ISSN NO : 2230-7850 IMPACT FACTOR : 5.1651 (UIF) VOLUME :9, ISSUE : 2, MARCH- 2019

INDIAN STREMS RESEARCH JOURNAL

Designing a Scalable and Area-Efficient Hardware Accelerator Supporting Multiple PQC Schemes

Dr. Rekha

Assistant professor in Electronics Government College (Autonomous)
Kalaburagi.

ABSTRACT:

In order to mitigate the threat posed by quantum computing to cryptographic security, this work presents a hardware accelerator to enable multiple Post-Quantum Cryptosystem (PQC) approaches. Even though PQCs are more secure, they also come with high computational requirements, which are problematic for lightweight devices in particular. It is inefficient that previous hardware accelerators are usually scheme-specific given that the National Institute of Standards and Technology (NIST) has several finalists. By concentrating on the common functions of these schemes, our method enables the simultaneous acceleration of several candidate PQCs by a single design. By distributing resources based on the findings of performance profiling, this is further improved. In comparison to the present state-of-the-art multi-scheme accelerator, our scalable and compact hardware accelerator supports four of the NIST PQC finalists, delivering an area efficiency of up to 81.85% while supporting twice as many

KEYWORDS: post-quantum security; kyber–dilithium; falcon; SPHINCS+; hardware accelerator.

1. INTRODUCTION

Cryptographic algorithms that are essential for system identification and authentication are the key exchange algorithm (KEA) and the digital signature method (DSA). Such classical cryptosystems have been implemented in a range of HW platforms, from low-end embedded/mobile devices [3] to high-end platforms [4], and have been employed as powerful security measures in a variety of domains, including the Internet of Things (IoT) [1] and autonomous industrial systems [2]. But the emergence of quantum computing, a well-known technology that has been propelling tremendous advancement, has greatly endangered classical cryptosystems. It was shown that they were susceptible to assaults from quantum computing systems [5], which made the development of novel cryptosystems with quantum-resistant architecture necessary.
NIST started the post-quantum cryptography (PQC) standardisation process in order to meet this demand.

A naive approach to supporting all four PQC finalist schemes would be to integrate four independent designs, each dedicated to one scheme. However, this would require excessive hardware area, limiting applicability across various platforms needed for wide- ranging fields. Our work proposes a design methodology enabling efficient implementation of all four schemes within hardware area constraints. This methodology is built on a com- prehensive analysis of the four PQC finalist schemes, aiming to create a flexible and efficient hardware design that adapts to various area constraints. We begin with performance profil- ing to identify computational *hotspots*—parts of each scheme where the most computational resources are used—and common operations across the schemes. This analysis reveals three key challenges: the diverse nature of polynomial operations, varying proportions of Keccak usage, and distinct high-level operation sequences among the schemes.

To address these challenges, our hardware design incorporates three main components: a scalable Keccak Acceleration Module (KAM), a versatile Joint Polynomial Arithmetic Unit (JPAU), and an efficient control unit. The KAM offers three variants to balance area and performance requirements, while the JPAU serves as a generic arithmetic unit capable ofhandling various polynomial operations common to all schemes. To manage the complexity of control flow, we implement a Unified Polynomial Control Unit (UPCU) separate from the main control unit, efficiently handling polynomial operations for all schemes. This modular and scalable approach allows for efficient resource utilization and performanceoptimization, achieving an area efficiency of up to 81.85% compared to the current state-of- the-art multi-scheme accelerator in [6], while supporting all four schemes instead of just two. Our evaluation shows an average throughput improvement ranging from 0.97 to 35.97 across the four schemes and three main operations, demonstrating the robustness and efficiency of our comprehensive design.

The remainder of this paper is organized as follows: Section 2 provides background information on post-quantum cryptography and detailed explanations of the four finalistschemes: Dilithium, Kyber, Falcon, and SPHINCS+. Section 3 discusses related works and outlines our motivation. In Section 4, we present our design methodology, includ- ing performance profiling and the proposed design architecture. Section 5 details the implementation of our design, while Section 6 provides a comprehensive evaluation of its performance. Finally, we conclude our work in Section 8, summarizing our contribu- tions and discussing potential future directions in the field of hardware acceleration for post-quantum cryptography.

2. Background

2.1. Post-Quantum Cryptography

Post-quantum cryptography (PQC) refers to cryptosystems that are considered secure against cryptanalytic attacks by quantum computers. Since 2016, NIST has been pursuing a PQC standardization program to select suitable schemes for key establishment and digital signature algorithms (KEAs and DSAs). Figure 1 depicts the general process of KEAs and DSAs. The KEA consists of three principal stages: key generation, encapsulation, and decapsulation. During the key generation stage, the receiver generates a pair of keys (public and secret) using Keygen() and broadcasts the public key. The sender, who wishes to send a message to the receiver, uses the public key to encapsulate the message using Encaps(), which the receiver decapsulates with the secret key using Decaps(). The DSA is composed of three stages: key generation, signature generation, and signature verification. The sender generates a pair of public and secret keys using Keygen(). With their private key, he generates a signature using Sign(), which the receiver can verify with the sender's public key using Verify(). The signature generation continues until a valid signature is produced. For a signature to be valid, it should satisfy a set of constraints to ensure that itdoes not convey similarity with the message.

When performing modular multiplication, reduction should follow the multiplicaton.In this case, the upper 48 bits of the output port are used, with these values stored in a temporary register outside and fed back to the JPAU for reduction. This design ensures accurate and efficient reduction operations, preventing overflow and maintaining consis-tency. Comparison operations can be performed by subtracting two data values, useful for condition checks such as rejection sampling or signature validation. The comparison result is outputted through a separated port.

Because each scheme uses different q values and coefficients for NTT, a Twiddle factor ROM is also attached to JPAU. This allows for flexible and accurate handling of various polynomial transformations needed for different algorithms. Since Kyber uses 12-bit q value and Dilithium uses the largest q value of 23 bits, we followed the approach of [6], which extends the ALU's datapath to 24 bits and computes four coefficients instead of two when using the Kyber scheme. This significantly increases throughput and utilization forKyber, optimizing the hardware for its specific requirements.

Each JPAU can perform coefficient-wise operations on two coefficients simultaneously, with each port receiving two coefficients from two different polynomials. Adding more JPAUs can further accelerate polynomial operations, enhancing overall computational efficiency. The JPAU is fully pipelined, maximizing throughput and minimizing latency by ensuring that multiple stages of computation can be processed concurrently without waiting for previous stages to complete. This pipelining is crucial for maintaining high performance across the supported cryptographic schemes.

Control Unit

The control unit is responsible for sending commands to JPAU and Keccak modules, as well as managing memory and MUX addresses. It is implemented as a large FSM with states for each scheme. Building a separate FSM for each scheme can result in a significant area overhead, due to the need to construct separate states for each of the four schemes.This can lead to a large state register and also delays in control signal paths.

To overcome this problem, we designed a Unified Polynomial Control Unit (UPCU) separate from main control unit. Figure 7 shows the diagram for main control unit and UPCU. The main control unit handles the high-level control flow for each scheme, includ-ing initializing operations and managing the overall sequence of tasks. For instance, in the Dilithium_sign operation, the control unit starts by initializing and performing the SHAKE256 operation, then moves to Keccak operations and matrix expansion. Similarly, for Falcon_sign, it handles random sampling and then proceeds to polynomial multiplication.

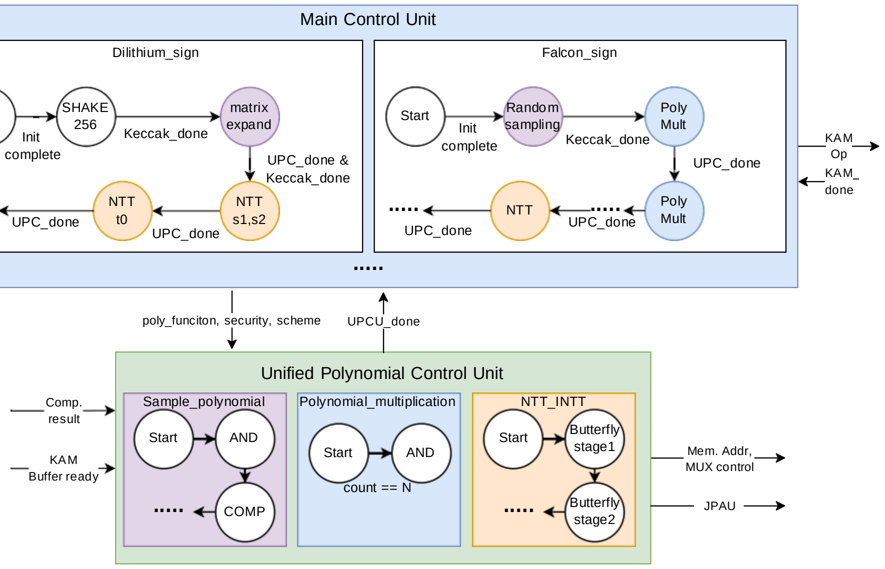
Main Control Unit
Dilithium_sign
Start
Init complete
SHAKE 256
Keccak_done
matrix expand
UPC_done & Keccak_done
NTT s1,s2
UPC_done
NTT t0
UPC_done
.....
Falcon_sign
Start
Init complete
Random sampling
Keccak_done
Poly Mult
UPC_done
Poly Mult
.....
UPC_done
NTT
UPC_done
.....
.....
KAM Op
KAM_ done
poly_funciton, security, scheme
UPCU_done
Unified Polynomial Control Unit
Comp. result
KAM Buffer ready
Sample_polynomial
Start
AND
COMP
.....
Polynomial_multiplication
Start
AND
count == N
NTT_INTT
Start
Butterfly stage1
Butterfly stage2
.....
Mem. Addr, MUX control
JPAU

Figure 7. Diagram of control unit with UPCU.

When a JPAU operation is needed, instead of main control unit sending all JPAU opcodes and MUX control signals, it sends a predefined polynomial function code to the UPCU. The UPCU then takes the function code along with information such as the schemeand security level and starts sending the appropriate JPAU opcodes and SRAM memory addresses. The UPCU adjusts parameters such as *N* for each scheme and security level, eliminating the need to create separate control sequence for each scheme. This *segregation* of detailed polynomial control to the UPC minimizes the FSM complexity in the main control unit. This design ensures that the control logic is streamlined and efficient, capable of handling various polynomial operations without excessive state overhead. The detailed operation of the UPCU can be summarized as follows:

Sample_polynomial. The UPCU initiates and manages the polynomial sampling process. This includes setting up necessary registers and handling data flow for efficient sampling. Polynomial_multiplication. The UPCU controls the sequence of multiplication and accumulation operations, coordinating data flow and setting up operands for the computation.

NTT_INTT. The UPCU manages the NTT and INTT operations, controlling the butterfly units and Montgomery reduction units. It ensures efficient operations by adjusting control signals and managing data flow through various stages, utilizing the Twiddle factor ROMfor different schemes.

By implementing these processes within the UPCU separately, the complexity of the overall FSM is significantly reduced, leading to higher area efficiency. This approach allows the control unit to handle the operations of all four PQC schemes without incurring a large area overhead, thus enhancing the overall performance and efficiency of the hardware design.

5. Implementation

We synthesized our design using Design Compiler N-2017.09-SP2 [27] with 15 nm Opencell library [28]. We used kGE as a metric to ensure a fair comparison across different silicon processes, as it normalizes the differences in technology nodes. This standardization allowed us to compare designs more effectively, regardless of the specific fabrication technology used.

Our target kGE (kilo Gate Equivalent) was set to 600 kGE, which represents the sum of kGE values from various reported works, as there is no single reference implementa- tion supporting all four PQC finalist schemes. This target was determined by combining the most efficient individual implementations for each scheme to achieve the minimum possible kGE sum. Specifically, we considered Gupta et al. [19]'s 157 kGE for Dilithium, Bisheh-Nisar et al. [23]'s 93 kGE for Kyber, Lee et al. [18]'s 98.729 kGE for Falcon verifica- tion and Soni et al. [29]'s 181.120 kGE for Falcon-1024 signing, and Wagner et al. [20]'s 84 kGE for SPHINCS+. Adding these values results in a total of 613.849 kGE. However, we conservatively set our target to 600 kGE to provide a more challenging and aggressive goal, ensuring a more efficient and streamlined design. It is worth noting that no implementation for Falcon's key generation was available, so this function's kGE was not included in our target kGE. Additionally, while Aikata et al. [6] implemented both Dilithium and Kyber, their reported 747 kGE was deemed too high. Thus, we opted for the combination of separate implementations by Gupta et al. [19] for Dilithium and Bisheh-Nisar et al. [23] for Kyber, as this resulted in a lower total kGE target. This approach allowed us to set a realistic and competitive target for our unified implementation of all four PQC finalist schemes.

We first built a small baseline design, **Ours_Baseline,** with 4 JPAU and KAM-Small variant, and extended our design by changing KAM-Small to KAM-Large to build **Ours_S** variant. Our first priority in scaling up was the JPAU cluster, which generally affects each scheme's performance. After scaling up the JPAU by 4 to 8 , we built the **Ours_M** variant; we then checked whether we had margin left. If we had spare resources, we could allocate more to use faster KAM modules. By changing to the KAM-FP, we built the **Ours_L** variant, which resulted in 611.389 kGE, which satisfied our target kGE. Table 6 shows synthesis results of our proposed design compared with other designs.

Table 6. Synthesis results compared with other works.

	Technolog	Clock Frequency	Area (mm^2)	kGE	Target kGE	Target Scheme
Ours_Baseline	15nm	1000MHz	0.056	284.939	-	Dilithium, Kyber, SPHINCS+, Falcon(Peregrine)
Ours_S	15 nm	1000 MHz	0.062	315.743	300	Dilithium, Kyber, SPHINCS+, Falcon(Peregrine)
Ours_M	15 nm	1000 MHz	0.115	584.624	613.849	Dilithium, Kyber, SPHINCS+, Falcon(Peregrine)
Ours_L	15 nm	1000 MHz	0.120	611.389	613.849	Dilithium, Kyber, SPHINCS+, Falcon(Peregrine)
Gupta et al. [19]	65 nm	1176 MHz	0.227	157.000	-	Dilithium
Aikata et al. [14]	65 nm	400 MHz	0.317	220.000	-	Dilithium, Saber
Aikata et al. [6]	28 nm	1000 MHz	0.263	747.000	-	Dilithium, Kyber
Wagner et al. [20]	120 nm	250 & 500 MHz	0.560	84.000	-	SPHINCS+
Wagner et al. [20] extended	120 nm	250 & 500MHz	0.476	98.800	-	SPHINCS+
Lee et al. [18]	28 nm	300 MHz	0.038	98.729	-	Falcon(Verification) (1)
Soni et al. [29] 512	65 nm	122 MHz	0.387	184.300	-	Falcon(Signing) (2)
Soni et al. [29] 1024	65 nm	173 MHz	0.380	181.120	-	Falcon(Signing) (2)
Bisheh-Nisar et al. [23]	65 nm	200 MHz	N/A	93	-	Kyber

(1) Implemented verification algorithm in Falcon. (2) Implemented signing algorithm in Falcon.

6. Evaluation

We compared the performance of our design variants presented in Section 5, namely **Ours_S**, **Ours_M**, and **Ours_L** with prior works.

To compare with other works, we compared our design with FoM (figure-of-merit) defined in [30] as shown below:

$$FoM = Throughput/Area = Throughput/kGE$$

This factor can show the area efficiency of the design. We also used kGE count instead of area for a fair comparison of accelerators with other technologies.

Parameters used for evaluation of each PQC scheme are listed in Tables 2 and 4. In cases where no prior HW implementation exists for certain operations (e.g., key generation for Falcon and SPHINCS+), we used the performance of CPU implementations with AVX extensions as a baseline for comparison. These CPU cycle counts, reported in the NIST reference submissions, are converted to throughput numbers to provide a point of reference. For ease of comparison and representations, we present the performance results for Dilithium and Kyber in a single subsection, as they were both implemented in the work by Aikata et al. [6]. The results for Falcon and SPHINCS+ are discussed in separate subsections due to their unique implementation characteristics and the lack ofcomprehensive HW implementations for all operations. This approach allows us to provide a comprehensive comparison across all schemes and operations, even in cases where direct hardware implementation comparisons are not available. It also enables us to highlight the advantages of our unified design across different PQC algorithms.

6.1. Dilithium and Kyber

In Dilithium and Kyber, we compared our design with current state-of-the-art ASIC ac- celerators that support more than two different parameters, namely Aikata et al. [14], Aikata et al. [6], and the state-of-art Dilithium ASIC accelerator, Gupta et al. [19]. Table 7 showsthe normalized throughput of our variants on Dilithium and Kyber compared to other accelerators. The throughput is calculated in the same manner as benchmarks in the NIST submission package, which performs Keygen, Sign/Encapsulate, and Verify/Decapsulateon each security parameter.

Table 7. Relative throughput and FoM on Dilithium and Kyber.

	Parameter	Gupta et al. [19]		Aikata et al. [14]		Akata et al. [6]		Ours_S		Ours_M		Ours_L	
		Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM
Keygen	Dilithium2	-	-	0.52	0.75	1.27	0.54	1.00	1.00	1.74	0.94	2.09	1.08
	Dilithium3	-	-	0.57	0.82	1.39	0.59	1.00	1.00	1.76	0.95	2.08	1.07
	Dilithium5	1.11	2.23	0.61	0.88	1.50	0.63	1.00	1.00	1.77	0.96	2.08	1.07
	Kyber512	-	-	-	-	4.66	1.97	1.00	1.00	1.04	0.56	2.18	1.12
	Kyber768	-	-	-	-	3.47	1.47	1.00	1.00	1.04	0.56	2.26	1.17
	Kyber1024	-	-	-	-	3.08	1.30	1.00	1.00	1.04	0.56	2.31	1.19
Sign	Dilithium2	-	-	0.96	1.38	2.31	0.98	1.00	1.00	1.90	1.03	2.01	1.04
	Dilithium3	-	-	1.08	1.55	2.63	1.11	1.00	1.00	1.92	1.04	2.01	1.04
	Dilithium5	2.39	4.81	1.35	1.94	3.30	1.40	1.00	1.00	1.93	1.04	2.01	1.04
	Kyber512	-	-	-	-	2.85	1.20	1.00	1.00	1.07	0.58	2.74	1.42
	Kyber768	-	-	-	-	2.57	1.09	1.00	1.00	1.07	0.58	2.71	1.40
	Kyber1024	-	-	-	-	2.34	0.99	1.00	1.00	1.07	0.58	2.68	1.39
Verify	Dilithium2	-	-	0.83	1.19	2.02	0.85	1.00	1.00	1.83	0.99	2.00	1.03
	Dilithium3	-	-	0.85	1.22	2.08	0.88	1.00	1.00	1.85	1.00	2.01	1.04
	Dilithium5	1.71	3.44	0.86	1.24	2.11	0.89	1.00	1.00	1.86	1.01	2.03	1.05
	Kyber512	-	-	-	-	2.26	0.95	1.00	1.00	1.11	0.60	2.65	1.37
	Kyber768	-	-	-	-	1.96	0.83	1.00	1.00	1.11	0.60	2.61	1.35
	Kyber1024	-	-	-	-	2.10	0.89	1.00	1.00	1.11	0.60	2.57	1.33

For Dilithium, due to significant loads on matrix generation, changing the Keccakmodule to the KAM-FP variant can improve performance by up to 8%. Compared withAikata et al. [14], who accelerated both Dilithium and Saber schemes, our **Ours_M** and**Ours_L** variants achieved a 3.11 and 3.69 speedup on Keygen, a 1.73 and×1.81 speed×up on Sign, and a 2.18 × and 2.38× speedup on Verify, on average, with 2.65 × and × × 2.77 larger kGE counts, an×d 1.44 and 1.07 larger FoM ×on average× respectively. **Ours_S** variant also achieved a speedup of 1.76 and 1.17 average on Keygen and Verify, whi×le having 0×.87 lower throughput on Sign, 1.43 larger kGE counts, and 0.76 low×er FoM on average. × ×

Comparing to Aikata et al. [6], the current state-of-the-art implementation for accel-erating both Dilithium and Kyber schemes, our **Ours_M** and **Ours_L** variants achieved an average throughput increase of 1.51and 1.27 in Keygen, respectively. In Sign, ourvariants had a slightly×lower throu×ghput of 0.75and 0.71 , and, in Verify, the throughput was 0.89 and 0.97 lower, on average, compa×red to Aika×ta et al. [6]. However, **Ours_M** and **Ours_L** variants had signi×ficantly low×er kGE counts, which were 0.27 and 22 lower than those of Aikata et al. [6], which resulted in 2.21 and 1.65 higher FoM averaged×onDilithi×um. **Ours_S** variant had a lower throughput of 0.72 , 0.37 , and 0.48 ave×rage onKey×gen, Sign, and Verify, respectively, while having 2.63 smaller kGE counts with 1.17 larger FoM. × × ×

× ×

For Kyber, each JPAU in our design can perform operations on four coefficients simultaneously, significantly reducing the time spent using the JPAU and increasing the proportion of time dedicated to the KAM. Replacing the KAM from KAM-Large to KAM-FP when transitioning from **Ours_M** to **Ours_L** variant significantly increased the throughputin Kyber. **Ours_S** and {**Ours_M**, **Ours_L**} variants showed an average of 0.72 , 0.29 , and × ×
0.62 lower throughput i×n Keygen. In Encapsulate, the **Ours_S** and {**Ours_M**, **Ours_L**} variants showed differences of 0.38 ,0.42 , and 1.05 , and, on Decapsula×te, 0.4×7 ,0.53 , a×nd 1.24 , respectively, comp×ared ×to Aikata et al. [6]. This is beca×use our design focused on breaking down each function to maximize shared functions, whereas Aikata et al. [6] focused on accelerating Kyber by consuming as many hardware resources as possible.

6.2. Falcon

Table 8 shows our speedup factor on Falcon [12] compared with other Falcon accelera- tors implemented on ASIC. Since it is difficult to find works that have implemented the full scheme in ASIC, we compared each functionality (namely Keygen, Sign, and Verify) with existing accelerators including SW implementation on CPU with AVX extensions reported in [9]. Our accelerator achieved a speedup of 8.06 , 14.76 , and 14.76 on Keygen, 4.03 , × × × ×
6.49 , and 6.49 on Sig×n, and 7.87× , 15.66 , and 1×5.76 o×n Verificatio×n compared to CPU with AVX extensions **Ours_S** and {**Ours_M**, **Ours_L**} variants, respectively. Since the Keccak operation accounts for about 0.1% of total Falcon operations, using KAM-FP almost does not affect the overall performance. Our accelerator outperformed prior works implemented in ASIC in all three variants with more than 100× speedups and more than
50× FoM.

Table 8. Relative throughput and FoM on Falcon.

	Parameter	CPU[AVX]	Lee et al. [18]		Soni et al. [29]		Ours_S		Ours_M		Ours_L	
		Thrpt.	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM
Keygen	Falcon512	0.15	-	-	-	-	1.00	1.000	1.82	0.983	1.82	0.940
	Falcon1024	0.11	-	-	-	-	1.00	1.000	1.84	0.992	1.84	0.948
Sign	Falcon512	0.24	0.002	0.006	-	-	1.00	1.000	1.60	0.863	1.60	0.826
	Falcon1024	0.25	0.002	0.006	-	-	1.00	1.000	1.62	0.874	1.62	0.836
Verify	Falcon512	0.12	-	-	0.01	0.015	1.00	1.000	1.98	1.072	2.00	1.034
	Falcon1024	0.13	-	-	0.01	0.015	1.00	1.000	1.99	1.076	2.00	1.033

6.3. SPHINCS+

In SPHINCS+, since the current state-of-art design is on FPGA, we compared our designs with both FPGA-based work and existing ASIC implementations in Table 9. When attempting to obtain a FoM comparable to other works, direct comparisons between existing SPHINCS+ implementations become complicated due to the significant differences between FPGA and ASIC technologies. The metrics used to evaluate the FPGA area (such as the number of LUTs or FFs) do not directly translate to ASIC metrics like kGE, making it challenging to establish a consistent basis for comparison. We also included a comparison of CPU performance with AVX extensions as reported in [10] as the baseline for Keygen, since no prior work has targeted the Keygen of SPHINCS+. SPHINCS+'s performance is highly dependent on the throughput of KAM. Due to heavy use of SHAKE256 hash functions for tree hashing, utilizing KAM-Large and KAM-FP dramatically increases overall throughput.

Table 9. Relative throughput and FoM on SPHINCS+.

Parameter	CPU(AVX)	Wagner et al. [20]		Wagner et al. [20] extended		Amiet et al. [31]		Ours_S		Ours_M		Ours_L	
	Thrpt.	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM	Thrpt.	FoM

Keygen	256s-simple	0.05	-	-	-	-	-	-	1.00	1.000	1.00	0.540	2.95	1.522
	256s-robust	0.04	-	-	-	-	-	-	1.00	1.000	1.00	0.540	2.97	1.535
Sign	256s-simple	0.03	-	-	-	-	0.82	-	1.00	1.000	1.00	0.540	2.95	1.522
	256s-robust	0.03	-	-	-	-	0.83	-	1.00	1.000	1.00	0.540	2.97	1.535
Verify	256s-simple	0.01	0.03	0.104	0.04	0.135	0.06	-	1.00	1.000	1.00	0.540	2.58	1.330
	256s-robust	0.01	0.02	0.077	0.04	0.131	0.08	-	1.00	1.000	1.00	0.540	2.59	1.336

Since there are many parameters in SPHINCS+, we selected the most time-consuming parameters, *256s-simple* and *256s-robust*, which are also implemented in [20]. The parameters are represented in Table 4. It should be noted that the robust parameter adds an extra layer of SHAKE to generate bitmasks and XOR uses the bitmask to the input when hashing each input. **Ours_S** and **Ours_M** variant outperform CPUs with a speedup of 33.3× in signing signatures. **Ours_L** variant, which has KAM-FP, can further accelerate up to 99×. Compared with the current state-of-art FPGA implementation in [31], our design has 1.2× and 3.6× higher throughput on {**Ours_S**, **Ours_M**} and **Ours_L** variants, respectively.

6.4. Power Consumption

Table 10 presents the power consumption of our accelerator and the energy used for each scheme compared with other accelerators. For a fair comparison, we used the Cadence Genus tool for power analysis, following the method used in Aikata et al. [6]. Power consumption represents the rate of energy use, typically measured in watts, and is obtained from the Genus tool. Energy consumption, on the other hand, is calculated by multiplyingthe power consumption by the execution time, providing a measure of the total energy used for a specific operation or set of operations. Our design achieved significantly lowerpower consumption compared to Aikata et al. [6], which used a larger design resulting in higher performance. Specifically, our design consumed 32.73 , 20.93 , and 20.34 less power for **Ours_S**, **Ours_M**, and **Ours_L** variants, respectively. For Dilithium, in terms of energy used for Sign/Verify combined, our variants **Ours_S**, **Ours_M**, and **Ours_L** were 13.45 , 16.30 , and 16.81 more efficient than Aikata et al. [6]. Furthermore, For Kyber, with Encapsulate/Decapsulate operations combined, our variants **Ours_S**, **Ours_M**, and **Ours_L** were 15.11×, 10.58×, and 24.67× more efficient than Aikata et al. [6].

Table 10. Power consumption compared with other accelerators.

	Dilithium3 (μJ)	**Kyber1024 (μJ)**	**SPHINCS + 256s (μJ)**	**FALCON1024 (μJ)**
Ours_S	2.01	0.61	0.174	0.10
Ours_M	1.66	0.88	0.272	0.08
Ours_L	1.61	0.38	0.095	0.08
Aikata et al. [6]	27.00	9.27	-	-
Lee et al. [18]	-	-	-	27.60
Amiet et al. [31]	-	-	189,300	-

For Falcon, we compared our design with Lee et al. [18], focusing on the energy used during Verify operations. Due to our larger design with more gate counts, our variants consumed more power: 1.89 , 2.97 , and 3.05 for **Ours_S**, **Ours_M**, and **Ours_L**, re- spectively, but consumed 276 less energy when performing Verify. However, by utilizing modular arithmetic instead of complex double-precision floating-point operations [12], we achieved significantly higher energy efficiency: 206.51 , 263.17 , and 256.82 for our three variants.

Regarding SPHINCS+, we compared our design with Amiet et al. [31], which reported power consumption for an FPGA accelerator platform. We chose this FPGA implementation for comparison due to the absence of other ASIC implementations in the literature, despite the platform difference. This state-of-the-art FPGA design served as our benchmark. It is important to note that, while this comparison provides valuable insights, the fundamental differences between ASIC and FPGA platforms should be considered when interpreting the following results. Our design variants **Ours_S**, **Ours_M**, and **Ours_L** consumed 1,087,931 , 695,955 , and 1,992,631 less energy, respectively, on the SPHINCS + 256s-simple param- eter. This substantial difference in power consumption can be largely attributed to the inherent differences between our PQC-specific ASIC implementation and the more general- purpose nature of FPGA platforms. Additionally, our design achieved up to 3.6 shorter processing time on SPHINCS+, further contributing to this significant energy efficiency gap.

7. Discussions

7.1. Architectural Differences against Others

Our design is unique in that it supports all four PQC finalist schemes by breaking down each scheme's operations into fundamental components and building a generalized ALU capable of handling these operations across different schemes. As discussed in Section 3, unlike prior works that implemented accelerators for only one or two schemes, we focused on creating a flexible and efficient hardware design from scratch, capable of performing fast computations for all four schemes. To support four different schemes, we analyzed the detailed operations of each algorithm and constructed a generalized ALU that can be utilized across multiple schemes. A similar approach can be seen in the work by [16], which implemented a coprocessor based on RISC-V architecture for widely used tasks such as NTT. However, our design takes a processor-like approach that is more customized for PQC needs, as opposed to a general CPU pipeline structure. Our design operates under a specialized control unit tailored for PQC, allowing it to efficiently support all four schemes with a structure specifically optimized for these cryptographic tasks, rather than a general-purpose architecture. This approach ensures that our hardware is not just a general solution, but a highly specialized one for the unique requirements of PQC.

7.2. Security and Reliability

Recent research has demonstrated side-channel and fault-injection vulnerabilities in existing cryptosystems [32,33], with similar vulnerabilities reported for NIST-selected PQC schemes in hardware accelerator implementations [31,34]. While our research primarily focused on developing an efficient hardware implementation without specific protection methods currently applied, we considered various security solutions as orthogonal workthat could be integrated into our design. For instance, to counter fault injection attacks that create glitches by adjusting power supply voltage [35], our design architecture allows for the potential duplication of parts like JPAU and KAM, enabling result comparison for enhanced defense [31]. Masking techniques to randomize intermediate values and prevent secret leakage [36] could also be incorporated, although this presents challenges for lattice- based schemes due to their complex operations and rejection sampling. Existing works proposed efficient masking techniques for these schemes [37,38] that could be also applied to our design. Our flexible architecture also allows for the potential implementation of additional masking operations during function computations, albeit at the cost of extra clock cycles. Alternatively, trusted execution like TrustZone may be utilized to provide a more secure environment, but it suffers from some vulnerability issues, as witnessed in [39,40]. Our design was implemented as an isolated IP providing only fixed interfaces specific to PQC operations. While TrustZone aims to separate general execution environments, our approach focused exclusively on cryptographic operations with a limited set of dedicated interfaces. This specialization potentially results in a smaller attack surface compared to more general-purpose secure execution environments, which may offer additional security benefits in the context of post-quantum cryptographic operations. However, a comprehensive analysis of this security aspect is beyond the scope of this paper and remainsan important area for future research.

7.3. Limitations and Future Works

Our design, while supporting multiple PQC schemes, has several limitations. Firstly, the absence of optimal performance for a single scheme is a limitation, as our accelerator primarily focuses on optimizations for multiple algorithms, making it challenging to achieve the best performance for any one scheme. Secondly, the performance ceiling of the Keccak Acceleration Module (KAM) is a concern. Due to the limited scalability of the Keccak algorithm, while the number of Joint Polynomial Arithmetic Units (JPAUs) can be scaled extensively, the KAM cannot be similarly scaled, leading to a performance bottleneck. Additionally, our design relies heavily on the Peregrine algorithm modifications to Falcon to avoid using double-precision floating-point operations. However, these modifications have not been extensively tested over a long period. Despite these limitations, experiments show that our design achieves throughput comparable to or sometimes better than other single- scheme accelerators, making it a viable option. In future works, we plan to support the original Falcon algorithm by integrating double-precision floating-point operations into our JPAUs and improve scalability by utilizing multiple KAM modules. We also aim to explore advanced optimization techniques to further enhance the performance for individual schemes and to develop more robust testing methodologies to ensure the reliability of our design over time. Furthermore, integrating adaptive algorithms to dynamically allocate resources based on workload could provide better performance balance across different PQC schemes.

8. CONCLUSIONS

The advent of quantum computing poses a significant threat to classical cryptosystems, creating a need for post-quantum cryptography (PQC). In response, NIST announced four schemes for standardization, each with its own advantages and disadvantages, such as Falcon's short signature length. Prior works have primarily focused on accelerating individual schemes, making integration with other schemes challenging. We presented a scalable accelerator designed to support all four NIST-selected algorithms, incorporatinga modified version of Falcon. Through function-level profiling, we identified common operations shared between schemes and designed each component of our accelerator accordingly. Our design achieved nearly the same throughput compared with state-of-the- art multi-scheme accelerators with small area overhead on Dilithium and Kyber and also achieved significant speedup compared other single-scheme accelerators on Falcon and SPHINCS+. Overall, our design provides a general speedup across all four NIST-selected schemes, demonstrating its effectiveness and versatility in addressing the challenges posed by quantum computing to cryptographic systems.

Funding: This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. RS-2022-00166529) , Korea Planning & Evaluation Institute of Industrial Technology(KEIT) grant funded by the Korea Government(MOTIE) (No. RS-2024-00406121, Development of an Automotive Security Vulnerability-based Threat Analysis System(R&D)) , National Research Foundation of Korea(NRF) grant funded by the Korea govern- ment(MSIT) (RS-2023-00277326) , Institute of Information & communications Technology Planning & Evaluation (IITP) under the artificial intelligence semiconductor support program to nurture thebest talents (IITP-2023-RS-2023-00256081) grant funded by the Korea government(MSIT) , Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.RS-2023-00277060, Development of open edge AI SoC hardware and software platform, 0.1) and Korea Evaluation Institute of Industrial Technology(KEIT) grant funded by the Korea government(MOTIE) (No.RS-2023-00277060, Development of open edge AI SoC hardware and software platform, 0.1). , MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Tech- nology Research Center) support program(IITP-2023-2020-0-01602) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) , Inter-University Semicon- ductor Research Center (ISRC) , BK21 FOUR program of the Education and Research Program for Future ICT Pioneers, Seoul National University in 2024. The EDA tool was supported by the IC Design Education Center(IDEC), Korea.

Data Availability Statement: The data used for experimental comparisons in this study, referring to the comparison figures, can be found in related research papers. Our hardware implementation code is protected under the proprietary rights of the funding project's institution and therefore cannot be made publicly available.

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

Abbreviations

The following abbreviations are used in this paper.

AVX	Advanced Vector eXtension
ASIC	Application Specific Integrated Circuit
ALU	Arithmetic and Logical Unit
DSE	Design Space Exploration
DSA	Digital signature algorithm
XMSS	eXtended Merkle Signature Scheme
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSM	Finite-State Machine
GE	Gate Equivalent
GPV	Gentry–Peikert–Vaikuntanathan
HW	Hardware
IoT	Internet of Things
INTT	Inverse Number Theoretic Transform
JPAU	Joint Polynomial Arithmetic Unit
KALU	Kecakk ALU
KAM	Keccak Acceleration Module
KEA	Key exchange algorithm
NIST	National Insitute of Standards and Technology
NTRU	Number Theory Research Unit
NTT	Number Theroetic Transform
PQC	Post-Quantum Cryptosystem
pk	Public Key
sk	Secret Key
SW	Software
UPCU	Unified Polynomial Control Unit
WOTS	Winternitz One-Time Signature

REFERENCES

1. Carracedo, J.M.; Milliken, M.; Chouhan, P.K.; Scotney, B.; Lin, Z.; Sajjad, A.; Shackleton, M. Cryptography for Security in IoT. In Proceedings of the 2018 Fifth International Conference on Internet of Things: Systems, Management and Security, Valencia, Spain, 15–18 October 2018; pp. 23–30. https://doi.org/10.1109/IoTSMS.2018.8554634.
2. Katzenbeisser, S.; Polian, I.; Regazzoni, F.; Stöttinger, M. Security in Autonomous Systems. In Proceedings of the 2019 IEEE European Test Symposium (ETS), Baden-Baden, Germany, 27–31 May 2019; pp. 1–8. https://doi.org/10.1109/ETS.2019.8791552.
3. Muzikant, P.; Willemson, J. Deploying Post-quantum Algorithms in Existing Applications and Embedded Devices. In *Proceedings of the Ubiquitous Security*; Wang, G., Wang, H., Min, G., Georgalas, N., Meng, W., Eds.; Springer: Singapore, 2024; pp. 147–162.
4. Kim, D.; Choi, H.; Seo, S.C. Parallel Implementation of SPHINCS+ With GPUs. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 2810–2823. https://doi.org/10.1109/TCSI.2024.3370802.
5. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Rev.* **1999**, *41*, 303–332.

6. Aikata, A.; Mert, A.C.; Imran, M.; Pagliarini, S.; Roy, S.S. KaLi: A Crystal for Post-Quantum Security Using Kyber and Dilithium. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 747–758. https://doi.org/10.1109/TCSI.2022.3219555.
7. Avanzi, R.; Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.M.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Kyber: Algorithm Specifications and Supporting Documentation, Submission to the NIST Post-Quantum Project. 2021. Available online: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf (accessed on 7 August 2024).
8. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation, Submission to the NIST Post-Quantum Project. 2021. Available online: https: //pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf (accessed on 7 August 2024).
9. Fouque, P.A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU, Specification v1.2. 2020. Available online: https: //falcon-sign.info/falcon.pdf (accessed on 7 August 2024).
10. Aumasson, J.P.; Bernstein, D.J.; Beullens, W.; Dobraunig, C.; Eichlseder, M.; Fluhrer, S.; Gazdag, S.L.; Hülsing, A.; Kampanakis, P.; Kölbl, S.; et al. SPHINCS+ Specification. Submission to the NIST Post-Quantum Project. 2020. Available online: https://sphincs.org/data/sphincs+-r3.1-specification.pdf (accessed on 7 August 2024).
11. NIST. Selected Algorithms 2022, July 2022. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography/selected-algorithms-2022 (accessed on 7 August 2024).
12. Seo, E.Y.; Kim, Y.S.; Lee, J.W.; No, J.S. Peregrine: Toward Fastest FALCON Based on GPV Framework. Cryptology ePrint Archive. 2022. Available online: https://eprint.iacr.org/2022/1495 (accessed on 7 August 2024).
13. Bernstein, D.J.; Hopwood, D.; Hülsing, A.; Lange, T.; Niederhagen, R.; Papachristodoulou, L.; Schneider, M.; Schwabe, P.; Wilcox-O'Hearn, Z. SPHINCS: Practical Stateless Hash-Based Signatures. In *Proceedings of the Advances in Cryptology–EUROCRYPT 2015*; Oswald, E.; Fischlin, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 368–397.
14. Aikata, A.; Mert, A.C.; Jacquemin, D.; Das, A.; Matthews, D.; Ghosh, S.; Roy, S.S. A Unified Cryptoprocessor for Lattice-Based Signature and Key-Exchange. *IEEE Trans. Comput.* **2023**, *72*, 1568–1580. https://doi.org/10.1109/TC.2022.3215064.
15. Basso, A.; Bermudo Mera, J.M.; D'Anvers, J.P.; Karmakar, A.; Sinha Roy, S.; Van Beirendonck, M.; Vercauteren, F. SABER: Mod-LWR Based KEM (Round 3 Submission) SABER Submission Package for Round 3. 2017. Available online: https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf (accessed on 7 August 2024).
16. Lee, J.; Kim, W.; Kim, J.H. A Programmable Crypto-Processor for National Institute of Standards and Technology Post-Quantum Cryptography Standardization Based on the RISC-V Architecture. *Sensors* **2023**, *23*, 9408. https://doi.org/10.3390/s23239408.
17. Nguyen, T.H.; Kieu-Do-Nguyen, B.; Pham, C.K.; Hoang, T.T. High-Speed NTT Accelerator for CRYSTAL-Kyber and CRYSTAL-Dilithium. *IEEE Access* **2024**, *12*, 34918–34930. https://doi.org/10.1109/ACCESS.2024.3371581.
18. Lee, Y.; Youn, J.; Nam, K.; Jung, H.H.; Cho, M.; Na, J.; Park, J.Y.; Jeon, S.; Kang, B.G.; Oh, H.; et al. An Efficient Hardware/Software Co-Design for FALCON on Low-End Embedded Systems. *IEEE Access* **2024**, *12*, 57947–57958. https://doi.org/10.1109/ACCESS.2024.3387489.
19. Gupta, N.; Jati, A.; Chattopadhyay, A.; Jha, G. Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 3234–3243. https://doi.org/10.1109/TCSI.2023.3274599.
20. Wagner, A.; Oberhansl, F.; Schink, M. To Be, or Not to Be Stateful: Post-Quantum Secure Boot using Hash-Based Signatures. In Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security, Los Angeles, CA, USA, 11 November 2022; ASHES'22; pp. 85–94. https://doi.org/10.1145/3560834.3563831.
21. Mandal, S.; Roy, D.B. KiD: A Hardware Design Framework Targeting Unified NTT Multiplication for CRYSTALS-Kyber and CRYSTALS-Dilithium on FPGA. In Proceedings of the 2024 37th International Conference on VLSI Design and 2024 23rd International Conference on Embedded Systems (VLSID), Kolkata, India, 6–10 January 2024; pp. 455–460. https://doi.org/10.1109/VLSID60093.2024.00082.
22. Beckwith, L.; Nguyen, D.T.; Gaj, K. Hardware Accelerators for Digital Signature Algorithms Dilithium and FALCON. *IEEE Des. Test* **2023**, 1. https://doi.org/10.1109/MDAT.2023.3305156.
23. Bisheh-Niasar, M.; Azarderakhsh, R.; Mozaffari-Kermani, M. A Monolithic Hardware Implementation of Kyber: Comparing Apples to Apples in PQC Candidates. In *Progress in Cryptology–LATINCRYPT 2021, Proceedings of the 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, 6–8 October 2021*; Longa, P.; Ràfols, C., Eds.; Springer: Cham, Switzerland, 2021; pp. 108–126.
24. Intel Inc. Intel Vtune Profiler, 2023. Available online: https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html (accessed on 7 August 2024).
25. Montgomery, P.L. Modular multiplication without trial division. *Math. Comput.* **1985**, *44*, 519–521.
26. Richard,T.; Chao.L; Myoung A. *Algorithms for Discrete Fourier Transform and Convolution*; Springer: Cham, Switzerland, 2021. https://doi.org/10.1007/978-1-4757-2767-8.
27. SYNOPSYS Inc. Synopsys Design Cimpiler. Available online: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html (accessed on 7 August 2024).
28. Martins, M.; Matos, J.M.; Ribas, R.P.; Reis, A.; Schlinker, G.; Rech, L.; Michelsen, J. Open Cell Library in 15nm FreePDK Technology. In Proceedings of the 2015 Symposium on International Symposium on Physical Design, Monterey, CA, USA, 29 March–1 April 2015; ISPD '15; pp. 171–178. https://doi.org/10.1145/2717764.2717783.

29. Soni, D.; Basu, K.; Nabeel, M.; Aaraj, N.; Manzano, M.; Karri, R. *Hardware Architectures for Post-Quantum Digital Signature Schemes*; Springer: Cham, Switzerland, 2021. https://doi.org/10.1007/978-3-030-57682-0.

30. Alharbi, A.R.; Hazzazi, M.M.; Jamal, S.S.; Aljaedi, A.; Aljuhni, A.; Alanazi, D.J. DCryp-Unit: Crypto Hardware Accelerator Unit Design for Elliptic Curve Point Multiplication. *IEEE Access* **2024**, *12*, 17823–17835. https://doi.org/10.1109/ACCESS.2024.3358213.

31. Amiet, D.; Leuenberger, L.; Curiger, A.; Zbinden, P. FPGA-based SPHINCS+ Implementations: Mind the Glitch. In Proceedings of the 2020 23rd Euromicro Conference on Digital System Design (DSD), 26–28 August 2020; pp. 229–237. https://doi.org/10.1109/DSD51259.2020.00046.

32. Kocher, P.C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the Advances in Cryptology—CRYPTO '96*; Koblitz, N., Ed.; Springer: Berlin/Heidelberg, Germany, 1996; pp. 104–113.

33. Bogdanov, A. Improved Side-Channel Collision Attacks on AES. In *Proceedings of the Selected Areas in Cryptography*; Adams, C., Miri, A., Wiener, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 84–95.

34. Ji, Y.; Wang, R.; Ngo, K.; Dubrova, E.; Backlund, L. A Side-Channel Attack on a Hardware Implementation of CRYSTALS-Kyber. In Proceedings of the 2023 IEEE European Test Symposium (ETS), 22–26 May 2023; pp. 1–5. https://doi.org/10.1109/ETS56758.2023.10174000.

35. Xagawa, K.; Ito, A.; Ueno, R.; Takahashi, J.; Homma, N. Fault-Injection Attacks Against NIST's Post-Quantum Cryptography Round 3 KEM Candidates. In *Proceedings of the Advances in Cryptology–ASIACRYPT 2021*; Tibouchi, M.; Wang, H., Eds.; Springer: Cham, Switzerland, 2021; pp. 33–61.

36. Zhao, Y.; Pan, S.; Ma, H.; Gao, Y.; Song, X.; He, J.; Jin, Y. Side Channel Security Oriented Evaluation and Protection on Hardware Implementations of Kyber. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 5025–5035. https://doi.org/10.1109/TCSI.2023.3288600.

37. Bos, J.W.; Gourjon, M.O.; Renes, J.; Schneider, T.; Vredendaal, C.v. Masking Kyber: First- and higher-order implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* **2021**, 173–214. https://doi.org/10.46586/tches.v2021.i4.173-214.

38. Migliore, V.; Gérard, B.; Tibouchi, M.; Fouque, P.A. Masking Dilithium. In *Proceedings of the Applied Cryptography and Network Security*; Deng, R.H., Gauthier-Umaña, V., Ochoa, M., Yung, M., Eds.; Springer: Cham, Switzerland, 2019; pp. 344–362.

39. Cerdeira, D.; Martins, J.; Santos, N.; Pinto, S. ReZone: Disarming TrustZone with TEE Privilege Reduction. In Proceedings of the 31st USENIX Security Symposium (USENIX Security 22), Boston, MA, USA, 10–12 August 2022; pp. 2261–2279.

40. Ryan, K. Hardware-Backed Heist: Extracting ECDSA Keys from Qualcomm's TrustZone. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; CCS '19, pp. 181–194. https://doi.org/10.1145/3319535.3354197.