Hybrid Post-Quantum Enhanced TLS 1.3 on Embedded Devices

Dominik Marchsreiter, Johanna Sepúlveda
Institute for Security in Information Technology, Technical University of Munich, Germany
Airbus Defense and Space GmbH, Germany
johanna.sepulveda@airbus.de

***Abstract*—Most of todays Internet connections are protected through the Transport Layer Security (TLS) protocol. Its client-server handshake mechanism provides authentication, privacy and data integrity between communicating applications. It is also the security base for the 5G connectivity. While currently considered secure, the dawn of quantum computing represents a threat for TLS. In order to prepare for such an event, TLS must integrate quantum-secure (post-quantum) cryptography (PQC). The use of hybrid approaches, that combines PQC and traditional cryptography are recommended by security agencies. Efficient PQC integration at TLS requires the exploration of a wide set of design parameters and platforms. To this end this work presents the following contributions. First, wide evaluation of PQC-enhanced TLS hybrid protocols, using end-to-end communication latency as metric. Second, the exploration and benchmarking in constrained embedded devices. Third, a wide traffic analysis, including the impact and behavior of PQC-enhanced hybrid TLS in real practical scenarios.**

I. Introduction

Public-key cryptography (PKC) provides the basis for establishing secured communication channels between multiple parties. It provides confidentiality, authenticity and non-repudiation of electronic communications and data storage. Web, Internet-of-Things (IoT) and Cloud computing, which enable many critical applications, heavily rely on secure protocols that use PKC. Many of these applications exchange data that require long term security, that is, data should remain secure for a long period of time.

The Transport Layer Security (TLS) 1.3 protocol was specified in the RFC 8446 by the Internet Engineering Task Force (IETF) [1]. TLS 1.3, which secures most Internet connections, uses symmetric and PKC to perform authenticated key exchange to encrypt data at transport. Advanced Encryption Standard (AES) is the current symmetric primitive for encrypting data at transport. PKC is used for key exchange and signatures. The key exchange was initially based on Rivest–Shamir–Adleman (RSA), and later approaches use Diffie-Hellman (DH) and Elliptic-curve Diffie-Hellman (ECDH). The signatures, used for authentication, are based on RSA and Elliptic Curve Digital Signature Algorithm (ECDSA). While TLS is secure against today's classical computers, it is vulnerable to future attacks from quantum computers.

The security of RSA and ECC relies on the hardness of factoring large integers or computing discrete logarithms, respectively. It has been shown that cryptography primitives will be affected by future quantum computers. By executing Shor's [2] and Grover's [3] quantum algorithms, these computers will be able to solve the problems on which classical PKC (RSA, ECC) relies in polynomial time and to decrease the security level of symmetric cryptosystems. While symmetric cryptosystems can be easily secured against quantum computers by choosing larger key sizes, securing PKC requires new hard mathematical problems.

Post-Quantum Cryptography (PQC) refers to a set of algorithms that rely on mathematical problems that are considered very hard to solve with current traditional computers and with future quantum computers. These algorithms can be deployed with traditional electronics technology. In order to continue using TLS 1.3 in the wider variety of applications, the secure TLS 1.3 protocol should be enhanced with PQC primitives. In order to provide a secure transition towards PQC-enhanced TLS 1.3 protocols different security agencies have recommended to use hybrid approaches that combines traditional and PQC primitives[4]. Such an approach ensures that even if the new PQC primitives are compromised, the security will not be worse than the security provided by the current traditional cryptography.

The National Institute of Standards and Technology (NIST) is currently running a process for post-quantum cryptography (PQC) standardization. From the initial 69 post-quantum algorithms submitted by cryptographers worldwide in 2016, currently, only 7 algorithms remain as *finalist* candidates. A set of these algorithms will be standardized until 2022 [5]. Each algorithm presents different trade-offs in terms of performance and key and signature/ciphertext sizes.

Hybrid PQC-enhanced TLS protocols has been proposed in the past. However, a wide exploration of different alternatives for the integration of PQC in TLS has not yet been performed. Furthermore, the impact on such protocols on constrained devices (including the end-to-end latency) has not been performed. In this work, we present a study on the evaluation of different alternatives for hybrid PQC-enhanced TLS protocol in constrained devices. This paper presents the following contributions:

- Evaluation of the performance of different alternatives of PQC-only and hybrid PQC-enhanced TLS 1.3 using end-to-end communication as metric;
- Handshake performance and communication evaluation of hybrid PQC-enhanced TLS 1.3 in constrained devices;
- Implementation of an application protected by an hybrid PQC-enhanced TLS 1.3 and wide traffic analysis.

Results show that while PQC can perform security functionalities even faster than traditional cryptography, it might have an influence in the end-to-end communication. This can be critical for constrained applications with limited network speed. Tight time requirements might not be reached. The selection and construction of hybrid PQC-enhance TLS 1.3 is critical.

II. Related Work

The Transport Layer Security (TLS) is the successor of the now-deprecated Secure Sockets Layer (SSL) and a proposed

2022 25th Euromicro Conference on Digital System Design (DSD) | 978-1-6654-7404-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/DSD57027.2022.00127

2771-2508/22/$31.00 ©2022 IEEE
DOI 10.1109/DSD57027.2022.00127

Internet Engineering Task Force (IETF) standard. TLS 1.0 [6] was defined in January 1999 by T.Dirks and C.Allen as an upgrade to SSL Version 3.0. The protocol was developed to provide security for computer network communication. The current version TLS 1.3 was introduced in August 2018 as successor to TLS 1.2 [7] and is specified in IETF RFC 8446 [1]. The approved protocol makes it the new industry standard for secure communication. TLS is an essential part of securing internet connections via HTTPS. The HyperText Transfer Protocol Secure (HTTPS) is the secure extension of HTTP that encrypts the communication protocol with TLS. It is also known as HTTP over TLS [8]. Moreover, it has applications in instant messaging with end-to-end encryption (E2EE), voice over internet protocol (VoIP) and email. However, the cryptographic primitives used to secure TLS are vulnerable to quantum attacks and post-quantum primitives should be used in the TLS protocol to ensure the security of the protocol even in the future. Cryptographic algorithms based on different hard problems that are post-quantum secure are explored to find a standard by the National Institute of Standards and Technology (NIST)[9]. The research on post-quantum cryptography is very topic but also gains a lot of interest from leading companies. Related work describes the implementation of those algorithms in TLS in different scenarios. The combined elliptic-curve and post-quantum 1 (CECPQ1) project, performed by a research team by Google, implemented the post-quantum key encapsulation mechanism (KEM) algorithm NewHope in a key agreement protocol [10]. The limited experiment used NewHope in TLS 1.2 by implementing it in Google Chrome 54 beta. For authentication, a non-PQ algorithm ECDSA with X25519 curve was used. The results showed that there was no unexpected impediment to deploying post-quantum algorithms such as NewHope. NewHope is not part of the NIST finalists anymore, but it addressed the additional latency for people on slow network connections due to the expensive data requirement of the algorithm [11]. The CECPQ2 project succeeded CECPQ1 in cooperation with Cloudflare [12]. In this project, the Google Chrome browser was modified to test hybrid key exchange with two PQ KEM algorithms, NTRU-HRSS and SIKE. The hybrid key exchange with X25519 serves as a fallback to ensure the security of the connection with classical cryptography. Again, non-PQ authentication with ECDSA and X25519 curve was used. The results showed that SIKE, with large computational cost but smaller key sizes, can not outperform NTRU-HRSS. The company Cisco evaluated with a research team the use of six PQ signature schemes and did a performance study on the authentication in TLS 1.3 [13]. For key exchange, classical elliptic-curve Diffie-Hellman (ECDH) with secp384r1 curve was used. The implementation of the post-quantum algorithms in TLS 1.3 was done by utilizing the Open Quantum Safe (OQS) OpenSSL library [14]. It was shown that there is a noticeable impact of signature and certificate chain sizes on the total handshake time. They recommended Dilithium and Falcon for time-sensitive applications and expect signature sizes and key sizes to have the most impact on the handshake. The research team by Microsoft tested combinations of classical and post-quantum primitives and emulated packet loss and link latency to their tests [15]. The first combination was classical authentication with ECDSA (NIST P-256 curve) and hybrid KEM with PQ algorithms SIKE, Kyber and FrodoKEM. The opposite combination used PQ authentication with Dilithium, qTesla and Picnic and classical key exchange with ECDH. The research concludes that on fast reliable networks the performance is determined by the cost of public-key cryptography. However, on unreliable networks the communication data size takes a dominant role. The maximum transmission unit (MTU) size imposed by the link layer influences the establishment performance. Last, Amazon has researched post-quantum implementation in embedded systems and used complete PQC for the TLS connections [16]. For authentication, SPHINCS$^+$ was used and Kyber as KEM. The focus was on the PQ handshake feasibility on embedded systems. The experiment showed that the biggest obstacle was the computation of signatures with SPHINXES$^+$. Most of the time was consumed by the PQC schemes but the performance of PQC-TLS for the client was similar to elliptic curve cryptography (ECC). For the use of an embedded system server, additional hardware accelerators should be considered.

While some of the related work has looked at the implementation of PQC in TLS 1.3 a complete analysis of all possible combinations of the NIST round 3 finalists have not been fully investigated yet. There are no results on different combinations of PQ-KEM and PQ signature algorithms. Although the NIST standardization process is not finished yet it is important to analyze the behavior and impact of the algorithms in practical applications now. There are 7 possible algorithms (and 8 alternatives) mentioned as NIST finalists of round 3. The OQS project features almost every NIST PQC finalist in their framework and they analyze the performance of the algorithms on two different platforms, Intel(R) Xeon(R) Platinum 8259CL and ARM Cortex-A53 [14]. However, there has no prior analysis been on the end-to-end latency. All the mentioned research only utilized an extract of PQC algorithms or used classical KEMs or classical authentication. Moreover, there has been no implementation on a restricted embedded device, namely Raspberry Pi 4, as a server for the final algorithms. Practical application scenarios to evaluate the performance and feasibility are only evaluated by Google with outdated algorithms. Last, there are no results on hybrid cryptography with PQC in TLS 1.3 reported.

III. Traditional TLS

A. Description

The Transport Layer Security (TLS) protocol is used to establish an encrypted connection between a client and a server. The protocol ensures an encrypted transmission of application data over a network. TLS is running on top of the transport layer, e.g., over TCP, but is located under the application layer and does not fit in any single layer of the OSI model [17]. The protocol is the successor of the SSL protocol and the latest version is TLS 1.3, which was defined in August 2018 [1] and has performance benefits over the currently widely used TLS 1.2 protocol. The new protocol reduces the round trip time of 2 (2-RTT) to only 1 (1-RTT) during connection establishment, resulting in a faster handshake. If connections have been prior established a zero round trip time (0-RTT) can be achieved by memorizing. Unsafe and outdated algorithms with cryptographic weaknesses that had security vulnerabilities are removed. The security of TLS 1.3 is based on a hybrid cryptosystem with public-key cryptography and symmetric stream ciphers. The key exchange got simplified by removing RSA [18] because of the lack of an ephemeral key mode. This is necessary for perfect forward secrecy, an

essential requirement of TLS 1.3. Therefore, the protocol uses ephemeral Diffie-Hellman [19]. For authentication, the signature algorithms are RSA, Elliptic Curve Digital Signature Algorithm (ECDSA), Edwards-Curve Digital Signature Algorithm (EdDSA) and the option for a pre-shared key. The session key for symmetric cryptography uses the Advanced Encryption Standard (AES) [20] with key lengths of 128-bit or 256-bit.

B. TLS handshake

The client and server agree on the various parameter in a handshake procedure. Figure 1 presents the complete TLS 1.3 handshake between a client initiating the connection to a remote server.

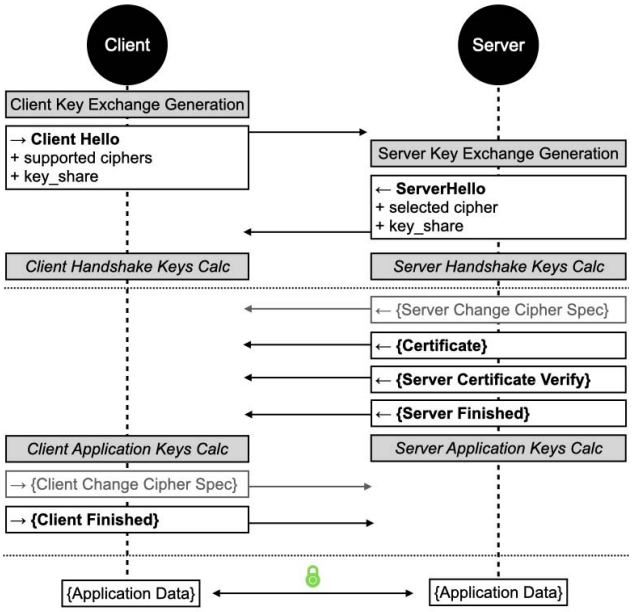

Fig. 1: Overview of the TLS 1.3 handshake. {} indicates messages are protected.

Before the client initiates the connection the public/private key pair for key exchange are generated. The handshake begins with the `Client Hello` message. The extensions contain a list of cipher suites and the calculated public keys. Afterwards, the server generates a public/private key pair and answers with the `Server Hello` message. To encrypt the following messages both parties calculate the handshake keys. The protocol uses outdated and leftover `Change Cipher Spec` messages from TLS 1.2. The messages are sent to help disguising the session as a TLS 1.2 session. The `Certificate` and `Certificate Verify`, which is the signature, is sent over to the client. The server closes with the `Server Finished` message. Now client and server calculate the application keys. After the client sends its `Client Finished` message, too, application data can be exchanged encrypted. The session key is used to encrypt and decrypt the application data until the connection closes.

C. Quantum Threat

In a post-quantum scenario, classical public-key cryptography (PKC) is not considered secure anymore. Shor's quantum algorithm [2] can solve in polynomial time the hard mathematical problems of current traditional PKC, such as RSA and ECDH. These algorithms are used in TLS for key exchange and authentication. Symmetric ciphers can be attacked with Grover's quantum algorithm [21]. In contrast to classical solutions, the Grover algorithm executed on a quantum computer can search in an unsorted database with quadratic speedup. This leads to brute force attacks on a 2^N-bit symmetric key in approximately $2^{N/2}$ iterations which means the security of the key size will reduce by half.

To prepare TLS for quantum attacks it is necessary to substitute the traditional cryptographic primitives with post-quantum secure cryptography.

IV. Post-Quantum Cryptography

A. Description

Post-quantum cryptography (PQC) refers to cryptographic algorithms that claim to be secure against attacks by quantum computers. They rely on different hard mathematical problems than the current cryptography. Traditional cryptography is based on integer factorization problem, discrete logarithm problem or elliptic-curve discrete logarithm problem, which are at risk if the performance of quantum computers increases in the future. PQC can be classified into five different families: i) *Codes*, based on the hardness of decoding a general linear code (e.g., Classic McEliece); ii) *Hash-trees*, based on the hardness of the security of hash functions (e.g., SPHINCS+); iii) *Multivariate*, based on the hardness of solving systems of multivariate polynomials over finite fields (e.g., Rainbow); iv) *Isogeny*, based on the hardness of finding the isogeny mapping between two supersingular elliptic curves with the same number of points (e.g., SIKE); and v) *Lattices*, based on hard problems described in lattices such as the Shortest Vector Problem (SVP), Closest Vector Problem (CVP), and Learning with Errors (LWE) problem together with its ring variant (Ring-LWE) over ideal lattices (e.g., Kyber, NTRU, Saber, Dilithium, Falcon).

These algorithms can be executed on traditional devices, but provide security against attacks performed with classical and quantum computers.

B. NIST Finalists

NIST, the National Institute of Standardization, announced at PQCrypto [22] in 2016 a competition to find alternative algorithms that are post-quantum secure [9]. The standardization process will go through a selection process of the submitted algorithms in several rounds until decide for a set of algorithms to be included in the PQC standard. The submitted algorithms are based on the five different types of PQC families. At each round, the security is widely and publicly analyzed. The most promising ones proceed to the next round. There are PQC algorithms for digital signature and key exchange. From the 82 PQC candidates submitted in the first round, only 7 algorithms remains as finalists in the third round. The finalist for key encapsulation mechanisms (KEM) include a code-based PQC (Classic McEliece) and three lattice-based PQC (Kyber, Saber, NTRU). The finalist for signature algorithms (SIG) include a multivariate-based (Rainbow) and two lattice-based (Dilithium and Falcon). However, recent attacks showed that intersection and rectangular MinRank attacks on Rainbow weaken the security level, which Beullens announced in [23]. Since this algorithm does not reach the requirements of the

competition anymore it was not considered in this work. In addition, Classic McEllice has huge key sizes and is not competitive for TLS integration. so we were only focusing on the lattice-based algorithms.

NIST has defined five different levels of security (I to V). Each PQC algorithm can implement some of these security levels by carefully selecting the different parameters.

V. PQC Enhanced TLS

A. Description

To enhance TLS with PQC the post-quantum algorithms should be included for the authentication and key exchange functionalities. TLS 1.3 uses three cryptographic operations: key exchange methods, signing/verification operations and symmetric-key algorithms. Symmetric cryptography can be still patched by doubling the key size. However, PKC should be enhanced with PQC. From the 7 NIST PQC finalists (4 KEM and 3 SIG), lattices have been proven attractive due to the acceptable trade-off between performance and key sizes. Two lattice-based SIG (Dilithium and Falcon) and three KEMs (Kyber, NTRU and Saber) are available.

There are two ways to integrate PQC into TLS: PQC-only enhanced and hybrid PQC-enhanced.

B. PQC-only enhanced

The PQC-only enhanced TLS version uses PQC primitives to fully replace the traditional PKC. The traditional cryptographic operations are substituted by PQC. While the KEMs are employed on key generations, encapsulations and encapsulations operations, the SIGs are used on signing and verification operations.

The certificate generation, which is carried out by the server, uses a PQC SIG (signature algorithm) to sign the public key. The signature is inserted in the X.509 certificate. The public keys that are used during the handshake for the key establishment process are calculated by key encapsulation mechanisms (KEM). The client can authenticate the server by using the same post-quantum digital signature algorithm to perform verification operations on the received documents of the client (signature and public key). The parameter set of the different algorithms is selected so as that the same security levels for KEM and SIG are always used.

To get an overview of the impact of the PQC integration, in the experimental section the performance on the server and the client were measured for the baseline implementation (traditional TLS 1.3) and the PQC-only enhanced. The number of operations during 10 seconds is measured and transformed into operations per second. Table I presents the results for KEM on different parameter sets with information on public-key, secret-key and ciphertext size. Table II the results for SIG algorithms. This test was executed on the Raspberry Pi and on the local machine.

The results in I show that KEMs are as good or faster than traditional ECDH. Just NTRU algorithms are very slow in key generation. Moreover, with increasing security levels, we reduce the speed of the algorithms. For SIG algorithms we see in table II that ECDSA is faster in signing than RSA. Signing operations with RSA are costly and create more load on the server's performance than the verifying operation on the client-side. To perform authentication operations on small embedded devices, the verification operation should be small in computational afford since the client only needs to do verification operations during the TLS handshake. Therefore, Falcon seems to be more suited for clients who are restricted on performance. However, Falcon is slower in signing compared to Dilithium. RSA behaves like Falcon in traditional cryptography with fast verification and slow signing.

The numbers show that signing operations take much longer than KEM and all algorithms are well suited to be implemented in TLS.

C. PQC enhanced with hybrid cryptography

The hybrid PQC-enhance is a layered approach that combines traditional PKC with PQC. This solution is the most recommended alternative for early adopters of PQC since even when a security breach is identified in the new PQC primitive, the protocol's security remains as secure as nowadays. A full PQC-only enhanced protocol is envisioned as a future solution available when enough confidence is built on the new PQC. In hybrid cryptography, two algorithms simultaneously work side

TABLE I: NIST KEM finalists Round 3 : sizes and performance. All the sizes are reported in bytes. The execution times of cryptographic operations are reported in op/s.

Parameter Set	Public Key	Secret Key	Ciphertext	Security level	**Intel Core i7-9750H CPU**			**ARM Cortex-A72**		
					KeyGen	Encaps	Decaps	KeyGen	Encaps	Decaps
ECDH secp256r1	65	32	65	I	21750,8			1236,3		
ECDH secp384r1	96	48	96	III	297.036			253,2		
Kyber512	800	1632	768	I	113838,5	94006,4	132621,5	8044,4	7442,6	7213,5
Kyber90s512	800	1632	768	I	157448,9	130173,2	209774,2	5037,2	4721,6	4600,1
NTRU HPS2048509	699	935	699	I	23785,8	136335,4	101422,3	284,4	4713,4	5307,5
Lightsaber	672	1568	736	I	75489,3	68601,4	72791,9	7661,3	7811,7	7805,8
Kyber768	1184	2400	1088	III	74833,9	64645,4	85081	5439,0	4931,9	4661,5
Kyber90s768	1184	2400	1088	III	116402,4	97318,5	149554,6	2996,4	2812,2	2714,8
NTRU HPS2048677	930	1234	930	III	14089,7	92916,7	60681,6	172,5	3594,9	4234,1
NTRU HRSS701	1138	1450	1138	III	13821,2	118283,8	60681,6	167,6	6314,5	3965,8
Saber	992	2304	1088	III	46963,6	44275,4	45247,4	4731,1	4516,0	4302,0
Kyber1024	1568	3168	1568	V	53735,1	41907,7	54888,6	3755,9	3401,8	3185,8
Kyber90s1024	1568	3168	1568	V	86514,6	64326,9	92282,5	1925,1	1819,4	1754,2
NTRU HPS4096821	1230	1590	1230	V	10179,7	78262,6	47949,8	121,3	2838,7	2987,7
Firesaber	1312	3040	1372	V	32614,8	30157,2	28765,6	3157,1	2921,4	2739,9

TABLE II: NIST SIG finalists Round 3: sizes and performance. All the sizes are reported in bytes. The execution times of cryptographic operations are reported in op/sec.

Parameter Set	Public Key	Secret Key	Signature	i7-9750H CPU		ARM Cortex-A72	
				Sign	Verify	Sign	Verify
ECDSA p384	48	48	48	0.481	0.979	240,4	304,7
RSA-3072	387	387	387	27.902	8.247	54,0	2868,3
Dilithium2	1312	2528	2420	315.197	4.424	881,8	3684,1
Falcon512	897	1281	690	380.198	6.724	76,7	6104,0
Dilithium3	1952	4000	3293	378.993	161.763	550,2	2313,8
Dilithium5	2592	4864	4595	378.993	161.763	462,7	1419,6
Falcon1024	1793	2305	1330	378.993	161.763	35,2	3001,3

by side and the public keys K_{pk} are joined while transmitted between a client and a server($K_{pk}\{K_{pk1},K_{pk2}\}$). Both types of algorithm will be executed for each cryptography operation in the TLS 1.3 protocol and the keys will be processed separately. That leads to additional computational costs and an increase in the transmitted data. The TLS protocol structure remains the same as the PQC-only TLS version; only the cryptographic blocks hold both algorithms. The layered strategy (using PQC together with ECDSA, RSA and ECDH), can be a good strategy to achieve a possible fallback to classical secure algorithms. Table III presents the results of the speed of hybrid digital signature algorithms.

TABLE III: NIST Hybrid SIG finalists Round 3: sizes and performance. All the sizes are reported in bytes. The execution times of cryptographic operations are reported in op/sec.

Parameter Set 1	Parameter Set 2	ARM Cortex-A72	
		Sign	Verify
ECDSA p256	Dilithium2	789,3	1583,9
RSA-3072	Dilithium2	51,0	1583,0
ECDSA p256	Falcon512	75,8	1921,8
RSA-3072	Falcon512	31,8	1922
ECDSA p384	Dilithium3	167,5	264,7
ECDSA p384	Dilithium5	76,2	110,8
ECDSA p384	Dalcon1024	25,4	116,0

Results show that RSA-3072 reduces the performance of signing operations a lot. Combinations with ECDSA are faster; However, algorithm speeds are in general slower than PQ-only algorithms. The performance behavior of hybrid cryptography can be investigated in the TLS handshake.

VI. Experimental Work

This section evaluates the performance of the TLS handshake with PQC and measures the data sizes during the initial communication. The data include the certificate and public key sizes. The results are benchmarked in comparison to classical and hybrid cryptography. The feasibility of PQC-TLS is tested with higher-level application integration like the Google Chrome browser and *nginx* server. For the benchmarking process, the focus is on the evaluation of the different algorithms and not on the optimization of performance or reduction of data sizes. It is assumed that the criteria of security of the finalists are correct; there is no verification of the security of the NIST finalist in this work.

A. Experimental setup

For the implementation of PQC-TLS, the framework of the Open Quantum Safe (OQS) project was used, which is open-source available. The repository on GitHub [14] contains the library liboqs and prototype integrations of liboqs in some protocols and applications. Liboqs is an open-source C library for quantum-resistant cryptographic algorithms. It integrates code from PQClean [24], which is the implementation of a variety of PQ algorithms and contains all NIST finalist algorithms. The prototype implementations involve the widely used OpenSSL library. This library is needed to perform PQ key exchange and authentication with the TLS protocol. In addition, the project library provides prototype applications such as Apache httpd, nginx, curl or OpenVPN. For this work, we only focused on the relevant implementations for PQC-TLS 1.3. All practical tests were obtained by using two platforms: A Raspberry Pi 4 Model B with Broadcom BCM2711 quad-core Cortex-A72 processor running at 1.5GHz and 64-bit architecture which was used to represent the server, and a computer with Skylake processor Intel Core i7 6.Gen i7-9750H running on 2,6 GHz with 6-cores configured as a client. Both devices were connected to a local LAN network with a bandwidth of 1Gb/s and Linux Ubuntu 20.04 was used as the operating system. With these two platforms five different setup combinations are created to test the algorithms and run different applications with it from the OQS software library. An overview of all experimental setup combinations is given in IV. First, the OpenSSL speed command is executed on both devices to measure the performance of the algorithms. Setup 3 is used to establish TLS handshakes and measure the end-to-end latency and setup 4 to discover the data sizes during the handshake with the application Wireshark. Last, the feasibility of a classical web setup is tested by using OQS-Chromium and OQS-Nginx. OQS-OpenSSL_1_1_1 snapshot release 2021-08 [14] was used for the implementation of the TLS protocol. It is a fork of OpenSSL 1.1.1 that adds quantum-safe key exchange and authentication algorithms using liboqs 0.7.0 for prototyping and evaluation purposes.

B. Performance analysis

To evaluate the performance of PQ-TLS the time a handshake takes until application data can be sent gives a good metric for benchmarking the performance. The handshake performance is decisive for how well the algorithms perform in the TLS protocol. Since there are a variety of different algorithms to choose from it is interesting to know which combination of SIG and KEM outperform others. Relevant is

TABLE IV: Experimental setup combinations used for the experimental work.

Setup	Hardware	Software	Application	Results
1	Local Machine	OQS-OpenSSL	openssl speed	KeyGen, encaps, decaps, sign, verify
2	Raspberry Pi 4	OQS-OpenSSL	openssl speed	KeyGen, encaps, decaps, sign, verify
3	Local Machine + Raspberry Pi 4	OQS-Curl	openssl s_time	Connections/sec
4	Local Machine + Raspberry Pi 4	OQS-Curl + Wirequard	openssl s_client	Data sizes
5	Local Machine + Raspberry Pi 4	OQS-Chromium + OQS-Ngnix	HTTPS request	Feasibility

the time until the TLS handshake is finished and application data can be transferred between the client and the server. For each combination of PQ algorithms, we established TLS connections and measured the number of handshakes during 30 seconds. The results were transformed into connections per second. The figure shows the results of the measurement sorted by the security levels. In comparison, classical algorithms are used and tested to give a reference to our current TLS handshake speed.

TABLE V: NIST finalists Round 3: Handshake performance.

SIG Parameter Set	KEM Parameter Set	connections/sec
ECDSA p256	ECDH	376,54
ECDSA p384	ECDH	109,10
RSA 3072	ECDH	42,72
Dilithium2	Kyber 512	235,61
Dilithium2	Kyber90s512	227,55
Dilithium2	NTRU-HPS-2048509	220,03
Dilithium2	Lightsaber	230,32
Falcon512	Kyber512	58,48
Falcon512	Kyber90s512	57,00
Falcon512	NTRU-HPS-2048509	57,84
Falcon512	Lightsaber	59,23
Dilithium3	Kyber768	188,52
Dilithium3	Kyber90s768	181,74
Dilithium3	NTRU-HPS-2048677	179,03
Dilithium3	NTRU-HRSS-701	181,74
Dilithium3	Saber	186,10
Dilithium5	Kyber1024	165,71
Dilithium5	Kyber90s1024	164,26
Dilithium5	NTRU-HPS-4096821	156,13
Dilithium5	Firesaber	160,03
Falcon1024	Kyber1024	29,84
Falcon1024	Kyber90s1024	29,58
Falcon1024	NTRU-HPS-4096821	29,35
Falcon1024	Firesaber	29,48

The result provides different observations: i) TLS handshakes with PQC are comparable fast to classical cryptography; ii) key encapsulation mechanisms do not influence the speed a lot compared to the signature algorithms; and iii) combinations with Dilithium are faster than with Falcon. The end-to-end (E2E) handshake time is the total of the messages transfer time and the time needed by the execution of the algorithms. The large share of transmission transfer time is between the `Server Hello` message and the `Client Change Cipher Spec` message. This time is also often referred to as `Certificate Transfer Time`. It includes the certificate transfer from the server to the client, the signing operation of the server and the verification operation of the client. Large certificates and slow signature operations increase the transfer time. The network's latency can also influence the time for completing the handshake, but in this setup, the network is fast enough and the ping between server and client is about 1ms. This means that in fast networks with devices with sufficient power, the transmission speed is not quite limiting the handshake time. Real-time applications, like VoIP, instant messaging or video conference applications have in these conditions no significant disadvantage over current cryptography.

C. Traffic analysis

Besides testing the time it takes to complete a full handshake and establishing a TLS connection, the data sizes are measured during transmission. The application Wireshark[1] was used to read out the file sizes after specific messages. First, all transmitted bytes B_h during the handshake until the application data is sent are measured. Second, the certificate size B_c, which includes the SIG public key K_{pk} of the signing operation and the signature S, is determined. Last, the KEM public key K_{pk} of the used KEM was measured.

The results show i) current certificate sizes varies between 0.7 and 1.5KB. PQC increases the size to 2.4 - 10KB; ii) the transmitted data is a lot more compared to our classical cryptography. The amount of data can be up to 10x higher; iii) the data size is most affected by the signature algorithm used; and iv) KEM key sizes are also bigger than ECDSA and RSA public keys but much smaller than those for authentication used.

D. Comparison to Hybrid Algorithms

SIG and KEM operations can also be executed with hybrid algorithm combinations. We tested hybrid combinations for SIG and KEM operations by implementing additional ECDSA, RSA and ECDH algorithms. Table VII present the results (speed of execution and transmitted bytes B_h) for all different hybrid algorithm combinations.

The result compares hybrid cryptography to PQ-only and classical cryptography. If using hybrid cryptography in TLS we get the following results: i) PQ SIG algorithms in combinations with ECDSA p256 are faster than with RSA 3072, ii) with high-security levels the calculation is more affected by the hybrid combination than with low-security levels. For level V the slow down of the calculations when using hybrid mode is about 80-90%, iii) in general we slow down the execution time by using hybrid cryptography, and iv) the transmitted bytes B_h per handshake did not increase noticeably because key sizes of traditional cryptography are small.

[1] www.wireshark.org

TABLE VI: NIST finalists Round 3: Traffic analysis. All the sizes are reported in bytes.

SIG Algorithm	KEM Algorithm	B_h	B_c	KEM K_{pk}
ECDSA p256	ECDH	1557	570	32
ECDSA p384	ECDH	1747	652	32
RSA 3072	ECDH	2734	1464	32
Dilithium2	Kyber 512	9566	5458	800
Dilithium2	Kyber90s512	9566	5458	800
Dilithium2	NTRU-HPS-2048509	9396	5458	699
Dilithium2	Lightsaber	9406	5458	672
Falcon512	Kyber512	5285	2484	800
Falcon512	Kyber90s512	5280	2484	800
Falcon512	NTRU-HPS-2048509	5112	2484	699
Falcon512	Lightsaber	5069	2484	672
Dilithium3	Kyber768	12926	7505	1184
Dilithium3	Kyber90s768	12926	7505	1184
Dilithium3	NTRU-HPS-2048677	12460	7505	930
Dilithium3	NTRU-HRSS-701	12876	7505	1138
Dilithium3	Saber	12518	7505	992
Dilithium5	Kyber1024	17142	10138	1568
Dilithium5	Kyber90s1024	16224	10138	1184
Dilithium5	NTRU-HPS-4096821	16682	10138	1230
Dilithium5	Firesaber	17006	10138	1312
Falcon1024	Kyber1024	9202	4535	1568
Falcon1024	Kyber90s1024	9250	4535	1568
Falcon1024	NTRU-HPS-4096821	8521	4535	1230
Falcon1024	Firesaber	8849	4535	1312

E. Webpage Analysis

To check if PQ-TLS is currently possible in a practical environment, a typical web scenario with PQC and high-level PQ-enabled applications is created. The test was designed to transmit an HTML site to a client and display the web page on the client-side. This is the closest scenario for practical PQ TLS 1.3. An HTTPS server was established by using the modified nginx version by the OQS project and the provided Google Chromium web browser was built based on the OQS-BoringSSL fork to create the client. The browser and the server needed to be completely Post-quantum enabled.

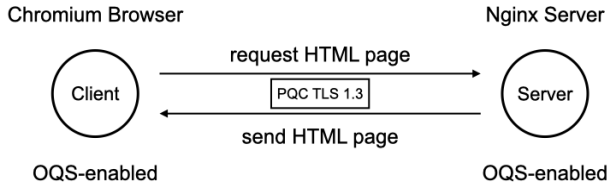

Fig. 2: Overview of a TLS webpage scheme.

The practical implementation had some limitations: i) OQS-Chromium 0.7.0 is only enabled to use hybrid cipher suits. In combination with NIST finalist round 3 algorithms the options are: p256_kyber90s512, p256_lightsaber and p256_NTRU-HPS-2048509, and ii) there is no option to load quantum-safe certificates into Chromium. This results in that PQ authentication was not possible. In addition, OQS-curl can be used to create a client and request the HTML page from the server in the Linux terminal. If using this method PQ authentication is working and the web page could be transferred without any issues. This is possible for any algorithm combination.

F. Impact and Discussion

We analyzed the performance of PQ-TLS from a different point of view and found out: i) PQ-TLS is slower in authentication, e.g. additional 1.6 - 25ms, ii) has an increase in key sizes, e.g. additional 3,7 - 15,5MB, and iii) symmetric cryptography remains classic with the double key size of the session key. Moreover, with session resumption the handshake is only performed during the first connection establishment; Therefore, the additional time and key size only minor affects in the beginning. The impact of PQC for TLS 1.3 for typical applications with sufficient computational power and bandwidth is small. Typical applications are instant messaging end-to-end encrypted (E2EE), voice over IP (VoIP), email and web applications. However, some applications have specific requirements that are critical for PQC. Two of them are web and automotive. Search engine optimization (SEO) sets the restriction on web pages. SEO also focuses on the time to first byte (TTFB), which is the time that it takes for a user´s browser to receive the first byte of web page content. Google requires that TTFB should be smaller than 200ms [25]. TTFB includes the DNS lookup and the TLS handshake time if the request is made over HTTPS. This requirement is set because users dislike when pages have a long loading time. To reach this goal, the handshake time should be $<<$200ms. Assuming client and server have a very fast time to process algorithms, then the speed is limited by the amount of data transmitted over a network. Since PQC has more data to transmit than current cryptography, the TTFB can be larger with insufficient network speeds. This can be especially with mobile networks. In this case, PQC would require at least a 3G connection with 72kbit/s to reach the $<<$200ms requirement in general. Classical cryptography only requires a 2G Edge connection. Hybrid algorithms are similar to only PQC due to the small increase in classical cryptography key sizes. If using hybrid algorithms, similar network speeds to PQ-only are required. Automotive requirements on reaction times are set to about 100ms or smaller. PQC handshake times vary between 4,2 - 34 ms. Only algorithm combinations with Dilithium can reach the 10ms requirement. This assumption is based on network bandwidth of 1Gb/s. However, 100Mb/s is more widely used in automotive networks due to cost efficiency. The slower network speed leads to longer public-key and certificate transmission times. This can be up to 137ms only for transmitting the data if using Dilithium for security level III. In these conditions, the requirements for automotive can not be achieved with data-heavy algorithms like Dilithium.

VII. Conclusion

In this work, we evaluate the suitability of the NIST PQC Round 3 finalists to the TLS 1.3 protocol. We explore different hybridization alternatives. The obtained results demonstrate that while PQC can perform security functionalities (encryption, decryption, signature, verification) even faster than traditional cryptography, it might have an influence in the end-to-end communication. While this is not an inconvenient for application characterized by sufficient bandwidth and computational power, it might become an issue for constrained applications. Time constraint applications with limited network speed may not reach high performance requirements. In addition, we show that signatures has the higher penalty on the execution time, thus causing a significant impact on the handshake time. We show that there is different trade-

TABLE VII: Hybrid cryptography: Handshake performance and traffic analysis. All the sizes are reported in bytes.

SIG Parameter Set 1	SIG Parameter Set 2	KEM Parameter Set 1	KEM Parameter Set 2	connections/sec	B_h
RSA 3072	Dilithium2	ECDH p256	Kyber 512	39,10	10913
RSA 3072	Dilithium2	ECDH p256	Kyber90s512	39,48	10913
RSA 3072	Dilithium2	ECDH p256	NTRU-HPS-2048509	39,16	10797
RSA 3072	Dilithium2	ECDH p256	Lightsaber	40,00	10699
RSA 3072	Falcon512	ECDH p256	Kyber512	27,35	6593
RSA 3072	Falcon512	ECDH p256	Kyber90s512	27,06	6591
RSA 3072	Falcon512	ECDH p256	NTRU-HPS-2048509	26,97	6527
RSA 3072	Falcon512	ECDH p256	Lightsaber	27,19	6430
ECDSA p256	Dilithium2	ECDH p256	Kyber 512	167,16	9845
ECDSA p256	Dilithium2	ECDH p256	Kyber90s512	162,29	9846
ECDSA p256	Dilithium2	ECDH p256	NTRU-HPS-2048509	176,10	9600
ECDSA p256	Dilithium2	ECDH p256	Lightsaber	167,00	9611
ECDSA p256	Falcon512	ECDH p256	Kyber512	54,16	5627
ECDSA p256	Falcon512	ECDH p256	Kyber90s512	51,94	5576
ECDSA p256	Falcon512	ECDH p256	NTRU-HPS-2048509	53,00	5408
ECDSA p256	Falcon512	ECDH p256	Lightsaber	52,55	5469
ECDSA p384	Dilithium3	ECDH p384	Kyber768	43,26	13258
ECDSA p384	Dilithium3	ECDH p384	Kyber90s768	39,13	13420
ECDSA p384	Dilithium3	ECDH p384	NTRU-HPS-2048677	39,68	12847
ECDSA p384	Dilithium3	ECDH p384	NTRU-HRSS-701	41,03	13424
ECDSA p384	Dilithium3	ECDH p384	Saber	41,58	13120
ECDSA p521	Dilithium5	ECDH p521	Kyber1024	19,90	17816
ECDSA p521	Dilithium5	ECDH p521	Kyber90s1024	20,16	17761
ECDSA p521	Dilithium5	ECDH p521	NTRU-HPS-4096821	20,03	17139
ECDSA p521	Dilithium5	ECDH p521	Firesaber	19,90	17464
ECDSA p521	Falcon1024	ECDH p521	Kyber1024	13,39	10049
ECDSA p521	Falcon1024	ECDH p521	Kyber90s1024	13,35	10053
ECDSA p521	Falcon1024	ECDH p521	NTRU-HPS-4096821	13,03	9210
ECDSA p521	Falcon1024	ECDH p521	Firesaber	13,52	9537

offs between the selection of different PQC alternatives and hybridization ways. While using Dilithium a faster cryptographic execution can be achieved, the amount of exchanged data between communication parties increases. In contrast, Falcon reduces the communication overhead while increasing the execution time of the security functionalities. The selection of an hybrid PQC-enhanced TLS 1.3 will heavily depend on the characteristics of the communication and processing capabilities of the application. Moreover, the whole security implications need further evaluation.

REFERENCES

[1] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[2] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. Ieee, 1994, pp. 124–134.

[3] G. L.K., "A fast quantum mechanical algorithm for database search," in *28th Annual ACM Symposium on the Theory of Computing*, May 1996, p. 212.

[4] "Cryptographic Mechanisms: Recommendations ," https://www.bsi.bund.de/EN/Topics/Cryptography/PostQuantumCryptography/post_quantum_cryptography_node.html, accessed: 2022-03-13.

[5] National Institute of Standards and Technology, "Status report on the first round of the NIST post-quantum cryptography standardization process," 2019, https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8240.pdf.

[6] C. Allen and T. Dierks, "The TLS Protocol Version 1.0," RFC 2246, Jan. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2246

[7] E. Rescorla and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5246

[8] E. Rescorla, "HTTP Over TLS," RFC 2818, May 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2818

[9] N. I. of Standards and Technology, "Announcing request for nominations for public-key post-quantum cryptographic

[10] M. Braithwaite, "Experimenting with post-quantum cryptography," Jul. 2016, zugriff am 05.01.2022. [Online]. Available: http://flexikon.doccheck.com/ICD-10

algorithms," 2016. [Online]. Available: https://csrc.nist.gov/news/2016/public-key-postquantum-cryptographic-algorithms

[11] s. o. a. G. Adam Langley, "Cecpq1 results," 28 Nov 2016. [Online]. Available: https://www.imperialviolet.org/2016/11/28/cecpq1.html

[12] K. Kwiatkowski, "Towards post-quantum cryptography in tls," 2019. [Online]. Available: https://blog.cloudflare.com/towards-post-quantum-cryptography-in-tls/

[13] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, "Post-quantum authentication in TLS 1.3: A performance study." Internet Society, 2020.

[14] "Oqs openssl." [Online]. Available: https://github.com/open-quantum-safe/openssl

[15] C. Paquin, D. Stebila, and G. Tamvada, "Benchmarking post-quantum cryptography in TLS," in *Post-Quantum Cryptography*. Springer International Publishing, 2020, pp. 72–91.

[16] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. Schneider, "Post-quantum TLS on embedded systems: Integrating and evaluating kyber and SPHINCS with mbed TLS," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*. ACM, oct 2020.

[17] J. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, 1983.

[18] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Tech. Rep., jan 1978.

[19] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, nov 1976.

[20] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," *AES Algorithm Submission*, September 3, 1999.

[21] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*. ACM Press, 1996.

[22] T. Takagi, Ed., *Post-Quantum Cryptography*. Springer International Publishing, 2016.

[23] W. Beullens, "Improved cryptanalysis of uov and rainbow," Cryptology ePrint Archive, Report 2020/1343, 2020, https://ia.cr/2020/1343.

[24] P. Project, "Pqclean." 2019. [Online]. Available: https://github.com/PQClean/PQClean

[25] G. P. Insights, "Improve server response time." [Online]. Available: https://developers.google.com/speed/docs/insights/Server