
Cyber Security (CSE)


Cyber Security and Robustness (CSR)

Thesis

# Quantum-safe TOR

**Post-Quantum Cryptography**

**TUJNER Zsolt**
Promo 2020 – Master EIT Digital Cyber Security

Academic Supervisor: Melek Önen PhD, EURECOM
Industrial Supervisors: Maran van Heesch MSc, TNO
Thomas Rooijakkers MSc, TNO

August 30, 2019


**DECLARATION POUR LE RAPPORT DE STAGE**

***DECLARATION FOR THE MASTER'S THESIS***

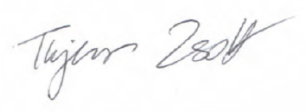
Je garantis que le rapport est mon travail original et que je n'ai pas reçu d'aide extérieure.
Seules les sources citées ont été utilisées dans ce projet. Les parties qui sont des citations directes ou des paraphrases sont identifiées comme telles.

*I warrant, that the thesis is my original work and that I have not received outside assistance.*
*Only the sources cited have been used in this report. Parts that are direct quotes or paraphrases are identified as such.*

À Biot, in Biot
Date : August 30, 2019

Nom Prénom : TUJNER Zsolt
*Name First Name*

Signature :



# Acknowledgments

I would like to express my gratitude to EIT Digital for the opportunity that allowed me to have this unique Master study experience.

Furthermore, I would like to express my sincere gratitude to TNO and my industrial supervisors Maran van Heesch and Thomas Rooijakkers for their expertise, ideas, mentoring and patience. I would also like to thank my academic supervisor Melek Önen for her expertise, ideas and supervision of my thesis.

Finally, I would like to thank my family, my friends in Hungary and the new friends I made during the studies for their support, that made it possible for me to complete this journey.

# Contents

**Abstract**

Nowadays there is vast information publicly available on the internet and it is ever-expanding. This knowledge should be available to anyone, but some governments or organizations consider the knowledge gained by information unwanted. They can abuse their power by blocking access to the websites on the internet thus, depriving people of their free will. These oppressive regimes also monitor internet traffic and people who are out of line may be severely punished. The Onion Router (TOR) network aims to grant people in such abusive areas anonymity on the internet, but with the emerge of quantum computers this anonymity is threatened.
In this work, we give insight into a quantum-safe TOR network that grants anonymity in the quantum world. To do so, we examined which parts of TOR would become vulnerable in a quantum world. We identified that the symmetric key, and the long-term, medium-term keys generated by asymmetric cryptography are the most concerning cryptographic challenges TOR faces in order to preserve forward secrecy. We suggest changes to these parts and predict drawbacks regarding the changes. To prove our predictions, in our experiments we used an existing implementation that simulates the behaviour of TOR. For benchmarking, an implementation called `SweetOnions` was used, later this implementation was updated by adding quantum-safe schemes. Purely quantum-safe and hybrid implementations were both tested. The experiments performed measured CPU cycles required for key generation, message encapsulation and decapsulation, and circuit creation. Message sizes were also measured to see how quantum-safe cryptography impacts network traffic. Message sizes pose the biggest challenge for TOR, because sending too many packets can cripple the whole network. Finally, based on our results, we show that quantum-safe TOR is possible and suggest two versions - one that can be used in a purely quantum-safe setting, and one that can be used in a hybrid setting.

De nos jours, un nombre d'informations en constante expansion sont disponibles publiquement sur l'Internet. Ces connaissances devraient être accessibles à tous, mais certains gouvernements ou organisations considèrent que les connaissances acquises grâce à ces informations sont indesirables. Ils peuvent abuser de leur pouvoir en bloquant l'accès aux sites Web sur l'Internet, privant ainsi les personnes de leur libre arbitre. Ces régimes oppressifs surveillent également le trafic Internet et les personnes qui sortes des limites peuvent être sévèrement punies. Le réseau "The Onion Router" (TOR) a pour objectif d'octroyer l'anonymat sur Internet à des personnes se trouvant dans de telles zones d'abus, mais avec l'émergence des ordinateurs quantiques, cet anonymat est menacé. Dans ce travail, nous donnons un aperçu d'un réseau de sécurité quantique qui garantit l'anonymat dans le monde quantique. Pour ce faire, nous avons examiné quelles parties de TOR deviendraient vulnérables dans un monde quantique. Nous avons constaté que la clé symétrique et les clés à moyen et à long terme générées par la cryptographie asymétrique constituaient les défis cryptographiques les plus préoccupants pour TOR afin que le secret soit préservé. Nous suggérons de modifier ces éléments et de prévoir les inconvénients liés à ces changements. Pour prouver nos prédictions, nous

avons utilisé dans nos expériences une implémentation existante qui simule le comportement de TOR. Pour l'analyse comparative, l'implémentation appelée `SweetOnions` a été utilisée. Cette implémentation a ensuite été mise à jour en ajoutant des schémas ) quantique-sécuriser. Des implementations purement quantiques et hybrides ont été testées. Les expériences effectuées ont mesuré les cycles de processeur nécessaires à la génération de clé, à l'encapsulation et à la décapsulation de messages et à la création de circuits. La taille des messages a également été mesurée pour déterminer l'impact de la cryptographie quantique sur le trafic réseau. La taille des messages constitue le plus gros défi pour TOR, car l'envoi d'un trop grand nombre de paquets peut paralyser tout le réseau. Enfin, sur la base de nos résultats, nous montrons qu'il est possible que TOR devienne quantique-securisé et proposons deux versions: une pouvant être utilisée dans un environnement purement quantique-sécuriser et une autre, dans un environnement hybride.

# 1 Introduction

Nowadays information available online is expanding in an unforeseen way, a vast amount of data is uploaded and shared through social media, IoT, etc. [23]. However, this data also attracts unwanted attention and might paint a bad image of some stakeholders. Consider the case of Edward Snowden who put the National Security Agency (NSA) in the spotlight by shedding light on how the American population was wiretapped [22]. When blowing the whistle on such a large scale one would aim to remain anonymous as this act can negatively affect the career and freedom of the individual. In oppressive regimes, where the freedom of speech is abused this is even more serious, as any type of negative speech, whistleblowing or expressing freedom of information may be recognized as an act of treason and punishments are severe.

The Onion Router (TOR) aims to grant its users anonymity when accessing the internet. In principle, when using TOR the messages or website connection requests are sent through a network of relays and after multiple 'hops' reach their destination. So, if Alice wants to send Bob a message, but does not want an eavesdropper to know that she initiated the contact, Alice can use TOR. The cryptographic schemes used today and in TOR are based on hard mathematical assumptions e.g. Discrete Logarithm Problem and integer factorization. These schemes are assumed to be secure against classical adversaries as solving them with the currently known algorithms cost exponential time, but with a quantum computer solving these problems become feasible.

In the mid 1990s, due to the rapid development of quantum technology, Lov Grover and Peter Shor proposed two algorithms for quantum computers that now are gaining more attention again. The algorithm proposed by Shor significantly speeds up evaluation of mathematical problems like factorization on a quantum computer. This breaks most of the current commonly used asym-

metric key cryptographic schemes [34]. The algorithm of Grover finds pairs in databases faster than a classical computer, it can be considered as a search algorithm and it weakens symmetric key cryptography [13]. With all the efforts put into building quantum computers the emerge of quantum computers seems inevitable. This means that the currently widely used cryptographic schemes will become obsolete and TOR will no longer provide anonymity. Therefore, there will be a need to switch to so called quantum-safe cryptographic schemes.

The transition from current - classical - cryptography needs to be started as fast as possible, but it is expected to have significant effects on IT infrastructure. This is due to the heavier operations quantum-safe cryptography requires reported by [33] for setting up connections. Furthermore, network load is also expected to increase as message sizes are bound to get larger due to the increased encryption sizes. TOR is run by volunteers across the globe and it uses the most common cryptographic schemes of today, so making it quantum-safe is a must do task and the effects of changing the cryptography are going to be felt. Not only by the volunteers running relays but also the average user who connects to the network is going to experience these drawbacks.

In this thesis we design a quantum-safe TOR. To do this, we investigate what parts of TOR need to be changed and how urgent the change is, the effects on user experience and provide recommendations on what the quantum-safe TOR anonymity network should look like.

## 1.1 Research goal

The goal of this research is to see how TOR can work in the quantum world. To help answering this question, we consider four sub-questions to aid our research. Answering these sub-questions makes it possible in the end to answer the goal of the research:

1. Which are the most pivotal parts in TOR that need to be made quantum-safe first?
2. Where and when should quantum-safe cryptography be added to TOR?
3. What performance drawbacks are users likely to face in a quantum-safe TOR and are these drawbacks confirmed by the experimental results?
4. What are the strengths and weaknesses of a quantum-safe TOR?

## 1.2 Outline

In Section 2 the preliminaries for this thesis can be found, these include cryptography: symmetric and asymmetric key, key exchange methods, and digital signature schemes. Furthermore, security definitions and levels of security are introduced for classical and quantum computing. Post-quantum cryptography

is also introduced in this section. In Section 3 the TOR network is introduced, including connection setup, the cryptographic building blocks and known issues. Attack surfaces on TOR with security guarantees are also discussed in this section. Section 4 explains what changes are needed to be made to TOR to become quantum-safe. Furthermore, the challenges faced during the transition to quantum-safe TOR are explained. In Section 5 we explain how the experiments were executed. The experimental results for key generation and circuit build times for quantum-safe TOR are discussed, and recommendations are given about what a quantum-safe TOR should look like. In the final section - Section 6 - the work is concluded with possible open problems and future research on the topic.

## 1.3 Related work

Before talking about TOR, first we need to investigate how a user can achieve anonymity online. Nowadays a user can choose from a variety of solutions to become anonymous on the internet [21]. When it comes to anonymity there are two techniques that are considered: high-latency and low-latency anonymity systems [14].

High-latency systems are also called mix-networks, or mixnets for short. Mixnets usually add a batching and mixing process to the messages entering the network adding latency, hence the name high-latency. This makes the linking of messages at the exit of the network hard. Two mixnet examples are Mixminion [24] and Mixmaster [17].

Low-latency systems do not modify the packet delay to achieve end-to-end latency as high-latency systems do. Rather, they pass the messages through multiple relays to achieve anonymity. One example for low-latency system is the Invisible Internet Project (I2P) [16, 2] that uses multiple relays to provide anonymity while users can chat with each other. Another example is the Java Anon Proxy (JAP) with which users can browse the internet anonymously.

TOR is a low-latency system, that provides both anonymous browsing and communication between users in a secure manner on the internet. The original paper on TOR by Roger Dingledinde [11] introduces the design, considerations and security goals of the TOR anonymity network. Furthermore, active and passive attack scenarios are discussed in the paper. From [11] it is clear that TOR faces issues in the quantum world as it uses the most common current cryptographic schemes. In a quantum world these cryptographic schemes are broken by a quantum computer and thus, TOR can no longer provide anonymity.

When it comes to quantum-safe TOR, there are two papers that are interesting: *Post-Quantum Forward Secure Onion Routing* by S. Ghosh and A. Kate [12]; and *Circuit-extension handshakes for Tor achieving forwards secrecy in a quantum world* by J. M. Schanck, W. Whyte, and Z. Zhang [33]. These are both

hybrid solutions - meaning they use a combination of current cryptography and quantum-safe cryptography. In [12] the Ring-Learning With Errors problem is used in their own cryptographic protocol design to achieve security against quantum adversaries. Whereas, in [33] a variant of a quantum-safe scheme (`NTRU`) is used with the current design of TOR. These papers are further analysed in Section 4.

Selecting the best fit candidate to replace current cryptography in a quantum world is not only hard for TOR. A lot of research in quantum-safe cryptography is done already, although the focus of the following papers is not directly quantum-safe TOR. The paper from ETSI [7] gives a lean introduction and overview of quantum-safe cryptography. In [37], D. Stebila et al. propose a key exchange method based on the Learning With Errors problem. J. W. Bos et al. [5] propose a key exchange to make the Transport Layer Security (TLS) protocol quantum-safe. A lot of work is done to design and analyse post-quantum cryptography, e.g. a project from the European Union's Horizon 2020 called FutureTPM is analysing quantum resistant primitives [15]. This project aims to identify suitable quantum-safe algorithms for future adoption.

In 2017, the National Institute of Standards and Technology (NIST) opened a call for proposals on the topic of quantum-safe cryptographic solutions for new quantum-safe standards [28]. The first round contained 69 submissions. On January 30, 2019 the candidates for the second round were announced, consisting of 17 asymmetric key encryption and key-establishment algorithms and 9 digital signature algorithms.

# 2 Preliminaries

In this section the preliminaries required to read this thesis are introduced. The basic cryptographic primitives are explained, followed by notions of security, quantum computing and post-quantum cryptography.

Symmetric key and asymmetric key cryptography, key exchange and key derivation functions, and signature schemes are introduced in Section 2.1. In Section 2.2 the definitions of security, notions of security followed by describing attacker types are explained. Finally, in Section 2.3 post-quantum computing is introduced.

## 2.1 Cryptographic schemes

In cryptography, the two fundamental operations are encryption and decryption. For encryption a plaintext message, an encryption function and an encryption key are needed. By applying an encryption function to the message using the encryption key, the function outputs a ciphertext. Decryption is the reverse operation of encryption. Using the ciphertext and applying the decryption function with the decryption key, the original message should be learned as an output.

When encrypting data, the aim of the encrypting party is to disallow anyone from reading the data who does not have the correct decryption key. Furthermore, the goal is to have the sent data received in the intended - original - form, and not altered by any third party. These two aims are called confidentiality and integrity, which are two pillars of the CIA triad - CIA in this setting stands for confidentiality, integrity and availability.

### 2.1.1 Symmetric key cryptography

When using symmetric key cryptography, the communicating parties agree on a symmetric secret key to use for both encryption and decryption. This key agreement can be done in person as doing this online is not always considered secure.

$$\begin{aligned} \mathrm{Enc}_k(m) &= c, \\ \mathrm{Dec}_k(c) &= m. \end{aligned} \tag{1}$$

In Equation 1 the formal representation of symmetric key encryption and decryption can be seen. Enc and Dec are encryption and decryption functions respectively, $k$ is the agreed key used for encryption and decryption, $m$ is the message and $c$ is the ciphertext after encryption. The most common and widely used scheme for symmetric key cryptography is the Advanced Encryption Standard (AES).

### 2.1.2 Asymmetric key cryptography

Asymmetric key cryptography works differently than symmetric key cryptography. For this scheme the parties each have two keys: a public key, that is known to every participant and a private key, that is only known to the person generating it. Encryption is done using the public key of the recipient, while for decryption the individual private key of the recipient is used. Using the public and private keys ensures that only the person who has the correct private key can decrypt the ciphertext.

As an example, two parties - Alice and Bob - would like to communicate with each other. Alice knows her own public key ($pk_A$), her private key ($sk_A$) and the public key of Bob ($pk_B$). She encrypts her message with the public key of Bob that guarantees her, that only Bob will be able to decrypt the ciphertext correctly, which can be represented as

$$\begin{aligned} \mathrm{Enc}_{pk_B}(m) &= c, \\ \mathrm{Dec}_{sk_B}(c) &= m. \end{aligned} \tag{2}$$

In Equation 2 the encryption of the message $m$ is done with the public key of Bob $pk_B$ to obtain the ciphertext $c$. To decrypt the ciphertext, Bob uses his private key $sk_B$ and obtains the message $m$. The most common scheme used for asymmetric key cryptography is the Rivest–Shamir–Adleman cryptosystem (RSA).

Because of the computational demand of RSA it is less commonly used to directly encrypt user data. Instead, RSA is used to encrypt shared keys used for symmetric key cryptography which has higher performance speed. Another approach to asymmetric key cryptography is Elliptic-curve cryptography (ECC). ECC uses the properties of elliptic curves over finite fields mostly for key exchange and digital signatures. The main benefit of ECC is that a smaller key size on an elliptic curve group could provide the same security level as RSA with large keys.

Asymmetric key cryptography is a good option when parties communicate through an untrusted network and do not have previously agreed symmetric keys. The downside of asymmetric key cryptography is that it is costly compared to symmetric key cryptography.

### 2.1.3 Key Encapsulation and Key Derivation functions

In this subsubsection methods are introduced on how encryption and decryption keys can be distributed between parties. A key encapsulation method (hybrid cipher) is introduced that uses both asymmetric key and symmetric key cryptography, the logic behind key exchange protocols is introduced, and finally key derivation functions are explained.

In the hybrid solution asymmetric key cryptography is used for a Key Encapsulation Mechanism (KEM) and symmetric key cryptography for a Data Encapsulation Mechanism (DEM). If Alice would like to communicate with Bob, she encapsulates a symmetric key $k$ with Bob's public key and sends the ciphertext to Bob. Using his private key, Bob can successfully decrypt the message and obtain the symmetric key that will be used for encrypting the messages.

$$\begin{aligned} \text{KeyGen}(K) &\rightarrow (pk_B, sk_B), \\ \text{Encap}_{pk_B} &\rightarrow (c, k), \\ \text{Decap}_{sk_B}(c) &\rightarrow k. \end{aligned} \tag{3}$$

The construction of an encapsulation method can be seen formally in Equation 3 and is as follows

1. Bob generates a public, private key pair from the domain $K$ using a key generation function,
2. If Alice wants to communicate with Bob, she generates a symmetric key $k$ and encapsulates it with the public key of Bob ($pk_B$) to obtain a ciphertext $c$,
3. Alice sends the ciphertext $c$ to Bob, who can decapsulate it using his private key ($sk_B$),
4. Now Bob and Alice share a symmetric key $k$, that they can use for symmetric encryption and decryption as in Equation 1.

Another way to create and use symmetric keys is by using key exchange protocols. These key exchange protocols - or KEXs - generate a shared secret between two parties that wish to communicate. Using the shared secret, the parties can use a Key Derivation Function to obtain a symmetric key that can be used for encryption and decryption. The construction of such a secret between two parties - Alice and Bob - is the following

1. Alice, using her private key ($sk_A$) and Bob's public key ($pk_B$) calculates the shared secret $\text{S}_\text{A}$,

$$\text{S}_\text{A} = f(sk_A, pk_B).$$

2. Bob does the same, using his private key ($sk_B$) and Alice's public key ($pk_A$) to calculate the secret $\text{S}_\text{B}$

$$\text{S}_\text{B} = f(sk_B, pk_A).$$

The function $f$ is chosen at the start of the key exchange scheme and known to both parties. After following the protocol, the secrets match ($S_A = S_B$) and can be used as the symmetric key for encryption and decryption.

Currently, the most widely used key exchange protocols are the Diffie-Hellman (DH) and the Elliptic Curve Diffie-Hellman (ECDH). In the past TOR also used the Diffie-Hellman key exchange protocol, but due to performance considerations it was replaced by Curve25519 - a type of Elliptic Curve cryptography.

**Key Derivation Functions**

Parties having the shared secrets may want to use the secret directly as a symmetric key. This can be problematic as symmetric key schemes assume a uniform key distribution. The problem arises if the function to calculate the shared secret is deterministic. In this case the scheme is not secure, and adversaries can always guess the symmetric key by having the encapsulation. Another possible issue with using the shared secret as the symmetric key relates to the size of the secret. It can be either too long or too short for it to become the symmetric key. As the current standard for symmetric cryptography AES uses key sizes of 128, 192 or 256 bits, the symmetric keys need to meet these criteria.

To overcome the issue for having the correct key format a Key Derivation Function (KDF) can be used. A KDF can be regarded as a Pseudo-Random Function, but for this specific use the input and output of the function is of arbitrary length. Cryptographic hash functions are the most common KDF functions as they are considered being one-way and meet certain properties:

1. Preimage Resistance: Given a cryptographic hash function $H$ and an output $y$ it should be infeasible to find $x$ such that $H(x) = y$,
2. Second Preimage Resistance: Given $x$, it should be hard to find $x' \neq x$ such that $H(x') = H(x)$,
3. Collision Resistance: Finding two distinct values $x$ and $x'$ ($x \neq x'$) such that $H(x) = H(x')$ should be infeasible.

Hash functions are widely used in cryptography as they are efficient to compute when generated, but hard when attempted to be reversed. A common use of hash functions is to ensure data integrity, that is explained in subsection 2.1.4.

### 2.1.4 Signature schemes

At this point we understand how two parties can achieve data confidentiality by using asymmetric key or symmetric key cryptography. But this is not enough if an adversary manages to intercept the communication and manages to modify the messages sent through the channel. The communicating parties would have no knowledge that in reality they are receiving messages from an eavesdropper.

The aim now is to achieve message integrity, which is the second pillar of the CIA triad.

We discuss two ways to achieve this:

- Message Authentication Codes (MACs), and
- Digital Signatures

In order to use a MAC, the two communicating parties need to share a secret key. Using this key, they can ensure that the data has not been tampered with by producing a tag with a MAC algorithm;

$$\text{MAC}_k(m) = \text{tag}.$$

where MAC is the function that produces the tag, $k$ is the shared secret key, and $m$ is the message they wish to keep intact.

Next to the MAC function it is essential that the receiver can verify if the message sent is still intact. A Verify function takes a tag and a message as an input and using the secret key, it outputs valid/invalid based on the tag and message matching;

$$\text{Verify}_k(\text{tag}, m) = \text{valid/invalid}.$$

Since MACs require a shared secret key with each party, it is not the best option for checking data integrity as having a shared secret key prior to the message exchange is not always possible. Rather asymmetric key cryptographic methods are used called Digital Signatures, as these rely on public-private key pairs.

If Alice wants to send a message to Bob, she can sign it with her private key to obtain the signature. Then she sends her message and her signature to Bob. Now Bob can use Alice's public key, the message he received from Alice and the signature that Alice provided to validate if it was really Alice who signed the message.
Similar to MACs, digital signature schemes also work with two algorithms:

- An algorithm to sign the message with the sender's private key ($sk$)
- An algorithm to verify the message with the sender's public key ($pk$)

Upon sending the message, Alice computes

$$\text{Sign}_{sk_A}(m) = \text{sig},$$

and sends ($m$, sig) to Bob. When receiving this from Alice, Bob can verify the message integrity using Alice's public key by computing

$$\text{Verify}_{pk_A}(m, \text{sig}) = \text{valid/invalid}.$$

## 2.2 Security

In this subsection topics related to security are introduced, specifically, definitions of security, $n$-bit security and attacker models. The security definitions define the security model and the assumed capabilities of an adversary. The $n$-bits of security explain how the security of cryptographic schemes is measured, while attacker models depend on the goals and capabilities of attackers.

### 2.2.1 Defining security

In [19, 35] security is defined by defining the security of encryption schemes. According to [35], an encryption scheme is information-theoretic secure if an adversary with infinite computing power cannot break the scheme, this is the definition of perfect security. It can be called a one-way function, where the adversary is given a publicly available function and is asked to invert the function on an element of the challenger's choosing - hash functions are a good choice for this as they are considered information-theoretic secure. On the other hand, an encryption scheme is considered computationally secure if a polynomially bounded adversary cannot break the scheme, this is also called semantic security. On top of perfect security and semantic security, there is another security definition for polynomial security called IND security, where IND stands for indistinguishability. To prove that a scheme has IND security, we have to show that no adversary can win the following "find and guess" game with greater probability than $\frac{1}{2}$;

1. Find: an adversary creates two messages of equal length denoted by $m_0$ and $m_1$.

2. Guess: the adversary is given an encryption of one of the messages denoted by $c$. The adversary has to correctly guess with a probability greater than half, if the encrypted message was $m_0$ or $m_1$.

If an encryption scheme achieves `IND` security, it is also considered semantically secure.

When attacking these schemes an adversary requires some attacker capabilities. For this reason, in the minimum security game the adversary has access to an encryption oracle. This encryption oracle serves as a 'black box' that the adversary can use to perform encryption on plaintexts of her choosing. This attack is called a chosen plaintext attack (CPA). Another type of chosen plaintext attack available is the adaptive chosen plaintext attack (CPA2). In this setting the adversary can decide from the list of inputs what to encrypt based on previous plaintexts and their corresponding ciphertexts [4].

In a more complex setting the adversary has access to a decryption oracle as well. Using this decryption oracle, she can decrypt ciphertexts of her own choosing, but to make the security game non-trivial, the adversary cannot request to decrypt the challenger's ciphertext by the oracle. This attack is called a chosen

ciphertext attack (CCA).

CCA attacks have a weaker definition for an attack compared to the original, called lunchtime attacks (CCA1). In this setting, the adversary has only limited access to the decryption oracle (during lunchtime). At a later time, the adversary is given a ciphertext and she has to decrypt it or try to learn about the plaintext without using the decryption oracle.

In the event that a cryptographic scheme is compromised, we would like to be sure that this does not have any effect on the secure communications we had in the past. This security notion is called forward secrecy and is defined by [35] as *"A system is said to have forward secrecy if compromising of a long-term key at some point, in the future does not compromise the security of communications made using that key in the past."*

By adding forward secrecy to the security requirements of a cryptographic scheme we can ensure that only the sessions that are using the current keys are compromised. When learning about the compromise, changing the keys mean that the adversary has to compromise the system again. Furthermore, our past connections also remain secure, as the compromised keys do not help adversaries to learn about past communications and data.

### 2.2.2 $n$-bit security

The most common way to measure the strength of a cryptographic primitive is in the number of bits of security. It is denoted by $n$-bit security, where $n$ stands for $2^n$ operations that an attacker needs to perform to break the security of a cryptographic primitive.

As mentioned in Section 2.1, the most commonly used cryptographic schemes are AES for symmetric key cryptography, and RSA, ECC and DH for asymmetric key cryptography. AES uses the same key for both encryption and decryption. This key can be size 128, 192, or 256 bits.

For asymmetric cryptography the case is different, as it relies on hard mathematical problems (e.g. ECC and DH rely on the discrete logarithm problem, and RSA relies on the integer factorization problem). RSA works with the integer factorization problem, using two large prime numbers that are kept secret a public and private key is generated. Anyone can use the public key to encrypt, and if it is large enough it is assumed to be infeasible for a computationally bounded adversary to decrypt the message without having some knowledge about the two prime numbers. Table 1 provides an overview of the security bits of the above-mentioned schemes.

| Cryptography | Algorithm | Key size (bits) | # bits of security |
|---|---|---|---|
| symmetric | AES | 128 | 128 |
| | AES | 192 | 192 |
| | AES | 256 | 256 |
| asymmetric | RSA | 1 024 | 80 |
| | RSA | 2 048 | 112 |
| | RSA | 3 072 | 128 |
| | RSA | 7 680 | 192 |
| | RSA | 15 360 | 256 |
| | ECC | 256 | 128 |
| | ECC | 384 | 192 |
| | ECC | 511 | 256 |

Table 1: Symmetric (AES) and asymmetric (RSA, ECC) cryptographic schemes with key sizes and security bits.

### 2.2.3 Attacker models

When discussing attacker models, we distinguish between two types of attackers: active and passive. When it comes to active attacks an adversary is allowed to interfere with the communication between the communicating parties - Alice and Bob; a passive attacker cannot.

Imagine a third party - Eve - who is eavesdropping in the communication and sends modified messages. Eve can join the communication as a relay between Alice and Bob. Eve may establish keys with both Alice and Bob, so when Alice communicates with Bob, in reality Alice is communicating with Eve who is intercepting messages that she can read and modify. This kind of attack is called a 'man-in-the-middle'-attack. Eve is also capable of blocking traffic between participants, this kind of attack is called Denial of Service (DoS) attack.

In a passive attack the attacker is not allowed to interfere with the communication, she is only allowed to listen to the communication channel. This limits attack surfaces, but storing the communication captured is a possibility. Once it is stored, the adversary can decrypt the data in the future when the needed computational power will be affordable. This is a worry as we are getting closer to quantum computing. A quantum adversary may be able to decrypt the stored data. This is called the 'store now, decrypt later' paradigm.

As a result, more pressure is put on cryptographers to introduce quantum-safe implementations, that will keep data secure against quantum adversaries.

## 2.3 Post-quantum cryptography

Post-quantum cryptography or quantum-safe cryptography - as suggested by its name - focuses on the cryptography after quantum computers are available. The goal is to have cryptography that is capable to withstand adversaries with quantum computing power even on classical computers. The most widely used asymmetric key cryptographic schemes (RSA, ECC) are going to become insecure against quantum adversaries. Symmetric cryptographic schemes also need to be considered for change. If transition to quantum-safe solutions is not done, current cryptographic schemes become vulnerable to quantum adversaries. This is why quantum-safe schemes are being designed, implemented and tested, so currently used cryptographic schemes can withstand a quantum adversary's attack. These quantum-safe schemes are based on mathematical problems that are hard to compute even for quantum computers.

In 1994 Peter Shor formulated a quantum algorithm [34] (that can run on a quantum computer) for solving the integer factorization problem. On a quantum computer this algorithm runs in polynomial time, which is an exponential speed up compared to classical computers, that can solve factorization problems in exponential time. This is a concern for asymmetric cryptography as it based on the integer factorization problem. With a quantum computer running Shor's quantum algorithm breaking the encryption becomes feasible.

Another notable advancement in quantum computing was done by Lov Grover, who proposed an algorithm that can be considered as a black-box search algorithm [13]. It can find a specific value in an unordered database. Unlike Shor's algorithm the speed up is "only" quadratic, but this means that brute-forcing symmetric key encryptions can be done faster. Brute-forcing a 128-bit AES encryption would take about $2^{64}$ iterations with Grover's algorithm instead of $2^{128}$. In the extended Table 2 an overview of security bits of currently used schemes can be found against classical and quantum adversaries.

Based on Table 2, it is clear that asymmetric key cryptography will be heavily affected in the quantum world, but keep in mind that symmetric key cryptographic schemes also need to be adjusted. Symmetric key cryptography can remain secure but requires using larger key sizes to achieve the current security guarantees.

The five most common types of post-quantum cryptographic schemes are

- Code-based,
- Lattice-based,
- Isogeny-based,
- Hash-based,

- Multivariate.

| Cryptography | Algorithm | Key size | # bits of security | |
|---|---|---|---|---|
| | | | classical | quantum |
| symmetric | AES | 128 | 128 | 64 |
| | AES | 192 | 192 | 96 |
| | AES | 256 | 256 | 128 |
| asymmetric | RSA | 1024 | 80 | - |
| | RSA | 2048 | 112 | - |
| | RSA | 3072 | 128 | - |
| | RSA | 7680 | 192 | - |
| | RSA | 15360 | 256 | - |
| | ECC | 256 | 128 | - |
| | ECC | 384 | 192 | - |
| | ECC | 511 | 256 | - |

Table 2: Symmetric and asymmetric cryptographic scheme security in a quantum world. Schemes marked with '-' are broken in polynomial time using a quantum computer.

### 2.3.1 NIST submissions

As already mentioned in Section 1, the National Institute for Standards and Technology (NIST) has called to arms experts working in cryptography to address the challenge to find candidates for quantum-safe cryptography [28]. Currently, the second round of submissions is being evaluated to find suitable schemes from the 17 candidates to replace asymmetric cryptography - RSA, Diffie-Hellman, Elliptic-Curve Cryptography [8].

The security level of the quantum-safe schemes can be measured in bits - just as with the case of classical computing. The NIST defined a metric that categorizes quantum-safe submissions into different categories depending on how secure they are. The submissions for the second round focus on three levels of security:

- Level 1 security is achieved when a scheme is considered 128-bit secure against a quantum adversary,
- Level 3 security is achieved when a scheme is considered 192-bit secure against a quantum adversary,
- Level 5 security is achieved when a scheme is considered 256-bit secure against a quantum adversary.

All three levels of security are assumed to be secure, but they cannot be fully tested in practice. Instead, because quantum computing capabilities are not known yet, cryptoanalysis is done on these schemes.

Achieving level 3 and level 5 security is a challenging task. The size of public keys, private keys, and ciphertexts usually get larger with increasing the security level. The increasing size of keys result in heavier calculations required for key generation, encapsulation, and decapsulation (an example for such a scheme is `Frodo` [1]). In [33] the technique used is the `NTRUEncrypt` scheme to achieve 128-bit (level 1) security for the TLS connection against a quantum adversary. In this thesis we aim to reach level 1 security against quantum adversaries. If this statement holds, then we can confirm that TOR is quantum-safe and can be used in the quantum world.

### 2.3.2 Transitioning to Post-quantum cryptography

The transition to quantum-safe cryptographic schemes is expected to be a lengthy process. This is due to the fact that cryptographic schemes are never fully tested, rather over time they become trusted. When schemes become well-researched and remain safe against attacks over time they are considered secure. If change from current cryptography to quantum-safe cryptography would be done quickly, without sufficient research and testing, there would be no scheme to fall back to when flaws were discovered in quantum-safe schemes. Instead, hybrid schemes are used - this is different from the KEM/DEM hybrid discussed in Section 2. In a hybrid scheme, currently used secure cryptographic schemes are combined with quantum-safe schemes to form a hybrid. This guarantees security against classical adversaries in case the quantum-safe part of the scheme is broken.

Hybrid schemes do not come without drawbacks. These drawbacks include increased bandwidth, the difficulty of code and key management, and multiple systems could all be using different hybrid schemes. Making these systems work together is challenging task. The adoption to hybrid schemes must be done in a cautious manner, not by crippling network traffic or create a chaos in key management.

# 3 TOR network

The Onion Router or TOR for short is an anonymity network operated by volunteers all around the globe. The roots of TOR go back to the 90s, when the concept of onion routing was introduced by United States Naval Laboratories [38]. The goal was to protect foreign U.S. intelligence communications from being intercepted and linked to source and destination. Later it was further developed by the Defense Advanced Research Projects Agency (DARPA). In 2004, Roger Dingledine, Nick Mathewson and Paul Syverson have published the first paper on the TOR network titled *Tor: The Second-Generation Onion Router* [11]. The network they propose claims to grant anonymity for the users between their client and the content they are accessing on the internet. They achieve this anonymity by using a set of intermediate nodes between the source (user) and the destination (website).

So, if Alice wants to send Bob a message, but does not want an eavesdropper to know that she initiated the contact, Alice can use TOR. Consider the following scenario, where Charles, David and Greg provide a TOR-like service. Alice sends a request to Charles to send a message to Bob. Charles does not know how to contact Bob, so he asks David to contact Bob and send him the message. David also does not know how to contact to Bob, so he sends the message to Greg, who knows Bob. Finally, Greg contacts Bob and delivers him the message. This is a simple example about the concept of TOR, but from this example it is clear that an eavesdropper does not know that Alice sent the message to Bob - the eavesdropper thinks it was Greg. Only multiple eavesdroppers working together could learn that it was Alice who sent the original message.

This is different to the traditional internet model, as there the traffic goes through a public network, and source can be linked with the destination. One solution that can be used to hide identity is a proxy service, although using a proxy does not resolve the issue completely. A proxy is a single point that is used for both entry and exit. When it comes to routing and an adversary controls the proxy service, the destination a user wishes to contact can directly be seen. Monitoring in- and outgoing traffic is also easy to do and linking source and destination is straightforward. Figure 1 shows a standard connection on the internet, in comparison Figure 2 shows a default connection in TOR.

The following sections give an overview of the TOR network. More specifically in Section 3.1 the technical side of TOR is introduced, Section 3.2 explains the building blocks and keys in more detail. Finally, in Section 3.3 the issues and attack surfaces are discussed.

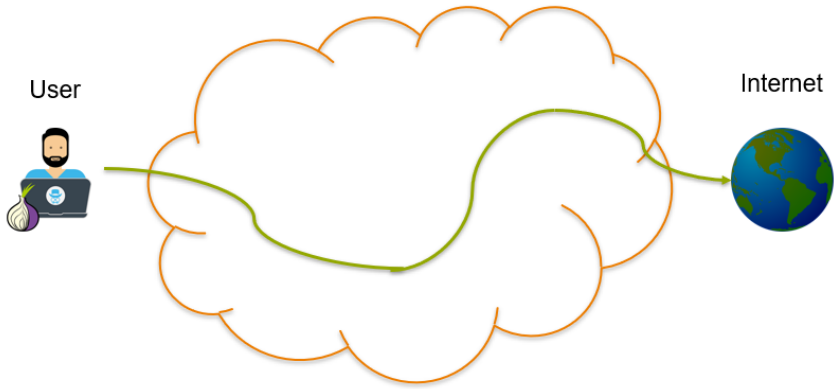



Figure 1: A standard connection on the internet.

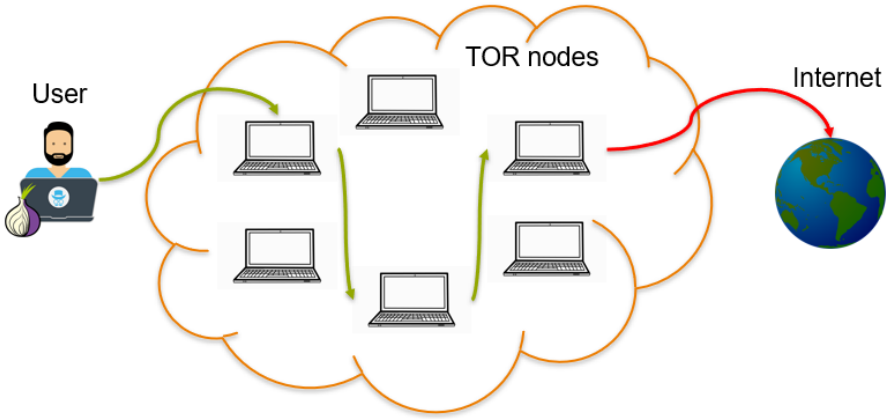



Figure 2: A default TOR connection.

## 3.1 Introduction to TOR

In TOR, when a message is sent through the network it is passed through a number of nodes before finally reaching its destination. This form of transporting a message makes it hard for adversaries to link source and destination together. The default number of nodes TOR uses for message transmission is three. Currently there are around 7 000 nodes in TOR and the way the network is governed is introduced in the next section.

### 3.1.1 Nodes in TOR

In order to keep a controlled state of the vast network of TOR, there are so called Directory Authorities (DA). These are hosted by trusted individuals who have worked on and are close to TOR. There are a total of nine DAs and they

are spread around the world - four of them are in The United States, and five in Europe. The DAs know about all of the available nodes in the network and their main function is to store, check, and verify the state of the network. The nodes controlled by the DAs are called directory nodes. A list of current TOR nodes is publicly available online [9]. Each node has to publish a descriptor to the DAs, containing information about the node like IP address, bandwidth, country of origin, public key, etc. The nodes need to sign this document and send it to the DAs proving that it is their own information. When the DAs received this document, they have a vote on the state of the network. This is the reason why there is an odd number of DAs, there cannot be a tie. Once they agree on one state a document called consensus is published and all of the DAs have this same document.

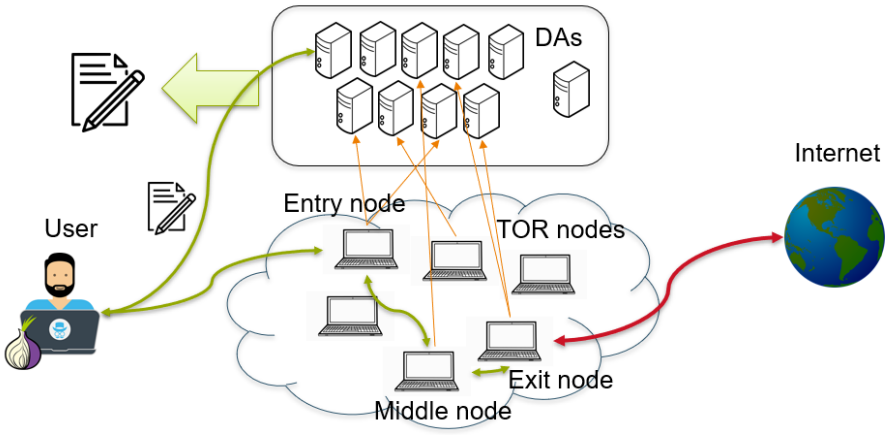



Figure 3: An overview of TOR containing the nine DAs, the Bridge Authority, and the TOR relays.

As mentioned earlier in Section 3.1 the default number of nodes TOR uses is three. These nodes have specific names: entry node, middle node, and exit node. When a user wants to connect to the TOR network, she must use one of the entry nodes to do so. As there is access to the network, the entry node forwards the message to the middle node. Finally, the middle node forwards the message to the exit node, where the message leaves the network and is directed to the destination. The nodes are chosen by the TOR software based on the consensus document. This design consideration is to prevent adversaries to be able to directly link source and destination. The entry node knows about the source of the message, but only knows the middle node and it is not the destination. The middle knows the entry and exit nodes, but neither of them are the source or the destination. As for the exit node, it knows the destination of the message, but does not know the source. Knowing the entry and exit nodes enable adversaries to deanonymize users, but the middle node makes the task more complex as entry and exit nodes do not know directly about each other.

TOR offers more than 'just' anonymous browsing of the internet, there are so called hidden services available in TOR. These are services that one of the nodes in TOR is hosting e.g. chat service, web-store, etc. A hidden service can only be accessed if the specific URL-like address of the service is known (onion address). In the case of hidden services, the default length of a circuit is five, as both users 'meet' at the third hop as shown in 4.

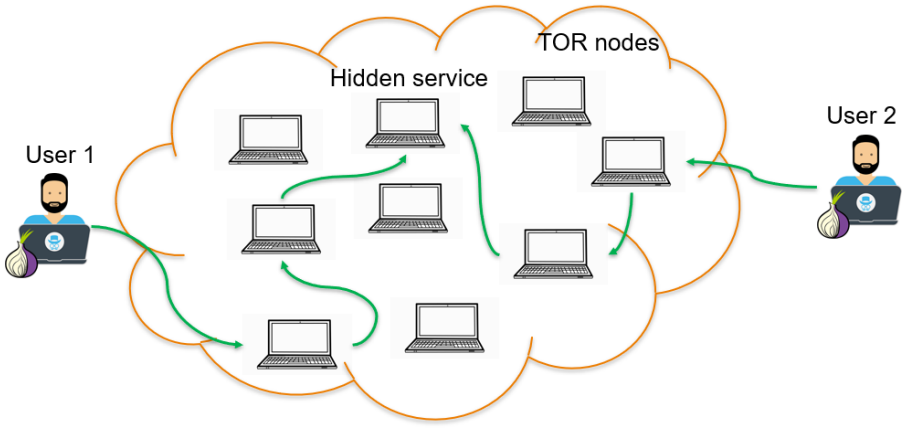



Figure 4: An overview of two users connecting and using a hidden service.

In some parts of the world anonymity may be considered as a problem and organizations or governments are trying to prevent internet users from achieving it. Due to the publicly available list of nodes, these oppressive regimes can block the IP addresses of the entry nodes thus, the user does not have a way to access TOR. In TOR there is a solution for this. Next to the nine Directory Authorities, there is one Bridge Authority that has a list of entry nodes that are not publicly available online. This makes it possible for people in oppressive regimes to access TOR as these entry nodes are not publicly known and cannot be blocked. The TOR software has a setting that enables the use of the Bridge Authority. In the unlikely event that the client cannot connect to the Bridge Authority due to a successful attack or a connection issue, the user can still connect to a bridge node. A list of bridge nodes is hard coded into the TOR client so users from more regulated and strict locations can still access the network. The node controlled by the Bridge Authority is called the bridge node.

A short summary of the nodes and functionalities of TOR:

- The directory nodes are controlled by Directory Authorities. Directory nodes know the state of the network and vote on a network state. The agreed state contains the list of available nodes and is sent to the TOR client.

- The entry node knows the source of the message and the first 'hop' on the TOR network but does not know the destination of the message.
- The middle node knows the entry node and the exit node but knows neither the source nor the destination of the message.
- The exit node knows the middle node and the destination but does not know about the source of the message.
- Hidden services are provided by users of TOR, they can be any service provided on the internet (e.g. chat, streaming media). They have a unique URL-like address called onion address and can only be accessed using them.
- The bridge node is controlled by the Bridge Authority. Bridge nodes serve as entry nodes to the TOR network and are not publicly available to avoid them being blocked.

### 3.1.2 Circuit creation

So far, we understand that the Directory Authorities have a view of the network, the next step is the creation of a secure circuit. The TOR client connects to one of the DAs to fetch the latest state of the network. From the consensus document the client constructs a path in a backwards fashion:

1. First the exit node is chosen, in order to have a way out of the network.
2. The middle node is chosen in the second step, this step can be repeated multiple times if more than three nodes are going to be used.
3. Finally, the entry node is selected by the client.

When a circuit is created in TOR, the network sends two kinds of messages to the nodes: `CREATE` and `EXTEND`. The `CREATE` message adds the node to the circuit and creates a shared symmetric key between the node and the client using RSA. If the addition of the node to the path is successful a `CREATED` message is sent to the client.

If the circuit is being extended the `EXTEND` message is sent through the nodes. Diffie-Hellman and Elliptic Curve cryptography can be used to handle the circuit extension request. When the `EXTEND` message reaches the recipient node, the node will append an extra node to the circuit using the `CREATE` message. Once the extension of the circuit is successful, the `EXTENDED` message is sent back to the client to confirm the successful operation. In the original Tor Authentication Protocol (TAP) Diffie-Hellman was used. In the current version of TOR, TAP is replaced by `ntor` that uses Elliptic Curve cryptography.

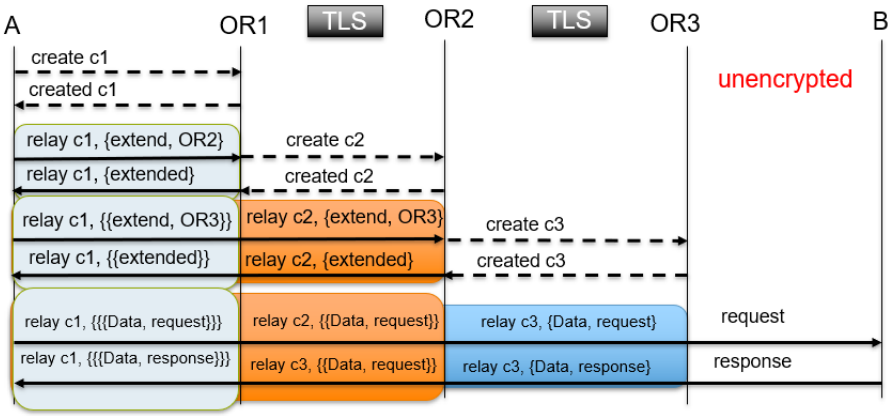



Figure 5: An overview of the messages used for circuit creation.

After this procedure the client knows the nodes it will use for the connection, so it creates a secure channel between the nodes and the client. Using asymmetric cryptography, the TOR client performs key exchange with the nodes to obtain a different symmetric key with each node. The client knows the symmetric key for each node on the circuit and encrypts the message with these keys in a layered way - hence the name onion routing. The destination and message are encrypted first using the symmetric key of the exit node, followed by the middle node and entry node:

$$\text{Enc}_{OR_1}(\text{Enc}_{OR_2}(\text{Enc}_{OR_3}(m, dest))) \tag{4}$$

In Equation 4 Enc denotes one layer of encryption. The symmetric keys of Onion Routers 1, 2, and 3 are denoted by $OR_1, OR_2$, and $OR_3$. The message sent is denoted by $m$ and the destination of the message is denoted by $dest$.

Sending the message through the network encrypted in this layered way means that each node is able to decrypt its corresponding layer and can forward the remaining encrypted message to the consecutive node. In the end, the exit node will know the destination of the message. Note, that there is no secure channel established between the exit node and the destination of the message.

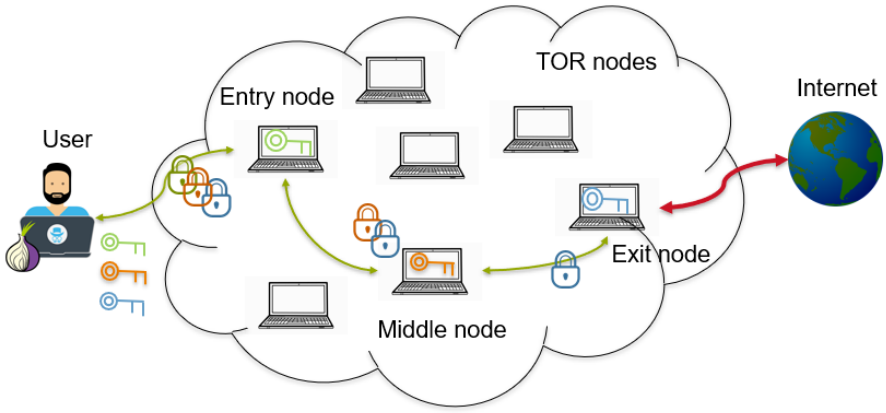



Figure 6: An overview of TOR with the symmetric keys.

## 3.2 Building blocks and keys

In this section the cryptographic building blocks behind TOR and keys used in TOR are explained. Both symmetric and asymmetric cryptography is used in TOR. Asymmetric cryptography is used for circuit creation, while symmetric cryptography is used for encrypting the message that has been sent through the network. The documentation for the latest cryptographic versions used in TOR is available online [10].

### 3.2.1 Cryptographic building blocks

The asymmetric cryptography used is:

- RSA with 1024-bit keys, with a fixed exponent 65537 and OAEP-MGF1 padding,
- Elliptic Curve cryptography is currently used in the form of `Curve25519` and `Ed25519`.

The symmetric cryptography used is:

- AES-128 , in counter mode with an initialization vector of 0s

Symmetric cryptography is used between the client and the TOR nodes to add and decrypt layers of encryption when a message is sent through the network. The security of currently used cryptographic schemes in TOR is listed in Table 3.

| Algorithm | Key size | # bits of security |
|---|---|---|
| RSA | 1 024 | 80 |
| ECC | 256 | 128 |
| AES | 128 | 128 |

Table 3: Cryptographic schemes used in TOR with key sizes and security bits.

### 3.2.2 Keys in TOR

Each node has to maintain a set of keys. These are called long-term, medium-term, and short-term keys. RSA and `Ed25519` are used for all three type of keys, but `Curve25519` is only used for medium-term keys. Table 5 contains an overview of the keys used in TOR with their functionalities.

The lifetime of these keys is not too strict, but there is a boundary for each category. Table 4 indicates the lifetime of the keys used by nodes.

| Key | Lifetime |
|---|---|
| short-term | minutes - 1 day |
| medium-term | 3 - 12 months |
| long-term | 12+ months |

Table 4: Keys and their lifetime used by TOR.

Nodes are uniquely identified using their identity key (RSA) and their master identity key (`Ed25519`). These two keys together form a unique authentication key pair. After a node has used this unique key pair to authenticate itself, none of the keys may be different in the future.

As of its first release in 2004, TOR has grown substantially. Nowadays there are about two million directly connecting users to the network and another sixty thousand are connecting through bridge nodes. TOR consists of roughly 7 000 nodes, of which there are about 3 000 entry nodes and 1 000 exit nodes [26]. One challenge is to keep the network functioning with the number of available nodes. When everything works as intended this is not an issue, but when a node starts to behave suspiciously, the network will isolate the node and not use it in the future. If there are too many misbehaving nodes the result could be a denial of service for the whole network.

| Type | Key lifetime | Key name | Function |
|---|---|---|---|
| RSA | long-term | identity key | Establish relay identity, sign documents and certificates. Since the introduction of `Ed25519`, RSA is only used to establish relay identity. |
| | medium-term | onion key | Decrypt cells at circuit creation. Used in `ntor` and TAP for handshakes. |
| | short-term | connection key | Establish TLS channels between nodes. |
| `Curve25519` | medium-term | - | Handle handshakes in the `ntor` protocol. |
| `Ed25519` | long-term | master identity key | Sign medium-term `Ed25519` key. This key never changes. |
| | medium-term | signing key | Replaces RSA identity key to sign documents and certificates. |
| | short-term | link authentication key | Authenticate handshakes after a TOR circuit negotiation. |

Table 5: Function of RSA, Curve25519 and `Ed25519` keys in TOR.

A second challenge that TOR faces also relates to usability: circuit build times [25]. In a standard connection there are three hops in the created circuit. The first hop takes the most time - more than 1.5 seconds in the worst case. The consecutive second and third hop take up to half a second or more. In the end circuit build times can take over 2.5 seconds. Introducing quantum-safe schemes is expected to have a higher load on computation - key generation, encryption and decryption - and on network communication.

The challenge is to keep user experience at an acceptable level but guarantee security against quantum adversaries. Furthermore, current nodes should be able to deal with computational and communicational tasks related to quantum-safe schemes.

## 3.3 Issues and attack surfaces

An adversary can have multiple goals e.g., decrypt messages, disable the network, expose flaws in programs used, etc. In the case of TOR, the main goal of an attack is to deanonymize users. In Section 2 two attack types were mentioned: active and passive. These hold for TOR as well. In [11] the authors mention that TOR is vulnerable against a global passive adversary who can monitor traffic at the entry and exit nodes of the network. A passive adversary may observe traffic patterns, and timing correlations. These techniques allow him to learn which website a user is visiting by linking source to destination. Such an attack is very costly for any adversary to carry out, as the adversary would need to monitor a great part of the entry and exit nodes of the network.

An active attack is more interesting as the attacker tries to compromise the TOR network. The scope of active attacks includes compromising of keys, running a malicious node, and attacks on the directory nodes. A publication in

early 2019 by Enrico Cambiaso et al. gives a clear overview on past and current attacks on TOR [6]. Cambiaso et al. distinguish between three kinds of attacks on anonymity:

- Attacks on the client,
- Attacks on the servers,
- Attacks on the network.

Client attacks focus on software related vulnerabilities of the TOR client or third-party applications. They also assume that the attacker has control over some entry and exit nodes and manages to force the user to use these nodes.

Server attacks focus on compromising TOR hidden services. The attacker tries to manipulate/modify the packets/TOR cells sent through the network. In general, the goal is to lead the user to a malicious node controlled by the adversary. If this is successful, the user loses anonymity.

The network attacks have multiple goals. One of the aims is to compromise bridge nodes as these nodes are not available in the public TOR node listing. If this is successful, the adversary learns the IP addresses of the bridge nodes and can block them. If the entry nodes and bridge nodes are blocked, users can no longer connect to TOR and cannot use the internet anonymously any longer. In general, the attacks on the network aim to deny access to the TOR network.

One common thing about these attacks on TOR is that none of the above attacks focus on the cryptographic weaknesses of TOR. Instead, the focus is on other vulnerabilities such as software and the abuse of trusting nodes in TOR.

In the next section we introduce scenarios where the cryptography in TOR can be abused by quantum adversaries. Furthermore, recommendations are made about how to make TOR quantum-safe.

# 4 Post-quantum TOR

In Section 1.3, two papers are mentioned that discuss a Post-quantum TOR network [33, 12], and Section 3.3 explained the possible attacks on TOR. In this section [33, 12] papers are further elaborated, and possible attack surfaces of a quantum adversary are discussed. Finally, we answer the first and second questions of our research.

## 4.1 Challenges

With quantum computing emerging, the cryptography of TOR needs to be adjusted. The quantum vulnerability of asymmetric and symmetric key cryptography will open a new attack surface for adversaries. Introducing quantum-safe cryptography to TOR is the task that needs to be done in order to keep cryptographic vulnerabilities off the list of attack surfaces. This is why it is pivotal to introduce quantum-safe cryptography to the keys of the nodes.

| **Scheme** | **Key** | **Attacker capability** |
|---|---|---|
| RSA | Identity key (long-term) | Impersonate a node, send spoofed descriptors that are signed by the compromised identity key. |
| | Onion key (medium-term) | Read the content of TOR cells until the next key rotation. |
| | Connection key (short-term) | See encrypted traffic between nodes. |
| Curve25519 | medium-term | Read the content of TOR cells when a circuit is created. |
| Ed25519 | Master identity key (long-term) | Create a new signing key. |
| | Signing key (medium-term) | Can sign modified documents and publish them to the directory servers. |
| | Link authentication key (short-term) | Can authenticate connections that should be not allowed. |

Table 6: Attacker capabilities with compromised asymmetric schemes.

Table 6 explains the attacker capabilities in case the RSA, `Curve25519` or `Ed25519` schemes become compromised and the attacker learns about the keys of the nodes. Symmetric schemes also need modification, although they are not directly related to quantum-safe schemes. Increasing the key sizes of the AES symmetric cryptographic schemes are the solutions for this. An AES256-bit scheme is claimed to achieve 128-bit security against quantum adversaries.

## 4.2 Attack scenarios

As most current attacks on TOR are based on other vulnerabilities than cryptography, in this section we consider attack scenarios on the keys of the nodes that a quantum adversary possesses. Furthermore, the addition of quantum-safe cryptography is also be considered in this section.

As discussed in Section 3, there are four types of keys in TOR and all of these can be compromised by an attacker:

- symmetric key,
- short-term key,
- medium-term key,
- long-term key.

Compromising the symmetric keys enables the adversary to decrypt layers of encryption and learn the destination of the message. In the case that the attacker only knows the symmetric keys and nothing else, the encrypted message must be intercepted before entering the network. Otherwise, the TLS connection will add an extra layer of security. If an adversary does not know all the symmetric keys, she cannot fully decrypt the message and thus, the circuit is not fully known, so source and destination remain anonymous.

Compromising the short-term key at an entry node would make the adversary capable to follow the full length of the circuit from sender to recipient. This would lead to deanonymization of the user. This attack is possible during the lifetime of the TLS connection.

In case the adversary knows the short-term key and the medium-term key of a node, the attacker can impersonate this node. This is doable as the medium-term key enables the attacker to decrypt `CREATE` messages and establish new circuits. Such a node can decrypt one layer of symmetric encryption when the messages are passed through it. Thus, the previous and next 'hop' in the circuit is known for the attacker. The attack is possible until the rotation of the medium-term keys.

The long-term key may also be compromised by the adversary. This would enable the adversary to impersonate the node and send forged descriptors to the directory nodes. Moreover, the attacker can see previous and consecutive 'hops' in the circuit with the encrypted cells.

Current successful attacks on TOR are carried out with colluding adversaries. If adversaries control the entry and exit nodes in the network, they can share information with each other and as a result deanonymize communicating parties.

Colluding adversaries at the entry and at the exit node who have the medium-term keys will both know the middle relay in a circuit. Sharing this knowledge enables them to attempt to deanonymize users, as the users using the common middle node has the biggest possibility to be communicating with each other.

The colluding attack can also be done when the adversaries at the entry and exit nodes have compromised the long-term keys. Such an adversary is able to alter the packets sent through the network. The adversary at the exit node can inspect the packet received and if the altered packet is received, they can deanonymize users successfully. Table 7 gives a short summary of the attacker capabilities when compromising the short/medium/long-term keys.

In the attacks described, the adversary needs to compromise keys generated by asymmetric schemes. These are currently considered a hard problem, but for a quantum adversary it becomes more feasible to compromise them. As described in Section 3.3 current attacks on TOR do not target the cryptography, but rather focus on vulnerabilities in TOR related software, hidden services, bridge node discovery, disabling the network, and on generic attacks like timing.

Common technique of adversaries is to introduce new nodes to the TOR network, but this is a lengthy process due to the policy of the network. New nodes are even more closely monitored than nodes already in the network for malicious patterns and if such is recognized, they are excluded from the network.

A quantum adversary can compromise the short/medium/long-term keys of certain nodes and gain control over it. This poses two issues:

- Deanonymizing users can be done easier if access to nodes is gained,
- Causing a Denial of Service attack becomes a concern if many nodes are taken offline because of TOR policy.

Table 8 shows the security gains for the attacks previously introduced after the addition of quantum-safe cryptography. Introducing quantum-safe cryptography is not going to solve current vulnerabilities of TOR. Rather, it prevents that cryptography is added to the list of vulnerabilities and attack surfaces.

| **Attack surface** | **Attacker capability** | **Impact of successful attack** |
|---|---|---|
| Compromise symmetric keys | Decrypt the message entering the network. | Message content learned, and anonymity lost. |
| Compromise short-term key at entry node (connection key) | Follow the circuit during the lifetime of the connection. | Anonymity lost. |
| Compromise short-term and medium-term keys of middle node (connection key, onion key) | Impersonate the node until the rotation of the medium-term key. | Decrypt `CREATE` ells, see encrypted traffic passing through the node. |
| Compromise long-term key of a node (identity key) | Replace the node by sending forged descriptors to directory authorities, see previous and consecutive hops/encrypted cells. | Direct uses to malicious nodes. |
| Colluding adversaries with compromised medium-term key at entry and exit node (onion key) | The middle nodes will be known by both parties, the colluding parties can match source to destination if the middle node is the same. | Anonymity lost. |
| Colluding adversaries with compromised long-term key at entry and exit node (identity key) | Colluding parties impersonate nodes and can modify packets sent through the network. Receiving such altered packet, the parties together deanonymize the user. | Anonymity lost. |

Table 7: Attacker capabilities with exploited keys of nodes.

| **Attack surface** | **Security gain after quantum-safe cryptography** |
|---|---|
| Compromise symmetric keys | Larger key size will make it harder or infeasible for the adversary to decrypt messages. |
| Compromise short-term key at entry node (connection key) | TLS becomes quantum-safe, adversary can no longer "look into" the traffic between nodes. |
| Compromise short-term and medium-term keys of middle node (connection key, onion key) | Impersonating the node will no longer be possible, as the medium-term key will be quantum-safe. |
| Compromise long-term key of a node (identity key) | Attacker cannot impersonate the node for indefinite time and send forged descriptors to the directory authorities. |
| Colluding adversaries with compromised medium-term key at entry and exit node (onion key) | Cryptographic vulnerability is solved, as the medium-term key remains secure, adversaries will not know about the middle hop. Adversaries monitoring entry and exit nodes can still link sender and destination using timing attacks for example. |
| Colluding adversaries with compromised long-term key at entry and exit node (identity key) | Cryptographic vulnerability is solved, as the long-term key remains secure, adversaries can no longer impersonate nodes and alter packets. Adversaries monitoring entry and exit nodes can still link sender and destination using timing attacks for example. |

Table 8: Security gain with quantum-safe schemes.

Based on Table 8 we can say that with the addition of quantum-safe schemes to TOR, the vulnerabilities caused by asymmetric key cryptographic schemes against quantum adversaries can be solved. Moreover, increasing the symmetric AES key will make 'store-now-decrypt-later' attacks infeasible.

## 4.3 Papers on quantum-safe TOR

In Section 1 two papers on quantum-safe TOR were introduced and in this section, they are further elaborated. These two papers are *Post-Quantum Forward-Secure Onion Routing (Future Anonymity in Today's Budget)* by S. Ghosh and A. Kate [12], and *Circuit-extension handshakes for Tor achieving forwards secrecy in a quantum world* by J. M. Schanck, W. Whyte, and Z. Zhang [33].

Published in 2015, in [12] the focus is on keeping forward secrecy and to do so the short-term keys are the focus. The proposed solution is a custom key exchange protocol design and it focuses on keeping the short-term keys TOR uses to negotiate session keys for TLS connections quantum-safe. For long-term keys, they use current classical cryptography. The protocol is called `HybridOR` and is based on lattice problems. In [12] the protocol is reported to be computationally more efficient compared to currently used `ntor`, but this can be due to performing less Diffie-Hellman operations. Drawbacks reported in the paper are related to network communication. Due to the quantum-safe scheme the amount of data communicated increases. `HybridOR` is assumed to be secure under the ring-Learning With Error (r-LWE) assumption. Although, if the r-LWE assumption becomes insecure against a classical adversary, it will no longer provide forward secrecy even against classical adversaries.

One year later another paper on quantum-safe TOR was published [33]. The paper, like [12] also focuses on making the short-term keys quantum-safe. In [33] the currently used `ntor` protocol is modified and is called `hybrid`. `Hybrid` uses the combination of long-term keys generated by Diffie-Hellman key exchange, and short-term keys generated by a quantum-safe scheme `NTRUEncrypt`. This implementation is beneficial, as if the quantum-safe part of the key becomes compromised, the protocol can fall back to the current `ntor` security guarantees thus, provide forward secrecy against classical adversaries. The `hybrid` scheme is 128-bit IND-CCA secure against quantum adversaries.

The authors of [33] have also made a performance comparison of the different TOR protocols. Table 9 contains this comparison. In TOR the standard packet size is set to 4 096 bits (512 bytes). Table 9 lists the number of bytes the client needs to send to set up a connection with the server, and the number of bytes the server sends as a response. Furthermore, the amount of time required by the client and the server to set up the connection is shown.

| | **ntor** | **hybrid** | **HybridOR**** |
|---|---|---|---|
| Packets (1 = max 512 bytes) | 1 | 2 | 3 |
| client → server (bytes) | 84 | 693 | 1 312 |
| server → client (bytes) | 64 | 673 | 1 376 |
| client init | 84 $\mu$s | 661 $\mu$s | 150 $\mu$s |
| server response | 263 $\mu$s | 306 $\mu$s | 150 $\mu$s |
| client finish | 180 $\mu$s | 218 $\mu$s | 150 $\mu$s |
| total | 527 $\mu$s | 1185 $\mu$s | 450 $\mu$s |

Table 9: A comparison of the currently used `ntor` protocol, `hybrid` [33], and `HybridOR` [12] protocols. ** [12] assume 100 $\mu$s for Diffie-Hellman group operations and 50 $\mu$s for multiplication and addition in r-LWE. The other costs, such as sampling of the r-LWE secrets are ignored.

Based on Table 9 and [12, 33] we conclude that that adding quantum-safe schemes to TOR increases the message sizes and the time the client and server need to communicate. In the case of [12] we can only draw conclusions about the message sizes and they are even larger than for `hybrid`. This means even though the assumed computation costs are more favourable, the increased packet size has an impact on network load. Furthermore, these works focus on making short-term keys quantum-safe, that are used when TLS connections are set up and long/medium-term keys are not considered.

## 4.4 First recommendations for quantum-safe TOR

At this point, we understand TOR, the vulnerabilities, and have a feeling what adding quantum-safe cryptography means for TOR. The first and second questions can be answered of our research.
Recall the first two questions:

- Which are the most pivotal parts in TOR that need to be made quantum-safe first?
- Where and when should quantum-safe cryptography be added to TOR?

TOR makes use of the public-private keys of the nodes thus, the most pivotal parts to be updated are the long-term/medium-term/short-term keys of the nodes and the symmetric keys. Note, in order to make the symmetric keys secure there is no need to add any quantum-safe scheme. The priority list of the focused components is:

1. AES symmetric key,
2. long-term/medium-term keys,
3. short-term key.

The AES symmetric key does not require a quantum-safe scheme to be updated, but this has to be changed as soon as possible. Increasing the key size from the currently used 128-bit keys to 256-bit keys would already solve the problem. This is a burning issue as adversaries storing current traffic will be able to decrypt the messages in the future when the needed computational power is available. Updating the symmetric keys makes 'store-now-decrypt-later'-attacks infeasible.

The most important asymmetric part that needs to be secured are the medium-term and long-term keys. These pose a threat as they do not have frequent rotations. Thus, the attacker has more time to compromise these keys and if succeeded, as the keys do not change for a long period of time, the adversary can impersonate the node until the next key rotation. A compromised node can redirect traffic to other nodes that have been compromised, this can lead to deanonymization. A compromised node can also start to behave suspiciously, and if so it can be excluded from the network. In case of too many nodes being excluded, TOR could become unstable. Key rotation time is not the sole factor for the need to update long-term keys. In order to keep the protocol forward secure, the long-term and medium-term keys need to be updated. This ensures that a compromised long-term key in the future does not aid the adversary to learn about past communications with the same key.

Finally, the short-term keys need to be updated. These keys rotate frequently - every 10 minutes if the connection has no issues - and because of this, the time for an adversary is limited to compromise the connection. If an adversary manages to compromise the short-term key used for the connection at the entry node, it can follow the circuit from start to end making it possible to deanonymize communicating parties. It must be mentioned, that TOR does not create new circuits for streaming protocol (IRC) to limit the possibilities for an attacker to link parties. This will also pose as a threat in the quantum world as the circuit will not be recreated after 10 minutes and an adversary will have more time to compromise the circuit. Furthermore, TOR currently uses short-term keys for TLS session negotiations and quantum-safe TLS is being researched [5].

It is clear that short/medium/long-term keys are the areas that need to be investigated as they are generated using asymmetric key cryptography. More technically, the nodes need to stop using RSA keys and switch to quantum-safe schemes for key generation.

AES keys need to be changed as soon as possible. The currently used 128-bit AES encryption is expected to be vulnerable against quantum adversaries. Attackers who are storing current encrypted data will very likely be able to decrypt it. RSA with 1024-bit keys has yet to be successfully factorized, still NIST advises to use RSA with 2048-bit keys [3]. A report released by RSALaboratories claim, that RSA keys of 2048-bits are sufficient until 2030 [18]. Using longer RSA keys may be the fast fix for the issue, but no matter how long keys are

used against a quantum computer, all will be vulnerable to quantum attacks. Based on [18] after 2030 we do not expect to further enlarge the size of the RSA keys, and by 2030 quantum-safe keys will be used. Transitioning from RSA to quantum-safe schemes is not as urgent as updating AES, but starting to use quantum-safe cryptography for testing and in practice in the upcoming 5-10 years is beneficial. It gives cryptographers and users a glimpse of the advantages and drawbacks related to quantum-safe schemes and thus, allows more time to consider other implementations and proposals if needed.

# 5 Experimental results

In this section we describe and discuss the experimental results on quantum-safe TOR implementations. Specifically, Section 5.1 explains the implementation used and the challenges faced. Section 5.2 gives an overview of the system specification used and the steps of the experiment. Section 5.4 contains the results for the experiments and recommendations on what quantum-safe TOR should look like. Finally, in Section 5.5 the third and fourth questions for the research are answered. The final recommendations based on our findings on quantum-safe TOR are discussed and the main question of the research is answered.

## 5.1 Implementation

### 5.1.1 Onion routing

In order to further understand the impact of quantum-safe cryptography on TOR we need to perform experimental testing. The original source code for the TOR project can be downloaded [39] but cannot be altered and experimented with easily. A TOR-like simulation called `SweetOnions`, where a small-scale version of onion routing is emulated using Python 2 was used [20]. The project is well documented and the instructions on how it works are clear. The cryptographic implementations differ from the ones TOR uses - 2048-bit RSA and 192-bit AES ciphers are used instead of 1024-bit RSA and 128-bit AES that TOR uses.

To emulate a TOR network with `SweetOnions` a minimum number of 6 machines is required:

- one server - the destination to where the messages are sent,
- one directory - the IP addresses and public keys of the nodes are stored here,
- one client - source of the message,
- three nodes - the IP addresses and public keys are shared with the directory, when receiving an encrypted message one layer is removed and sent to the next node.

The `SweetOnions` implementation can be used to get a feeling on how quantum-safe schemes will affect the performance of the network. CPU cycles can be measured for key generation as well as for encryption and decryption. The packet size that TOR uses is 512 bytes, `SweetOnions` also emulates this size for messages, but we need to be careful. The buffer size `SweetOnions` uses checks for 4 096 character length, which is not equal to 4 096 bytes. A challenge when working with the `SweetOnions` codebase is that the initial code was written for Python 2, but the Open Quantum Safe [30] libraries use Python 3.

### 5.1.2 Open Quantum Safe library and challenges

The Open Quantum Safe library [30] has multiple implementations of post-quantum secure schemes. The master branch contains both key encapsulation and signature schemes. In the scope of this thesis the key encapsulation schemes are considered.

The original implementation of these schemes is in `C`, but there is a wrapper available that can be used for Python [31]. In the Python implementation there are schemes for NIST security levels 1 to 5, but only the schemes that achieve level 1 NIST security are tested. The reason behind this decision is if level 1 security is achieved, we consider TOR to be quantum-safe. When changing to new cryptographic schemes it is important to try to keep the fundamental parts as unmodified as possible. This is because the currently used protocols are well-tested, researched and trusted in their current form and changing the fundamentals can result in some unforeseen challenges. If TOR can be used with the current packet size, network load and can also be quantum-safe, that would be extremely beneficial as the core of TOR would not require any change. The tested schemes to provide level 1 NIST security are

- Frodo-640-AES,
- Frodo-640-SHAKE,
- Kyber512,
- NewHope-512-CCA,
- NTRU-HPS-2048-509,
- Sike-503.

As mentioned in 5.1 the original codebase for `SweetOnions` is Python 2, but in order to use the Open Quantum Safe library Python 3 is required. The main challenges to solve when converting to Python 3 from Python 2 are:

- Formatting,
- Networking,
- Encryption and decryption.

There is a Python package called `future` that makes the necessary conversions between Python 2 and Python 3 [32]. Using future solved a good part of our problems such as converting `print` statements and input reading operations to Python 3 format, but there still remained issues. Network communication works differently in Python 3. With the original implementation string objects could be sent through the network, but in Python 3 `byte` type objects are expected. The same challenge holds for encryption and decryption functions, they also output `byte` objects. Thus, new encryption and decryption functions were

written, and modifications were made to the network communication as well. As a result, the `SweetOnions` implementation can now be run using Python 3 and can be combined with the Open Quantum Safe library.

### 5.1.3 Expected drawbacks

Before running the experiments, some expectations about the negative effects that will be observed when introducing quantum-safe schemes are listed:

1. CPU computation times will increase for key generation and for encapsulation/decapsulation compared to RSA. This expectation is based on [33], where client initiation and server response times are reported to take more time than for the current `ntor` protocol.

2. Public and private key sizes will increase which will result in heavier network traffic for Directory Authorities. Based on [12, 33] message sizes increased, which can be caused by larger encryption keys.

3. Message sizes will increase due to quantum-safe encryption of messages. As a result, the 512-byte long packets will not be sufficiently large and fitting one connection into one packet as done for `ntor` will be not possible [33]. This means one of the fundamentals of TOR could need modification.

## 5.2 Experimental setup

In this section we give an overview of the system used for our experiments and how these experiments were carried out. Furthermore, we introduce the design of our classical, our purely quantum-safe and our hybrid schemes.

### 5.2.1 System setup

For the experiment local and virtual environments are both used. The technical specification of the notebook used for the local experiments is Dell Latitude E7240 TNO Managed Kepler Computer, with Intel Core i5-4310U CPU @ 2.00 - 2.60GHz processor, 8GB RAM, Samsung SSD SM841N mSATA 128GB for storage and Windows 10 Enterprise 64-bit operating system. Furthermore, an Ubuntu 18.04 LTS subsystem was installed.

In order to emulate the TOR network, 6 virtual machines were used with Intel Core Processor (Broadwell) @ 2.4 GHz processors, 60GB storage, a virtual network adapter, and Linux version 4.15.0 operating system.

### 5.2.2 Experiment setup

To have a reliable overview of running the experiments, the different stages of testing were planned to build up on each other.

First, the theorical parts of classical, quantum-safe and the hybrid schemes were considered. These included planning about what encryption and decryption of messages should look like and also the format of the messages.

Secondly, the public key, private key and ciphertext sizes are measured considering the Open Quantum Safe codebase. CPU cycles for key generation is measured for the classical and quantum-safe schemes.

Thirdly, the encapsulation and decapsulation CPU cycles are measured for RSA and each quantum-safe scheme.

The `SweetOnions` TOR network was used for measuring circuit build times and message sizes for three scenarios: classical implementation, quantum-safe implementation, hybrid implementation. As this is not the original TOR network the classical implementation serves as a benchmark measure and based on the numbers, predictions can be given for the actual TOR network.

In order to have comparable results, the CPU cycles were measured. This allows a better comparison method for the experiment as the network has minimal latency because the virtual machines are in the same virtual network. Measuring the circuit build times for the original, quantum-safe, and the hybrid implementations demonstrate the CPU cycles required for circuit creation without the network latency having any effect.

### 5.2.3 Classical scheme

In the original `SweetOnions` implementation the client sends a message containing RSA and AES encryption through the network. The client creates the correct message format as follows

1. The client generates an AES key and encrypts the message,
2. The client uses the RSA public key of a node to encrypt the AES key used to encrypt the message,
3. The client combines the two encryptions as one message and sends it to the node.

The format of the sent message is (RSA(AES key), AES[message]), where `()` denotes asymmetric key encryption and `[]` denotes symmetric key encryption. This notation is the same for the other two methods. The client repeats these steps three times to have the correct wrapping of the message.

After receiving a message in this format, the node can remove the encryption layer from the message:

1. The node splits the message to an asymmetric and symmetric part,

2. The node uses his RSA private key to decrypt the asymmetric encryption and learns the AES key,

3. After learning the AES key the node decrypts the AES encryption and learns the message.

### 5.2.4 Quantum-safe schemes

In order to keep measurements most accurate, we aim not to modify the original format of sending/receiving messages too much. The quantum-safe schemes do not use RSA encryption to encrypt the symmetric key, which means that the RSA part of the message sent in the original implementation should be replaced. The implementation used for quantum-safe schemes work the following way:

1. The client uses a quantum-safe scheme to generate a shared secret - that is used and referred to as the (AES key) - and a ciphertext of this shared secret using the quantum-safe public key of the node (PQC(AES key)),

2. The client uses the AES key to encrypt the message,

3. The client combines the ciphertext of the AES key and the encrypted message as one and sends it to the node.

The message format the client sends is (PQC(AES key), AES[message]).
When a node receives this message, he can remove the encryption layer from the message:

1. The node splits the message into the post-quantum and symmetric part,

2. The node uses his quantum-safe private key to decapsulate the ciphertext and learns the AES key,

3. After learning the AES key the node decrypts the symmetric encryption and learns the message.

### 5.2.5 Hybrid schemes

For security considerations a combination of classical and quantum-safe schemes is tested. In an unforeseen security issue of the quantum-safe schemes, a classical adversary should not be able to break the encryption. If this were the case and adversaries managed to decapsulate the AES keys from the ciphertext when purely quantum-safe schemes are used, the content of the message can be learned. To overcome this issue, instead of using purely an encapsulated AES key, the preferred method is to combine RSA and quantum-safe cryptography. This ensures that an adversary would still need to break the security of RSA in order to compromise the weakness of the quantum-safe scheme.

The Open Quantum Safe library has a limitation when encapsulating a shared

secret. The encapsulation function uses a public quantum-safe key and generates a shared secret that can be used as an AES key for encryption, and a ciphertext of this shared secret. Due to this, when using a hybrid, instead of the client generating an AES key for encrypting the message and using the quantum-safe scheme to generate a shared secret that can be used as a second AES key for encryption, it uses the quantum-safe scheme to create the shared secret that is used for AES encryption, and then the ciphertext of the shared secret is encrypted with RSA encryption.

The implementation of the scheme is the following:

1. The client generates a shared secret that is used and referred to as the AES key and a ciphertext of this shared secret (PQC(AES key)),
2. The client uses the AES key to encrypt the message,
3. The client uses the RSA public key of a node to encrypt PQC(AES key),
4. The client combines the two parts and sends the message to the node.

The resulting message format is RSA(PQC(AES key)), AES[message]). When receiving a message like this, the node can remove the encryption layer as follows:

1. The node splits the message into the asymmetric part and the symmetric part,
2. The node uses his private RSA key to decrypt and obtain the ciphertext of the shared secret,
3. The node uses his quantum-safe private key to decapsulate the ciphertext of the shared secret to learn the AES key,
4. After learning the AES key, the node decrypts the symmetric encryption to learn the message.

After these considerations the format of the `SweetOnions` protocol does not need to be changed. Each node publishes a public key towards the directory server, this will be either an RSA or a quantum-safe key. The client connects to the directory server to receive a list of available nodes and their corresponding public keys. The client picks a route from the list of nodes and in the case of RSA, generates AES keys that will be used for the symmetric encryption, and in the case of the quantum-safe schemes uses the scheme to generate the shared secret that is used as the AES key. The client encrypts the message using the AES key and either encrypts the AES key using RSA encryption (RSA(AES key)), simply uses the ciphertext generated by the quantum-safe scheme (PQC(AES key)) or encrypts the ciphertext of the shared secret using RSA (RSA(PQC(AES key))).

The node receives a (RSA(AES key), AES[message]) tuple, a (PQC(AES key), AES[message]) tuple, or a (RSA(PQC(AES key)), AES[message]) tuple. After splitting the message, the first part of the message is decrypted to learn the AES key.

After learning the AES key, the second part of the received message is decrypted to learn the message.

## 5.3 Benchmark results of quantum-safe cryptography performance

In this section we present the results for our measurements, specifically for key generation, public key, private key and ciphertext sizes, and encryption/decryption times. The benchmark measures were executed on the virtual machines. The order of these measurements are:

- Public key, private key and ciphertext sizes,
- CPU cycles for RSA key generation,
- CPU cycles for quantum-safe key generation,
- CPU cycles for RSA encryption and decryption,
- CPU cycles for quantum-safe encapsulation and decapsulation.

In order to get an average result for the CPU cycle measurements, 1 000 iterations were run with each test. Table 10 contains the public key, private key and ciphertext sizes. Table 11 contains the CPU cycles required for key generation, this is also shown in Figure 8. Furthermore, Table 11 contains the time required for key generation in seconds. To do this the CPU cycles were measured for one second and this number is 2 399 753 472 cycles. Table 12 contains the CPU cycles needed for encapsulation and decapsulation of a message, Figure 7 demonstrates this for all the schemes tested.

| **Scheme** | **Public key size (bytes)** | **Private key size (bytes)** | **Ciphertext size (bytes)** |
|---|---|---|---|
| RSA-1024 | $< 128$ | $< 128$ | 128 |
| RSA-2048 | $< 256$ | $< 256$ | 256 |
| RSA-4096 | $< 512$ | $< 512$ | 512 |
| RSA-6144 | $< 756$ | $< 756$ | 756 |
| RSA-9216 | $< 1\,152$ | $< 1\,152$ | 1 152 |
| Frodo-640-AES | 9 616 | 19 888 | 9 720 |
| Frodo-640-SHAKE | 9 616 | 19 888 | 9 720 |
| Kyber512 | 800 | 1 632 | 736 |
| NewHope-512-CCA | 928 | 1 888 | 1 120 |
| NTRU-HPS-2048-509 | 699 | 935 | 699 |
| Sike-p503 | 378 | 434 | 402 |

Table 10: The key sizes for RSA and quantum-safe schemes.

Key and ciphertext sizes play an important part. Keys have an effect on network load as they are sent to the Directory Authorities, and the Directory Authorities distribute them to the client. The ciphertext is important because it is going to be sent in the purely quantum-safe implementation and in the hybrid, it is sent after RSA encryption. In Table 10 these key sizes are listed. The ciphertext sizes for quantum-safe schemes range from 402 to 9 720 bytes. Both `Frodo-640-AES` and `Frodo-640-SHAKE` have ciphertext sizes of 9 720 bytes. This is problematic for two reasons. First, it would require 19 packets to send only the ciphertext using the purely quantum-safe schemes, and secondly encrypting 9 720 bytes with RSA is extremely time consuming.

RSA can encrypt messages as large as its key size. The RSA-1024 that TOR currently uses is not suitable to encrypt any of the quantum-safe scheme ciphertexts, as it can encrypt message sizes up to 128 bytes. Splitting the to be encrypted message is a possibility when using RSA-1024 but there is no standard on how to split messages and thus, it is not advised. Splitting would raise further challenges like creating too much overhead or decryption becoming expensive as discussed in [29]. Thus, the message is not split into smaller chunks but instead the size of RSA keys are increased to securely perform encryption. In case of `Frodo` an RSA key size of 77 824 bits (9 728 bytes) would be required. Generating a key of this size and encryption, decryption is extremely lengthy. In the case of `Kyber` and `NTRU` an RSA with key size of 6 144 bits should be used (768 bytes), for `NewHope` a key size of 9 216 bits (1 152 bytes), and for `Sike` a key size of 4 096 bits (512 bytes) is required. These also require significant time, but nothing compared to `Frodo`. Due to this, we decided not to continue testing the `Frodo` schemes.

The key generation times affect the nodes as they will generate a public-private key pair once every month as defined by default in TOR. The results show that changing the current 1024-bit RSA key to larger keys in order to have hybrids would have the biggest impact on CPU cycles.

| **Scheme** | **Number of CPU cycles** | **Time needed** |
|---|---|---|
| RSA-1024 key generation | 61 568 194 | 0.026s |
| RSA-2048 key generation | 266 140 623 | 0.11s |
| RSA-4096 key generation | 1 946 034 083 | 0.81s |
| RSA-6144 key generation | 7 718 455 577 | 3.22s |
| RSA-9216 key generation | 32 849 985 752 | 13.69s |
| Kyber512 key generation | 152 973 | 0.0000637s |
| NewHope-512-CCA key generation | 193 367 | 0.0000806s |
| NTRU-HPS-2048-509 key generation | 27 632 969 | 0.011s |
| Sike-p503 key generation | 90 800 645 | 0.038s |

Table 11: The CPU cycles needed for RSA and quantum-safe key generation.

Another observation that can be made based on Table 11 is, that all the lattice-based quantum-safe schemes (`Kyber, NewHope, NTRU`) require less CPU cycles for generating keys than currently used RSA-1024. Even the most costly lattice-based scheme, `NTRU`, requires 55% less CPU cycles to generate key pairs. On the other hand, `Sike` requires 147% more CPU cycles for the generation of keys compared to RSA-1024. The interesting fact is the huge gap that is present in the lattice-based schemes (e.g. `NTRU` is about 181 times more computational heavy than `Kyber`).

Key generation only affects the nodes as they generate keys based on the time defined by TOR. The factor that affects both the nodes and the client is the time needed to encapsulate and decapsulate messages. Benchmark encapsulation and decapsulation measurements can be seen in Table 12. The client uses encapsulations at least three times when preparing the layers of encryption. Opposed to key generation times where all lattice-based schemes were better in performance compared to RSA-1024, here `NTRU` requires more CPU cycles for encryption. Decapsulation on the other hand looks promising for lattice-based implementations because they all require less CPU cycles to perform decapsulation than RSA. The case is similar to the key generation performances, `NTRU` requires the most CPU cycles among the lattice-based schemes, but it is about 23% faster than the currently used RSA-1024. The supersingular isogeny-based quantum-safe scheme `Sike` requires the most CPU cycles among the tested schemes, it is 124 times more computationally heavy than RSA-1024 for CPU cycles needed for encapsulations and decapsulations in total.

| **Encapsulation scheme** | **Encapsulation operations** | **Decapsulation operations** | **Total** |
|---|---|---|---|
| RSA-1024 | 410 402 | 2 078 161 | 2 488 563 |
| RSA-2048 | 730 570 | 5 718 858 | 6 449 428 |
| RSA-4096 | 1 975 617 | 26 433 228 | 28 408 845 |
| RSA-6144 | 3 846 650 | 72 843 854 | 76 690 504 |
| RSA-9216 | 7 906 810 | 216 176 067 | 224 082 877 |
| Kyber512 | 170 856 | 195 106 | 365 962 |
| NewHope-512-CCA | 228 687 | 247 457 | 476 144 |
| NTRU-HPS-2048-509 | 636 263 | 1 609 748 | 2 246 011 |
| Sike-p503 | 149 691 623 | 159 119 760 | 308 811 383 |

Table 12: The CPU cycles needed for RSA and quantum-safe encapsulation and decapsulation.

All lattice-based schemes (`Kyber, NewHope, NTRU`) have better scores for CPU cycles than the RSA schemes, which is why we believe that [12, 33] both used

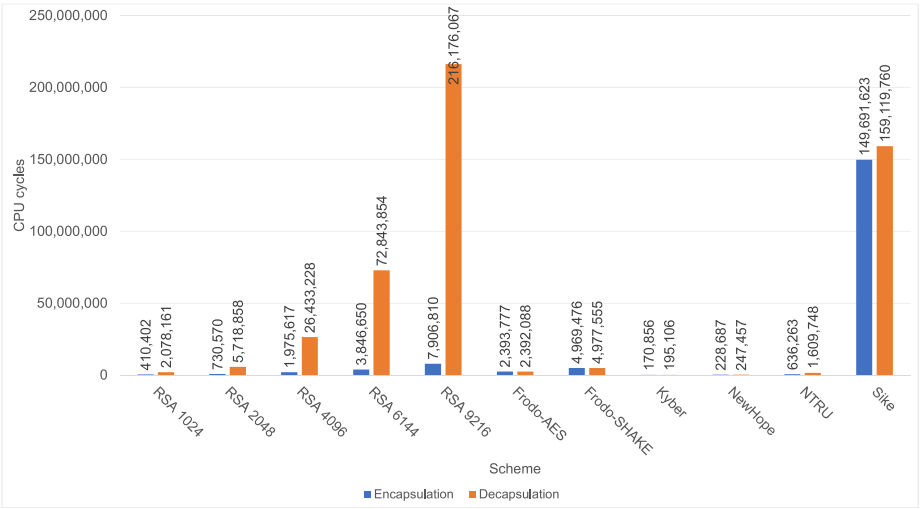



Figure 7: CPU cycles needed for cryptographic protocols.

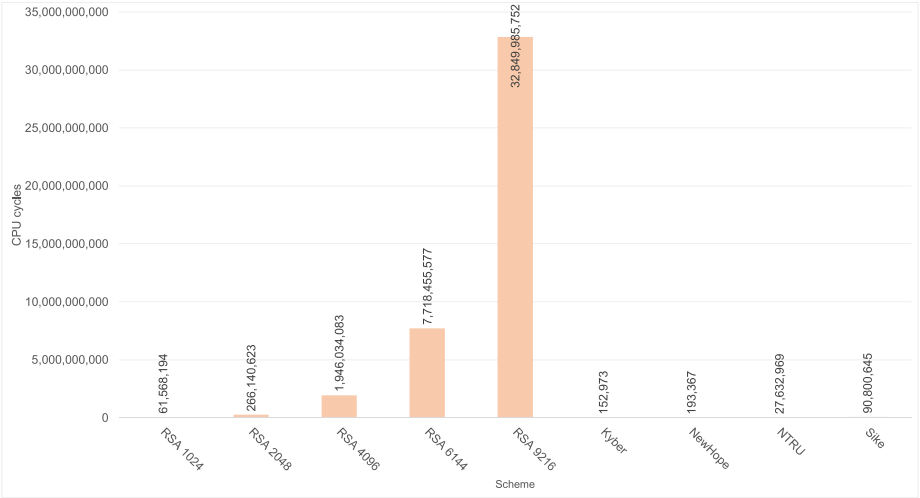



Figure 8: CPU cycles needed for key generation.

lattice-based schemes. Based on ciphertext sizes, `Sike` is the most fitting candidate as the ciphertext size it generates fits in one packet that TOR currently uses, but the CPU cycles `Sike` uses makes it not so appealing as it is more than standard RSA-1024. If the standard of splitting the ciphertexts is agreed or the basic packet size is increased to 1 024 bytes, both `Kyber` and `NTRU` become fitting.

Based on the benchmark measurements, we recommend using either `Kyber` or `NTRU` in TOR as `Kyber` needs the least CPU cycles and both ciphertext sizes fit

in two packets that TOR currently uses.

## 5.4 Experimental results of quantum-safe circuit builds

In this section the circuit build times are discussed on the `SweetOnions` TOR network. Table 13 contains the measurements for wrapping the layers of encryption, decapsulating one layer of encryption, the total circuit build time, the message sizes, and the number of packets required. The total circuit build time combines the wrapping of encapsulation layers and calculates the removal of layers three times. Furthermore, the time needed for circuit creation is also indicated in seconds. The measurements are done for the original `SweetOnions` implementation, the quantum-safe implementations, and the hybrid implementations. Message sizes and circuit build times are visualized in Figures 9 and 10.

In the following subsections the original implementation of TOR in `SweetOnions` measurements serve as the basis. Having these numbers makes it possible to make a comparison with the addition of quantum-safe cryptography and to validate the recommendation made previously about using `Kyber` or `NTRU`.

| Scheme | Wrap encryption layers | Remove one layer | Total circuit build | Message size (bytes) | Packets needed | Time needed |
|---|---|---|---|---|---|---|
| Original | 5 131 765 | 13 714 147 | 46 274 206 | 1 223 | 3 | 0.02s |
| Kyber | 1 371 999 | 917 080 | 4 123 240 | 3 248 | 7 | 0.0017s |
| NewHope | 1 618 934 | 1 119 668 | 4 977 938 | 4 832 | 10 | 0.002s |
| NTRU | 2 803 358 | 4 149 134 | 15 250 759 | 3 099 | 7 | 0.006s |
| Sike | 452 691 951 | 271 667 313 | 1 267 693 889 | 1 874 | 4 | 0.52s |
| Hybrid Kyber | 17 441 512 | 141 947 032 | 443 282 608 | 3 360 | 7 | 0.18s |
| Hybrid NewHope | 29 976 092 | 343 266 964 | 1 059 776 985 | 4 944 | 10 | 0.44s |
| Hybrid NTRU | 18 348 903 | 149 772 336 | 467 665 911 | 3 360 | 7 | 0.19s |
| Hybrid Sike | 458 781 848 | 312 030 657 | 1 394 873 819 | 2 320 | 5 | 0.58s |

Table 13: The CPU cycles needed for building a circuit and message sizes.

### 5.4.1 First quantum-safe results

When adding the quantum-safe schemes to the implementation, after our benchmark measurements, we expected that the circuit build times would take less CPU cycles for the lattice-based schemes (`Kyber, NewHope, NTRU`) and that the message sizes would require more packets due to the ciphertext sizes, than for the original implementation. The results confirm that the lattice-based schemes had less CPU cycles than RSA, but the increase in packets used is more than double. On the other hand, `Sike` takes more CPU cycles to build a circuit, but the increase in the number of packets required is minimal compared to the original implementation.

In the case of `Kyber` the CPU cycles required for circuit creation is 91% less than the original circuit build time. The size of the encrypted message is 266% larger than for the original implementation and requires 2.3 times more packets.

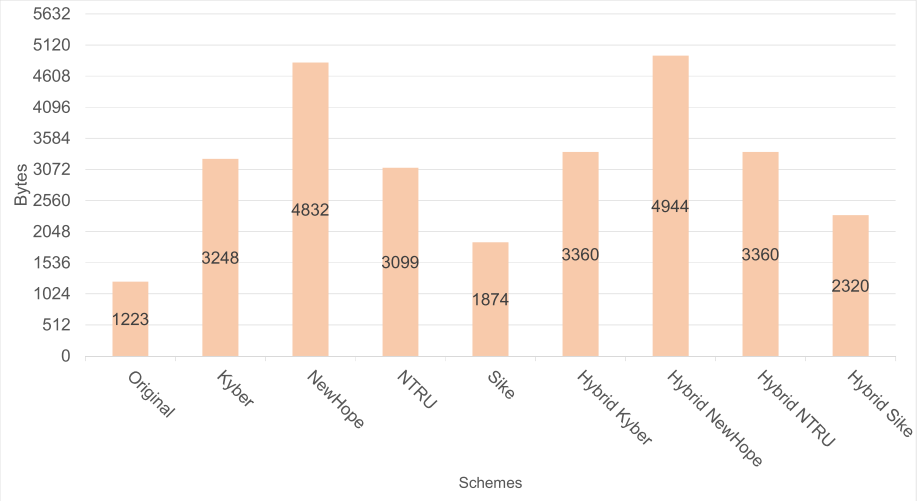



Figure 9: Encrypted message sizes in bytes. The maximum data one packet can currently store is 512 bytes.

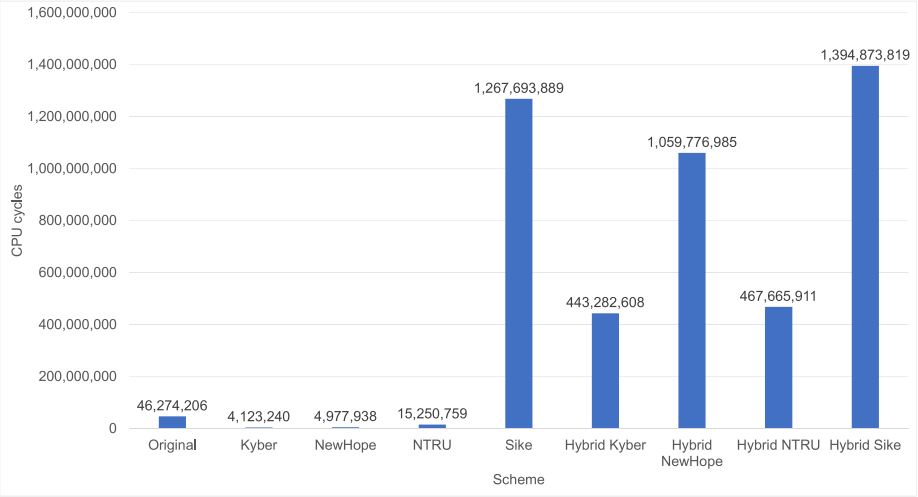



Figure 10: CPU cycles needed building a circuit.

The results for using `NewHope` are not unexpected after understanding the `Kyber` results. The circuit create time is slightly slower compared to `Kyber`, but still it is 89% faster than the original circuit build time. As `NewHope` has the largest ciphertext size of the three lattice-based schemes it needs 3.3 times more packets than the original implementation. The ciphertext size is 395% larger than the original implementation and 149% larger than `Kyber512`.

The final lattice-based scheme tested is `NTRU` and because of it having the smallest ciphertext size of the three lattice schemes we were hoping it to fit in less packets than `Kyber`. Even by having the least amount of ciphertext size, `NTRU` also requires the same number of packets as `Kyber`. `NTRU` has the most operations required among the lattice-based schemes - about 3.7 times more than `Kyber` and 3.1 times more than `NewHope`.

The testing of `Sike` was important, because so far only lattice-based schemes were tested. For `Sike` key generation and encapsulation, decapsulation took more time than for the lattice-based schemes - as `Sike` is based on supersingular isogeny - we were expecting to get different (larger) CPU cycles for building a circuit. Results show that circuit creation when using `Sike` is 27.4 times more CPU heavy than the original circuit build time, and 83.1 times heavier than the most computationally demanding lattice-based scheme `NTRU`. Although the circuit build time might make `Sike` undesirable to use, it has the smallest ciphertext size and the encrypted message size is 1847 bytes - 153% more than for the original implementation. However, even this increase makes it possible to fit the encrypted message in four packets.

Having access to the results we would like to make an update on the recommendation in Section 5.3. In order to have the least impact on the network load of TOR, instead of using `Kyber` or `NTRU`, `Sike` should be used. Although it requires more CPU operations than lattice-based schemes, the fact that encrypted messages fit into four packets means that the network load would not be affected in a major way and only a minor additional latency would be added compared to the RSA implementation.

Using a lattice-based scheme may be considered as they require less CPU cycles than `Sike`. If so, `Kyber` is a fitting candidate as it uses the same number of packets as `NTRU` but requires less CPU cycles. Using lattice-based schemes have implications on packet sizes and network load and this means two things:

1. Standards for splitting the ciphertext of the shared secret need to be introduced or an increase in packet size is required to fit the ciphertext in one packet,

2. Circuit build times require less CPU cycles, meaning that circuits are created faster, and the packets can be sent through the network quicker, which can cripple the network.

Current packet size of 4 096 bits (512 bytes) is a factor of 2 ($2^{12}$), thus the next packet size would be 8 192 bits (1 024 bytes; $2^{13}$), meaning the new standard message size is the double of the old. Sending two packets of the current size also means that the message size is doubled but can have further implications if the splitting is not standardized. Both mean that network load would double but increasing the packet size would be the better solution. This is because

during sending multiple packets some could get lost during transmission. If a packet is lost the TCP protocol requests the packets again, the client needs to resend all the packets, adding further latency and traffic to the network. If lattice-based schemes are used, circuits are created faster, meaning the client can send or resend packets faster if TCP packets are lost. Sending too many packets in quick succession can have an effect similar to a Denial of Service attack on the network.

### 5.4.2 Hybrid implementation results

Keeping part of the original implementation adds the security that RSA has against classical adversaries. Due to the need of using RSA with large keys (4 096, 6 144, 9 216 bits), we expected circuit build times to take more than for the original implementation and expected encrypted message sizes to be larger than for the quantum-safe implementations.

As the updated recommendation for the fully quantum-safe TOR was to use `Sike`, this is examined first. Using the combination of the original implementation with `Sike`, does not have a large increase in CPU cycles for building a circuit compared to `Sike` - only a 110% increase -, the encrypted message size increased by about 124%, which means it does no longer fit in four packets, but five.

Using the combination of the original implementation with `Kyber` is 107.5 times more computational heavy for building a circuit comparted to the purely quantum-safe implementation of `Kyber`. The encrypted message is also larger, but only 103% larger than for the purely quantum-safe, and this also fits in the same number of packets that purely quantum-safe `Kyber` uses.

The combination of RSA and `NTRU` performs a bit worse than hybrid `Kyber` for circuit creation, it requires 105% more CPU cycles. The encrypted message sizes are the same as for hybrid `Kyber`. This is due to using the same RSA key size to encrypt the ciphertext generated by the quantum-safe scheme.

Finally, the last lattice-based scheme `NewHope` was tested with the combination of the original implementation. Compared to the purely quantum-safe `NewHope`, the circuit build time requires about 212.9 times more CPU cycles. Similar to the purely quantum-safe schemes, the encryption size is the largest, but it still fits in the packets used for the purely quantum-safe implementation.

When using the hybrid implementation only the lattice-based schemes of the tested schemes have message sizes that fit into the same packet sizes as the purely quantum-safe schemes. This is due to the length that the large RSA encapsulation key adds. Thus, having a new standard about how to split the packet or increasing the packet size when transitioning to a hybrid quantum-safe TOR seems inevitable.

Based on the combined results, the recommendations can be updated. When using the hybrid implementations all the lattice-based schemes use the same number of packets as the purely quantum-safe ones (7 or 10), but circuit build times increase radically. Due to this, we recommend using `Sike` when transitioning to a quantum-safe TOR. Compared to purely `Sike` the CPU cycles are slightly increased for hybrid `Sike`, network load is increased by 166%, but still requires less packets than `Kyber` or `NTRU`.

## 5.5 Conclusions

To understand the results, it is important to note, that because the virtual machines are all on the same virtual network, the network latency is minimal. In reality TOR nodes are further away and if nodes that are on another continent are selected, the network latency is significant.

All of the experimental results illustrate that lattice-based schemes have better performances for key generation and for encryption/decryption than currently used RSA - as stated by [12]. In [12] it is also reported that larger encrypted message sizes are required, which is also confirmed by our experiments. None of the tested schemes fit in the number of packets that TOR currently uses thus, there is additional network load.

The tested supersingular isogeny-based scheme required more CPU cycles for key generation and for building a circuit than RSA, `Kyber, NewHope` or `NTRU`. The important fact to note is, that the usage of `Sike` generates a ciphertext of the shared secret that fits the current packet size of TOR, meaning the original packet sizes for TOR can be used. Additional packets are needed to fit the encrypted message, but only one, which does not contribute to large network load as lattice-based schemes.

To keep the security against classical adversaries in case of quantum-safe schemes prove to have flaws, the original implementation was combined with the quantum-safe schemes. The lattice-based hybrid schemes experienced a huge increase in CPU cycles, whereas the increase in `Sike` was not so marginal. Message sizes also increased, but this was only an issue for `Sike` as it required an extra additional packet.

Comparing to our initial thoughts on performance in Section 5.1.3 we notice that answering the third question for the research is not black and white. Recall the third sub-question: *What performance drawbacks are users likely to face in a quantum-safe TOR and are these drawbacks confirmed by the experimental results?* The CPU cycles for key generation and encapsulation/decapsulation does not increase when using lattice-based schemes (`Kyber, NewHope, NTRU`). However. the use of `Sike` and 4 096, 6 144, 9 216 bit variants of RSA would increase the CPU cycles for these operations. This trend can also be observed at

the circuit build times, where all the purely lattice-based schemes achieved less CPU cycles used, and the supersingular isogeny-based scheme achieved more CPU cycles used compared to the original `SweetOnions` implementation.

We confirm, that the public and private key sizes increase compared to currently used RSA. When using quantum-safe implementations the Directory Authorities will experience increased network traffic. This is going to further increase, when using the hybrid schemes as next to storing the large RSA public keys the Directory Authorities also need to store the quantum-safe public keys.

We also confirm, that the message sizes increase for all quantum-safe and hybrid schemes. Using any of the three lattice-based schemes result in more than double of current packets required. Using supersingular isogeny-based scheme - `Sike` - also adds at least one extra packet in order to fit messages. This can be appealing even if the circuit build time takes longer, as the aim is not to overload the network with sending too much traffic [27].

Now, after understanding the drawbacks of quantum-safe TOR it is possible to discuss the strengths and the weaknesses of the tested implementations. Of course, the biggest strength that all the tested implementations have in common is that they remain secure against quantum adversaries. Apart from this the other strength that the lattice-based schemes have is the efficient CPU cycle usage for key generation and circuit creation compared to RSA. The strength of `Sike` is that when used it does not add as heavy network load to TOR as lattices.

The weakness and the downside of using quantum-safe implementations is the message sizes and packets required to send them. `Kyber, NewHope` and `NTRU` all require at least 7 packets to send data through the circuit, which means that network load is doubled at minimum. As for `Sike`, apart from the increase of message sizes the CPU cycles also increase. Both of these downsides are inevitable as for the transitioning period from current to quantum-safe cryptography a hybrid implementation is the most fitting.

To conclude, based on our experiments `Sike` should be considered as the candidate to replace RSA in quantum-safe TOR. `Sike` has the least network load generated which is fundamental for TOR in order not to cause a Denial of Service. Quantum-safe TOR is possible, and no matter if lattice or supersingular isogeny-based schemes are considered in the future, network load will increase.

# 6 Conclusion

## 6.1 Own contribution

In this work we investigated the concept, possibility and implications of a quantum-safe TOR. We identified four sub-questions to guide us in answering the question what a quantum-safe TOR would look like. The sub-questions focused on the vulnerable parts of the current TOR network and the changes needed to make these parts quantum-safe, the implications of having a quantum-safe TOR, and the strengths and performance limits of the proposed quantum-safe TOR.

We argued that the most important part to be made quantum-safe first is the AES symmetric key, although this does not require any quantum-safe schemes. It can be done by doubling the key sizes and using AES-256 instead of AES-128. Secondly, we point out that the long-term and medium-term keys need to be replaced in order to prevent an adversary from impersonating the node. We listed the short-term keys to be changed lastly. As the design of TOR, circuits change every 10 minutes, which means the adversary has limited time to exploit the connection. This is a different approach compared to literature on quantum-safe TOR networks, both [12, 33] focus on the short-term keys in their work.

The AES keys need to be changed as soon as possible as they are already vulnerable to 'store-now-decrypt-later'-attacks. This update does not require the use of quantum-safe cryptographic schemes but would have a life changing effect. People living in oppressive regimes need to have life-long protection of their privacy which needs to be ensured. As for the long-term and medium-term keys, they should be considered for change in the near future. These keys are generated using asymmetric cryptography and even though NIST advise the use of 2048-bit RSA keys, TOR still uses 1024-bit keys. Although 1024-bit RSA keys have not yet been successfully factored, still updating to RSA-2048 should be done first, and only after this switch to a hybrid or purely quantum-safe scheme.

Updating TOR to a quantum-safe setting, we expected increase in CPU cycles for key generation and encapsulation/decapsulation, and increased message sizes due to quantum-safe encryption. To test these assumptions, we modified an implementation of TOR. Post-quantum cryptographic implementations are added from the Open Quantum Safe library to the original implementation, three lattice-based quantum-safe schemes (`Kyber, NewHope, NTRU`) and one supersingular isogeny-based quantum-safe scheme (`Sike`) are tested in a purely quantum-safe and a hybrid setting.

We observed that in the purely quantum-safe setting the CPU cycles only increase for `Sike` compared to the original implementation, but for the lattice-based schemes they are even less, which was also confirmed in [12]. The message sizes do increase for all tested schemes, but `Sike` makes it possible to add the

smallest amount of extra network load. For the lattice-based schemes the current packet size is not large enough to send the ciphertext of the key, and they would require modifications in related to packet sizes. Based on the current TOR implementation, if the usage of purely quantum-safe scheme is the goal we recommend the usage of `Sike` as it does not contribute hugely to network load as this is currently the main performance issue in TOR. In the transition phase where both classical and quantum-safe schemes are used in combination, the recommendation would still be to use `Sike` as it adds the least network load to the network. The Open Quantum Safe library is a research library and implementations can still improve.

Finally, based on the results the strengths and performance limitations were discussed. The overall strength the schemes have are security against quantum adversaries. More generally, the strengths and limitations are the trade-offs of the CPU cycles used and the message sizes of the schemes. The lattice-based schemes have low CPU cycle costs but large message sizes whereas the supersingular isogeny-based scheme has high CPU cycle costs but small message size.

To conclude, quantum-safe TOR is possible, but it will have some performance drawbacks that contribute to network load.

## 6.2 Future work

For future work it would be interesting to test other type of schemes e.g. code-based like `BIKE`. Testing the remaining lattice and isogeny-based schemes is also an interesting future topic as they might have better performance measurements than the ones currently available in the Open Quantum Safe library.

In the Open Quantum Safe library, the encapsulation of messages posed as a challenge. Current encapsulation works with a quantum-safe public key and generates a random (shared secret, ciphertext) tuple. Investigating the encapsulation more closely and making it possible to encapsulate actual messages to produce a (shared secret, ciphertext) tuple of the message would be very beneficial for future work.

One of the current attacks on TOR specifically target hidden services. Adding our proposed updates to the hidden services might not be enough to keep them quantum-safe as they are in the spotlight for attackers. Investigating the additional security measures that users hosting hidden services can make in order to remain quantum-safe could be rewarding.

As for field experiments, an implementation of TOR is available called `TorLAB` [36] that simulates TOR on a private network of Raspberry PIs. It would be beneficial to recreate the network and extend the measurements of our research to the network load. This would ensure a more realistic result for expected circuit build times, as now network latency is omitted.

# References


[1] Erdem Alkim et al. *FrodoKEM Learning With Errors Key Encapsulation*. https://frodokem.org/files/FrodoKEM-specification-20190330.pdf. Accessed on 2019-08-13.

[2] Felipe Astolfi, Jelger Kroese, and Jeroen van Oorschot. *I2P - The Invisible Internet Project*. Leiden University, 2015.

[3] Elaine Barker and Quynh Dang. *Recommendation for Key Management. Part 3: Application-Specific Key Management Guidance*. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf. Accessed on 2019-08-15.

[4] Alex Biryukov. *Adaptive Chosen Plaintext Attack*. https://link.springer.com/referenceworkentry/10.1007/978-1-4419-5906-5_545. Accessed on 2019-08-12.

[5] Joppe W. Bos et al. *Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem*. IEEE Symposium on Security and Privacy Conference Paper, 2015.

[6] Enrico Cambiaso et al. *Darknet Security: A Categorization of Attacks to the Tor Network*. Consiglio Nazionale delle Ricerche (CNR-IEIIT), 2019.

[7] Matthew Campagna et al. *Quantum Safe Cryptography and Security; An introduction, benefits, enablers and challenges*. ETSI, 2014.

[8] Safe Crypto. *NIST Round 2 Candidates*. https://www.safecrypto.eu/pqclounge/round-2-candidates/. Accessed on 2019-06-14.

[9] Dan. *TOR Node List*. https://www.dan.me.uk/tornodes. Accessed on 2019-06-14.

[10] Roger Dingledine and Nick Mathewson. *Tor Protocol Specification*. https://gitweb.torproject.org/torspec.git/tree/tor-spec.txt. Accessed on 2019-06-14.

[11] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The Second-Generation Onion Router*. 2004.

[12] Satrajit Ghosh and Aniket Kate. *Post-Quantum Forward-Secure Onion Routing (Future Anonymity in Today's Budget)*. ACNS Conference Paper, 2015.

[13] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. Proceedings, 28th Annual ACM Symposium on the Theory of Computing (STOC), 1996, pp. 212–219.

[14] Nicholas Hopper, Eugene Y. Vasserman, and Eric Chan-Tin. *How much anonymity does network latency leak?* ACM Trans. Info. Syst. Sec. 13, 2, Article 13, 2010. DOI: 10.1145/1698750.1698753.

[15] Tommaso Gagliardoni (IBM). *Future Proofing the Connected World: A Quantum-Resistant Trusted Platform Module*. 2018.

[16] InvisibleNet. *Invisible Internet Project (I2P)*. 2003.

[17] JonDoNYM. *Mixmaster Remailer*. https://anonymous-proxy-servers.net/en/help/jondo-live-cd18.html. Accessed on 2019-08-12.

[18] Burt Kaliski. *TWIRL AND RSA KEY SIZE*. https://web.archive.org/web/20170417095741/https://www.emc.com/emc-plus/rsa-labs/historical/twirl-and-rsa-key-size.htm. Accessed on 2019-08-15.

[19] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. CRC Press, 2014.

[20] LeonHeTheFirst. *SweetOnions - Making Onion Routing Great Again*. https://github.com/LeonHeTheFirst/SweetOnions. Accessed on 2019-07-29.

[21] Bingdong Li et al. *An Analysis of Anonymity Technology Usage*. TMA'11 Proceedings of the Third international conference on Traffic monitoring and analysis, 2011.

[22] Ewen Macaskill and Gabriel Dance. *NSA Files: Decoded*. https://www.theguardian.com/world/interactive/2013/nov/01/snowden-nsa-files-surveillance-revelations-decoded#section/1. Accessed on 2019-06-14.

[23] Bernard Marr. *How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read*. https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/#7b9dbc3260ba. Accessed on 2019-08-12.

[24] Nick Mathewson. *Mixminion: A Type III Anonymous Remailer*. https://www.mixminion.net. Accessed on 2019-08-12.

[25] TOR Metrics. *Performance*. https://metrics.torproject.org/torperf.html. Accessed on 2019-06-14.

[26] TOR Metrics. *Users*. https://metrics.torproject.org/userstats-relay-country.html. Accessed on 2019-06-14.

[27] Tor Blog - mikeperry. *Tor's Open Research Topics: 2018 Edition*. https://blog.torproject.org/tors-open-research-topics-2018-edition. Accessed on 2019-08-23.

[28] NIST. *Post-Quantum Cryptography Standardization*. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization. Accessed on 2019-06-14.

[29] pandoragami. *RSA maximum bytes to encrypt, comparison to AES in terms of security?* https://security.stackexchange.com/questions/33434/rsa-maximum-bytes-to-encrypt-comparison-to-aes-in-terms-of-security. Accessed on 2019-08-28.

[30] Open Quantum Safe Project. *C library for quantum-resistant cryptographic algorithms*. https://github.com/open-quantum-safe/liboqs. Accessed on 2019-07-29.

[31] Open Quantum Safe Project. *Python 3 bindings for liboqs*. `https://github.com/open-quantum-safe/liboqs-python`. Accessed on 2019-07-29.

[32] Python-Future. *futurize: Py2 to Py2/3*. `https://python-future.org/futurize.html`. Accessed on 2019-07-29.

[33] John M. Schanck, William Whyte, and Zhenfei Zhang. *Circuit-extension handshakes for Tor achieving forwards secrecy in a quantum world*. Proceedings on Privacy Enhancing Technologies, (4), 2016, pp. 219–236.

[34] Peter W. Shor. *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*. SIAM Journal on Computing, 1995.

[35] Nigel P. Smart. *Cryptography Made Simple*. Springer, 2016.

[36] Dark Web Solutions. *TorLAB*. `https://github.com/dws-pm/TorLAB`. Accessed on 2019-08-11.

[37] Douglas Stebila and Michele Mosca. *Post-Quantum Key Exchange for the Internet and the Open Quantum Safe Project*. 2017.

[38] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. *Anonymous Connections and Onion Routing*. SP '97. Washington, DC, USA: IEEE Computer Society, 1997, pp. 44–.

[39] TOR. *Download Tor Source Code*. `https://www.torproject.org/download/tor/`. Accessed on 2019-07-29.