Article

Not peer-reviewed version

# An Applied Analysis of Securing 5G/6G Core Networks with Post-Quantum Key Encapsulation Methods

Paul Scalise , Robert Garcia , Matthew Boeding , Michael Hempel * , Hamid Sharif

Posted Date: 24 October 2024

doi: 10.20944/preprints202410.1802.v1

Keywords: post quantum cryptography; key encapsulation methods; CRYSTALS-kyber; 5G; 6G; core network; open source

*Article*

# An Applied Analysis of Securing 5G/6G Core Networks with Post-Quantum Key Encapsulation Methods

**Paul Scalise, Robert Garcia, Matthew Boeding, Michael Hempel *, Hamid Sharif**

Department of Electrical and Computer Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588, USA
* Correspondence: mhempel@unl.edu

**Abstract:** 5th Generation (5G) cellular networks have been adopted worldwide since the rollout began around 2019. It brought with it many innovations and new services, such as Enhanced Mobile Broadband (eMBB), Ultra Reliable and Low Latency Communications (URLLC), and Massive Internet of Things (mIoT). 5G also introduced a more scalable approach to network operations using fully software-based Virtualized Network Functions (VNF) in Core Networks (CN) rather than the prior hardware-based approach. However, while this shift towards a fully software-based system design provides numerous significant benefits, such as increased interoperability, scalability, and cost-effectiveness, it also brings with it an increased cybersecurity risk. Security is crucial to maintaining trust between vendors, operators, and consumers. Cyberattacks are rapidly increasing in number and sophistication, and we are seeing a shift towards zero-trust approaches. This means that even communications between VNFs inside a 5G core must be scrutinized and hardened against attacks, especially with the advent of quantum computers. The National Institute of Standards and Technology (NIST), over the past 10 years, has led efforts to standardize Post Quantum Cryptography (PQC) to protect against quantum attacks. This paper covers a custom implementation of the open-source free5GC CN, to expand its HTTPS capabilities for VNFs by introducing PQC Key Encapsulation Methods (KEM) for Transport Layer Security (TLS) v1.3. This paper provides the details of this integration with a focus on the latency of different PQC KEMs in initial handshakes between VNFs, on packet size, and the implications on a 5G environment. This work also conducts a security comparison between the PQC-equipped free5GC and other open-source 5G CNs. The presented results indicate a negligible increase in UE connection setup duration and a small increase in connection setup data requirements, strongly indicating that PQC KEM's benefits far outweigh any downsides when integrated into 5G and 6G core services.

**Keywords:** post quantum cryptography; key encapsulation methods; CRYSTALS-kyber; 5G; 6G; core network; open source

## 1. Introduction

5G has revolutionized cellular communications by providing higher connection throughputs, higher User Equipment (UE) density, lower latencies, improved network configurability, the software virtualization of network services, and a resulting shift towards more affordable commodity hardware platforms for 5G operations and deployments [1]. These improvements have also brought with them a rapidly expanding ecosystem of open-source 5G implementations and tools and even allow them to run or experiment with personal or private 5G networks. Currently, available free and open-source 5G tools also include full-stack implementations that provide all the functionality needed for implementing UE, gNodeB (gNB) software implementations using Universal Software Radio Peripherals, and all the required Core Network (CN) software. Some solutions also provide support for Open Radio Access Network (O-RAN), allowing researchers and customers to mix and match various components to meet their needs, with all of these solutions complying with 3rd Generation Partnership Program (3GPP) standards [2].

 doi:10.20944/preprints202410.1802.v1



Within this realm of commercial and open-source solutions around 5G, security is an ever-evolving topic with many focus areas and areas of research interest. This paper, with its focus on security aspects of open-source CNs, is part of this domain. Reviewing the available 5G open-source CN software packages, three notable solutions are at the forefront: OpenAirInterface, Open5GS, and free5GC. All of these powerful tools leverage the standards set in place by 3GPP but have significant differences when looking at the security capabilities they provide to their users. In an escalating climate of cyber hacks and data leaks, security is ever more important [3]. Users on a public or private 5G network rely on the systems in place to safeguard their data. As our review has shown, free5GC has the most robust security measures in place among these three prevalent open-source 5G projects, incorporating TLS-enabled HTTPS requests between its 5G Core's Virtualized Network Functions (VNF).

While this is a great capability considering current security standards, the feasibility, availability, and capability of quantum computers are rising. Quantum computing represents a revolutionary leap in computational power, leveraging the principles of quantum mechanics to perform complex calculations at speeds unattainable by traditional non-quantum computers. However, this advancement poses a significant threat to current cryptographic systems, as quantum computers could potentially break widely used encryption methods by recovering cryptographic keys currently infeasible to be recovered by classical machines.

In cryptography, we consider two primary types of encryption: symmetric and asymmetric encryption. A common example of symmetric encryption, where the same cryptographic key is used for both encryption and decryption, is the Advanced Encryption Standard (AES). Asymmetric encryption utilizes key pairs comprised of a public and a private key that are related to each other. It is primarily used for digital signatures but can also be used for data exchanges. For example, it can be found in key exchange algorithms used with symmetric encryption to ensure the same key is available to both parties. These challenges have been explored by European Telecommunications Standards Institute (ETSI) [4] and others [5,6]. One common example of that role is found in TLS, which, in turn, is the key security mechanism behind HTTPS and other secure communications approaches. In TLS, symmetric encryption using AES is used for the actual data exchange, but asymmetric methods are used for TLS session establishment and the key exchange. Here, it is important to know that while AES-256 is generally considered quantum-safe, there is an increased risk specifically for asymmetric cryptographic methods, which in turn could pose the risk of quantum computing attacks revealing the symmetric keys being exchanged during TLS handshaking and thus allowing an attacker to decrypt all data being exchanged by both parties.

To counter this threat, post-quantum cryptography (PQC) is being developed as a suite of cryptographic algorithms designed to be secure against the capabilities of quantum computers. PQC algorithms are based on mathematical problems that are believed to be resistant to quantum attacks, such as lattice-based, hash-based, and code-based cryptographic methods. Implementing PQC is crucial for future-proofing secure communications and ensuring that sensitive data remains protected as quantum technology evolves. The National Institute of Standards and Technology (NIST)'s efforts in this domain focus on standardizing algorithms to address threats to key encapsulation and digital signature algorithms from quantum computing [6].

PQC is also crucial for securing 5G core networks due to the introduction of VNFs in the 3GPP 5G standards. While 3GPP security standards do not explicitly list support for PQC in their supported TLS guideline [7], expansion of security profiles Subscription Permanent Identifier (SUPI) protection allows the introduction of quantum-resistant protection [8]. VNFs decentralize and abstract network functions into virtualized software services running on standard hardware, increasing flexibility and efficiency but also expanding the attack surface if the communication between all of these software services, which predominantly use HTTP-based REpresentational State Transfer (REST) API calls between themselves, is not properly secured. In a typical perimeter-based cyber defense posture, where we focus on keeping attackers out of the 5G Core Network, any successful intruder that can establish a foothold inside of the 5G core poses a grave threat to any and all 5G core activities and

 doi:10.20944/preprints202410.1802.v1



its users. With current security measures, all 5G CNs are at risk of Store Now Decrypt Later (SNDL) attacks. This is the idea of collecting encrypted data and decrypting it later using, for example, a quantum computer. In the event an attack like this succeeds, sensitive customer connection data will be leaked. In order to protect against these possible attacks, CNs must be equipped with strong Key Encapsulation Methods (KEM).Current 5G CNs that have TLS capabilities, like free5GC, are still vulnerable to SNDL attacks. Specifically, connections between the VNFs within the CN are at risk if this data is ever leaked and must adopt PQC features to mitigate this vulnerability.

The remainder of this paper is organized as follows. Section 2 outlines an overview of our approach, Section 3 provides an overview of related research efforts, Section 4 outlines current open-source 5G cores and compares their implemented security features, and Section 5 gives an overview of our integration of PQC algorithms into the open-source free5GC package. Section 6 will outline the results of our 5G core efforts, including our obtained bandwidth and latency results for connecting UEs to 5G services when different PQC KEM algorithms are being employed, and Section 7 will conclude our work and provide future research directions.

## 2. PQC Protections for 5G Core Networks

Open Quantum Safe (OQS) [9] has assembled open-source packages to address this specific issue, integrating quantum-resistant KEMs into OpenSSL. With the current 5G CNs being susceptible to SNDL attacks, our research, and development of a quantum-safe CN shows that it is absolutely feasible for production systems, with only a small amount of associated overhead. Figure 1 showcases the main systematic components of a 5G network, including the focus of this paper – a quantum-safe 5G CN. Our research is already continuing to expand these security provisions to the communications between the Radio Access Network (RAN) gNodeB base stations and the 5G Core, as well as for end-to-end protections for the user plane traffic, especially when it involves Multi-Access Edge Computing integrated with the 5G infrastructure. These efforts are also shown in Figure 1.

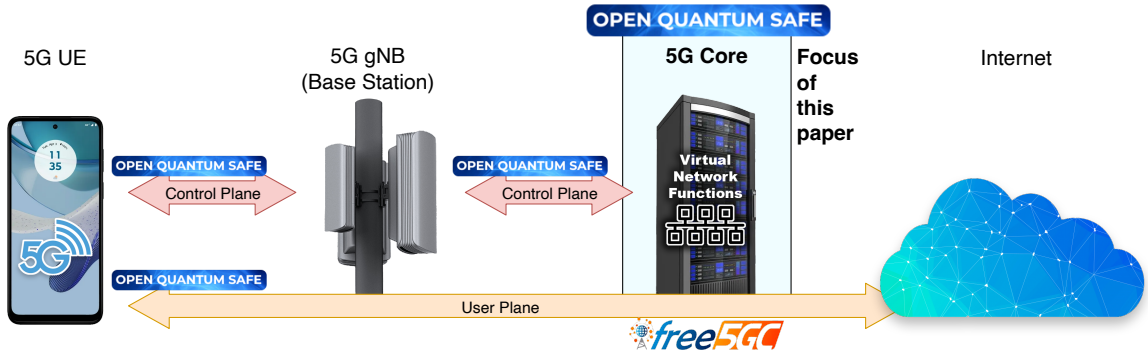


**Figure 1.** 5G System Overview and the Inclusion of Post-Quantum Cryptography for Secure End-to-End and Control Communications.

In Figure 2 below, we are providing a more in-depth look into the 5G core architecture and its set of VNFs, all of which communicate with each other using HTTP REST API calls, which for security would utilize TLS to secure the transport layer. TLS, as mentioned earlier, uses asymmetric cryptography for the session establishment, and it is here that we focused our research in order to provide PQC protections for the TLS key exchange process.

 doi:10.20944/preprints202410.1802.v1



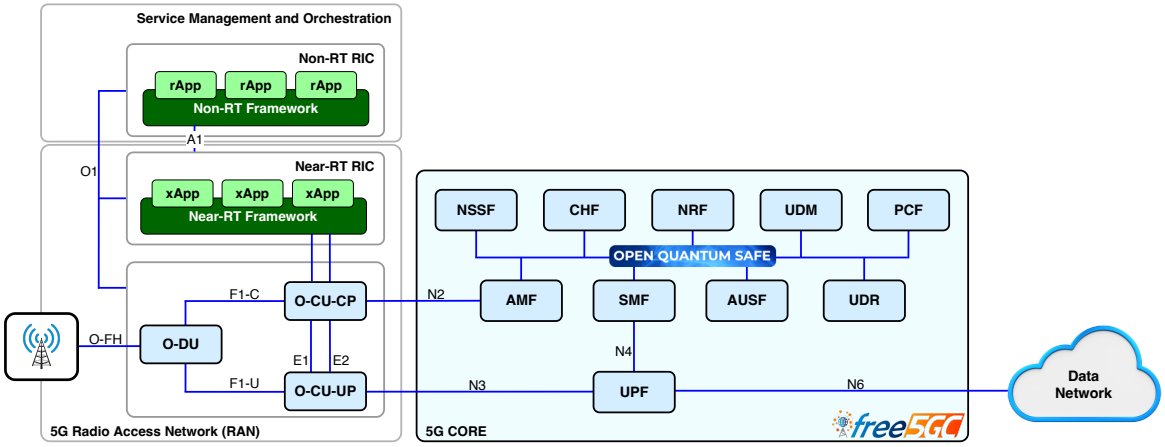


**Figure 2.** Addition and analysis of Post Quantum Cryptography in a 5G Core Network.

This paper introduces an integration of PQC algorithms within an open source 5G core network, free5GC. A detailed analysis of these algorithms and their effect on latency and bandwidth for UE connections to a CN. From the results presented in this paper, there are existing PQC implementations that achieve similar latency to traditional security methods, while having limited effect on a network's utilized bandwidth.

## 3. Related Works

As Quantum Computing (QC) continues to evolve, finds practical applicability, and is more widely available, the threat of current encryption methods becoming vulnerable to quantum algorithms able to determine the encryption key and thus enabling the unauthorized decryption of the protected information grows as well. The discovery of various Quantum Algorithms (QA) has also shown that current TLS is at risk given a sufficiently powerful quantum computer infrastructure. Current cryptography techniques leverage math problems that would be too resource and time-intensive to solve with a typical computer [10]. With a traditional Turing-based computer, the difficulty of factorization of a number grows exponentially with the size of the number. In 1999, Peter Shor developed Shor's Factorization Algorithm – a quantum computing algorithm that can factorize any given number in logarithmic time, potentially reducing a factorization that would take thousands of years down to a few minutes [11]. Similarly, in 1996, Grover's algorithm was discovered, which leverages quantum computing in order to take the problem of finding a random record in N records from O(N) difficulty down to O(log(N)) [12]. Due to their ease of solving the mathematical problems behind current encryption methods, Quantum computers have shown to be relatively effective at circumventing different types of currently employed encryption methods such as Rivest–Shamir–Adleman (RSA), Diffie-Hellman (DH), and Elliptic Curve Cryptography (ECC) [13]. Therefore, an urgent need to create and implement algorithms that cannot be easily solved by either a traditional or quantum computer has emerged.

Much of the current internet infrastructure relies on one or more of these encryption algorithms, and due to the widespread nature of the problem, a need was identified to find quantum-safe cryptographic algorithms. In 2016, the United States NIST called for proposals of alternative Public Key Encryption Schemes, Key Encapsulation Methods, and Digital Signature Schemes, which would be resistant to a quantum attack [14]. The goal of this process is to identify a PQC algorithm best suited for widespread adoption and standardization. In the first round of submissions, 69 total algorithms were submitted. These were then evaluated based on three major characteristics: security, cost and performance, and algorithm implementation characteristics. NIST completed three rounds of evaluation, and encryption methods were improved and resubmitted as the evaluation progressed [15]. As of August 13, 2024, NIST has released its first three finalized encryption standards, which will

 doi:10.20944/preprints202410.1802.v1



significantly reduce the threat posed by quantum attacks on systems that implement PQC approaches. NIST has also stated that it will continue to evaluate alternative encryption methods, but these should be considered backups to the finalized selections [16].

While these algorithms can be applied to regular internet traffic, they are far more advisable for consideration in addressing specific complexities and needs in areas such as 5G security. 5G UEs must first communicate with a base station, referred to as a gNB by 3GPP, which then communicates with a 5G CN. This 5G CN subsequently performs authentication and provides the UE with access to the internet. One of the main features of 5G is that many of the prior generations' hardware-based network operations are now represented by software services and are able to be virtualized [17]. These network functions use HTTP REST API calls to communicate with each other [18]. HTTP communications do not provide any secure encrypted communications. For that, HTTPS was created, facilitating security through Transport Layer Security (TLS) capabilities. Whenever one of the VNFs requires the services of another VNF, it will first need to establish an HTTP/HTTPS connection to that VNF service before exchanging data securely. The capabilities of software virtualization in 5G mean that there are no restrictions about where each VNF could be running. A 5G Core's entire set of VNFs could be deployed on the same physical host or be distributed across a number of physical hosts. But regardless of where they are deployed, they do require network communications to function properly, and for those communications to be secure. One of the goals of the research presented in this paper is thus to strengthen the secure communication between these network functions, ensuring that any information exchanged is quantum-safe.

There is existing research on applying PQC algorithms to the 5G domain. In [19], Figlarz and Hessel apply a PQC signature method to enhance the security of the 5G-AKA key exchange. Hoque, Abdullah, and Zeydan present a PQC key distribution framework for Internet of Things (IoT) networks that utilize 5G networks in [20]. In [21], the authors Minhas and Mansoor propose securing communication between edge-computing devices for e-Health applications in IoT, which often rely on a cellular connection through 5G. One of the key performance indicators of 5G networks is to be able to support multi-access edge computing (MEC), or a form of computing where applications and data are stored at the network edge, greatly reducing latency and improving scalability, compared to a cloud-based service, and allowing for low latency computation to be performed off of constrained devices [22]. Previous works have created an implementation of this service, such as a 2016 report that presents an architecture for MEC and, through their tests, has shown that MEC is capable of applications with a network delay of as little as 17ms [23]. Other works have shown that it is practical and has shown a minimum latency of as little as 10ms in the network [24]. However, there is also existing concern about the speed at which PQC can be utilizes, especially in TLS. In [25], Zheng et al. evaluated PQ-KEM algorithms and found hardware advancements speed up PQC key generation by 3.5 to 4.9 times. In this paper, we expand on the current 5G security research by implementing PQC key exchange for communication between each of the VNFs within the 5G Core network and present the resulting performance impact to latency and bandwidth compared to traditional cryptographic algorithms.

In [26], Zeydan et al. introduce a novel architecture that integrates Quantum Key Distribution (QKD) with Blockchain-based Self-Sovereign Identity (SSI) within the Open RAN (O-RAN) framework for 6G networks, aiming to enhance security and identity management. It explores the practical applications, challenges, and future directions of this integration, emphasizing its potential to transform telecommunication networks into more secure and decentralized systems. The authors of [27] explore the integration of quantum computing, federated learning, and 6G networks to enhance security and privacy in the Internet of Things (IoT) ecosystem, addressing the unique challenges posed by the proliferation of IoT devices. They also provide an overview of recent developments, a conceptual framework for implementing these technologies, and insights into future advancements that could significantly improve IoT security. Nakka et al. [28] address the threat of quantum computing in distributed energy resources (DER) and explore the adoption of PQC within the IEEE 2030.5 standard

 doi:10.20944/preprints202410.1802.v1



for DER networks and presents a real-time hardware-in-the-loop co-simulation testbed to validate the feasibility of implementing PQC in DER systems. Additional work as been conducted evaluating PQC for IP Security (IPSec), which achieved 100 Gbps throughput in a high-speed data center and 0.486 Gbps in a local-to-cloud architecture [29].

## 4. Current Open-Source 5G Core Security

While the focus of this paper is on implementing Post Quantum Cryptography for a custom implementation of free5GC and the resulting performance considerations, we first present our comparative analysis of prominent, currently available open-source implementations of 5G cores in order to compare and contrast what security measures are currently in place for each of them. There are currently five implementations of open-source 5G networks available. Of these five, there are three that have complete core network implementations: Open5GS, free5GC, and Open Air Interface (OAI) [30]. Current published research analyzing the security of their network functions on these three. The study of these three Core Networks (CN) identified and briefly described any existing vulnerabilities. In our own review, outlined in Table 1, it was found that the OAI CN didn't implement TLS at all. Every REST API call between 5G Core VNFs is performed using unencrypted HTTP. Furthermore, it is critical to analyze the projects utilizing TLS to determine vulnerabilities in the TLS communication used throughout the network. The OAI CN was excluded from this review because it did not implement TLS [31].

**Table 1.** Comparison of Open-Source 5G Core Networks (CNs) with TLS and PQC Support.

| Open-Source 5G CNs | TLS | Total Ciphers | Weak Ciphers | Vulnerabilities / Weaknesses | Cipher Ordering | PQC KEMs |
|---|---|---|---|---|---|---|
| OAI | No | N/A | N/A | N/A | N/A | N/A |
| Open5GS | v1.2 - v1.3 | 16 | 10 [31] | None Found | No | No |
| free5GC | v1.0 - v1.3 | 14 | 8 | Sweet32, TLS Version Intolerance [31] | Yes | No |
| Our PQC free5GC | v1.3 | 31 | *TBD* | *TBD* | Yes | Yes |

TLS is recommended by 3GPP to be used for any 5G Core communications to improve security. Hence, we can see that OAI doesn't fully meet 3GPP recommendations, at least in the state it was in during the research [2]. The free5GC project currently implements TLS versions 1.0-1.3, while Open5gs implements versions 1.2 and 1.3. Having these outdated versions of TLS opens free5GC up to a downgrade attack where, if a TLS version cannot be agreed upon, then the version will be downgraded to a less secure one until the two communicating parties both agree on a version. Limiting this to only recent TLS versions decreases the risks of downgrading attacks in Open5gs, giving it an advantage in this regard. In terms of TLS vulnerabilities and weaknesses, the tools used in this research, TLS-Scanner [32], SSLyze [33], and PySSLScan [34], were unable to find any vulnerabilities in Open5GS, while free5GC is vulnerable to the "sweet32" attack and has the "version intolerance" bug.

Additionally, we observed that in both implementations that have TLS, a default key and certificate were used. As stated in the research [31], this could have catastrophic consequences if these cores were ever commercially deployed unmodified, as anyone could retrieve the public key from GitHub and then would be able to decrypt messages being sent between network functions. It was found that free5GC makes use of 14 different ciphers, all of which are ordered by strength. The application then uses the most complex first, going down the list if the strongest cannot be used. In research, it has been found that 8 of these 14 cipher suites are weak, with some being broken as far back as 2017 [31]. Open5GS also uses weak cipher suites, with 10 of the 16 ciphers used being considered weak. Additionally, Open5GS does not have cipher ordering, meaning there are no guarantees that the strongest available cipher will be used in communication. Open5GS and OAI can also benefit from the addition of PQC. OAI would first need to implement TLS capabilities, and since OAI is written in the C language, direct access to OpenSSL at the system level can be integrated. Like OAI, Open5GS could also benefit from the integration of PQC, utilizing a similar migration process as suggested for OAI.

 doi:10.20944/preprints202410.1802.v1



However, there are some challenges and risks associated with integrating PQC algorithms into projects such as these, as PQC implementations are inherently unproven and can have potentially negative security ramifications if implemented incorrectly. While this is a concern in general, several projects already use the OQS implemented we selected successfully, specifically to mitigate this risk. Additionally, there is no straightforward approach for empirically evaluating the security improvements provided by such PQC implementations, which we consider out of the scope of this paper. It is also possible for malicious parties to subvert open-source third-party packages in what is referred to as a software supply chain attack, in order to introduce vulnerabilities then propagating to dependent packages. While not detailed in this paper, we performed due diligence and reviewed all utilized packages for any reported vulnerabilities.

There are also computational trade-offs for utilizing PQC in a system. Additional processing for encryption and decryption, higher bandwidth utilization, and increased latency are all potential concerns when utilizing PQC instead of more traditional cryptographic approaches. During our testing, no noticeable increase of CPU usage was observed within the system. Thus, the primary focus of our study is on the implication of our PQC integration on the utilized bandwidth and latency, our study of which is presented in Section 6 of this paper.

While the aim of this research is on current implementations of 5G CNs, research towards 6G networks identifies additional capabilities. Current research suggests 6G implementations will include Holographic MIMO, which leverages large intelligent surfaces (LIS) [35], the implementation of AI/ML for radio access [36], AI/ML for more efficient receivers [37], and Reconfigurable Intelligent Surfaces (RIS) for improved energy efficiency and adaptive beam-forming [38]. Core network improvements will focus on Quality of Service improvements and may have lower latency and bandwidth tolerances than presented in this paper [36]. MEC is also predicted to experience increased adoption with the introduction of 6G standards, which often have different requirements for latency and processing time. However, there are concerns with adding PQC methods to Fog and Cloud Computing applications, as key size, encryption processing time, and data overhead may increase latency [39].

## 5. Methodology

As NIST continues to standardize PQC, the expectation is that this technology will experience a rapid acceleration of its integration into a wide variety of existing and future software and infrastructure projects in an effort to maintain a high level of security and privacy protections. To aid in this effort, the open-source community started a new project called "Open Quantum Safe" (OQS). OQS' efforts focus on creating and maintaining a free implementation of PQC algorithms in the form of "liboqs", as well as integrations of this capability into common providers and architectures, such as OpenSSL [40], BoringSSL [41], and more. Specifically, enabling the OQS extension for OpenSSL allows for a more straightforward approach to testing and research of these algorithms in lab and production systems before, during, and after their standardization by NIST. The specific repository used in this research, oqs-provider, lists its included PQC KEM and Signature algorithms. A list of these KEM and Signature Algorithm groups is shown in Table 2.

**Table 2.** KEM and Signature Algorithm Groups.

| Category | Main Groups |
|---|---|
| **KEM Algorithms** | BIKE, CRYSTALS-Kyber, FrodoKEM, HQC, ML-KEM |
| **Signature Algorithms** | CRYSTALS-Dilithium, ML-DSA, Falcon, SPHINCS-SHA2, SPHINCS-SHAKE, MAYO |

Specifically, the main focus of testing is put on CRYSTALS-Kyber, as it is the main KEM that is being considered for standardization by NIST. The full list of KYBER PQC and Hybrid (a combination of traditional methods with PQC) KEMs included in oqs-provider is detailed in Table 3.

 doi:10.20944/preprints202410.1802.v1



**Table 3.** CRYSTALS-Kyber Algorithms.

| KEM Algorithm | Subgroups |
|---|---|
| **CRYSTALS-Kyber** | kyber512, p256_kyber512, x25519_kyber512, kyber768, p384_kyber768, x448_kyber768, x25519_kyber768, p256_kyber768, kyber1024, p521_kyber1024 |

The full incorporation of PQC into a 5G CN does require careful consideration of understanding the components and interfaces that require changes. If there are any missed in the process, essential CN functions will no longer work, causing fatal system errors.

The process outlined in Figure 3 details the high-level steps taken to add PQC into a 5G CN. This methodology can also be used as a roadmap to incorporate PQC into other 5G systems, as the underlying structure of different CNs is comparable in accordance with 3GPP standards.

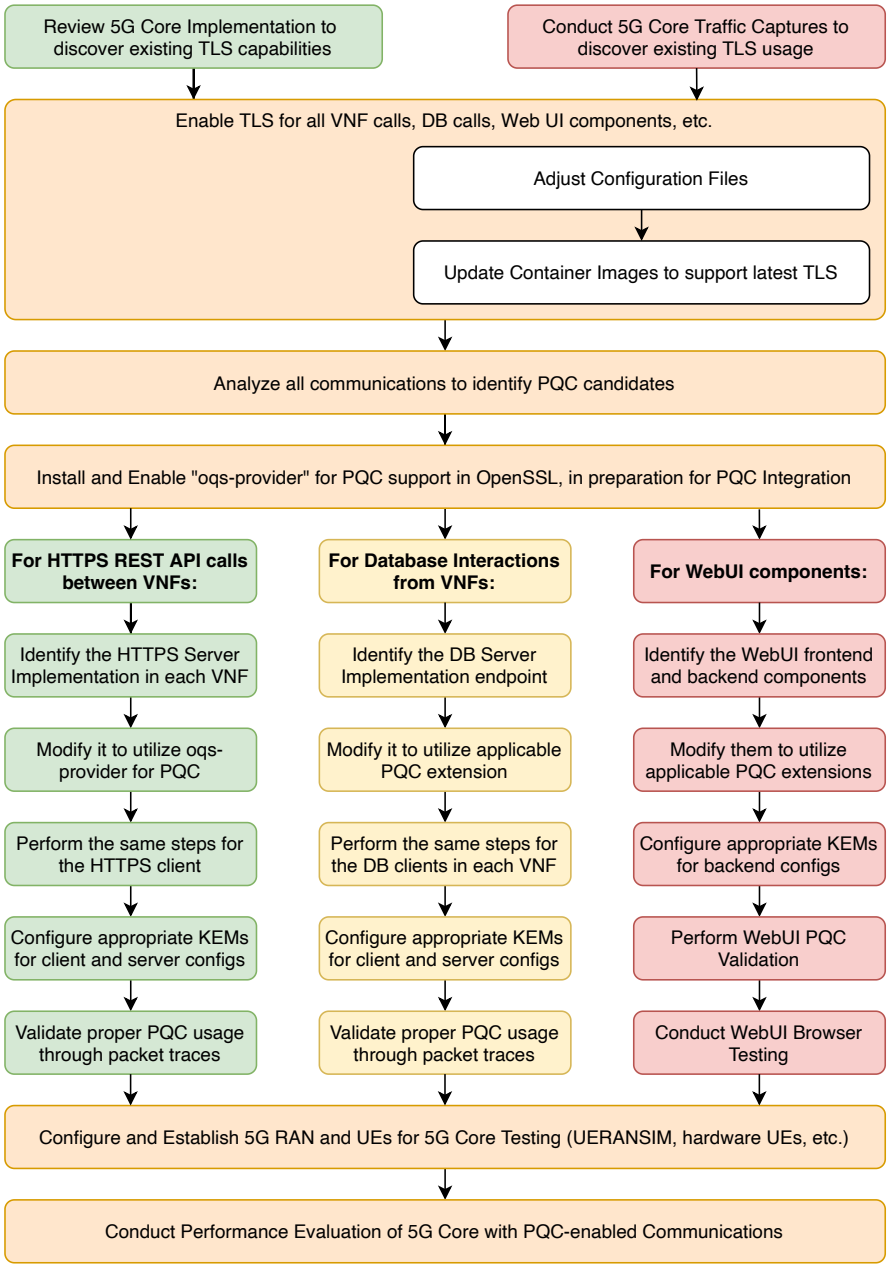


**Figure 3.** Steps taken in the discovery, integration, validation, and analysis of PQC in 5G Core Networks.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 October 2024 doi:10.20944/preprints202410.1802.v1



In order to utilize oqs-provider, it needs to be compiled from source, which ultimately generates the necessary shared library file that OpenSSL utilizes to incorporate the PQC KEMs and Signature algorithms. In Listing 1 in Appendix A, we are demonstrating the instructions for this procedure that we have incorporated inside each of the Dockerfiles associated with all the NFs included in free5GC.

To fully enable the features provided by this extension in a variety of environments, OQS provides detailed instructions in their documentation as well. For the purpose of this research, the CRYSTALS-Kyber KEMs were activated at the system level by adding their values to the 'openssl.cnf' configuration file. Figure 4 shows the lines that were added or updated in order to enable OQS.

```
55  ssl_conf = ssl_sect
56
57  # List of providers to load
58  [provider_sect]
59  default = default_sect
60  oqsprovider = oqsprovider_sect
61

74  [default_sect]
75  activate = 1
76
77  [oqsprovider_sect]
78  activate = 1
79
80
81  ####################################################################
82  [ ca ]
83  default_ca  = CA_default        # The default ca section

398 [ssl_sect]
399 system_default = system_default_sect
400
401 [system_default_sect]
402 CipherString = DEFAULT:@SECLEVEL=2
403 Groups = kyber1024
404
```

**Figure 4.** Openssl.cnf with the edited files highlighted to enable OQS.

The free5GC project is a free and open-source 5G CN that is currently compliant with 3GPP Release 15, with ongoing development aiming at compliance with 3GPP Release 17 [42]. In our research, we cloned the original free5GC repositories and subsequently modified them to incorporate PQC capabilities using oqs-provider. Free5GC has a "docker-compose" deployment method that allows for seamless integration and use with systems that support Docker. Figure 5 illustrates the Docker images that are running after updating the free5GC codebase to allow PQC.

```
tel-mfoss-core@telmfosscore:~/free5gc-compose$ docker ps
CONTAINER ID   IMAGE                          COMMAND                  CREATED          STATUS          PORTS                                                  NAMES
3fd315421393   free5gc-compose-free5gc-n3iwf  "sh -c './n3iwf-ipse…"   25 seconds ago   Up 23 seconds                                                          n3iwf
bdf60e32637a   free5gc-compose-free5gc-chf    "./chf -c ./config/c…"   25 seconds ago   Up 23 seconds   2122/tcp, 8000/tcp                                     chf
d68f6dd75aa8   free5gc-compose-free5gc-pcf    "./pcf -c ./config/p…"   25 seconds ago   Up 24 seconds   8000/tcp                                               pcf
6d3bac1d3324   free5gc-compose-free5gc-udm    "./udm -c ./config/u…"   25 seconds ago   Up 24 seconds   8000/tcp                                               udm
6211d78496e5   free5gc-compose-free5gc-amf    "./amf -c ./config/a…"   25 seconds ago   Up 24 seconds   8000/tcp                                               amf
cfe547e0683d   free5gc-compose-free5gc-udr    "./udr -c ./config/u…"   25 seconds ago   Up 24 seconds   8000/tcp                                               udr
8c6d72eb61c4   free5gc-compose-free5gc-webui  "./webui -c ./config…"   25 seconds ago   Up 24 seconds   2121/tcp, 0.0.0.0:5000->5000/tcp, :::5000->5000/tcp    webui
f328745ce0ec   free5gc-compose-free5gc-smf    "./smf -c ./config/s…"   25 seconds ago   Up 24 seconds   8000/tcp                                               smf
e9fad4753faf   free5gc-compose-free5gc-ausf   "./ausf -c ./config/…"   25 seconds ago   Up 24 seconds   8000/tcp                                               ausf
8af0c1ff8ab3   free5gc-compose-free5gc-nssf   "./nssf -c ./config/…"   25 seconds ago   Up 24 seconds   8000/tcp                                               nssf
4e6d358fdf33   free5gc-compose-free5gc-nrf    "./nrf -c ./config/n…"   25 seconds ago   Up 24 seconds   8000/tcp                                               nrf
461ba83b2d3b   pqc-mongo                      "docker-entrypoint.s…"   25 seconds ago   Up 25 seconds   27017/tcp                                              mongodb
f2a75e2ac510   free5gc-compose-free5gc-upf    "bash -c './upf-ipta…"   25 seconds ago   Up 25 seconds                                                          upf
```

**Figure 5.** Essential PQC free5GC CN VNFs up during runtime.

In our custom implementation of free5GC, each VNF container is running a lightweight version of Ubuntu 22.04 Jammy with OpenSSL v3.0.2 installed. After compiling oqs-provider from source

 doi:10.20944/preprints202410.1802.v1



for the respective platform, the resulting shared library file is installed in each VNF Docker container, enabling PQC integration at a system level.

Additionally, some modifications to the source of free5GC are necessary in order to utilize OpenSSL. Virtually all aspects of free5GC are written in GoLang [43], which lacks direct native support for using OpenSSL. A Go program traditionally uses Go's own TLS and Crypto libraries to handle TLS communications [44]. To address this challenge, we selected the open-source Go-Openssl wrapper from 'pexip' [45] and incorporated it within each VNF to enable the use of OpenSSL for cryptography within free5GC rather than Go's crypto library. At the time of publication of this paper, the custom CN implementation is built using free5GC v3.4.2. The integration of Go-Openssl focuses on two main areas within each VNF in free5GC's source code: server.go and the OpenAPI system. Within server.go, the network function is typically utilized to create a TCP listener, which subsequently utilizes the Go TLS to secure the HTTP connection on the server side. Listing 2 in Appendix A shows a code snippet of the two functions that are amended within the code base, in this specific case for 5G's the "Access and Mobility Management Function" (AMF) VNF. In our PQC-enabled implementation, if encryption (https) is enabled in free5GC's YAML configuration file, then an OpenSSL-enabled listener is created within the 'NewServer' function. The newly created server uses the OpenSSL context that is stored in the server struct from the logic in the NewServer function. The built-in functions are used to obtain the path to the certificates used for context creation, and certificate verification is set to "none" for testing purposes.

Next, our focus was on enabling each of the HTTPS clients used throughout free5GC for API calls to similarly be enabled for OpenSSL usage. This process was not as straightforward as the corresponding server-side effort. Free5GC utilizes an OpenAPI-driven approach to API implementation and handling of client HTTP requests. Analyzing the call flow across multiple different files that include client attributes revealed that, ultimately, they all utilize a call to the CallAPI() function within the free5GC OpenAPI repository, which is located in the client.go file. In this function, an OpenSSL context is created using two specific functions: CreateOpenSSLClientCtx() and GetCurrentNetworkFunctionUsingAPI(). The OpenSSL context is managed in the same manner as the server. However, in the case of a client, there are no built-in functions to obtain the path to the client certificates. Client certificates are not strictly required for TLS connectivity. However, they are needed when mutual TLS authentication (mTLS) is desired, which can provide a significant improvement in security between both client and server. Thus, GetCurrentNetworkFunctionUsingAPI() obtains the specific network function that is making a request by parsing the stack trace. Returned from this function is effectively the name of the VNF making the HTTP request, which is appended to the path where the certificates are stored inside the Docker containers. In either scenario, if the configuration includes a custom HTTPClient setup requirement, a custom client transport method that includes OpenSSL support is added. The logic sequence for this process is shown in CreateOpenSSLInnerClientTransport(). The dialTLS function that handles the transport protocols for an HTTP client in Go is overridden to use the OpenSSL dial wrapper. The request is then made using the OpenSSL-enabled dialTLS() function during the invocation of the Do() method. This process is shown in the code Listing 3 contained in Appendix A.

One major goal of this customization was to make it as seamless as possible for a free5GC administrator to install and operate it with support for PQC. With the extensible nature of Docker, this goal was met while maintaining the same basic structure for downloading and installing our PQC-enabled free5GC build. The user only needs to run one additional command to install MongoDB from source, which, for convenience, has been incorporated into a single shell script. The steps to clone, build, install, and run PQC-enabled free5GC that are used by this script are also shown in Appendix A in Listing 4.

Once we established a full integration of PQC into all HTTPS REST API endpoints (server code) and all corresponding client calls to these APIs, we could proceed to collect results that demonstrate

 doi:10.20944/preprints202410.1802.v1



the feasibility of this integration of PQC into free5GC operations and also quantify the impact on latency and bandwidth usage, which are shown and discussed in the next section.

## 6. Results

The results from this study aim to evaluate the impact of using standard Elliptic Curve Cryptography (ECC), PQC, and Hybrid PQC KEMs within the communications of the VNFs in a 5G CN using free5GC. All of the CRYSTALS-Kyber KEMs displayed in Table 3 are included in this evaluation, along with the classical ECC KEM x25519. Furthermore, these different methods are compared to a CN that does not use TLS to secure the 5G CN API calls and instead uses unencrypted HTTP calls. This acts as the baseline for our tests. All of them were conducted on a Dell Server equipped with an AMD Epyc 7313p 16-Core, 32-Thread CPU, running the aforementioned custom implementation of free5GC. Included in this test environment is another computer, which is responsible for running UERANSIM, a tool provided by the free5GC package to simulate a gNB as well as multiple UE connections in accordance with 3GPP standards. Any performance metrics of the UERANSIM host machine are inconsequential to this evaluation since all the signaling requests and communications activities are measured relative to the TLS activity within the CN machine. Both machines are running Ubuntu 22.04 LTS.

### *6.1. Testing Scenarios*

Each test is conducted with 100 total UE connections in order to maintain statistically relevant, consistent, and comparable results. Each UE is scheduled with a seven-second cadence between connections, allowing the CN sufficient time to complete each client attachment and to be able to adequately capture and measure each TLS handshake. The data gathered from these studies represents the varying total UE connection times between the different KEMs available in this system. Prior hypotheses were that the PQC KEMs would result in a significant additional latency during a UE connection, but as our results clearly show, this is not the case. In our testing, all connections use a consistent cipher, AES_256_GCM_SHA384, to maintain high security and uniform data communication latency results. Due to OpenSSL typically only accepting signed certificates from a trusted Certificate Authority (CA), all tests are conducted with OpenSSL set to skip certificate verification. This is due to the fact that free5GC includes self-signed certificates. However, it is highly recommended for production systems to use certificates that are signed by a trusted CA. The use of certificates from a CA could increase the latency of the UE connections, due to the small additional time required for verification, but this additional time would be true for any KEM that is used – traditional or PQC.

Wireshark is utilized to capture packets within the CN, displaying the communications between the VNFs. Data from these packets can be extrapolated to further understand connection times, bytes sent, etc. Prior to each test, one KEM is enabled in each of the VNFs openssl.cnf file, ensuring proper control over the test environment. The KEM that is under test is also validated in the final Wireshark capture, which can be seen during the Client or Server Hello message under the 'key_share' extension field. Then, the CN is brought up, and the gNB connection is established. Once all of these processes are complete, Wireshark begins capturing packets on the free5GC-br network interface. Once Wireshark begins capturing, the UERANSIM program is executed, with inputs configured for 100 total UE connections and a seven-second cadence between each subsequent connection. Each UE connects in its own time interval in order to gain a large data set to understand the average UE connection time with each respective KEM active in the CN, and other information which is analyzed in the next section.

### *6.2. Data Analysis*

One important test we conducted focuses on the measurement and evaluation of the total client connection setup duration, which represents the time from the UE initially requesting a connection via the gNodeB until the UE can send user data through the 5G connection. Figure 6 shows a box

 doi:10.20944/preprints202410.1802.v1



plot of the different KEMs and their respective UE connection times we obtained. We observe that the baseline connection time when using no encryption (plain-text HTTP requests) is slightly less than 0.5 seconds. This is expected due to the fact that there are significantly fewer computations required when we are not encrypting and decrypting each communication request.

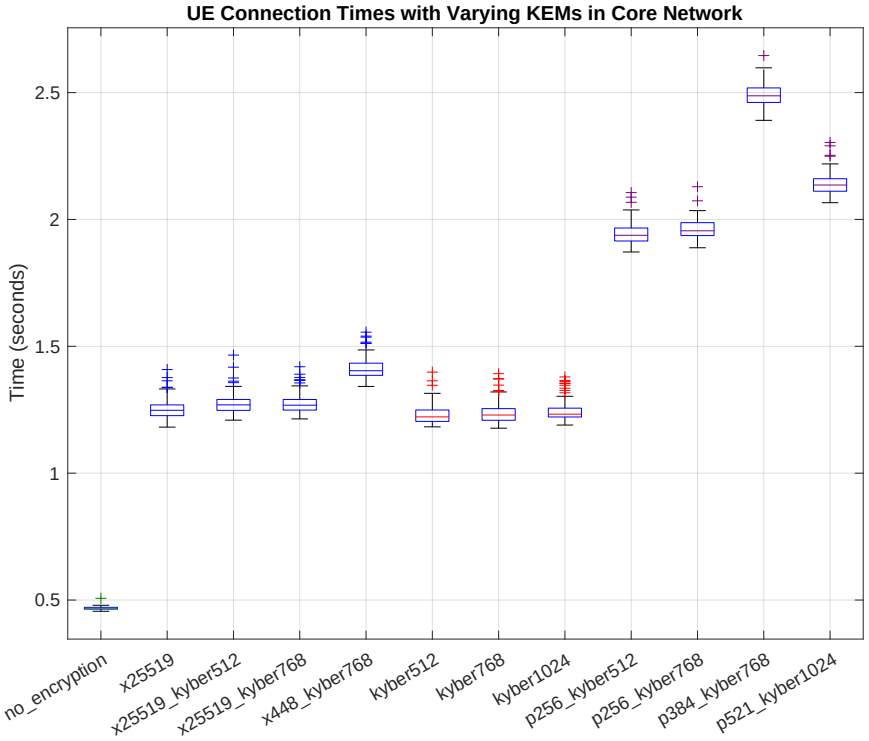


**Figure 6.** UE Connection times measured with no encryption, x25519, PQC and Hybrid PQC KEMs in the CN.

It is important to stress, however, that we conducted this analysis solely for baseline purposes. In no way do we recommend implementing or operating a 5G Core Network using unencrypted communications, as this would pose a significant and unnecessary risk to user data and control data. As a secondary baseline, therefore, we are utilizing the $\times$25519 KEM. This method is a commonly used ECC KEM known for its high performance and adequate security, utilizing a key size of 256 bits. It can be seen that the hybrid x25519 algorithms are nearly identical in total connection time compared to the regular Kyber KEMs, in terms of additional latency. The P256, P384, and P521 hybrid variants, however, do add significantly more latency to the handshake and total connection process times.

Figure 7 outlines box plots that represent the amount of data, in Megabytes, transmitted in the CN during an attachment of a UE. Showing a similar outcome to the results in Figure 6, the tests conducted and shown here for unencrypted communications and the x25519 KEM once again show the lowest amount of bytes exchanged. With the understanding that encryption by nature adds more data to a packet, along with more packets overall being required due to the addition of the TLS handshakes, it is no surprise that there is significantly less data transferred during the unencrypted baseline. Likewise, x25519 uses a 32-byte key, which is the smallest key size compared to other PQC and Hybrid PQC KEMs. It is obvious from these results that with this increase in key sizes we observe an increase in required data being exchanged. For example, kyber512 has a median data volume of 1.12 MB, and kyber1024, with double the public key and cipher text size as seen in Table 4, is responsible for an increase by nearly 200 KB of data that is sent during the UE connection process.

 doi:10.20944/preprints202410.1802.v1



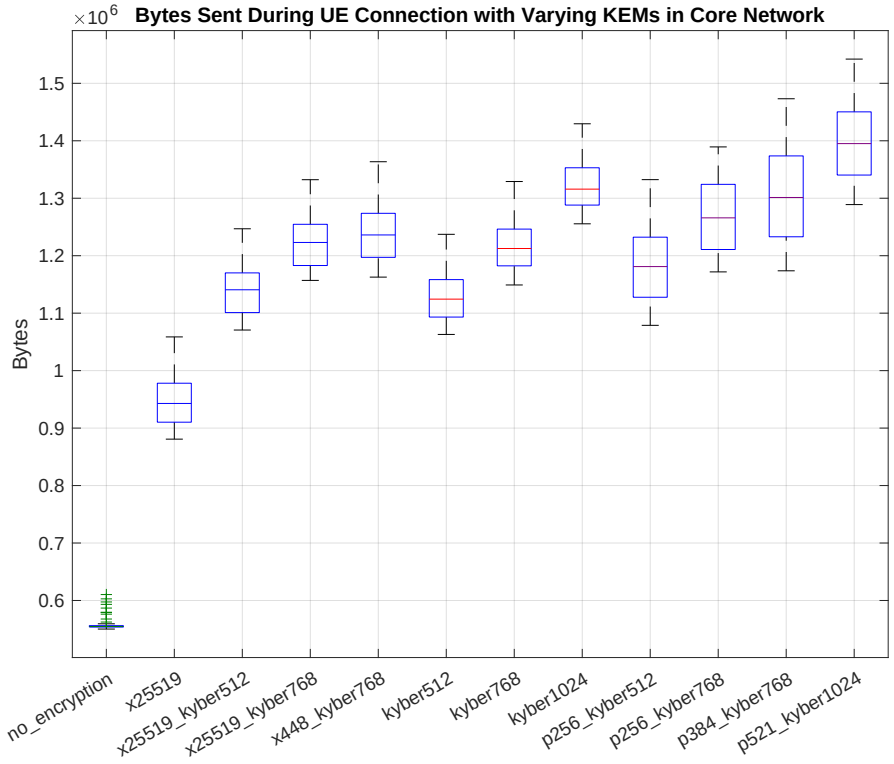


**Figure 7.** Measurement of Megabytes of data transmitted in the CN during a UE connection while varying the KEM used in TLS 1.3 within the CN.

**Table 4.** Key sizes and cycle counts for different versions of Kyber on Haswell CPUs [46].

| **Kyber-512** | **Sizes (in bytes)** | **Haswell cycles (ref)** | **Haswell cycles (avx2)** |
|---|---|---|---|
| sk: | 1632 | gen: 122,684 | gen: 33,856 |
| pk: | 800 | enc: 145,424 | enc: 45,200 |
| ct: | 768 | dec: 187,960 | dec: 34,572 |
| **Kyber-768** | **Sizes (in bytes)** | **Haswell cycles (ref)** | **Haswell cycles (avx2)** |
| sk: | 2400 | gen: 199,408 | gen: 52,732 |
| pk: | 1184 | enc: 235,260 | enc: 67,624 |
| ct: | 1088 | dec: 274,900 | dec: 53,156 |
| **Kyber-1024** | **Sizes (in bytes)** | **Haswell cycles (ref)** | **Haswell cycles (avx2)** |
| sk: | 3168 | gen: 307,148 | gen: 73,544 |
| pk: | 1568 | enc: 346,648 | enc: 97,324 |
| ct: | 1568 | dec: 396,584 | dec: 79,128 |

**Legend:** *sk* - Secret Key, *pk* - Public Key, *ct* - Ciphertext.

Further correlation between larger key sizes and bytes exchanged can also be observed throughout the dataset collected for the Hybrid PQC KEM results. We hypothesize that the observed variance in the data is due to ongoing internal CN communication that can vary from time to time, thus creating some tests that record a differing amount of bytes during a UE connection operation. Hybrid PQC KEMs are specifically important in the role of quantum-resistant algorithms. In a blog post from Google engineers, the authors state the two main rationale points for using Hybrid KEMs is 'Mitigating risks associated with new algorithms' and 'A safe transition' to PQC [47]. Hybrid KEMs provide the

 doi:10.20944/preprints202410.1802.v1



security of PQC combined with the trust afforded to ECC. Thus, many systems are expected to adopt the hybrid approach, and the data presented on the network performance of Hybrid PQC KEMs in 5G is promising.

Figures 8 and 9 illustrate the relative differences in time between the different KEM algorithms used during UE connection operations in relation to the unencrypted operation and to the ECC x25519 KEM, respectively. For example, Figure 9 takes the average UE connection time of each tested PQC and Hybrid PQC KEM and subtracts the average UE connection time of the standard ECC x25519 KEM. A similar approach is used in Figure 8, subtracting the connection time we observed for unencrypted 5G core operations from the obtained PQC and Hybrid PQC results. The previously mentioned trends are extrapolated here to give a clearer view of the relative time differences. We would also like to specifically point out the performance shown for p384_kyber768, as it is nearly 50% slower than its counterparts. Our hypothesis is that the cause of this outcome is the lack of optimization capability for the specific P-384 algorithm in software, combined with additional computational complexities when working with that specific KEM. Figure 9 notes the time differences compared to the classical ECC x25519 KEM. Interestingly, all of the standalone Kyber KEMs performed better than the baseline ECC x25519 KEM. This could be due to heightened optimizations within the OQS OpenSSL library, along with using an AVX2-enabled machine.

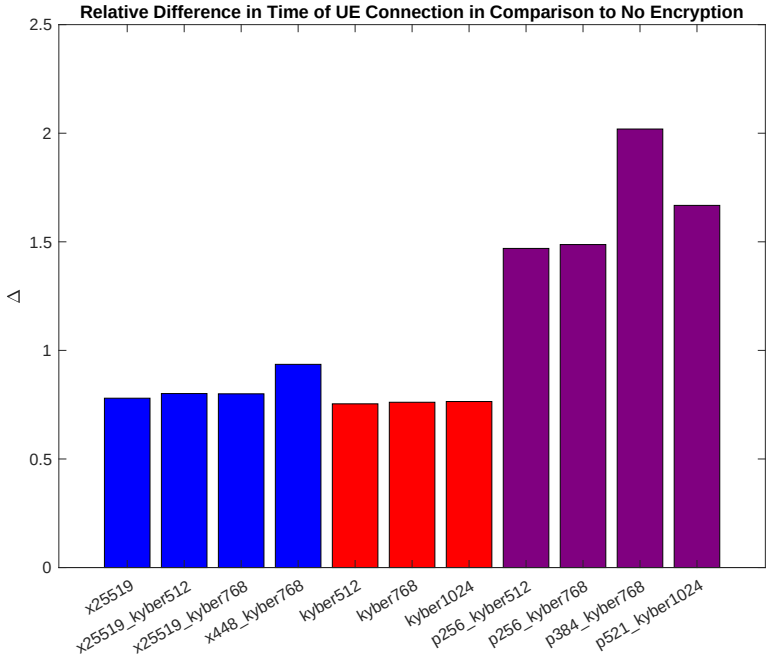


**Figure 8.** Difference in unit time of the KEM latency compared to no encryption.

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 24 October 2024 doi:10.20944/preprints202410.1802.v1



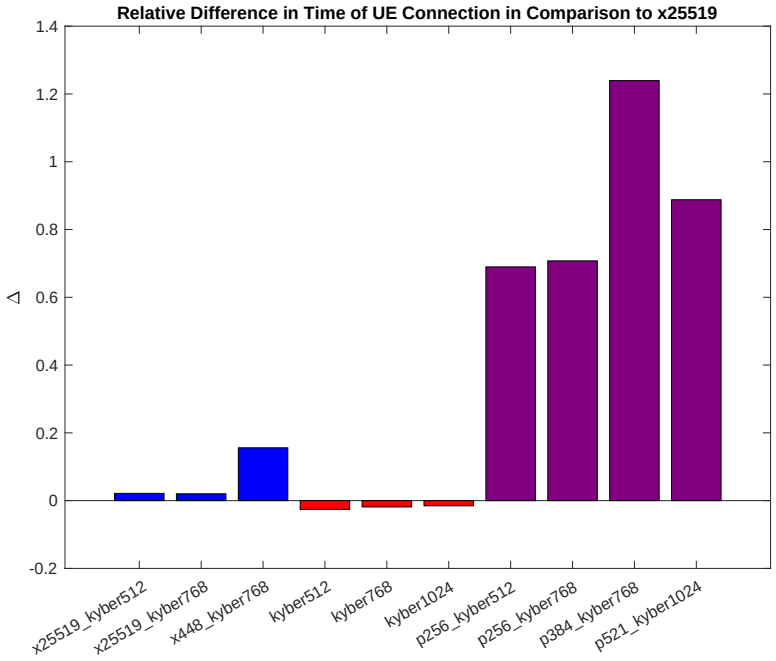


**Figure 9.** Difference in unit time of the KEM latency compared to x25519.

In order to better understand the complexity of a UE connection operation, we wanted to show the entire process and all API calls required to complete this process. Thus, in Figure A1, shown in Appendix A, we are showing a detailed call sequence for one entire UE connection sequence, observing a total of 129 TLSv1.3 handshakes. Contained in this capture is each control signal sent or received by the gNB, along with a condensed version of the TLS client hello messages. Column 'Qty' details how many duplicate consecutive packets were sent. For example, during the first TLS handshakes of the UE connection, the AMF sends 3 separate requests to the NRF. This figure is helpful in that it quantifies how many TLS handshakes occur during an attachment and also provides more insight into the duration of each handshake.

Further information on the time it takes to complete a transaction is illustrated in Figure 10. The differences in handshake time between p521_kyber1024 and plain kyber1024 demonstrate that the additional latency shown in Figure 9 is indeed attributable solely to the TLSv1.3 handshake process, which includes the KEM.

p521_kyber1024 TLSv1.3 Handshake

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | AMF | NRF | TLSv1.3 | 2026 | Client Hello (SNI=nrf.free5gc.org) |
| 2 | 0.000020930 | NRF | AMF | TCP | 66 | irdmi(8000) > 50264 [ACK] Seq=1 Ack=1961 Win=501 Len=0 TSval=1634323114 TSecr=603203809 |
| 3 | 0.003669746 | NRF | AMF | TLSv1.3 | 3164 | Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data |
| 4 | 0.003691688 | AMF | NRF | TCP | 66 | 50264 > irdmi(8000) [ACK] Seq=1961 Ack=3099 Win=497 Len=0 TSval=603203812 TSecr=1634323117 |
| 5 | 0.008226219 | AMF | NRF | TLSv1.3 | 146 | Change Cipher Spec, Application Data |
| 6 | 0.008327794 | AMF | NRF | TLSv1.3 | 429 | Application Data |

kyber1024 TLSv1.3 Handshake

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000000000 | AMF | NRF | TLSv1.3 | 1893 | Client Hello (SNI=nrf.free5gc.org) |
| 2 | 0.000016713 | NRF | AMF | TCP | 66 | irdmi(8000) > 56616 [ACK] Seq=1 Ack=1828 Win=501 Len=0 TSval=1541772042 TSecr=510652737 |
| 3 | 0.000952064 | NRF | AMF | TLSv1.3 | 3031 | Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data, Application Data |
| 4 | 0.000967093 | AMF | NRF | TCP | 66 | 56616 > irdmi(8000) [ACK] Seq=1828 Ack=2966 Win=497 Len=0 TSval=510652738 TSecr=1541772043 |
| 5 | 0.001857607 | AMF | NRF | TLSv1.3 | 146 | Change Cipher Spec, Application Data |
| 6 | 0.001940036 | AMF | NRF | TLSv1.3 | 429 | Application Data |

**Figure 10.** Example packet captures with relative timestamps of TLS handshakes using p521_kyber1024 (top) and plain kyber1024 (bottom)

 doi:10.20944/preprints202410.1802.v1



Burst connections of multiple UEs simultaneously are part of our future work, where the focus is on the scalability of both the 5G Core provided by free5Gc and also of the 5G Core VNFs' communications using PQC KEMs. With the understanding that Massive Machine-Type Communications (mMTC) could potentially need to support 1 million devices per $km^2$, situations in which devices are connecting and disconnecting quickly could have a real impact on performance in the CN with PQC enabled [48]. Kyber1024 has nearly 400KB or 3.2Mb more data transferred during a single UE connection, and in the extreme case of 1 million devices connecting, 400GB or 3.2Tb of data would be transferred in this scenario. Therefore, the systems employed for CN operations would need to withstand these connection bursts, with adequate load balancing and compute resources. Scalability is of particular focus also in commercial deployments, where the concern is not about adequate resource availability but about the cost for those resources, thereby driving the need for efficiency.

## 7. Conclusions

5G continues its worldwide rollout, allowing millions of people worldwide access to high-speed cellular service. At the same time, the need for customer data privacy protections and strong operational cybersecurity is rapidly increasing due to the growing sophistication and reach of cyberattacks in recent years. As such, we need to protect all aspects of today's technology systems, especially 5G cellular services and their importance to our modern information society. We also need to be looking ahead at the cybersecurity requirements for the upcoming 6G and beyond. Future work in this area can expand upon the tests presented in this paper, with further testing using physical UEs, different channel bandwidths, testing with commercial grade equipment, and expanding PQC to end-to-end communication channels. Utilizing physical UEs with differing channel bandwidths may provide interesting insight into the device implementation differences between different devices, which may be resource-constrained.

With the rising possibility of usable quantum computers in the near future and the threats of SNDL attacks, 5G and 6G must adapt PQC in order to keep customer data quantum-safe. The research we present in this paper demonstrates that the modifications we conducted to integrate PQC into free5GC resulted in only a negligible latency impact and a minimal increase in data transfer volume during the UE connection phase to the 5G network from the inclusion of certain PQC or Hybrid PQC KEMs. These positive test outcomes were collected from capturing and analyzing a large number of 5G Core Network communications utilizing TLS v1.3 with PQC KEMs during numerous UE attachments using a variety of different KEMs. The results indicate that standalone Kyber KEMs can actually outperform the traditional ECC x25519 KEM in terms of the UE client connection duration tests. Thus, we can conclude that without any UE latency impact, we can achieve a significantly strengthened cybersecurity posture for the entire 5G Core Network. With these modifications, every API call between VNFs is not only AES-256 protected, which is currently considered quantum-safe, but we also achieved strong protections for the key exchange during the TLS handshaking for quantum safety. Thus, based on our tests, we anticipate similarly encouraging results regarding the latency and bandwidth impact of introducing PQC into TLS connections for all control plane and user plane operations in 5G and beyond. Our future work encompasses significant steps in that direction, with efforts already underway to secure RAN communications, Multi-Access Edge Communications in 5G, and end-to-end user data protections for 5G UEs.

**Author Contributions:** Investigation, P.S., R.G., M.B., M.H. and H.S.; writing—original draft preparation, P.S., R.G., M.B. and M.H.; writing—review and editing, P.S., R.G., M.B., M.H. and H.S.; supervision, H.S. and M.H.; project administration, H.S. and M.H.; funding acquisition, H.S. and M.H. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research has been supported by the University of Nebraska-Lincoln's Advanced Telecommunications Engineering Laboratory (TEL) and Belcan Government Solutions (BGS) under grant# TEL-Belcan-5G23.

 doi:10.20944/preprints202410.1802.v1



**Data Availability Statement:** The original data presented in the study are openly available in: https://git.unl.edu/tel-papers/2024_pqc_for_securing_5gcn_mdpi.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Appendix A

```
RUN apt update && apt upgrade -y
RUN apt install -y git gcc cmake ninja-build openssl libssl-dev
RUN git clone https://github.com/open-quantum-safe/oqs-provider.git
RUN cd oqs-provider
WORKDIR oqs-provider
RUN sh ./scripts/fullbuild.sh
RUN cmake --install _build
WORKDIR /free5gc
```

Listing 1: Dockerfile commands for building free5GC containers with OQS.

```
import "github.com/pexip/go-openssl"
...
func NewServer(amf ServerAmf, tlsKeyLogPath string) (*Server, error){
...
  s.tlsCtx, err = openssl.NewCtxFromFiles(cfg.GetCertPemPath(), cfg.GetCertKeyPath())
  if err != nil {
    logger.InitLog.Errorf("Initialize OpenSSL context failed: %v", err)
    return nil, err
  }
  s.tlsCtx.SetVerify(openssl.VerifyNone, nil)
...
}
...
func (s *Server) startServer(wg *sync.WaitGroup){
...
    } else if scheme == "https" {
      listener, err := openssl.Listen("tcp", cfg.GetSbiBindingAddr(), s.tlsCtx)
      if err != nil {
        logger.SBILog.Errorf("Failed to create OpenSSL listener: %v", err)
        return
      }
      err = s.httpServer.Serve(listener)
    } else {
      err = fmt.Errorf("no support this scheme[%s]", scheme)
    }
...
}
```

Listing 2: Creation of HTTP server using Go-OpenSSL Wrapper.

```
import "github.com/pexip/go-openssl"
...
var (
  innerOpenSSLClient    *http.Client
  innerOpenSSLClientCtx *openssl.Ctx
)
...
func GetCurrentNetworkFunctionUsingAPI() string {
  var buf bytes.Buffer
  buf.Write(debug.Stack())
  var trace = buf.String()
  traceLines := strings.Split(trace, "\n")
  pathPattern := regexp.MustCompile('/go/src/free5gc/NFs/[^/]+/')
  nfPath := regexp.MustCompile('/go/src/free5gc/NFs/([^/]+)/')
  webuiPath := "/go/src/free5gc/webconsole"
```

 doi:10.20944/preprints202410.1802.v1



```
  for _, line := range traceLines {
    isolatedPath := pathPattern.FindString(line)
    if isolatedPath != "" {
      nf := nfPath.FindStringSubmatch(isolatedPath)
      if len(nf) > 1 {
        return nf[1]
      }
    }
  }

  // if it is not a NF using the API, check for the webui
  if strings.Contains(trace, webuiPath) {
    return "webui"
  }
  // if nothing is found:
  fmt.Println("Couldn't find the specified path or network function. Trace:")
  fmt.Println(trace)
  return ""
}
...
func CreateOpenSSLInnerClientTransport() error {
  // Custom dial function to use OpenSSL for TLS connections
  dialTLS := func(network, addr string) (net.Conn, error) {
    conn, err := openssl.Dial(network, addr, innerOpenSSLClientCtx, openssl.InsecureSkipHost
    Verification)
    if err != nil {
      return nil, err
    }
    return conn, nil
  }
  // Create a custom transport using the custom dial function
  tr := &http.Transport{
    TLSClientConfig: &tls.Config{
      InsecureSkipVerify: true, // Skip certificate verification,
    },
    TLSNextProto: make(map[string]func(authority string, c *tls.Conn) http.RoundTripper),
    DialTLS:      dialTLS,
  }
  innerOpenSSLClient = &http.Client{Transport: tr}
  return nil
}
...
func CallAPI(cfg Configuration, request *http.Request) (*http.Response, error) {
  opensslCtx := CreateOpenSSLClientCtx(GetCurrentNetworkFunctionUsingAPI())
  if opensslCtx == nil {
    return nil, fmt.Errorf("Error Creating OpenSSL Context")
  }
  if cfg.HTTPClient() != nil {
    CreateOpenSSLClientTransportWithCfg(cfg, opensslCtx)
    return cfg.HTTPClient().Do(request)
  }
  if request.URL.Scheme == "https" {
    innerOpenSSLClientCtx = opensslCtx
    CreateOpenSSLInnerClientTransport()
    return innerOpenSSLClient.Do(request)
...
    return nil, fmt.Errorf("unsupported scheme[%s]", request.URL.Scheme)
}
```

Listing 3: Creation of HTTP client using Go-OpenSSL Wrapper.

 doi:10.20944/preprints202410.1802.v1



```
# Clone the project
git clone https://git.unl.edu/tel-papers/2024_pqc_for_securing_5gcn_mdpi/free5gc-pqc.git
cd free5gc-pqc
# Build the mongodb from source
./install_mongo.sh
# Build the images
make all
docker compose -f docker-compose-build.yaml build
# Alternatively you can build specific NF image e.g.:
make amf
docker compose -f docker-compose-build.yaml build free5gc-amf
```

Listing 4: Shell commands for building and installing free5GC.

| # | Time | Qty | Client | Dest | Protocol | Info |
|---|---|---|---|---|---|---|
| 1 | 0.000 | 1 | gNB | AMF | NGAP/ NAS-5GS | InitialUEMessage, Registration request |
| 2 | 0.006 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 5 | 0.032 | 1 | AMF | AUSF | TLSv1.3 | Client Hello |
| 6 | 0.039 | 3 | AUSF | NRF | TLSv1.3 | Client Hello |
| 9 | 0.065 | 1 | AUSF | UDM | TLSv1.3 | Client Hello |
| 10 | 0.074 | 3 | UDM | NRF | TLSv1.3 | Client Hello |
| 13 | 0.100 | 2 | UDM | UDR | TLSv1.3 | Client Hello |
| 15 | 0.114 | 1 | AMF | gNB | NGAP/ NAS-5GS | SACK (Ack=0, Arwnd=106496) , DownlinkNASTransport , Authentication request |
| 16 | 0.115 | 1 | gNB | AMF | NGAP/ NAS-5GS | SACK (Ack=0, Arwnd=106496) , UplinkNASTransport, Authentication response |
| 17 | 0.123 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 18 | 0.132 | 1 | AMF | AUSF | TLSv1.3 | Client Hello |
| 19 | 0.139 | 1 | AUSF | NRF | TLSv1.3 | Client Hello |
| 20 | 0.148 | 1 | AUSF | UDM | TLSv1.3 | Client Hello |
| 21 | 0.156 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 22 | 0.165 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 23 | 0.169 | 1 | AMF | gNB | NGAP/ NAS-5GS | SACK (Ack=1, Arwnd=106496) , DownlinkNASTransport , Security mode command |
| 24 | 0.170 | 1 | gNB | AMF | NGAP/ NAS-5GS | SACK (Ack=1, Arwnd=106496) , UplinkNASTransport, Security mode complete, Registration request |
| 25 | 0.176 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 28 | 0.203 | 1 | AMF | UDM | TLSv1.3 | Client Hello |
| 29 | 0.210 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 30 | 0.219 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 31 | 0.229 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 34 | 0.255 | 1 | AMF | UDM | TLSv1.3 | Client Hello |
| 35 | 0.262 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 36 | 0.271 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 37 | 0.280 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 38 | 0.292 | 1 | AMF | UDM | TLSv1.3 | Client Hello |
| 39 | 0.299 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 40 | 0.308 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 41 | 0.317 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 42 | 0.326 | 1 | AMF | UDM | TLSv1.3 | Client Hello |
| 43 | 0.333 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 44 | 0.343 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 45 | 0.351 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 46 | 0.361 | 1 | AMF | UDM | TLSv1.3 | Client Hello |
| 47 | 0.368 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 48 | 0.377 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 49 | 0.385 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 50 | 0.394 | 1 | AMF | UDM | TLSv1.3 | Client Hello |
| 51 | 0.401 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 52 | 0.411 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 53 | 0.418 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 56 | 0.445 | 1 | AMF | PCF | TLSv1.3 | Client Hello |
| 57 | 0.454 | 3 | PCF | NRF | TLSv1.3 | Client Hello |
| 60 | 0.480 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 61 | 0.489 | 3 | PCF | NRF | TLSv1.3 | Client Hello |
| 64 | 0.515 | 1 | PCF | AMF | TLSv1.3 | Client Hello |
| 65 | 0.518 | 1 | AMF | gNB | NGAP/ NAS-5GS | InitialContextSetupRequest, Registration accept |
| 66 | 0.518 | 1 | gNB | AMF | NGAP | SACK (Ack=2, Arwnd=106496) , InitialContextSetupResponse |
| 67 | 0.723 | 1 | gNB | AMF | NGAP/ NAS-5GS | UplinkNASTransport, Registration complete, UplinkNASTransport, UL NAS transport, PDU session establishment request, UplinkNASTransport, UL NAS transport, PDU session establishment request |
| 68 | 0.723 | 1 | AMF | gNB | NGAP/ NAS-5GS | Downlink NAS Transport, Configuration update command |
| 69 | 0.729 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 72 | 0.787 | 1 | AMF | NSSF | TLSv1.3 | Client Hello |
| 73 | 0.795 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 76 | 0.819 | 1 | AMF | SMF | TLSv1.3 | Client Hello |
| 77 | 0.829 | 3 | SMF | NRF | TLSv1.3 | Client Hello |
| 80 | 0.857 | 1 | SMF | UDM | TLSv1.3 | Client Hello |
| 81 | 0.865 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 82 | 0.874 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 83 | 0.884 | 5 | SMF | NRF | TLSv1.3 | Client Hello |
| 88 | 0.931 | 1 | SMF | PCF | TLSv1.3 | Client Hello |
| 89 | 0.940 | 1 | PCF | NRF | TLSv1.3 | Client Hello |
| 90 | 0.950 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 91 | 0.972 | 1 | PCF | NRF | TLSv1.3 | Client Hello |
| 92 | 0.981 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 93 | 0.989 | 1 | PCF | NRF | TLSv1.3 | Client Hello |
| 94 | 0.998 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 95 | 1.006 | 2 | PCF | NRF | TLSv1.3 | Client Hello |
| 97 | 1.024 | 3 | SMF | NRF | TLSv1.3 | Client Hello |
| 100 | 1.053 | 1 | SMF | CHF | TLSv1.3 | Client Hello |
| 101 | 1.069 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 102 | 1.072 | 1 | SMF | NRF | TLSv1.3 | Client Hello |
| 103 | 1.079 | 1 | AMF | NSSF | TLSv1.3 | Client Hello |
| 104 | 1.081 | 1 | SMF | AMF | TLSv1.3 | Client Hello |
| 105 | 1.084 | 1 | AMF | gNB | NGAP/ NAS-5GS | PDU session establishment accept |
| 106 | 1.086 | 1 | gNB | AMF | NGAP | PDU Session Resource Setup Response |
| 107 | 1.087 | 3 | AMF | NRF | TLSv1.3 | Client Hello |
| 110 | 1.112 | 1 | AMF | SMF | TLSv1.3 | Client Hello |
| 111 | 1.119 | 3 | SMF | NRF | TLSv1.3 | Client Hello |
| 114 | 1.146 | 1 | SMF | UDM | TLSv1.3 | Client Hello |
| 115 | 1.153 | 1 | UDM | NRF | TLSv1.3 | Client Hello |
| 116 | 1.162 | 1 | UDM | UDR | TLSv1.3 | Client Hello |
| 117 | 1.171 | 5 | SMF | NRF | TLSv1.3 | Client Hello |
| 122 | 1.217 | 1 | SMF | PCF | TLSv1.3 | Client Hello |
| 123 | 1.224 | 1 | PCF | NRF | TLSv1.3 | Client Hello |
| 124 | 1.234 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 125 | 1.251 | 1 | PCF | NRF | TLSv1.3 | Client Hello |
| 126 | 1.260 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 127 | 1.267 | 1 | PCF | NRF | TLSv1.3 | Client Hello |
| 128 | 1.276 | 1 | PCF | UDR | TLSv1.3 | Client Hello |
| 129 | 1.283 | 2 | PCF | NRF | TLSv1.3 | Client Hello |
| 131 | 1.299 | 3 | SMF | NRF | TLSv1.3 | Client Hello |
| 134 | 1.324 | 1 | SMF | CHF | TLSv1.3 | Client Hello |
| 135 | 1.336 | 1 | AMF | NRF | TLSv1.3 | Client Hello |
| 136 | 1.339 | 1 | SMF | NRF | TLSv1.3 | Client Hello |
| 137 | 1.345 | 1 | AMF | SMF | TLSv1.3 | Client Hello |
| 138 | 1.349 | 1 | SMF | AMF | TLSv1.3 | Client Hello |
| 139 | 1.351 | 1 | AMF | gNB | NGAP/ NAS-5GS | PDU session establishment accept |
| 140 | 1.353 | 1 | gNB | AMF | NGAP | PDU Session Resource Setup Response |
| 141 | 1.358 | 2 | AMF | NRF | TLSv1.3 | Client Hello |

**Figure A1.** All stages of a typical UE connection with a total of 129 TLSv1.3 handshakes.

 doi:10.20944/preprints202410.1802.v1



## References


1. Sultan, A. "5G System Overview — 3gpp.org". https://www.3gpp.org/technologies/5g-system-overview, 2022. [Accessed 23-09-2024].
2. 3rd Generation Partnership Program. "3GPP Release 18". https://www.3gpp.org/specifications-technologies/releases/release-18, 2024. [Available Online].
3. Neto, N.N.; Madnick, S.; de Paula, A.M.G.; Borges, N.M. "Developing a Global Data Breach Database and the Challenges Encountered". *J. Data and Information Quality* **2021**, *13*. doi:10.1145/3439873.
4. European Telecommunications Standards Institute. "ETSI GR QSC 006 v1.1.1: Quantum-Safe Cryptography (QSC); Limits to Quantum Computing applied to symmetric key sizes". https://www.etsi.org/deliver/etsi_gr/QSC/001_099/006/01.01.01_60/gr_qsc006v010101p.pdf, 2017.
5. McNeely, D. "Transitioning to Quantum-Safe Encryption". https://delinea.com/blog/quantum-safe-encryption, 2024.
6. National Institute for Standards and Technology. "Post-Quantum Cryptography". https://csrc.nist.gov/projects/post-quantum-cryptography, 2017.
7. 3rd Generation Partnership Program. TS 33.210 V18.1.0 Technical Specification Group Services and System Aspects; Network Domain Security (NDS); IP network layer security (Release 18). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2279, 2024. [Available Online].
8. 3rd Generation Partnership Program. TS 33.501 V19.0.0 Technical Specification Group Services and System Aspects; Security architecture and procedures for 5G system (Release 19). https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3169, 2024. [Available Online].
9. Open Quantum Safe. "Software for the transition to quantum-resistant cryptography". https://openquantumsafe.org, 2024. [Available Online].
10. Mounica, G.R.; Manimaran, G.; Jerome, L.B.; Bhattacharjee, P. "Implementation of 5-Qubit approach-based Shor's Algorithm in IBM Qiskit". 2021 IEEE Pune Section International Conference (PuneCon), 2021, pp. 1–6. doi:10.1109/PuneCon52575.2021.9686492.
11. Shor, P.W. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". *SIAM J. Comput.* **1997**, *26*, 1484–1509. doi:10.1137/S0097539795293172.
12. Grover, L.K. "A fast quantum mechanical algorithm for database search". Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing; Association for Computing Machinery: New York, NY, USA, 1996; STOC '96, p. 212–219. doi:10.1145/237814.237866.
13. Zeydan, E.; Turk, Y.; Aksoy, B.; Ozturk, S.B. "Recent Advances in Post-Quantum Cryptography for Networks: A Survey". 2022 Seventh International Conference On Mobile And Secure Services (MobiSecServ), 2022, pp. 1–8. doi:10.1109/MobiSecServ50855.2022.9727214.
14. Dam, D.T.; Tran, T.H.; Hoang, V.P.; Pham, C.K.; Hoang, T.T. "A Survey of Post-Quantum Cryptography: Start of a New Race". *Cryptography* **2023**, *7*. doi:10.3390/cryptography7030040.
15. Bavdekar, R.; Jayant Chopde, E.; Agrawal, A.; Bhatia, A.; Tiwari, K. "Post Quantum Cryptography: A Review of Techniques, Challenges and Standardizations". 2023 International Conference on Information Networking (ICOIN), 2023, pp. 146–151. doi:10.1109/ICOIN56518.2023.10048976.
16. National Institute for Standards and Technology. "NIST Releases First 3 Finalized Post-Quantum Encryption Standards — nist.gov". https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards, 2024. [Accessed 24-09-2024].
17. Scalise, P.; Boeding, M.; Hempel, M.; Sharif, H.; Delloiacovo, J.; Reed, J. "A Systematic Survey on 5G and 6G Security Considerations, Challenges, Trends, and Research Areas". *Future Internet* **2024**, *16*. doi:10.3390/fi16030067.
18. Clancy, T.C.; McGwier, R.W.; Chen, L. "Post-quantum cryptography and 5G security: tutorial". Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks; Association for Computing Machinery: New York, NY, USA, 2019; WiSec '19, p. 285. doi:10.1145/3317549.3324882.
19. Rossi Figlarz, G.; Passuelo Hessel, F. "Enhancing the 5G-AKA Protocol with Post-quantum Digital Signature Method". Advanced Information Networking and Applications; Barolli, L., Ed.; Springer Nature Switzerland: Cham, 2024; pp. 99–110.

20. Hoque, S.; Aydeger, A.; Zeydan, E. "Exploring Post Quantum Cryptography with Quantum Key Distribution for Sustainable Mobile Network Architecture Design". Proceedings of the 4th Workshop on Performance and Energy Efficiency in Concurrent and Distributed Systems; Association for Computing Machinery: New York, NY, USA, 2024; PECS '24, p. 9–16. doi:10.1145/3659997.3660033.
21. Minhas, N.N.; Mansoor, K. "Edge Computing-Based Scheme for Post-Quantum IoT Security for e-Health". *IEEE Internet of Things Journal* **2024**, pp. 1–1. doi:10.1109/JIOT.2024.3418959.
22. Hu, Y.C.; Patel, M.; Sabella, D.; Sprecher, N.; Young, V. "Mobile Edge Computing A key technology towards 5G". https://infotech.report/Resources/Whitepapers/f205849d-0109-4de3-8c47-be52f4e4fb27_etsi_wp11_mec_a_key_technology_towards_5g.pdf, 2015. [Available Online].
23. Zhang, J.; Xie, W.; Yang, F.; Bi, Q. "Mobile edge computing and field trial results for 5G low latency scenario". *China Communications* **2016**, *13*, 174–182. doi:10.1109/CC.2016.7405733.
24. Chen, Q.; Wang, Z.; Su, Y.; Fu, L.; Wei, Y. "Educational 5G Edge Computing: Framework and Experimental Study". *Electronics* **2022**, *11*. doi:10.3390/electronics11172727.
25. Zheng, J.; Zhu, H.; Dong, Y.; Song, Z.; Zhang, Z.; Yang, Y.; Zhao, Y. "Faster Post-quantum TLS 1.3 Based on ML-KEM: Implementation and Assessment". Computer Security – ESORICS 2024; Garcia-Alfaro, J.; Kozik, R.; Choraś, M.; Katsikas, S., Eds.; Springer Nature Switzerland: Cham, 2024; pp. 123–143.
26. Zeydan, E.; Blanco, L.; Mangues-Bafalluy, J.; Aydeger, A.; Arslan, S.; Turk, Y. Integrating Quantum-Secured Blockchain Identity Management in Open RAN for 6G Networks. 2024 IEEE 49th Conference on Local Computer Networks (LCN), 2024, pp. 1–7. doi:10.1109/LCN60385.2024.10639816.
27. Javeed, D.; Saeed, M.S.; Ahmad, I.; Adil, M.; Kumar, P.; Islam, A.N. Quantum-empowered federated learning and 6G wireless networks for IoT security: Concept, challenges and future directions. *Future Generation Computer Systems* **2024**, *160*, 577–597. doi:https://doi.org/10.1016/j.future.2024.06.023.
28. Nakka, K.; Ahmad, S.; Kim, T.; Atkinson, L.; Ammari, H.M. Post-Quantum Cryptography (PQC)-Grade IEEE 2030.5 for Quantum Secure Distributed Energy Resources Networks. 2024 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT), 2024, pp. 1–5. doi:10.1109/ISGT59692.2024.10454235.
29. Lawo, D.C.; Abu Bakar, R.; Cano Aguilera, A.; Cugini, F.; Imaña, J.L.; Tafur Monroy, I.; Vegas Olmos, J.J. Wireless and Fiber-Based Post-Quantum-Cryptography-Secured IPsec Tunnel. *Future Internet* **2024**, *16*. doi:10.3390/fi16080300.
30. open5G. "Core Network | Open, Programmable and Virtualized 5G Networks". https://open5g.info/core/, 2024. [Available Online].
31. Linh, A.B.N. "Analysing open-source 5G core networks for TLS vulnerabilities and 3GPP compliance". https://cs.ru.nl/bachelors-theses/2023/Alex_Bui_Nhat_Linh___1040308___Analysing_open-source_5G_core_networks_for_TLS_vulnerabilities_and_3GPP_compliance.pdf, 2023. [Available Online].
32. TLS-Attacker. "TLS-Scanner". https://github.com/tls-attacker/TLS-Scanner, 2023. [Available Online].
33. Diquet, A. "SSLyze". https://github.com/nabla-c0d3/sslyze, 2024. [Available Online].
34. DinoTools. "PySSLScan". https://github.com/DinoTools/pysslscan, 2022. [Available Online].
35. Adhikary, A.; Deb Raha, A.; Qiao, Y.; Saad, W.; Han, Z.; Seon Hong, C. Holographic MIMO With Integrated Sensing and Communication for Energy-Efficient Cell-Free 6G Networks. *IEEE Internet of Things Journal* **2024**, *11*, 30617–30635. doi:10.1109/JIOT.2024.3411695.
36. Chen, W.; Lin, X.; Lee, J.; Toskala, A.; Sun, S.; Chiasserini, C.F.; Liu, L. 5G-Advanced Toward 6G: Past, Present, and Future. *IEEE Journal on Selected Areas in Communications* **2023**, *41*, 1592–1619. doi:10.1109/JSAC.2023.3274037.
37. Boeding, M.; Hempel, M.; Sharif, H. Novel Approach towards a Fully Deep Learning-Based IoT Receiver Architecture: From Estimation to Decoding. *Future Internet* **2024**, *16*, 155.
38. Chen, R.H.; Zhou, J.; Zhu, Y.; Zhang, K. When to Simply Use Passive RIS as Beamformer: An Information-Theoretic Analysis and A Novel Single-RF MIMO Transceiver Architecture. *IEEE Transactions on Wireless Communications* **2024**, pp. 1–1. doi:10.1109/TWC.2024.3451531.
39. Karakaya, A.; Ulu, A. A survey on post-quantum based approaches for edge computing security. *Wiley Interdisciplinary Reviews: Computational Statistics* **2024**, *16*, e1644.
40. OpenSSL. "OpenSSL". https://www.openssl.org, 2024. [Available Online].
41. BoringSSL. "BoringSSL — github". https://github.com/google/boringssl/, 2024. [Available Online].

 doi:10.20944/preprints202410.1802.v1




42. free5GC. "free5GC Roadmap". https://free5gc.org/#next-steps, 2024. [Available Online].
43. Go. "Build simple, secure, scalable systems with Go". https://go.dev, 2024. [Available Online].
44. Go. "crypto/tls Go Library". https://pkg.go.dev/crypto/tls, 2024. [Available Online].
45. Pexip. "Go-Openssl Wrapper". https://pkg.go.dev/github.com/pexip/go-openssl?utm_source=godoc, 2024. [Available Online].
46. Schwabe, P. "Performance Overview of CRYSTALS-Kyber". Technical report, PQ-CRYSTALS, 2020. [Available Online].
47. Kölbl, Stefan and Schmieg, Sophie and Endignoux, Guillaume. Why Hybrid Deployments Are Key to Secure PQC Migration. https://bughunters.google.com/blog/5266882047639552/why-hybrid-deployments-are-key-to-secure-pqc-migration, 2024. [Available Online].
48. Chen, X.; Ng, D.W.K.; Yu, W.; Larsson, E.G.; Al-Dhahir, N.; Schober, R. Massive access for 5G and beyond. *IEEE Journal on Selected Areas in Communications* **2020**, *39*, 615–637.