Hardware Accelerators for Digital Signature Algorithms Dilithium and FALCON

Luke Beckwith, Duc Tri Nguyen, Kris Gaj
George Mason University, USA
{lbeckwit, dnguye69, kgaj}@gmu.edu

***Abstract*—Digital signature algorithms are the foundation of many secure communication protocols, including those used in Internet of Things (IoT) applications. While the current generation of signature schemes is secure against classical attacks, they are potentially vulnerable to attacks using quantum computers. Because of this threat, multiple new schemes have been developed and evaluated in recent years. From among these schemes, the National Institute of Standards and Technology standardized two and selected additional three for near-term standardization. For use in IoT, these schemes must be sufficiently efficient in terms of their public-key and signature sizes and the timing of major operations. In this paper, we analyze the choice between two primary schemes considered for extensive use in IoT, CRYSTALS-Dilithium and FALCON, from the point of view of developing efficient hardware accelerators supporting cryptographic operations performed by IoT clients and servers.**

I. Introduction

Public key cryptography is a central component of current network security protocols. Digital signatures allow for strong authentication guaranteeing the integrity and claimed origin of a message. Currently, algorithms such as RSA and Elliptic Curve Digital Signature Algorithm (ECDSA) are used for these purposes. However, these algorithms base their security upon problems that are difficult to solve on current computers but become relatively easy to solve using future quantum computers. In particular, a sufficiently powerful quantum computer can employ Shor's algorithm to break both RSA and ECC [1].

Because of this upcoming threat, the National Institute of Standards and Technology (NIST) has pursued a two-pronged approach to developing new digital signature standards. First, it selected two well-established stateful hash-based digital signatures, XMSS (eXtended Merkle Signature Scheme) and LMS (Leighton-Micali Signature), for short-term standardization. Secondly, it included stateless digital signatures in its long-term post-quantum cryptography (PQC) standardization process initiated in December 2016. This effort resulted in the selection of three stateless digital signature schemes, CRYSTALS-Dilithium, FALCON, and SPHINCS+, for near-time standardization.

Stateful digital signatures, XMSS and LMS, require state tracking because a secret key (a.k.a. private key) is updated every time a signature is generated. Additionally, their digital signatures are relatively large. Hence, the key management and transmission requirements of these schemes make them ill-equipped for Internet of Things (IoT) applications. Among the three other schemes, SPHINCS+ is based on similar mathematical principles, which makes its signature sizes relatively large. Consequently, CRYSTALS-Dilithium (since here referred to as Dilithium) and FALCON are two algorithms with the best near-term potential for adoption in IoT applications.

While quantum computers large enough to break RSA and ECC have not yet been developed, work to deploy quantum-secure cryptography needs to begin now. Integrating these new algorithms into security protocols and deploying hardware accelerators to make their physical realizations efficient and secure will take a substantial amount of time. In this paper, we compare and contrast Dilithium and FALCON with a focus on their use in IoT client devices, performing primarily signature verification.

II. Background

A. Cryptography in IoT

Cryptographic algorithms provide many functionalities that are used in the Internet of Things. The simplest example is non-repudiation (a.k.a. strong authentication), where a digital signature is used to ensure that a server cannot deny having sent a message, such as code updates and instructions for IoT clients. In this case, the server uses its secret key to sign the message and sends the signed message along with the server's certificate to the client. The certificate contains a copy of the server's public key signed by a trusted certificate authority. The client can then verify the server's public key using the certificate authority's public key and then use the server's public key to verify the message. In this scenario, the computational cost of the client is the verification of two signatures, and the overall transmission cost is approximately equal to the size of the server's public key, the signature of the certificate authority, and the server's signature on the message. The only information the client must store is a public key of the certification authority.

Another common application of public-key cryptography is in the popular Internet protocol called Transport Layer Security (TLS). In this protocol, public-key algorithms are used for secure key exchange. However, even in this protocol, clients are usually not required to either generate their public-secret key pairs or sign messages. Thus, the primary concerns for the client device are the performance of the signature verification and the size of the public key and signature.

B. Dilithium and FALCON

Both Dilithium and FALCON are lattice-based digital signature schemes. This means that they base their security

upon the difficulty of specific hard problems over algebraic structures called lattices. Examples of such problems include the Module Learning with Errors (M-LWE), Module Short Integer Solution (M-SIS), and NTRU Short Integer Solution (NTRU-SIS) [2], [3].

The core operations of the Dilithium and FALCON verification are polynomial multiplications modulo an irreducible polynomial x^n+1. Each polynomial has n coefficients, where $n = 256$ for Dilithium and $n = 512$ or 1024 for FALCON. Most polynomials have coefficients in the range between 0 and $q-1$, where $q = 8380417 = 2^{23} - 2^{13} + 1$ for Dilithium and $q = 12289 = 12 \cdot 2^{10} + 1$ for FALCON. However, some polynomials have coefficients limited to a much smaller range. We refer to the two types of polynomials mentioned above as polynomials with large and small coefficients, respectively. Multiplication of polynomials may be performed using various algorithms, with different trade-offs between execution time and area. However, for the specific values of parameters n and q selected in Dilithium and FALCON, the most efficient method is using the so-called Number Theoretic Transform (NTT). The NTT allows performing multiplication in the time proportional to $n \cdot \log_2(n)$ while keeping the area of the hardware implementation (other than the memory usage) independent of n.

1) Dilithium: Our work is based on the NIST PQC round 3 parameter set of Dilithium described in [2]. Inputs, outputs, and intermediate results in Dilithium have the form of matrices and vectors of polynomials mentioned above. While all polynomials have the same number of coefficients, n, the security level determines the matrix and vector dimensions. In particular, Dilithium uses matrices of polynomials of the dimensions $k \times l$ = 4×4 for security level 2 (equivalent to the security of SHA-256), 6×5 for security level 3 (equivalent to the security of AES-192), and 8×7 for security level 5 (equivalent to the security of AES-256). If only one NTT unit is used, the execution time of these operations is approximately proportional to the product $k \cdot l$. At the same time, resource utilization is almost independent of security level. The overhead of supporting all security levels in one implementation is negligible.

The two primary operations determining the execution time and resource utilization of Dilithium's verification are 1) generating a $k \times l$ matrix A of polynomials with n pseudorandom coefficients (a.k.a. polynomial sampling). The pseudorandom string is obtained using the SHA-3 eXtandable Output Function (XOF) - SHAKE128 - with the input seed, ρ being a part of the public key. The output string is then divided into equal-size chunks and converted into coefficients uniformly distributed in the range between 0 and $q-1$ through the process called rejection sampling. 2) Multiplying the mentioned above $k \times l$ matrix A by an $l \times 1$ polynomial vector z being a part of the signature.

The execution time of the former operation is determined by the speed and the number of SHA-3 cores (a.k.a. Keccak cores). The execution time of the latter is determined by the speed and the number of NTT units.

In Dilithium, key generation (KeyGen) and signing (Sign) involve very similar basic operations as those used during verification (Verify). This feature benefits applications in which all three operations must be performed by the same device (typically a server).

Another interesting quality of Dilithium is that its signature generation time is variable. Since some attempts at signing may lead to signatures leaking information about the secret key, these attempts must be repeated (with different random input) until the predefined security conditions are met. Consequently, it is common to report both the best (the first-attempt) and the average time required for signing.

2) FALCON: The major operations of FALCON's verification include 1) multiplication of a polynomial h, obtained by decoding the public key, and the polynomial s_2 obtained by decompressing a part of the signature, 2) decoding the public key, 3) calculating the norm of two vectors s_1 and s_2. The execution time of all these operations is proportional to the number of coefficients in each polynomial, n, which is 512 for security level 1 (equivalent to AES-128) and 1024 for security level 5 (equivalent to AES-256). One of the remaining major operations, hashing the message using SHAKE256, can be, in most cases, overlapped with the polynomial multiplication.

At security level 5 (equivalent to AES-256), the total size of the public key and signature in FALCON is 3.1 KB, compared to 7.2 KB for Dilithium, making the related certificate more than two times smaller. However, in FALCON, key generation and signature generation are very complex, involving floating-point Fast Fourier Transform, matrix decomposition, and a complex trapdoor sampler [3]. Consequently, FALCON is not very suitable for applications in which the client device must perform all three operations or where there are strict performance goals for all operations.

III. Previous Work

Prior to this work, the fastest implementation of Diltihium (considering all timing metrics other than the average time of signing) was by Zhao et al. [4]. This design utilized a high-performance NTT architecture for multiplication and a high-performance Keccak module, which allowed on-the-fly generation of matrices and vectors in parallel with polynomial multiplications. This implementation was also designed to pipeline major operations, which supported high performance with relatively low resource utilization.

The best lightweight implementation was by Gupta et al. [5]. This design used a single compact Keccak implementation and a smaller NTT implementation. Due to the slower Keccak design, the designers chose to compute the public matrix only once and then keep it in memory between signature attempts. These design decisions led to much higher latency and BRAM usage but much lower LUT, FF, and DSP utilization as compared to Zhao et al.

Our work builds upon the publicly available implementation of Dilithium by Beckwith et al., described in [6]. We revisited this design, focusing on achieving high-performance results comparable to those reported by Zhao et al. We then narrowed

down the functionality of this design to achieve lower resource utilization and perform a fair comparison with FALCON.

FALCON has not received any full hardware implementation prior to this work. This is likely due to the complexity of the algorithm. In particular, the key generation and signing functions are both complex and not naturally hardware-friendly. For example, these functions require floating-point FFT operations with 53 bits of precision, which are costly to implement on FPGAs. The tree sampling of signature generation requires recursive FFT operations, which are challenging to implement in hardware and require a large amount of memory.

The only previous work related to FALCON in hardware is the hardware/software codesign of the verification operation of FALCON reported by Karl et al. [7]. This work discussed a RISC-V processor with hardware accelerators to perform some of the operations needed in Dilithium and FALCON. In particular, hardware accelerators are used for the Keccak hash function and the NTT operations of both algorithms.

IV. Methodology

A. Dilithium

We developed three hardware architectures for Dilithium using as a starting point our own high-performance design reported in [6]. The first architecture is a relatively minor modification of the original design, optimized for speed and energy usage. It supports all operations and security levels and allows selection between them at run-time. This design is particularly suitable for accelerating the operation of IoT servers.

We then developed two architectures focused on client devices. One is a high-performance verification-only design for clients that need high performance and are not substantially constrained by resource utilization. The second is a lightweight verification-only design for devices that are constrained in terms of resource utilization but run applications that are less demanding in terms of speed.

In all architectures, the primary challenge is to properly optimize the polynomial sampling and NTT operations. Sampling of polynomials requires a substantial amount of pseudorandom data from the SHAKE XOF functions. These functions are time-consuming to perform and costly to accelerate. Similarly, the NTT operation is the slowest polynomial operation required in Dilithium and needs to be performed for each multiplication. The NTT operation requires an expensive modular multiplier and significant RAM bandwidth to perform quickly. These two operations are the primary bottlenecks of Dilithium.

1) High-Performance General-Purpose: A high-level view of our high-performance general-purpose architecture is shown in Fig. 1. The design is partitioned into several functional units: a) polynomial arithmetic, which performs the NTT and other polynomial operations, b) polynomial packing unit, which encodes polynomials into byte-arrays for hashing and generating final results, c) polynomial sampling, which samples the polynomial matrices and vectors, d) hash modules, which contain the Keccak instances for hashing and sampling, e) polynomial unpacking modules, which unpack the encoded polynomials within keys and signatures, and f) memory bank, which contains the BRAM for storing temporary results. The connections among these units are wide enough to handle four polynomial coefficients in parallel.

Internally, we use the same NTT architecture as discussed by Beckwith et al. [6]. This architecture uses four butterfly units in a 2×2 configuration, which enables the NTT operations to be performed in $256+d$ cycles, where d is the pipeline depth of the module. All basic polynomial operations require only 64 cycles. However, we have made substantial changes to the other modules and the top-level architecture to reduce the LUT and BRAM resources required and improve the design's performance. First, we noticed that several modules had similar datapaths and were not used within the same operation. For example, the samplers for the random y vector and secret vectors were both uniform samplers, but the y sampler was only used in signing, and the secret sampler was only used in key generation. Thus, they could be easily merged to reduce resource utilization without any reduction in performance since they never needed to operate at the same time. The number of samples processed per cycle was also adjusted so that these modules minimize area while remaining just faster than the NTT operation. This ensures these modules do not limit the performance of the design while also ensuring no extra area is wasted.

Similarly, we identified several other modules which could be reduced in the area without any loss in performance. The hint unpacking module and the challenge sampler both consumed a substantial amount of area as they were designed to minimize the latency of their respective operations. However, by adjusting the ordering of operations, these functions could be performed in parallel with slower NTT operations. Doing so allowed these modules to be scaled down without any performance loss. The Keccak module used in the original design also consumed more resources than those used in other works. The interface of the design prioritized simplicity and flexibility. However, Dilithium has very limited requirements for the hash unit as only the SHAKE modes are used, and most hash inputs are byte-aligned. Thus, we were able to create a specialized wrapper for the hash core that required fewer resources. To reduce BRAM utilization, we adopted the on-the-fly sampling approach for the public matrix used by Zhao et al. [4].

To improve the performance of the design, we decided to increase the number of Keccak instances from three to four and to use four polynomial arithmetic units. This approach improved the pipeline balance during matrix multiplication since the four Keccak units could sample the next four matrix polynomials in parallel with the polynomial arithmetic units performing polynomial multiplication. Due to the area improvements of the previously discussed submodules, the net look-up table (LUT) and block RAM (BRAM) consumption was still reduced. However, the flip-flop (FF) and digital signal processing unit (DSP) utilization did increase.

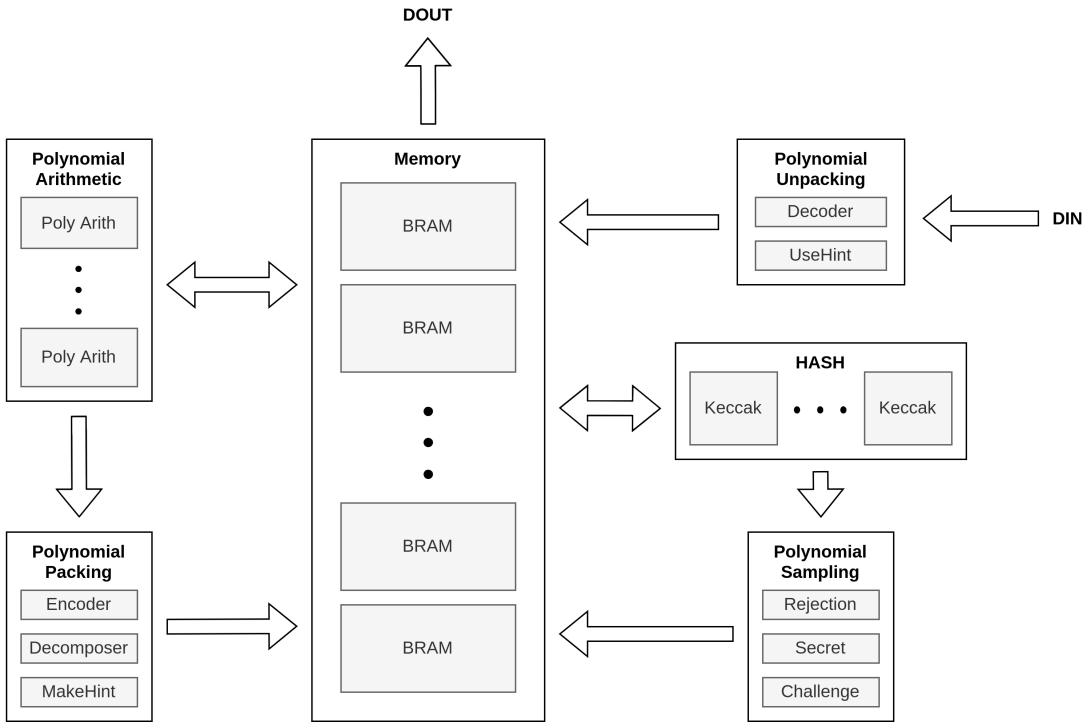

Fig. 1: High-Level Block Diagram of Dilithium Hardware Accelerator

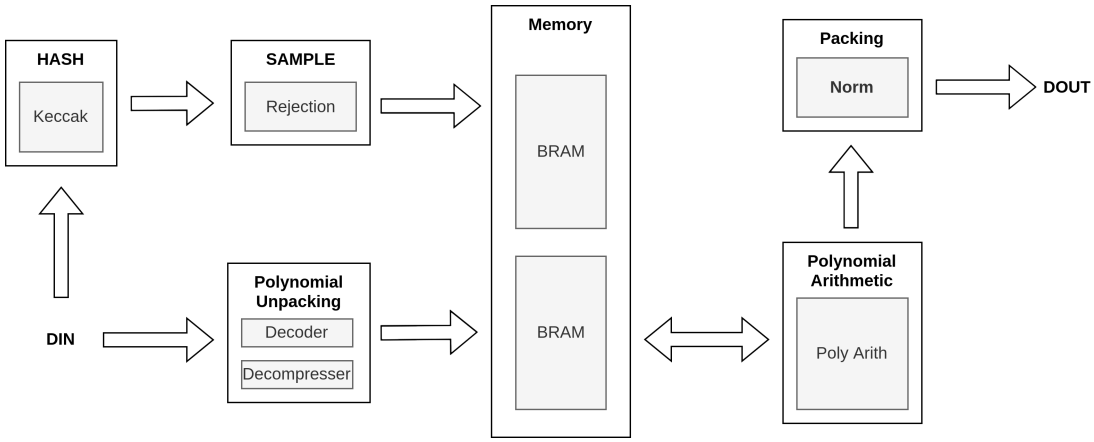

Fig. 2: High-Level Block Diagram of FALCON Verify Hardware Accelerator

2) Verification-Only: The Dilithium high-performance verification-only design is able to slightly reduce resource utilization since only a few submodules are only required for key generation and signing. In particular, the secret sampling, some decoding modes, make hint, and control logic for key generation and signing are not needed for the verification-only architecture. The rest of the architecture is unchanged for the high-performance design. However, since most of the resources are consumed by the hash and polynomial arithmetic units, it is only a slight improvement.

The lightweight architecture keeps the same top-level design but reduces the datapath width from four coefficients wide to one coefficient wide. The internal submodules are also scaled down proportionally. For example, in the high-performance design, four polynomial coefficients were decoded per cycle in the unpacking modules, whereas in the lightweight architecture, only one is decoded per cycle. Additionally, only one Keccak instance is used for hashing, and only one polynomial arithmetic module with a single modular multiplier is used. This change has increased the latency of operations by approximately a factor of nine but has greatly reduced the design's resource consumption.

B. FALCON Verify

As mentioned previously, we have focused on the verify operation of FALCON. Similarly to our Dilithium design, we present both high-performance and lightweight architectures

to support both low-cost and low-latency applications. The high-level block diagram of our FALCON-verify design can be seen in Fig. 2. The same structure is used for both the high-performance and lightweight designs, with the primary change between the architectures being in the level of parallelization within the submodules. For both architectures, the hashing of the message and sampling of the c polynomial is performed in parallel with the decoding and multiplication of the public key and signature polynomials.

1) High-Performance: The datapath of the high-performance architecture is four coefficients wide to take advantage of the same NTT architecture used for Dilithium. Due to the larger number of coefficients of the polynomials used in FALCON, the NTT latency is $640 + d$ and $1280 + d$ cycles for level 1 and level 5, respectively. The only modifications required were a) changing the arithmetic units to support the smaller modulus of FALCON, b) updating to use the twiddle factors of FALCON's NTT, and c) the addition of a conditional bypass of the two NTT butterflies since FALCON512 requires an odd number of NTT layers. Decoding the public key polynomial is a simple deserialization that can be performed using a bus width converter. Decompression is somewhat more complicated, but it can be performed in hardware using a priority encoder and a conditional subtraction. The decoder is optimized to handle four coefficients per cycle so that the first NTT operation can begin as soon as possible. However, decompression is performed in parallel with an NTT optimization. Thus, it only decompresses one coefficient per cycle to minimize area. The norm module calculates the sum of squares for all coefficients of s_1 and s_2 and signals to the controller that the signature is invalid if the norm exceeds the predefined limit.

2) Lightweight: For the lightweight architectures, the primary change is the reduction of the datapath to one coefficient wide. All submodules are scaled down to match this width. The sampler, norm module, and decoder are reduced to handle one polynomial per cycle. The polynomial arithmetic module uses a single modular multiplier in place of the four used in the high-performance design. This change reduces the area but with a substantial increase in latency.

V. Results

We benchmarked all our designs and provide results for Artix-7 FPGAs to enable a fair comparison with the best previously reported implementations. We focus on level 5 security as it is the only security level supported by both Dilithium and FALCON. It is also the only security level approved for use in the Commercial National Security Algorithm (CNSA) 2.0 Suite [8], which is used for United States national security systems. All compared implementations of Dilithium (except the lightweight design by Gupta et al.) support all security levels, which can be selected at run time. The design by Gupta supports only security level 5. All FALCON designs support only one security level, level 1 or level 5, based on the selection made at the synthesis time.

We provide a complete set of results for our designs in Table I. Figures 3 and 4 illustrate the comparison with the best other implementations we are aware of, briefly summarized in Section III.

A. FALCON-Verify vs. Dilithium-Verify

Our most important result is the comparison between Verify-only hardware implementations of Dilithium and FALCON for the common security level, 5, equivalent to the security of AES-256. This comparison is summarized in Table I. For high-speed designs, FALCON outperforms Dilithium by 10%. Additionally, the resource utilization of FALCON1024-VER-HS is smaller than Dilithium-VER-HS by the factors $3.7\times$ for LUTs, $5.1\times$ for FFs, $8\times$ for DSP units, and $6\times$ for BRAMs. For lightweight designs, FALCON outperforms Dilithium by a factor of 2.2 in terms of verification latency. Additionally, the resource utilization of FALCON1024-VER-LW is smaller than Dilithium-VER-LW by the factors $1.3\times$ for LUTs, $1.6\times$ for FFs, $2\times$ for DSP units, and $3\times$ for BRAMs.

B. Comparison with other hardware implementations

Compared to the previous work, we have made substantial performance improvement over our own earlier design, denoted in Figs. 3 and 4 by Dilithium-HS (GMU), reported in [6]. As shown in Fig. 3, the latency of key generation and verification is reduced by approximately $4\times$, and signing latency is reduced by approximately $2.4\times$. The increase in the number of polynomial arithmetic and hashing units enabled by our improved operation scheduling is the primary reason for this improvement. This speed-up comes at the cost of doubling the number of DSP units, as shown in Fig. 4. However, the number of LUTs and BRAMs is slightly reduced due to more efficient implementation of the encoding modules and on-the-fly sampling of the public matrix.

The high-performance design by Zhao et al. [4] had a similar performance to our previous design but used fewer FPGA resources. Compared to the improved design, Dilithium-HS (TW), the resource utilization reported by Zhao et al. is still lower. However, our design achieves much better performance, with the key generation, average-case signing, and verification being $2.8\times$, $2.9\times$, and $3\times$ faster, respectively.

Our verification-only high-performance design provides slight area improvements, requiring 14% fewer LUTs, 6% fewer FFs, and 46% fewer BRAMs than the general-purpose design. The number of DSP units remains the same. Only a few submodules are specifically required for key generation and signing. Thus, the elimination of these modules does not significantly benefit resource utilization.

The lightweight verification-only design is much more compact, as shown in Figure 4. However, it comes at almost a $9\times$ increase in latency. While the design's average power is lower, the energy consumption is higher due to the increased latency. Thus the lightweight architecture is primarily suitable for designs with strict area or power limits. Compared to the lightweight work by Gupta et al. [5], this design achieves lower latency with approximately 40% fewer LUTs, half the DSPs,

TABLE I: Full Performance and Area results. Sign performance is the average latency. **TW** denotes this work. *Reports best-case sign latency, not the average latency

Implementation	Designer	Area					Latency (μs) KeyGen/Sign/Verify			
		LUT [K]	FF [K]	DSP	BRAM	f [MHz]	Level 1	Level 2	Level 3	Level 5
Dilithium-HS	**TW**	48.6	30	32	22.5	185	-	12.2/98.3/14.4	22.6/166.7/25.4	30.3/196.3/33.2
Dilithium-VER-HS	**TW**	42	28.2	32	12	185	-	-/-/14.4	-/-/25.4	-/-/33.2
FALCON1024-VER-HS	**TW**	11.4	5.5	4	2	160	-	-	-	-/-/29.5
FALCON512-VER-HS	**TW**	12.1	6.2	4	2	160	-/-/15.3	-	-	-
Dilithium-VER-LW	**TW**	8.1	6.1	2	6	185	-	-/-/121.9	-/-/181.2	-/-/258.8
FALCON1024-VER-LW	**TW**	6.3	3.9	1	2	160	-	-	-	-/-/117
FALCON512-VER-LW	**TW**	6.3	3.9	1	2	160	-/-/54.4	-	-	-
Dilithium-HS (Zhao)	[4]	30	10.4	10	11	96.9	-	42.3/326.1/45.4	60.9/510.8/64	90.8/570.7/92.9
Dilithium-HS (GMU)	[6]	53.2	28.3	16	29	116	-	41.3/257.1/55.9	70.5/424.8/83.4	120.4/473.9/125.6
Dilithium-LW (Gupta)	[5]*	14	6.8	4	35	163	-	-	-	387/699/416

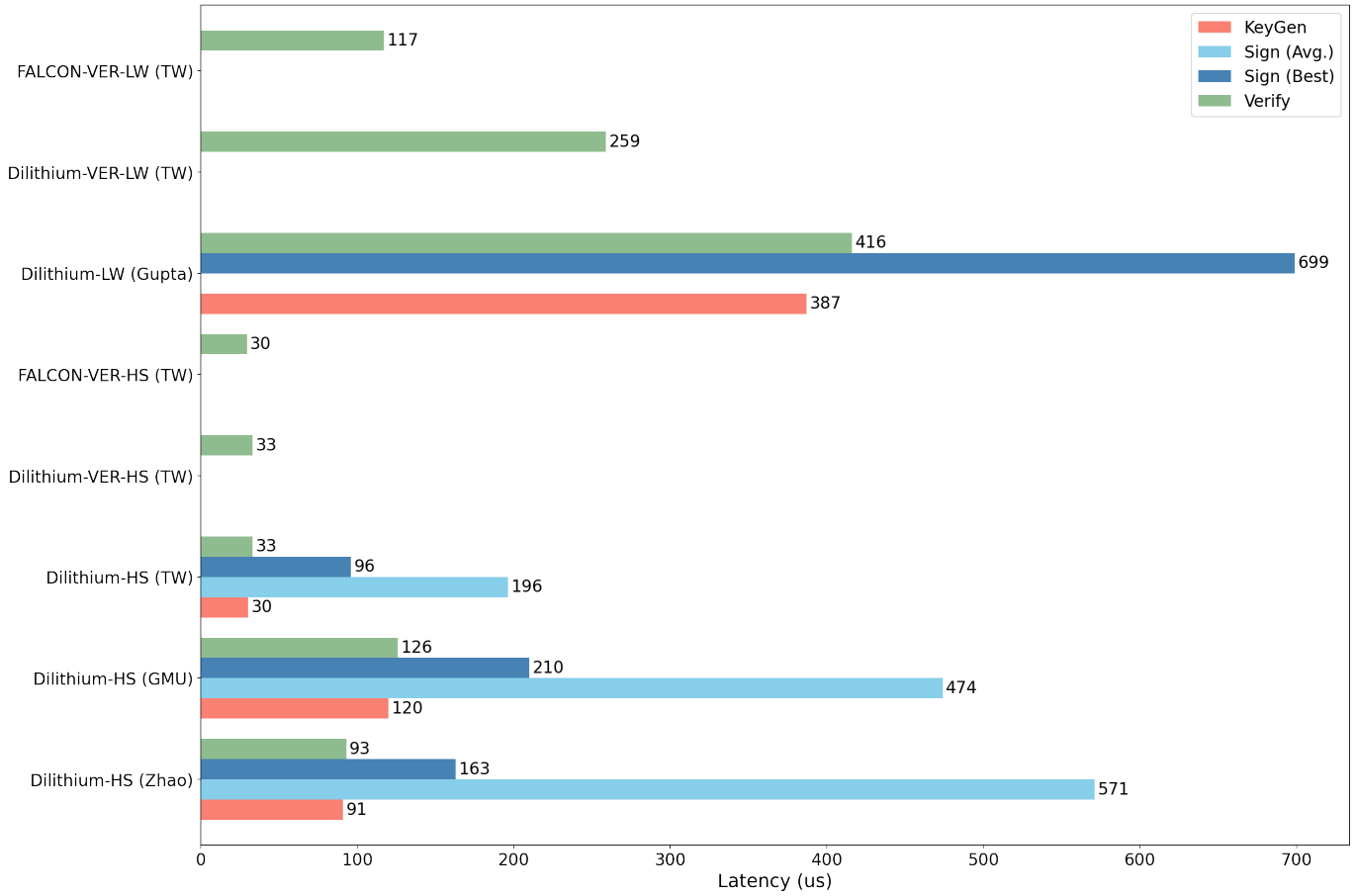

Fig. 3: Accelerator Latency Results for Level 5 Security on Artix-7 FPGA

and one-sixth of the BRAMs. However, the design by Gupta et al. also supports key generation and signing.

In our FALCON design, each instance only supports one security level. Due to the need to bypass the last layer of the NTT, the area of high-performance FALCON512 is slightly larger than for FALCON1024. The only hardware work available for comparison is the hardware/software co-design by Karl et al. [7]. This work uses a set of shared coprocessors connected to a RISC-V soft-core processor to accelerate all operations of Dilithium and the Verify operation of FALCON. The maximum frequency for FPGA is not reported. However, in terms of cycles, our high-performance hardware implementation has 130× lower latency, and our lightweight design has 32× lower latency. The overhead of the coprocessor used in the hardware/software co-design is 7.2K LUTs, 3.3K FFs, 6 BRAMs, and 7 DSPs, which is larger than our lightweight hardware implementation of FALCON-Verify.

VI. Conclusions

Based on verification performance and transmission cost (public key + signature), FALCON is more suitable for IoT applications than Dilithium. It supports faster verification and

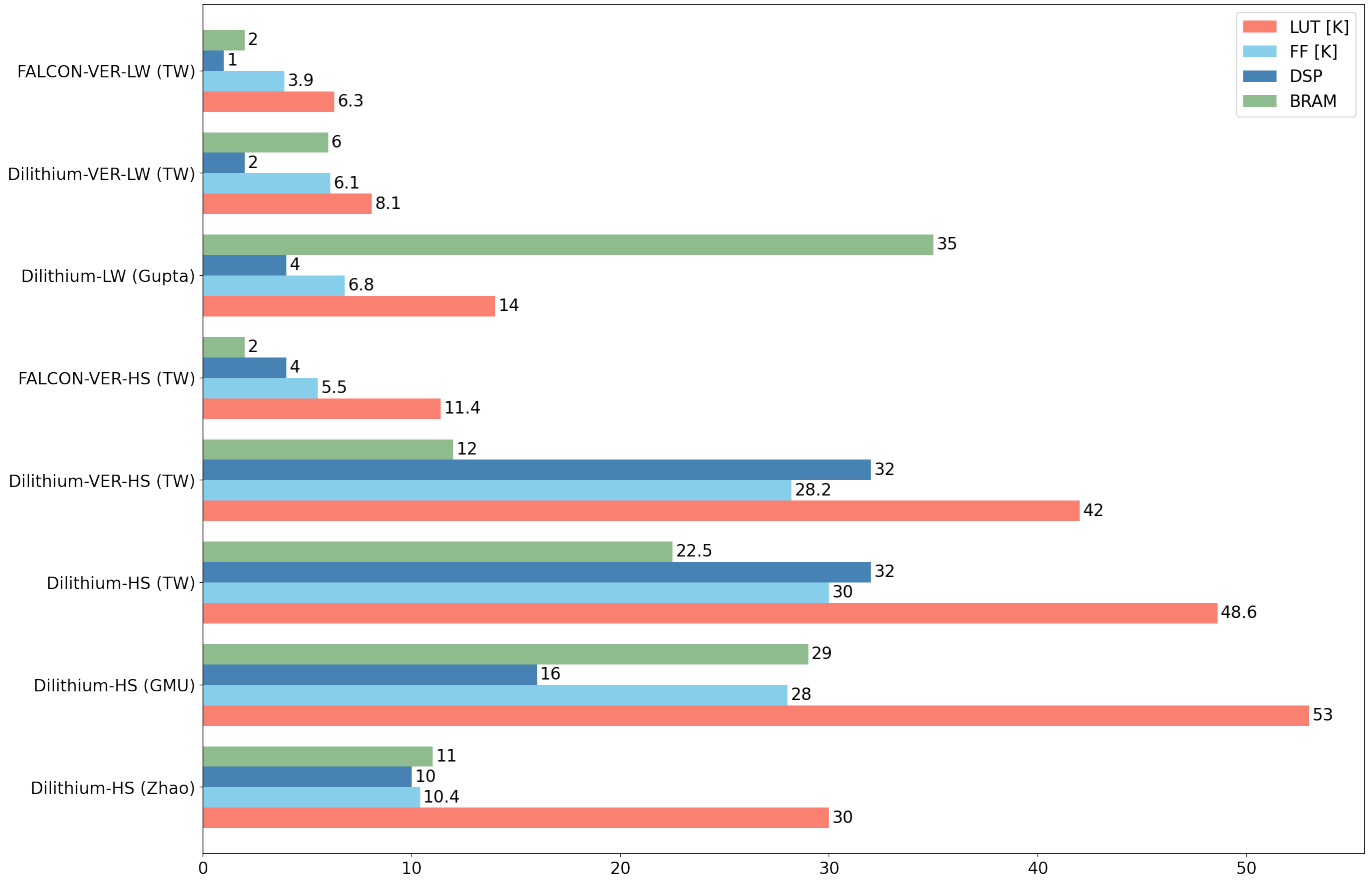

Fig. 4: Accelerator Resource Utilization Results for Level 5 Security on Artix-7 FPGA

has a combined public key and signature size that is less than half the combined size for Dilithium. In FPGAs, FALCON verification achieves higher performance using fewer hardware resources than Dilithium. For ASICs, this feature will likely translate to a substantially better time-area product. However, the complexity of FALCON's key generation and signing makes Dilithium a better option when new keys are frequently needed or when client devices must perform signing.

VII. Acknowledgments

This work has been partially supported by the National Science Foundation under Grant No.: CNS-1801512 and by the US Department of Commerce (NIST) under Grant No.: 70NANB18H218.

References

[1] P. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," in *35th Annual Symposium on Foundations of Computer Science*, IEEE Comput. Soc. Press, 1994.

[2] S. Bai, L. Ducas, E. Kiltz, *et al.*, "CRYSTALS-Dilithium: Algorithm Specifications and Supporting Documentation (Version 3.1)," Feb. 2021. [Online]. Available: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[3] P.-A. Fouque, J. Hoffstein, P. Kirchner, *et al.*, "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU," [Online]. Available: https://falcon-sign.info/.

[4] C. Zhao, N. Zhang, H. Wang, *et al.*, "A Compact and High-Performance Hardware Architecture for CRYSTALS-Dilithium," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, pp. 270–295, Nov. 2021.

[5] N. Gupta, A. Jati, A. Chattopadhyay, and G. Jha, "Lightweight Hardware Accelerator for Post-Quantum Digital Signature CRYSTALS-Dilithium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 8, pp. 3234–3243, Aug. 2023.

[6] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-Performance Hardware Implementation of CRYSTALS-Dilithium," in *2021 International Conference on Field-Programmable Technology (ICFPT)*, Dec. 2021.

[7] P. Karl, J. Schupp, T. Fritzmann, and G. Sigl, "Post-Quantum Signatures on RISC-V with Hardware Acceleration," *ACM Trans. Embed. Comput. Syst.*, Jan. 2023.

[8] NSA, *Cybersecurity Advisory Announcing the Commercial National Security Algorithm Suite 2.0*, Sep. 2022.

Luke Beckwith received his M.S. degree in computer engineering from Virginia Tech in 2020. He is currently working towards his Ph.D. degree with the Cryptographic Engineering Research Group at George Mason University. His research interests include hardware accelerators for lattice and code-based digital signature schemes.

Duc Tri Nguyen earned his Ph.D. degree in Electrical and Computer Engineering from George Mason University in 2023, specializing in cryptographic engineering. He is currently a cryptography engineer at SandboxAQ, focusing on the high-performance software implementation of post-quantum cryptography.

Kris Gaj is a professor and a co-director of the Cryptographic Engineering Research Group at George Mason University. He received his Ph.D. degree in Electrical Engineering from Warsaw University of Technology. He has been involved in most previous and current cryptographic competitions, from AES to PQC.