`KaLi`: A Crystal for Post-Quantum Security using Kyber and Dilithium

Aikata Aikata, Ahmet Can Mert, Malik Imran, Samuel Pagliarini, Sujoy Sinha Roy

***Abstract*—Quantum computers pose a threat to the security of communications over the internet. This imminent risk has led to the standardization of cryptographic schemes for protection in a post-quantum scenario. We present a design methodology for future implementations of such algorithms. This is manifested using the NIST selected digital signature scheme CRYSTALS-Dilithium and key encapsulation scheme CRYSTALS-Kyber. A unified architecture, `KaLi`, is proposed that can perform key generation, encapsulation, decapsulation, signature generation, and signature verification for all the security levels of CRYSTALS-Dilithium, and CRYSTALS-Kyber. A unified yet flexible polynomial arithmetic unit is designed that can processes Kyber operations twice as fast as Dilithium operations. Efficient memory management is proposed to achieve optimal latency.**

`KaLi` is explicitly tailored for ASIC platforms using multiple clock domains. On ASIC 28nm/65nm technology, it occupies 0.263/1.107 mm^2 and achieves a clock frequency of 2GHz/560MHz for the fast clock used for memory unit. On Xilinx Zynq Ultrascale+ZCU102 FPGA, the proposed architecture uses 23,277 LUTs, 9,758 DFFs, 4 DSPs, and 24 BRAMs, at 270 MHz clock frequency. `KaLi` performs better than the standalone implementations of either of the two schemes. This is the first work to provide a unified design in hardware for both schemes.

***Index Terms*—CRYSTALS-Dilithium, CRYSTALS-Kyber, Cryptoprocessor, NIST PQC Standardized**

I. Introduction

COMMUNICATION over the internet forms the backbone of the digitalized world. Every communication packet passes through various insecure channels and untrusted servers before reaching the destination. Data and communication leaks in the past led to the development of public key cryptographic (PKC) schemes to ensure end-to-end security and privacy of communication. These schemes use the hard problems of the discrete logarithm, integer factorization, etc. In 1994, Peter Shor [1] proposed an algorithm that can help a powerful quantum computer solve them in polynomial (realistic) time, thus breaking the classical PKC schemes. Since then, the past eighteen years have witnessed a giant leap in the development of quantum computers. In 2019, Google claimed quantum supremacy by developing a 53-qubit quantum computer *Sycamore* [2]. Sycamore could solve a task in 200 seconds which would take a classical computer 10,000 years. Various labs across the world have developed even stronger quantum computers [3]. This raises the existential question of whether our communication packets containing emails, passwords, etc., are already insecure. The answer to this is - *yes*. Even though quantum computers built until now are not strong enough to break classical public key cryptography, emails and passwords sent now can be stored and decrypted later.

This inevitable breach of security paved way for the development of post-quantum secure PKC schemes based on the hard problems that are safely sustained in a post-quantum scenario. Many standardizations were launched to select the best PKC candidates for digital signature and key-encapsulation algorithms [4]. Key encapsulation schemes allow the communicating parties to agree on the same key securely, which can then be used for symmetric key-based encryption-decryption of messages. Thus, ensuring the security and privacy of the communications. A digital signature scheme allows the receiver to verify the authenticity of the messages. Both these schemes will replace the classical PKC schemes in various applications, like the TLS networking protocol.

These standardizations have now concluded, and the industry is now starting to gear up toward implementing standardized candidates. After finalizing the implementations, a transition phase will start for all the devices to switch from classical to post-quantum secure PKC schemes [5]. This transition will not only take years but also lead to a large amount of wastage in terms of chips and hardware resources which are now obsolete. However, now that we know that change is inevitable, and what we believe to be secure now might again be broken in the next 10-20 years, there is an urgent need for a design methodology for future implementations to prevent loss of time and hardware resources.

This work proposes a design methodology that covers three vital aspects for the future implementations of the PKC algorithms. The first is the need to make a *unified design*. A majority of PKC applications require both digital signature and key encapsulation schemes. Therefore, the design decisions should be adapted to help unify the two algorithms for saving area via resource sharing. Secondly, the design must be *compact*. The new PKC schemes require much larger memory and logical units to store and process the keys. If we do not attempt to make these designs compact, a lot of resource-constrained CPUs that were designed for classical PKC schemes will be

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Aikata Aikata, Ahmet Can Mert, and Sujoy Sinha Roy are affiliated to Institute of Applied Information Processing and Communications, Graz University of Technology, Graz, Austria. Their work was supported in part by Semiconductor Research Corporation (SRC) and the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.{aikata, ahmet.mert, sujoy.sinharoy}@iaik.tugraz.at
Malik Imran and Samuel Pagliarini are with the Centre for Hardware Security, Tallinn University of Technology, Tallinn, Estonia Their work has been partially conducted in the project “ICT programme” which was supported by the European Union through the European Social Fund. It was also partially supported by European Union’s Horizon 2020 research and innovation programme under grant agreement No 952252 (SAFEST). {malik.imran, samuel.pagliarini}@taltech.ee

rendered inoperable. The final aspect is *agility/flexibility*. The architecture design should consider the ever-evolving nature of these algorithms. It will not only help prevent the huge wastage of hardware resources but also enable a smooth transition.

To exhibit the applicative advantages of this design methodology, we take the NIST finalized lattice-based digital signature scheme CRYSTALS-Dilithium [6] and key-encapsulation scheme CRYSTALS-Kyber [7]. We unify the extensive building blocks of these schemes and call the resultant architecture of the two *CRYSTALS* schemes as `KaLi`. The design choices for `KaLi` favor reducing area over improving performance. As a step toward agility, `KaLi` is modeled as an instruction-set cryptoprocessor. From here on, we will refer to CRYSTALS-Dilithium as Dilithium, and CRYSTALS-Kyber as Kyber.

Prior works in literature propose the hardware implementation of PKC schemes. Most of them focus on standalone efficient implementations [8]–[23]. The real-life applications would require both the types of schemes. Therefore, these works fail to provide complete area and timing results for the implementations that make the communication post-quantum secure. The authors in [24], [25] present hardware/software (HW/SW) co-designs for Dilithium and Kyber. Since they keep some part of the design in software, it is not sufficient to provide a good estimate for hardware-only architectures. There is a need for a unified implementation of these two types of schemes completely in hardware to get better performance. We show how `KaLi` follows the proposed design methodology and performs better than the state-of-the-art.

Our contributions can be summarized as follow:

1) Polynomial multiplication is the most computation-intensive operation in the Dilithium signature scheme and Kyber encapsulation scheme. We propose a compact polynomial multiplier architecture that works optimally for the two cryptographic algorithms. Dilithium has a 23-bit prime modulus, whereas Kyber has a 12-bit prime modulus. A unified polynomial arithmetic unit is designed for both, Dilithium and Kyber, to save time and area. This unit has a 24-bit datapath. The core operations: addition, subtraction, multiplication, and modular reduction, are made flexible to either process two sets of 12-bit Kyber coefficients or one set of 23-bit Dilithium coefficients. This, in combination with efficient memory management, enables performing arithmetic operations for Kyber twice as fast as Dilithium.
2) We customized the Keccak-based SHA-SHAKE and pseudo-random number generation unit to make an efficient sampling unit for both Dilithium and Kyber. The samplers for both schemes are unified and added into the Keccak block to prevent redundant writing and reading of pseudo-random numbers. The remaining primitive building blocks of Dilithium and Kyber are designed discretely while ensuring low area consumption, simplicity and flexibility. The proposed arithmetic units altogether form the unified cryptoprocessor `KaLi`. It can perform key generation, encapsulation, decapsulation, signature generation, and signature verification operations for all security levels of Dilithium and Kyber. This is the **first** work that implements a unified cryptoprocessor for Kyber and Dilithium solely in hardware.
3) We propose an instruction set architecture for flexibility. The instructions are divided into two sets, and `KaLi` can run instructions from these two sets in *parallel*, thus improving the latency, while keeping the area consumption low. This leads to a 35% reduction in runtime.
4) `KaLi` is engineered separately for the ASIC platform to reduce area overhead. It uses two clock domains, where the memory unit works at a higher clock frequency than the logic unit. This allowed us to use single port memory instead of dual port memory used in FPGA implementation, thus reducing the area consumption.

The paper is organized as follows. Section II provides a high-level overview of Dilithium and Kyber. The major contributions of the paper are described in Section III. It includes the design methodology for implementing the PQC schemes and implementation details. In Section IV, we give the results and compare them with the existing works in the literature, and add benchmarking estimates. Section V concludes our paper.

II. Preliminaries

Kyber and Dilithium are part of the Cryptographic Suite for Algebraic Lattices (CRYSTALS), which are recently selected for standardization by the American National Institute of Standards and Technology (NIST). Kyber's security relies on the hardness of solving learning-with-errors in module lattices (MLWE), while Dilithium's security is based on MLWE and Shortest Integer Solution (SIS) problems. The polynomials and algebraic operations are assumed to be over the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/x^n + 1$. For Dilithium $n = 256$ and $q = 8380417 = 2^{23} - 2^{13} + 1$, and for Kyber $n = 256$ and $q = 3329 = (2^{12} - 3 \cdot 2^8 + 1)$. Next, we give a brief overview of these schemes and their building blocks.

A. Dilithium

This digital signature scheme has three main algorithms: key generation, signature generation, and signature verification. The sender generates a public and secret key using the key generation algorithm. Then he uses his private key to sign a message using the signature generation algorithm. The receiver can verify the signature using the sender's public key and signature verification algorithm. The signature generation algorithm continues to generate a signature until a valid signature is generated. For a signature to be valid, a set of constraints have to be satisfied to ensure that the signature does not bear any similarity with the message. Readers may refer to [26] for the original specification of Dilithium. Dilithium has three variants for NIST security levels 2, 3, and 5. Several building blocks used by these algorithms are explained below.

- **Polynomial generation**: SHAKE-128 is used to generate the polynomials of the public matrix $\boldsymbol{A} \in R_q^{k\times\ell}$ by expanding the seed $\rho \in \{0,1\}^{256}$ along with 16-bit nonce values. The secret polynomial vectors $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ $\in S_\eta^\ell \times S_\eta^k$ are generated using SHAKE-256. For each polynomial, the seed ς and a 16-bit nonce are fed to SHAKE-256 and passed through rejection sampling in the range $\{-\eta, \eta\}$. The two types of generations are named as $\texttt{ExpandA}()$ and $\texttt{ExpandS}()$. $\texttt{ExpandMask}()$, is used to generate a polynomial vector in the range $[0, 2\gamma_1 - 1]$ using a rejection sampler. $\texttt{SampleInBall}()$ is used during signature generation and verification, to generate a polynomial with only τ coefficients set to $+1$ or -1 and the remaining coefficients as 0.

- **Polynomial Arithmetic**: Polynomial multiplications are performed using the Number Theoretic Transform (NTT) method. The addition and subtraction operations are coefficient-wise linear operations.
- **Hash functions**: SHAKE-256 is used to make a collision resistant hash function- $\texttt{CRH}()$.
- **Power2Round** : The function, $\texttt{Power2Round}_q()$, takes an element $r = r_1 \cdot 2^d + r_0$ and returns r_0 and r_1, where $r_0 = r \mod {}^{\pm}2^d$ and $r_1 = (r - r_0)/2^d$.
- **Decompose and other related functions**: Let α be a divisor of $q - 1$. The function $\texttt{Decompose}_q()$ is defined in the same way as $\texttt{Power2Round}()$ with α replacing 2^d. The $\texttt{HighBits}_q()$/$\texttt{LoweBits}_q()$ return r_1/r_0 from the output of $\texttt{Decompose}_q()$. $\texttt{MakeHint}$ uses $\texttt{HighBits}_q()$ to produce a hint $\boldsymbol{h}$. $\texttt{UseHint}$ uses the hint $\boldsymbol{h}$ produced by $\texttt{MakeHint}_q()$ to recover the high-bits.

B. Kyber

Kyber is an IND-CCA2-secure key encapsulation scheme. It has three principal algorithms: key generation, encapsulation, and decapsulation. The receiver generates a public and secret key using the key generation algorithm and broadcasts the public key. When the sender wishes the send a message, he/she can encapsulate it using the receiver's public key through the encapsulation algorithm. The receiver can then decapsulate it using her/his secret key through the decapsulation scheme. Three variants of Kyber, Kyber-512, Kyber-768, and Kyber-1024 are provided for NIST Security levels 1, 3, and 5, respectively. The variants differ in module dimensions and coefficient distributions. Readers may refer to [7] for the detailed specifications of Kyber. Kyber has the following internal routines:

- **Pseudorandom functions**: Kyber uses PRF (SHAKE-256) and XOF (SHAKE-128) to generate the pseudo-random numbers for polynomial coefficients.
- **Hash functions**: Kyber provides functions H and G for SHA3-256 and SHA3-512, respectively, for hashing.
- **Key-derivation function** (KDF): It is instantiated using SHAKE-256 in Kyber.
- **Polynomial Arithmetic**: Kyber uses *a new method* NTT-based polynomial multiplication unit. Polynomial addition and subtractions are also supported.
- **Samplers**: Uniform sampling (Parse) is used to generate the public polynomials, and Binomial sampling (CBD) is used to generate secret and error polynomials.
- **Encode/Decode**: These modules are used to serialize/deserialize the polynomials to/from byte arrays.
- **Compress/Decompress**: They are used to reduce the size of ciphertext by discarding low-order bits. They are defined on an element $x \in \mathbb{Z}_q$ as $\lceil (2^d/q) \cdot x \rfloor \pmod{2^d}$ and $\lceil (q/2^d) \cdot x \rfloor$ respectively, where $d < \lceil \log_2(q) \rceil$. The value x' such that $x' = \texttt{Decompress}(\texttt{Compress}(a, d), d)$ is an element close to x.

C. NTT-based Polynomial Multiplication

Polynomial multiplication of $(n-1)$-degree polynomials has been the focus of works for PQC implementations. Most implementations use the traditional NTT-based multiplication technique, while others show how methods like schoolbook $O(n^2)$, Karatsuba $O(n^{1.59})$, etc., can be used. NTT-based multiplication has a time complexity of $O(n(log\ n))$. The designers of Dilithium and Kyber select polynomials in Ring $\mathcal{R}_q = \mathbb{Z}_q[x]/x^n + 1$, where modulus q is an NTT-friendly prime. Thus, making it easier to use the fast NTT-based multiplication method.

Forward NTT transform converts an $(n-1)$-degree polynomial (coefficient representation) to n 0-degree polynomials (value representation). Then two polynomials in their value-representation form (NTT domain) can be multiplied coefficient-wise to get the multiplied values in the NTT domain. Now, if we need to get the polynomial in coefficient representation again, a backward NTT transform (INTT) is used. The conversion to-and-from NTT domain has a time-complexity of $O(n(log\ n))$. Coefficient-wise multiplication has a time-complexity of $O(n)$. Thus, a total time complexity of $O(n(log\ n))$. Various algorithms exist in the literature to facilitate these transformations. The most used ones are the Cooley-Tukey (Algorithm 1) transform for NTT and Gentleman-Sande for INTT. For more information on NTT/INTT, refer to [27].

Next, we discuss the major optimizations made to realize the design methodology in the context of Dilithium and Kyber.

III. Proposed Unified Hardware Architecture

The first and foremost goal is to unify the digital signature scheme and the key-encapsulation scheme. While doing this, it is important to ensure that the design is compact and flexible. Unification has a very straightforward three-step approach. First, we must identify the most area and time-consuming

Algorithm 1 The Cooley-Tukey NTT Algorithm [28]

```
In: An $n$-element vector $x = [x_0, \cdots, x_{n-1}]$ where $x_i \in [0, q-1]$
In: $n$ (power of 2), modulus $q$ ($q \equiv 1 \pmod{2n}$)
In: $\mathbf{g}$ (precomputed table of $2n$-th roots of unity, bit-reversed order)
Out: $x \leftarrow NTT(x)$
1: $t \leftarrow n/2; m \leftarrow 1$
2: while ($m < n$) do
3:     $k \leftarrow 0$
4:     for ($i \leftarrow 0; i < m; i \leftarrow i + 1$) do
5:         $S \leftarrow \mathbf{g}[m + i]$
6:         for ($j \leftarrow k; j < k + t; j \leftarrow j + 1$) do
7:             $U \leftarrow x[j]$                              ▷ Butterfly starts
8:             $V \leftarrow x[j + t] \cdot S \pmod q$
9:             $x[j] \leftarrow U + V \pmod q$
10:            $x[j + t] \leftarrow U - V \pmod q$              ▷ Butterfly ends
11:        end for
12:        $k \leftarrow k + 2t$
13:    end for
14:    $t \leftarrow t/2; m \leftarrow 2m$
15: end while
16: return $x$
```

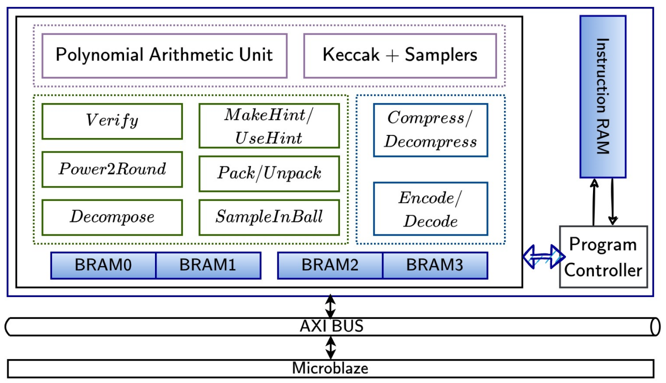


Fig. 1. High-level architecture of `KaLi`

building blocks, the Giants. This is because unifying the low area and time-consuming building blocks (the Dwarves) will not reduce the area consumption significantly, and instead limit the flexibility of the design. The next step is to find the algorithmic synergies between the Giants of the two schemes. The final step is to discern if some of the Dwarves which are dependent on the Giants can be unified with Giants to reduce both area and time consumption.

A high-level view of the proposed architecture `KaLi` is given in Fig. 1. The Keccak-based SHA-SHAKE unit and polynomial arithmetic unit are the two Giants in both schemes. The remaining building blocks are deemed as Dwarves since they comprise only 20% of the total area consumption. Unifying the Keccak-based SHA-SHAKE unit is relatively easy since we can use a common Keccak core for both schemes. Therefore in this section, we will discuss how we efficiently unified the polynomial arithmetic unit. We will also discuss how we efficiently manage the memory for the two schemes. Another facet of the work, the optimization for ASIC platforms, is also presented. We utilized multiple clock domains and boosted the memory bandwidth budget on ASIC platforms to reduce area consumption.

A. *The colossal Giant: Polynomial Arithmetic Unit*

The Polynomial Arithmetic Unit performs polynomial addition, subtraction, and multiplication. Polynomial addition and subtraction are simple coefficient-wise operations, hence cheap. Polynomial multiplication is rather complex, and it is what makes the polynomial arithmetic unit a Giant. Both schemes perform this using NTT, as discussed in Section II-C. Although the two schemes use NTT-based polynomial multiplication units, there are many differences between the two schemes that make their NTT units quite distinct.

1) *A clash of the Giants: The differences between the presumed similar NTTs*

The first distinction between the NTTs used by the two schemes lies in the algorithm itself. The NTT-based polynomial multiplication method used in Dilithium requires the existence of $2n$-th root of unity that mandates $q \equiv 1 \pmod{2n}$. Accordingly, Dilithium uses a *complete-NTT*. After a *complete-NTT* transform of an n-degree polynomial, we get n polynomials of degree 0. In [29], Lyubashevsky *et al.* propose a new method for NTT-based polynomial multiplication that requires only $q \equiv 1 \pmod{n}$, without pre-processing and post-processing operations. This technique is adopted by Kyber, and their 12-bit prime modulus does not have a $2n$-th root of unity. Therefore, Kyber has to use an *incomplete-NTT*. An *incomplete-NTT* gives us $n/2$ polynomials of degree 1. These polynomials cannot be multiplied coefficient-wise.

For the *incomplete-INTT*, multiplication operation of two degree-1 polynomials is performed in the ring $\mathbb{Z}_q[x]/(x^2 - \omega^i)$ where ω is the n-th root of unity and i depends on the index of coefficients. For the details, readers may follow original Kyber specifications [7] or related prior works in the literature [30]. Along with this, they also have a difference in datapath design. Dilithium has a 23-bit prime modulus, while Kyber has a 12-bit prime modulus. Therefore, while Kyber requires 12-bit adder/subtracter/multiplier units, Dilithium requires them in 23-bits. Designing a datapath for one of them and using it for the other one would lead to over-or-under saturation.

Next, we will discuss in detail how we achieved a unified polynomial multiplication unit with full utilization. The unit of interest here is the butterfly unit (BFU). Each BFU performs dyadic addition, subtraction, and multiplication, on the two input coefficients. The results are reduced by modulo q. This is shown by steps 7-10 in Algorithm 1. Since modulus multiplication is the most expensive operation, we will discuss how we unify this unit. Then, we will discuss how with a few more changes, the entire BFU can be consolidated.

2) *Flexible fusion of Modular Multiplier Unit*

As discussed above, if we naively use the 23-bit Dilithium polynomial multiplier unit for Kyber, then it will always be undersaturated as half of it will be unused. Instead, if we aim to use a 12-bit Kyber unit for Dilithium, it will require extra control logic but also slow down Dilithium's NTT. Therefore, we need to find a solution using a 23-bit Dilithium unit that does not lead to undersaturation. The modular multiplier unit has two parts: (i) integer multiplier and (ii) modular reduction unit. We propose an algorithm (Algorithm 2) to make the integer multiplication unit designed for Dilithium flexible for Kyber. It performs two 23-bit$\times$12-bit integer multiplications. The result is added for Dilithium and concatenated for Kyber. This algorithm gives us one multiplied coefficient in the case of Dilithium and two multiplied coefficients in the case of Kyber.

The modular multiplier unit, designed to support modular multiplication using both primes, uses two DSP units of Xilinx FPGAs. The hardware architecture of the re-configurable integer multiplier is shown in Fig. 2. The datapath depends on the scheme type and is heavily pipelined. We used internal registers of DSP units to synchronize two DSP unit outputs and achieve a high clock frequency. Now we need to design a modular reduction unit accordingly.

3) *Versatile Modular Reduction Unit*

The naive solution is to design separate modular reduction units for the two primes. It would require one modular reduction unit for the Dilithium prime and two reduction units for the Kyber prime, which will result in extra hardware costs. To avoid this, we propose a unified modular reduction unit. Both Dilithium $(2^{23} - 2^{13} + 1)$ and Kyber $(2^{12} - 2^9 - 2^8 + 1)$ primes have pseudo-Mersenne structure. For

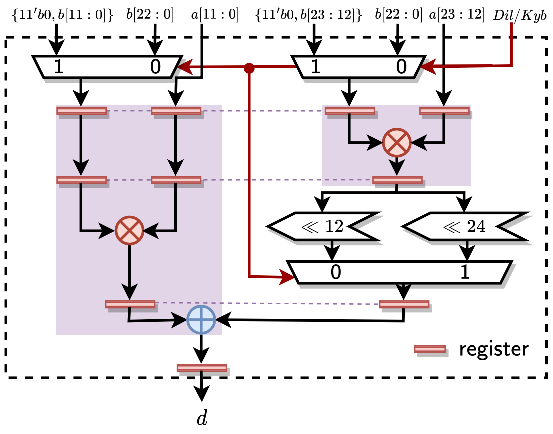


Fig. 2. Flexible yet compact integer multiplier. The red lines show control signals, and the black lines show data movement. The two DSP units used are highlighted in purple.

```
Algorithm 2 Integer Multiplication Algorithm
In:  $a, b \in \mathbb{Z}_{8380417}$ or $a[23:12], b[23:12], a[11:0], b[11:0] \in \mathbb{Z}_{3329}$
In:  $sel \in \{0, 1\}$ (0 for Dilithium and 1 for Kyber)
Out: $d = a \cdot b$ or $d = \{a[23:12] \cdot b[23:12], a[11:0] \cdot b[11:0]\}$
  1: $d_0 = (sel)\ ?\ b[23:12]\ :\ b$
  2: $m_0 = d_0 \cdot a[23:12]$
  3: $m_1 = (sel)\ ?\ (m_0 \ll 24)\ :\ (m_0 \ll 12)$
  4: $d_1 = (sel)\ ?\ b[11:0]\ :\ b$
  5: $d = d_1 \cdot a[11:0] + m1$
  6: return $d$
```

Dilithium prime, we followed the method described in [31] which uses $2^{23} \equiv 2^{13} - 1$ equation recursively. Using this equation, we can reduce a 46-bit integer $d \pmod{2^{23} - 2^{13} + 1}$ to the integer $2^{13}d[45:24] + d[22:0] - d[45:23]$ which consists of addition/subtraction of 36-bit and 23-bit partial results. If we apply this operation recursively, we will obtain $\overline{d}$ $(= d[22:0] + (d[32:23] + d[42:33] + d[45:43])2^{13} - d[45:23] - d[45:33] - d[45:43])$. Similarly, for Kyber, we followed add-shift-based method proposed in [30] which generates partial results using equations $2^{12} \equiv 2^9 + 2^8 - 1$ and $2^{11} \equiv -2^{10} - 2^8 - 1$ recursively.

Summing all partial results using carry propagate adders (CPAs) will result in either a very long carry chain or multiple pipeline stages. In order to avoid long carry chain and pipeline delays, we used carry-save adders (CSAs) along with CPA. The proposed unified modular reduction unit is shown in Fig. 3, where the boxes 'D' and 'K' represent the partial result generation circuits for Dilithium and Kyber primes, respectively. All the subtraction operations are converted into additions by taking the 2's complements of partial results. In Fig. 3, each number inside a box represents a bit index of the input integer (0 to 45 for Dilithium and 0 to 22 for Kyber). The white and brown (terracotta) boxes represent the normal and negated bits. When 2's complements operation is applied to the partial results, extra plus ones, along with sign extensions, come into the picture. These are represented with blue circles.

After adding all of these partial results, we also perform a final correction which brings the resulting integer from the range $(-q, 3q)$ to the range $[0, q)$. The proposed modular reduction unit can either perform one reduction for the Dilithium prime or two reductions for the Kyber prime. The latency of the

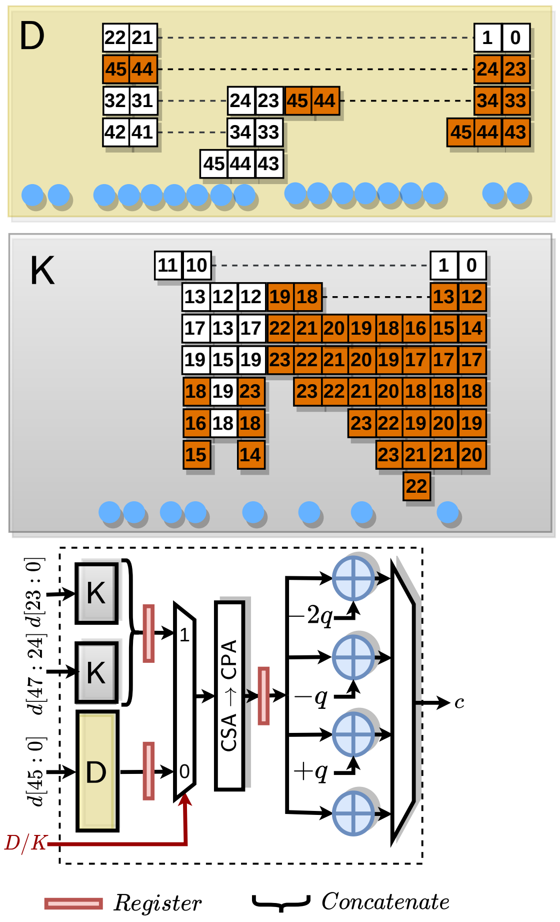


Fig. 3. Unified modular reduction unit for Dilithium and Kyber primes.

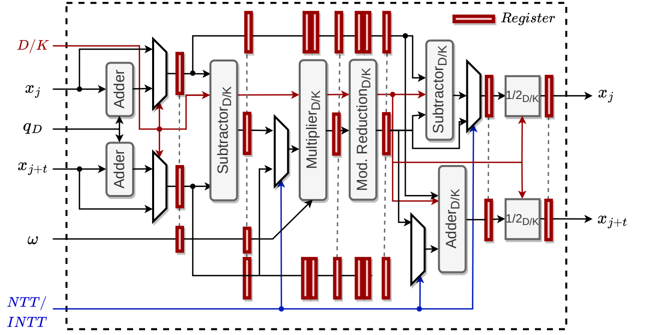


Fig. 4. Compact butterfly unit (BFU) with flexibility for both Dilithium and Kyber. The red and blue lines show control signals, and the black lines show data movement.

modular reduction unit is two cycles and it is fully pipelined.

4) *Coalesced datapath for the Butterfly Unit*

Now that we have unified the modular multiplication unit, we propose a unified BFU (Fig. 4). It can perform one butterfly operation for Dilithium and two butterfly operations for Kyber using the same datapath. All the arithmetic units are made re-configurable to work for both schemes. New re-configurable adder and subtractor units are shown in Fig. 5. The idea is to divide each 24-bit adder/subtractor into two small 12-bit parts and select proper input signals based on the scheme. The complete unified butterfly unit, designed using the re-configurable arithmetic units, is shown in Fig. 4.

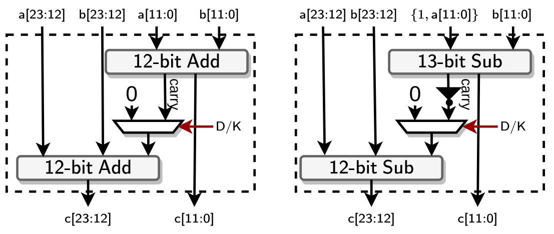


Fig. 5. Unified adder and subtractor for the butterfly unit. The red lines show control signals, and the black lines show data movement.

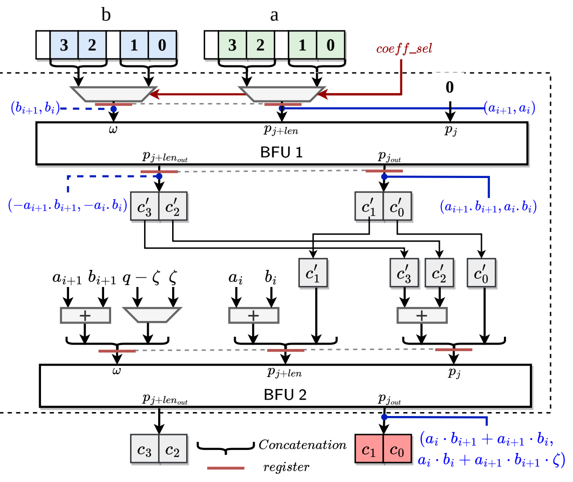


Fig. 6. Butterfly feedback unit for Kyber's NTT-domain polynomial multiplication.

The schoolbook multiplication for Kyber requires five multiplications for multiplying two linear (i.e., degree one) polynomials. We have two flexible butterfly units that act as four butterfly units for Kyber and allow four multiplications only. One way to perform these five multiplications is to add another DSP multiplier just for the extra multiplication. This unit will not be useful for any other operation. To avoid this extra multiplier, we condense five multiplications into four using Karatsuba-like reduction. Then we use these four independent butterfly units as a set of two. The output of the first set is the input to the second set as shown in Fig. 6. The inputs and outputs for the BFUs are highlighted in blue. The control flow here is separated from the Dilithium polynomial multiplication control flow, for simplicity. The entire flow is pipelined to achieve a high clock frequency.

The coefficient consumption during NTT/INTT is shown in Fig. 7. Owing to the flexible datapath and efficient memory arrangement (discussed in the next subsection), the Kyber NTT coefficients can be consumed faster. This enables full utilization of the datapath. The complete polynomial arithmetic unit consumes 3,487 LUTs, 1,918 FFs, 4 DSPs, and 1 BRAM. The BRAM is used to store the powers of roots-of-unity (twiddle factors) required during NTT/INTT operation. In the next section, we will discuss the efficient memory arrangement designed to optimally feed the polynomial arithmetic unit.

B. Memory Arrangement

The polynomial arithmetic unit is designed to consume the Kyber coefficients twice as fast as the Dilithium coefficients. It requires that the memory unit feeds it at the same rate, otherwise, making these unifications will not help improve the performance. Dilithium coefficients are 23-bit, and we have designed the NTT/INTT unit using two butterfly cores. Each of the cores requires exclusive access to the read/write port of a memory. Therefore, we split the memory into two blocks, each storing two Dilithium coefficients per address. For Kyber, each memory block stores four 12-bit Kyber coefficients. Fig. 8 shows the storage of Kyber polynomials in one 64-bit word of memory. One Dilithium polynomial coefficient will occupy two of these coefficients, thus requiring twice the amount of storage. It also ensures that the two required coefficients during NTT/INTT are always stored across different BRAMs. Fig. 8 shows an example of the coefficients storage during Kyber's incomplete-NTT iterations for a 16-coefficient polynomial.

Next, we will discuss how we used multiple clock domains to reduce area consumption in ASIC platforms.

C. Multi-clock domains: Customization for ASIC platforms

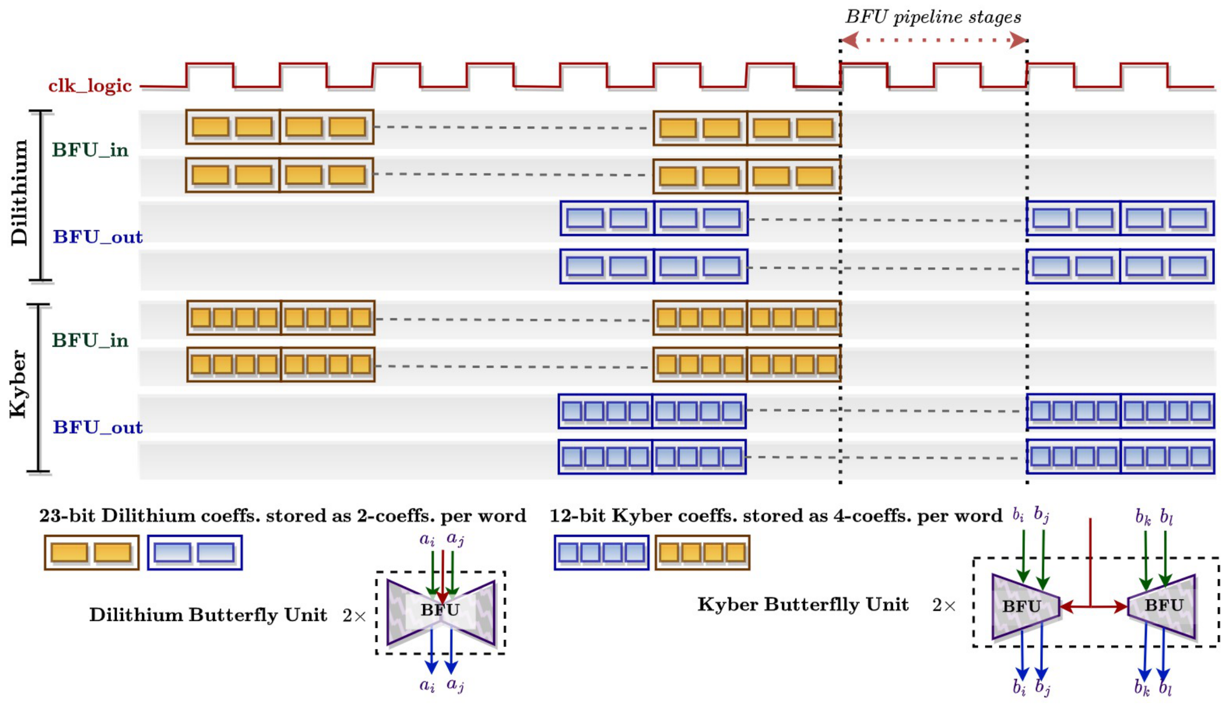
The memory organization discussed above has two sets of BRAMs to feed the two BUF. These BRAMs are used by all the remaining building blocks as well. It is generated using dual-port BRAMs in FPGA. In ASIC, dual-port RAMs consume more area than single-port RAMs. Therefore, to reduce the area consumption, we decide to replace dual-port RAMs with single-port RAMs, which work at a clock frequency twice as fast as the rest of the design. Using two different sources for the two clocks leads to an asynchronous setting. This creates meta-stability problems due to clock-domain crossing. To avoid these problems, we decided to keep the clocking synchronous and generate the slow clock (clock_logic) using the fast clock (clock_mem).

Fig. 9 describes the handshake between memory and logic. A wrapper is provided to process the simultaneous reads and writes to the memories. The read operation is given preference over the write operation to ensure data is valid when the building blocks fetch it and avoid any issues due to clock glitches. The read latency is three clock cycles, and all the building blocks are tailored accordingly. This design helps reduce the area for ASIC designs. Note that a similar modification will not change the FPGA area consumption and instead cause timing problems running the memory at a high clock frequency. Therefore, this adaptation specifically targets ASIC platforms.

Until now we discussed the major contributions of the work. Next, we will briefly discuss how we efficiently implement the remaining building block. We will start with the rejection samplers used in both schemes. These are the Giant dependent Dwarves that might help reduce the area and time consumption without compromising the flexibility of the design.

D. The Giant and the Dwarves: Keccak-based SHA-SHAKE unit and the rejection samplers

Dilithium requires SHAKE-128 and SHAKE-256 for pseudo-random number generation and hashing. Kyber re-



BFU pipeline stages
clk_logic
Dilithium
BFU_in
BFU_out
Kyber
BFU_in
BFU_out
23-bit Dilithium coeffs. stored as 2-coeffs. per word
12-bit Kyber coeffs. stored as 4-coeffs. per word
Dilithium Butterfly Unit 2×
Kyber Butterfly Unit 2×
BFU

Fig. 7. Timeline showing the unified butterfly unit processing Dilithium and Kyber coefficients.

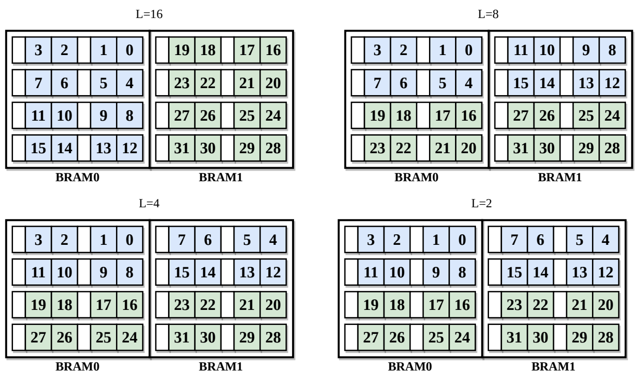


Fig. 8. Storage of coefficients during Kyber's NTT for 16-coefficient polynomial

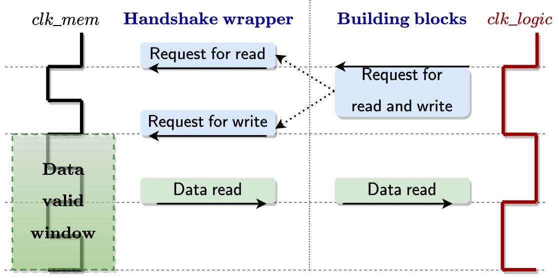


Fig. 9. The data read-and-write handshake between memory and logic unit

quires SHA3-256 and SHA3-512 for hashing and SHAKE-128 and SHAKE-256 for KDF and pseudo-random number generation. These different Keccak-based functions are implemented as modes of the same Keccak output. Therefore, we can use the same Keccak instance for all these modes. Both schemes employ different sampling for the generation of secret and error polynomials. While some of these fully consume the Keccak output, the remaining have to keep track of the leftover bits.

We combine the rejection sampler with the Keccak unit using a book-keeping approach similar to [31]. It improves the performance of the sampling operation, as we do not need to store and then read the Keccak output in between. The base implementation of Keccak follows a high-speed and parallel directive. The control and datapath are modified to work for rejection samplers as it depends on coefficients passing the rejection constraints. The complete Keccak unit consumes 12,326 LUTs and 3,560 FFs.

We have unified all the Giants, so now we will discuss the optimized implementation techniques for the Dwarves.

E. *Optimizations for the* Dwarves

Making a design compact while keeping it agile increases the life and usability of KaLi on the FPGA and ASIC platforms. However, this comes with a series of challenges. We must ensure that for keeping the design agile/flexible, we do not pay a huge price in terms of area. Similarly, while making the design compact, the performance should not get worse. We now discuss how to make certain building blocks of the two schemes compact, while maintaining flexibility.

1) *Compress/Decompress Unit*

The decompress unit performs division by power-of-two and rounding operation which is trivial to implement in hardware. On the other hand, the compress operation requires division by q and rounding. Some works in the literature use Barrett reduction and division algorithms to perform the compress operation. We decide to use sufficient precision and convert

Algorithm 3 The Proposed Compression Algorithm

```
In: $x \in \mathbb{Z}_{3329}$, $d \in \{1, 4, 5, 10, 11\}$
Out: $y = \lceil (2^d/q) \cdot x \rfloor$
1: switch $d$ do
2:     case 1: $t = (10079 \cdot x); y = (t \gg 24) + (t[23] \gg 23)$
3:     case 4: $t = (315 \cdot x); y = (t \gg 16) + (t[15] \gg 15)$
4:     case 5: $t = (630 \cdot x); y = (t \gg 16) + (t[15] \gg 15)$
5:     case 10: $t = (5160669 \cdot x); y = (t \gg 24) + (t[23] \gg 23)$
6:     case 11: $t = (10321339 \cdot x); y = (t \gg 24) + (t[23] \gg 23)$
7: end switch
8: return $y \pmod{2^d}$
```

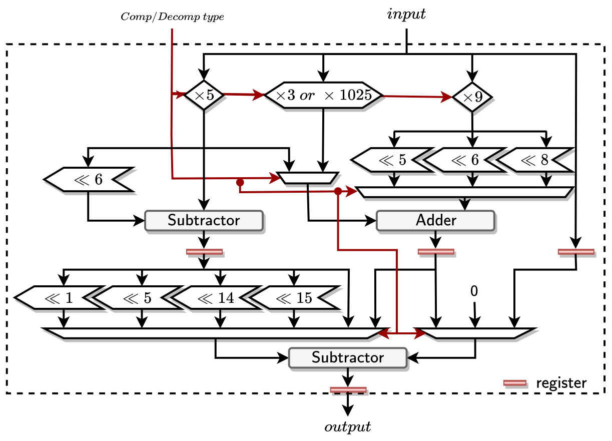


Fig. 10. Architecture of the Compress/Decompress unit. The red lines show control signals, and the black lines show data movement.

division by q operation into multiplication and shift operations. The proposed multiplication-based compress algorithm is shown in Algorithm 3. The input is the Kyber coefficient x and the type of compression required d. The compressed coefficient y is returned as the output.

Since the multiplications are by constant values, we implement these operations using add and shift technique utilizing the LUTs. Fig. 10 shows the hardware architecture of this multiplication unit, used for retrieving the t values in Algorithm 3. This is unified and works for both compress and decompress operations. The control flow is dependent on the type of compression or decompression required.

2) Encode/Decode Unit

Encode and decode units perform coefficient-to-byte and byte-to-coefficient conversions, for all security levels of the Kyber scheme. We used a similar idea as proposed in [32] which uses a 32-bit interface. Our architecture uses a 64-bit interface and thus the proposed encode unit uses a 104-bit buffer. It can encode 1-bit, 4-bit, 5-bit, 10-bit, and 11-bit long coefficients. The decode unit can decode 64-bit inputs into 1-bit, 4-bit, 5-bit, 10-bit, and 11-bit long coefficients using a 72-bit buffer.

3) Pack/Unpack unit

Similar to Kyber, Dilithium requires coefficient-to-byte and byte-to-coefficient conversions for various coefficient sizes. Pack and unpack units perform these conversions for all security levels of Dilithium for coefficient sizes 3, 4, 6, 10, 13, 18, and 20 bits. We again followed the idea proposed in [32] for the pack and unpack units.

The remaining blocks of both schemes are different and unifying them would not save any area and instead complicate the control logic and reduce the flexibility of the design. These building blocks do not require any DSP units and comprise simple bit-wise packing, unpacking, or addition/subtraction operations. They are implemented as individual blocks and they occupy only 18% of the cryptoprocessor's area.

F. Instruction set cryptoprocessor

We made the building blocks compact while ensuring flexibility, but this is insufficient. What happens if, in two years, the Keccak pseudo-random number generation and hashing unit are obsolete? Do we then need to redesign the entire cryptoprocessor? To counter this and increase agility as well as flexibility, we design an instruction set architecture (ISA), where each instruction is a building block required by the cryptographic schemes. A simple program controller runs the cryptographic protocols by executing the necessary instructions, manages the synchronization of parallel instructions, and avoids back-and-forth CPU-Cryptoprocessor communication. Note that the program controller is not a 'control processor', and it does not contain arithmetic circuits to process operand data. It simply decodes an instruction and then activates the corresponding module inside the cryptoprocessor. It consumes only 8% of the cryptoprocessor's area. The instructions and the corresponding hardware modules are listed in Table I.

G. Running the Giants and the Dwarves in parallel

Our goal was to make the unified design compact and agile. However, does this mean we have to pay an equal price in terms of performance? To some extent, this is correct. However, we should continue to ponder on some methods that could boost the performance without increasing the area consumption. One such way is to run the Giant instructions in parallel to each other or to multiple Dwarf instructions, as shown in [31]. We make sure that two Giants, the Keccak unit and the polynomial arithmetic unit, can always run in parallel to cancel each other's run-time. It leads to a reduction of 35% in the total run-time. Similar to [31], we define two instruction sets (S-1 and S-2), as shown in Table I. Every instruction belonging to the first set can be run in parallel with any instruction that belongs to the second set. The instruction opcodes for each instruction are shown in the INS column of Table I. Following the design methodology, we design the unified cryptoprocessor- `KaLi` as shown in Fig. 1.

IV. Results

In this section, we present the performance and area results of `KaLi`. The proposed architecture is described in Verilog. It is synthesized and implemented for Zynq Ultrascale+ ZCU102 with a performance-optimized strategy using Vivado 2019.1 tool and achieves 270 MHz clock frequency on FPGA. The proposed architecture is also implemented with 65nm and 28nm ASIC technologies using the Cadence Genus tool. On 65nm/28nm ASIC technology, it achieves 280 MHz/1 GHz for the slow clock (in logic units), and 560 MHz/2 GHz for the fast clock (in memory units).

TABLE I
AREA OF KaLi ON THE ZYNQ ULTRASCALE+ ZCU102 FPGA PLATFORM. ALL SECURITY LEVELS OF DILITHIUM AND KYBER ARE SUPPORTED.

	Unit	S-1	S-2	INS	LUT	DFF	DSP	BRAM
	Comp.Core				**21K**	**9.2K**	**4**	**21**
Dilithium (D)	⌊Decompose	✓	-	1	474	338	0	0
	⌊Pow2Round	-	✓	1	55	84	0	0
	⌊MakeHint	-	✓	2	61	124	0	0
	⌊UseHint	-	✓	3	565	433	0	0
	⌊Encode_$\mathcal{H}$	-	✓	4	202	233	0	0
	⌊Pack	✓	-	2	582	181	0	0
	⌊Unpack	✓	-	3	315	182	0	0
	⌊SampleInBall	-	✓	5	505	285	0	0
	⌊Refresh	✓	-	4	8	7	0	0
	⌊VerifyEq.	-	✓	6	13	76	0	0
Kyber (K)	⌊Encode	-	✓	7	517	190	0	0
	⌊Decode	✓	-	5	237	180	0	0
	⌊Com./Decom.	✓	-	6/7	272	376	0	0
	⌊Verify	-	✓	8	102	216	0	0
	⌊CMOV	-	✓	9	20	120	0	0
	⌊COPY	-	✓	10	15	120	0	0
D+K	⌊Memory	-	✓	11	268	12	0	20
	⌊Keccak	✓	-	8-18	12K	3.5K	0	0
	⌊Multiplier	-	✓	12-16	3.5K	2K	4	1
	Prog.Contr.				2K	296	0	3
	Total				**23K**	**9.7K**	**4**	**24**

TABLE II
PERFORMANCE RESULTS FOR DILITHIUM AND KYBER-KEM IN FPGA

Operation	**Dilithium-2 Kyber-512**		**Dilithium-3 Kyber-768**		**Dilithium-5 Kyber-1024**	
	Cycle	μs	**Cycle**	μs	**Cycle**	μs
Dil.Gen	14,594	54.05	23,619	87.48	39,737	147.17
Dil.Sign$_{pre}$	7,883	29.2	9,640	35.7	12,943	46.27
Dil.Sign	21,812	80.79	36,643	135.72	53,965	199.87
Dil.Sign$_{post}$	1,967	7.23	2,463	9.12	3,271	12.12
Dil.Verify	15,423	57.12	26,124	96.76	46,671	172.86
Kyb.Keygen	3,395	12.6	6,291	23.2	9,089	33.7
Kyb.Encaps	4,956	18.4	7,862	29.11	11,351	42.04
Kyb.Decaps	6,807	25.21	11,291	41.82	13,905	51.5

TABLE III
COMPARISON TABLE FOR DILITHIUM-3 FPGA IMPLEMENTATIONS

Ref.	**Plat.**	**Performance** (in μs)	**Freq.** (*MHz*)	**Resources** (LUT/FF/DSP/BRAM)
[9]†	Zynq	-/8.8K/9.9K	100	2.6K/-/-/-
[10]^{a}	US+V	51.9/-/-	350	54.1K/25.2K/182/15
[10]^{b,d}	US+V	-/63.1/-	333	68.4K/86.2K/965/145
[10]^{c}	US+V	-/-/95.1	158	61.7K/34.9K/316/18
[11]^{d}	Ar.-7	229/0.3K/0.2K	145	30.9K/11.3K/45/21
[11]^{e}	Ar.-7	229/0.85K/0.2K	145	30.9K/11.3K/45/21
[6]^{d}	Ar.-7	60/0.12K/63.8	96.9	30K/10.34K/10/11
[6]^{e}	Ar.-7	60/0.46K/63.8	96.9	30K/10.34K/10/11
[12]^{d}	US+V	32/63/39	145	55.9K/28.4K/16/29
[12]^{e}	US+V	32/193/39	145	55.9K/28.4K/16/29
[31]^{d,f}	US+Z	114.7/237/127.6	200	18.5K/9.3K/4/24
KaLi^{d,f}	**US+Z**	**82.8/171.3/96.7**	**270**	**23K/9.7K/4/24**

^{a}: Implements K. Gen. ^{b}: Implements Sign. ^{c}: Implements Verify. ^{d}: Reports best-case scenario. ^{e}: Reports average-case scenario. ^{f}: Supports multiple schemes. †: HW/SW co-design. US+V/Z refers to Virtex/Zynq US+ platforms.

TABLE IV
COMPARISON TABLE FOR DILITHIUM-3 ASIC IMPLEMENTATIONS

Ref.	**Tech.** (*nm*)	**Perf.*** (μs)	**Freq.** (*MHz*)	**Area** (*KGE*)	**SRAM** (*KB*)	**Energy*** (μJ)
[25]†	40	18,266	72	106	40.25	88.89
[24]$^{\dagger a}$	28	747	540	697	24.75	62.39
[31]^{a,b}	65	182.3	400	854	34.82	-
KaLi^{a,b}	**65**	**262.7**	**280/560**	**769**	**34.82**	117.9
KaLi^{a,b}	**28**	**73.55**	**1K/2K**	**747**	**34.82**	27

*:Performance/Energy is measured as total time/energy for signature generation and verification (key generation can be done offline). †: HW/SW co-design. ^{a}:Supports multiple schemes. ^{b}: Reports best-case scenario.

A. Area and Performance Results

Table I presents the detailed utilization of each building blocks in KaLi for UltraScale+ ZCU102 platform. The proposed cryptoprocessor uses 23,347 LUTs (8.4%), 9,798 DFFs (1.7%), 4 DSPs (0.1%), and 24 BRAMs (2.6%). On ASIC, KaLi consumes 1.107 mm^2 (769.04 KGE) in 65nm technology, and 0.263 mm^2 (747.81 KGE) in 28nm technology.

Table II presents the cycle count and latency (in μs) for the operations of Dilithium and Kyber. With 270 MHz clock frequency in the FPGA, the CCA-secure key generation, encapsulation and decapsulation operations for Kyber-768 take 23.2, 29.11, and 41.82 μs, respectively. For the best-case scenario, where a valid signature is generated after the first loop iteration [31], the key generation, signature generation, and signature verification operations for Dilithium-3 take 87.48, 179.91, and 96.76 μs, respectively. The ASIC implementation with 65nm/28nm technology (with 560 MHz/2 GHz clock frequency for the memory unit) can perform the operations for Kyber-768 and Dilithium-3 in 22.07/6.18, 27.59/7.73, and 39.62/11.09 μs, and 82.87/23.2, 171.03/47.89, and 91.66/25.67 μs, respectively. Next, we compare these results with the existing works in the literature.

B. Comparison with unified designs in literature

In [24], the authors design a unified architecture for Dilithium and Kyber. They present both HW/SW co-design as well as HW results for Kyber while keeping some parts of Dilithium in the software. Their NTT unit occupies 25,674 LUTs, 3,137 DFFs, 64 DSPs, and 6 BRAMs on a Xilinx Artix-7 FPGA. The NTT unit alone occupies more LUT and DSP units than our entire design. On ASIC, it occupies 697 KGE on 28nm technology [24] which is very close to our unified cryptoprocessor's 747 KGE area consumption. Their implementation shows similar performance for Kyber even though they target a high-speed design of Kyber in hardware and use 32 butterfly units for NTT, making their NTT unit 8× faster than KaLi. For Dilithium, KaLi shows 10× better results. The energy consumption of KaLi is also approximately half of their design for both Kyber and Dilithium.

To the best of our knowledge, no work exists in the literature that *unifies* Dilithium and Kyber solely in hardware. Therefore, next, we compare our work with standalone implementations of Dilithium or Kyber in hardware.

C. Comparison with Dilithium-only designs in literature

Comparison with FPGA-based implementations:

Different works in the literature use different FPGA platforms. Hence, drawing a one-to-one comparison between works is not always feasible. When we started the hardware implementation of the proposed architecture, we chose Ultrascale+ as the platform and thereafter pipelined the building blocks for achieving around 300 MHz frequency on this platform. Several works in the literature optimized their designs for Artix-7 or other FPGAs. While optimizing the critical paths of architecture for meeting the desired clock frequency is heavily dependent on the technology of the platform, area requirements (LUT/DSP/FF/BRAM) do not change significantly

TABLE V
COMPARISON TABLE FOR KYBER-1024 FPGA IMPLEMENTATIONS

Ref.	Platform	Performance[*] (in μs)	Freq. (*MHz*)	Resources (LUT/ FF/DSP/BRAM)
[21][‡]	Cortex-M4	33,850	100	-/-/-/-
[25][†]	Artix-7	18,560	25	15K/3K/11/14
[18][†]	Zynq	-	-	24K/11K/21/32
[20][†]	Artix-7	85,559	59	2K/2K/5/34
[13]	Virtex-7	1,260	192	133K/-/548/202
[14]	Artix-7	154	161	7K/5K/2/3
[15]	Artix-7	63	210	12K/12K/8/15
[16]	Artix-7	56	185	13K/12K/16/16
[17]	Artix-7	286	112	16K/6K/12/17
[17]	Virtex-7	205	156	16K/6K/12/17
[22]	US+Z	23.5	450	11.6K/12K/8/8.5
[23]	US+Z	3.4 (Encap)	450	18.4K/13.7K/2/0
[23]	US+Z	4.1 (Decap)	450	15.9K/12.9K/2/0
KaLi[a]	**US+Z**	**93**	**270**	**23K/9.7K/4/24**

[*]:Performance is measured as the total time for encapsulation and decapsulation (key generation can be done offline). [a]:Supports multiple schemes. [†]: HW/SW co-design. [‡]: SW design. US+Z refers to Zynq Ultrascale+ platforms.

TABLE VI
COMPARISON TABLE FOR KYBER-1024 ASIC IMPLEMENTATIONS

Ref.	Tech. (*nm*)	Perf.[*] (μs)	Freq. (*MHz*)	Area (*KGE*)	SRAM (*KB*)	Energy[*] (μ*J*)
[25][†]	40	6,444	72	106	40.25	36.06
[18][†]	65	18,444	45	170	465	307.68
[19][†]	28	727	300	979	12	19.57
[17]	65	160	200	104	190	-
[24][†,a]	28	206	540	697	24.75	16.24
[24][a]	28	22/17.7[b]	540	623	36.75	-
KaLi[a]	**65**	**90.2**	**280/560**	**769**	**34.82**	40.48
KaLi[a]	**28**	**25.26**	**1K/2K**	**747**	**34.82**	9.27

[*]:Performance/Energy is measured as the total time/energy for encapsulation and decapsulation (key generation can be done offline). [†]: HW/SW co-design. [a]:Supports multiple schemes. [b]:Depending on the type of schedule.

across FPGA technologies. In Table III, we present the FPGA implementation results of Dilithium-3 from the literature.

Zhou et al. [9] present an HW/SW co-design and they only implement the polynomial arithmetic unit in hardware. Thus, they consume less area but report an inferior performance. Ricci et al. [10] provide separate designs for each of the Dilithium variants. These designs in total occupy 9× more area compared to our design and still perform as good as our design for signature verification. For a signature generation, their implementation shows only 3× improvement. Thus, our design gives a much better area-time trade-off result.

The authors in [6], [11], [12] present Dilithium implementations, which consume much more area compared to our design. Note that across technologies, the area consumption does not change notably. A lower frequency in [6], [11] can be justified by the use of Artix-7 FPGA, which is technologically inferior to our Ultrascale+ platform. A limitation of [6] is that it uses a segmented pipeline and hence, an inflexible data path for Dilithium. The implementation in [12] uses a better platform than ours, consumes 3× more area (LUT+FF), and achieves a speed-up of only 2.5× (Sign+Verify). In [31], the authors present a unified cryptoprocessor for Dilithium and Saber [33]. Their area is almost comparable to ours, considering the difference between Kyber and Saber. We achieve a higher clock frequency and report 1.4× better performance.

Comparison with ASIC-based implementations: Table IV gives the comparison of implementation results for Dilihtium-3 on ASIC platforms. Banerjee et al. [25] present ASIC results for HW/SW co-design of Dilithium with Round 2 parameters. KaLi outperforms them significantly in terms of performance. Our hardware only design gives 45× better performance at the cost of only 7.5× more area. KaLi consumes almost the same area as reported in [24] but gives a 10× and 2.8× better performance with 28nm and 65nm technology, respectively. [31] reports a higher number of logic gates than our design. KaLi sets new records for energy consumption, in both 28nm and 65nm technologies.

Thus, our FPGA and ASIC models are the most compact compared to all the existing Dilithium implementations.

D. Comparison with Kyber-only designs in literature

Comparison with FPGA-based implementations: Table V gives the comparison of implementation results for Kyber-1024 on FPGA platform. Banerjee et al. [25] present an HW/SW co-design for Kyber. KaLi surpasses their performance results on both platforms, at the cost of some area. Observe that KaLi gives better results compared to software only [21] as well as HW/SW co-designs [18], [20], [25]. The hardware-only designs [13]–[17] target Artix-7 or Virtex-7 FPGAs. Our KaLi consumes significantly smaller area than [13]. Authors in [15]–[17], [22] target a high-speed Kyber implementation and therefore achieve better performance and frequency. In [23], the authors present separate results for all the Kyber variants, and present individual encapsulation and decapsulation architectures (unlike KaLi combines all operations in a single architecture). Their design goal is also high-speed, and their standalone implementation of Kyber-1024 encapsulation consumes more area than KaLi. Note that the area of our design is determined by Dilithium and not by Kyber. Therefore, even though the results show that we consume a very high area, we only consume the bare minimum and give almost the best performance results on the FPGA platform.

Comparison with ASIC-based implementations: Table VI gives the comparison of implementation results for Kyber-1024 on ASIC platform. On ASIC platform, KaLi consumes the same area as reported in [24] but gives a 2.3/8.1× better performance under 65nm/28nm technology. In fact, we surpass all existing designs [17]–[19], [25] in terms of performance. However, compared to some of the designs, we use more area, and for this, we must remind again that Kyber is the recessive scheme among the two, and therefore this area is higher when compared to Kyber-only implementations. KaLi consumes the least energy of all for respective technologies.

We have now established that KaLi transcends all the state-of-the-art works that exist in literature. Thus, showing that the proposed design methodology yields better results. Next, we discuss the aspect of application benchmarking.

E. Application-Benchmarking and Impact

Several works, for example, [34], [35], present application benchmarking using the existing libraries. The authors in [34] provide results for TLS protocol using the mbed TLS library and use Kyber for KEM and SPHINCS+ for digital signature. Runtimes are reported for Raspberry Pi 3 Model B+, ESP32-PICO-KIT V4, Fieldbus Option Card, and LPC11U68 LPCXpresso. Compared to these works, `KaLi`'s FPGA implementation shows 85×, 1349×, 6190×, and 23809× speedups, respectively, noting that the ASIC models of `KaLi` will further improve the timings.

In [35], the authors evaluate PQ TLS 1.3, which is a post-quantum variant of TLS version 1.3. It supports Round 3 parameters for both Kyber and Dilithium, along with other schemes. They use ARM Cortex-M4 embedded platform NUCLEO-F439ZI, with and without hardware acceleration. These boards can reach a maximum frequency of 180MHz. Compared to their results for Kyber's decapsulation, we achieve a speedup of 131× if we run our design at 180MHz. The authors report that replacing RSA+ECDHE with Dilithium3+Kyber5 in TLS handshake increased the runtime by 64%. `KaLi` can help bridge this gap. Thus, replacing these devices with `KaLi` would give significant speedup. `KaLi` only occupies 8% of the available resources on the Zynq US+ FPGA board, implying the ability to run twelve such unified cores in parallel, further improving the speedup.

There are several data center and network security appliances where high-performance SIMD processors (e.g., Intel/AMD with AVX) are too expensive to deploy or extremely constrained (or passively powered) devices are too slow to use. There are commercial cryptoprocessors that target such applications. For example, NXP's C29x family of crypto coprocessors [36] (which are battery-powered) use dedicated hardware acceleration for speeding up the RSA and elliptic curve-based public-key cryptographic computations. It computes up to 32K RSA2048 public-key operations per second. More constrained platform OPTIGA™ TPM SLB 9672 from Infenion [37] has hardware acceleration for RSA-4096. They use the same RSA engine for both public-key signature and key agreement.

Our unified Kyber+Dilithium coprocessor performs faster than the public-key engines of [36] and [37] and, at the same time, requires only 0.263 mm^2 area in a 28nm node. When a smaller area is required by an application, some of the design parameters (e.g, number of NTT cores, Keccak's data-width, etc.) can be tuned accordingly to meet the area budget at the cost of speed. The proposed design techniques and architecture will be useful to replace the classical public-key cryptography used in conventional cryptoprocessors with post-quantum key agreement and signature.

V. Conclusions and future work

Post-quantum key encapsulation and digital signature algorithms are required for securing communication. In this paper, we presented a design methodology for efficient and compact hardware implementation of both post-quantum key encapsulation and digital signature algorithms in a unified cryptoprocessor architecture. Following the proposed methodology, we designed and implemented the first unified cryptoprocessor architecture `KaLi` that can perform all the cryptographic protocol operations of the Dilithium signature and Kyber key encapsulation algorithms for all the security levels. Architectural optimizations in the data path of the cryptoprocessor were performed to reduce the cycle count and improve the clock frequency. Experimental evaluation of `KaLi` on FPGA and ASIC platforms showed that `KaLi` outperforms all the existing implementations. Therefore, the design of `KaLi` is a significant step towards making post-quantum cryptography compact and agile on hardware platforms. The proposed design methodology can be customized to meet different application-specific constraints and requirements.

The hardware implementation presented in this paper is resistant to timing attacks but does not incorporate any countermeasure, for example, masking against more powerful side-channel attacks. Side-channel protection of the unified cryptoprocessor architecture will require significant research and is considered future work. There are several works in the literature on masking Kyber [38], [39]. However, at the time of writing this paper, the authors are not aware of any reported masked implementation of the NIST standardized version of Dilithium. How to design an 'optimal and unified' side-channel protection mechanism for a unified hardware implementation of Kyber and Dilithium is an interesting topic that needs to be researched. Furthermore, researching protection techniques against fault injection-based attacks will be very important due to the vast deployment of these cryptographic schemes in various embedded devices.

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, p. 1484–1509, oct 1997.

[2] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, and many more, "Quantum supremacy using a programmable superconducting processor," Nature, 2019, https://doi.org/10.1038/s41586-019-1666-5.

[3] M. Gong, S. Wang, C. Zha, M.-C. Chen, H.-L. Huang, Y. Wu, Q. Zhu, Y. Zhao, S. Li, S. Guo, and e. a. Haoran Qian, "Quantum walks on a programmable two-dimensional 62-qubit superconducting processor," *Science*, vol. 372, no. 6545, pp. 948–952, 2021.

[4] "Post-quantum cryptography- call for proposals," 2017. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography

[5] D. Joseph, R. Misoczki, M. Manzano, J. Tricot, F. D. Pinuaga, O. Lacombe, S. Leichenauer, J. Hidary, P. Venables, and R. Hansen, "Transitioning organizations to post-quantum cryptography." *Nature, 605(7909), 237–243.*, 2022.

[6] C. Zhao, N. Zhang, H. Wang, B. Yang, W. Zhu, Z. Li, M. Zhu, S. Yin, S. Wei, and L. Liu, "A compact and high-performance hardware architecture for crystals-dilithium," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2022, no. 1, pp. 270–295, 2022.

[7] P. Schwabe, R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, G. Seiler, and D. Stehle, "CRYSTALS-KYBER," Proposal to NIST PQC Standardization, 2021, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[8] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Crypt. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 443–466, 2020.

[9] Z. Zhou, D. He, Z. Liu, M. Luo, and K.-K. R. Choo, "A software/hardware co-design of crystals-dilithium signature scheme," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 14, no. 2, Jun. 2021.

[10] S. Ricci, L. Malina, P. Jedlicka, D. Smékal, J. Hajny, P. Cibik, P. Dzurenda, and P. Dobias, "Implementing crystals-dilithium signature scheme on fpgas," in *The 16th International Conference on Availability, Reliability and Security*, ser. ARES 2021. New York, NY, USA: Association for Computing Machinery, 2021.

[11] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal - implementing dilithium on reconfigurable hardware," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 355, 2021.

[12] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of crystals-dilithium," Crypto. ePrint Arch., Report 2021/1451, 2021.

[13] Y. Huang, M. Huang, Z. Lei, and J. Wu, “A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse,” *IEICE Electron. Express*, vol. 17, no. 17, p. 20200234, 2020.

[14] Y. Xing and S. Li, “A compact hardware implementation of cca-secure key exchange mechanism CRYSTALS-KYBER on FPGA,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 2, pp. 328–356, 2021.

[15] V. B. Dang, F. Farahmand, M. Andrzejczak, K. Mohajerani, D. T. Nguyen, and K. Gaj, “Implementation and benchmarking of round 2 candidates in the NIST post-quantum cryptography standardization process using hardware and software/hardware co-design approaches,” *IACR Cryptol. ePrint Arch.*, p. 795, 2020.

[16] M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, “High-speed ntt-based polynomial multiplication accelerator for crystals-kyber post-quantum cryptography,” *IACR Cryptol. ePrint Arch.*, p. 563, 2021.

[17] M. Bisheh-Niasar, R. Azarderakhsh, and M. M. Kermani, “Instruction-set accelerated implementation of crystals-kyber,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 68, no. 11, pp. 4648–4659, 2021.

[18] T. Fritzmann, G. Sigl, and J. Sepúlveda, “RISQ-V: tightly coupled RISC-V accelerators for post-quantum cryptography,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 4, pp. 239–280, 2020.

[19] G. Xin, J. Han, T. Yin, Y. Zhou, J. Yang, X. Cheng, and X. Zeng, “VPQC: A domain-specific vector processor for post-quantum cryptography based on RISC-V architecture,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 67-I, no. 8, pp. 2672–2684, 2020.

[20] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, “ISA extensions for finite field arithmetic accelerating kyber and newhope on RISC-V,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 219–242, 2020.

[21] L. Botros, M. J. Kannwischer, and P. Schwabe, “Memory-efficient high-speed implementation of kyber on cortex-m4,” in *Progress in Cryptology - AFRICACRYPT 2019*, vol. 11627. Springer, 2019, pp. 209–228.

[22] V. B. Dang, K. Mohajerani, and K. Gaj, “High-speed hardware architectures and FPGA benchmarking of crystals-kyber, ntru, and saber,” *IACR Cryptol. ePrint Arch.*, p. 1508, 2021.

[23] Z. Ni, A. Khalid, D. Kundi, M. O’Neill, and W. Liu, “Efficient pipelining exploration for A high-performance crystals-kyber accelerator,” *IACR Cryptol. ePrint Arch.*, p. 1093, 2022.

[24] Y. Zhao, R. Xie, G. Xin, and J. Han, “A high-performance domain-specific processor with matrix extension of RISC-V for module-lwe applications,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 69, no. 7, pp. 2871–2884, 2022.

[25] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, “Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols (extended version),” *IACR Cryptol. ePrint Arch.*, p. 1140, 2019.

[26] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “CRYSTALS-Dilithium,” Proposal to NIST PQC Standardization, Round3, 2021, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[27] D. Sprenkels, “The Kyber/Dilithium NTT,” https://dsprenkels.com/ntt.html.

[28] M. Scott, “A note on the implementation of the number theoretic transform,” in *Cryptography and Coding - 16th IMA International Conference, IMACC 2017*. Springer, 2017.

[29] V. Lyubashevsky and G. Seiler, “NTTRU: Truly Fast NTRU Using NTT,” *IACR Trans. on CHES*, vol. 2019, no. 3, pp. 180–201, May 2019.

[30] F. Yaman, A. C. Mert, E. Öztürk, and E. Savaş, “A hardware accelerator for polynomial multiplication operation of crystals-kyber pqc scheme,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1020–1025.

[31] Aikata, A. C. Mert, D. Jacquemin, A. Das, D. Matthews, S. Ghosh, and S. S. Roy, “A unified cryptoprocessor for lattice-based signature and key-exchange,” Cryptology ePrint Archive, Report 2021/1461, 2021.

[32] Y. Xing and S. Li, “A compact hardware implementation of cca-secure key exchange mechanism crystals-kyber on fpga,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 328–356, 2021.

[33] J.-P. D’Anvers, A. Karmakar, S. S. Roy, F. Vercauteren, J. M. B. Mera, M. V. Beirendonck, and A. Basso, “SABER,” Proposal to NIST PQC Standardization, Round3, 2021, https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions.

[34] K. Bürstinghaus-Steinbach, C. Krauß, R. Niederhagen, and M. Schneider, “Post-quantum TLS on embedded systems: Integrating and evaluating kyber and SPHINCS+ with mbed TLS,” in *ASIA CCS ’20: The 15th ACM Asia Conference on Computer and Communications Security, 2020*. ACM, 2020, pp. 841–852.

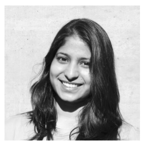
[35] T. George, J. Li, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, “Performance evaluation of post-quantum TLS 1.3 on embedded systems,” *IACR Cryptol. ePrint Arch.*, p. 1553, 2021.

[36] “Nxp’s c29x family of crypto coprocessors.” [Online]. Available: https://www.nxp.com/docs/en/fact-sheet/C29XFAMFS.pdf

[37] “Optiga™ tpm slb 9672 from infenion.” [Online]. Available: https://www.infineon.com/cms/en/about-infineon/press/market-news/2022/INFCSS202202-051.html

[38] J. W. Bos, M. Gourjon, J. Renes, T. Schneider, and C. van Vredendaal, “Masking kyber: First- and higher-order implementations,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, pp. 173–214, 2021.

[39] T. Fritzmann, M. Van Beirendonck, D. Basu Roy, P. Karl, T. Schamberger, I. Verbauwhede, and G. Sigl, “Masked accelerators and instruction set extensions for post-quantum cryptography,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 1, p. 414–460, Nov. 2021.



Aikata obtained her Bachelors in Technology degree from IIT Bhilai, India, in 2020 and Masters degree from Graz University of Technology, Austria, in 2022. She is currently a PhD student at Institute of Applied Information Processing and Communications, Graz University of Technology. Her research interests include lattice-based cryptography and hardware design.

Ahmet Can Mert received his PhD degree in electronics engineering from Sabanci University, Turkey in 2021. Currently, he is working as a postdoctoral researcher at the Institute of Applied Information Processing and Communications, Graz University of Technology, Austria. His research interest include homomorphic encryption, lattice-based cryptography and hardware design.

Malik Imran received his bachelor’s and master’s degrees from Pakistan in 2011 and 2015, respectively. Now, he is in with the Center for Hardware Security, Tallinn University of Technology (TalTech), Tallinn, Estonia, as a doctoral student. Before joining TalTech, Malik contributed to different research labs for efficient hardware accelerators for intrusion detection systems and asymmetric cryptography.

Samuel Pagliarini (M’14) received the PhD degree from Telecom ParisTech, Paris, France, in 2013. He has held research positions with the University of Bristol, Bristol, UK, and with Carnegie Mellon University, Pittsburgh, PA, USA. He is currently a Professor with Tallinn University of Technology (TalTech) in Tallinn, Estonia where he leads the Centre for Hardware Security.

Sujoy Sinha Roy is an Assistant Professor of cryptographic engineering at IAIK, Graz University of Technology. He is a Co-Designer of “Saber,” which is a finalist key encapsulation mechanism (KEM) candidate in NIST’s Post-Quantum Cryptography Standardization Project. He is interested in the implementation aspects of cryptography.