2024 IFIP Networking Conference (IFIP Networking)

A Quantum of QUIC: Dissecting Cryptography with Post-Quantum Insights

Marcel Kempf, Nikolas Gauder, Benedikt Jaeger, Johannes Zirngibl, Georg Carle
Technical University of Munich, Germany
{kempfm, gauder, jaeger, zirngibl, carle}@net.in.tum.de

***Abstract*—QUIC is a new network protocol standardized in 2021. It was designed to replace the TCP/TLS stack and is based on UDP. The most current web standard HTTP/3 is specifically designed to use QUIC as transport protocol. QUIC claims to provide secure and fast transport with low-latency connection establishment, flow and congestion control, reliable delivery, and stream multiplexing. To achieve the security goals, QUIC enforces the usage of TLS 1.3. It uses authenticated encryption with additional data (AEAD) algorithms to not only protect the payload but also parts of the header. The handshake relies on asymmetric cryptography, which will be broken with the introduction of powerful quantum computers, making the use of post-quantum cryptography inevitable.**

This paper presents a detailed evaluation of the impact of cryptography on QUIC performance. The high-performance QUIC implementations LSQUIC, quiche, and MsQuic are evaluated under different aspects. We break symmetric cryptography down to the different security features. To be able to isolate the impact of cryptography, we implemented a `NOOP` AEAD algorithm which leaves plaintext unaltered. We show that QUIC performance increases by 10 to 20 % when removing packet protection. The header protection has negligible impact on performance, especially for AES ciphers. We integrate post-quantum cryptographic algorithms into QUIC, demonstrating its feasibility without major changes to the QUIC libraries by using a TLS library that implements post-quantum algorithms. Kyber, Dilithium, and FALCON are promising candidates for post-quantum secure QUIC, as they have a low impact on the handshake duration. Algorithms like SPHINCS$^+$ with larger key sizes or more complex calculations significantly impact the handshake duration and cause additional issues in our measurements.

***Index Terms*—QUIC, Cryptography, Performance Evaluation, Post-Quantum, Secure Transport Protocols**

I. Introduction

QUIC is a new transport protocol designed to improve on the widely used TCP/TLS stack, standardized by the Internet Engineering Task Force (IETF) in 2021 [1]. It is the basis for new protocols like HTTP/3 and MASQUE, which powers Apple's private relay service. Like TCP, it is connection-oriented, reliable, and features flow and congestion control. Additionally, QUIC has numerous advantages over TCP, *e.g.*, support for connection migration, stream multiplexing, and always-on encryption. To achieve the latter, TLS 1.3 is strictly integrated into QUIC [2]. The QUIC handshake combines both the transport and TLS handshake, which allows fast connection establishment. Furthermore, it encrypts all following payload and adds additional header protection.

ISBN 978-3-903176-63-8

These properties are desirable in many use cases. However, always requiring TLS is often criticized for inducing additional overhead in scenarios where those properties are not required [3, 4]. Jaeger et al. [5] have shown that crypto is the second most expensive component of QUIC besides packet I/O.

Besides the effect of symmetric cryptography on performance during the connection, the integration and performance evaluation of quantum-safe algorithms in QUIC has not been evaluated in detail. Traditional asymmetric cryptography, which is used during the QUIC handshake, will be broken with the introduction of powerful quantum computers. The National Institute of Standards and Technology (NIST) has been working on selecting quantum-safe cryptographic algorithms for standardization since 2017 [6].

In this work, we perform a detailed evaluation of the impact of cryptography on QUIC performance. We analyze the impact of symmetric cryptography in form of packet and header protection during a connection. Different post-quantum key exchange and signature algorithms are integrated into two QUIC implementations to evaluate the performance impact on the handshake.

Our key contributions in this work are:

(i) We evaluate the impact of cryptography on QUIC performance for different libraries in detail. We differentiate between the impact of payload encryption and header protection in a controlled environment.

(ii) We analyze if larger Maximum Transmission Units (MTUs) can mitigate the impact of encryption and improve the performance of QUIC. This is especially relevant in controlled environments, *e.g.*, in-datacenter networks.

(iii) We integrate quantum-safe cryptographic algorithms into two of the chosen QUIC implementations to evaluate the performance impact. We dissect the handshake to precisely locate performance bottlenecks and limitations.

(iv) We publish versions of BoringSSL and OpenSSL with `NOOP` ciphers following the required interface. These can be used to evaluate the impact of cryptography within other QUIC libraries or to remove the impact of cryptography for other evaluations.

We explain relevant background regarding cryptography in QUIC in Section II and cover related work in Section III. In Section IV, we introduce our approach and the measurement setup. Our evaluations are presented in Section V. Finally, the main findings are concluded in Section VI.

978-3-903176-63-8/24/$31.00 ©2024 IEEE

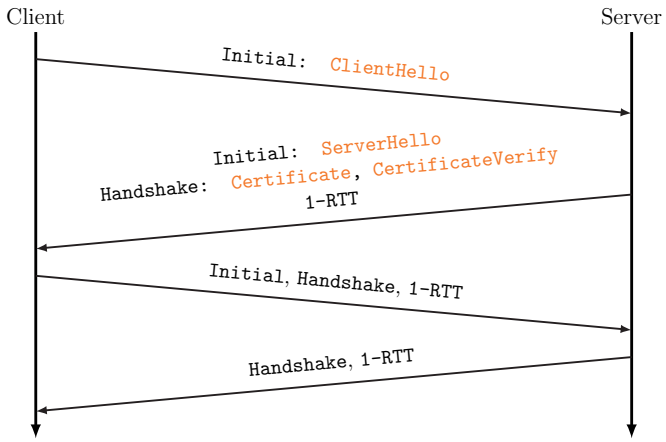

Fig. 1: Simplified illustration of a QUIC 1-RTT handshake.

II. Background

This section introduces relevant background about cryptography in QUIC, followed by an overview of post-quantum cryptography (PQC) and how QUIC is affected when integrating PQC.

A. QUIC Cryptography

QUIC includes always-on encryption with TLS 1.3. While asymmetric cryptography is used during the handshake, symmetric cryptography is used during the connection. In Figure 1, a simplified illustration of a QUIC handshake is shown. Asymmetric cryptography is only happening in the orange parts. It is important to note that all shown handshake components may be spread over multiple QUIC packets. This happens especially with PQC, where the certificate is too large to fit into a single packet.

For performance and security considerations, TLS 1.3 limits the amount of available ciphers to only authenticated encryption with additional data (AEAD) algorithms [7]. They are designed to encrypt data while applying integrity protection to the data itself and also additional metadata in one single pass. During the connection, the packet payload is encrypted, and the header is integrity-protected along with the payload. All non-AEAD algorithms have been pruned from the standard and only five are available for TLS 1.3. QUIC further limits this set to four: `AES_128_GCM`, `AES_128_CCM`, `AES_256_GCM`, and `CHACHA20_POLY1305` [2]. The Advanced Encryption Standard (AES) algorithms are block ciphers which profit from hardware acceleration on modern `x86` CPUs [8], while ChaCha20 [9] is a stream cipher performing well when hardware acceleration is lacking.

Besides protecting the payload, parts of the header are also encrypted. To prevent network ossification and ensure header authenticity, all fields not required for decryption are protected during the connection. This includes the packet number and several bits in the header. Figure 2 shows the way a QUIC packet is protected and which keys are involved. First, the packet protection is applied by encrypting the payload, using the AEAD cipher. The header serves as additional data and

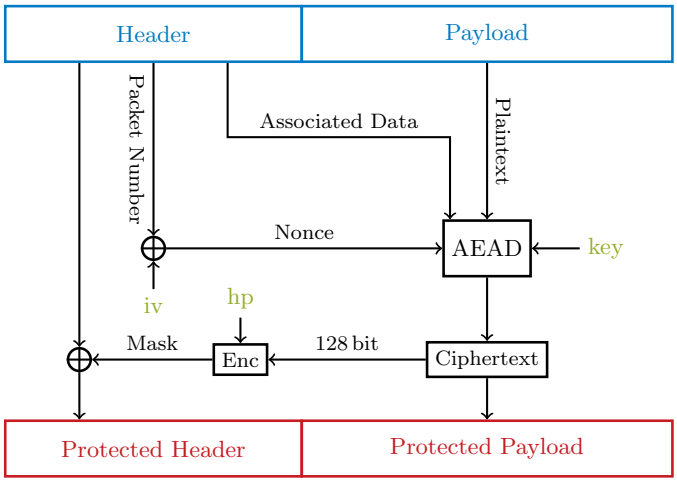

Fig. 2: Packet and header protection in QUIC using the initialization vector (iv), the header protection key (hp), and the QUIC key (key). All three are derived from the connection's TLS secrets.

is not encrypted but authenticated. The nonce is derived by XORing the packet number with the initialization vector, ensuring that the nonce is unique for every packet. From the resulting ciphertext of the AEAD encryption, a 128 bit sample is taken and used as input to a cipher. The AEAD algorithm for the packet protection determines the respective cipher. If an AES cipher suite is used, the respective AES cipher is applied in Electronic Code Book (ECB) mode. For the ChaCha20 cipher suite, the raw ChaCha20 function is used. From the ciphertext of this single encryption call, 8 B are used to mask the fields to be protected in the header [2].

B. Quantum-Safe Cryptography

Traditional asymmetric cryptography in TLS is based on prime factorization or the (elliptic-curve) discrete logarithm problem. Both can be solved with powerful quantum computers using Shor's algorithm [10]. Using Grover's algorithm [11], symmetric keys can be brute-forced more efficiently, halving the security level in bit. However, symmetric cryptography is not as vulnerable, as larger keys can mitigate this problem. Hence, PQC introducing new quantum-secure cryptographic systems is needed as a replacement. In light of this, the NIST launched the *Post-Quantum Cryptography Standardization* program in 2017 to standardize quantum-secure cryptographic primitives [6].

In this work, we focus on the following post-quantum key exchange and signature algorithms: Kyber, BIKE, HQC, Dilithium, Falcon, and SPHINCS$^+$. Detailed information about these algorithms can be found in the *liboqs* documentation from Open Quantum Safe (OQS) [12]. For the ones selected for standardization, the NIST has assigned new names, *e.g.*, ML-KEM for Kyber [13]. In this paper, we use the old names for the algorithms, as they are more commonly known. The algorithms are grouped into hash-, code-, and lattice-based algorithms, each coming with unique advantages and limitations [14].

NIST also established different quantum security strength categories to compare various algorithms regarding their security. Relevant to our work are the so-called NIST levels I, III, and V. Algorithms in NIST level I are at least as hard to break as AES-128 through exhaustive key search. The NIST levels III and V correspond to the strength of AES-192, respective AES-256.

III. Related Work

Jaeger et al. [5] performed a broad and comprehensive performance evaluation of QUIC libraries. Among other implementations, they evaluated LSQUIC [15] and quiche [16] as well as TCP over TLS. In their comparison of the goodput they revealed that the performance of hardware accelerated AES is superior to ChaCha20. In their CPU utilization measurements, they found that cryptographic operations contribute between 10 % and 20 % to the total CPU utilization. We further extend the evaluation with MsQuic [17] and focus on the goodput for the different cipher suites rather than on the effect of hardware acceleration. We also provide a fine-grained analysis of the different components of the symmetric cryptography, *i.e.*, the header and packet protection, and the impact of the MTU on the performance of QUIC and cryptography. Lastly, we integrate quantum-safe cryptographic algorithms to evaluate the performance impact on the handshake and identify possible problems when integrating PQC into QUIC.

Yang et al. [18] analyzed different QUIC implementations in the context of Network Interface Card (NIC) offloading, aiming to define a set of primitives that a NIC should offer to efficiently offload QUIC. They looked at the following implementations of QUIC: Quant [19], Quicly [20], picoquic [21], and mvfst [22]. Like Jaeger et al., they showed that high costs are associated with crypto: up to 40 % of the CPU usage is attributed to cryptographic operations (Quant). More specifically, they discovered that around 75 %–80 % of the crypto-related CPU usage is associated with AEAD function calls. We analyze the cryptographic operations more thoroughly and break them down in more detail to show the impact of QUIC's security features, *i.e.*, the header and packet protection.

Apart from the works previously presented, various papers deal with the analysis of QUIC itself without focusing specifically on cryptography [23, 24]. They performed comparative studies on the performance of TCP and QUIC, *e.g.*, Yu and Benson [24] ran their tests under different network conditions and workloads against production endpoints from Google, Cloudflare, and Facebook.

Marx et al. [25] researched QUIC features in 15 HTTP/3 implementations, *e.g.*, flow and congestion control, stream multiplexing, and the 0-RTT handshake. They summarize that there are significant differences regarding the quality of the QUIC implementations and that most of them are not completely optimized, wasting potential performance gains. However, in this work, the focus is on the cost of cryptography.

While some studies evaluated path MTU discovery in general or whether it is implemented (*e.g.*, Marx et al. [25]), to the best of our knowledge, no study evaluated the impact of larger MTUs on QUIC performance and its relation to cryptography.

Sosnowski et al. [26] investigated the performance implications of using post-quantum algorithms in TLS 1.3 handshakes over TCP. Their results reveal that PQ algorithms (including hybrids) can be faster than the state-of-the-art in ideal network conditions. However, in low-bandwidth environments, the increased data usage becomes a bottleneck. Moreover, they found that the large key sizes can cause unwanted side effects, e.g., additional RTTs due to the slow start phase of the TCP congestion control. We use QUIC instead of TCP and also evaluate the integration into QUIC libraries and the corresponding issues.

Raavi et al. [27] analyzed the impact of PQC on the QUIC handshake. They found that the handshake takes longer with increasing security levels or worse network conditions. They only looked at the two lattice-based signature algorithms Dilithium and FALCON and did not analyze key encapsulation mechanisms, what we do in this work.

IV. Approach

In this section, we present our approach to conduct measurements and evaluate collected metrics. After introducing the selected QUIC libraries, we present the adjustments made to the TLS libraries to allow for measurements without cryptography.

A. Measurement Framework

To execute our measurements in a reproducible manner, we extended the adapted QUIC Interop Runner presented by Jaeger et al. [5]. This framework was built to orchestrate measurements on bare-metal servers and to provide a reproducible environment for QUIC measurements with extensible configuration and logging capabilities. It is based on the QUIC Interop Runner presented by Seemann and Iyengar [28], which was initially designed to perform interoperability tests between different QUIC implementations. We added features to change the path MTU and modified the build process to include our custom TLS libraries presented in Section IV-D.

B. Hardware Configuration

All machines used for the measurements are equipped with an AMD EPYC 7543 32-Core CPU, 512 GB memory, and a 10GBASE-T Broadcom BCM57416 NIC. We use Debian Bullseye on 5.10.0-8-amd64 as the operating system without additional configurations.

C. Implementations

For our evaluation, we consider LSQUIC [15], quiche [16], and MsQuic [17], as those implementations are widely used for production QUIC servers [29]. Related work also showed that these implementations perform better than the majority of other tested QUIC implementations [5, 30]. We use the example server and client applications provided by the respective libraries for interop testing. While we configured LSQUIC and quiche to use HTTP/3, MsQuic provides only an HTTP/0.9

implementation. For the MsQuic measurements, we used the `QUIC_PARAM_CONN_DISABLE_1RTT_ENCRYPTION` connection parameter to disable the 1-RTT encryption completely. We refer to this configuration as `NOENC` in the following. As this feature is only for testing and performance evaluation, a constant must be defined to enable the so called "insecure features". Additionally, we use a TCP / TLS stack consisting of a server using *nginx* and a client using *curl* for comparison.

D. Adjustments to BoringSSL and OpenSSL

QUIC is strongly coupled with TLS encryption and is generally not designed to operate without it. Completely removing cryptography from a QUIC implementation requires major adjustments to the library. Therefore, we opted for implementing a `NOOP` cipher for *BoringSSL* [31] and *OpenSSL* [32] which just returns the plain text as cipher text and thus does not perform any cryptographic operations. This approach allows us to easily perform measurements with other QUIC libraries using *BoringSSL* or *OpenSSL* as the TLS library.

Our cipher suite `TLS_NOOP_SHA256` uses SHA-256 for hashing and the custom `NOOP` algorithm for encryption and decryption. The asymmetric part during the handshake remains unchanged. As the algorithm for header protection depends on the selected AEAD algorithm [2, Section 5.4.1], we decoupled it from the AEAD algorithm in the used QUIC libraries and thus are able to choose the header protection algorithm. This approach allows for easy integration and evaluation of other QUIC libraries with our custom TLS libraries. The patched *BoringSSL* and *OpenSSL* libraries are available on GitHub [33]. To be able to also perform measurements with quantum-resistant cryptographic algorithms, we also integrated our changes into the *BoringSSL* fork of the OQS project [34]. The quantum-safe key exchange and signature algorithms are included via the *liboqs* library, which also originates from the OQS project.

V. Evaluation

With the previously presented measurement framework and the modified QUIC implementations and TLS libraries, we evaluate the cost of cryptography in QUIC.

Besides evaluating the different AEAD algorithms and comparing them with our `NOOP` implementation, we also evaluate the impact of QUIC's header protection on the goodput. The cost of the different cryptographic operations is evaluated in detail with profiling output.

After analyzing symmetric cryptography, we benchmark post-quantum key exchange and signature algorithms integrated into LSQUIC and quiche. Also looking at hybrid approaches, the impact on the handshake latency is evaluated.

In every measurement, the client downloads an 8 GiB file over HTTP. To ensure a large enough sample, every measurement was repeated 25 times and the average was taken. We also set the UDP receive buffer size to 6656 KiB which is 32 times the default size of 208 KiB. Jaeger et al. [5] have shown that the default buffer size is too small for high-rate links. The congestion control algorithm was fixed to `cubic` for all measurements. If not stated otherwise, no header protection is applied in the measurements with the `NOOP` cipher. All shown boxplots use a horizontal line for the median and an icon such as ▲ for the mean. The boxes are drawn from quartile Q_1 to Q_3.

A. AEAD Algorithms

Comparing TLS over TCP with QUIC highlights how QUIC's use of UDP affects performance. Figure 3 shows the goodput achieved with different AEAD ciphers `AEAD_AES_128_GCM`, `AEAD_AES_256_GCM`, and `AEAD_CHACHA20_POLY1305` as well as our `NOOP` implementation. TLS over TCP consistently outperforms all tested QUIC implementations across all AEAD algorithms, likely due to TCP's mature optimizations.

There are no noticeable differences in performance between the 128-bit and the 256-bit AES cipher. The AEAD algorithm `AEAD_CHACHA20_POLY1305` using the ChaCha20 stream cipher is about 9 % to 16 % slower than AES-based algorithms in combination with hardware acceleration. As it was shown by Jaeger et al., ChaCha20 shows a significant performance improvement over AES when hardware acceleration is not available and is therefore a valuable alternative for endpoints with hardware constraints [5]. With the `NOOP` cipher, the achieved goodput increases between 10 % and 20 % for the tested QUIC implementations and 12 % for TLS over TCP.

We also measured the goodput of the MsQuic implementation with `NOENC`. This configuration completely disables the 1-RTT encryption and thus the cryptographic operations. In comparison to the `NOOP` cipher, the goodput increases by 13.6 % to 4782 Mbit/s. The main difference between the two is that `NOENC` does not even call *OpenSSL* for the protection of 1-RTT packets, while `NOOP` still calls *OpenSSL* performing operations like `memcpy()`. This performance impact is also analyzed in Section V-C, where we take a closer look at the CPU utilization of the different cryptographic operations.

B. Header Protection

With a combination of the options to select the AEAD algorithm and the header protection algorithm, we can evaluate the impact of header protection on the goodput.

Figure 4 shows the goodput achieved with different AEAD algorithms, with and without header protection. As it can be seen, the AES header protection has a negligible impact on the goodput. While it has less than 1 % impact for LSQUIC and MsQuic, the goodput of quiche is slightly increased by 2 %. The header protection with ChaCha20 has a higher impact on the goodput, increasing it by 4 % to 6 %. As ChaCha20 showed slightly lower goodput than AES in Figure 3 already, this observation is not surprising. It can be concluded that the header protection, especially for AES, is virtually free and does not have a significant impact on the goodput.

C. CPU Time Consumption for Packet and Header Protection

To understand what limits the goodput and where bottlenecks are, we take a closer look at the CPU utilization of

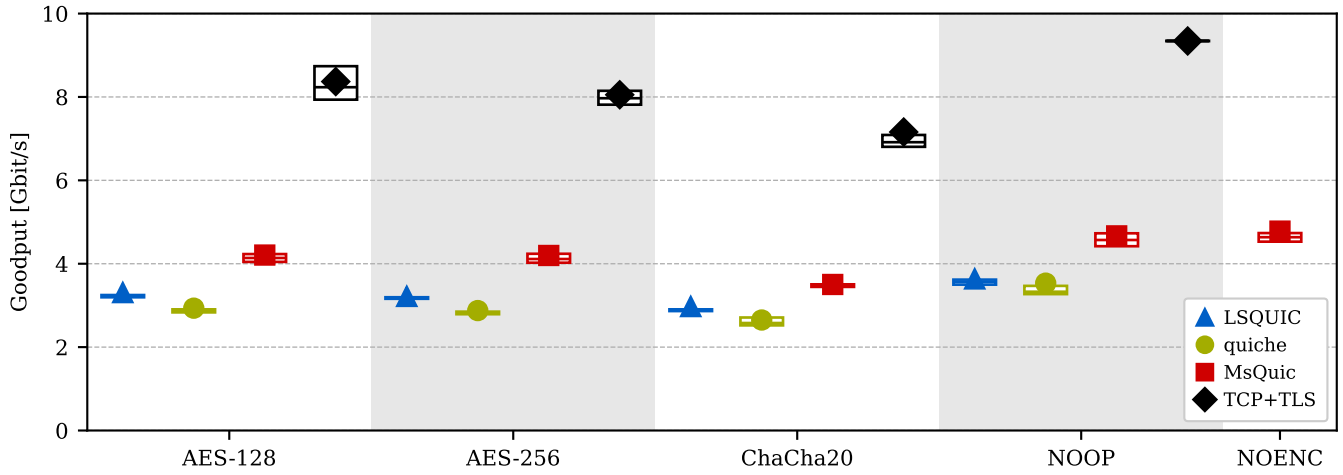

Fig. 3: Goodput achieved with different AEAD ciphers.

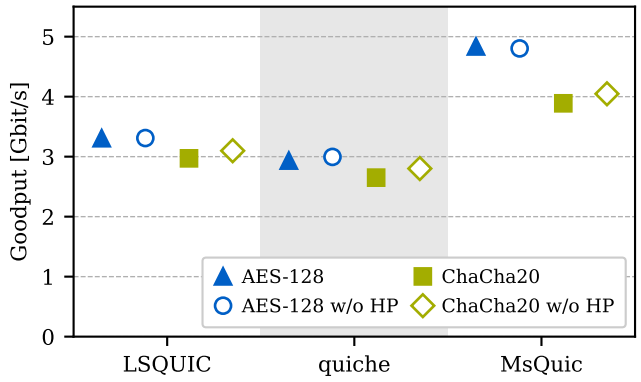

Fig. 4: Impact of header protection for different AEAD algorithms and QUIC implementations on the goodput.

client and server. We use `perf` in combination with a custom mapping to retrieve the number of perf samples for packet and header protection.

Table I shows the distribution of samples for the packet and header protection mechanisms in relation to the total number of samples belonging to the respective QUIC implementation. Each implementation was tested with `AEAD_AES_128_GCM`, `AEAD_CHACHA20_POLY1305`, and our `NOOP` cipher for a better comparison. AES and ChaCha20 use their respective header protection algorithm. For MsQuic, we also included results from measurements with `NOENC`. As AES-256 again shows similar performance to AES-128, it is not included in the table.

The results show that the packet protection is the primary contributor to the CPU time consumption on both endpoints. As expected, the results with an AES cipher show a lower percentage of samples for packet and header protection than those with a ChaCha20 cipher. The fact that ChaCha20 does not profit from hardware acceleration is reflected in the higher percentage of samples for packet and header protection and a lower goodput, as it was shown in Figure 3.

TABLE I: Distribution of `perf` samples for the packet protection (PP) and header protection (HP) mechanisms in relation to the total number of samples belonging to the respective QUIC implementation.

		Client		Server	
Cipher	**Impl.**	**PP [%]**	**HP [%]**	**PP [%]**	**HP [%]**
AES	LSQUIC	16.83	0.31	14.83	0.08
	quiche	15.30	2.45	14.75	0.92
	MsQuic	16.79	0.68	27.25	0.66
ChaCha20	LSQUIC	21.68	3.29	20.40	2.71
	quiche	21.02	4.49	20.37	3.32
	MsQuic	29.12	3.69	43.32	5.23
NOOP	LSQUIC	3.11	0.03	2.96	0.01
	quiche	1.91	2.07	1.35	0.07
	MsQuic	1.94	0.59	2.54	0.01
NOENC	MsQuic	0.37	0.52	0.05	0.01

It is also noticeable that the percentage of samples for header protection is higher for quiche, also with the `NOOP` cipher. This is caused by the fact that quiche performs operations for header parsing in the same function where the header protection is removed. After a closer look at the quiche source code, the operations with zero-copy mutable byte buffers and the used return type are the main contributors to the higher cost for header protection.

When analyzing the results with the `NOOP` cipher, it can be seen that these operations contribute around 2 % on the quiche client. After subtracting the 2 % for header protection, from the other values for quiche clients, we receive similar values as for the other clients.

Comparing the results for MsQuic with `NOENC` to the results with the `NOOP` cipher, we can see a further reduction of the collected samples for packet protection. This computational difference is the cost of the `NOOP` cipher, which still calls *OpenSSL* then executing `memcpy()`.

As we have already shown in Section V-B, the header

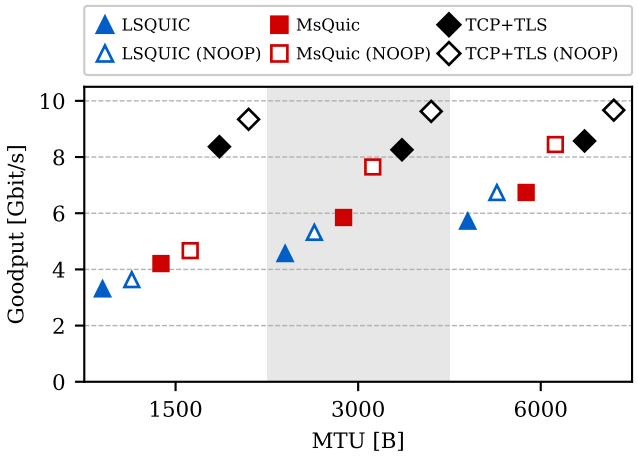

Fig. 5: Goodput achieved with different MTUs for different QUIC implementations, each one with AES and `NOOP` cipher.

protection for AES ciphers does not have a significant impact on the performance, contributing less than 1 % to the total CPU time consumption on both endpoints.

D. MTU Impact

Even though transmitting IP packets of larger than 1500 B through the Internet is unrealistic, it is attractive for datacenter and local company networks. As with TCP, the MTU has a significant impact on the performance of QUIC. With larger MTUs, the amount of packets to be sent is reduced, resulting in fewer per-packet overheads. To elaborate on the advantages of larger MTUs in combination with QUIC's always-on encryption, we performed measurements with MTU values of 1500 B, 3000 B, and 6000 B. For each MTU, we measured with LSQUIC, MsQuic and the TCP / TLS stack for comparison, each one with AES and `NOOP` cipher. It has to be noted that TCP / TLS stack is limited by the link rate of 10 Gbit/s and therefore does not benefit from larger MTUs here. The base MTU was set to 1500 B for measurements with a MTU over 1500 B. The implementations perform MTU discovery and adjust the MTU accordingly. We do not include results with quiche here, as quiche does not support changing the MTU without introducing major changes to the library.

In Figure 5, the results of the measurements are shown. As expected, the goodput increases with larger MTUs for every tested implementation. Both LSQUIC and MsQuic show an increase in goodput of almost 40 % when the MTU is increased from 1500 B to 3000 B.

When looking at packets sent from server to client, both QUIC implementations reach an average packet size of at least 1498 B when the MTU is set to 1500 B. TCP / TLS only reaches such values with our `NOOP` cipher, sending packets with a slightly lower average size of 1489 B with encryption enabled. For the increased MTU of 3000 B, MsQuic and TCP / TLS reach an average frame size between 2953 B and 2999 B. LSQUIC does not use the larger MTU and only reaches an average frame size of around 2350 B, leaving more than 600 B per packet unused. With the MTU set to 6000 B, this behavior is even more pronounced. The LSQUIC server sends packets of an average size of 3510 B, while the MsQuic makes use of the larger MTU and sends packets of an average size of more than 5960 B. However, for none of the tested QUIC implementations, the relative gain in goodput when switching to the `NOOP` cipher increases with larger MTUs.

As the goodput gap between QUIC and TCP / TLS decreases with larger MTUs, the performance of QUIC is more competitive with TCP / TLS on 10 Gbit/s links. In controlled environments with high bandwidth scenarios like datacenters, increasing the MTU makes QUIC a viable alternative to TCP / TLS.

TABLE II: Median TTFB and QUIC handshake packet count for different QUIC implementations and post-quantum KEMs at different NIST levels. In the packet count columns, the first/second number represents the amount of packets sent by the client/server. All instantiations were measured with an RSA-2048 certificate and `AEAD_AES_128_GCM` as AEAD algorithm. **Bold** algorithms are not quantum-safe.

	KEM	TTFB [ms]		Packets [C / S]	
		LSQUIC	quiche	LSQUIC	quiche
I	**X25519**	3.91	3.57	3 / 3	3 / 3
	Kyber512	4.08	3.39	3 / 3	3 / 3
	BIKE-L1	6.59	5.86	4 / 4	4 / 5
	HQC-128	5.57	4.21	5 / 6	5 / 8
	P-256	3.90	3.49	3 / 3	3 / 3
	P-256 + Kyber512	4.43	3.74	3 / 3	3 / 3
	P-256 + BIKE-L1	6.95	6.27	4 / 4	4 / 5
	P-256 + HQC-128	5.99	4.52	5 / 6	5 / 8
III	Kyber768	4.23	3.78	4 / 3	4 / 5
	BIKE-L3	11.75	10.49	5 / 5	5 / 6
	HQC-192	7.57	4.81	6 / 10	7 / 12
	P-384	7.36	6.76	3 / 3	3 / 3
	P-384 + Kyber768	8.99	8.67	4 / 4	4 / 5
	P-384 + BIKE-L3	16.69	15.14	5 / 5	6 / 7
	P-384 + HQC-192	12.42	9.44	7 / 10	7 / 12
V	Kyber1024	4.43	3.81	4 / 4	4 / 5
	BIKE-L5	22.27	20.08	7 / 7	7 / 8
	HQC-256	10.15	6.04	9 / 15	10 / 17
	P-521	12.24	11.86	3 / 3	3 / 3
	P-521 + Kyber1024	15.98	15.12	4 / 4	4 / 5
	P-521 + BIKE-L5	33.67	31.99	7 / 7	7 / 8
	P-521 + HQC-256	22.11	17.78	9 / 15	10 / 17

E. Post-Quantum Cryptography

With the *BoringSSL* fork from OQS introduced in Section IV-D, we measured the additional costs by using PQC during the QUIC handshake. MsQuic was not included in these measurements, as it uses *OpenSSL*. Since only the handshake is affected in these measurements the goodput is not a suitable metric. The filesize of the requested file was reduced to 1 B and the time between the different steps of the handshake was measured. We define the metric TTFB as the time between the client sending its `ClientHello` and being able to send its HTTP/3 request to the server. The amount of RTTs needed for the handshake can be neglected here, as the RTT in our measurement setup is below 0.1 ms. Additionally, the difference in RTTs until the client is able

to start sending the HTTP/3 request is not greater than 1 RTT between LSQUIC and quiche for most of the performed measurements.

First, we take a look at the KEMs Kyber, BIKE, and HQC. Kyber was chosen for standardization by the NIST [35] while BIKE and HQC were selected for the fourth round of the NIST PQC standardization process [36].

As a baseline, the traditional key exchange method ECDHE was tested with Curve25519 (X25519), P-256, P-384, and P-521. Additionally, hybrid KEM algorithms were measured, which combine the respective post-quantum KEM with pre-quantum ECDHE using P-256, P-384 or P-521. By using multiple key exchange algorithms simultaneously and combining the results, security is provided even if one of the two used algorithms turns out to be broken [37]. This might happen if either the new quantum-safe algorithms turn out to be insecure or if the traditional algorithms are broken by a quantum computer.

In Table II, the results of the measurements are shown. The tested KEMs are grouped by their respective NIST levels. In all NIST levels, Kyber shows the fastest handshake from the tested post-quantum KEMs. As Kyber is the only KEM that is lattice-based, it profits from smaller key sizes. The public key sent as key share in the `ClientHello` and the ciphertext sent as a key share in the `ServerHello` influence the number of packets each endpoint sends during the handshake. The different sizes of the public keys and ciphertexts for the benchmarked PQC KEM algorithms are visualized in Figure 6. For all tested ECDHE algorithms, the size of the public keys and ciphertexts is between 32 B and 133 B and therefore negligible small. Kyber provides pleasantly small public keys and ciphertexts of under 2 kB, even with Kyber1024 on NIST level V. While a 800 B public key is sent in the `ClientHello` with Kyber512, HQC-128 needs to send 2249 B. The TTFB measured with Kyber as KEM is only slightly increased compared to X25519, even on NIST level V.

We noticed a slightly higher TTFB with LSQUIC than with quiche. After taking a closer look, it was noticeable that the LSQUIC client needs more time to process the `ServerHello` message. With HQC-256, this difference is most pronounced, with the LSQUIC client needing around 5 ms instead of the around 1.5 ms quiche needs.

In our measurements, BIKE is the slowest KEM on all NIST levels. With BIKE-L5, the TTFB extends five times that of Kyber1024 and more than twice that of the likewise code-based HQC-256. Although HQC-256 has nearly triple the ciphertext size of BIKE-L5 and a bigger public key, it still has an over 50 % lower TTFB.

As the hybrid KEMs concatenate the public keys and ciphertexts of the post-quantum KEM and the pre-quantum ECDHE algorithm, the sizes for the hybrid KEMs are the sum of the sizes of the KEMs being hybridized. With P-384 and P-521, the TTFB is approximately the sum of the TTFB for P-384, respectively, P-521 and the post-quantum KEM. Due to the high cost of ECDHE in these cases, the hybrid KEMs

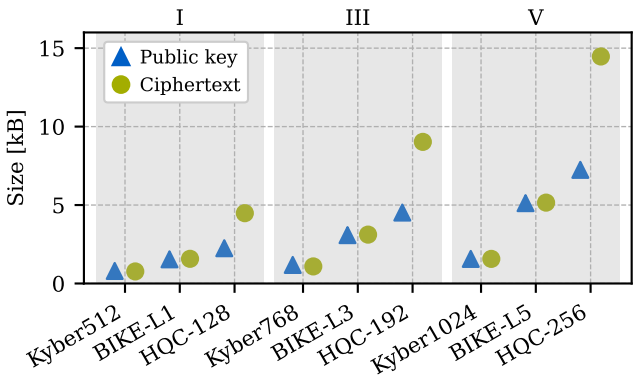

Fig. 6: Public key and ciphertext sizes of the benchmarked post-quantum KEMs grouped by NIST level.

are more expensive than the respective post-quantum KEMs. The hybrid KEMs with P-256 are only slightly slower than the used post-quantum KEMs and therefore a good choice for a hybrid approach.

To summarize, the major bottleneck of post-quantum key exchange algorithms in our measurements is the larger amount of data that needs to be transferred during the handshake. This increases the latency and TTFB. Since the sizes for hybrid schemes are only marginally larger and, with an efficient elliptic curve such as P-256, only marginally slower, it is best to use them if post-quantum security is desired as they guarantee security even if the post-quantum KEM turns out to be insecure.

For post-quantum signature schemes, we selected FALCON, Dilithium, and SPHINCS$^+$, which are all chosen for standardization by NIST [36]. We fixed the key exchange algorithm to X25519 ECDHE for our measurements. The traditional pre-quantum signature scheme RSA is used as a baseline with 1024 bit, 2048 bit, and 4096 bit keys.

The results of the measurements are shown in Table III, listing the aforementioned signature algorithms grouped by their respective NIST levels. With increasing sizes of the public key and the signature, the certificate grows bigger. Due to the increasing signature size, the TLS `CertificateVerify` also expands in size. This leads to more packets being sent by the server, which can be observed with Dilithium, FALCON, and SPHINCS$^+$, where the server must send up to 92 packets in the handshake. The client must also send extra packets to acknowledge the additional packets from the server. FALCON-512 is the post-quantum scheme with the slightest increase in latency: The TTFB for client and server is only about 0.6 ms higher compared to RSA-1024. FALCON-512 even performed better, in terms of latency, than RSA-2048.

For the RSA-4096, the TTFB for the client and server rises to an extreme 14 ms, making it the slowest of the tested pre-quantum schemes. It was slower than all the post-quantum schemes we looked at, except for SPHINCS$^+$. The SPHINCS$^+$ variants reach by far the highest TTFB of all evaluated signature schemes, caused by the huge signature

TABLE III: Median TTFB, QUIC handshake packet count, public key, signature, and certificate sizes for different QUIC implementations and post-quantum signature algorithms at different NIST levels. In the packet count columns, the first/second number represents the amount of packets sent by the client/server. All instantiations were measured with X25519 as KEM and `AEAD_AES_128_GCM` as AEAD algorithm. **Bold** algorithms are not quantum-safe.

	Signature algorithm	TTFB [ms]		Packets [C / S]		Sizes [B]		
		LSQUIC	quiche	LSQUIC	quiche	Public key	Signature	Certificate
	RSA-1024	2.11	1.88	3 / 2	3 / 2	128	128	528
	RSA-2048 (default)	3.91	3.57	3 / 3	3 / 3	256	256	789
I	**RSA-4096**	14.02	13.46	3 / 3	3 / 3	512	512	1301
	FALCON-512	2.78	2.05	3 / 4	3 / 4	897	666	1793
	SPHINCS$^+$-SHA2-128s	198.46	195.00	4 / 16	5 / 17	32	7856	8131
	SPHINCS$^+$-SHA2-128f	28.54	20.89	4 / 32	5 / 33	32	17088	17363
	SPHINCS$^+$-SHAKE-128s	385.60	367.45	5 / 16	5 / 17	32	7856	8131
	SPHINCS$^+$-SHAKE-128f	50.95	41.89	4 / 32	5 / 33	32	17088	17363
III	Dilithium3	4.26	2.14	4 / 10	4 / 10	1952	3293	5508
	SPHINCS$^+$-SHA2-192s	325.46	325.37	5 / 31	5 / 32	48	16224	16516
	SPHINCS$^+$-SHA2-192f	-	35.31	- / -	6 / 67	48	35664	35956
	SPHINCS$^+$-SHAKE-192s	590.73	601.04	6 / 31	5 / 32	48	16224	16516
	SPHINCS$^+$-SHAKE-192f	-	67.51	- / -	6 / 66	48	35664	35955
V	Dilithium5	5.15	2.30	4 / 13	4 / 12	2592	4595	7449
	FALCON-1024	4.25	3.02	4 / 7	4 / 6	1793	1280	3299
	SPHINCS$^+$-SHA2-256s	-	297.50	- / -	6 / 56	64	29792	30100
	SPHINCS$^+$-SHA2-256f	-	68.21	- / -	7 / 91	64	49856	50164
	SPHINCS$^+$-SHAKE-256s	-	536.84	- / -	6 / 56	64	29792	30100
	SPHINCS$^+$-SHAKE-256f	-	106.11	- / -	7 / 92	64	49856	50164

sizes. The fast variants (denoted by an f suffix) are still slower than any other signature scheme measured. The more compact signatures of the small variants (denoted by an s suffix) come at the expense of calculation time: The TTFB of SPHINCS$^+$-SHAKE-192s is over 142 times higher than Dilithium3's. The versions of SPHINCS$^+$ that used SHAKE were about twice as slow as those using SHA-2, even though they produced signatures of the same size.

In conclusion, post-quantum signature schemes come with larger public key and signature sizes than the RSA variants. FALCON is the quantum-safe signature scheme with the smallest signature and certificate size. Dilithium is larger while still having an acceptable signature and certificate size compared to the hash-based SPHINCS$^+$ with huge signatures of up to 49 kB. Moreover, we saw that increasing the RSA key size is insufficient to improve security while keeping the performance impact minimal. Instead, a performant post-quantum signature scheme like FALCON should be employed if quantum security is desired.

As can be seen from the missing values for LSQUIC in Table III, the measurements with certificates larger than 30 kB have failed. LSQUIC's server had problems sending out the `ServerHello`. For the handshake, LSQUIC uses so-called *mini connections*, which allocate less memory to protect the server from DoS attacks. Those mini connections use bitmasks to keep track of packet numbers.[1] Due to the variable length of the bitmasks, only up to 64 packets are supported. This limit is exceeded with the huge signature and certificate sizes of SPHINCS$^+$, as it can be seen for the quiche measurements. This indicates that LSQUIC is not ready for post-quantum signature schemes with huge certificate and signature sizes like some parameter sets of SPHINCS$^+$, even if its use in QUIC is of questionable value because of the poor performance.

Another issue arises with large certificate sizes in combination with QUIC's address validation. The server is not allowed to send more than three times as many bytes as the number of received bytes if the client address is not yet validated [1, Section 8.1]. To validate the client's address before completing the TLS handshake, the server can send a `Retry` packet. However, this causes an additional RTT and therefore increases the TTFB. By using larger MTU values, the `ClientHello` can be padded to larger sizes, which can mitigate this issue.

VI. Conclusion

In this work, we evaluate the impact of cryptography on QUIC performance. The cost of symmetric cryptography during the connection, consisting of packet and header protection, is analyzed. We additionally evaluate the asymmetric cryptography happening during the handshake with precise measurements. We integrate quantum-safe cryptographic algorithms into the chosen QUIC implementations to evaluate the performance impact and identify possible problems when integrating PQC into QUIC.

Our analysis of cipher suites shows that hardware-accelerated `AEAD_AES_128_GCM` is the most efficient AEAD algorithm for header and packet protection. Compared to packet protection, header protection has little impact on CPU time consumption and goodput. Especially for AES ciphers, the header protection is virtually free. We reveal that increasing the MTU does not mitigate the impact of

[1] https://lsquic.readthedocs.io/en/v4.0.0/internals.html#mini-ietf-connection

encryption, as using the `NOOP` cipher shows performance improvements also for larger MTUs.

The integration of quantum-safe cryptographic algorithms into QUIC is feasible without major changes to the QUIC libraries using *BoringSSL*. While algorithms with larger key sizes or more complex calculations have a significant impact on the handshake duration, algorithms like Kyber and Dilithium are promising candidates for post-quantum secure QUIC, as they have a low impact on the handshake duration. Large certificate sizes lead to different problems in our measurements, as the packet number space for `Handshake` packets might be limited or QUIC's address validation mechanism can cause an extra RTTs.

To allow for evaluations of other QUIC implementations, we publish the modified *BoringSSL* and *OpenSSL* libraries [33].

Acknowledgment

The European Union's Horizon 2020 research and innovation programme funded this work under grant agreements No 101008468 and 101079774. Additionally, we received funding by the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of the project 6G Future Lab Bavaria. This work is partially funded by Germany Federal Ministry of Education and Research (BMBF) under the projects 6G-life (16KISK001K) and 6G-ANNA (16KISK107) and the German Research Foundation under the project HyperNIC (CA595/13-1).

References

[1] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," RFC 9000, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[2] M. Thomson and S. Turner, "Using TLS to Secure QUIC," RFC 9001, May 2021. [Online]. Available: https://rfc-editor.org/rfc/rfc9001.txt

[3] QUIC IETF Mailinglist. (2020) A non-TLS standard is needed. Accessed: 2024-02-29. [Online]. Available: https://mailarchive.ietf.org/arch/msg/quic/SBetxLwCq5I7un2tkzFb7tXhJMU/

[4] ——. (2024) Historic TLS Discussion. Accessed: 2024-03-08. [Online]. Available: https://mailarchive.ietf.org/arch/msg/quic/rDUtUDVqz95JspgptALSNYcnn5c/

[5] B. Jaeger, J. Zirngibl, M. Kempf, K. Ploch, and G. Carle, "QUIC on the Highway: Evaluating Performance on High-Rate Links," in *International Federation for Information Processing (IFIP) Networking 2023 Conference (IFIP Networking 2023)*, Barcelona, Spain, Jun. 2023.

[6] United States National Institute of Standards and Technology, "Post-Quantum Cryptography Standardization," 2023, accessed: 2024-02-29. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

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

[8] S. Gueron. (2010) Intel® Advanced Encryption Standard (AES) New Instructions Set. Accessed: 2024-02-29. [Online]. Available: https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf

[9] Y. Nir and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols," RFC 7539, May 2015. [Online]. Available: https://www.rfc-editor.org/info/rfc7539

[10] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM review*, 1999.

[11] L. K. Grover, "A Fast Quantum Mechanical Algorithm for Database Search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, 1996.

[12] Open Quantum Safe project, "Algorithms in liboqs," 2024, accessed: 2024-02-29. [Online]. Available: https://openquantumsafe.org/liboqs/algorithms/

[13] Bas Westerbaan, "The state of the post-quantum Internet," March 5, 2024, accessed: 2024-03-08. [Online]. Available: https://blog.cloudflare.com/pq-2024

[14] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post Quantum Cryptography*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[15] LiteSpeed Tech. (2024) lsquic. Accessed: 2024-02-13. [Online]. Available: https://github.com/litespeedtech/lsquic

[16] Cloudflare. (2024) quiche. Accessed: 2024-02-13. [Online]. Available: https://github.com/cloudflare/quiche

[17] Microsoft. (2024) MsQuic. Accessed: 2024-02-13. [Online]. Available: https://github.com/microsoft/msquic

[18] X. Yang, L. Eggert, J. Ott, S. Uhlig, Z. Sun, and G. Antichi, "Making QUIC Quicker With NIC Offload," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2020. [Online]. Available: https://doi.org/10.1145/3405796.3405827

[19] NetApp. (2024) Quant. Accessed: 2024-02-13. [Online]. Available: https://github.com/NTAP/quant

[20] H2O Project. (2024) Quicly. Accessed: 2024-02-13. [Online]. Available: https://github.com/h2o/quicly

[21] Private Octopus. (2024) picoquic. Accessed: 2024-02-13. [Online]. Available: https://github.com/private-octopus/picoquic

[22] Facebook. (2024) mvfst. Accessed: 2024-02-13. [Online]. Available: https://github.com/facebookincubator/mvfst

[23] S. Bauer, P. Sattler, J. Zirngibl, C. Schwarzenberg, and G. Carle, "Evaluating the Benefits: Quantifying the Effects of TCP Options, QUIC, and CDNs on Throughput," in *Proceedings of the Applied Networking Research Workshop*, 2023.

[24] A. Yu and T. A. Benson, "Dissecting Performance of Production QUIC," in *Proceedings of the Web Conference 2021*, 2021.

[25] R. Marx, J. Herbots, W. Lamotte, and P. Quax, "Same Standards, Different Decisions: A Study of QUIC and HTTP/3 Implementation Diversity," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2020.

[26] M. Sosnowski, F. Wiedner, E. Hauser, L. Steger, D. Schoinianakis, S. Gallenmüller, and G. Carle, "The Performance of Post-Quantum TLS 1.3," in *Companion of the 19th International Conference on emerging Networking EXperiments and Technologies*, 2023.

[27] M. Raavi, S. Wuthier, P. Chandramouli, X. Zhou, and S.-Y. Chang, "QUIC Protocol with Post-quantum Authentication," in *Information Security*, 2022.

[28] M. Seemann and J. Iyengar, "Automating QUIC Interoperability Testing," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, 2020.

[29] J. Zirngibl, F. Gebauer, P. Sattler, M. Sosnowski, and G. Carle, "QUIC Hunter: Finding QUIC Deployments and Identifying Server Libraries Across the Internet," in *Proc. Passive and Active Measurement (PAM)*, 2024.

[30] M. König, O. P. Waldhorst, and M. Zitterbart, "QUIC(k) Enough in the Long Run? Sustained Throughput Performance of QUIC Implementations," in *2023 IEEE 48th Conference on Local Computer Networks (LCN)*, 2023.

[31] Google, "BoringSSL," 2024, accessed: 2024-02-29. [Online]. Available: https://boringssl.googlesource.com/boringssl

[32] OpenSSL Project Authors, "OpenSSL," 2024, accessed: 2024-02-29. [Online]. Available: https://www.openssl.org/

[33] M. Kempf, N. Gauder, B. Jaeger, J. Zirngibl, and G. Carle. (2024) Publication of modified TLS Libraries. [Online]. Available: https://github.com/tumi8/quic-crypto-paper

[34] Open Quantum Safe project, "BoringSSL," 2024, accessed: 2024-02-29. [Online]. Available: https://github.com/open-quantum-safe/boringssl

[35] National Institute of Standards and Technology, "FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard," 2024, accessed: 2024-02-29. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.203.ipd

[36] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," 2022, accessed: 2024-02-29. [Online]. Available: https://doi.org/10.6028/NIST.IR.8413-upd1

[37] D. Stebila, S. Fluhrer, and S. Gueron, "Hybrid key exchange in TLS 1.3," Internet Engineering Task Force, Internet-Draft draft-ietf-tls-hybrid-design-09, Sep. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design/09/