IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 17 February 2024, accepted 5 March 2024, date of publication 7 March 2024, date of current version 15 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3374629

RESEARCH ARTICLE

Falcon/Kyber and Dilithium/Kyber Network Stack on Nvidia's Data Processing Unit Platform

D. C. LAWO[1,2], R. FRANTZ[1,2], A. CANO AGUILERA[1,2], (Graduate Student Member, IEEE), X. ARNAL I CLEMENTE[1], M. P. PODLEŚ[1,2], JOSÉ L. IMAÑA[3], I. TAFUR MONROY[1,2], (Senior Member, IEEE), AND J. J. VEGAS OLMOS[2]

[1]Department of Electrical Engineering, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands
[2]Software Architecture, NVIDIA Corporation, Tel Aviv-Yafo 6121002, Israel
[3]Department of Computer Architecture and Automation, Complutense University of Madrid, 28040 Madrid, Spain

Corresponding author: D. C. Lawo (d.c.lawo@tue.nl)

This work was supported in part by the Quantum Resistant Communications (QUARC): Advance Learning in Applied PQC Training Project by the European Union Horizon Europe Research and Innovation Program within the Framework of Marie Skłodowska-Curie Actions under Grant 101073355, and in part by the Collaborative edge-cLoud continuum and Embedded AI for a Visionary industry of thE futuRe (CLEVER) Project by the Key Digital Technologies Joint Undertaking Program under Grant 101097560.

ABSTRACT Commercially available quantum computers are expected to reshape the world in the near future. They are said to break conventional cryptographic security mechanisms that are deeply embedded in our today's communication. Symmetric cryptography, such as AES, will withstand quantum attacks as long as the key sizes are doubled compared to today's key lengths. Asymmetric cryptographic procedures, e.g. RSA, however are broken. It is therefore necessary to change the way we assure our privacy by adopting and moving towards post-quantum cryptography (PQC) principles. In this work, we benchmark three PQC algorithms, Falcon, Dilithium, and Kyber. Moreover, we present an implementation of a PQC stack consisting of the algorithms Dilithium/Kyber and Falcon/Kyber which use hardware accelerators for some key functions and evaluate their performance and resource utilization. Regarding a classic server-client model, the computational load of the Dilithium/Kyber stack is distributed more equally among server and client. The stack Falcon/Kyber biases the computational challenges towards the server, hence relieving the client of performing costly operations. We found that Dilithium's advantage over Falcon is that Dilithium's execution is faster while the workload to be performed is distributed equally among client and server, whereas Falcon's advantage over Dilithium lies within the small signature sizes and the unequally distributed computational tasks. In a client server model with a performance limited client (i.e. Internet-of-Things - IoT - environments) Falcon could proof useful for it constrains the computational hard tasks to the server and leaves a minimal workload to the client. Furthermore, Falcon requires smaller bandwidth, making it a strong candidate for deep-edge or IoT applications.

INDEX TERMS Data processing units, Dilithium, falcon, Kyber, post-quantum cryptography.

I. INTRODUCTION

Since several years now, quantum computing is a subject of intense research [1]. A commercial quantum computer is expected to be available within the upcoming years while prototype systems [2] or digital annealers [3] as well as quantum annealers [4] that operate on similar terms are already available. That poses a severe thread to our nowadays used communication systems based on classic cryptographic infrastructure and methodologies. Asymmetric cryptography, such as for example RSA, is said to be broken by a quantum processor [5]. Symmetric cryptography, e.g. AES, is expected to be quantum safe as opposed to asymmetric cryptography as long as its key size is doubled [6]. The currently used asymmetric cryptography algorithms need to be replaced by quantum secure algorithms or so-called post-quantum cryptography which are developed specifically to be resilient against quantum computers.

The associate editor coordinating the review of this manuscript and approving it for publication was Lei Shu.

As a result of this reality, Fig. 1 shows the narrative on which future communication networks will most likely evolve: deep-edge [7] verticals that process data employing confidential computing (CC) [8] techniques locally and communicate securely to edge nodes using PQC. The edge nodes employ not only confidential computing, but software defined perimeters (SDP) [9], [10], [11] to isolate processes, utilize quantum resilient communications and smart orchestrators [12] to distribute workloads federatively. When necessary, they connect to the cloud which also operates under the same principles.

SDPs and CC are both related to improving the security of cloud computing environments. SDPs are a security architecture that creates a secure, isolated network connection between the user and the application. This connection is dynamically established using cryptographic techniques. Access to the application is granted only to authorized users. SDPs, which may include Trust Zones [13], are fundamental in some verticals where the infrastructure is massively shared by different, if not multiple, stakeholders. In cellular networks, to give an example, SPDs are normally leveraged to isolate data traffic from signalling traffic and Operation and Maintenance (OAM) traffic [14], or to support multi-tenancy [15]. This leads to the core of the issue investigated in this work: provided that in some verticals PQC encryption will run from the deep edge (mobile user) to the cloud, how do we select the most suitable PQC algorithm depending on the resources available by the owners of each of the communication channels?

Confidential computing is a set of technologies and practices that protect data in use, ensuring that sensitive data is never exposed, not even to the cloud provider [8]. CC technologies use hardware-based encryption to isolate and protect data in memory and prevent unauthorized access. The technology therefore uses trusted execution environments and dedicated hardware [16]. While we envision a transition from non-confidential computing towards confidential computing, we include in Fig. 1 an intermediate state where CC nodes and non-CC nodes coexist. SDPs and CC can be used together to enhance the security of cloud computing environments. SDPs can provide an additional layer of security by ensuring that only authorized users can access the application [11], while CC can protect the data that is being processed within the application. Together, these technologies can help mitigate the risks associated with data breaches and other cyber-attacks. They need, however, to integrate PQC systems, and that includes the possibility of different PQC technologies co-existing on end-to-end links that span from the deep edge all the way into the cloud data center.

The overall goal of enabling quantum and cyber resilient infrastructures offering an edge-cloud continuum [17] supporting heavy AI-based workloads is achieved through the interconnection of diverse processing building blocks. A heterogeneous fabric that is dynamic, that changes over time through upgrades and replacements. Figure 2 shows this edge-cloud continuum: deep edge nodes are operating under limited connectivity and low power conditions. Those can be mobile devices supporting fintech applications, autonomous robots in factory environments, or connected cars that operate under similar terms like [18], to name a few examples. However, this does not preclude local processing power for AI processes and establishment of quantum resilient links. These deep edge nodes contain both communication accelerators as well as processing units upgradeable through PQC-based processes. Since the processing capabilities of those deep edge nodes are low, they can access edge computing platforms. The edge computing platform may be highly dense edge micro data centers or lite edge nodes; regardless of the entrance point, processes can be offloaded through light containers [19]. In addition, cloud fabrics can be connected to the edge platform. All these links are based on PQC principles, creating a quantum resilience for data-in-transfer. At the edge platform, processing blocks operate under confidential computing principles, in which trust is secured at both HW/SW level. In addition, SDP can be created to cluster CC-based processors.

In this work, we investigate experimentally different PQC algorithms when deployed on high-capacity network interface cards, including hardware accelerators to offload some key functions. The herein presented comparison is done in order to discern the resources needed for the PQC algorithms Kyber [20], Dilithium [21], and Falcon [22]. Hence, designers can be equipped better for their choice of system depending on the communication channel segment they are deploying their application on. This is achieved by implementing the PQC software stack for key sharing between a server and a client. This stack consists of two parts, a PQC signature algorithm for authentication of the partners and a PQC key exchange mechanism (KEM). The implemented signature algorithms are Dilithium [21] and Falcon [22]. Furthermore, the PQC KEM is Kyber [20] as it is the only KEM left in the NIST competition [23]. Fundamentally, Dilithium's security is based on the hardness of lattice problems over module lattices [24]. It builds on learning-with-errors (LWE) and Short Integer Solution (SIS) problems. Like Kyber, another algorithm used in this work, Dilithium is a part of the Cryptographic Suite for Algebraic Lattices (CRYSTALS). Similarly, Falcon's security roots on NTRU-lattices [22]. Kyber relies on the challenge of solving learning-with-errors problems over module lattices [20] in order to facilitate the secure digital key exchange [25].

A. RELATED WORKS

There has been extensive work on different implementations of PQC algorithms using a variety of different base platforms. Some of the works focus on using special hardware platforms such as FPGA [26], [27], GPU [28], RISC-V [29], [30] or combination of platforms [31] to accelerate computationally heavy mathematical parts of algorithms. Some parts of the PQC algorithms, e.g. Number Theoretic Transform (NTT),

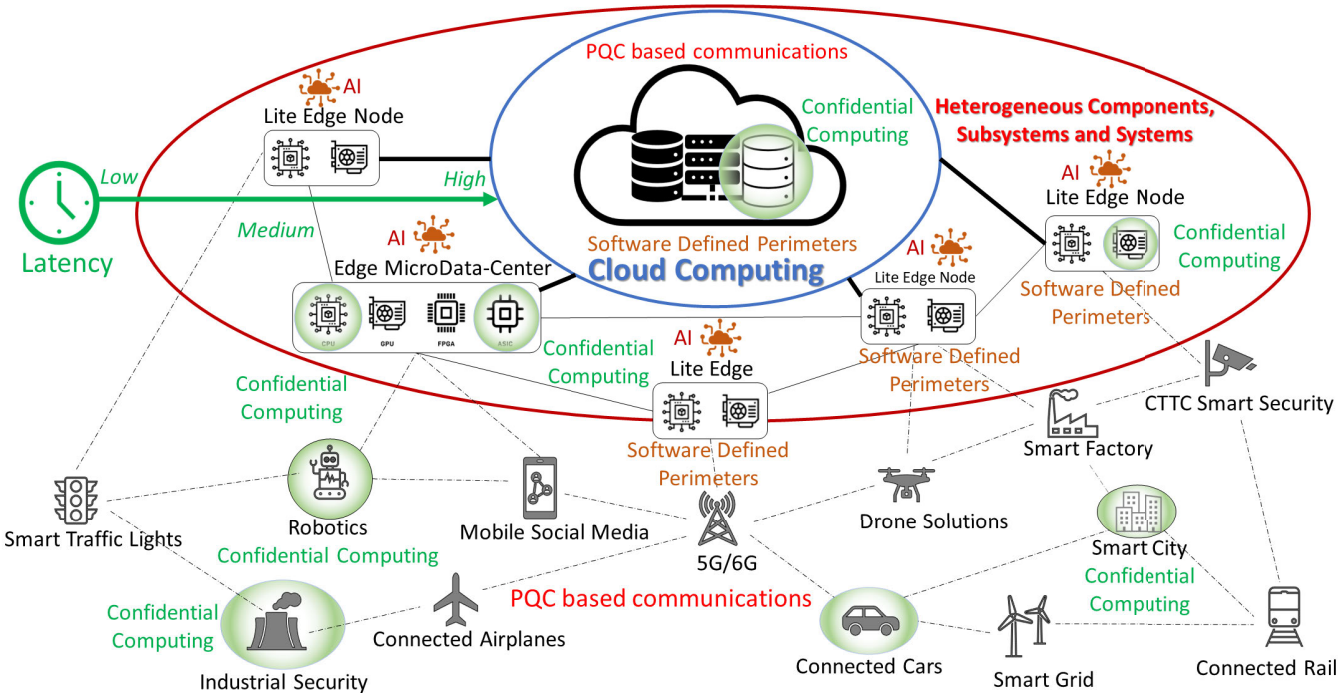

FIGURE 1. Overall vision of the overlay of communication network segments and different emerging key applications. Cloud computing is foreseen to massively employ confidential computing [8], including trusted execution environments [16], and software-defined perimeters [9], [10], [11] to isolate AI workloads on shared infrastructure. We anticipate a shift from non-confidential computing to confidential computing, incorporating transitional stages that encompass both non-confidential and confidential computing nodes. Cloud platforms are interconnected through edge computing platforms with different degrees of processing power and are able to implement confidential computing too. Both cloud and edge computing platforms rely on a large and dynamic number of point-to-point connections which will rely on PQC protocols.

can be parallelized and therefore benefit from platforms that can provide heavy parallelization, such as GPUs or FPGAs.

In [32], the authors implement Falcon on a FPGA. Deploying Falcon on a FPGA is a challenge because Falcon uses floating point operations and recursive functions which require modifications before they can be placed in hardware. The authors state that their implementation of Falcon's key generation shows a high latency compared to an Intel I7 software-based implementation while having a high hardware utilization footprint [32]. They therefore state that implementing Falcon's key generation on a FPGA is not attractive. Furthermore, they demonstrate that implementing the signing process, as well as the verification process, on a FPGA could be beneficial for the latency can be significantly reduced compared to a pure software-based solution.

Other works focus on better software implementations that leverage special features of general purpose computers [33], [34]. Dilithium and Kyber, for example, offer implementations [20], [21] that take advantage of the Advanced Vector Extensions operations (AVX2) [35]. The optimized versions of Dilithium and Kyber that leverage AVX2 achieve a significant speedup [20], [21].

This paper focuses on the reference implementations of the PQC algorithms Falcon, Dilithium, and Kyber on Nvidia's data processing unit platform. Compared to mentioned work, this paper presents the first in-line acceleration using DPUs at data center linerates.

II. EXPERIMENTAL SETUP

This section describes the experimental setup with that the herein presented results are obtained. It is similar to the one presented in [36], but expanding the software stack to the new PQC algorithms under evaluation. The schematics of the experimental setup can be seen in Fig. 3. Two Dell PowerEdge R750[1] host each one NVIDIA BlueField 2 MBF2M516A-CEEO[2] data processing unit (DPU) connected point-to-point via copper cable. Each DPU is capable of transmitting and receiving data at 100 Gb/s linerate. The DPUs are connected to their respective server through peripheral component interconnect express (PCIe) bridges. Eight on-board ARMv8 processor cores clocked at 2750 MHz are part of each DPU. The goal of the experiment is to share keys between server A / DPU A and server B / DPU B shown in Fig. 3. For this purpose we use Falcon/Kyber and Dilithium/Kyber software stacks. Once both communicating parties have retrieved the key, they communicate over symmetric encryption which is said to be quantum resistant [6]. Both DPUs are equipped with hardware accelerators for this purpose, and the software stack employs them throughout the calling of the different processes. It is worth pointing out that although there are

[1]https://i.dell.com/sites/csdocuments/Product_Docs/en/poweredge-r750-spec-sheet.pdf

[2]https://docs.nvidia.com/networking/display/bluefield2DPUENUG/specifications

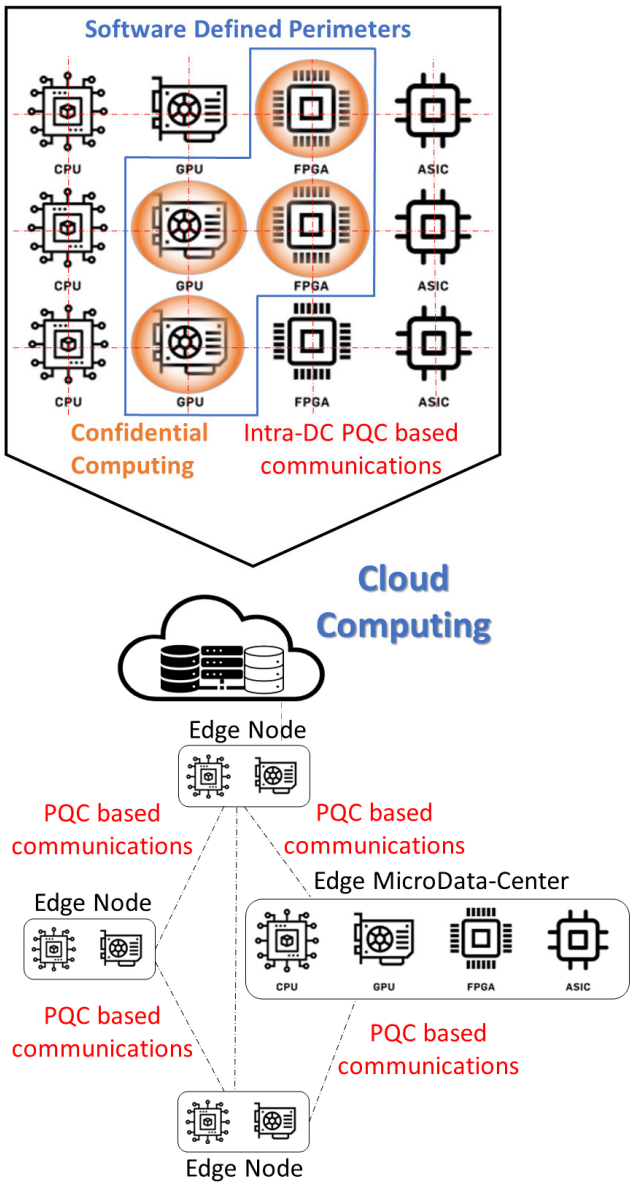

FIGURE 2. Confidential Computing processing blocks can be clustered through Software Defined Perimeters while all blocks are securely exchanging data using PQC protocols. These large cloud computing platforms are then connected to edge computing platforms of different sizes which are interconnected through communication channels using PQC encrypted protocols as well. Depending on communication scenario, the type of node, the devices used for communication, or other factors, the ideally employed PQC algorithm can differ.

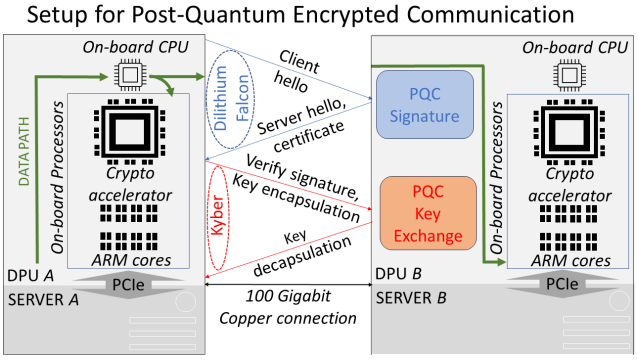

FIGURE 3. Experimental Setup for establishing a PQC encrypted handshake based on either Dilithium/Kyber or Falcon/Kyber. Both DPUs, A and B, are hosted in two separate servers and connected to them over PCIe. The DPUs are interconnected point-to-point by a copper wire connection capable of providing up to 100 Gb/s.

reports on implementations of hardware accelerators for Dilithium [29] and Falcon [30], this paper presents the first in-line acceleration at data center linerates.

A. IMPLEMENTATION

We first start the protocol by establishing a channel between client and server using openssl [37]. For the initial step of authentication we use self-signed certificates for both client and server. In case the protocol was to be used for real-world use cases, the self-signed certificates would need to be replaced by real certificates provided by a certificate authority [38]. Then, depending on the version of the software stack, we use the PQC signature schemes Falcon or Dilithium in order to benchmark them in their various security levels. Subsequently, we exchange keys using Kyber. For all three PQC schemes we use their reference implementation application programming interfaces (APIs) of Falcon [39], Dilithium [40], and Kyber [41]. Both reference implementations of Dilithium and Kyber are based on a library that implements the FIPS-202 algorithms [42] such as NTT for multiplication of polynomials with coefficients mod integers and barret and montgomery reductions for modular operations. We implement the software stack by using the reference implementation of the algorithms. For benchmarking purposes the algorithms Falcon, Dilithium, and Kyber are executed repeatedly for 10.000 times to gain statistically relevant results. In the real-world scenario, the algorithms are executed only once. After the signature scheme and the key exchange, we have the shared key from the openssl session and the PQC key exchanged using Kyber. Then we mix the openssl shared key with the PQC keys by performing an XOR operation. This way we ultimately retrieve a secure ephemeral 256-bit key that is used for AES-256 symmetric key encryption [6] as explained in [43].

III. RESULTS

All algorithms consist of multiple processing intensive parts. In this part we are analysing the main functions of Falcon's [22], Dilithium's [21] and Kyber's [20] reference implementations. The main three steps of the signature algorithms Falcon and Dilithium are key generation, verification, and sign. Key generation and signing are done by the server. It is the client's responsibility to ensure the validity of the server's signature. This step is called verification.

A. BENCHMARK: FALCON

Figure 4 shows the reference implementation of Falcon presented in [22], profiled in terms of cpu cycles that are

required per execution. Falcon is one of three remaining signature algorithms in the NIST competition [23]. Falcon relies on the theoretical foundation developed by Gentry, Peikert, and Vaikuntanathan for lattice-based signature schemes [44]. This framework is applied to NTRU lattices, utilizing a trapdoor sampler known as "fast Fourier sampling." The fundamental challenge addressed is the short integer solution problem (SIS) over NTRU lattices [22], [45].

As signature algorithm Falcon comprises the three main subroutines key generation, signing, and verification [22]. Falcon 512 has a public key size of 897 bytes and a signature size of 666 bytes while achieving NIST security level 1. Falcon 1024 satisfies NIST level 5 with a public key size of 1793 bytes and a signature size of 1280 bytes [22]. Falcon 256 is not considered NIST secure. Regardless, it is included in the reference implementation for research purposes. Therefore, it is included in Fig. 4 as a reference but excluded in the detailed comparison.

During the key generation the algorithm employs AES-generated pseudo-random numbers as seeds to initialize SHAKE-256 for calculating random polynomials with a Gaussian distribution. If the squared norm of these polynomials is beyond bounds, or if the orthogonalized vector norms deviate, the algorithm discards them and generates new polynomials. The Fast-Fourier Transform (FFT) is utilized to compute orthogonalized vector norms. Using the polynomials, the algorithm generates a public key polynomial. The key generation component solves the NTRU equation to compute the key polynomials.

As shown in Fig. 4, the computationally most challenging part is the key generation. This procedure of Falcon is by almost two orders of magnitude slower than all other functions of the algorithm. Upgrading security from Falcon 512/NIST 1 to Falcon 1024/NIST 5 comes with a penalty of a 93% increase of required cpu clock cycles for the key generation. Signing takes 69% more clock cycles for Falcon 1024 compared to Falcon 512. The verification of a signature requires 70% more clock cycles for the NIST 5 version of Falcon confronted with the NIST 1 version. Moreover, it is clear to see that the step of the key generation is subject to the largest standard deviation compared to the other functions.

The functions A, B, C, D, E, and F benchmarked in Fig. 4 are performed by the server while the function G and H are executed by the client.

B. BENCHMARK: DILITHIUM

Dilithium's security standards 2, 3 and 5 vary in size of their corresponding public key and signature size. Dilithium 2 has a public key size of 1312 bytes and a signature size of 2420 bytes. Using Dilithium 3 implies a public key size of 1952 bytes and a signature size of 3293 bytes. The highest security level of the algorithm, Dilithium 5, has a 2592 bytes public key and a 4595 bytes signature [21].

The herein tested implementation of Dilithium uses SHAKE for matrix expansion, vector masking, and sampling of the secret polynomials. A version of Dilithium's reference

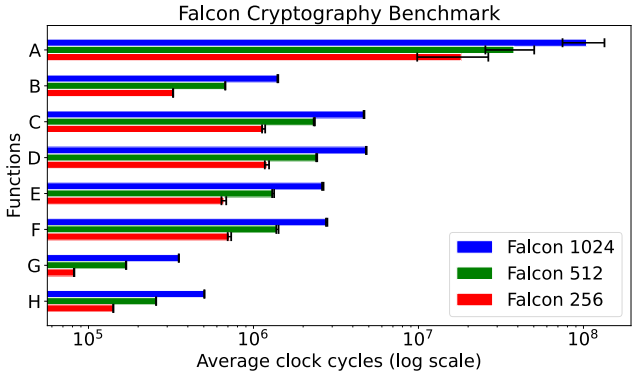

FIGURE 4. CPU performance of main Falcon functions conducted on DPU. The functions' names have been kept as in the reference implementation: *A = keygen, B = expand private key, C = sign without expanded key, E = sign with expanded key, G = verify.* Additionally, the functions *D, F,* and *H* are respectively identical to the functions *sign (C), sign with expanded key (E),* and *verify (G)* with constant-time hash-to-point.

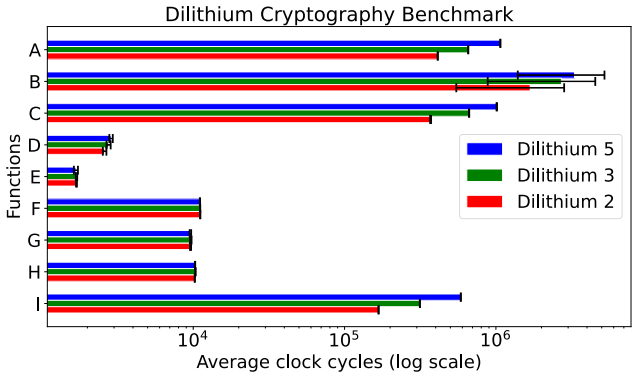

FIGURE 5. Benchmark of Dilithium's main functions in required cpu cycles per execution. *A = Verify, B = Sign, C = Keypair, D = poly_challenge, E = poly_pointwise_montgomery, F = poly_ntt, G = poly_uniform_gamma1, H = poly_uniform_eta,* and *I = polyvec_matrix_expand.*

implementation using AES for the aforementioned steps exists. However, this version is not considered here since it requires AVX2 operations and the DPU's ARM cores do not support AVX2. Therefore, during the key generation SHAKE is used. Moreover, NTT is employed during the signature generation [21], [45].

As with Falcon in the previous section, here we benchmark Dilithium's reference implementation [21] in the unit cpu cycles. The results of this experiment are shown in Fig. 5. In terms of verification, upgrading the security level from Dilithium 2 to Dilithium 3 comes with a 45% penalty. From NIST level 3 to NIST level 5 the costs are increased further by 47%. Signing rises the required clock cycles for upgrading Dilithium 2 to Dilithium 3 by 46% and an even further 20% increase from Dilithium 3 to 5.

Generating a key pair costs a 57% difference in required clock cycles while increasing the security level from 2 to 3. An additional 42% are needed if going up from Dilithium 3 to Dilithium 5. Signing is the function that has the largest standard deviation compared to other functions of Dilithium.

The functions A, and E benchmarked in Fig. 5 are executed by the client. B, C and F are performed by the server. D, G, and I are used by both the server and the client.

C. BENCHMARK: KYBER

Kyber is the only remaining KEM in the NIST competition [23] and it will most likely be the main algorithm for the key exchange stage of PQC-based communication. As such, the reference implementation includes Kyber 512, 768 and 1024 [20] where the user can select this segment's security level of the key exchange. The main routines of Kyber are called key generation, key encapsulation, and key decapsulation. During the key generation, a key pair consisting of a public key and a private key is generated. The public key is sent openly over a network while the private key is kept privately at all time. The purpose of the key encapsulation is to encrypt the key to be shared with the public key. Ultimately, the key decapsulation serves for retrieving the key that had been encrypted with the public key in the encapsulation step.

Kyber's three security levels employ different public key, private key and cipher text sizes. The lowest security level, Kyber 512, has a 1632 bytes private key, an 800 bytes public key, and a 768 ciphertext. Kyber 768 requires a 2400 bytes private key, a 1184 bytes public key, and a 1088 bytes ciphertext. Kyber 1024, the highest security level, uses a 3168 bytes private key, a 1568 bytes public key, and a 1568 bytes cyphertext [20]. Like Dilithium, Kyber uses NTT to achieve its security. The algorithm performs arithmetic operations on 256 bit polynomials over a polynomial ring. Regardless of the security level, the polynomials' size and modulus remain unchanged. Only the number of polynomials involved grows with increasing security level [20].

The results of the benchmark test are shown in Fig. 6. Comparing the three main routines of each Kyber NIST level in terms of the key pair generation reveals a 47% penalty when upgrading from Kyber 512 to Kyber 768. Further enhancing the security to Kyber 1024 comes with a cost of 41% more required clock cycles compared to Kyber 768. Confronting the key encapsulation of Kyber 512 with Kyber 768 shows that 26% more cpu cycles are needed for Kyber 768. Moreover, advancing towards Kyber 1024 requires 37% more clock cycles. A security level upgrade of Kyber 512 to Kyber 768 costs 41% for the key decapsulation routine. Choosing the highest NIST level of Kyber is penalized with an additional 34% during the decapsulation of a key compared to Kyber 768. The functions A, C, and D benchmarked in Fig. 6 are executed by the server. B and E are performed by the client. F, G, and H are used by both the server and the client.

IV. COMPARISON: FALCON DILITHIUM

In Fig. 7, the main three functions of signature algorithms, key generation, sign, and verify, have been repeatedly executed over a given time. Generally speaking, the computational load that needs to be performed for Dilithium

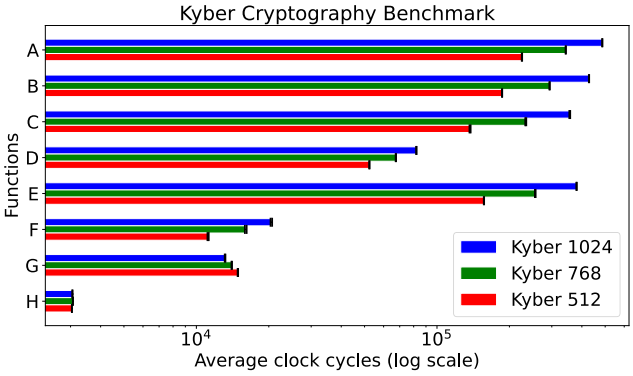

FIGURE 6. CPU performance of main Kyber functions. Using the labeling of the reference implementation [20], the functions represented here are: *A = kyber_decaps, B = kyber_encaps, C = kyber_keypair, D = indcpa_dec, E = indcpa_enc, F = polyvec_basemul_acc_montgomery, G = INVNTT,* and *H = NTT.*

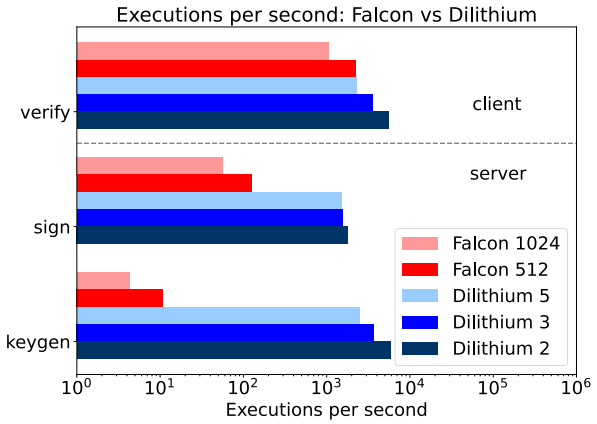

FIGURE 7. Comparison between the main three parts, key generation, sign, and verify, of Dilithium (blue) and Falcon (red). The step 'verify' is performed by the client machine while the server executes the 'keygen' and the 'sign' steps. Dilithium has three NIST security levels, 2, 3 and 5 where Falcon has two Nist security levels, 1 and 5. Falcon's key generation is slower by several orders of magnitude compared to Dilithium. Moreover, Falcon's signing procedure is significantly slower than Dilithium's equivalent. The speed of Falcon's verification procedure is comparable to Dilithium's verification.

by client and server is similar for both, server and client. Regarding Falcon, however, the computational load distribution is different. Falcon's server has to perform very intense computational tasks, especially for the key generation. The client only computes the verification that is computationally less demanding compared to the key generation and signing which are the server's responsibility. This implies different use cases for Dilithium compared to Falcon. It is notable that increasing Dilithium's security level from 2 to 3 to 5 comes at a small computational penalty. Falcon's key generation is extremely slow compared to Dilithium. However, that step is usually performed on a computationally powerful machine. Moreover, that step does not need to be done very often as the keys are generated once and then distributed to the client. Both algorithms' working principles, as well as their security levels, are described in [46] in further detail.

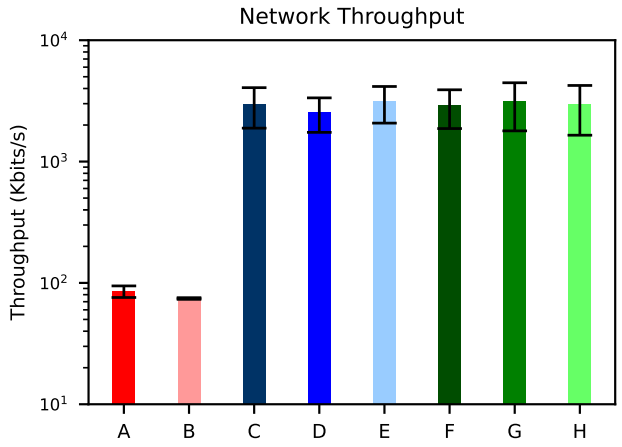

FIGURE 8. Network throughput of PQC algorithms Falcon, Dilithium, and Kyber. A = Falcon 512, B = Falcon 1024, C = Dilithium 2, D = Dilithium 3, E = Dilithium 5, F = Kyber 512, G = Kyber 768, and H represents Kyber 1024. The network load required by the execution of Falcon 512 and Falcon 1024 is significantly lower than Dilithium's and Kyber's network load. It is evident that Dilithium generastes a higher network throughput than Falcon.

A. NETWORK LOAD

As presented, it is faster to execute Dilithium as signature algorithm than Falcon. Hence, the generated network load resulting from this is higher. This can be seen in Fig. 8. The throughput of Falcon, however, is significantly lower by about the factor of two orders of magnitude. In data centers where the higher network load of Dilithium is negligible compared to the available network capacity, Falcon is less suitable than Dilithium. Moreover, in the data center scenario, the equal computational load distribution has no impact since client and server are likely to be both powerful machines. Considering mobile or IoT applications where the systems' priorities could be shifted towards lower network and processing loads on the client side, Falcon offers an interesting alternative to Dilithium. The computationally most expensive parts of the algorithm, the key generation, and the signature, are done on the server side while the client only has to perform the verification. The client requirements in terms of network bandwidth or processing power are very small compared to Dilithium. Especially in battery powered clients, the low computational requirements of the verification combined with a low network load is an advantage.

B. COMPARISON: IMPLEMENTATIONS ON OTHER HARDWARE PLATFORM

In this section we compare the results that we are presenting in this work to implementations reported on other hardware platforms. This can be seen in Fig. 9. In [32] the authors implement Falcon 512 and Falcon 1024 on a Zynq Ultrascale+ FPGA ZCU104[3] from AMD-Xilinx. The authors use the Falcon reference C-code [22] submitted to the NIST PQC standardization process [23] for their work.

TABLE 1. Number of clock cycles required per function reported in [32]. As the latency of the key_gen depends on a random seed, in [32], only the latency including the standard deviation is indicated. However, the thereof resulting number of clock cycles can be easily derived using the operating clock frequency.

security	Function	Clock Cycles	Latency (ms)	Clock (MHz)
512	key_gen	†	113.7 ± 22.2	100.0
	sign_tree	787.441	4.2	187.5
	verify	132.482	0.6	214.3
1024	key_gen	†	320.3 ± 69.1	100.0
	sign_tree	1.638.253	8.7	187.5
	verify	269.608	1.3	214.3

To achieve their results, the authors use the frameworks Vivado and Vitis-HLS that are provided by AMD-Xilinx. The execution time of the keypair generation, signature, and verification is reported in the paper in the unit of ms. For the purpose of comparability, we have converted the values reported in [32] into executions per second. Moreover, as the advantage of a hardware implementation lies not only within the potential gain in speed but also in the deterministic execution, Table 1 shows clock cycles required for the execution of Falcon's functions as presented in [32]. The results in executions per seconds are shown in Fig. 9 in red. It can be seen that verification achieves a speedup with respect to the software-based implementation of the DPU and the Raspberry Pi 4. However, implementing the signature and the key generation of Falcon on a FPGA is a very hard task since these parts of the algorithm uses floating-point numbers and recursive functions which makes it a challenge from a hardware design point of view [32]. Therefore, the obtained results for the signature and key generation are considerably slower compared to the DPU and the Raspberry PI 4 as it can be seen in Fig. 9.

In [47], the authors present an implementation of Dilithium and Kyber on a Xilinx Ultrascale+ ZCU102 platform.[4] With a performance-optimized strategy using the Vivado 2019.1 tool they achieve 270 MHz clock frequency on the FPGA. The authors call the architecture that they propose KaLi. It is specifically tailored for ASIC platforms and it can perform all steps of all security levels of Dilithium and Kyber. Table 2 shows their results for each Dilithium security level and function in terms of clock frequency, required clock cycles, and therefore resulting latency in µs. Their results can be seen in Table 2. As the authors report the latency in µs, we converted the latency into executions per seconds for comparability and included the results in Fig. 9 in dark red. The hardware implementation presented in [47] outperforms all software-based solutions presented in this work. Comparing Table 1 with Table 2 demonstrates that the implementation of Dilithium [47] yields more executions per second with respect to the implementation of Falcon presented in [32].

In [48] the authors report an implementation of Falcon 512, Falcon 1024, Dilithium 2, Dilithium 3 and Dilithium 5 on

[3]https://www.xilinx.com/products/boards-and-kits/zcu104.html#information

[4]https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html#information

TABLE 2. Number of clock cycles required per function reported in [47]. The clock frequency is indicated to be 270 MHz for all functions and security levels. A higher clock frequency combined with a lower required count of clock cycles per execution yields an overall lower latency per execution compared to the implementation of Falcon presented in [32].

security	Function	Clock Cycles	Latency (μs)	Clock (MHz)
Dil 2	key_gen	14.594	54.05	270
	sign	21.812	80.76	270
	verify	15.423	57.12	270
Dil 3	key_gen	23.619	87.48	270
	sign	36.643	135.72	270
	verify	26.124	96.76	270
Dil 5	key_gen	39.737	147.17	270
	sign	53.965	199.87	270
	verify	46.671	172.86	270
Kyb 512	key_gen	3.395	12.6	270
	encaps	4.956	18.4	270
	decaps	6.807	25.21	270
Kyb 768	key_gen	6.291	23.2	270
	encaps	7.862	29.11	270
	decaps	11.291	41.82	270
Kyb 1024	key_gen	9.089	33.7	270
	encaps	11.351	42.04	270
	decaps	31.905	51.5	270

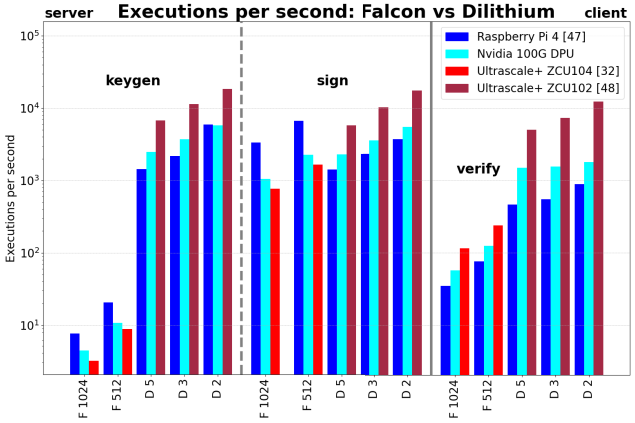

FIGURE 9. Comparison of implementation of the PQC signature algorithms Falcon and Dilithium on different hardware platforms: Raspberry Pi 4 [48], represented in blue, Nvidia's DPU in cyan, Falcon 512/1024 on a Xilinx Ultrascale+ ZCU 104 FPGA [32] in red, and Dilithium 2/3/5 on a Xilinx Ultrascale+ ZCU 102 FPGA [47] in dark red. F 512/1024 stands for Falcon 512/1024. D2/3/5 represent Dilithium 2/3/5. In our software stack the key generation and sign processes are performed by the server while the client executes the verification.

a Raspberry Pi 4 board[5] with an 1.5 GHz Quad-core ARM 64-bit processor equipped with 4 GB of RAM. As operating system a 64-bit Ubuntu 21.04 is installed. The authors report the execution time of the keypair generation, signature, and verification in ms. To achieve their results, the authors use the Open Quantum Safe[6] project with its C-library liboqs for their PQC examinations. In their work, the authors do not disclose the number of clock cycles that were required for each step. However, this value can be deduced using the processor's clock frequency divided over the number of executions per second. For the purpose of comparability, we have converted the results reported in [48] from latency in µs per execution into operations per second. The Raspberry Pi 4 performs well during Falcon's key generation according to the values denoted in [48] with respect to the Ultrascale+ ZCU104 FGPA and the DPU. Dilithium's key generation is harder for the Raspberry Pi 4 compared to the Ultrascale+ ZCU102 FPGA and the DPU. However, it outperforms the ZCU104 and the DPU during Falcon's signing procedure. According to the results shown in Fig. 9, Dilithium's signature as well as the verification for both algorithms, Falcon and Dilithium, appears to be harder for the Raspberry compared to the DPU and the Xilinx FPGAs.

[5]https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[6]https://openquantumsafe.org/

V. CONCLUSION

In this work, we presented a secure network stack for authentication and key exchange based on PQC algorithms and benchmarked experimentally two strong contenders to be used in deployed systems. As of now, Kyber remains the only KEM left in the NIST competition [23], and hence, it is included in both versions of the herein presented network stack: Falcon/Kyber and Dilithium/Kyber. As to the choice of signature algorithm added to the KEM, Dilithium exceeds Falcon in environments where resources are fairly available. E.g. in data centers the available network capacity through processing capabilities and hardware accelerators can easily provide for Dilithium's higher network throughput. Dilithium/Kyber is hence well positioned to be used in data center environments, together with SDP technologies and embedded with other CC technologies.

Falcon's strengths, however, lie within an environment where resources are restricted, devices' capabilities are limited as they are potentially battery powered, and the available bandwidth might be low. In this kind of setting, the server carrying out the majority of the computational tasks is an advantage.

The results highlight a byproduct of the current PQC algorithms development: multiple PQC algorithms are bound to co-exist, each operating on a different segment of the network depending on the processing power available and network constraints.

REFERENCES

[1] A. Parra-Rodriguez, P. Lougovski, L. Lamata, E. Solano, and M. Sanz, "Digital-analog quantum computation," *Phys. Rev. A, Gen. Phys.*, vol. 101, no. 2, Feb. 2020, Art. no. 022305.

[2] L. Crippa, F. Tacchino, M. Chizzini, A. Aita, M. Grossi, A. Chiesa, P. Santini, I. Tavernelli, and S. Carretta, "Simulating static and dynamic properties of magnetic molecules with prototype quantum computers," *Magnetochemistry*, vol. 7, no. 8, p. 117, Aug. 2021.

[3] P. Codognet, D. Diaz, and S. Abreu, "Quantum and digital annealing for the quadratic assignment problem," in *Proc. IEEE Int. Conf. Quantum Softw. (QSW)*, Jul. 2022, pp. 1–8.

[4] F. Hu, L. Lamata, C. Wang, X. Chen, E. Solano, and M. Sanz, "Quantum advantage in cryptography with a low-connectivity quantum annealer," *Phys. Rev. Appl.*, vol. 13, no. 5, May 2020, Art. no. 054062.

[5] M. Sharma, V. Choudhary, R. S. Bhatia, S. Malik, A. Raina, and H. Khandelwal, "Leveraging the power of quantum computing for breaking RSA encryption," *Cyber-Phys. Syst.*, vol. 7, no. 2, pp. 73–92, Apr. 2021.

[6] X. Bonnetain, M. Naya-Plasencia, and A. Schrottenloher, "Quantum security analysis of AES," *IACR Trans. Symmetric Cryptol.*, vol. 2019, no. 2, pp. 55–93, Jun. 2019.
[7] Y. Gong, J. Wang, and H. Yao, "Distributed multi-agent empowered resource allocation in deep edge networks," in *Proc. Int. Wireless Commun. Mobile Comput. (IWCMC)*, 2021, pp. 974–979.
[8] D. C. G. Valadares, N. C. Will, M. A. Spohn, D. F. de Souza Santos, A. Perkusich, and K. C. Gorgánio, "Confidential computing in cloud/fog-based Internet of Things scenarios," *Internet Things*, vol. 19, Aug. 2022, Art. no. 100543.
[9] A. Moubayed, A. Refaey, and A. Shami, "Software-defined perimeter (SDP): State of the art secure solution for modern networks," *IEEE Netw.*, vol. 33, no. 5, pp. 226–233, Sep. 2019.
[10] J. Singh, A. Refaey, and J. Koilpillai, "Adoption of the software-defined perimeter (SDP) architecture for infrastructure as a service," *Can. J. Electr. Comput. Eng.*, vol. 43, no. 4, pp. 357–363, Fall. 2020.
[11] M. Lefebvre, D. W. Engels, and S. Nair, "On SDPN: Integrating the software-defined perimeter (SDP) and the software-defined network (SDN) paradigms," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2022, pp. 353–358.
[12] A. Di Stefano, A. Di Stefano, and G. Morana, "Ananke: A framework for cloud-native applications smart orchestration," in *Proc. IEEE 29th Int. Conf. Enabling Technol., Infrastruct. Collaborative Enterprises (WETICE)*, Sep. 2020, pp. 82–87.
[13] B. Han, S. Wong, C. Mannweiler, M. Dohler, and H. D. Schotten, "Security trust zone in 5G networks," in *Proc. 24th Int. Conf. Telecommun. (ICT)*, May 2017, pp. 1–5.
[14] A. Hakiri and P. Berthou, *Leveraging SDN for the 5G Networks*. Hoboken, NJ, USA: Wiley, 2015, ch. 5, pp. 61–80.
[15] H. AlJahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, "Multi-tenancy in cloud computing," in *Proc. IEEE 8th Int. Symp. Service Oriented Syst. Eng.*, Apr. 2014, pp. 344–351.
[16] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang, and D. Meng, "Enabling rack-scale confidential computing using heterogeneous trusted execution environment," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1450–1465.
[17] K. Oztoprak, Y. K. Tuncel, and I. Butun, "Technological transformation of Telco operators towards seamless IoT edge-cloud continuum," *Sensors*, vol. 23, no. 2, p. 1004, Jan. 2023.
[18] U. Kumar, M. Garg, S. Kumari, and D. Dharminder, "A construction of post quantum secure and signal leakage resistant authenticated key agreement protocol for mobile communication," *Trans. Emerg. Telecommun. Technol.*, vol. 34, no. 1, p. e466, Jan. 2023.
[19] J. Jang, K. Tulkinbekov, and D.-H. Kim, "Task offloading of deep learning services for autonomous driving in mobile edge computing," *Electronics*, vol. 12, no. 15, p. 3223, Jul. 2023.
[20] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS–kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy*, Apr. 2018, pp. 353–367.
[21] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-dilithium: A lattice-based digital signature scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 238–268, Feb. 2018.
[22] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. *Fast-Fourier Lattice-based Compact Signatures Over NTRU*. Accessed: Oct. 15, 2023. [Online]. Available: https://falcon-sign.info/
[23] M. V. Yesina, Y. V. Ostrianska, and I. D. Gorbenko, "Status report on the third round of the NIST post-quantum cryptography standardization process," *Radiotekhnika*, no. 210, pp. 75–86, Sep. 2022.
[24] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," *J. ACM*, vol. 60, no. 6, pp. 1–35, Nov. 2013.
[25] D. Chaudhary, U. Kumar, and K. Saleem, "A construction of three party post quantum secure authenticated key exchange using ring learning with errors and ECC cryptography," *IEEE Access*, vol. 11, pp. 136947–136957, 2023.
[26] E. Karabulut and A. Aysu, "A hardware–software co-design for the discrete Gaussian sampling of falcon digital signature," *IACR Cryptol. ePrint Arch.*, vol. 2023, p. 908, May 2023.
[27] J. Howe, T. Oder, M. Krausz, and T. Güneysu, "Standard lattice-based key encapsulation on embedded devices," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, pp. 372–393, Aug. 2018.
[28] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "PQC acceleration using GPUs: FrodoKEM, NewHope, and kyber," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 575–586, Mar. 2021.
[29] N. Gupta, A. Jati, A. Chattopadhyay, and G. Jha. (2022). *Lightweight Hardware Accelerator for Post-Quantum Digital Signature Crystals-Dilithium*. Cryptol. ePrint Arch., Paper 2022/496. [Online]. Available: https://eprint.iacr.org/2022/496
[30] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl. (2022). *Post-Quantum Signatures on RISC-V With Hardware Acceleration*. [Online]. Available: https://eprint.iacr.org/2022/538
[31] F. Yaman, A. C. Mert, E. Öztürk, and E. Savas, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 1020–1025.
[32] M. Schmid, D. Amiet, J. Wendler, P. Zbinden, and T. Wei. (2023). *Falcon Takes Off—A Hardware Implementation of the Falcon Signature Scheme*. Cryptol. ePrint Arch., Paper 2023/1885. [Online]. Available: https://eprint.iacr.org/2023/1885
[33] Y. Kim, J. Song, and S. C. Seo, "Accelerating falcon on ARMv8," *IEEE Access*, vol. 10, pp. 44446–44460, 2022.
[34] D. T. Nguyen and K. Gaj, "Fast falcon signature generation and verification using ARMv8 NEON instructions," in *Proc. Int. Conf. Cryptol. Afr.*, 2023, pp. 412–441.
[35] G. Seiler, "Faster AVX2 optimized NTT multiplication for Ring-LWE lattice cryptography," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 39, Jul. 2018.
[36] A. C. Aguilera, X. A. I. Clemente, D. C. Lawo, I. T. Monroy, and J. J. V. Olmos, "First end-to-end PQC protected DPU-to-DPU communications," *Electron. Lett.*, vol. 59, no. 17, p. e1290, Sep. 2023.
[37] OpenSSL Project. (2022). *OpenSSL Toolkit*. OpenSSL Softw. Found. [Online]. Available: https://www.openssl.org/
[38] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. Schoen, and B. Warren, "Let's encrypt: An automated certificate authority to encrypt the entire web," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, Nov. 2019, pp. 2473–2487.
[39] T. Pornin. (Sep. 2020). *Fast-Fourier Lattice-Based Compact Signatures Over NTRU*. [Online]. Available: https://falcon-sign.info/impl/falcon.h.html
[40] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, G. Seiler, P. Schwabe, and D. Stehlé. (2021). *Pq-Crystals/Dilithium*. [Online]. Available: https://github.com/pq-crystals/dilithium
[41] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. (2021). *Pq-Crystals/Kyber*. [Online]. Available: https://github.com/pq-crystals/kybe
[42] *FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2015.
[43] K. Meher and D. Midhunchakkaravarthy, "New approach to combine secret keys for post-quantum (PQ) transition," *Indian J. Comput. Sci. Eng.*, vol. 12, no. 3, pp. 629–633, Jun. 2021.
[44] C. Gentry, C. Peikert, and V. Vaikuntanathan. (2007). *Trapdoors for Hard Lattices and New Cryptographic Constructions*. Cryptology ePrint Archive, Paper 2007/432. [Online]. Available: https://eprint.iacr.org/2007/432
[45] D. Soni, K. Basu, M. Nabeel, N. Aaraj, M. Manzano, and R. Karri, *Hardware Architectures for Post-Quantum Digital Signature Schemes*. Cham, Switzerland: Springer, 2021.
[46] M. Raavi, S. Wuthier, P. Chandramouli, Y. Balytskyi, X. Zhou, and S.-Y. Chang, "Security comparisons and performance analyses of post-quantum signature algorithms," in *Applied Cryptography and Network Security*, K. Sako and N. O. Tippenhauer, Eds. Cham, Switzerland: Springer, 2021, pp. 424–447.
[47] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "KaLi: A crystal for post-quantum security using kyber and dilithium," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 2, pp. 747–758, Feb. 2023.
[48] K. Jain and P. Krishnan, "Analysis of post-quantum cryptography for Internet of Things," in *Proc. 6th Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, May 2022, pp. 387–394.