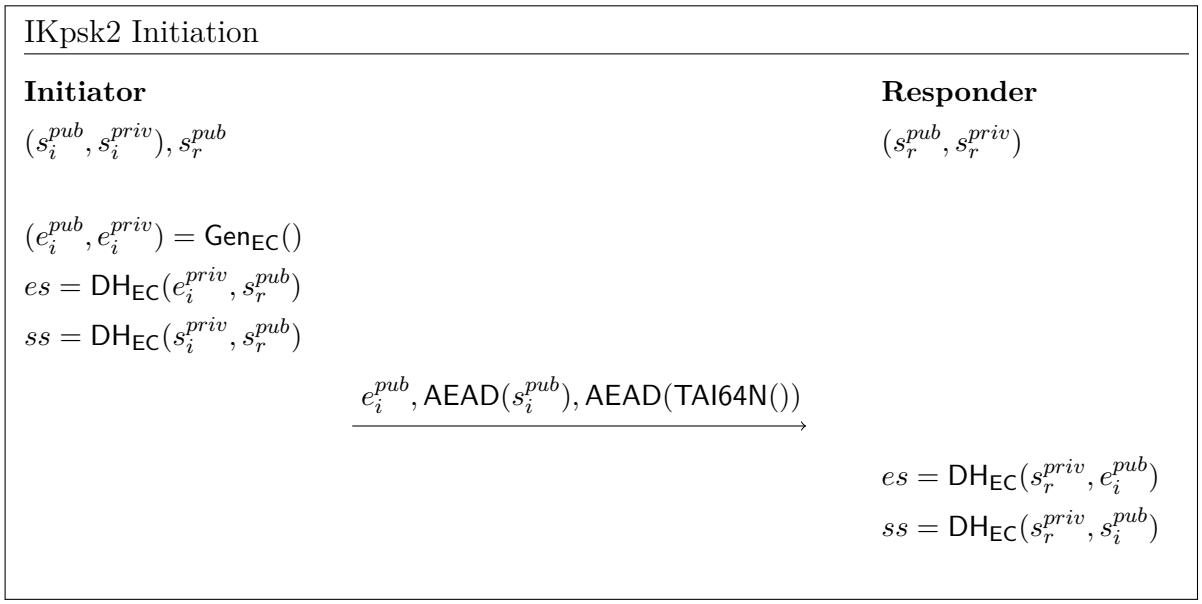

Figure 2.1.: Simplified diagram of the initiation step of WireGuard's IKpsk2 protocol.

provides some identity hiding for the initiator, because you at least need either the initiator's ephemeral private key or the responder's static private key. The responder can, upon receiving the initiation message, take the analogous steps to calculate the two ECDH secrets as well, and reach the same state as the initiator. The whole initiation, everything that happens immediately before sending and after receiving the message, is shown in Figure 2.1.

After the first step of the key exchange, both parties know each others public static keys and the responder also knows the initiators ephemeral public key. The responder then generate their own ephemeral key pair. They continue by calculating the two ECDH secrets that result from combining their ephemeral private key with both of the initiator's public keys.

WireGuard's second key exchange message, the **response message**, contains (most importantly) the responder's public ephemeral key in plaintext, and also the empty string under AEAD with a key resulting from the final chaining key. The empty string under AEAD is simply the associated data, i.e. a MAC value. As in the initiation step, upon receiving the message the initiator can take steps analogous to those of the responder to finally reach the same state. The whole response, everything that happens immediately before sending and after receiving the message, is shown in Figure 2.2.

As specified by Noise, the final chaining key is used for deriving two symmetric session keys. One for the messages going from initiator to responder, the other for the

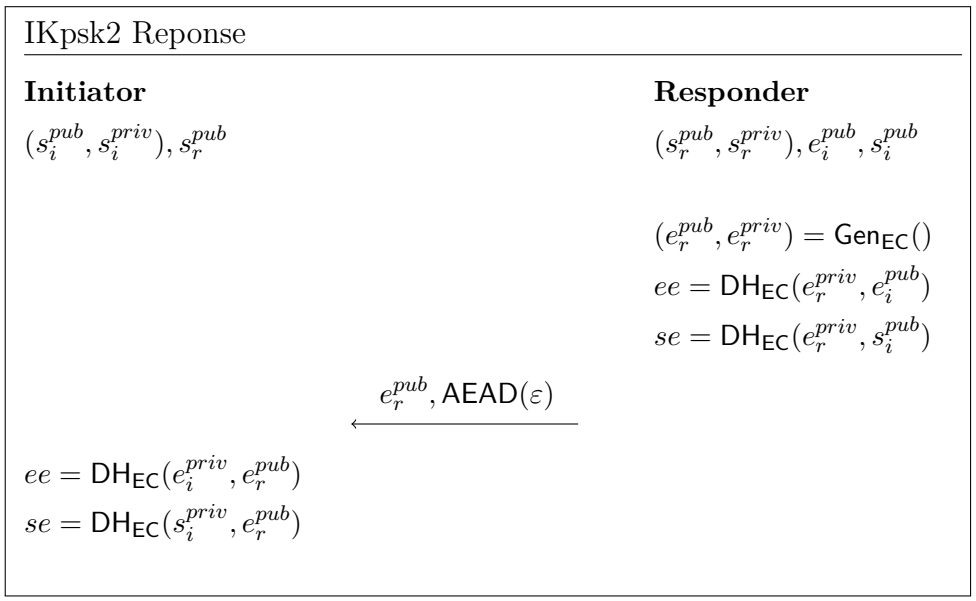

Figure 2.2.: Simplified diagram of the response step of WireGuard's IKpsk2 protocol.

other direction.

There also exists a formal proof over the WireGuard protocol, showing the general construction to be secure. [16] This proof is performed by using the Tamarin Prover protocol verification software.

Cryptographic Primitives

WireGuard's handshake protocol is performing ECDH on the elliptic curve called Curve25519 [5]. This curve uses 256-bit keys and provides 128-bit security in a classical setting. In a quantum setting this primitive is completely broken by Shor's algorithm.

During the handshake, the BLAKE2s cryptographic hash function is used with 256-bit output. It is not only used for the continuous hash, but also as pseudo-random function (PRF) in key derivation based on HKDF. HKDF is built upon the HMAC construction, which does not suffer from collision attacks on the underlying hash function. As far as the underlying hash function can be assumed to be a PRF the HMAC can also be assumed to be a PRF for any key. This was shown by Bellare [3] in 2006. Therefore, HMAC and consequently HKDF provide 256-bit security, when based on 256-bit keys and the 256-bit BLAKE2s function. The chaining hash on the other hand uses BLAKE2s directly. For now though, it seems reasonable to accept Bernstein's criticism, and assume the best practical attack against hash functions still

is of complexity $\mathcal{O}(2^{n/2})$. Also, the security of the handshake does not rely on the continuous hash being collision-resistant.

After the handshake is completed, WG then uses ChaCha20 and Poly1305 for AEAD. Both of these primitives are based on 256-bit symmetric ciphers. In a classical setting these therefore provide 256-bit security. As we have seen in the section about Grover's algorithm, the security level may be lower in a quantum setting, but it is not yet clear by how much. Though, by definition, these achieve NIST level V as defined below.

2.5. State of Post-Quantum Cryptography

With PQ cryptographic primitives there are not yet any established standards. The United States' National Institute for Standards and Technology (NIST) is currently running a project looking to standardize PQ cryptography. Probably our best bet when looking for promising cryptographic primitives is to look at those submitted to this NIST project, especially the ones which made it in round 2 of the project. As those have already undergone public scrutiny, and are still considered viable candidates for standardization. All cryptographic primitives mentioned in here are submissions to round 2 of the NIST project. [24]

NIST Level	Classical Primitive	Attack Cost
I	AES-128	2^{106}
II	SHA-256	2^{146}
III	AES-192	2^{169}
IV	SHA-384	2^{210}
V	AES-256	2^{234}

Table 2.1.: Post-quantum security levels defined by NIST. Adapted from table in [23].

In their call for proposals for the PQ standardization project, the NIST defines five levels of security for PQ cryptographic primitives. [23] Here these are labeled with roman numerals I-V, to better distinguish them from another security classification introduced in the next chapter. Each of these levels is defined as being at least as strong as current classical cryptographic primitives AES, SHA3 with specific parameters, as shown in Table 2.1. The column 'Attack Cost' shows a current estimate for the cost of an attack on the reference primitive, based on numbers from the NIST call for proposals [23].

Next, there is an overview of some of those PQ cryptographic primitives, and whether they might be useful for bringing PQ security into WG.

SIDH

Supersingular isogeny Diffie-Hellman (SIDH) is a key exchange protocol [20], which brings the idea of the ECDH protocol to a PQ setting by using a different mathematical foundation. Instead of dealing with points on a single elliptic curve, this method is based on **isogenies over supersingular elliptic curves**. This is exactly what makes this method resistant against attacks based on Shor's algorithm. Namely, the underlying supersingular isogeny problems are not reducible to the DLP. It is a unique primitive among the NIST submissions: "[It is] the only candidate based on arithmetic properties of elliptic curves over finite fields." [1] Still, the mathematics behind it is well understood, as research on isogenies and supersingular elliptic curves has long been going on.

Notably, SIDH allows for smaller key sizes than any of the other NIST submissions. And they can be further reduced through compression, as shown by Costello et al. at Microsoft. [12] Unfortunately, computations of this type are expensive, and compression makes them even more expensive. This method is around an order of magnitude slower than many of the other candidates, and also way slower than its classical equivalents.

McEliece

The McEliece [22] cryptosystem is one of the many **code-based** key exchange schemes submitted to the NIST competition. Security of this scheme is based on the hardness of decoding random linear codes. It was also the first method of this type, and has been around since 1978. Therefore, it is considered to be one of the more proven PQ cryptographic schemes.

The major disadvantage of this category of primitives is the size of their public keys. Especially the McEliece system has keys that are hundreds of Kilobytes in size, up to around one Megabyte for the parameters submitted to the NIST competition. Even the BIKE code-based key exchange, which the NIST calls "competitive with ring and module lattice schemes" [1] in terms of key size, has key sizes which are 2 4 times larger than its lattice-based competitors.

NTRU

NTRU is a key exchange protocol developed in 1996. [19] Multiple parameter sets were proposed for round 2 of the NIST standardization process, claiming NIST security levels I, III and V. It represents the largest group of submissions, those using **lattice-based** cryptography. The underlying mathematical problem is called the shortest vector problem in lattices, for which it is of course assumed that there is no efficient quantum algorithm.

This category of primitives currently is most promising, as they offer a good trade-off in public key sizes and computational speed. Specifically, NTRU has been around for a long time. Also, it uses more reliable security properties than SABER [13] and CRYSTALS-Kyber [9]. These are both also lattice-based cryptosystems, based on less proven assumptions about Module Learning with Rounding. Another contestant with more conservative assumptions is Google's submission, called NewHope [2].

The rest of the thesis will always talk about PQ primitives in terms of a **key encapsulation mechanism (KEM)**. That is, we will assume they provide three relevant functions. Firstly, a key pair generation function, securely pseudo-randomly generating a pair of matching private and public keys. Secondly, an encapsulation function, taking a public key and returning an encapsulated value and a shared secret. Lastly, a decapsulation function, which in turn can be used on an encapsulated value together with a private key. If the provided private key matches the public key which was used to generate the encapsulated value, the decapsulation is successful and returns the same shared secret that resulted from the original encapsulation.

3. Methodology

3.1. Feature Specification

The primary goal of the adaptations proposed in this thesis is to provide security against future attackers, who are already recording traffic at the current time for later decryption. Therefore, the two main properties that need to be ensured are:

Perfect Forward Secrecy: Revelation of the static private key of any party should give no advantage in attacking any message encrypted before the keys were compromised. WireGuard only protects us in this way from attackers with classical computers because a quantum adversary can get the ephemeral private key from the public key through Shor's algorithm. The proposals in this chapter will also consider attackers with strong quantum computers.

Identity Hiding: Unless the responder's static private key is compromised, the initiator's identity, that is their static public key, should not be revealed. This is the same level of identity hiding WireGuard offers against classical attackers.

There are also secondary goals, which were considered as much as possible while still achieving all the primary goals. These secondary goals closely match the main selling points of WireGuard:

- high throughput (over tunnel)
- low ping (on handshake and tunnel packets)
- 1-RTT handshake

Security against an active quantum-capable attacker is an optional low-priority goal because at the current moment this is not a clear threat. On the other hand, someone could currently be collecting encrypted traffic to decrypt it once they have a quantum computer. This threat is already real and current.

3.2. Protocol Design

First of all, there will be a definition of 4 security levels for PQ handshake protocols. These satisfy the original feature set WireGuard has in a classical setting to varying degrees in a quantum setting. In contrast to the security levels defined by NIST, these levels only make claims about general properties of key exchange protocols, regardless of the primitives and key lengths used. The NIST security levels on the other hand concern cryptographic primitives and their parameters, and are defined by a specific level of computational cost that would be needed to break them. One could achieve each of the above handshake security levels for any of the NIST security levels. When using different primitives or different parameters for one primitive in a single protocol, one could even achieve different NIST security levels for different parts of the same protocol. In this sense, the two security levels are orthogonal.

This classification starts from level 0, which in a quantum setting gives none of WireGuard's interesting security features, and goes up to level 3, which should be at least as secure against quantum attackers as WireGuard is against classical attackers:

Level 0 is defined by what the WG specification allows. PQ security is very limited: It provides **basic encryption**, but neither PFS, nor identity hiding, nor any security against active quantum-capable attackers. Thus, if a quantum adversary learns the PSK, everything is compromised. By definition IKpsk2 is a valid level 0 handshake if the PSK comes from a PQ KEM.

Level 1 provides **PFS** on all tunnel packets, but there is no guarantee of identity hiding, and no guarantees against active attackers. As soon as available quantum computers can break the classical primitives used in WireGuard, the identity of the initiator has to be seen as compromised.

Level 2 guarantees PFS and **identity hiding** on the same level as WG does against classical adversaries. It still only considers passive attackers though. Therefore, once strong quantum computers exist, protocols that achieve this security level or lower need to be considered unusable.

Level 3 is only considered achieved if the protocol has the same general security properties in a PQ setting, as WG's handshake does in a classical setting. That is, it needs to be **secure against active attackers** with quantum computers, in addition

to all capabilities of a level 2 key exchange. Such a protocol can therefore no longer rely on classical cryptography for authenticating the peers to each other.

To achieve any of these goals, except for level 0 of course, adaptations to the handshake protocol definitely need to be made. The rest of the cryptography does not need to be adapted. All currently known quantum algorithms do generally break neither symmetric ciphers nor hash functions. They might force people to use longer keys than they would have done otherwise, but 256-bit keys should be enough for the foreseeable future.

Now the three handshake protocols will be presented. All of these use classical and PQ cryptographic methods in a hybrid setting, meaning every PQ KEM is backed up by a classical key agreement serving the same purpose. The reason for this decision is that all currently available PQ KEMs are in an early developmental stage. No sufficient testing and analysis has taken place. It would therefore be reckless to rely on these alone for the protocol's security. That is why results of both types of exchanges are incorporated into the keys, thus giving at least the security of the classical exchange, even if the PQ primitives turn out to be less secure. All the following handshakes are simple extensions of the protocol used in WireGuard. The way how Noise protocols use the chaining key, they naturally allow for extension. For adding new key agreements as parts of the handshake, one can simply add the results of these key agreements into the chaining key to incorporate them into the protocol. The proposals meet PQ security levels 1, 2, and 3 respectively, as defined above.

In the diagrams depicting the protocol calculations and message exchanges, everything that stays the same is omitted. This is a reasonable simplification because every increasing level only ever adds steps to the previous protocol. No steps of the previous protocol are ever modified or removed. Also, for legibility, the computations for updating the continuous hashes and the chaining keys have been omitted from the diagrams.

3.2.1. L1 Handshake

Firstly, the aim is to go from the default level 0 protocol to a level 1 handshake, i.e. to achieve PFS in a quantum setting.

A simple ephemeral key encapsulation is performed for some PQ cryptographic primitive. The initiator generates an ephemeral key pair, and sends their public ephemeral PQ key with the initiation message in plaintext, just as they do with the classical ephemeral key. Upon receiving the initiation, the responder performs a

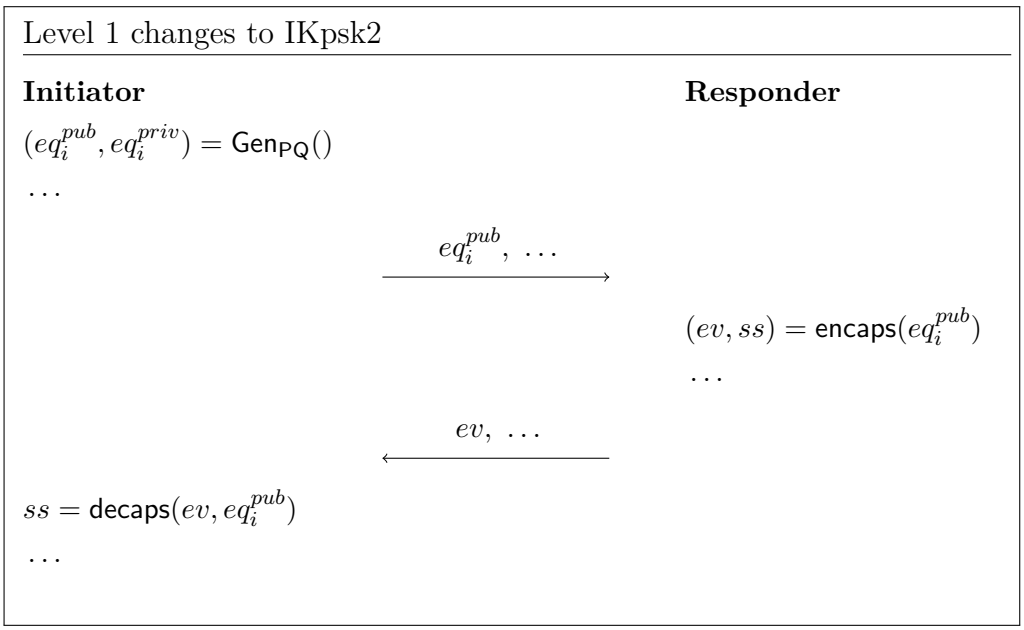

Figure 3.1.: Steps added to the IKpsk2 protocol to achieve level 1 PQ security.

key encapsulation on that public key. This results in them having two values: The shared secret and an encapsulated value, which is based upon that same secret and the initiator's public key. The response message includes the encapsulated value, also in plaintext. Upon receiving the response the initiator can then do the inverse, by decapsulating the shared secret from the encapsulated value using their private key. Both parties are then in the same state again, and have performed an ephemeral key exchange. Figure 3.1 shows all steps that need to be added to the level 0 handshake protocol in order to arrive at a handshake with level 1 PQ security.

Here is the level 1 handshake protocol in something resembling Noise notation, where changes to the IKpsk2 protocol are highlighted in bold:

```
<- s
...
-> e, eq, es, s, ss
<- e, enc(eq), ee, se, psk
```

This protocol obviously provides PFS in a PQ setting because both parties agree upon an ephemeral secret in a way that is secure against quantum adversaries. PFS is also at least as strong against classical adversaries as in WG because all ECDH exchanges are still part of the protocol. Authentication is given by the underlying classical handshake

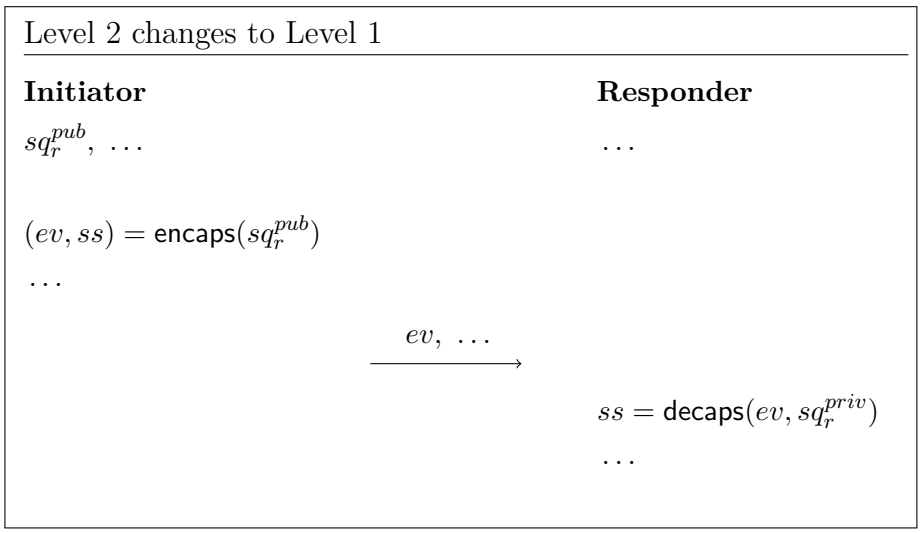

Figure 3.2.: Steps added to the level 1 handshake protocol to achieve level 2 PQ security.

because the handshake can still only succeed if both parties proof that they own the private key corresponding to their identity.

3.2.2. L2 Handshake

Secondly, the goal is to go from level 1 to level 2 PQ security for the handshake protocol, i.e. to achieve identity hiding as long the responder's static private key stays secure.

To achieve that, a static PQ key pair is added on the responder's side. Assuming the PQ static public key is also known to the initiator from the beginning, as is the classical one. The initiation message can then already contain the encapsulated value generated from it. On the other hand, the shared secret from the encapsulation can be integrated into the chaining key, prior to encrypting the classical static public key of the initiator. In Figure 3.2 you can see all steps that need to be added to the level 1 handshake protocol shown previously in order to achieve level 2 PQ security.

Following is this level 2 handshake protocol in pseudo-Noise notation, where changes to the level 1 protocol have been highlighted in bold:

```
<- s, sq
...
-> e, eq, enc(sq), es, s, ss
<- e, enc(eq), ee, se, psk
```

The initiator's identity is now encrypted under the new PQ shared secret. Assuming

the PQ KEM used is secure, the only way for the responder to derive the shared secret, and thus the initiator's identity, is to know the secret key corresponding to sq. No malicious party can therefore find out the initiator's identity, without first having compromised the responder's private static key. This is the same level of identity hiding WG provides in the classical case. Identity hiding is also at least as strong as in WG because all ECDH exchanges are still part of the protocol.

Inadvertently, this protocol also serves to authenticate the responder to the initiator on their PQ identity because the handshake can only finish successfully if the responder manages to decrypt the initiator's identity.

3.2.3. L3 Handshake

Lastly, the target is to go from level 2 to level 3 PQ security for the handshake protocol, i.e. to also achieve security against active attacks by quantum adversaries.

Another static key pair is added, this time on the initiator's side. The basic structure of the PQ parts of the protocol then resembles a Noise IK handshake, which is the same as IKpsk2 only without the PSK. Both parties have a static key pair to authenticate each other's identities, where the responder's static identity is known to the initiator in advance. Also, the responder is first to authenticate their identity, they both agree on an ephemeral secret together, and finally the initiator authenticates their identity.

The PQ static public key of the initiator is sent in the same way as the classical equivalent. Both are under an encryption key based on shared secrets derived from both static public keys of the responder, the classical and the PQ one. Thus, both keys have the same level of identity hiding, which should be as strong as the strongest of the classical and the PQ primitive. After the responder received the initiation message, they can then decrypt the initiator's PQ static public key, and subsequently encapsulate it. Finally, the responder sends the encapsulated value back in the response message, and integrates the shared secret into the chaining key. To arrive at the same state, the initiator does the equivalent steps on their side upon receiving the response message. Figure 3.3 displays all steps that need to be added to the level 2 handshake protocol shown previously in order to reach level 3 PQ security.

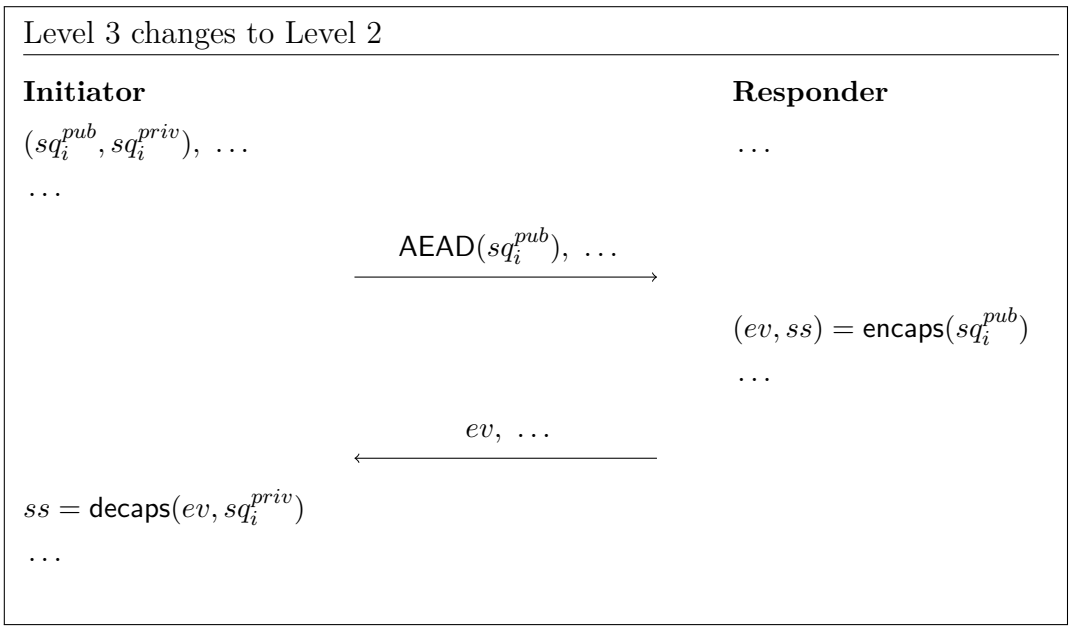

Figure 3.3.: Steps added to the level 2 handshake protocol to achieve level 3 PQ security.

This is the level 3 protocol in pseudo-Noise notation, where everything that has changed from the previous level 2 protocol is written in bold text:

```
<- s, sq
...
-> e, eq, enc(sq), es, s, sq, ss
<- e, enc(eq), enc(sq), ee, se, psk
```

These changes serve to authenticate the initiator on their PQ identity. This is indeed achieved, because the initiator can only arrive at the same final state as the responder if they know the shared secret from the encapsulation. Assuming the PQ primitive used is secure, this is only possible if they can decapsulate the encapsulated value for the public key they sent. This in turn should only be possible if they know the private key matching that public key. Therefore, if the first data packet sent by the initiator has valid authentication, the responder can be confident that the initiator knew the private key corresponding to the public key they sent in the initiation message.

4. Implementation

4.1. Engineering Process

Two libraries already offer a unified interface for implementations of many NIST candidates. The two libraries in question are liboqs [30] and libpqcrypto [7]. Both these PQ cryptography libraries and most of the original reference implementations heavily depend upon the C standard library. The problem then is that they can not be directly compiled as part of a kernel module. This led to the decision of basing the proof-of-concept implementation on a user space implementation of WireGuard, as opposed to the original kernel space implementation. Cloudflare has developed an implementation of the WireGuard software in user space. They call this implementation BoringTun and it is open-source on GitHub. [11] It is written in the Rust programming language. This project was used as basis for the proof of concept.

An additional benefit of this decision is that it allows for easy benchmarking via Criterion.rs [18], a Rust crate which will be introduced in the next chapter. This is especially useful for a proof of concept.

As the proof-of-concept code directly extends upon a WireGuard implementation, the design choices that were made in developing WireGuard were considered here. The fact that the proof-of-concept code works with Cloudflare's Rust implementation, instead of the original C code, makes some memory-safety design choices of the WireGuard code obsolete. Care has been taken to not break any of the general design principles through the changes made.

liboqs

Open Quantum Safe is an open-source project started by Douglas Stebila and Michele Mosca of Waterloo University. Main part of the project is liboqs [30], a C library of post-quantum cryptography functions. This is also the library used for the PQ cryptography in the proof-of-concept implementations, where the C functions are called via the Rust's foreign function interface.

One advantage of this library over libpqcrypto is, that it has implementations for SIDH. Also, it is updated and maintained more frequently. Therefore, adaptations to the NIST submissions are integrated earlier.

4.2. Proof-of-concept Code

Source code of the proof-of-concept implementation, including the benchmarks we discuss in the next chapter, is available on GitHub as a project forked from Cloudflare's BoringTun [11]:

`https://github.com/qkniep/pqwg-rust`

The project is split among two branches: `master` (implementing L0 and L1), and `lvl23` (implementing L2 and L3 handshakes). Whenever the proof-of-concept implementation is mentioned throughout this thesis, the commits with commit message 'BA Submission Version', and with the following hash values, are referenced:

```
master: b2a19d5531dce66439190e7be58780c4e19f96f9
lvl23:  706f05d23da4192c5ff9b3a9082de205a9cfe150
```

A basic manual on how to run the benchmarks can be found in the README file of the repository.

5. Results and Critical Discussion

As the main question here is one of usability, the following will mainly be a performance comparison, comparing the proof-of-concept handshake protocol implementations to the default WG protocol, as implemented in BoringTun. To accommodate for any of the three higher security levels defined in the Feature Specification, only the handshake code was changed. Therefore, most focus will be put on the handshake computations, as well as initiation and response message transmissions.

5.1. Message Sizes

Adding more key exchange primitives to the handshake protocol obviously adds additional data, in the form of public keys and encapsulated values. This data needs to be transmitted in the initiation or response message.

With PQ KEMs that data is relatively large for network transmissions, hundreds of Bytes up to hundreds of Kilobytes in the worst cases. Thus, packets may easily become larger than the Ethernet maximum transmission unit (MTU) of 1500 Bytes. Since WG is based on UDP, there is no segmentation functionality on the Transport Layer. If large data is sent over UDP, packet fragmentation may happen on the IP layer. Relying on this functionality is considered fragile, and the IETF specifically advises against it. [8] This is because the IP standard allows routers to drop large packets silently.

A way around IP fragmentation is to split the datagrams on the application layer. When using UDP, this can still be terrible for performance because the handshake will fail anytime one of the datagrams is lost. The more datagrams we need for one message, the higher the probability for losing one of them is. Thus, breaking up messages into more datagrams makes the handshake take longer on average.

A datagram size of 1436 Bytes is reasonable for this, because according to analysis by Shannon and Moore [31] around 98% of MTUs are between 1484 1500 Bytes. Subtracting 8 Bytes for the UDP header and 40 Bytes for the possibility of an IPv6

header, we arrive at the number above. Also, the IPv6 standard *recommends* an MTU of at least 1500 Bytes, and *requires* acceptance of packets up to that same size. [14]

Connection	**dt** [ms]
Wired-Cable	23.1
Wireless-House	29.2
Wired-DSL	32.2
Wireless-Apartment	36.7

Table 5.1.: Average time impact of sending an additional UDP datagram as part of the WG handshake, for different connection types.

In WireGuard datagram loss is especially problematic because: "Under no circumstances will WireGuard send an initiation message more than once every Rekey-Timeout." [15] By default, this *Rekey-Timeout* is set to five seconds (5000 ms). Table 5.1 shows the average cost of an additional UDP datagram being sent, that results from this Rekey-Timeout policy combined with a study about typical packet loss rates published by Raghavendra [29]. This gives an upper-bound estimate of about 23 37 ms, depending on connection type, for the average cost of each additional datagram. These numbers ignore the impact of half-finished key exchanges. Anyways, this impact is negligible, at around 0.5 ms for a very expensive handshake, which takes 200 ms for its computations. Going on from here, all calculations will use the average of **30 ms per additional datagram**.

Choosing cryptographic primitives with small key sizes is thus essential to prevent unnecessary overhead, in the form of excessive transmission times. Cryptographic primitives with excessively large public keys or encapsulated values have thus been excluded from all following considerations. For example, all code-based cryptosystems, which need many Kilobytes for a public key.

Amplification Attacks

The SABER series of cryptographic primitives [13] has a specific problem regarding message sizes. WireGuard relies on the initiation message being larger than the response message, to prevent DoS attacks based on amplification. For each of the SABER primitives (LightSABER, SABER, FireSABER), the encapsulated values are more than 56 Bytes larger than the public keys. These 56 Bytes are exactly the difference in length standard WG has in favor of the initiation message. In the L1 and L3 handshakes previously defined, the response message contains one more encapsulated

value, whereas the initiation message contains one more public key. Therefore, using a SABER KEM for these handshakes would lead to vulnerability against amplification attacks, as the response message would then become larger than the initiation message. Though to use a SABER primitive anyways, the initiation message can be padded to be larger again. The numbers of datagrams in the following always accommodate for this padding.

Memory Exhaustion Attack

Another problem when splitting messages, here especially the initiation message, into multiple UDP datagrams is the possibility of a memory exhaustion attack. [8] This is a type of DoS attack, in which an attacker tries to fill the target's memory. In this case they would achieve that by sending datagrams, which appear to be incomplete initiation messages. If the responder saves all the datagrams, waiting for another datagram to finish the message. The malicious initiator may continue sending incomplete messages, until the responder runs out of memory.

WireGuard already has a system that uses MACs and a concept called IP-binding cookies to prevent CPU-exhaustion attacks. [15] This system could be expanded, to also prevent the type of attack described above. IP-binding cookie messages should be sent, not only when the system is under computational load, but also if the system is running short on memory.

An alternative approach would be to include the data WG would send in the initiation message in every datagram of the split initiation message. Only the PQ data would still be split among the datagrams. Upon receiving any of the datagrams, the responder could already start checking the classical WG data for validity. They could then decide to keep the datagrams already received, and potentially wait for the other datagrams to arrive, if the classical WG data is valid so far. In the other case, they could throw away the datagrams. Of course this has some overhead in message size. Another disadvantage is, that this approach would make parsing incoming messages more difficult. Thus increasing the amount of code needed for that functionality.

Cryptographic Primitive	PK EV [Bytes]	L1 Ini/Rsp [Datagrams]	L2 Ini/Rsp [Datagrams]	L3 Ini/Rsp [Datagrams]
NewHope1024	1824 2208	2 / 2	3 / 2	5 / 4
Kyber-1024	1568	2 / 2	3 / 2	4 / 3
FireSABER	1312 1472	2 / 2	3 / 2	4 / 3
NTRU4096	1230	1 / 1	2 / 1	3 / 2
SIDHp751	564	1 / 1	1 / 1	2 / 1
SIDHp751c	334	1 / 1	1 / 1	1 / 1

Table 5.2.: Number of datagrams of size 1436 needed for the different handshake messages, with some NIST level V primitives.

Cryptographic Primitive	PK EV [Bytes]	L1 Ini/Rsp [Datagrams]	L2 Ini/Rsp [Datagrams]	L3 Ini/Rsp [Datagrams]
Kyber-768	1184 1088	1 / 1	2 / 1	3 / 2
SABER	992 1088	1 / 1	2 / 1	3 / 2
NTRU2048	930	1 / 1	2 / 1	3 / 2
SIDHp610	462	1 / 1	1 / 1	2 / 1
SIDHp610c	273	1 / 1	1 / 1	1 / 1

Table 5.3.: Number of datagrams of size 1436 needed for the different handshake messages, with some NIST level III primitives.

5.2. Handshake Benchmark

All runtime benchmark results that follow come from the different benchmarks being run on the same workstation, with the following hardware and system specifications:

- CPU: Intel Xeon E3 1230 v3 (4×3.3 GHz, 8 MB Cache, AVX2, AES-NI)
- RAM: 8 GB DDR3 1600
- OS: Arch Linux x86_64
- Kernel: 5.3.7 arch1 2 ARCH

Except for a simple window manager and the terminal emulator, no user processes were running on the workstation during benchmarking.

The benchmark code is based on the test code written by Cloudflare and provided with the BoringTun source code. Specifically, the benchmark calls the handshake test function. The test code has been adapted to send just a single packet over the

Cryptographic Primitive	PK [Bytes]	EV [Bytes]	L1 Ini/Rsp [Datagrams]	L2 Ini/Rsp [Datagrams]	L3 Ini/Rsp [Datagrams]
NewHope512	928	1120	1 / 1	2 / 1	3 / 2
Kyber-512	800	736	1 / 1	2 / 1	2 / 2
LightSABER	672	736	1 / 1	2 / 1	2 / 2
NTRU2048	699		1 / 1	2 / 1	2 / 2
SIDHp434	330		1 / 1	1 / 1	1 / 1

Table 5.4.: Number of datagrams of size 1436 needed for the different handshake messages, with some NIST level I primitives.

WireGuard tunnel, instead of the 128 packets in the original code. This still ensures the handshake is actually performed, and that a packet can successfully be transmitted, but it measures mostly the runtime of the handshake itself. On the reference system, sending one packet over the tunnel takes less than 0.25 ms.

Criterion.rs

Criterion.rs [18] is the Rust crate that was used for running the benchmarks and doing the basic statistical evaluation. Criterion takes a number of samples s and a measurement timespan m as inputs for running a benchmark. It then runs the benchmark a few times for warm-up, to have data already loaded into caches if applicable. Following this, the actual measurement part begins. During this phase, Criterion performs s samples $S_1, \ldots, S_s$. For each sample S_x it runs $x \cdot i$ iterations, where i is the minimum integer for which all samples can run in a total time of about m. Also, i is at least 1, even if the total benchmark runtime then is way more than m. In this way, sample size can override the measurement time.

Unless otherwise specified, all following benchmarks were run on the reference system above, with sample size set to 100, and the measurement time set to at least one minute.

Results

In the following, results of the runtime benchmarks will be presented. These use a selection of the most interesting cryptographic primitives, based on the results from the Message Sizes section and results from the benchmark in liboqs (Appendix A). SABER and NewHope primitives have been removed from the selection because both

have slower computation and are no better in message sizes than Kyber. For the same reason the NIST level I variant of NTRU is not part of the selection.

L0 Time [ms]	1.245		
Primitive	**L1 Time** [ms]	**L2 Time** [ms]	**L3 Time** [ms]
Kyber-512 (I)	1.646	1.742	1.986
SIDHp434 (I)	66.20	113.1	160.6
Kyber-768 (III)	1.859	2.066	2.459
NTRU2048 (III)	18.77	18.96	20.09
Kyber-1024 (V)	2.123	2.387	3.010
NTRU4096 (V)	26.33	27.47	28.86

Table 5.5.: Average runtime of each handshake with different NIST primitives. Measuring the handshake and sending a single packet over the tunnel.

The runtime for any combination of the PQ handshakes with the NIST primitives is shown in Table 5.5. It can be seen that the Kyber family of primitives is the fastest by a big margin. SIDHp434 is so slow, that the slower variants which would achieve NIST levels III and V have not been considered.

5.3. Use Cases

Home VPN Server (L1)

In this scenario, identity hiding is not as important as the data going through the tunnel. An example for this would be a VPN setup where there is no identity hiding anyways. For example, if you have a RaspberryPi set up at home that you alone use as a VPN server. To which you connect when you are on the go, to be less vulnerable through unprotected Wi-Fi networks.

Kyber-768 seems almost perfect for the L1 handshake, as it is almost as fast as WG (see Figure 5.1), and also fits into one datagram per message.

If one wants to achieve NIST level V instead, choice of cryptographic primitives is not as easy. As we have seen in Table 5.2, we have a choice between SIDH and NTRU, to keep everything in one datagram per message. Then again, NTRU is much faster than SIDH, see Table 5.5. Kyber-1024 needs three additional datagrams. This might seem daunting, but computationally it is way faster than NTRU. The trade-off may be worth it in some cases, for example if the connection is known to be reliable.

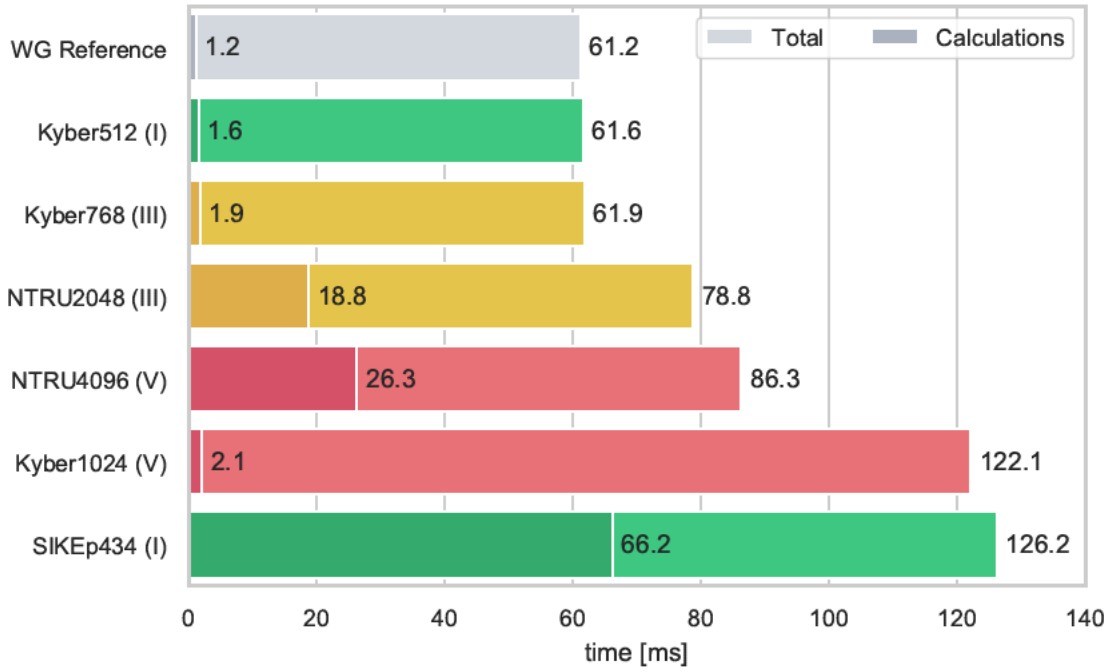

Figure 5.1.: Time the L1 handshake takes with different PQ crypto primitives.

Going for a lower NIST level than III is not reasonable. The cost advantage of using Kyber-512 is minimal, at less than 0.5 ms. At the same time, NIST level I is significantly less secure than level III, providing an expected 64 bits less security, see Table 2.1. So, unless forced to by a really low-resource environment, going for **Kyber-768 or NTRU4096** would be the more reasonable choices.

Trusted Network Access (L2)

For this use case, we want to protect the data and the clients' identities. For example, the VPN server is a secure entry point of some sort: to a company, university, or some other trusted network. Many different users connect to the server, to securely access computer infrastructure inside the trusted network.

Achieving NIST level V on an L2 handshake is the most interesting case. It corresponds to the original goal of having the same security guarantees in a PQ setting that WireGuard has against classical adversaries. Here, the cost trade-off of sending additional datagrams and doing more costly computations has to be pondered. Choosing NTRU over SIDH is well worth it, as we have seen. Also, the same trade-off of using Kyber-1024 over NTRU4096, as for the L1 handshake, may be pondered. As

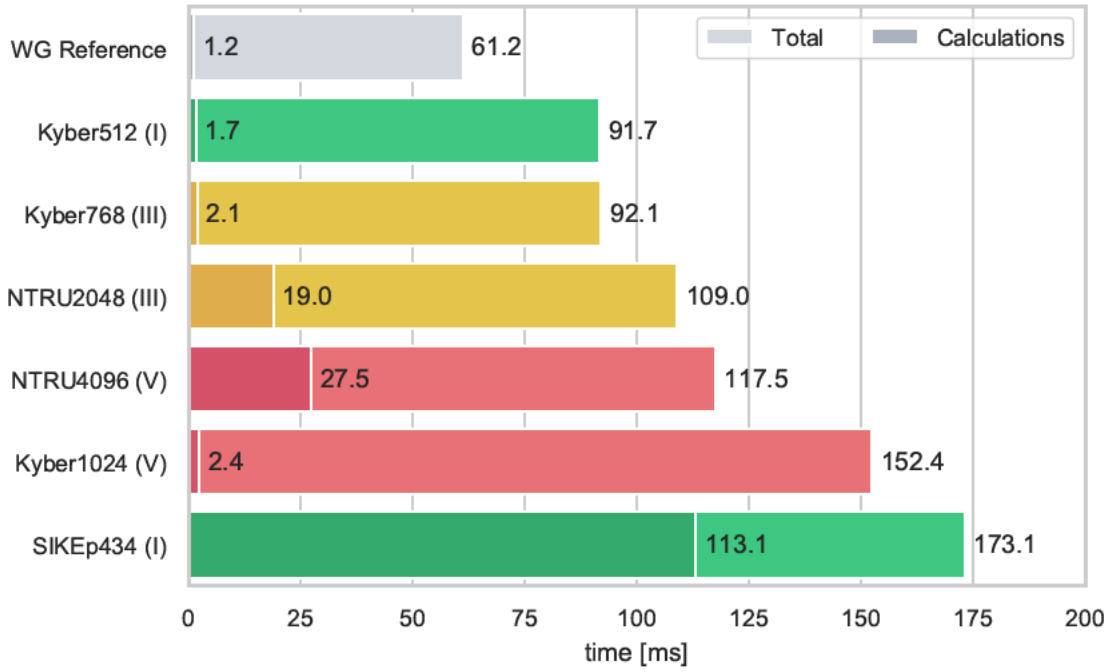

Figure 5.2.: Time the L2 handshake takes with different PQ crypto primitives.

can be seen in Table 5.5, for the L2 handshake Kyber-1024 takes 25 ms less time in computation than NTRU4096. Also, the difference in transmission times is lower than for the L1 handshake. Especially in settings where computation power is more scarce than connection stability, Kyber-1024 can be favored.

It could again be reasoned to only go for NIST level III, for which the much faster Kyber-768 can be used, while at the same time using no more datagrams than NTRU4096 on L2. Going for even weaker cryptographic primitives is once again not warranted. In general though, it is recommended to use **NTRU4096 or Kyber-1024** for an L2 handshake. This achieves all goals set in the Feature Specification at a reasonable cost.

Future Proof System (L3)

In cases where a system is developed or infrastructure built that should not need to be adapted again once capable quantum computers arrive, the system needs to already implement an L3 handshake. If it were reasonably inexpensive to implement, one could even argue to skip L2 and directly include L3 everywhere.

Implementing the L3 handshake with NIST level V primitives costs at least an

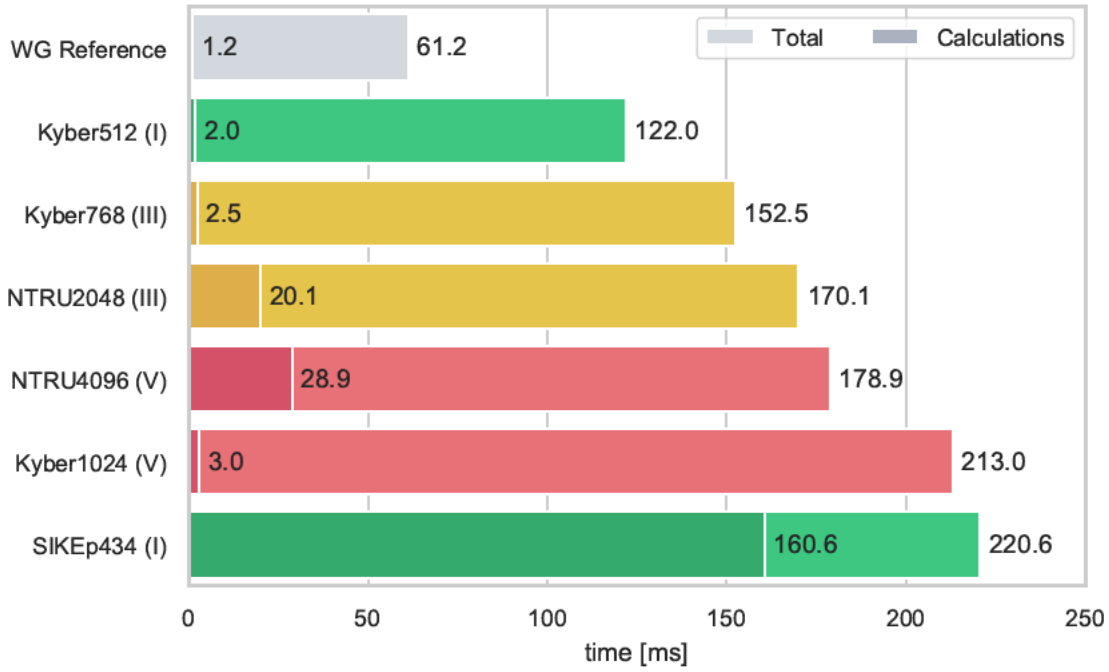

Figure 5.3.: Time the L3 handshake takes with different PQ crypto primitives.

additional two datagrams, i.e. around 60 ms, over L2. Alternatively, one could switch to SIDH, which is even more expensive. That is a lot to pay for preventing attacks which are not possible until major developments in building quantum computers are achieved. Skipping L2 is therefore in general not warranted. Only in cases where a reliable connection is absolutely guaranteed, it might make sense to already introduce an L3 handshake because the major cost for additional datagrams would vanish.

An alternative approach would be to lower the NIST level to III. Then, the fastest solution would be Kyber-768 again, needing three additional datagrams over the L0 handshake. Additionally, this has the cost of also weakening our defenses against the (more relevant) passive attacks. Only targeting NIST level III (or lower) for such a future-oriented system is probably a bad idea anyways.

In Summary, if one really wants to be this future-proof, a significant price in performance needs to be paid. Using either **NTRU4096 or maybe Kyber-1024** as the cryptographic primitive looks suitable for this use case.

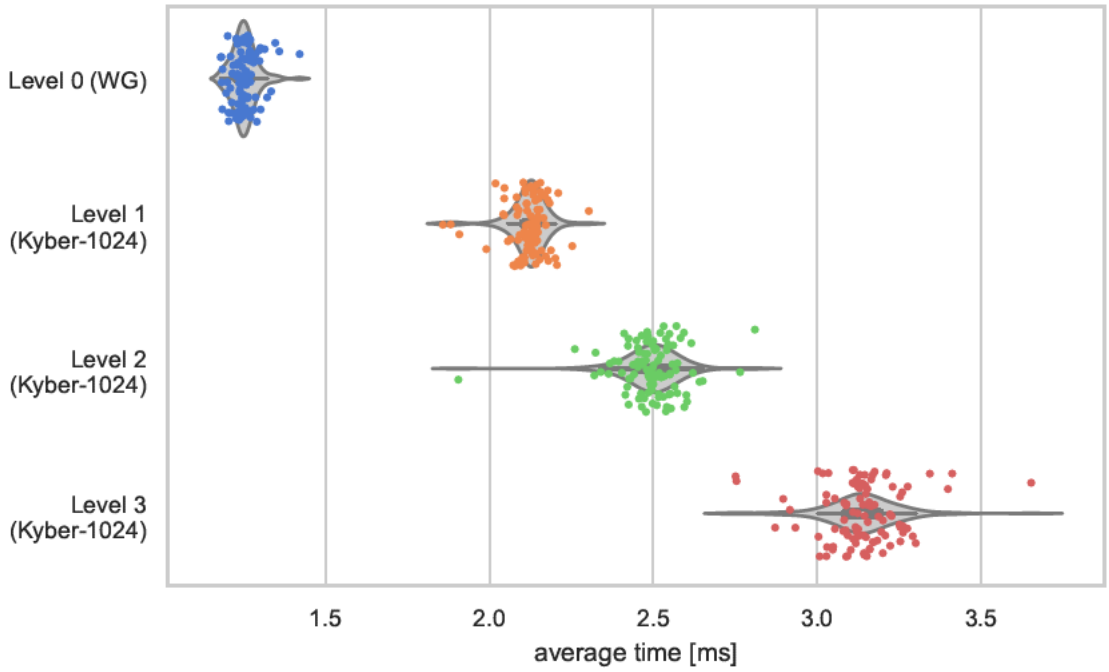

Figure 5.4.: Runtime of the handshakes with different PQ security levels. Each performing the handshake and sending 1 packet over the tunnel.

5.4. Throughput, Ping, Reliability

The visual representation used for these results is called a violin plot. Also, the single measurement points, each representing a single Criterion sample, are overlaid on top. The violin plot in the background simply shows the probability distribution of the samples.

To incorporate the desired PQ security features, only the handshake had to be changed. Notably, nothing about the symmetric cryptography was changed. Once the tunnel is established only symmetric crytpography, in form of the AEAD based on ChaCha20 and BLAKE2s, is used. Therefore, it is to be expected that there is no noticeable difference in throughput and average packet ping. Especially when measured over the course of a longer networking session.

When we benchmark just the handshake and sending a single packet over the WireGuard tunnel, we still see significant differences between the handshakes. This can be seen in Figure 5.4.

But even after sending only 2000 packets, it is already apparent that the key exchange makes little difference on the total time it takes. This can be seen in Figure 5.5. In

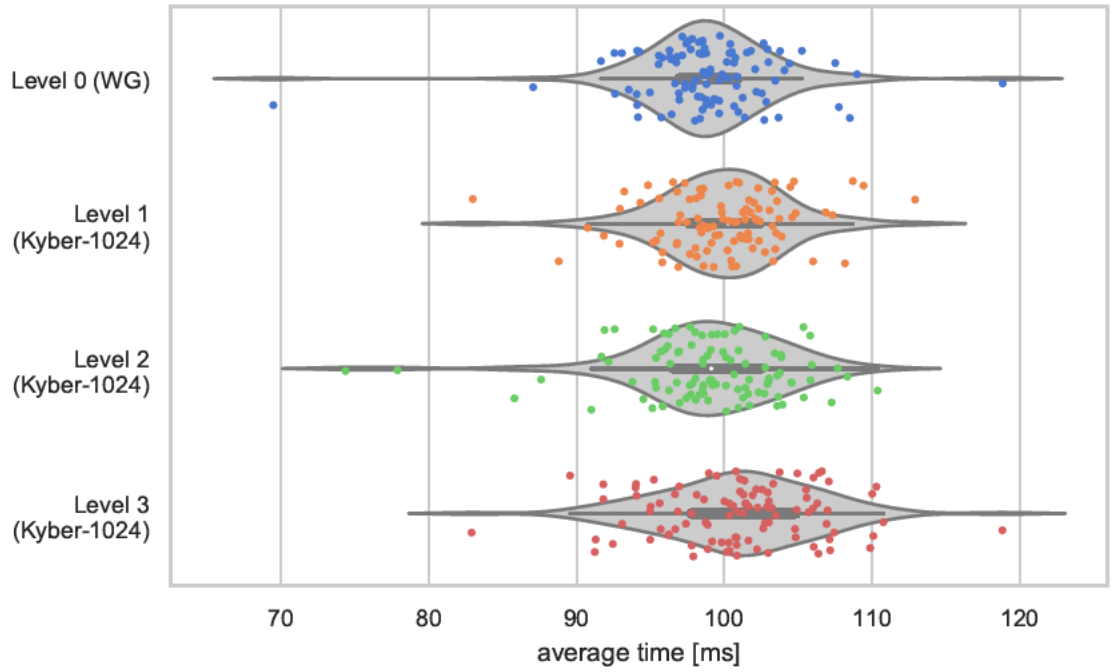

Figure 5.5.: Runtime of the handshakes with different PQ security levels. Each performing the handshake and sending 2000 packets over the tunnel.

these examples packets are sent to *localhost*, and latency is thus minimal. Results are expected to be even more similar for the different handshakes as number of packets transmitted increases, or if packets are sent over an unreliable connection. Also, standard deviation is much higher than with only a single packet. This is also expected because packet transmission times are highly variable. When increasing packet count or reducing reliability of the connection, this change should also become more apparent. This benchmark already serves as an indicator that throughput does indeed stay the same when using the higher security level handshakes.

WireGuard already has a feature that ensures that service is not disrupted during the handshake. There are two timeouts for a WG session. A soft timeout after two minutes, when WG will start the new handshake. The hard timeout is only after three minutes, when WG will no longer accept packets encrypted with the old session key. Consequently, there is a sixty seconds grace period, during which the old key can still be used to send packets over the tunnel. For the results above this means that increasing the probability of the handshake failing, as most PQ methods do by increasing the number of datagrams, is not as big a deal as might at first be assumed.

6. Conclusion

6.1. Summary

This work has shown that it is indeed already feasible to implement basic PQ security measures, especially in a setting where key exchanges do not happen too frequently. VPN software is therefore a perfect starting point, whereas TLS with ephemeral PQ keys may be a long time in coming.

At the moment, the cost in performance is still high though. Reasonably, application developers might therefore want to give users the choice to enable or disable PQ cryptography. Depending on the use case, the threat model may not even include attacks that far into the future. Transmitted data can lose its value fast enough that PQ security is not of concern.

Furthermore, for almost any use case, it is probably too expensive to implement security against active quantum adversaries. The threat is just not there yet, and the additional cost is significant.

6.2. Future Work

Results from this work can be used as a basis for estimating the cost of including PQ measures into key exchanges. From this point one can decide, for a specific use case, whether the gain in security against possible future attacks is worth the additional cost in computation and transmission times today.

What is still missing is a formal proof for the proposed PQ handshake protocols. This could probably be done by building upon the work of Donenfeld and Millner [16]. The Tamarin model could be extended, analogous to the way the code was extended to implement the protocols.

Regarding implementation, the proof of concept given definitely has to be regarded as such. It has to be cleaned up, optimized for performance, and undergo security review, before it can be considered production-ready. Also, datagram splitting on the

application layer and one of the methods for preventing memory exhaustion attacks still have to be implemented.

Furthermore, research in the applications of post-quantum cryptography needs to be continued. Especially efforts at cryptanalysis of all the currently proposed cryptographic primitives are necessary. Only this way can the confidence in these methods increase.

Lastly, research into finding another mathematical foundation for PQ cryptographic primitives may be warranted. Code-based methods have the disadvantage of very large public keys, whereas supersingular isogeny methods are very slow to compute. Currently, only the lattice based methods seem to offer a good middle ground. If the shortest vector problem proofs to be not as hard as now assumed, we may be in big trouble if we have no reasonable alternatives.

Bibliography

[1] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, and Yi-Kai Liu. Status report on the first round of the NIST post-quantum cryptography standardization process. `https://doi.org/10.6028/NIST.IR.8240`, Jan. 2019. Accessed November 17, 2019.

[2] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange a new hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327 343, 2016.

[3] Mihir Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In *Annual International Cryptology Conference*, pages 602 619. Springer, 2006.

[4] Daniel J. Bernstein. TAI64, TAI64N, and TAI64NA. `https://cr.yp.to/libtai/tai64.html`, 1997. Accessed November 17, 2019.

[5] Daniel J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207 228. Springer, 2006.

[6] Daniel J. Bernstein. Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? Jan. 2009.

[7] Daniel J. Bernstein. libpqcrypto. `https://libpqcrypto.org/`, 2018. Accessed November 17, 2019.

[8] Ron Bonica, Fred Baker, Geoff Huston, Robert Hinden, Ole Troan, and Fernando Gont. Ip fragmentation considered fragile. Internet-Draft draft-ietf-intarea-frag-fragile-17, IETF Secretariat, Sep. 2019. `http://www.ietf.org/internet-drafts/draft-ietf-intarea-frag-fragile-17.txt`.

[9] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353 367. IEEE, 2018.

[10] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In Cláudio L. Lucchesi and Arnaldo V. Moura, editors, *LATIN'98: Theoretical Informatics*, pages 163 169, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[11] Cloudflare. Boringtun. `https://github.com/cloudflare/boringtun`, Mar. 2019. Accessed November 17, 2019.

[12] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of SIDH public keys. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 679 706. Springer, 2017.

[13] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology AFRICACRYPT 2018*, pages 282 305. Springer International Publishing, 2018.

[14] Stephen Deering and Robert M. Hinden. Internet protocol, version 6 (ipv6) specification. STD 86, RFC Editor, July 2017.

[15] Jason A. Donenfeld. WireGuard: Next generation kernel network tunnel. In *Proceedings of the 2017 Network and Distributed System Security Symposium*, NDSS'17, San Diego, CA, USA, Feb. 2017.

[16] Jason A. Donenfeld and Kevin Milner. Formal verification of the WireGuard protocol. `https://www.wireguard.com/papers/wireguard-formal-verification.pdf`, 2017. Accessed November 17, 2019.

[17] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212 219, New York, NY, USA, 1996. ACM.

[18] Brook Heisler. Criterion.rs. `https://github.com/bheisler/criterion.rs`, Mar. 2014. Accessed November 17, 2019.

[19] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267 288. Springer, 1998.

[20] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19 34. Springer, 2011.

[21] Julian Kelly. A preview of bristlecone, Google's new quantum processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`, Mar. 2018. Accessed November 17, 2019.

[22] Robert J. McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114 116, 1978.

[23] National Institute of Standards and Technology. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf, Dec. 2016. Accessed November 17, 2019.

[24] National Institute of Standards and Technology. Round 2 submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions, 2017. Accessed November 17, 2019.

[25] Deborah Netburn. Q&A: Google claims 'quantum supremacy.' what could that mean for the future of computing? https://www.latimes.com/science/story/2019-10-23/quantum-supremacy-google-computers, Oct. 2019. Accessed November 17, 2019.

[26] Paul C. Van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of cryptology*, 12(1):1 28, 1999.

[27] Trevor Perrin. The Noise protocol framework. https://noiseprotocol.org/noise.pdf, Jul. 2018. Accessed November 17, 2019.

[28] John M. Pollard. A Monte Carlo method for factorization. *BIT Numerical Mathematics*, 15(3):331 334, Sep. 1975.

[29] Ramya Raghavendra and Elizabeth M. Belding. Characterizing high-bandwidth real-time video traffic in residential broadband networks. In *In WiOpt '10: Proceedings of the 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, pages 597 602, 2010.

[30] Open Quantum Safe. liboqs. https://github.com/open-quantum-safe/liboqs/, Aug. 2016. Accessed November 17, 2019.

[31] Colleen Shannon, David Moore, and Kimberly C. Claffy. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Transactions on Networking (TON)*, 10(6):709 720, 2002.

[32] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, SFCS '94, pages 124 134, Washington, DC, USA, Nov. 1994. IEEE Computer Society.

[33] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746 2751, Oct. 1999.

A. liboqs Benchmark

The following are results from running the benchmark code in the liboqs project on the reference system mentioned in the Results chapter. It compares all the Kyber, NewHope, NTRU, and SABER primitives. The table has been edited to only contain the most relevant columns.

Operation	Iterations	Time (us): mean	pop. stdev
Kyber512-90s			
keygen	83297	36.016	1.256
encaps	66960	44.803	1.484
decaps	55860	53.706	1.664
Kyber768-90s			
keygen	48125	62.338	1.732
encaps	40487	74.099	1.878
decaps	34578	86.761	1.951
Kyber1024-90s			
keygen	31879	94.107	2.238
encaps	27874	107.630	2.466
decaps	24274	123.593	2.606
NewHope-512-CCA			
keygen	59839	50.135	1.411
encaps	41474	72.336	1.601
decaps	37665	79.651	1.929
NewHope-1024-CCA			
keygen	30112	99.631	10.367
encaps	20542	146.049	2.969
decaps	18554	161.697	3.028
NTRU-HPS-2048-509			
keygen	337	8915.852	42.533

encaps	17425	172.175	3.167
decaps	6907	434.377	6.614
NTRU-HPS-2048-677			
keygen	191	15733.529	36.640
encaps	10373	289.233	4.867
decaps	3989	752.110	9.586
NTRU-HPS-4096-821			
keygen	131	23048.802	55.053
encaps	7269	412.725	6.486
decaps	2733	1097.829	13.797
NTRU-HRSS-701			
keygen	179	16800.654	42.811
encaps	10629	282.271	4.431
decaps	3700	810.994	10.958
LightSaber-KEM			
keygen	46097	65.081	1.759
encaps	33484	89.597	2.035
decaps	26999	111.116	2.403
Saber-KEM			
keygen	22565	132.954	2.432
encaps	17658	169.902	3.119
decaps	14750	203.391	3.760
FireSaber-KEM			
keygen	13298	225.613	3.856
encaps	10853	276.423	4.624
decaps	9364	320.395	4.918

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 18. November 2019 .....