Special Session: The Recent Advance in Hardware Implementation of Post-Quantum Cryptography

Jiafeng Xie[1], Kanad Basu[2], Kris Gaj[3], Ujjwal Guin[4]
[1]Department of Electrical & Computer Engineering, Villanova University (jiafeng.xie@villanova.edu)
[2]Department of Electrical & Computer Engineering, University of Texas at Dallas (kanad.basu@utdallas.edu)
[3]Department of Electrical & Computer Engineering, George Mason University (kgaj@gmu.edu)
[4]Department of Electrical & Computer Engineering, Auburn University (ujjwal.guin@auburn.edu)

***Abstract*—The recent advancement in quantum technology has initiated a new round of cryptosystem innovation, i.e., the emergence of Post-Quantum Cryptography (PQC). This new class of cryptographic schemes is intended to be mathematically resistant against any known attacks using quantum computers, but, at the same time, be fully implementable using traditional semiconductor technology. The National Institutes of Standards and Technology (NIST) has already started the PQC standardization process, and the initial pool of 69 submissions has been reduced to 26 Round 2 candidates. Echoing the pace of the PQC "revolution," this paper gives a detailed and thorough introduction to recent advances in the hardware implementation of PQC schemes, including challenges, new implementation methods, and novel hardware architectures. Specifically, we have: (i) described the challenges and rewards of implementing PQC in hardware; (ii) presented the novel methodology for the design-space exploration of PQC implementations using high-level synthesis (HLS); (iii) introduced a new underexplored PQC scheme (binary Ring-Learning-with-Errors), as well as its novel hardware implementation for possible lightweight applications. The overall content delivered by this paper could serve multiple purposes: (i) provide useful references for the potential learners and the interested public; (ii) introduce new areas and directions for potential research to the VTS community; (iii) facilitate the PQC standardization process and the exploration of related new ways of implementing cryptography in existing and emerging applications.**

Keywords: Post-quantum cryptography, hardware implementation, challenges, high-level synthesis, binary Ring-LWE

I. Introduction

The rapid advancement in quantum computing has triggered a new round of cryptographic engineering research, as the existing public-key cryptosystems, such as Rivest Shamir Adleman (RSA) and elliptic curve cryptography (ECC), are likely to become vulnerable to the polynomial-time realizations of the Shor's algorithm using quantum computers [1], [2]. As it is anticipated that a well-equipped quantum computer will become available during the next 10-15 years, alternative solutions, capable of replacing widely-deployed RSA and ECC standards, are truly needed. Post-quantum cryptography (PQC) is defined as a class of cryptosystems that can resist quantum attacks [1], but at the same time can be implemented using traditional semiconductor technology. In recent years, PQC has gained substantial attention from the research community and governmental institutions. Many research projects, supported by thorough security analysis, have been carried out on developing practical and secure PQC schemes. Meanwhile, the National Institute of Standards and Technology (NIST) has already initiated the PQC standardization process, and, as of the first half of 2020, 26 submissions remain under consideration for future standards [3].

During the first round of the NIST evaluation process, conducted in the period between Dec. 2017 and Jan. 2019, the primary attention was focused on security. Initially, only reference software implementations existed for the majority of the candidates. Optimized software implementations, taking advantage of vector instructions of modern microprocessors, followed. Only then, the first software/hardware and purely hardware implementations were reported [3].

Although pilot studies, concerning hardware implementations of earlier versions of PQC algorithms, were conducted well before an official launch of the NIST standardization process, the results of these studies need to be used with considerable caution. This is because multiple changes in the functionality and parameter values of even well-established candidates, such as NTRUEncrypt, Rainbow, Unbalanced Oil-and-Vinegar, McEliece, have been introduced at the launch of both rounds of the NIST evaluation [4]. Additionally, the old implementations used very divergent assumptions, optimization targets, APIs, and sources of randomness. Only a few candidates in the NIST PQC standardization process have been fully implemented in hardware to date.

In this paper, we present three aspects of recent advances in hardware implementation of PQC:

- We give detailed analysis of unique challenges related to hardware implementation of PQC, ranging from the fundamental mathematical complexity to the difficulty of protection against side-channel and fault attacks.
- We introduce an overall design flow for the high-level synthesis (HLS)-based implementation of the NIST PQC candidates. Main topics include: (i) motivation for employing HLS to design accelerators for the NIST PQC candidates; (ii) proposed optimization steps based on HLS approach; (iii) practical implementation challenges and recent advances in the HLS implementation of PQC schemes.
- We also introduce a new strategy to implement a lattice-based PQC scheme, called binary Ring-Learning-with-

Errors (BRLWE), on a hardware platform. The BRLWE is a new variant of the Ring-LWE scheme and has the potential to be used in many resource-constrained applications due to its small key size and very low implementation complexity. The thorough research on efficient hardware implementation of this new scheme has not been well reported in the literature to date.

The rest of this paper is organized as follows. Section II discusses the challenges and potential rewards of implementing PQC in hardware. Section III introduces the HLS-based exploration of PQC implementations on a hardware platform. A novel BRLWE implementation, as well as the corresponding algorithm, are proposed in Section IV. Finally, the conclusions are given in Section V.

II. Challenges and Rewards of Post-Quantum Cryptography Revolution

Implementing PQC algorithms in hardware involves a unique set of challenges. Some of these challenges are specific to the current state of standardization, in which a large number of candidates still remain, and the primary goal of the implementation is a fair evaluation of submissions to the NIST standardization process. Other challenges will remain in place even after the first PQC standards are published.

At this point, one of the biggest challenges seems to be the mathematical complexity of the specifications, which are often written by cryptographers, concerned primarily with demonstrating the security of their schemes, rather than the ease of their implementation. Understanding these specifications often requires a solid background in number theory, abstract algebra, coding theory, and other related disciplines. Hardware implementers, with a classical background in computer engineering, are often ill-equipped with understanding these specifications, which are frequently riddled with complex formulas and high-level mathematical operations that need to be "deciphered" by reading and understanding the corresponding C source code. Clearly, a reference implementation in C, although helpful and potentially suitable as an input to high-level synthesis (HLS) tools, may also hide a lot of potential optimizations, possible only in hardware, and accomplished by performing a given high-level operation using a different low-level algorithm.

At the current stage of the standardization process, another major challenge is the fact that the remaining candidates belong to five different families: lattice-based, code-based, multivariate, symmetric-based, and isogeny-based [3]. Each of these families has several subfamilies. For example, lattice-based candidates are divided into schemes with structured (a.k.a. random) and unstructured (a.k.a. ideal) lattices. Similarly, code-based schemes have been classified by NIST into three subfamilies: Algebraic, Short Hamming (a.k.a Quasi-Cyclic), and Low Rank [3]. Each family or even subfamily requires a different mathematical background, involves a different set of basic operations, and poses its own set of optimization challenges.

In general, basic operations of PQC schemes are very different from those used by traditional public-key schemes, such as RSA or ECC. In particular, the operations on large integers (such as modular multiplication modulo a large prime or a product of two large primes), determining the complexity of the majority of current public-key cryptography standards, remain relevant in only one out of five major PQC families, isogeny-based cryptography [4]. Some of the new operations, found in various PQC schemes, offer major advantages, such as higher potential for parallelization, ability to use optimization methods developed as part of different branches of science and engineering (such as Fast Fourier Transform [3], typically associated with digital signal and image processing), or suitability for very compact implementations. Others pose new challenges, such as sequential nature, large memory requirements, or variable execution time.

Some of the underlying operations might have never been implemented in hardware before. For example, codes used in code-based cryptography are typically different from those commonly used to achieve the error-free communication [3]. As a result, there may have never been a strong incentive to implement the corresponding coding and decoding algorithms in hardware. The in-depth knowledge of coding theory required for optimized implementations of these algorithms has been for years beyond the scope of traditional cryptographic engineering, computer arithmetic, or even computer engineering. Additionally, in other branches of science and engineering, there is no tradition of making the code of best implementations available as open-source, which has been a major factor driving improvements in cryptographic engineering (see, e.g., [5] and [6]). The number of publications on hardware implementations of candidates for PQC standards is still relatively small, and the reported results very difficult to compare due to divergent assumptions and optimization targets. Consequently, little is known about an optimal way of implementing these schemes in hardware, targeting various optimization criteria, such as minimum area, power, and energy per bit; maximum speed; and minimum product of latency times area.

An additional challenge is that a significant percentage of candidates submitted to Round 1 of the NIST standardization process has been partially broken. These breakthroughs discouraged any early attempts at hardware and optimized software implementations. The common perception was that such implementations were premature and involved considerable risk. The worst-case scenario involved a major implementation and optimization effort, followed by an inability to publish results because, in the meantime, the target of the investigation was shown vulnerable to powerful attacks, and hence deemed unsuitable for future standardization.

Due to the mathematical and algorithmic complexity of the PQC algorithms, and the limited amount of previous work, the workload for even a single algorithm, implemented using the traditional RTL approach, can easily reach several man-months. Close collaboration with the algorithm designers is highly recommended and often indispensable.

The HLS approach, attempted already during the previous cryptographic competition, the CAESAR contest [6], [7], appears to be very appealing (especially in light of a very large number of candidates and the availability of reference and optimized implementations in C). However, using this approach offers its own set of challenges, especially when used for ranking candidates competing to become future national and international standards. These challenges include a potential for unfair comparisons, due to the lack of any clear indication when the optimization of the HLS-ready C code should end, and the limited experience of both hardware designers and software programmers with the HLS methodology, especially when applied to algorithms from the domains other than digital signal and image processing. As a result, different approaches to this problem emerged, as described in [8]–[11]. In order to address concerns regarding the fairness of the comparison, a parallel RTL implementation may still be required to confirm any HLS-based performance and resource utilization estimates [7], [9], [10].

Another major challenge is the storage of large public keys, private keys, and internal state, inside of a hardware module, which may prohibit truly lightweight implementations and effect the key agility of all hardware implementations. For example, one of the primary candidates for adoption as an early PQC standard, Classic McEliece [3], has public-key sizes exceeding a quarter of a megabyte, half-a-megabyte, and one megabyte, for its three major parameter sets, with the security levels equivalent to various variants of AES. These numbers need to be compared with the public-key sizes in the range of 256-512 bytes (2048-4096 bits) for RSA, and 32-64 bytes (256-512 bits) for ECC. NIST Round 2 PQC candidates belonging to the subfamily based on unstructured lattices have the second largest set of key sizes, in the range of 4096-8192 bytes. At the same time, it should be noted that at least one family, isogeny-based cryptography, has public key sizes smaller than or equal to those used in RSA. Large public keys may prevent low-area, low-memory implementations suitable for the Internet of Things. They may also prevent caching a large number of keys, thus adding the key loading time to the overall time required for encryption.

Decryption failures, possible in a significant subset of PQC schemes, may force repeating time-consuming computations, and thus increase both average and worst-case decryption times. Another difficulty is the requirement for random numbers, used as inputs for encryption, signature generation, and key encapsulation with the specific (e.g., the uniform or Gaussian) distribution. The use of random samplers may require access to a True Random Number Generator, as well as conversions between random values with different distributions. This kind of circuits are rarely used in other digital system applications and may need to be developed from scratch. These circuits are also difficult to validate and may become the source of considerable side-channel leakage.

Not much is known, to date, about the resistance of PQC schemes against side-channel attacks. Potential threats are based on timing, power or electromagnetic analysis, and cache manipulation and leakage. For high-speed hardware and hardware/software implementations, used as accelerators for high-end servers, protecting against timing attacks might be sufficient, as no physical access to the accelerator by a potential attacker is expected. For lightweight implementations, targeting constrained environments and mobile devices, an attacker is assumed to have easy physical access to the device during its regular operation. Thus, additional protection against power and electromagnetic attacks is expected. Any embedded software implementation, especially if targeting platforms on which multiple users share common resources, may require protection against cache leakage and manipulation attacks.

Unlike it was the case for secret-key algorithms, the majority of countermeasures against side-channel attacks are algorithm-specific, and yet unexplored for the majority of PQC schemes. Their development and experimental validation is likely to require a very considerable and prolonged effort, including substantial modifications and extensions of experimental frameworks [12]. Even constant-time implementations, necessary to protect both high-speed and lightweight implementations against powerful timing attacks, are non-trivial to develop for many PQC algorithms [13], [14].

Equally important is protection against fault attacks, in which attackers induce faults on a particular unit of a hardware implementation or a specific phase of computations. The more control the attacker has over the exact nature, location (in time and space), and outcome of a fault, the harder it is to protect against these attacks.

Similarly, little is known about the optimal ways of partitioning public-key and private-key operations into software and hardware. Research in this area has just started [9], [10], [14]–[16]. The total speed-up obtained by offloading an operation to hardware depends on two major factors: the percentage of the execution time taken in software by the operation offloaded to hardware, and the speed-up for the offloaded operation itself. In order to maximize the first factor, the priority needs to be given to operations that take the largest percentage of the execution time. These operations may involve a single function call, several adjacent function calls, or a sequence of consecutive instructions. It is preferred that a given operation is executed only once, or only a few times, as each transfer of control and data between software and hardware involves a certain fixed timing overhead, independent of the size of input and output to the accelerator. In order to maximize the second factor, the priority needs to be given to operations that have a high potential for parallelization in hardware, and a small total size of inputs and outputs (which will need to be transferred to and from the hardware accelerator, respectively).

Most of the data required to make informed decisions regarding software/hardware partitioning can be obtained by profiling the purely-software implementation, possibly extended with some small modifications required to gather all relevant data. However, determining the potential for parallelization requires some knowledge of hardware or at least basic concepts of concurrent computing.

At the stage of evaluating candidates for future standards, in order to assure the fairness of the comparison, it is instrumental to offload to hardware all operations common to or similar across the implemented algorithms, and all operations that contributed significantly to the total execution time. Nevertheless, it should be understood that this heuristic procedure may need to be repeated several times because, after each round of offloading to hardware, different software operations may emerge as taking the majority of the total execution time. This process can stop when the development effort required for offloading the next most-critical operation to hardware is disproportionately high compared to the projected speed-up.

In order to simplify the deployment of the newly developed algorithms in real-life applications, such as hardware accelerators for TLS, IPSec, and SSH, the implemented modules will need to be plug-and-play replacements for existing public-key cryptography modules, based on classical schemes. To make matters even more challenging, in the transition period (likely to last at least several years) hybrid systems, based on both types of public-key algorithms (classical and post-quantum) are likely to be used simultaneously (in the parallel or cascade fashion). Such hybrid systems will protect against both the progress in quantum computing (by employing a PQC scheme), as well as potential vulnerability of emerging new NIST PQC standards to novel and hard-to-predict attacks using conventional codebreaking machines (by employing an existing and time-tested NIST standard, such as ECC). An additional challenge is that the hybrid modes have not been clearly defined yet and are beyond the scope of the current NIST standardization effort [4].

Other challenges, present at the time of evaluating candidates for future standards, and common with previous cryptographic competitions, include no clear performance target (caused by a large variety of future applications and possible implementations platforms), the fact that the existing field-programmable gate array (FPGA) and application-specific integrated circuits (ASIC) tools are not designed to operate under such assumptions (and instead attempt to always reach a predefined target, rather than the absolute maximum) [17], the difficulty of describing resource utilization in FPGAs using a single metric (due to the use of diverse resources, such as configurable logic blocks, DSP units, memory blocks, hardwired embedded microprocessors, etc.), and differences among the skills and time commitment of hardware designers responsible for implementing particular algorithms.

The first step in addressing some of the aforementioned challenges was the development of the PQC Hardware API [18]. This API includes the minimum compliance criteria, interface, communication protocol, and timing characteristics, aimed at the fair evaluation of all developed implementations. Two approaches to overcome the long development time have emerged. The first is software/hardware codesign using either hard-core processors, such as ARM Cortex-A53 present in Zynq UltraScale+ MPSoC [9], [10], [16], or soft-core processors, such as RISC-V, embedded inside of traditional FPGAs, such as Xilinx Artix-7 [14], [15]. The second is the use of HLS [8], [9]. The results of the corresponding RTL and HLS implementations have been compared for the first time in [9], [10]. Multiple implementations of SCA-protected implementations of lattice-based PQC schemes have been reported in [12], [19]–[23]. The next step might be the creation of the hardware library of basic building blocks of major PQC schemes competing in Round 2 of the NIST PQC Standardization Process.

Table I: A Summary of PQC Algorithms Selected for NIST Round 2

PQC	Signature	Encryption/KEM	Total
Lattice	3	9	12
Code	0	7	7
Multivariate	4	0	4
Symmetric	2	0	2
Isogeny	0	1	1
Total	**9**	**17**	**26**

As a result, NIST and other standardization organizations, as well as cryptographers from all over the world involved in developing and/or evaluating PQC submissions could greatly benefit from the involvement of the VTS community. They count on this community's contributions to come up with the most efficient and secure hardware architectures for a multitude of PQC candidates, as well as less time-consuming, but accurate and fair, implementation and benchmarking approaches.

The next 5–10 years are very likely to bring the biggest revolution in cryptography since the invention of public-key cryptography in mid-1970s. The majority of software and hardware implementations of public-key cryptography will need to be first enhanced and then gradually replaced by implementations of their PQC counterparts. By getting involved in implementation of novel PQC schemes, hardware designers have a unique opportunity to influence the choice of future cryptographic standards, which are likely to be developed within the next decade and remain in use for the significant portion (if not the rest) of the 21st century. An early involvement offers multiple advantages in terms of new unexplored research directions, visibility in the cryptographic community, as well as multiple entrepreneurial opportunities to make a difference in the emerging post-quantum world.

III. High level synthesis-based exploration of PQC implementations

As discussed in Section I, NIST has started the PQC standardization process and has selected 26 candidates for the second round. Out of these 26 candidates, 9 are signature schemes and 17 are encryption/key encapsulation mechanism (KEM) schemes. Table I gives an overview of all the 26 algorithms. Both KEMs and signature schemes can be further classified into lattice-based, code-based, multivariate, symmetric-based and isogeny-based [4].

All submissions to the NIST standardization process have the corresponding reference implementation in C/C++. In some cases, optimized implementations in the same languages are provided as well. In this section, we will discuss techniques to design dedicated hardware accelerators from these software implementations using HLS. There are several advantages for designing dedicated accelerators. For example, if these accelerators are being used in servers, they can be designed to optimize latency. On the other hand, if these accelerators are designed for low power Internet of Things (IoT) devices, they can be designed to optimize area and power. First, we will explain and motivate the reasons for using HLS for designing these accelerators.

A. *Introduction to High-Level Synthesis*

Over the past decade, there has been a wide performance gap between Moore's Law and current technology development. As a result, researchers have started developing dedicated customized hardware accelerator to bridge this gap. Hardware accelerators are being used for deep learning, cryptography, and image processing, among many other applications. HLS has found its way to be a popular method for designing these accelerators. HLS starts with a high-level C/C++ implementation and translates that to a RTL (Verilog/VHDL) backend hardware implementation. HLS incorporates an advantage of being able to perform thorough design space exploration. The same C/C++ code can be used to generate multiple RTLs, each with a different area and latency. Another advantage of using HLS is easy verification. The same high-level C/C++ testbench can be used for verifying a RTL, using an unique feature known as C-RTL co-verification.

These advantages have led a lot of premier semiconductor companies to adopt HLS for designing accelerators. The prominent commercial tools include Xilinx Vivado HLS and Mentor Catapault. HLS uses the classical Finite State machine with Data (FSMD) unit to design hardware components, which are arranged hierarchically, according to the various function calls in the C design specification. HLS design flow consists of the following steps:

- **Compiler Phase:** In this phase, the HLS compiler reads in the C description of the algorithm and applies various compiler-level transformations.
- **HLS Phase:** This phase is the most critical phase of HLS. The FSM of the design is obtained, and this is used for scheduling, i.e., deciding the timing of different tasks to be performed in the algorithm. The next step in this phase is binding, where various hardware resources are allocated to the design units. For example, C-based arrays can be assigned to memory units. In the final stage, the controller and datapath are generated. The controller is obtained from the FSM.
- **Backend Phase:** In this phase, the controller and datapath are synthesized to generate the RTL and testbenches. C-RTL co-simulation verifies the design functionality.

The overall flow of HLS-based design is shown in Figure 1.

B. *HLS-based optimizations*

As discussed before, HLS provides various optimizations for detailed design space exploration. In this section, we will discuss two such optimizations, loop unrolling and loop pipelining.

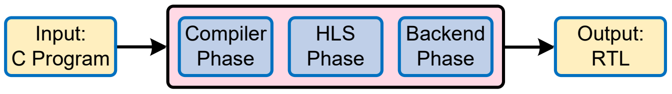


Figure 1: HLS design flow.

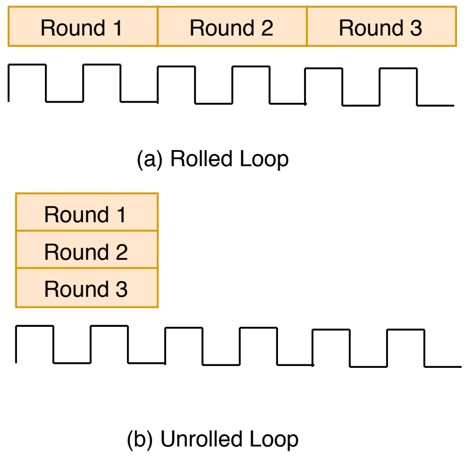
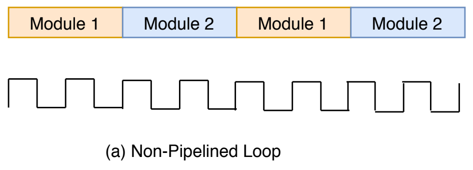


Figure 2: Loop Unrolling.

1) Loop Unrolling: The latency of a design depends on the cumulative latency of each of the internal components. For example, consider a loop consisting of three rounds, as shown in Figure 2(a). If the latency of each round is 2 cycles, the total latency of the entire operation is 6 cycles. However, if there are no inter-dependencies, HLS provides an optimization, known as loop unrolling, as shown in Figure 2(b). In this case, each round is executed in parallel, and hence, the total latency depends on the maximum latency of one round, which is 2 cycles. Therefore, we obtain a three times speedup. Hence, this form of implementation is preferred when latency needs to be optimized. On the other hand, in a rolled loop, the same functional unit can be re-used. For example, in Figure 2(a), only one functional unit is sufficient for all the three rounds. On the other hand, when a loop is unrolled, each round requires its own functional unit; thus, resulting in an increased area overhead.

2) Loop Pipelining: Let us consider a loop with multiple modules, as shown in Figure 3(a)[*]. In the normal scenario, when there are no pipeline stages, these modules are executed sequentially, and the overall latency is 8 clock cycles. Loop pipelining entails re-purposing a module when its role in one loop is finished. For example, in Figure 3(b), when module 2 starts the first loop, the hardware unit reserving module 1 is empty. As a result, module 1 of loop 2 can proceed with its operation. Thus, the overall latency is reduced from 8 to 6 cycles. It should be noted that unlike for the loop unrolling, individual loop latency does not reduce; it is still 4 cycles.

[*]In this case, we have two modules.



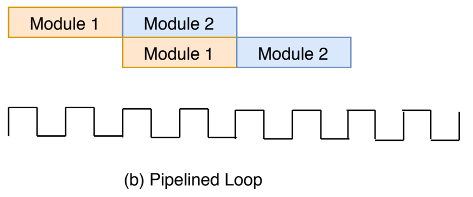


Figure 3: Loop Pipeline.

C. Challenges for HLS-based PQC implementations

Although HLS provides rapid translation of high-level (C/C++)-based designs to RTL, there are certain restrictions. All C/C++ constructs are not translatable by HLS, and hence, these need to be taken care of before the C code is used for synthesis. In this section, we will describe some of them:

- **Dynamic arrays:** HLS cannot synthesize dynamic arrays. Hence, it is important to rewrite them as static array variables. For example, let us consider a function with a dynamically sized array as an input parameter. This function can be instantiated multiple times, with the size of the array being dynamically allocated in each instant. HLS will not be able to translate this array into RTL code. It is necessary to rewrite the function in order to make this array have a fixed dimension, before using HLS.
- **System Function:** System functions like "memcpy" cannot be translated using HLS. It is necessary to either rewrite those functions or use invariants, such as loops, to represent them.
- **Pointers:** HLS can not translate pointers to RTL. Therefore, pointers must be converted to static variables before applying HLS.
- **Recursion:** HLS can not translate recursion to RTL. Any PQC algorithm with a recursive code needs to be rewritten before applying HLS.

D. Recent Advances

Application of HLS for designing post-quantum cryptography accelerators was first proposed by [8], [24]. The authors used 13 algorithms in their study, of which, 7 were KEMs and 6 signature schemes. The algorithms were implemented on a Xilinx Virtex-7 FPGA. The authors also applied the two optimization techniques mentioned in Section III-B, for both encapsulation and decapsulation algorithms. It was shown that the maximum latency reduction achieved was 45x, while the maximum increase in area overhead was 12x. The authors further improved on their approach to develop power, area, speed, and security (PASS) tradeoffs using a C to ASIC design flow [25]. The authors developed ASIC designs for two PQC algorithms – qTESLA and Crystals Dilithium. It was demonstrated that for both algorithms, higher security level entails more power, area, and timing overhead. Furthermore, memory requirements of both algorithms amounted to almost 50% of the total ASIC area. The authors have published a thorough description of these implementations in [26]. A video description of HLS-based implementation of the PQC algorithm CRYSTALS-Dilithium can be found in [27].

In [11], the authors used HLS to perform a design space exploration for Number Theoretic Transform (NTT) - the core arithmetic function for lattice-based cryptography. The authors compared their implementation with the hand-coded RTL design and concluded that the HLS-generated implementations have significantly lower performance. However, HLS provides an important advantage of fast design space exploration, which is inconvenient using hand-written RTL.

Although, over the past year, there have not been a plethora of work on applying HLS to designing PQC accelerators, we have seen several important articles, as described in this section. In the future, we expect several more papers on the same topic, especially on the application of HLS-based design space exploration to the design of PQC accelerators.

IV. Novel implementation of Binary Ring-LWE PQC (A New Underexplored Scheme)

A. Overall Introduction

Ring-LWE lattice-based cryptography is one of the most promising candidates for PQC standardization [4], [28] due to its relatively low computational complexity and ease of implementation (as demonstrated by many studies carried out on hardware platforms [21], [29]–[33]). Recently, a new variant of Ring-LWE, namely the BRLWE scheme, has been introduced [32]. Unlike the Ring-LWE, where the errors are based on Gaussian distribution, the BRLWE scheme uses binary errors to achieve much smaller computational complexity [32]. The security analysis of the LWE scheme based on binary errors has been recently given in [34]. Since the first introduction in [32], where the cryptosystem was realized only on the software platform, two consecutive reports about efficient implementation of BRLWE on the hardware platforms have been released very recently [21], [33], which represent the major advance in the field.

The BRLWE based PQC has high potential to be used in the Internet-of-Things (IoT) servers and edge computing devices, and hence different processing styles of the BRLWE-based cryptoprocessors need to be thoroughly explored. In this section, we propose a novel implementation strategy, based on serial processing, aimed at efficient hardware implementation of BRLWE. Our strategy includes the following steps:

- A new algorithm for the polynomial multiplication over hybrid fields – the main operation of the BRLWE scheme – has been proposed through rigorous mathematical derivation. This algorithm is based on the concept of serial processing.
- A high-performance & low-complexity BRLWE architecture for the decryption operation has been developed.

- A series of thorough complexity analyses and comparisons have been carried out to demonstrate the efficiency of the proposed design over designs reported earlier in the literature.

B. Information of the BRLWE Scheme

Binary Ring-Learning-with-Errors (BRLWE) Scheme. The BRLWE scheme uses binary errors to achieve low-complexity implementation and this scheme overall consists of three main steps, namely the **key generation**, **encryption**, and **decryption**, as depicted in Fig. 4. For simplicity of discussion, we have the following definitions of notations: (i) both Alice and Bob know the public parameter a (polynomial with integer coefficients); (ii) r_1 and r_2 are two binary polynomials, where r_2 is the secret key; (iii) message, denoted by m, is a n-bit binary polynomial to generate the ciphertext (after encryption); (iv) $f(x) = x^n + 1$ is the ring polynomial; and (v) n denotes the BRLWE security level (also the size) and q is used in the modulo operation determined by the ring. The details of each step are as follows:

- **Key generation**. The main operation of the key generation is $p = r_1 - a \cdot r_2$. After this operation, p will be sent to Bob as the public key. In this step, the secret and public keys have n and $n\log_2 q$ bits, respectively.
- **Encryption**. As shown in Fig. 4, message m (represented by $m = m_0 + m_1 x + \cdots + m_{n-1}x^{n-1}$) is encoded into $\widetilde{m}$ according to (1). Then, three binary errors e_1, e_2, and e_3 are employed to deliver the ciphertext c_1 and c_2 (the ciphertext has $2n\log_2 q$ bits) [21], [32].

$$(m_0, \ldots, m_{n-1}) \Rightarrow \sum_{i=0}^{n-1} m_i(\frac{q}{2})x^i. \tag{1}$$

- **Decryption**. Alice uses the secret key r_2 to decrypt the received ciphertext $\widetilde{m}$ into original message m. Note that a threshold decoder function is used here to produce the final results. When the coefficient of the final polynomial is within the range of $(q/4, 3q/4)$, the output is '1' (otherwise the output is '0') [21], [32].

Inverted BRLWE. The very recent report of [33] has proposed an inverted BRLWE scheme, i.e., the coefficients of the polynomials can be expressed in the two's complement form if these coefficients are in the inverted range of $(-\lfloor\frac{q}{2}\rfloor, \lfloor\frac{q}{2}\rfloor - 1)$ [33] as

$$\begin{aligned} &S \geq q/2 : S - q = S + (2^k - q) = S + (2^k - 2^k) = S, \\ &S < -q/2 : S + q = S + 2^k = 2^k - \overline{S} = -\overline{S} = S, \end{aligned} \tag{2}$$

where in this case $q = 2^k$ and S is the number denoted by the two's complement form (i.e., $\overline{S} = 2^k - S$) [33]. Hence, all the modular additions/subtractions can be computed without any extra reduction cost. Of course, the original function of (1) needs to be adjusted as $\sum_{i=0}^{n-1} m_i(-\frac{q}{2})x^i$, while the other parameter settings follow the same as those in the original scheme. Note that the proposed structure is also based on this scheme.

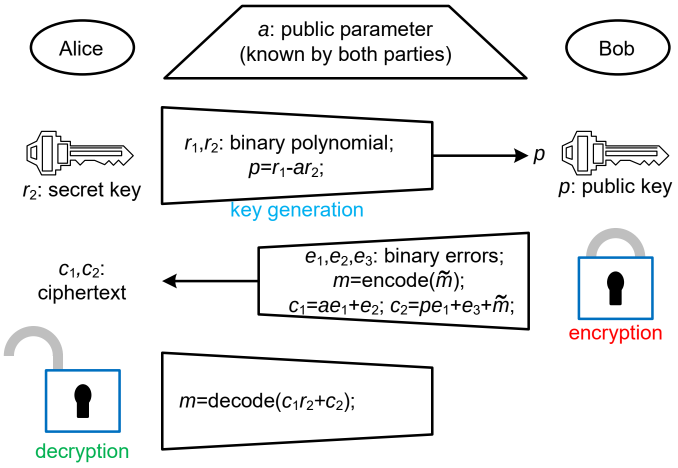

Alice
a: public parameter (known by both parties)
Bob
r1,r2: binary polynomial; p=r1-ar2;
p
r2: secret key
p: public key
key generation
c1,c2: ciphertext
e1,e2,e3: binary errors; m=encode(m̃); c1=ae1+e2; c2=pe1+e3+m̃;
encryption
m=decode(c1r2+c2);
decryption

Figure 4: Three steps of the BRLWE scheme.

C. Proposed BRLWE Algorithm

As seen from Fig. 4, the main operations of each step can be summarized as

$$\begin{aligned} \textbf{key generation} &: \bigotimes \rightarrow \bigoplus \rightarrow p \\ \textbf{encryption} &: \bigotimes \rightarrow \bigoplus \rightarrow c_1 \\ & \bigotimes \rightarrow \bigoplus \rightarrow \bigoplus \rightarrow c_2 \\ \textbf{decryption} &: \bigotimes \rightarrow \bigoplus \rightarrow m, \end{aligned} \tag{3}$$

where $\bigotimes$ and $\bigoplus$ denote the corresponding polynomial multiplication and addition, respectively (the subtraction in the **key generation** step is executed by the addition under the two's complement representation).

From (3), it is clear the main operation involved within this scheme is the polynomial multiplication over the ring $\mathbb{Z}_q/f(x)$ ($f(x) = x^n + 1$), where one polynomial is represented in the integer field and the other is expressed in the binary field. The polynomial addition and subtraction in the ring are very straightforward but the multiplication involves the modulo operation over the ring polynomial $f(x) = x^n + 1$ [32] and hence is more complicated.

Let us firstly consider the major operation for the three steps of the BRLWE scheme, i.e., the polynomial multiplication followed by the addition (there is the slight adjustment on the **encryption** step due to one extra addition). For simplicity of presentation, one can actually use the operation of the **decryption** step as a basic module, which can be easily extended to other steps.

Definition 1. Define that the main operation in the decryption step is $H = BD \bmod f(x) + G$, where $H = \sum_{i=0}^{n-1} h_i x^i$, $B = \sum_{i=0}^{n-1} b_i x^i$, $D = \sum_{i=0}^{n-1} d_i x^i$, and $G = \sum_{i=0}^{n-1} g_i x^i$, for h_i, d_i, and g_i are integers and $b_i \in \{0, 1\}$. Note that (in this case) B, D, and G correspond to r_2, c_1, and c_2 of the decryption step of Fig. 4, respectively.

Define again

$$T = BD \bmod f(x), \tag{4}$$

where $f(x) = x^n + 1$ and $T = \sum_{i=0}^{n-1} t_i x^i$ (t_i is integer). We can then have

$$T = \sum_{i=0}^{n-1} b_i x^i D \bmod f(x) = \sum_{i=0}^{n-1} b_i D^{(i)}, \tag{5}$$

where $D^{(i)} = x^i D \bmod f(x)$ and $D^{(0)} = D$. Define again $D^{(i)} = \sum_{j=0}^{n-1} d_j^{(i)} x^j$, we can have

$$\begin{aligned} D^{(i+1)} =& x^{i+1} D \bmod f(x) = D^{(i)} x \\ =& (d_0^{(i)} + d_1^{(i)} x + d_2^{(i)} x^2 + \cdots + d_{n-1}^{(i)} x^{n-1}) x \\ =& d_0^{(i)} x + d_1^{(i)} x^2 + d_2^{(i)} x^3 + \cdots + d_{n-1}^{(i)} x^n. \end{aligned} \tag{6}$$

Considering $f(x) = x^n + 1$ ($x^n + 1 \equiv 0 \Rightarrow x^n \equiv -1$), which can be substituted into (8) to have (D is an integer polynomial)

$$D^{(i+1)} = -d_{n-1}^{(i)} + d_0^{(i)} x + d_1^{(i)} x^2 + \cdots + d_{n-2}^{(i)} x^{n-1}. \tag{7}$$

Meanwhile, we also have $D^{(i+1)} = d_0^{(i+1)} + d_1^{(i+1)} x + d_2^{(i+1)} x^2 + \cdots + d_{n-1}^{(i+1)} x^{n-1}$, which can be used to compare with (7) to have

$$d_0^{(i+1)} = -d_{n-1}^{(i)}, \; d_1^{(i+1)} = d_0^{(i)}, \; \ldots \;, \; d_{n-1}^{(i+1)} = d_{n-2}^{(i)}, \tag{8}$$

which can be used to transform (5) into the form of

$$\begin{aligned} t_0 =& (d_0, -d_{n-1}, -d_{n-2}, \ldots, -d_1) \odot (b_0, b_1, \ldots, b_{n-1}), \\ t_1 =& (d_1, d_0, -d_{n-1}, \ldots, -d_2) \odot (b_0, b_1, \ldots, b_{n-1}), \\ & \cdots \; \cdots \; \cdots \\ t_{n-1} =& (d_{n-1}, d_{n-2}, d_{n-3}, \ldots, d_0) \odot (b_0, b_1, \ldots, b_{n-1}), \end{aligned} \tag{9}$$

where $(\cdot) \odot (\cdot)$ denotes the point-to-point multiplication. For simplicity of discussion, we can define again $D'_0 = (d_0, -d_{n-1}, -d_{n-2}, \ldots, -d_1)$, $D'_1 = (d_1, d_0, -d_{n-1}, \ldots, -d_2)$, ..., $D'_{n-1} = (d_{n-1}, d_{n-2}, d_{n-3}, \ldots, d_0)$, and $B' = (b_0, b_1, \ldots, b_{n-1})$. One can note that B'_0 can be circularly shifted (with every shifted value, i.e., the very right value, added with a negative sign) to obtain other B'_l ($1 \leq l \leq n-1$).

For serial-processing, i.e., t_i ($0 \leq i \leq n-1$) is produced sequentially, one can rotate the coefficients of operand D'_0 to be processed with another fixed operand B' for point-wise multiplication. The whole computation process, including the final addition with G, is thus proposed in Algorithm 1.

Algorithm 1 Proposed serial-processing based algorithm for the BRLWE scheme (**decryption** step)

```
Inputs: $B$, $D$, and $G$ are elements over hybrid fields.
Outputs: $m = \text{decode}(H = BD \bmod f(x) + G)$.
1. Initialization step
1.1. rearrange the order of the coefficients of $D$ to get $D'_0$.
2. Main computation step
2.1. for $i = 0$ to $n-1$.
2.2. get $t_i = D'_i \odot B$ (for the following cycles, one can obtain $D'_{i+1}$ from $D'_i$).
2.3. obtain $h_i = t_i + g_i$.
2.4. perform the final decoding operation on $h_i$ according to the decryption step of Fig. 4.
2.5. end for.
3. Final step
3.1. deliver final $n$-bit of output $m$.
```

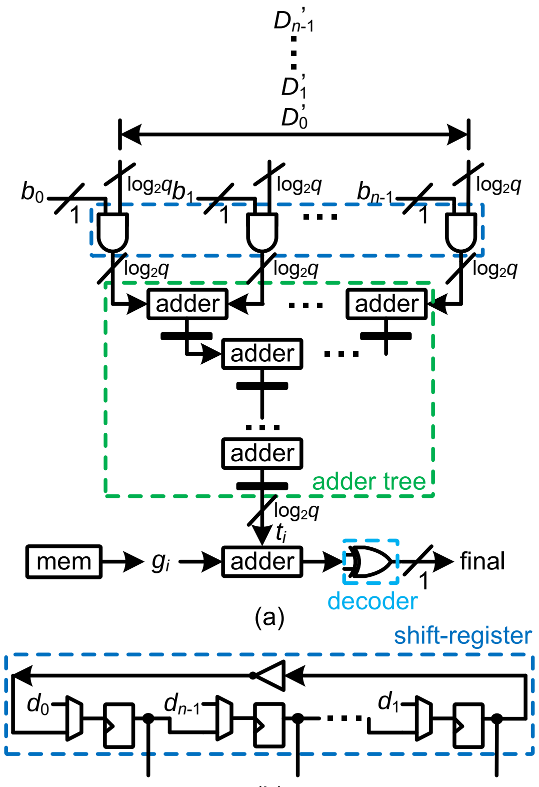


Figure 5: Proposed serial processing structure for the decryption phase of the BRLWE scheme, where the black box denotes the registers. (a) The proposed structure (mem denotes the memory). (b) The possible structure of the input module (a shift-register).

where one output bit is delivered at every cycle and the whole outputs are available after n cycles.

D. The Proposed Structure

The proposed Algorithm 1 can be mapped into the desired serial processing structure as shown in Fig. 5, where the structure consists of three main modules, namely the input module, the main computation module (AND gates and an adder tree), and the final adder and decode module. The input module, possibly can be realized by a shift-register, consists of n pairs of registers and MUXes (both $\log_2 q$-bit), where the MUXes function to initiate the values stored in the registers according to D'_0. Then, in the following cycles, each of the bit-register ($\log_2 q$-bit) circularly shifts (with the help from an inverter such that the far right one is inverted to be the first

Table II: Comparison of Main Complexities for Various BRLWE Schemes

design	AND	XOR	adder ($\log_2 q$-bit)	register (1-bit)	MUX ($\log_2 q$-bit)	MUX (1-bit)	critical-path	latency
[21][1]	$n\log_2 q$	n	n	$\simeq 3\log_2 qn$	$\simeq 2n$	-	$\geq T_{Mem} + T_{MUX}$	$\simeq n+3$
[33][2]	$n\log_2 q$	n	n	$n\log_2 q$	$n+1$	–	$\geq T_A + T_{ADD} + T_{MUX}$	$n+1$
Fig. 5	$n\log_2 q$	1	n	$\log_2 q(2n-1)$	n	–	$T_X + T_{ADD}$	$n + \log_2 n + 1$

The complexities of all the structures are estimated at the **decryption** step. [2]: T_A is the delay time of an AND gate.
[1]: T_{Mem}, denotes the memory access time according to the structure of [21].
The actual implementation critical-path (of all the structures) is also affected by the specific hardware platform deployed, which differs from each other.

left one) the corresponding values to the neighboring one to form the following D'_i $(1 \leq i \leq n-1)$, as shown in Fig. 5(b). Meanwhile, the output of each register is connected with the AND gate cells to be ANDed with the related value of B', where each AND cell has $\log_2 q$ AND gates in parallel connected with each bit of corresponding b_i $(0 \leq i \leq n-1)$. The outputs of n AND cells are then added together through an adder tree to deliver the t_i $(0 \leq i \leq n-1)$, which is then added with the corresponding g_i delivered from the memory. Note that the registers are inserted into the adder tree to realize the pipelining function. The produced output h_i is then sent to the final decoder function (an XOR gate connected the most two significant two bits of h_i according to the decryption function described in Section IV-B and Fig. 4) to produce the final output serially. The black box in the structure of Fig. 5(a) denotes the registers, which sets the critical-path as $T_X + T_{ADD}$ (T_X and T_{ADD} are the delay time of an XOR gate and an adder of $\log_2 q$-bit, respectively).

E. Complexity and Comparison

Complexity. To list the theoretical complexity of the BRLWE scheme, we have used: (i) security-level (or size) of n; (ii) bit-width of the integer coefficients (polynomials) as $\overline{q} = \log_2 q$. The area-time complexities of Fig. 5 are listed in Table II along with those of the existing designs of [21], [33].

As shown in Table II, Fig. 5 has smaller critical-path than the existing ones of [21], [33] yet with slightly larger number of registers. Overall, to better estimate the efficiency of proposed design along with the existing ones of [21], [33], we have implemented them on the platform (as follows).

Comparison. We have then coded the proposed designs with VHDL and synthesized them using Intel Quartus Prime 17.0 on the Stratix-V 5SGXMA9N1F45C2 device. Following [21], [33], we have chosen $n = 256$, $n = 512$, $q = 256$, and $\log_2 q = 8$ (according the recent analysis of [32], [35], $n = 512$ and $n = 256$ can provide equivalent 190/140 and 84/73 bits of class and quantum securities, respectively).

For a fair comparison, we have also re-code the existing one of [33] and synthesized it on the same FPGA platform (the design of [33] has shown its efficiency over [21], here we just compare with [33]). We have used the same type of adder as that in the proposed designs, i.e., 8-bit ripple carry adder, as well as other required resources. All the input values are delivered to the BRLWE cryptoprocessor (both the existing and proposed designs) through the input modules while the output follows the specific design style of the structure. The

Table III: Comparison of the Complexities for Both the Proposed and Existing BRLWE Structures (**Decryption Step**)

design	ALMs	Fmax*	latency	delay[!]	ADP[1]
$n = 256$					
[33]	6,691	148.28	257	1,733	11,596
Fig. 5	6,159	253.68	265	1,045	6,436
$n = 512$					
[33]	13,343	154.18	513	3,327	44,392
Fig. 5	12,418	194.56	522	2,683	33,317

*: MHz (unit). [!]: ns (unit).
[1]: ADP=#ALM$\times$delay ($\times 10^3$).

area-time complexities, namely the number of adaptive logic module (ALM), maximum frequency (MHz), latency cycles, delay (critical-path$\times$latency cycles), and area-delay product (ADP), are listed in Table III.

The proposed design outperforms the existing one, e.g., the proposed Fig. 5 has at least 44.5% and 24.9% less ADP than the one of [33] for $n = 256$ and $n = 512$, respectively. Benefited from the proper arrangement of registers in the adder tree for regular pipelining (very efficient for mapping in the FPGA platform), the proposed design also achieves less area occupation (while the existing designs do not enjoy this advantage). Besides that, the proposed design also has smaller critical-path than the existing one (faster maximum frequency). These two unique features eventually lead to the higher efficiency of the proposed design.

Though the primary focus of this section is to obtain efficient implementation of the BRLWE scheme, the reports of resisting attacks and equipping other countermeasures [36] are still applicable to our work, which can also be our future research direction.

V. Conclusions

This paper gives a high-level introduction to the recent advances in hardware implementation of PQC, including challenges, new implementation approaches, and optimization techniques aimed at constrained environments. Specifically, we have: (i) discussed the challenges and rewards of the PQC revolution, with the special focus on hardware and software/hardware implementations; (ii) presented highlights of the comprehensive HLS-based methodology for the design-space exploration, capable of matching the requirements of a given application with the features of a specific hardware architecture for PQC; (iii) introduced a new underexplored

PQC scheme (binary Ring-Learning-with-Errors), as well as its novel hardware implementation, targeting lightweight applications, such as IoT. The authors hope that the provided introduction would encourage readers to further explore this exciting area, and contribute to its growth through research, education, standardization, and entrepreneurship.

References

[1] D. J. Bernstein, "Introduction to post-quantum cryptography," in *Post-quantum cryptography*. Springer, 2009, pp. 1–14.

[2] P. W. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proc. 35th Annual Symposium on Foundations of Computer Science*. IEEE, 1994, pp. 124–134.

[3] "Post-quantum cryptography round 2 submissions," https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions.

[4] "Post-Quantum Cryptography Standardization," https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, 2020.

[5] M. Hutter, J. Schilling, P. Schwabe, and W. Wieser, "NaCl's Crypto_box in Hardware," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, vol. 9293. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 81–101.

[6] Cryptographic Engineering Research Group (CERG) at George Mason University, "Hardware Benchmarking of CAESAR Candidates," https://cryptography.gmu.edu/athena/index.php?id=CAESAR, 2019.

[7] E. Homsirikamol and K. Gaj, "Toward a new HLS-based methodology for FPGA benchmarking of candidates in cryptographic competitions: The CAESAR contest case study," in *2017 International Conference on Field Programmable Technology, FPT 2017*. Melbourne, Australia: IEEE, Dec. 2017, pp. 120–127.

[8] K. Basu, D. Soni, M. Nabeel, and R. Karri, "NIST Post-Quantum Cryptography- A Hardware Evaluation Study," Cryptology ePrint Archive 2019/047, May 2019.

[9] F. Farahmand, V. B. Dang, D. T. Nguyen, and K. Gaj, "Evaluating the Potential for Hardware Acceleration of Four NTRU-Based Key Encapsulation Mechanisms Using Software/Hardware Codesign," in *10th International Conference on Post-Quantum Cryptography, PQCrypto 2019*, ser. LNCS. Chongqing, China: Springer, May 2019.

[10] F. Farahmand, D. T. Nguyen, V. B. Dang, A. Ferozpuri, and K. Gaj, "Software/Hardware Codesign of the Post Quantum Cryptography Algorithm NTRUEncrypt Using High-Level Synthesis and Register-Transfer Level Design Methodologies," in *29th International Conference on Field Programmable Logic and Applications, FPL 2019*. Barcelona, Spain: IEEE, Sep. 2019, pp. 225–231.

[11] E. Ozcan and A. Aysu, "High-level-synthesis of number-theoretic transform: A case study for future cryptosystems," *IEEE Embedded Systems Letters*, vol. PP, pp. 1–1, 12 2019.

[12] B. Hettwer, S. Gehrer, and T. Güneysu, "Applications of machine learning techniques in side-channel attacks: A survey," *Journal of Cryptographic Engineering*, Apr. 2019.

[13] F. Farahmand, M. U. Sharif, K. Briggs, and K. Gaj, "A High-Speed Constant-Time Hardware Implementation of NTRUEncrypt SVES," in *2018 International Conference on Field-Programmable Technology, FPT 2018*. Naha, Okinawa, Japan: IEEE, Dec. 2018, pp. 190–197.

[14] W. Wang, S. Tian, B. Jungk, N. Bindel, P. Longa, and J. Szefer, "Parameterized Hardware Accelerators for Lattice-Based Cryptography," Cryptology ePrint Archive 2020/054, Jan. 2020.

[15] W. Wang, B. Jungk, J. Wälde, S. Deng, N. Gupta, J. Szefer, and R. Niederhagen, "XMSS and Embedded Systems - XMSS Hardware Accelerators for RISC-V," in *Selected Areas in Cryptography – SAC 2019*, ser. LNCS, vol. 11959. Waterloo, Ontario, Canada: Springer, 2019, pp. 523–550.

[16] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and Benchmarking Three Lattice-based Post-Quantum Cryptography Algorithms Using Software/Hardware Codesign," in *2019 International Conference on Field Programmable Technology, FPT 2019*. Tianjin, China: IEEE, Dec. 9-13, 2019, pp. 206–214.

[17] F. Farahmand, A. Ferozpuri, W. Diehl, and K. Gaj, "Minerva: Automated Hardware Optimization Tool," in *2017 International Conference on Reconfigurable Computing and FPGAs - ReConFig 2017, Cancun, Mexico, Dec. 4-6, 2017*, Dec 2017, pp. 414–421.

[18] A. Ferozpuri, F. Farahmand, V. Dang, M. U. Sharif, J.-P. Kaps, , and K. Gaj. (2018, Mar.) Hardware API for Post-Quantum Public Key Cryptosystems. [Online]. Available: https://cryptography.gmu.edu/athena/index.php?id=PQC

[19] O. Reparaz, S. Sinha Roy, F. Vercauteren, I. Verbauwhede, F. Vercauteren, and I. Verbauwhede, "A Masked Ring-LWE Implementation," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, vol. 9293. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 683–702.

[20] O. Reparaz, R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Additively Homomorphic Ring-LWE Masking," in *Post-Quantum Cryptography*, T. Takagi, Ed. Cham: Springer International Publishing, 2016, pp. 233–244.

[21] A. Aysu, M. Orshansky, and M. Tiwari, "Binary Ring-LWE hardware with power side-channel countermeasures," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1253–1258.

[22] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu, "Practical CCA2-Secure and Masked Ring-LWE Implementation," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. Volume 2018, pp. Issue 1–, Feb. 2018.

[23] D. Liu, C. Zhang, H. Lin, Y. Chen, and M. Zhang, "A Resource-Efficient and Side-Channel Secure Hardware Implementation of Ring-LWE Cryptographic Processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1474–1483, Apr. 2019.

[24] D. Soni, K. Basu, M. Nabeel, and R. Karri, "A Hardware Evaluation Study of NIST Post-Quantum Cryptographic Signature schemes," in *Second PQC Standardization Conference*. NIST, 2019.

[25] D. Soni, M. Nabeel, K. Basu, and R. Karri, "Power, Area, Speed, and Security (PASS) Trade-offs of NIST PQC Signature Candidates Using a C to ASIC Design Flow," in *IEEE Conference on Computer Design*, 2019.

[26] D. Soni, K. Basu, M. Nabeel, M. Manzano, N. Aaraj, and R. Karri, *Hardware Architectures for Post-Quantum Cryptography - Power-Performance-Area-Security Trade-offs of PQC-based Signature Generation and Verification*. Springer Nature, 2020.

[27] D. Soni, https://youtu.be/bZVjiHeQjoU, accessed: 2020-02-07.

[28] D. Micciancio and O. Regev, "Lattice-based cryptography," in *Post-quantum cryptography*. Springer, 2009, pp. 147–191.

[29] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede, "High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 157–166, 2014.

[30] D. Liu, C. Zhang, H. Lin, Y. Chen, and M. Zhang, "A resource-efficient and side-channel secure hardware implementation of ring-LWE cryptographic processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 4, pp. 1474–1483, 2018.

[31] J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and K. Beeden, "Lattice-based encryption over standard lattices in hardware," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2016, pp. 1–6.

[32] J. Buchmann, F. Göpfert, T. Güneysu, T. Oder, and T. Pöppelmann, "High-performance and lightweight lattice-based public-key encryption," in *2nd ACM International Workshop on IoT Privacy, Trust, and Security*, 2016, pp. 2–9.

[33] S. Ebrahimi, S. Bayat-Sarmadi, and H. Mosanaei-Boorani, "Post-Quantum Cryptoprocessors Optimized for Edge and Resource-Constrained Devices in IoT," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5500–5507, 2019.

[34] J. Buchmann, F. Göpfert, R. Player, and T. Wunderer, "On the hardness of LWE with binary error: Revisiting the hybrid lattice-reduction and meet-in-the-middle attack," in *International Conference on Cryptology in Africa*. Springer, 2016, pp. 24–43.

[35] M. Liu and P. Q. Nguyen, "Solving bdd by enumeration: An update," in *Cryptographers' Track at the RSA Conference*. Springer, 2013, pp. 293–309.

[36] T. Schneider, A. Moradi, and T. Güneysu, "Parti–towards combined hardware countermeasures against side-channel and fault-injection attacks," in *Annual International Cryptology Conference*. Springer, 2016, pp. 302–332.