



**BERGISCHE
UNIVERSITÄT
WUPPERTAL**

BACHELOR THESIS

Comparing Post-Quantum Instantiations of the TLS 1.3 Handshake

Jonas Dinspel

20.02.2026

Chair for IT Security and Cryptography
University of Wuppertal

First Examiner: **Prof. Dr.-Ing. Tibor Jager**

Second Examiner: **Jun.-Prof. Dr.-Ing Malte Mues**

Use of AI

By submitting this thesis, I declare that I have carefully read and followed the university's guidelines and the instructions of the ITSC research group on the use of AI.



Eidesstattliche Versicherung

Affirmation in lieu of an oath

Name, Vorname/ surname, first name

Matrikelnummer/ student ID number

☐ Bachelorarbeit/ bachelor's thesis

☐ Masterarbeit (auch Staatsexamen)/ master's thesis

mit dem Titel/ title

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt; dies umfasst insbesondere auch KI-Systeme, Software und Dienste zur Sprach-, Text- und Medienproduktion. Ich erkläre, dass für den Fall, dass die Arbeit in unterschiedlichen Formen eingereicht wird (z. B. elektronisch, gedruckt, geplottet, auf einem Datenträger) alle eingereichten Versionen vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Die Strafbarkeit einer falschen eidesstattlichen Versicherung ist mir bekannt, namentlich die Strafandrohung gemäß § 156 Strafgesetzbuch (StGB) bis zu drei Jahren Freiheitsstrafe oder Geldstrafe bei vorsätzlicher Begehung der Tat bzw. gemäß § 161 Abs. 1 StGB bis zu einem Jahr Freiheitsstrafe oder Geldstrafe bei fahrlässiger Begehung.

I hereby affirm in lieu of an oath that I have completed this thesis with the above title independently and without unauthorized assistance from third parties (in particular academic ghostwriting). I have not used any other sources or aids than those indicated; this includes in particular AI-systems, software and services for language, text, and media production. In the event that the work is submitted in different formats (e.g. electronically, printed, plotted, on a data carrier), I declare that all the submitted versions are fully identical. I have not previously submitted this work, either in the same or a similar form to any examining authority.

I am aware of the criminal liability of a false Affirmation in lieu of an oath, namely the threat of punishment according to § 156 StGB up to three years imprisonment or fine for intentional committal of the offence or according to § 161 Abs. 1 StGB up to one year imprisonment or fine if committed by negligence.*

**Please be aware that solely the German version of the Affirmation in lieu of an oath ("Eidesstattliche Versicherung") is the official and legally binding version.*

Ort, Datum/ place, date

Unterschrift/ signature

Belehrung

Official notification

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 156 StGB (German Criminal Code): False declaration in lieu of oath

Whoever falsely makes a declaration in lieu of an oath before an authority which is competent to administer such declarations or falsely testifies whilst referring to such a declaration incurs a penalty of imprisonment for a term not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Strafflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

§ 161 StGB (German Criminal Code): Negligent false oath; negligent false declaration in lieu of oath

(1) Whoever commits one of the offences referred to in sections 154 to 156 by negligence incurs a penalty of imprisonment for a term not exceeding one year or a fine.

(2) No penalty is incurred if the offender corrects the false statement in time. The provisions of section 158 (2) and (3) apply accordingly.

Abstract

Some Advice. Think of the abstract as a short version of your thesis. Motivate the topic of your thesis, and give a brief summary of its contents. Keep in mind that the abstract (and the remainder of your thesis) should be comprehensible for fellow students of yours. It is often expected that abstracts do not exceed one page.

Contents

Abstract	iv
1 Introduction	1
2 Related Works	3
3 Preliminaries	4
3.1 TLS 1.3 Handshake	4
3.1.1 Key-Exchange	6
3.1.2 Signatures	10
3.1.3 Extensions	11
3.2 Post-Quantum Cryptography	13
3.2.1 Shors Algorithm	13
3.2.2 ML-KEM	13
3.2.3 HQC	14
3.2.4 Hybrid Usage	14
3.3 Security Strength Categories	14
3.4 Illustrated Components	15
4 Method	16
4.1 Limitations	16
4.2 Capabilities	17
4.3 Data Source	17
4.4 Calculator	18
4.4.1 Underlying Equation	18
4.4.2 UI	19
5 Results	21
5.1 Benchmarking	21
5.2 Implications for Traffic	22
5.3 Use Cases	22
6 Conclusion	23
6.1 Relevance	23
6.2 Future Work	23

1 Introduction

With the continuous progress in the development of quantum computers, new challenges arise alongside of their many benefits. One, if not the biggest challenge for the field of cryptography, is the threat posed to widely used public key cryptography. These public key cryptography schemes help to ensure the integrity, confidentiality and authenticity of modern-day communications. They rely on complex mathematical problems like the factorization of large integers or the discrete logarithm problem, which, without the right information or keys, cannot be solved efficiently with today's most powerful computers. Algorithms such as shor's algorithm are able to efficiently solve these problems in polynomial time. For example, in encryption schemes, attackers would be able to derive shared secrets used in public key cryptography. Using the shared secret an attacker can easily decrypt any data encrypted using this secret, which leads to the loss of confidentiality and forward security, especially in the face of collect now, decrypt later attack schemes. Additionally, attackers gain the ability to fake signatures, leading to the loss of authenticity as well. These risks underline the urgency of a fast transition to post-quantum public key cryptography to ensure the confidentiality and authenticity of transmitted data in the future. One of the affected protocols is Transport Layer Security [TLS], which is used for a wide range of applications from simply surfing the internet or using instant messaging to voice over IP. Especially the handshake, which relies on asymmetric cryptography, is susceptible to post quantum attacks, but the symmetric encryptions used in the following data transfer are not resistant either. Even the most recent TLS1.3 standard itself is not inherently quantum-secure. There already are hybrid solutions which combine classical encryptions with modern post-quantum algorithms like ML-KEM, which are already included in current versions of web browsers. As with all increases in security, the transition to post-quantum cryptography comes at a cost, as quantum-secure encryptions tend to require more computational power and produce larger cryptographic objects, thus an overall decrease in the performance of relying protocols is expected. Benchmarks for these hybrid solutions and already exist, but it is still hard to compare different cryptographic TLS configurations against each other. To address this issue and further support the post-quantum transition of TLS, this thesis will introduce a tool to view, create and compare different TLS configurations, including classic, hybrid and post-quantum schemes. This will give a better overview of the current state of research and understanding

of the implications on performance of the transition to post-quantum TLS, thus giving easier approaches for further research.

2 Related Works

There already is a broad range of research regarding post-quantum TLS.

3 Preliminaries

In following we will discuss the TLS 1.3 handshake and its messages and components to give a solid understanding of mechanisms and schemes included in the proposed formula and the calculator using it.

3.1 TLS 1.3 Handshake

The TLS 1.3 Handshake, as described in [?], is the backbone of today's communication via the internet, as it enables peers to communicate privately without having to agree on any secret in advance. It is the successor of TLS 1.2, offering improved security and performance, only allowing robust encryption schemes and reducing the minimum amount of messages exchanged until the connection is secured. The handshake protocol is used to negotiate the security parameters of a connection. Handshake messages are supplied to the TLS record layer, where they are encapsulated within one or more `TLSP Plaintext` or `TLSCiphertext` structures which are processed and transmitted as specified by the current active connection state. Figure 3.1 shows a simplified overview of the handshake, reduced to messages which contain cryptographic objects used in the proposed calculator:

These four messages sent, `Client Hello`, `Server Hello`, `Server Certificate` and `Server Certificate Verify` contain every information needed to agree on a shared secret without ever revealing it to any possible third party as well as verifying the server's identity. This shared secret is used to symmetrically encrypt any further messages sent. In real-world situations these messages contain a lot of additional information, but for the proposed use case the transmitted cryptographic objects as shown in the figure are fully sufficient. With the TLS handshake three security properties are to be achieved:

- **Confidentiality** - Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information [?].
- **Integrity** - Guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity [?].
- **Authenticity** - The property that data originated from its purported source [?].

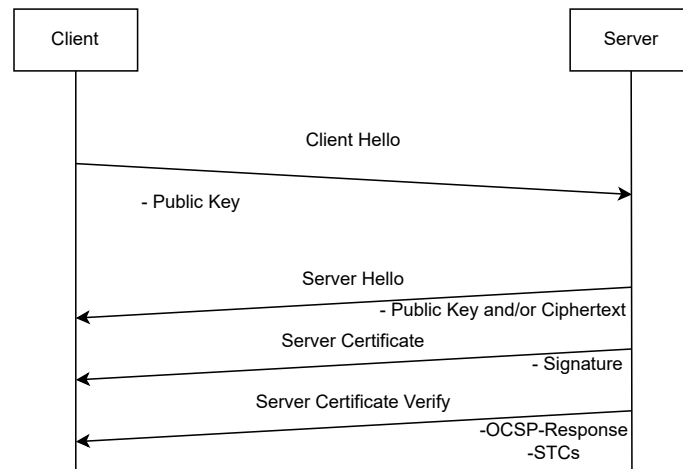


Figure 3.1: TLS 1.3 Handshake, reduced to transmitted cryptoobjects

Client Hello The client hello message contains information about the clients supported protocols and ciphersuites and protocols as well as an initial keyshare, which consists of public keys for multiple different key exchange schemes. The latter is new to the TLS 1.3 handshake, as TLS 1.2 did not include key within the **Client Hello** but waited for the servers supported ciphersuites. If the server accepts one of the shared keys an additional roundtrip is saved. The client hello includes the following four components [?]

- Client Random Data - this is used for several hashes within the handshake to ensure the **integrity** of shared data
- The clients supported cipher suites
- A list of public keys the server might see eligible for key exchange
- Protocol versions which are supported by the client

Within the **Client Hello** the base for the key exchange and further handshake is set, as the client shares every information needed to continue the handshake.

Server Hello Within the **Server Hello** the server agrees on a proposed protocol and ciphersuite based on the clients offer. The public key or ciphertext needed for the chosen ciphersuite is also included within this message. The server hello message includes the following four components [?]:

- Server Random Data

- The selected cipher suite
- A public key for key exchange
- The negotiated protocol version

Server and Client now have the information to calculate the shared secret, which is used to encrypt the rest of the handshake. With completion of this step **confidentiality** of any further messages against third parties is ensured.

Server Certificate The **Server Certificate** message conveys the server's certificate, which contains the server's public key's signature, to the client. Sending a certificate is mandatory for any server if a secured connection is to be established. The contained signature is proof of the server's identity. Proof of ownership is made in an additional message, **Server Certificate Verify**. The client on the other hand only must send a certificate if the server requested client authentication via **CertificateRequest** [?]. The **Server Certificate** message contains the server's certificate or certificate chain and ensures the **authenticity** of the established connection.

Server Certificate Verify The **Server Certificate Verify** message contains, if agreed on, a **OCSP-Response** used for **OCSP-Stapling** and a list of **Signed Certificate Timestamps SCT** used by **Certificate Transparency**. The latter being mandatory in most modern web-browsers since [wann wurde google nochmal impersonated?]. With this message **authenticity** and **integrity** are further improved.

3.1.1 Key-Exchange

There are three different available approaches to agree on a shared secret. Public-key encryption used for key-exchange such as RSA or ECDHE, referred to as **PKE**, and Key-Encapsulation Mechanisms, referred to as **KEM**. The third method available are hybrid approaches, which combine **KEMs** and **PKI**.

The key-exchange is based on asymmetric cryptography, in which two peers agree on a secret without having to exchange any secret in advance. A message m encrypted using the public key pk of a pair (pk, sk) can only be decrypted using the private key sk . The way encryption and decryption work and how pk and sk are generated depend on the used scheme. The key-exchange is mandatory to achieve **confidentiality**.

Public-Key Encryption

A public-key encryption scheme is a tuple of probabilistic, polynomial-time algorithms (Gen, Enc, Dec) that satisfies the following as described in [?]:

1. The key-generation algorithm **Gen** takes as input a security parameter 1^n and outputs a pair of keys (pk, sk) . We refer to the first of these as the public key and the second as the private key. We assume for convenience that pk and sk each have length at least n , and that n can be determined from pk, sk .
2. The encryption algorithm **Enc** takes as input a public key pk and a message m from some underlying plaintext space. It outputs a ciphertext c , and we write this as $c \leftarrow Enc_{pk}(m)$.
3. The decryption algorithm **Dec** takes as input a private key sk and a ciphertext c , and outputs a message m or a special symbol \perp denoting failure. We assume without loss of generality that **Dec** is deterministic, and write this as $m := Dec_{sk}(c)$.

We require that for every n , every (pk, sk) output by $Gen(1^n)$, and every message in the appropriate underlying plaintext space, it holds that

$$Dec_{sk}(Enc_{pk}(m)) = m.$$

Exchanging an encrypted message via PKE is described in the following figure:

Within the TLS handshake PKE is used for key-exchange rather than encrypting messages. The following figure describes, strongly simplified, how two parties can agree on a shared secret $K = K'$ using a PKE scheme. The figure does rather represent the idea behind using PKE schemes for key-exchange than the actual execution of these schemes:

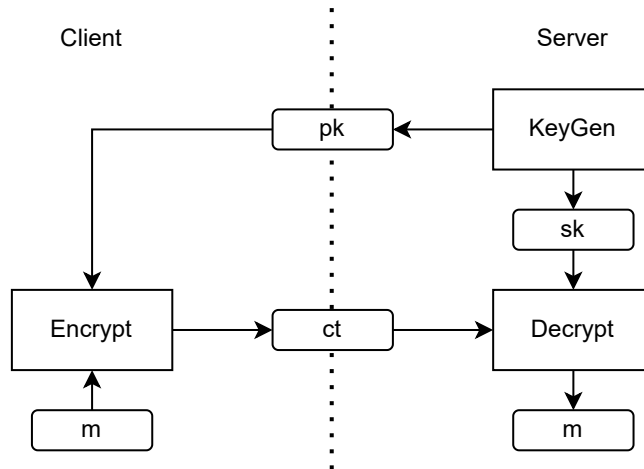


Figure 3.2: Using PKE to encrypt and decrypt a message using a pair of public and private keys

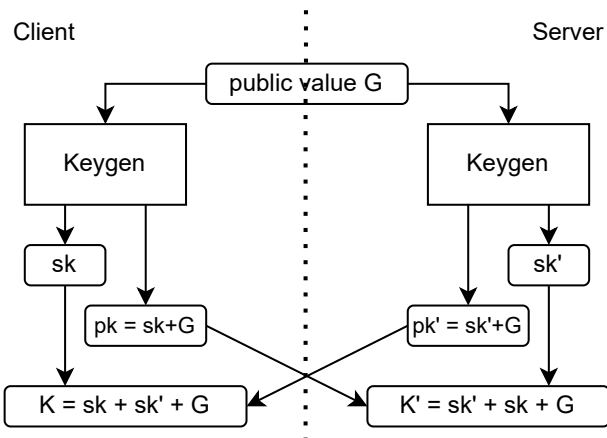


Figure 3.3: Using PKE to agree on a shared secret

Using PKE, client and server agree on a shared secret by exchanging their public keys derived from a shared number, from which the shared secret can be calculated. During the handshake both **Client Hello** and **Server Hello** will include the respective public key. The authenticity of the servers public key is backed by the servers certificated, which will be discussed later.

Key Encapsulation Mechanisms

Similar to PKEs KEMs can be used to agreed on a shared secret over an unsecured channel. A KEM Π consists of four components as proposed in [?]

1. $\Pi_{ParamSets}$: A collection of parametersets.
2. Π_{KeyGen} : The key-generation algorithm. An efficient probabilistic algorithm that accepts a parameter set $p \in \Pi_{ParamSets}$ as input and produces an encapsulation key ek and a decapsulation key dk as output.
3. Π_{Encaps} : The encapsulation algorithm. An efficient probabilistic algorithm that accepts a parameter set $p \in \Pi_{ParamSets}$ and an encapsulation key ek as input and produces a shared secret key K and a ciphertext c as output.
4. Π_{Decaps} : The decapsulation algorithm. An efficient deterministic algorithm that accepts a parameter set $p \in \Pi_{ParamSets}$, a decapsulation key dk , and a ciphertext c as input and produces a shared secret key K' as output.

The key-encapsulation correctness experiment for a KEM Π and parameter set $p \in \Pi_{ParamSets}$ consists of the following three steps:

1. $(ek, dk) \leftarrow \Pi_{KeyGen}(p)$
2. $(K, c) \leftarrow \Pi_{Encaps}(p, ek)$
3. $K' \leftarrow \Pi_{Decaps}(p, dk, c)$

The KEM is correct if, for all $p \in \Pi_{ParamSets}$, the correctness experiment for p results in $K = K'$ with all but negligible propability. A key-exchange using a KEM is executed as shown in the following figure:

Within the Handshakes key-exchange, (ek, dk) will be generated by Π_{KeyGen} on the clients side, and the ek will be sent with the **Client Hello**. The server will use the recieved ek to run Π_{Encaps} to recieve its copy of the shared secret K and generate the ciphertext c which will be sent in the **Server Hello**. Using c the client can run Π_{Decaps} to recieve its own copy of the shared secret, K' .

Hybrid Key Exchange

Hybrid key exchange utilizes a combination of PKE and KEM, where each handshake message contains a concatenation of public key pk and encapsulation key ek or ciphertext ct as described in [?]. Thus the **Client Hello** will include the public key of the used PKE scheme and the encapsulation key of the used KEM scheme. The **Server Hello** will include the public key of the used PKE scheme and the

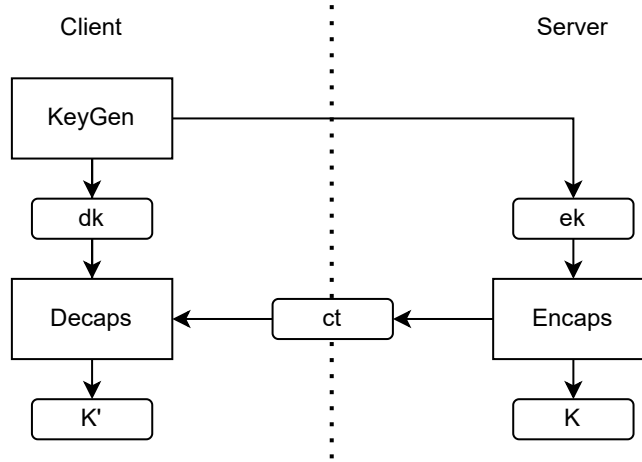


Figure 3.4: Using KEMs to agree on a shared secret

ciphertext containing the encapsulated public key of the used KEM scheme. With both messages containing two cryptographic objects, these messages average size is larger than using a single approach. The advantage of hybrid approaches is increased forward security, as the newer KEMs are additionally secured by long tested PKE, which can be expected to be mathematically safe, as flaws within different KEMs math might still be discovered.

3.1.2 Signatures

A signature schme is, as defined in [?], a tuple of propabilistic polynomial-time algorithms **Gen**, **Sign** and **Verify** satisfying the following:

1. The **key-generation algorithm** **Gen** takes as input a security parameter 1^n and outputs a pair of keys pk and sk . These are called public key and private key respectively. We assume that pk and sk have a length n which can be determined from pk and sk .
2. The **signin algorithm** **Sign** takes as input a private key sk and a message m from some underlying message space that may depend on pk . It outputs a signature σ , and we write this as $\sigma \leftarrow \text{Sign}_{sk}(m)$
3. The **deterministic verification algorithm** **Vrfy** takes as input a public key pk , a message m , and a signature σ . It outputs a bit b with $b = 1$ meaning *VALID* and $b = 0$ meaning *INVALID*. We write this as $b := \text{Vrfy}_{pk}(m, \sigma)$

We require that that for every n , every (sk, pk) output by $Gen(1^n)$ and every message m in the appropriate underlying plaintext space, it holds that

$$\text{Vrfy}_{pk}(m, \text{Sign}_{sk}(m)) = 1.$$

We say σ is a valid signature on a message m if $\text{Vrfy}_{pk}(m, \sigma) = 1$. In context of the TLS 1.3 Handshake signature schemes are used within the certificates, as these are signed by the Certificate Authority, and in the **Server Certificate Verify** message, where the server sends a signed hash of the handshake. The latter guarantees that the message were exchanged between the same peers and were not tampered with. This guarantees **integrity** and **authenticity**. Authenticity is ensured by the **Server Certificate** message, which contains the servers certificate signed by the CA. Integrity is ensured in the **Server Certificate Verify** message where a hash of the handshake is signed by the server which is already authenticated at this point. If messages or certificates were manipulated or signed by potential attackers the **Vrfy** algorithm would return **INVALID**, as $\text{Vrfy}_{pk}(m, \text{Sign}_{sk}(m)) = 1$ would no longer hold.

3.1.3 Extensions

Aside the regular handshake messages there are three TLS extensions which impact the security of the handshake and transmit some sort of cryptographic object are available for created configurations.

Encrypted Client Hello

Encrypted client hello (ECH) encrypts, as proposed in [?], the client hello message, adding an additional layer of security and privacy to the key-exchange. ECH hides the Server Name Indication (SNI) within the **Client Hello** from any possible attackers, the internet service provider ISP or other observers. Only the target IP address stays visible. With ECH, the ClientHello message part is split into two separate messages: an inner part and an outer part. The outer part contains the non-sensitive information such as which ciphers to use and the TLS version. It also includes an outer SNI. The inner part is encrypted and contains an inner SNI. The outer SNI is a common name that represents that a user is trying to visit an encrypted website on the chosen DNS provider. Because the DNS provider controls that domain we have the appropriate certificates to be able to negotiate a TLS handshake for that server name.

The inner SNI contains the actual server name that the user is trying to visit. This is encrypted using a public key and can only be read by the DNS provider. Once the handshake completes the web page is loaded as normal, just like any other

website loaded over TLS.

This further increases **confidentiality** of send data. With ECH enabled two public keys are used within the **Client Hello**, thus the public key size doubles.

OCSP-Stapling

Online Certificate Status Protocol stapling, short OCSP-stapling is a standart for checking the revocation status of digital certificates, ensuring that revoked certificates are not used to impersonate peers further increasing authenticity of connections. As decribed in [?] OCSP-stapling additionally avoids transmission of Certificate Refocation List and therefor safes bandwidth. In order to indicate their desire to receive certificate status information, clients may include an extension of type **statusRequest** in the client hello. Servers that receive a client hello containing the **statusRequest** extension may return a suitable certificate status response to the client along with their certificate. This the OCSP-response is sent within the **Server Certificate Status** message right after the **Server Certificate** message. The OCSP-response is defined in [?] as follows, but reduced to the responses portion containing relevant sizes for the proposed calculator. A definitive response message is composed of:

- Version of the response syntax
- Name of the responder
- Responses for each of the certificates in a request
- Optional extensions
- Signature algorithm OID
- Signature computed across hash of the response

Within the calculator, only the signatur computed across the hash of the response is taken into consideration. The main goal of OCSP-Stapling is to increasy hand-shake efficiency, as neither a CRL is needed nor does the client need to check the certificates status with the corresponding CA. Nontheless does OCSP-Stapling add to the **authenticity** of the established connection.

Cerificate Transparency

Certificate transparency, as described in [?], aims to mitigate the problem of mi-sissued certificates by providing publicly auditable, append-only, untrusted logs of all issued certificates. The logs are publicly auditable so that it is possible for anyone to verify the correctness of each log and to monitor when new certificates

are added to it. The logs do not themselves prevent misissuance, but they ensure that interested parties can detect such misissuance. Each log consists of certificate chains, which can be submitted by anyone. In order to avoid logs being spammed into uselessness, it is required that each chain is rooted in a known CA certificate. When a chain is submitted to a log, a signed certificate timestamp (SCT) is returned, which can later be used to provide evidence to clients that the chain has been submitted. These SCTs will be appended to the **Server Certificate Status** message within the SignedCertificateTimestampList. This list contains the last n STCs, but must at least contain 1 STC. Certificate Transparency gives those who are responsible for given domains the ability to check whether unexpected certificates have been issued to their domain. Similar to OCSP-stapling, adds to the authenticity of a connection. Certificate Transparency is mandatory since in [wann war das mit google nochmal?] attackers managed to issue a certificate on googles domain and intercept usernames and passwords of users connecting to google through their counterfeit certificate. By using Certificate Transparency, the **integrity** and **authenticity** of the established connection is increased.

3.2 Post-Quantum Cryptography

In this section we discuss threats imposed on the TLS 1.3 Handshake by cryptographic relevant quantum computers and schemes which are able to mitigate posed threats. Schemes which are vulnaerable to these threats will be referd to as **classic cryptography**.

3.2.1 Shors Algorithm

Shors algorithm [?] is the major threat which the Handshake is faced with. Using Shors algorithm potential attackers gain the ability to factor large numbers in polynomial time. This especially poses a thread to the widely used RSA algorithm, as its security is solely based on the assumption that large integers can not be factorized within any feasible timespan.

3.2.2 ML-KEM

Module-Lattice-Based KEM, which is based on CRYSTALS-Kyber, is a post-quantum key-exchange scheme which is already standardized by NIST. Within the standardized submission are three parametersets, 512, 768 and 1024, ranging from NIST-Level 1 to 5[].

3.2.3 HQC

HQC, short for **Hamming Quasi-Cyclic**, is another post-quantum key-exchange scheme which is already standardized by NIST. It is a code-based KEM based on the hardness of solving the Quasi-Cyclic Syndrom Decoding[]. Similar to ML-KEM its standardized with three parametersets, HQC-1, HQC-3 and HQC-5, again ranging from NIST-Levels 1 to 5.

3.2.4 Hybrid Usage

In addition to purely post-quantum key-exchange there also are hybrid solutions, which combine on of the proposed post-quantum schemes with algorithms from classic cryptography such as RSA or ECDHE. This drastically reduces the attack surface of mathematically breaking the encryption as schemes as RSA and ECDHE are used for a such a long period of time that most if not all exploits in their implementation and appliance are already fixed.

3.3 Security Strength Categories

Any instantation of a post-quantum scheme standardized by NIST is classified in security strength categories as described in [?]. These categories can be referd to as NIST-Levels, ranging from 1 to 5. With the uncertainties of yet to be discoverd quantum attacks and the limited ability to predict performance metrics for future quantum computers, these categories are defined by reference primitives rather than bits of security. These will serve as the base of a wide variety of metrics relevant in practical security. Each level is defined as follows:

1. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 128-bit key (e.g. AES128)
2. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 256-bit hash function (e.g. SHA256/ SHA3-256)
3. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 192-bit key (e.g. AES192)
4. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for collision search on a 384-bit hash function (e.g. SHA384/ SHA3-384)

5. Any attack that breaks the relevant security definition must require computational resources comparable to or greater than those required for key search on a block cipher with a 256-bit key (e.g. AES 256)

These categories can also be applied to schemes not standardized by nist, such as FrodoKEM.

3.4 Illustrated Components

Next we discuss how the proposed components will be illustrated and used with the formula which calculates the size of the handshakes cryptographic objects. The selection of at least one key-exchange scheme or signature scheme is mandatory. Usage of available extensions is optional.

Key-Exchange

Depending on the selected key-exchange method a combination of public key pk and ciphertext ct is used in the calculation, according to the following table:

	Client Hello	Server Hello
PKI	pk	pk
KEM	pk	ct
hybrid	$pk_{PKI} + pk_{KEM}$	$pk + ct$

Table 3.1: Cryptographic objects used from key-exchange

Signatures

For the selected signature scheme the size of the resulting signature σ which is part of the **Server Certificate** message is used in the calculation.

Extensions

Depending on the extension different elements are taken into the calculation:

ECH - the additional public key is taken into the calculation by doubling the size of the public key from the selected key exchange scheme

OCSP-Stapling - the signature part of the attached OCSP-response σ_{OCSP} is used in the calculation, the rest of the response is not used

Certificate transparency - the attached list with given length l of SCTs σ_{SCT} is used in the calculation

4 Method

We propose an UI based calculator to configure and compare the size of transmitted cryptographic objects of different instantiations of the TLS 1.3 handshake. These instantiations consist of key-exchange and used signature schemes as well as different TLS extensions. Available extensions are **OCSP-Stapeling**, **Certificate Transparency** and **Encrypted Client Hello**. The underlying datasets for key-exchange and signing include different pre- and post-quantum schemes with different parametersets available for each scheme.

classic	post-quantum
DHE	HQC
ECDHE	ML-KEM
FFDHE	FrodoKEM
RSA	X-Wing
RSA-OAEP	

Table 4.1: Aviable key-exchange schemes

For signature schemes, there is a broad spectrum of different post-quantum schemes with different NIST-Status, including on-ramp and not fully proven as secure applications. As this calculator focuses on post-quantum instantiations of the TLS 1.3 handshake, all included schemes from classic cryptography are those which are included in [rfc8446], where the TLS 1.3 handshake is formally defined.

4.1 Limitations

The formula which is used by the calculator only includes the size of cryptographic objects during the handshake, stopping at and already excluding the shared secret K . Everything aside the cryptographic objects in each payload is not taken into consideration. This includes package information, additional extensions and headers, even these used in OCSP or Certificate Transparency. The computational effort of used schemes is not taken into consideration either. By excluding these factors I ensure the compareability and consitancy of generated results, regardless of connected host or computing machine.

scheme	status	scheme	status
EdDSA	Pre-quantum	CROSS	On-ramp
RSA	Pre-quantum	Feast	On-ramp
DHE	Pre-quantum	Falcon	t.b.s
UOV	On-ramp	Hawk	On-ramp
SQIsign	On-ramp	Less	On-ramp
SNOVA	On-ramp	MAYO	On-ramp
SLH-DSA	FIPS	ML-DSA	FIPS
SDitH	On-ramp	MQOM	On-ramp
RYDE	On-ramp	Mirath	On-ramp
QR-UOV	On-ramp	PERK	On-ramp
RSA-PSS	Pre-quantum	ECDSA	Pre-quantum
ED	Pre-quantum		

Table 4.2: Aviable signature schemes

4.2 Capabilities

This calculator can be used to quickly compare the size of transmitted cryptographic objects during **Client Hello**, **Server Hello**, **Server Certificate** and **Server Certificate Verify** as well as the total size, without setting up and reconfiguring a dedicated server. The following objects are included:

- The used public key pk , which is doubled if the extension Encrypted Client Hello is enabled
- Transmitted ciphertext ct , which will be used if the key exchange is handled by a KEM
- Signatures σ
- The signature of OCSP-responses σ_{OCSP} , if OCSP-Stapeling is enabled
- The signature of SCTs σ_{SCT} , if Certificate Transparency is enabled

4.3 Data Source

The data used for calculating the size of the key-exchange is sourced from their individual NIST-publications [?] [?] or compareable works[RSA][RSA-OAEP][Frodo][X-Wing][ECDHE][FFDHE]. Each signature dataset is sourced from the repository of the "PQ Signatures Zoo" open source project [?]. By using as consistent as possible sources for each dataset we further ensure the compareability of generated results.

4.4 Calculator

The calculator offers the ability to compare the size of cryptographic objects exchanged within the TLS 1.3 handshake. This includes the key exchange, signatures and OCSP-stapling, Encrypted Client Hello, and Certificate Transparency. The key exchange offers a wide variety of schemes from classical and post-quantum cryptography, as well as hybrid key exchange, which can be freely configured from offered PKI and KEM schemes.

4.4.1 Underlying Equation

The total size is calculated using the following equation:

$$y_1 * a * pk_{client1} + y_2 * a * pk_{client2} + y_3 * pk_{server} + y_4 * ct + y_5 * \sigma + y_6 * \sigma_{ocsp} + y_7 * b * \sigma_{ct} \quad (4.1)$$

$$\text{s.t.} \quad pk_{client1}, pk_{client2}, pk_{server}, ct, \sigma, \sigma_{ocsp}, \sigma_{ct} \in \mathbb{N} \quad (4.2)$$

$$y_i \in \{0, 1\}, \forall i \in \{1, 2, 3, 4, 5, 6\} \quad (4.3)$$

$$a \in \{1, 2\} \quad (4.4)$$

$$b \in \mathbb{N} \quad (4.5)$$

$$y_1 + y_4 \geq 1 \quad (4.6)$$

The formula and subjected restrictions are to be understood as follows:

(4.1) Calculate total size, considering all aspects of the represented instantiation

(4.2) Represents the size of corresponding cryptographic object in bytes. Needs to be a positiv whole number

$pk_{client1}$ - Client Public Key

$pk_{client2}$ - Second Client Public Key - for hybrid key-exchange

pk_{server} - Server Public Key

ct - Server Ciphertext

σ - Signature

σ_{ocsp} - OCSP-response signature

σ_{ct} - Certificate Transparency signature

(4.3) Represents if component is selected or not.

(4.4) Factor for Client public key. If Encrypted Client Hello is enabled public key size is doubled. Impacts both public keys if hybrid key-exchange is enabled

(4.5) Factor for Certificate Transparency, represents log length.

(4.6) At least key-exchange or signature need to be included

4.4.2 UI

This formula is embedded in the shown in fig 4.1 browser based user interface, enabling users to configure different instatations within the subjected restrictions proposed earlier. The UI shifts selection options based on made inputs, so instantiations outside of the given constraints can not be created.

The image shows a web-based user interface for configuring a handshake. It is divided into two main sections: 'Handshake Setup' on the left and 'Configurations' on the right. The 'Handshake Setup' section contains three main parts: 'Key Exchange' (with a dropdown menu), 'Parameterset' (with a dropdown menu and a 'Use hybrid' checkbox), and 'Signature Scheme' (with a dropdown menu and a 'Parameterset' dropdown). Below these are three checkboxes: 'OCSP stapling ?' (checked), 'Certificate Transparency ?' (checked), and 'Encrypted Client Hello ?' (checked). At the bottom of this section is a blue button labeled 'Add to Comparison'. The 'Configurations' section displays a list of configurations. The first configuration is selected and shows details: 'ML-KEM / ML-DSA' with a size of '6195 B'. Below this, it lists 'Key Exchange' (Scheme: ML-KEM, Parameterset: 512, NIST Level: 1), 'Signature Scheme' (Scheme: ML-DSA, Parameterset: ML-DSA-87, NIST Level: 5), 'Client Hello' (Public Key Size: 800 bytes, Client Hello Size: 800 bytes), and 'Server Hello' (Ciphertext: 768 bytes, Signature: 4627 bytes, Server Hello Size: 5395 bytes). At the bottom of the configuration list, it shows a total size of '6195 bytes Handshake'. Red numbered circles (1-10) are overlaid on the image to indicate specific UI elements and their sequence.

Figure 4.1: UI of the calculator

Creating Instantiations

1. Select how the key-exchange is handled. All available schemes are within the upper dropdown. Once a scheme is selected, the desired parameters can be selected from the second dropdown. If hybrid key exchange is enabled, there will be two sets of dropdowns instead. The upper set defines the used KEM, the lower set defines the used PKI.
2. Select the signature scheme. The upper dropdown again includes a list of available schemes, the lower one the possible parameters.

3. Additional extensions can be toggled on and off. If Certificate Transparency is enabled, the length of its backlog can be set
4. By clicking on "Add to Comparison", the created instantiations will be added to the list of comparable instantiations.

Viewing Configurations

Each created instantiations will be added to the list on the right hand side of the UI. It is identified by the schemes used for key-exchange and creating the signature. The total size of transmitted crypto objects will also be visible[5.]. Instantiations can be expanded by clicking on them, revealing details of the selected components. These include:

6. The selected key-exchange scheme(s) name, which is a link to some external side with additional information about each scheme, the selected parameters and the corresponding NIST-Level
7. The selected signature scheme's name, which is a link to some external website with additional information about each scheme, the selected parameters and the corresponding NIST-Level
8. Client Hello with the size and type of each transmitted crypto object as well as the total size of all crypto objects within the client hello message.
9. Server Hello with the size and type of each transmitted crypto object as well as the total size of all crypto objects within the server hello message.
10. The combined size of all transmitted crypto objects during the handshake

The **Server Certificate** and **Server Certificate Verify** messages are displayed within the **Server Hello** to offer a clearer visualisation.

5 Results

5.1 Benchmarking

Table of max and min size for key-exchange and signature, as well as total handshake size and possible nist levels (1,3 and 5 as 2 and 4 are not within my source data) - script is already done, results just need to be written down here The following tables show the schemes with the largest and smallest resulting cryptographic objects. The size for key-exchange schemes represents either the sum of the public key and ciphertext or twice the public key, as the sum represent the transmitted size within the handshake.

nist level	ke scheme	params	size	sig scheme	params	size
pre-quantum	ECDHE, RSA	(ffdhe)4096	1024	RSA	2048	256
1	FrodoKEM	Frodo-1244	22876	CROSS	R-SDP 1 fast	18,432
3	FrodoKEM	Frodo-976	16720	CROSS	R-SDP 3 fast	41,406
5	FrodoKEM	Frodo-640	10360	CROSS	R-SDP 5 fast	74,590

Table 5.1: Schemes with largest cryptographic objects per level

nist level	ke scheme	params	size	sig scheme	params	size
pre-quantum	ECDHE	X25519	64	EdDSA	Ed25519	64
1	KYBER	512(-90s)	1568	UOV	ls-pkc	96
3	KYBER	768(-90s)	2272	SNOVA	(56 25 2)	168
5	KYBER	1024(-90s)	3136	UOV	V-pkc	260

Table 5.2: Schemes with smallest cryptographic objects per level

As for the smallest signatures, all represented schemes have some kind of known vulnerability depending on the used parameters and their status is still on-ramp, meaning they are not yet standartized.

5.2 Implications for Traffic

With the drastic increases in key- and signature size some impact on network traffic is to be expected. On the one hand, TLS records have a size limit of 2^{14} or 16384 bytes as described in [?]. Including padding and overhead TLSCiphertext can be slightly larger with $2^{14} + 256$ bytes. IP/TCP traffic on the other hand has a regular maximum transmission unit (MTU) size of around 1500 bytes. TCP technically supports up to 65535 bytes [?], but this value is more theoretical than practical. The largest signature size within our source data even exceeds this theoretical limit with 74,509 bytes. Considering that we only inspect cryptographic objects the transmitted records will be even larger. Under these circumstances fragmentation is to be expected, at least to a certain degree. This especially impacts the performance of the handshake, but also adds possible attack surface.[not done yet]

5.3 Use Cases

The calculator offers fast comparison of different instantiations with insights on security and size of created crypto-objects, as well as offering easily accessible deeper insights for used schemes and extensions by linking their respective paper or publication. By toggling extensions, even those mandatory in real-life scenarios, on and off their impact on the handshake size can be easily visualized. Nonetheless this calculator does not (yet) replace testing within fully configured server setups as it only represents a small scope of the whole TLS 1.3 handshake, but gives a quick comparison of viable configurations

6 Conclusion

6.1 Relevance

Given the current uncertainty when the firsts cryptographically relevant quantum computer will be fully developed, it is necessary to prepare the backbones of our means of private communication such as TLS for the impending risks. Thus it is important to be able to compare different instantiations, quantum-proof and those at risk, of the TLS handshake, to develop efficient approaches which will withstand the threats posed by future quantum computers.

6.2 Future Work

Even with the proposed calculator already functional, there still is potential for improvements and additional components as:

- Quality of life upgrades such as exporting and importing the created list of instantiations as well as deleting or editing already created instantiations
- Adding a second measuring unit for used schemes, their runtime, measuring elapsed CPU cycles for different CPU architectures
- Giving users the ability to add custom schemes for key-exchange and signatures
- Adding KEMTLS as supported handshake, where authenticity is granted by using a KEM instead of a signature
- Giving the possibility to inspect the full size of send records, including headers, metadata, etc.
- In addition to the expected size and CPU cycles, a backend could run a real time benchmark of a created instantiation and deliver test results and complete generated records for further inspection
- Generating a visual comparison as charts or comparable depictions
- Visualizing how the messages are fragmented, if record size is exceeded

Especially with these features implemented the calculator can be used to create instantiations offering usefull comparisms for a wide range of use cases, including optimizing bandwith or computing power in relation to the given level of security. With backend simulation implemented, at least the inital need for test environments would be obsolete as well.