Post-quantum Lattice-based Cryptography Implementations: A Survey*

HAMID NEJATOLLAHI, University of California Irvine
NIKIL DUTT, University of California Irvine
SANDIP RAY, University of Florida
FRANCESCO REGAZZONI, ALaRi
INDRANIL BANERJEE, Qualcomm Technologies Inc.
ROSARIO CAMMAROTA, Qualcomm Technologies Inc.

The advent of Quantum computing threatens to break many classical cryptographic schemes, leading to innovations in public key cryptography that focus on post-quantum cryptography primitives and protocols resistant to quantum computing threats. Lattice-based cryptography is a promising post-quantum cryptography family, both in terms of foundational properties as well as its application to both traditional and emerging security problems such as encryption, digital signature, key exchange, homomorphic encryption, etc. While such techniques provide guarantees, in theory, their realization on contemporary computing platforms requires careful design choices and trade-offs to manage both the diversity of computing platforms (e.g., high-performance to resource constrained), as well as the agility for deployment in the face of emerging and changing standards. In this work, we survey trends in lattice-based cryptographic schemes, some recent fundamental proposals for the use of lattices in computer security, challenges for their implementation in software and hardware, and emerging needs for their adoption. The survey means to be informative about the math, to allow the reader to focus on the mechanics of the computation, ultimately needed for mapping schemes on existing hardware, or synthesizing, part or all of a scheme, on special purpose hardware.

Additional Key Words and Phrases: Post-quantum cryptography; Public-key encryption; Lattice based cryptography; Ideal lattices; Ring-LWE

ACM Reference Format:
Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. 2018. Post-quantum Lattice-based Cryptography Implementations: A Survey. *ACM Comput. Surv.* 1, 1, Article 1 (January 2018), 38 pages. https://doi.org/10.1145/3292548

1 INTRODUCTION

Advances in computing efficiency steadily erode computer security at its foundation that enable prospective attackers to use ever more powerful computing systems and the best cryptanalysis

*An extended version of this paper can be found in [1]

This work was supported in part with a gift from Qualcomm Technology Inc.
Authors' addresses: Hamid Nejatollahi, University of California Irvine, Irvine, California, 92697-3435, hnejatol@uci.edu; Nikil Dutt, University of California Irvine, Irvine, California, 92697-3435, dutt@ics.uci.edu; Sandip Ray, University of Florida, sandip@ece.ufl.edu; Francesco Regazzoni, ALaRi, regazzoni@alari.ch; Indranil Banerjee, Qualcomm Technologies Inc. San Diego, CA, 92121-1714, ibanerje@qti.qualcomm.com; Rosario Cammarota, Qualcomm Technologies Inc. San Diego, CA, 92121-1714, ro.c@qti.qualcomm.com.

0360-0300/2018/1-ART1 $15.00
https://doi.org/10.1145/3292548

algorithms to increase the attack speed. One aspect of this arises from Moore's Law, that enables traditional computing systems to become more powerful than ever before, allowing increasingly larger brute-force attacks. Another aspect of concern is the rise of alternative computing paradigms, such as Quantum computing and its algorithms [2, 3] - that seem to be closer than ever reality,[1] [2], which in turn promises to further weaken the strength of current, standardized cryptography and its applications. Against quantum computers traditional public key cryptography is ineffective for any key length algorithm. The Shor's algorithm for quantum computers is designed to solve prime factorization of large primes and the discrete logarithm problem in polynomial time.

The emergence of new computing platforms, such as cloud Computing, software defined networks and Internet of Everything, demands the adoption of an increasing number of security standards, which in turn requires the implementation of a diverse set of cryptographic primitives, but this is only part of the story. At the computing platform level, we are seeing a diversity of computing capability, ranging from high-performance (real-time) virtualized environments, such as cloud computing resources and software defined networks, to highly resource-constrained IoT platforms to realize the vision of Internet of Everything. This poses tremendous challenges in the design and implementation of emerging standards for cryptography in a single embodiment, since the computing platforms exact diverging goals and constraints. On one end of the spectrum, in the cloud computing and software defined network space, applications demand high-performance, and energy efficiency of cryptographic implementations. This calls for the development of programmable hardware capable of running not only individual cryptographic algorithms, but full protocols efficiently, with the resulting challenge of designing for agility, e.g., designing computing engines that achieve the efficiency of Application-Specific Integrated Circuits (ASICs), while retaining some level of programmability. On the other end of the spectrum, in the IoT space, implementations of standardized cryptography to handle increased key sizes become too expensive in terms of cost, speed, and energy, but are necessary, e.g., in the case of long lived systems such as medical implants and image encryption algorithms [4]. In part, this demands the development of new and strong lightweight alternatives to symmetric key cryptography as well [5]. Furthermore, given the variety of applications and their interplay with the cloud, even in this case agility in the implementation becomes a key requirement.

As a result of the trends in technology, the need to strengthen current practices in computer security, including strengthening and adding more variety in the cryptographic primitives in use, has become a widely accepted fact. The examples above, to name a few, form a compelling argument to call for innovation in the computer security space, including and beyond the foundations, i.e., down to the actual implementation and deployment of primitives and protocols to satisfy the emerging business models and their design constraints - latency, compactness, energy efficiency, tamper resistance and, more importantly, agility.

Innovation in public key cryptography focuses on the standardization of the so called post-quantum cryptography primitives and their protocols. Among the post-quantum cryptography families, the family of lattice-based cryptography (LBC) appears to be gaining acceptance. Its applications are proliferating for both traditional security problems (e.g., key exchange and digital signature), as well as emerging security problems (e.g., homomorphic schemes, identity-based encryption and even symmetric encryption). Lattice-based cryptographic primitives and protocols provides a rich set of primitives which can be used to tackle the challenges posed by deployment across diverse computing platforms, e.g., Cloud vs. Internet-of-Things (IoT) ecosystem, as well as for diverse use cases, including the ability to perform computation on encrypted data, providing strong (much better understood than before) foundations for protocols based on asymmetric key cryptography against powerful attackers

[1] http://spectrum.ieee.org/tech-talk/computing/software/rigetti-launches-fullstack-quantum-computing-service-and-quantum-ic-fab
[2] https://newatlas.com/ibm-next-quantum-processors/49590/

(using Quantum computers and algorithms), and to offer protection beyond the span of traditional cryptography. Indeed, lattice-based cryptography promises to enhance security for long-lived systems, e.g., critical infrastructures, as well as for safety-critical devices such as smart medical implants [6].

In this paper, we review the foundations of lattice-based cryptography, some of the more adopted instances of lattices in security, their implementations in software and hardware, and their applications to authentication, key exchange and digital signatures. The purpose of this survey is to focus on the essential ideas and mechanics of the cryptosystems based on lattices and the corresponding implementation aspects. We believe that this survey is not only unique, but also important for guidance in the selection of standards, as well as in the analysis and evaluation of candidate implementations that represent state-of-the-art.

The rest of the paper is organized as follows. In Section 2 we provide the relevant background to make the paper self-contained. We briefly describe different lattice constructions and LBC in Section 2.5. Section 3 discusses various implementations of lattice-based schemes. Section 3.1 discusses related art on improvements for arithmetic computation, i.e., polynomial/matrix multiplication or/and noise sampler. Sections 3.2 and 3.3 present software and hardware implementations of different LBC. Sections 4 concludes with an outlook.

2 BACKGROUND

2.1 Standard lattices

Lattice $\mathcal{L}$ is the infinite set (discrete) of points in n-dimensional Euclidean space with a periodic structure [7]. A basis (B) of the lattice is defined as $\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n \in \mathbb{R}^{d \times n}$ to be n-linearly independent vectors in $\mathbb{R}^d$. B is a $d \times n$ matrix in which i^{th} column is $\mathbf{b}_i$ vector such that $B = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n]$. Integer n and d are rank and dimension of the lattice $\mathcal{L}(B)$. If $n = d$, $\mathcal{L}(B)$ is a full rank (or dimension) lattice in $\mathbb{R}^d$. All integer combinations generated by the basis matrix B (with integer or rational entries) are forming the lattice $\mathcal{L}$.

$$\mathcal{L}(B) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^n\} = \mathcal{L}(\mathbf{b}_1, ..., \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}, 1 \leq i \leq n \right\} \tag{1}$$

2.2 Ideal lattices

An ideal lattice is defined over a ring $R = \mathbb{Z}_q[x]/(f(x))$, where $f(x) = x^n + f_n x^{n-1} + \ldots + f_1 \in Z[x]$ (cyclotomic polynomial) and R contains all the polynomials with modulo q integer coefficients. In the case where $f(x)$ is monic (leading coefficient is 1) irreducible polynomial degree-n, $R = \mathbb{Z}_q[x]/(f(x))$ includes polynomials of degree less that equal $n - 1$. For instance, $R = \mathbb{Z}_q[x]/(x^n + 1)$ is an ideal lattice when n is a power of 2; however, $R = \mathbb{Z}_q[x]/(x^n - 1)$ is not an ideal lattice since $(x^n - 1)$ is a reducible polynomial.

2.3 Lattice problems and their applications to cryptography

The breakthrough work of Ajtai [8] provides confidence for adopting lattice-based schemes in cryptography. Ajtai proves that solving some NP-hard lattice problems, e.g., Shortest Vector Problem, in average-case is as hard as to solve in the worst case assumption. In order to solve SVP problem, one should solve the problem for any input basis B (all instances of SVP problem should be solved). It is conjectured that there is no a probabilistic polynomial time algorithm that can approximate certain computational problems on lattices to within polynomial factors [9]. This is the basis for security of lattice-based cryptography schemes. The fastest algorithm to solve the SVP problem has the time and memory complexity of $2^{O(n)}$ [10–12]. We take the below definitions from [13]:

2.3.1 Shortest Vector Problem (SVP). There are three variants of the SVP [13] that can be reduced to each other. The first is to find the shortest nonzero vector; the second is to find the length of the shortest nonzero vector; and the third determines if the shortest nonzero vector is shorter than a given real number. Given B as the lattice basis, $v \in \mathcal{L}(B)\backslash\{0\}\}$ is defined as the shortest nonzero vector in lattice $\mathcal{L}(B)$ such that $\|v\| = \lambda_1(\mathcal{L}(B))$. Output of the SVP problem is **a**, the shortest nonzero vector in the lattice which is unique. SVP can be defined to an arbitrary norm (we use norm 2 here as the Euclidean norm). In γ-Approximate Shortest Vector Problem (SVP_γ), for $\gamma \geq 1$, the goal is to find the shortest nonzero vector $v \in \mathcal{L}(B)\backslash\{0\}$ where $\|v\| \leq \lambda_1(\mathcal{L}(B))$. The special case of $\gamma = 1$ is equivalent to the exact SVP. The decision variant of the Shortest Vector Problem ($G_{AP}SVP_\gamma$) is defined as determining if $d < \lambda_1(\mathcal{L}(B)) \leq \gamma.d$ where d is a positive real number.

2.3.2 Closest Vector Problem (CVP). Let B and t be the lattice basis and the target point (might not be a member of the lattice), respectively; CVP is defined as finding vector $v \in \mathcal{L}$ where its distance ($\|v - t\|$) to a target point is minimized [13]. In γ-Approximate Closest Vector Problem (CVP_γ), for $\gamma \geq 1$, the goal is to find vector $v \in \mathcal{L}(B)$ such that $\|v - t\| \leq \gamma.dist(t, \mathcal{L}(B))$ where $dist(t, \mathcal{L}(\mathcal{B})) = inf\{\|v - t\| : v \in \mathcal{L}\}$ is the distance of target point t to the lattice $\mathcal{L}$.

2.3.3 Shortest Independent Vectors Problem (SIVP). Given lattice basis B and prime integer q, the shortest independent vector problem (SIVP) is defined as finding n linearly independent lattice vectors $\{v = v_1, ..., v_n : v_i \in \mathcal{L}(B) for 1 \leq i \leq n\}$ that minimizes $\|v\|$=max$_i\|v_i\|$ [13]. Given an approximate factor $\gamma \geq 1$, the $\gamma - approximate$ Shortest Independent Vector Problem ($SIVP_\gamma$) is defined as finding n-linearly independent vectors lattice vectors $\{v = v_1, ..., v_n : v_i \in \mathcal{L}(B)$ such that max$_i\|v_i\| \leq \lambda_n(\mathcal{L}(B)$ where λ_n denotes the n^{th} success minima. For an n-dimensional lattice, λ_i, i^{th} success minima, is the radius of the smallest ball that contains i linearly independent lattice vectors. The decision version of SIVP is called ($G_{AP}SIVP_\gamma$) and is defined as determined if $d < \lambda_n(\mathcal{L}(B)) \leq \gamma.d$ where d is a positive real number.

There is a close relationship between hard lattice problems (e.g., CVP and SVP) and average-case lattice-based problems, Learning With Error (LWE) [7] and Shortest Integer Solution (SIS) [8].

2.3.4 Shortest Integer Solution (SIS). Let a_1, a_2,..., $a_n \in \mathbb{Z}^{m\times n}$ be an arbitrary vector and q is an integer prime number. The SIS problem is defined as finding the vector $x \in \mathbb{Z}^{m\times n}$ such that $x_1.a_1 + x_2.a_2 + ... + x_n.a_n = 0 \pmod q$. Short is usually translates as $z_i \in \{-1, 0, +1\}$. Considering q-ary lattices, let A=(a_1, a_2,..., a_n) be a vector in $\mathbb{Z}^{m\times n}$ and $\Lambda_q^\perp(A) = \{z \in \mathbb{Z}^m : Az = 0 \pmod q\}$; SIS problem is to find the shortest vector problem for the lattice $\Lambda_q^\perp$. Based on the Ajtai's theorem, if polynomial time algorithm A solves the SIS problem, an efficient algorithm B exists that can solve SVP (or SVIP) problem for any lattice of dimension n in polynomial time.

Ring-SIS. Let A=($a_1, a_2,..., a_n$) be a vector with $a_i \in \mathbb{Z}_q[x]/(x^n+1)$ and $\Lambda_q^\perp(A) = \{z \in \mathbb{Z}_q[x]/(x^n+1) : Az = 0 \pmod q\}$; Ring-SIS problem is to find the shortest vector problem for the lattice $\Lambda_q^\perp$ [14].

2.3.5 Learning With Error (LWE). Let a be the polynomial with coefficients sampled uniformly at random in $\mathbb{Z}_q^n$, where n and q are degrees of lattice and modulus (prime integer), respectively. Recovering the (**unique**) random secret s (uniformly at random in $\mathbb{Z}_q^n$) from m ($m \geq n$) samples of the form $(a, a.s + e \bmod q)$ is known as the Learning With Error (LWE) problem where e is the error that is sampled from error distribution χ. Worst case hardness assumption of the LWE problem is proven under the quantum [7] and classical [15] reductions. Besides, If secret s is sampled from the same error distribution as e, hardness assumption of the LWE problem is still valid [16].

Ring-LWE. let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ to be the ring of polynomials where n is power of 2 and q is an integer. Recovering random secret s with uniform coefficients in R from $m \geq 1$ samples

$(a_i, a_i.s + e_i \bmod q)$ is known as ring learning with error problem (Ring-LWE) where $e \in R$ is the error with coefficients sampled from error distribution χ [17].

Learning With Rounding (LWR). Complexity and inefficiency of the decisional (LWE) problem prohibit it to be used for PRGs. The learning with rounding (LWR) problem [18] is the "derandomized" variant of the LWE with improved speedup (by eliminating sampling small errors from a Gaussian-like distribution with deterministic errors) and bandwidth (by rounding Z_q to the sparse subset R_p). LWE hides the lower order bits by adding a small error; however, LWR concealed the lower order bits with rounding. Let a_i be sampled uniformly at random in Z_q^n and $\lfloor \cdot \rceil : Z_q \rightarrow Z_p$ for $p < q$ be the modular "rounding function" where $\lfloor x \rceil_p = \lfloor (p/q) \cdot x \rceil \bmod p$. Similar to LWE, LWR is defined as recovering the (**unique**) secret s (uniformly at random in $\mathbb{Z}_q^n$) from $m \geq 1$ samples $(a_i, \lfloor \langle a_i, s \rangle \rceil_p) \in Z_q^n \times Z_p$. In other words, there is no probabilistic polynomial time algorithm that can distinguish pairs of $(a_i \leftarrow Z_q, \lfloor \langle a_i, s \rangle \rceil_p) \in Z_q^n \times Z_p$ with $(a_i \leftarrow Z_q, \lfloor u \rceil_p)$ where u is uniformly random in Z_q. LWR is assumed to be hard under the hardness assumption of LWE when the number of samples is bounded.

Module-LWE. Let n and d be dimensions of R (degree of ring R_q) and rank of module $M \in R^d$. The hardness of a scheme with $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ and $d = 1$ is based on Ring-LWE and Ring-SIS problems; however $R_q = \mathbb{Z}_q$ and $d > 1$ end up with LWE and SIS problems. Module-LWE/SIS is a generalization of LWE/SIS and Ring-LWE/SIS in which the parameters are $R = \mathbb{Z}_q[x]/(x^n + 1)$ and $d \geq 1$. Security reduction of lattice problems in module M depends on $N = n \times d$ (dimension of the corresponding module lattice) [19]. Suppose A is a $d \times d$ random matrix; security reduction of LWE/SIS problem for $d = 1$ (Ring-SIS/LWE) and ring of dimension n is the same as $d = i$ and a ring of dimension n/i. In the former case, matrix A contains n elements in Z_q; however, in the latter case, A contains $i^2 \times n$ elements in Z_q.

Let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ and $R = \mathbb{Z}[x]/(x^n + 1)$ be the ring of polynomials where n is a power of 2 and $q \in Z$. Vector a is uniformly sampled in R_q^d. Recovering the random secret s with coefficients sampled from the error distribution χ^d in R from $m \geq 1$ samples $(a_i, a_i.s + e_i \bmod q) \leftarrow R_q^d \times R_q$ is known as module learning with error (Module-LWE) problem where $e_i \in R$ is the error with coefficients sampled from error distribution χ [19].

Module-LWE/Module-LWR is a middle ground problem for LWE/LWR and their ring variant RLWE/RLWR which reduces computational pressure and bandwidth of the standard lattices and improve the security of the ideal lattices. With the same arithmetic foundation, *Module-LWE/Module-LWR* provides a trade-off between the security and cost (computation and bandwidth).

2.4 Arithmetic and Components of Lattices

In this section, we provide an evaluation of the components in a LBC that guides the actual implementation. Two components are critical: (a) the polynomial multiplication for ideal lattices, and matrix multiplication for standard lattice are the main speedup bottlenecks; (b) the discrete Gaussian sampling to sample the noise in order to hide the secret information. There are various algorithms for the sampler and multiplier in the literature, by providing the designer with a specific goal [20]. We briefly review different algorithms and outline their practical implementations in Section 3.1

Two main classes of lattice-based algorithms used in cryptography are NTRU and Learning with Error (LWE). The security of NTRU is based on hardness not-provably reducible to solving the Closest Vector Problem (CVP) in a lattice, whereas the security of LWE relies on provably reducible solving the Shortest-Vector Problem (SVP) in a lattice. Consequently, NTRU suffers from security guarantees, but in practice provides more flexibility and efficiency in the implementation. On the contrary, LWE

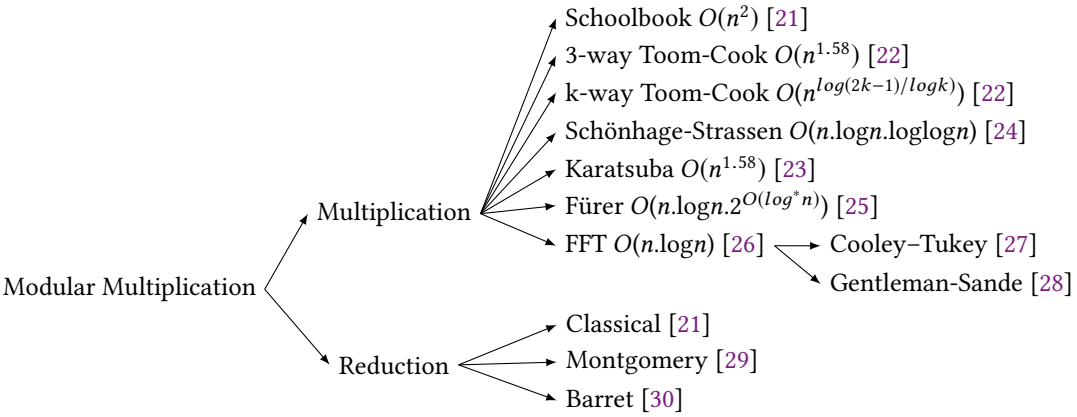

Fig. 1. Common modular Multiplication algorithms.

problems are resistant to quantum attacks, while their relatively inefficient nature led researchers to devise more efficient formulations, e.g., over rings - Ring Learning with Errors (Ring-LWE).

Implementations are broadly classified in pure software, pure hardware, and hardware/software co-design cryptographic engines [20]. Implementation of the modulo arithmetic (multiplication and addition of big numbers) is a bottleneck in LBC. For Standard LWE schemes, matrix multiplication algorithms are adopted, whereas number theoretic transform (NTT) is a better choice for polynomial multiplication in Ring-LWE. A summary of modular arithmetic is presented in Figure 1.

The other bottleneck in LBC is the extraction of the random/noise term, which usually is implemented with a discrete noise sampler, from a discrete Gaussian distribution and can be done with rejection, inversion, Ziggurat, or Knuth-Yao sampling.

Implementation of standard LWE-based schemes exhibits large memory footprint due to the large key size - hundreds of kilobyte for the public key - which renders a straightforward implementation of standard LWE-based schemes impractical. The adoption of specific ring structures, e.g., Ring-LWE, offers key size reduction by a factor of n compared to standard LWE [17], making Ring-LWE an excellent candidate for resource-constrained devices, such as Wi-Fi capable smart devices, including medical implants. Another avenue to address resource-constrained devices is that memory footprint can be traded off with security assurance, which improves both efficiency and memory consumption.

Practical software implementations of standard lattices, encryption scheme [31] and key exchange [32], have been published. For hardware implementations, FPGAs provide flexibility and customization but not agility. Hardware implementations of lattice-based schemes (e.g., BLISS-I [33] with higher security level) are about an order of magnitude faster than the hardware implementation of RSA-2048 [34]. Another candidate platform to implement LBC is application specific integrated circuits (ASICs) of which there appear to be no such implementations in the literature at this time. However, the main advantages and challenges for ASIC design of LBC are presented in [35].

2.4.1 The multiplier component. Matrix multiplication is used for the standard lattices, while polynomial multiplication is employed for ideal lattices. Arithmetic operations for a Ring-LWE based scheme are performed over a ring of polynomials. The most time and memory consuming part is the polynomial multiplication. The easiest way to multiply two polynomials is to use the Schoolbook algorithm with the time complexity of $O(n^2)$ [21]. Let n, power of two, and p be degree of the lattice and a prime number ($p = 1 \bmod 2n$), respectively. Z_p denotes the ring of integers modulo p and x^{n+1}

is an irreducible degree n polynomial. The quotient ring R_p, contains all polynomials with the degree less than n in Z_p, that defines as $R_p = Z_p/[x^{n+1}]$ in which coefficients of polynomials are in [0,p).

The number theoretic transform (NTT) is a generalization of Fast Fourier Transform (FFT), which is carried out in a finite field instead of complex numbers. The latter could achieve time complexity of $O(n\log n)$. In other words, exp(-2πj/N) with n^{th} primitive root of unity ω_n which is defined as the smallest element in the ring that $\omega_n^n = 1 \bmod p$ and ${\omega_n}^i \neq 1 \bmod p$ for i ≠n. The main idea behind this is to use the point value representation instead of the coefficient representation by applying NTT in $O(n\log n)$; thereafter performing point-wise multiplication in $O(n)$ and finally converting the result to coefficient representation by applying Inverse NTT (INTT) in $O(n\log n)$.

$$a(x) \times b(x) = NTT^{-1}(NTT(a) \odot NTT(b)) \tag{2}$$

Where $\odot$ is the point-wise multiplication of the coefficients. If NTT is applied to $a(x)$ with $(a_0, ..., a_{n-1})$ as coefficients, we would have $(\hat{a_0}, ..., \hat{a_{n-1}}) = NTT(a_0, ..., a_{n-1})$ where

$$\hat{a_i} = \sum_{j=0}^{n-1} a_i \omega^{ij} mod(p), i = 0, 1, ..., n-1 \tag{3}$$

To retrieve the answer from the point value representation using NTT^{-1}, it is sufficient to apply NTT function with $-\omega$ and divide all the coefficients by n:

$$a_i = \sum_{j=0}^{n-1} \hat{a}_i \omega^{-ij} mod(p), i = 0, 1, ..., n-1 \tag{4}$$

In order to compute NTT(a), we pad the vector of coefficients with n zeros that leads to doubling the input size. Negative wrapped convolution technique [36], avoids doubling the input size.

To improve efficiency of polynomial multiplications with NTT, combining multiplications of powers of ω with powers of ψ and ψ^{-1} ($\psi^2 = \omega$) is beneficial which requires storage memory for precomputed powers of ω and ψ^{-1} in bit-reversed order [37–39]. NTT can be used only for the $p = 1 \pmod{2n}$ case where p is a prime integer. Suppose $a' = (a_0, \psi a_1, ..., \psi^{n-1} a_{n-1})$, $b' = (b_0, \psi b_1, ..., \psi^{n-1} b_{n-1})$, $c' = (c_0, \psi c_1, ..., \psi^{n-1} c_{n-1})$ to be coefficient vectors of the a, b, c that are multiplied component-wise by $(1, \psi^1, ..., \psi^{n-1})$. Based on the negative wrapped convolution theorem, modulo $(x^n + 1)$ is eliminated and the degree of NTT and NTT^{-1} is reduced from $2n$ to n.

$$c' = NTT^{-1}(NTT(a') \odot NTT(b')) \tag{5}$$

$$c = (\psi^0 {c'}_0, \psi^{-1} {c'}_1, ..., \psi^{-n+1} {c'}_{n-1}) \tag{6}$$

Two common algorithms to compute NTT are Cooley-Tukey butterfly (CT) [27] and Gentleman-Sande butterfly (GS) [28]. CT, decimation-in-time, outputs the result in the bit-reverse order by getting the input in the correct order. GS, decimation-in-frequency, receives the input in the reverse order and produces the output in the correct order [39]. Employing GS to compute both NTT and NTT^{-1} involves in bit-reverse calculation [40]; however, bit-reverse step can be avoided by using CT for NTT and GS for NTT^{-1} [39, 41].

Other popular multiplication algorithms in literature are the Karatsuba algorithm (with time complexity of $O(n^{log3/log2})$ [23]), and subsequent variants of it [42]. Schönhage-Strassen with time complexity of $O(n.\log n.\log\log n)$ [24] outperforms the Karatsuba algorithm [43].

An extensive analysis and comparison for hardware complexity of various modular multiplications, including Schoolbook (classical), Karatsuba, and FFT, with different operand size, are presented in [42]. Authors calculate hardware complexity of each multiplier by decomposing it into smaller units

such as the full adder, half adder, multiplexer, and gate. Rafferty et al. [44] adopt the same approach to analyze large integer multiplications of both combined and individual multipliers. Karatsuba multiplier outperforms for operands greater or equal to 32 bits. Schoolbook imposes large memory footprint in order to store partial products which negatively impact performance that is mitigated by Comba [45] with the same time complexity but relaxing memory addressing by optimizing the sequence of partial products. Rafferty et al. compare hardware complexity, in terms of +, −, and ∗ units, of different combinations of classical (Comba), Karatsuba, and FFT (NTT for integers) for up to multipliers with 65536-bit operands. However, they evaluate the latency and clock frequency by implementing in hardware (Xilinx Virtex-7 FPGA) for up to 256-bits for the combination of NTT+Comba and 4096-bit for Karatsuba+Comba which are not in a good range for lattice-based cryptography (e.g., $1024 \times 14 = 14336$ bits are used for NewHope key exchange). Based on their (analytical) hardware complexity analysis, the combination of Karatsuba-Schoolbook is the best choice for operands under 64 bits. Karatsuba-Comba is preferable for operands for 64 bits to 256 bits. For larger operands, the lowest hardware complexity is achieved by combined multiplier NTT-Karatsuba-Schoolbook.

2.4.2 The Sampler Component. The quality of a discrete Gaussian sampler is determined by a tuple of three parameters: (σ,λ,τ). In such a tuple σ is the standard deviation (adjusts dispersal of data from the mean), λ is the precision parameter (controls statistical difference between a perfect and implemented discrete Gaussian sampler), and τ is the distribution tail-cut (determines amount of the distribution that we would like to ignore). Each of these parameters affects the security and efficiency of the sampler. For instance, a smaller standard deviation decreases the memory footprint required to store precomputed tables. For encryption/decryption schemes σ=3.33 [46] is suggested. Digital signature sampling from a Gaussian sampler involves large σ=215 [33]. However, by employing Peikert's convolution lemma [47], the standard deviation can be reduced by order of magnitude which is a remarkable improvement on the precomputed table size. Speed and memory footprint is λ dependent, i.e., higher λ results in more secure but a slower and bigger sampler. The tail of the Gaussian distribution touches the x-axis at x=+∞ (considering only the positive side due to the symmetry) with negligible probability. Tail-cut parameter (τ) defines the amount of the distribution that we would like to ignore, hence random number e is sampled in $|e| \in \{0, \sigma \times \tau\}$ instead $|e| \in \{0, \infty\}$. Instead of the statistical distance, Rényi divergence technique [48] can be employed to measure the distance between two probability distributions (e.g., [40]). More precisely, in lattice-based cryptography, Rényi divergence is used to generalized security reductions. Sampling the discrete Gaussian distribution is one the most time and memory hungry parts of lattice-based cryptosystems, due to the demands of high precision, many random bits, and huge lookup tables. To be more specific, the obligatory negligible statistical distance between the implementation of a Gaussian sampler (approximated) and the theoretical (perfect) discrete Gaussian distribution imposes expensive precise floating point arithmetic (to calculate the exponential function) or large memory footprint (to store precomputed probabilities). To keep statistical distance less than $2^{-\lambda}$, floating point precision with more than standard double-precision is obligatory which is not natively supported by the underlying platform; thus software libraries should be used to perform higher floating-point arithmetic. It is impractical to sample from a perfect Gaussian sampler; hence a λ-bit uniform random integer is used to approximate the discrete sampler. Fortunately, it is proven that the Gaussian sampler could be used to achieve $\lambda/2$ security level (approximated sampler) instead of λ level (perfect sampler), since (to date) there is no algorithm that can distinguish between a perfect sampler (λ bits) and an approximate sampler ($\lambda/2$ bits) [49]. In other words, we can cut half of the bits in the sampler which results in a smaller and faster sampler [31, 50, 51]. Reduction in precision parameter (from λ to $\lambda/2$) changes tail-cut parameter (τ) as $\tau = \sqrt{\lambda \times 2.ln(2)}$ [52]. Sampling from a Gaussian distribution may

lead to a timing side-channel attack, which can be avoided by using a constant generic time Gaussian Sampling over integers [53]. Folláth provides a survey of different Gaussian samplers in lattice-based cryptography schemes with a more mathematical outlook [54]. Gaussian samples are classified into six categories and guidelines provided for choosing the best candidate on different platforms for specific parameter ranges. However, we organize Gaussian samplers into the following types discussed below. A Summary of advantage and disadvantages of each sampler are listed in Table 1.

Rejection Sampler. Firstly, x is sampled in $(-\tau\sigma,\tau\sigma)$ uniformly at random where τ and σ are tail-cut and standard deviation of Gaussian distribution. After that, x is rejected with the probability proportional to $1\text{-}\exp(-x^2/2\sigma^2)$. The high rejection rate of samples (on average eight trials to reach acceptance) along with expensive calculation of `exp()` are the main reasons for the inefficiency [55]. Göttert et al. [56] employ rejection sampler for the first time within LBC. Remarkable speed and area improvement could be achieved by accomplishing rejection operation using lazy floating-point arithmetic [57].

Bernoulli Sampler. Bernoulli is an optimized version of rejection sampling in order to reduce average required attempts for a successful sampler from 10 to around 1.47 with no need to calculate `exp()` function or precomputed tables [58]. Bernoulli is introduced in [33] for lattice-based cryptography and used widely in the research community [31, 59, 60]. The main idea behind Bernoulli sampling is to approximate sampling from $D_{\mathbb{Z},k\sigma_2}$ using the distribution $k \cdot D_{\mathbb{Z}^+,\sigma_2} + \mathbb{U}(\{0,\ldots,k-1\})$ where $\mathbb{U}$ is the uniform distribution. The procedure for Bernoulli sampler is shown below in 5 steps.

(1) sample $x \in \mathbb{Z}$ according to $D_{\mathbb{Z}^+,\sigma_2}$ with probability density of $\rho_{\sigma_2} = e^{-x^2/(2\sigma_2^2)}$
(2) sample $y \in \mathbb{Z}$ uniformly at random in $\{0,\ldots,k-1\}$ and calculate $z \leftarrow y + kx$, $j \leftarrow y(y+2kx)$
(3) sample $b \leftarrow \mathcal{B}_{-j/2\sigma^2}$ where $\sigma = k\sigma_2$ and $\mathcal{B}$ is Bernoulli distribution. To sample from $\mathcal{B}_c$ where c is a precomputed constant value, a uniform number $u \in [0,1)$ with λ-bit precision is sampled; 1 is returned if $u < c$, otherwise 0 should be returned.
(4) if $b = 0$ goto to step (1)
(5) if $z = 0$ go to step (1), otherwise generate $b \leftarrow \mathcal{B}_{1/2}$ and return $(-1)^b z$ as the output

The standard deviation of the target Gaussian sampler $D_{\mathbb{Z},k\sigma_2}$, equals $k\sigma_2$ where $\sigma_2 = \sqrt{\frac{1}{2ln2}} \approx 0.849$ where $D_{\mathbb{Z},\sigma_2}$ and $k \in \mathbb{Z}^+$ are uniform distribution parameters. For schemes with small standard deviation (e.g., public key encryption) sampling from binary Gaussian distribution can be eliminated [59], while for digital signatures with large standard deviation, using Gaussian distribution is mandatory [60]. Gaussian distribution can be replaced with other distributions (e.g., uniform distribution [61]). Bernoulli approach avoids long integer calculation and employs single bit operations that make it beneficial for hardware implementation. The time dependency of Bernoulli makes it vulnerable to timing attacks that are resolved in the hardware implementation of BLISS in [60]. Precomputed tables in Bernoulli sampling are small; binary Gaussian distribution (easy to sample intermediate sampler) is independent of σ hence Peikert convolution lemma (smaller σ) does not have a considerable impact on the area. However, convolution lemma reduces the area of precomputed tables in CDT sampler by a factor of 23× for `BLISS` (reduces σ from 215 to 19.47).

Binomial Sampler. Centered Binomial distribution (ψ_k) is a close approximation of the rounded Gaussian sampler (ξ_σ) that eliminates the need for computing $exp()$ and precomputed large tables. Let $\sigma = \sqrt{8}$ be the standard deviation of ξ_σ and a binomial distribution is parameterized with $k = 2\sigma^2$; choosing ψ_k as the sampling distribution has negligible statistical difference with rounded Gaussian sampler with $\sigma = \sqrt{8}$ [40]. Centered Binomial distribution(ψ_k) for integer $k \geq 0$ is defined as sampling $2 \cdot k$ random numbers uniformly from $\{0,1\}$ as $(a_1,\ldots,a_k,b1,\ldots,b_k)$ and output $\sum_{i=1}^{k}(a_i,b_i)$ as the random sample [40]. Since k scales with power 2 of σ, it is not practical to use binomial

sampling for digital signatures with large standard deviation. Binomial sampler has been employed inside software implementation of NewHope [62–65], HILA5 [66], LAC [67], LIMA [68], Kyber [41, 69] and Titanium [70]. It also is used in hardware implementation of NewHope [71].

Ziggurat Sampler. Ziggurat sampler is a variation of the rejection sampler introduced in [72] for a continuous Gaussian sampler [3]. The discrete version of Ziggurat sampler is proposed in [76] which is suitable for schemes with large standard deviation. The area under the probability density function is divided into n rectangles with the same area whose size is proportional to the probability of sampling a point in each rectangle. The left and right corners of each rectangle are on the y-axis and Gaussian distribution curve, respectively. Each rectangle is stored using its lower right coordinates (Figure 2). Firstly, rectangle R_i and point x_i inside the rectangle is chosen uniformly at random. Since we are considering positive x_i, a random sign bit, s, is also required. If $x_i \leq x_{i-1}$, x_i resides below the curve and would be accepted. Otherwise, a uniformly random y_i is sampled and if $y_i \leq exp(x_i)$, the random point (x_i, y_i) is accepted; otherwise new random x should be sampled and the process is repeated.

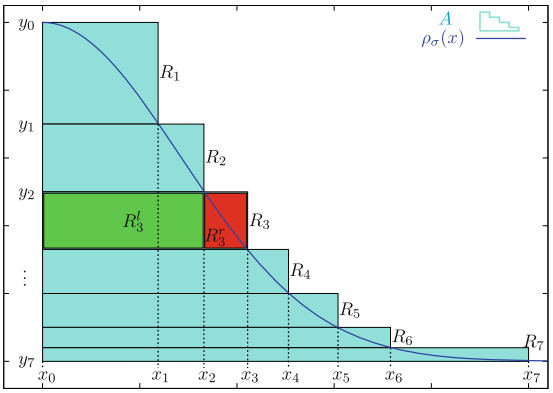

Fig. 2. A partition of Ziggurat [76].

More rectangles leads to reduction in the number of rejections, better performance, and higher precision. Due to its flexibility offers a trade-off between memory consumption and performance, Ziggurat is suitable for resource-constrained embedded devices. More precision and performance require more precomputed rectangles that impose remarkable memory overhead; however, the increase in the number of precomputed rectangles could drop the performance if the cache is fully occupied. Consequently, the software implementation is preferred to any hardware implementation. There is only one hardware implementation of Ziggurat in the literature which hints at the impracticality of realizing the Ziggurat sampler in hardware [52].

Cumulative Distribution Table (CDT) Sampling. CDT is also known as the inversion sampling method [47]. CDT is faster than rejection and Ziggurat samplers by avoiding the use of expensive floating-point arithmetic [77]. Since in cumulative distribution all the numbers are less than 1, it is sufficient to use the binary expansion of the fraction. CDT requires a large table to store values of the cumulative distribution function (CDF) of the discrete Gaussian (highest memory footprint [54]) for which their size is a function of distribution tail-cut (τ) and Gaussian parameter

[3]Another method to sample from a continuous Gaussian is **Box-Muller** [73]. Box-Muller method transforms two independent uniforms into two independent discrete Gaussian distributions. pqNTRUSign [74] and NTRUEncrypt [75] use Box-Muller based Gaussian sampler.

(σ). Variable r is sampled uniformly at random in the range [0,1) with λ bits of precision. The goal is to find an x whose probability is $\rho(x) = S[x+1] - S[x]$ where $S[x]$ equals the value of CDF at x. CDT performs a search, usually binary search, on the CDFto find the corresponded x. A standard CDT needs table of size at least $\sigma\tau\lambda$ bits; for instance BLISS-IV ($\sigma = 13.4, \tau = 215, \lambda = 128$) and BLISS-I ($\sigma = 13.4, \tau = 19.54, \lambda = 128$) need at least pre-computed tables of size 630 kbits and 370 kbits for 192 and 128 bit post-quantum security, which is not practical for resource-constrained embedded devices [33]. By employing Peikert's convolution lemma [47], the size of precomputed tables are dropped by the factor of 11 by sampling twice from $\tau' = \frac{\tau}{\sqrt{1+k^2}} = 19.47$ instead of sampling from a $D_{Z,\tau}$ with $\tau = 215$. Further improvements on table size are presented in [58] by employing adaptive mantissa size which halves the table size.

Pöppelmann et al. [78] proposed a hardware implementation of CDT sampler which is further optimized in [60, 79, 80]. Du et al. [81] propose an efficient software implementation of CDT that is vulnerable to timing attack which is resolved in [82] by suggesting a time-independent CDT sampler. Pöppelmann et al. [59] combine the CDF and rejection sampling to achieve a compact and reasonably fast Gaussian sampler.

Knuth-Yao sampler. The Knuth-Yao sampler [83] provides a near optimal sampling (suitable for the high precision sampling) thanks to its near entropy consumption of the random bits required by the sampling algorithm. Assume n to be number of possible values for each random variable r with the probability of p_r. The probability matrix is constructed based on the binary expansion of each variable whose r^{th} row denotes the binary expansion of p_r. According to the probability matrix, a discrete distribution generating the binary tree (DDG) is built whose i^{th} level corresponds to the i^{th} column of the probability matrix. Sampling is the procedure of walking through the DDG tree until a reaching a leaf and returning its value as the sampling value. At each level, a uniformly random bit indicates whether the left child or right child of the current node should be visited in the future. The Knuth-Yao sampler is suitable for schemes with small standard deviations; thus Knuth-Yao is not suitable for digital signature because of its slow sampling caused by a high number of random bits. In order to minimize the statistical distance between the approximated distribution and the true Gaussian distribution, Knuth-Yao sampler needs large memory to store probability of the sample points with high precision, which is an issue on resource-constrained platforms. Combination of Knuth-Yao and CDT results in about halving the table sizes, which is still prohibitively large; however, the Bernoulli sampler offers the best-precomputed table size [33].

De Clercq et al. [84] introduce an efficient software implementation of the Ring-LWE based cryptosystem with Knuth-Yao as a Gaussian sampler. Using a column-wise method for sampling, Roy et al. [85] propose the first hardware implementation of the Knuth-Yao sampling with small standard deviation which results in faster sampling. Same authors improve their implementation in [86].

Based on the presented results in [76], with the same memory budget, CDT beats rejection sampling and discrete Ziggurat. The Ziggurat sampler outperforms CDT and rejection sampling for larger values of the standard deviation. Ziggurat sampler bears almost the same speedup as Knuth-Yao, while it improves the memory footprint by a factor of 400. As stated earlier, due to the large standard deviation necessary for digital signature, Knuth-Yao sampler is not suitable for digital signatures. Inside an encryption scheme with $\sigma_{LP} = 3.3$ [46], with the same security level ($\lambda = 64$), Knuth-Yao sampler beats CDT in terms of number of operations performed in one second per slice of FPGA (Op/s/S) for time-independent implementation [82]. However, for a time-dependent Gaussian sampler with the same security level ($\lambda = 94$), the CDT sampler proposed by Du and Bai [80] outperforms Knuth-Yao implementation of [86] in term of Op/s/S.

Table 1. Comparison of different Gaussian samplers; partially extracted from [33].

Sampler	Speed	FP exp()	Table Size	Table Lookup	Entropy	Features
Rejection	slow	10	0	0	$45+10log_2\sigma$	suitable for constrained devices
Ziggurat	flexible	flexible	flexible	flexible	flexible	suitable for encryption requires high precision FP arithmetic not suitable for HW implementation
CDT	fast	0	$\sigma\tau\lambda$	$log_2(\tau\sigma)$	$2.1+log_2\sigma$	suitable for digital signature easy to implement
Knuth-Yao	fastest	0	$1/2\sigma\tau\lambda$	$log_2(\sqrt{2\pi e}\sigma)$	$2.1+log_2\sigma$	not suitable for digital signature
Bernoulli	fast	0	$\lambda log_2(2.4\tau\sigma^2)$	$\approx log_2\sigma$	$\approx 6+3log_2\sigma$	suitable for all schemes
Binomial	fast	0	0	0	$4\sigma^2$	not suitable for digital signature

The Size of precomputed tables in a Bernoulli sampler is two orders of magnitude smaller than that of CDT and Knuth-Yao sampler [33]; however, CDT has three times more throughput than Bernoulli for hardware implantation of BLISS in [60].

2.5 Lattice-based schemes

Security of the LBC schemes are based on the hardness of solving two average-case problems, a.k.a Short Integer Solution (SIS) [87] and the Learning With Errors (LWE) problem [7].

Regev [7] propose the LWE problem which can be reduced to a worst-case lattice problem like the Shortest Independent Vectors Problem (SIVP). In the proposed scheme, the ciphertext is $O(n\log n)$ times bigger than plaintext; however, in [88] ciphertext has the same length order compared to the plaintext. A smaller key size for LWE-based encryption is introduced as the LP scheme in [46]. Another difference between Regev's encryption scheme and LP scheme is that the former uses discrete Gaussian sampler in the key generation step, while LP employs Gaussian Sampler in encryption in addition to the key generation step.

2.5.1 Public Key Encryption. Public key encryption (PKE) is employed to encrypt and decrypt messages between two parties. Additionally, it is a basis for other cryptography schemes such as digital signatures. Generally, it consists of three steps including key generation, encryption, and decryption. The first party (Alice) generates two set of keys and keeps one of them private (sk_{Alice}) and distributes the other key (pk_{Alice}) to other party (Bob). In order to send the message M to Alice, Bob encrypts the message as $S = Enc(M, pk_{Alice})$ where pk_{Alice} is Alice's public and Enc is encryption cipher. Thereafter, Alice decrypts the message as $M = Dec(S, sk_{Alice})$ where Dec is the decryption cipher and sk_{Alice} is Alice's private key. Similarly, Alice should encrypt her message with Bob's public key (pk_{Bob}) if she wants to send message to Bob. Encryption and decryption ciphers are one-way functions and are known publicly; hence the only secret data are private keys of the recipients which should be impossible to uncover using their corresponding public keys. Practical lattice-based PKE schemes are either based on NTRU related [89, 90] or LWE [7] (and its variants including RLWE [17], MLWE [19], ILWE [91] and MPLWE [92]) assumptions.

2.5.2 Digital Signature. Digitally signing a document involves sending the signature and document separately. In order to verify the authenticity of the message, the recipient should perform verification on both the signature and the document. Digital signature consists of three steps including key generation, Sign_{sk}, and Verify_{pk}. In the first step, secret key (sk) and public key (pk) are generated; signer keeps the secret key and all verifier parties have the public key of signer. During the sign step, signer applies the encryption algorithm on input message M with its private key and produces output S as $S = \text{Sign}_{sk}(M, sk_{signer})$. Signer sends the tuple (M,S) to the Verifier who applies Verify_{pk} (M,S) and outputs 1 if M and S are a valid message and signature pair; otherwise, S is rejected as the signature of message M. As an example of hash and sign procedure: in the sign step, message M is

hashed as $D = h(M)$ where D is the digest. Signer applies the encryption algorithm on D with its private key with the output of $S = Enc(D, sk_{signer})$. Afterwards, signer sends the pair of (M,S) to the verifier. Subsequently, verifier uses public key to decrypt S as $D' = Dec(S, pk_{signer})$. Then verifier compares $D = h(M)$ (same hash function is shared between signer and verifier) and D'; signature is verified if $D' = D$; otherwise, the signature is rejected. The steps outlined for sign and verify are just an example (used in RSA); hence sign and verify steps might be slightly different for various signature schemes, but follow the same idea.

Lattice-based signature schemes belong to one of two classes including hash-and-sign (e.g. GPV [93]) and Fiat-Shamir signatures (e.g. BG [94], GLP [61], and BLISS [33]). GPV is a provably secure framework to obtain "hash-and-sign lattice-based signature schemes"; GGH [95] and NTRUSign [96] were the two first works that propose lattice-based signatures which are not provably secure (due to the deterministic signing). Because of the high standard deviation of Gaussian sampler, implementation of lattice-based digital signatures that use Gaussian sampler is challenging. Besides, the need for hash function components and rejection step makes digital signature more complex.

2.5.3 Key exchange mechanism. Key exchange is the process of exchanging keys between two parties in the presence of adversaries. If parties use symmetric keys, the same key is shared between them; otherwise, public key of parties should be exchanged. There are numerous public key exchange methods reported in the literature. For classical cryptography, Diffie–Hellman is a practical public key exchange that has been used widely. Establishing a shared key in a lattice-based key encapsulation (KEM) or key exchange (KEX) scheme can be done by either a *reconciliation-based* or *encryption-based* method [62]. Ding [97] propose the first lattice-based key agreement using the reconciliation-based method. There are plenty of reconciliation-based key agreement schemes based on the LWE (e.g., Frodo [32]), RLWE (e.g., BCNS [98] and NewHope [40]) and MLWE (e.g., Kyer[41]) that require less bandwidth compare to the simper encryption-based ones (e.g., NewHope-Simple [62] and NewHope-512/1024 [65]).

Fujisaki-Okamoto transform. In the random oracle model, an IND-CPA [4] public key encryption (PKE) can be transformed into a IND-CCA [5] PKE using the $Fujisaki - Okamoto$ (FO) transform [99]. Hofheinz et al. [100] introduce a variant of the FO transform that performs transformation from CPA-security into CCA-security in the quantum random oracle model. By applying this variant of FO on a CPA-secure PKE, an IND-CCA key encapsulation mechanism (KEM) is achieved. This transformation is widely used in submitted proposals to "NIST call for post-quantum algorithms" [101] where authors first make a IND-CPA PKE and then build the CCA-KEM, with the same parameter space, by applying the KEM version of the transform [100].

There are plenty of submitted proposals to the NIST PQC standardization call that a single proposal (e.g., LOTUS [102]) supports both public key encryption (e.g., LOTUS-PKE) and key agreement (e.g., LOTUS-KEM); in Table 2 and Table 3, we list those proposals as two separate schemes. To avoid the redundancy, in section 3, we describe each scheme under only one of the categories of public key encryption (section 3.2.1) or key exchange mechanism (section 3.2.2). Similarly, to avoid the redundancy, an article with both software and hardware implementations is mentioned only once (software implementation in section 3.2 or hardware implementation in section 3.3).

The contemporary LBC schemes existed in the literature are listed in Table 2. Details on the security level, public key, secret key and ciphertext size of lattice-based public key encryption (PKE), key establishment (KEX/KEM) schemes can be seen in Table 3. Furthermore, details on the security

[4]Indistinguishability under chosen plaintext attack
[5]Indistinguishability under chosen ciphertext attack

level, secret key, public key, and signature size of the lattice-based digital signature schemes are listed in Table 4.

LAC should be added to Table 2

Table 2. Contemporary lattice-based schemes.

Lattice Type	Schemes		
	Public Key Encryption	**Digital Signature**	**Key Exchange**
Standard Lattices	LP [46] Lizard [103, 104] NTRUEnrypt [105] EMBLEM.CPA [106] FrodoPKE [107] LOTUS-PKE [102] Odd-Manhattan [108] uRound2.PKE [109]	GGH [95] NTRUSign[1][96] GPV [93] Lyubashevsky [110] BG [94] TESLA [111]	Frodo [32] EMBLEM [106] FrodoKEM [107] SPKEX[112] OKCN/AKCN-LWE/LWR[4][113, 114] Lizard [104] LOTUS-KEM [102] Odd-Manhattan [108] uRound2.PKE [109]
Ideal Lattices	NTRU [89] NTRU Prime[115] Ring-Lizard (RLizard) [103, 104] trunc8 [116] HILA5 [66, 117] R.EMBLEM.CPA [106] NewHope-CPA-PKE [65] ntru-pke, ss-ntru-pke [75] u/nRound2.PKE [109]	Lyubashevsky [118] GLP [61] GPV [119] BLISS [33] BLISS-B[120] Ring-TESLA [121] TESLA# [51] BLZZRD [50] GLYPH#[122] FALCON [123] qTESLA [124]	JARJAR, NewHope [40] NewHope-Simple [62] BCNS [98] HILA5 [66, 117] NTRU KEM [125] Ding Key Exchange [126] R.EMBLEM [106] OKCN/AKCN-RLWE[4][113, 114] AKCN/OKCN-SEC[4][114] LIMA-sp/2p [68] RLizard [104] NewHope-CPA/CCA-KEM [65] ntru-kem, ss-ntru-kem [75] NTRU-HRSS-KEM [127] Streamlined-NTRU-Prime, NTRU-LPRime [128] u/nRound2.KEM [109]
Module Lattices	Kyber PKE [41, 69] AKCN-MLWE-CCA[4][114] KINDI$_{CPA}$ [129] SABER [130]	Dilithium [131, 132] pqNTRUSign [74]	Kyber KEM[41, 69] CNKE, OKCN/AKCN-MLWE[4][114] KINDI$_{CCA-KEM}$ [129] SABER [130] THREEBEARS[3][91]
Middle Product Lattices	Titanium-CPA [70]	-	Titanium-CCA [70]

[1] Adapted from GGH digital signature scheme;
[2] Adapted from GLP digital signature scheme;
[3] Based on the integer version of the MLWE problem;
[4] From the KCL [114] family;

3 IMPLEMENTATION CHALLENGES

In this section, we consider various implementations of lattice-based cryptographic schemes using software, hardware, software/hardware codesign and DSP techniques. First we focus on the implementation of key arithmetic modules such as Gaussian sampler and matrix/polynomial multiplication in Section 3.1. Software and hardware implementations of lattice-based cryptographic schemes are described in Section 3.2 and Section 3.3, respectively. Section 3.4 describes implementations of lattice-based schemes using hardware/software codesign techniques. The only implementation of a lattice-based scheme using DPS implementation is described in Section 3.5.

Table 5 presents a birds-eye view of popular implementation schemes and can be helpful as a visual/organizational reference during the discussion of various implementation schemes.

3.1 Implementation of Arithmetic Modules

In this section, practical implementations of the Gaussian sampler and polynomial multiplication on both hardware and software platforms are presented. There is only one hardware implementation of matrix multiplication (for standard lattices) available in the literature which we detail in Section 3.1.2.

3.1.1 Gaussian Sampler. Dwarakanath et al. [151] provide a survey of different algorithms of computing the exponential function efficiently on resource-constrained devices regarding

Table 3. Comparison of contemporary lattice-based public key encryption and key exchange schemes.

Scheme	PQ security		Failure	Size (bytes)		
	SVP	CCA	Probablity[1]	Secret Key	Public Key	Ciphertext
BCNS (KEX) [98]	78	-	?	4096	4096	4224
JarJar (KEX) [40]	118	-	55	896	928	1024
NewHope (KEX) [40]	255	-	61	1792	1824	2048
Frodo rec. (KEX) [32] [2]	130	-	36	1280	11296	11288
Kyber light (KEX) [41]	102	169	169	832	736	832
Kyber rec. (KEX)[41] [2]	161	142	142	1248	1088	1184
Kyber paranoid (KEX) [41]	218	145	145	1664	1440	1536
NTRU KEM [125]	123	∞	∞	1422	1140	1281
NTRU Prime (KEM) [115]	129	∞	∞	1417	1232	1141
HILA5 (KEM/PKE) [117]	255	135	135	1792	1824	2012
trunc8 [116] [§]	131	-	45	128	1024	1024
NTRU ees743ep1 (PKE) [89]	159	-	112	1120	1027	980
Ding Key Exchange [126] [4]	AES-256 [*]	-	60	3072	2064	2176
EMBLEM (KEM)[106]	AES-128	-	140	2039180	2036736	78368
R.EMBLEM (KEM) [106]	AES-128	-	140	6144	4096	3104
FrodoKEM (Frodo-976) [107]		AES-192	199	31272	15632	15762
FrodoKEM (Frodo-640) [107]		AES-128	148	19872	9616	9736
KCL (e.g., AKCN-RLWE) (KEM) [114] [4]		AES-256	40	1664	1,696	2083
KINDI (e.g., KINDI-512-3-2-1) (PKE/KEM) [129] [4+]		AES-256	276	2752	2368	3392
LAC (e.g., LAC256) [67] [4+]		AES-256	115	2080	1056	2048
LIMA (e.g., CCA.LIMA-2p2048) [68] [4+]		SHA-512	314	18433	12289	7299
LIMA (e.g., CCA.LIMA-sp2062)[68] [4+]		SHA-512	244	24745	16497	9787
Lizard.CCA (PKE) [104] [4]		AES-256	381	557056	6553600	3328
RLizard.CCA (PKE) [104] [4]		AES-256	305	513	8192	8512
Lizard.KEM [104] [4]		AES-256	381	34880	4587520	35904
RLizard.KEM [104] [4]		AES-256	305	769	8192	8256
LOTUS (PKE/KEM) [102] [4+]		AES-256	256	1630720	1470976	1768
NewHope1024 (KEM) [65] [5]		AES-256	216	3680	1824	2208
NewHope512 (KEM) [65] [5]		AES-128	213	1888	928	1220
Streamline-NTRU-Prime (KEM) [128]		AES-256	∞	1600	1218	1047
NTRU-LPRime (KEM) [128]		AES-256	∞	1238	1047	1175
NTRU-HRSS-KEM [127]		AES-128	∞	1418	1138	1278
NTRUEncrypt (e.g., ntru-kem-743) [75] [4]		AES-256	112	1173	1023	1023
Odd-Manhattan (KEM) [108] [4]		AES-256	?	4456650	4454241	616704
uRound2 (uround2_kem_nd_l5) [109] [4]		AES-256	65	169	709	868
nRound2 (nround2_kem_nd_l5) [109] [4]		AES-256	45	165	691	818
uRound2 (uround2_pke_nd_l5) [109] [4]		AES-256	137	1039	830	953
nRound2 (nround2_pke_nd_l5) [109] [4]		AES-256	164	1039	830	1017
SABER (PKE/KEM) [130] [4]		AES-256	165	3040	1312	1472
THREEBEARS (KEM) [91] [4]		AES-256	188	40	1584	1697
Titanium-CPA (PKE) [70] [4]		AES-256	30	32	23552	8320
Titanium-CCA (KEM) [70] [4]		AES-256	85	26944	26912	8352
DH-3072	-	-	?	416	384	384
ECDH-256	-	-	?	32	32	64

[1] Failure is $-log_2$ of the failure probability;
[2] For recommended parameters;
[3] AES-128 and AES-192 are also available;
[4] Scheme with the highest security is selected;
[5] INP-CPA KEM is also available;
[§] Ring-LWE encryption and authentication system;
[+] INP-CPA PKE is available;
[*] The scheme is at least as hard as AES-256 as a requirement by NIST (same is applied to schemes with security of AES-128, AES-192 and SHA-512);

memory capacity. In order to decrease memory footprint, pipelining the sampling algorithm is offered. To be more specific, authors divide the distribution curve into rectangles with the same probability and choose the rectangles according to the Knuth-Yao method which means the Knuth-Yao method is employed another round for rectangle itself. Tail probabilities in discrete distribution are relatively small which provide the chance of approximating them with lower precision arithmetic. Consequently, on the fly tail construction using standard double-precision floating-point precision is suggested. Although the offered idea could significantly reduce memory (lookup table) footprint since lookup tables only store non-tail probabilities; however, considerable floating point arithmetic overhead is imposed on the system.

Table 4. Comparison of popular lattice-based key digital signature schemes

Scheme	Security		Size		
	PreQ	PostQ	Secret Key	Public Key	Signature
GPV [93]	100	?	256 B	1.5 kB	1.186 kB
BG [94]	128	?	0.87 MB	1.54 MB	1.46 kB
TESLA–128 [111]	128	?	1.01 MB	1.33 MB	1.280 kB
TESLA–256 [111]	256	128	1.057 MB	2.2 MB	1.688 kB
GLP [61]	100	<80	256 B	1.5 kB	1.186 kB
BLISS-I [33] [2] BLISS-BI [120][1]	128	<66	256 B	896 B	700 B
TESLA#–I [51]	128	64	2.112 kB	3.328 kB	1.616 kB
TESLA#–II [51]	256	128	4.608 kB	7.168 kB	3.488 kB
Ring-TESLA-II [121]	118	64	1.92 kB	3.328 kB	1.568 kB
Dilithium rec. [131] [3]	138	125	3.504 kB	1.472 kB	2.701 kB
Dilithium high. [131][4]	176	160	3.856 KB	1.760 kB	3.366 kB
FALCON (falcon1024) [123]	AES256	128	8.193 kB	1.793 kB	1.233 kB
FALCON (falcon768) [123]	AES192	96	6.145 kB	1.441 kB	1.077 kB
FALCON (falcon512) [123]	AES128	64	4.097 kB	897 B	690B
pqNTRUsign [74] [5]	AES256	128	2.604 kB	2.065 kB	2.065 kB
qTESLA (qTesla_256) [124] [5]	AES256	128	8.256 kB	8.224 kB	6.176 kB
qTESLA (qTesla_192) [124] [5]	AES192	96	8.256 kB	8.224 kB	6.176 kB
qTESLA (qTesla_128) [124] [5]	AES128	64	2.112 kB	4.128 kB	3.104 kB
DSA-3072	128	0	416 B	384 B	384 B
ECDSA-256	128	0	32 B	32 B	64 B

[1] BLISS-BI speeds up BLISS-I by factor of 1.2×;
[2] Speed optimized;
[3] For recommended parameters;
[4] The highest security level;
[5] For both Gaussian-1024 and Unifrom-1024 variants;

Table 5. Popular implementation of lattice-based schemes.

Lattice Type	Schemes		
	Software	Hardware	Hardware/Software
Standard Lattices	PKE: [103, 104] [105] [106] [107] [102] [108] [109] DS: [94] [119][133] [111] KEX: [112] [32] [106] [107] [112] [113, 114] [104] [102] [108] [109]	PKE: [31] [106]	-
Ideal Lattices	PKE: [84] [134] [38] [135] [136] [137] [115] [103, 104] [116] [66, 117] [106] [65] [75] [109] DS: [61] [138] [119] [139] [60] [140] [38, 141] [142] [119] [33] [120] [121] [51] [50] [122] [123] [124] KEX: [97] [113] [40] [62] [63] [143] [125] [62] [98] [66, 117] [126] [106] [113, 114] [114] [68] [104] [65] [75] [127] [128] [109]	PKE: [56] [79] [37] [59] [106] [135] DS: [61] [60] [144] [145] [146] KEX: [106] [71] [147] [146]	DS: [148] [149]
Module Lattices	PKE: [41, 69] [130] [129] [114] DS: [131, 132] [74, 150] KEM: [41, 69] [91] [130] [129] [114]	-	-
Middle Product Lattices	PKE:[70] KEM:[70]	-	-

Software Implementation. Buchmann et al. [76] design and implement, in C++, a flexible Gaussian sampler named Ziggurat which sets a trade-off between memory footprint, precision, and execution time. The area under the probability density function (PDF) is divided into rectangles with the same area that is employed to minimize calculation of expensive exponential function. More rectangles (more memory footprint) results in higher precision and better performance. The Ziggurat sampler is attractive because of the potential flexibility that makes it a good candidate to use in crypto-engines of either high-performance servers (allocate more memory to reach better performance and precision) or low-speed resource constraint embedded devices (use few numbers of rectangles to minimize memory consumption). In order to increase the performance of Ziggurat sampler, more rectangles are required which imposes significant memory overhead since it needs to re-compute all the rectangles and save them in the memory. A better way to improve Ziggurat sampler is to increase the number of approximation lines (by adding two extra points) which

culminates in decreasing number of the exponential function calculation [152]. In other words, more approximation lines decrease the probability of rejection by providing a more precise approximation of the curve. Consequently, performance and memory occupation are improved by employing more approximate lines.

Hardware Implementation. Roy et al. [85] implement the first high precision and low area hardware implementation of Knuth-Yao sampler with small standard deviation on a Xilinx Virtex5 FPGA. Knuth-Yao involves a random walk tree traversal which imposes expensive sequential bit scanning and wide ROM footprint. To improve performance, authors traverse the discrete distribution generating (DDG) tree by employing relative distance of intermediate nodes. Column-wise storing of the samples probability in ROM improves the performance of the sampler. Numerous zeros in the probability matrix are well compressed by applying one-step compression which culminates in a near-optimal number of random bits to generate a sample point. Presented Knuth-Yao sampler suffers from vulnerability to timing and power attacks due to the non-constant time random walk which is solved by a random shuffle approach in order to eliminate leaking the timing information [86]. Authors offer the solution (random shuffle) but do not evaluate hardware implementation of the shuffler. In the new implementation, the efficiency of the Knuth-Yao is enhanced by employing small LUTs with the input of the random bits and output of the sample point with high probability or an intermediate node positioned in the discrete distribution generation tree. Employing a lookup table with 8-bit input results in hitting a sample point (eliminate expensive bit-scanning procedure) with the probability of 97% by eight random bits. Additionally, a more compact sampler is achieved by reducing the width of ROM and random bit generator.

Du et al. [80] implement a precise (large tail bound) and area efficient cumulative distribution function (CDF) inversion variant of the discrete Gaussian sampler on Xilinx Spartan-6 FPGA. Authors reduce area occupation by employing piece-wise comparison (results in 90% saving of the random bits) and avoiding comparison of large numbers which is an improvement of [79]. Further improvement is achieved by employing small LUTs with high hit rate which results in performance improvement. Performance of the proposed Gaussian sampler is improved twofold by the same authors with a software implementation (Intel Core i7-4771) [81]. The primary challenge to improve performance on a general purpose processor is the large number of random bits that are consumed by the sampler to generate a random number. This obstacle is alleviated (around 30%) by employing multi-level fast LUT (using eight smaller lookup table instead of one). Further speed improvement of the sampler is gained by applying the multithreading technique. The main security drawback of both hardware and software implementation of the discrete Gaussian sampler by Du and Bai is its vulnerability to timing attacks because of the non-constant time traversal of the binary tree [52].

Howe et al. [52] propose a comprehensive evaluation of various practical hardware implementation of time-independent discrete Gaussian samplers, including Bernoulli, discrete Ziggurat (First hardware design on FPGA), CDT, and Knuth-Yao. They present each sampler's weaknesses/strengths and perform the comparison with state-of-the-art designs regarding memory footprint, performance, and FPGA resource consumption. Authors analyze different quantum secure parameters for public key encryption scheme and digital signature. CDT Gaussian sampler provides higher throughput with lower memory footprint when it is used for digital signature. Due to the inferior performance of the hardware Ziggurat implementation, authors prohibit the use of Ziggurat sampler for digital signatures. Similarly, CDT sampler achieves better-balanced area and throughput for public key encryption. However, allowing the use of BRAMs makes Knuth-Yao variant a much superior design in terms of area and throughput. Authors use BRAMs in order to decrease occupied slices in FPGA and to save precomputed values which significantly improves performance.

Pöppelmann and Güneysu [59] propose an area optimized hardware implementation of Bernoulli sampler that employs Bernoulli evaluation instead of evaluation of `exp()`. Rejection probability is high due to the absence of binary Gaussian distribution (easy to sample intermediate sampler) which results in increasing the entropy consumption and runtime. Although proposed Gaussian sampler is suitable for encryption schemes, it would be challenging to be employed inside digital signatures as the sampling component. Pöppelmann et al. [60] propose a hardware implementation of Bernoulli sampler with binary Gaussian distribution for `BLISS` scheme on Xilinx Spartan-6 FPGA.

Göttert et al. [56] propose the first hardware implementation of the discrete Gaussian sampler. Authors use rejection sampling in their software implementation; however, because of the obligatory floating point arithmetic, authors prefer to employ lookup tables in their hardware implementation of implement Gaussian which the Gaussian distributed values in the stored array are indexed using a pseudo-random bit generator. The proposed sampler has an unsatisfactory precision which has far distance from golden discrete Gaussian distribution due to the small tail bound. Authors employ a fully parallel architecture to design a polynomial multiplier which provides high throughput but makes the design extremely big which cannot be fitted into the largest Virtex-7 FPGA family.

Roy et al. [37] propose a compact Ring-LWE cryptoprocessor to optimize NTT multiplication which is accomplished by the reduction in fixed computation (4 NTT instead of 5 NTT) and in reducing the pre-scaling overhead. Besides, NTT Memory access is minimized by storing two coefficients in a single word, processing two pairs of coefficients together, and eliminating idle cycles. Authors avoid using ROM to save the twiddle factors and instead compute the twiddle factors on demand. Besides, they reduce security by limiting coefficients of the secret key to be binary, instead of Gaussian distributed, which gives the opportunity to replace multiply with addition operations. Small lookup tables are used in the Knuth-Yao discrete Gaussian sampler to avoid expensive bit scanning [85] to improve speedup; additionally, ROM widths are reduced which results in a more compact and faster sampler than Bernoulli [59].

3.1.2 Multiplier.

Software Implementation. Emeliyanenko [153] proposes an efficient 24-bit modular multiplication to achieve high throughput polynomial multiplications using NTT algorithm (on an Nvidia GPU). Employing CUDA FFT kernel and Chinese Remainder Theorem (CRT), the proposed method provides better speedup compared to the NTL [154] libraries for moderate coefficient bit-length.

Akleylek et al. [155] propose the sparse polynomial multiplication for the first time which improves the performance of digital signature proposed in [144] by 34%. Authors implement the Schönhage-Strassen polynomial multiplication on an NVIDIA GPU and compare its performance with existed multiplication schemes, including iterative NTT, parallel NTT and CUDA-based FFT (cuFFT) for different integer size.

Akleylek et al. [156] propose a software implementation (Intel Core i5-3210M) of the sparse polynomial multiplication using a sliding window that bears around 80% speed improvement compared to NTT [138]. Authors assume to have polynomials with coefficients with three possible values including -1,0 and +1. Multiplications by zero are avoided, and for +1 and -1 cases addition and subtraction are used, respectively. The method can be used for polynomials with arbitrary coefficients by substituting a multiplication with a loop of additions. Performance depends on high number zeros and numerous identical patterns which make the system prone to timing attacks.

`FNLlib` [157] is a scalable, efficient, and open source C++ library contains optimized arithmetic operations on the polynomials for the ideal lattice-based cryptography schemes. Compared to the generic libraries for polynomial arithmetic, several orders of magnitude improvement in the speed is achieved by employing algorithm optimizations, including fixed sized `CRT`, scalar modular multiplication and NTT algorithm, and programming-level optimizations, such as SSE and AVX2

SIMD. Authors use NFLlib for the RLWE encryption scheme and homomorphic encryption and compare their efficiency with classical cryptographic schemes (like RSA) and libraries (like NTL).

Longa et al. [158] propose an efficient modular reduction by limiting the coefficient length to 32 bits. Consequently, by employing the new technique in NTT, the reduction is only required after multiplication. Combined with the lazy reduction in NTT, the speed improvement of 1.9x for C implementation (on a 64-bit platform) and 1.25x for AVX2 vector implementation compared to NewHope (tolerant against timing attacks) is achieved. However, due to the lack of 64-bit register, proposed reduction technique does not provide any speed up on 32-bit microcontrollers [63]. Additionally, authors use signed integer arithmetic which optimizes the number of add operations in both sampling and polynomial multiplication.

Hardware Implementation. Howe et al. [31] propose the only hardware implementation of standard lattice-based encryption scheme based on LWE problem. Authors perform the multiply-accumulate (MAC) operations of matrices in the encryption scheme by utilizing a dedicated DSP48A1 unit of the Spartan-6 FPGA to achieve an area optimized hardware implementation of standard LWE based encryption engine.

Pöppelmann et al. [78] propose the first hardware optimization of polynomial multiplication (NTT) for the ideal lattice-based encryption schemes on a Xilinx Spartan-6 FPGA with the primary goal of minimizing the area. Authors design a sequential NTT (one butterfly operator) that stores twiddle factors in a dedicated ROM which imposes memory overhead but achieves decent performance. An optimized version of the presented NTT polynomial multiplier is employed in [79] to be imployed in a Ring-LWE encryption engine. With the acceptable runtime, Aysu et al. [37, 159] present an optimized hardware implementation of the NTT introduced in [78] to compute polynomial multiplication in a Ring-LWE on the smallest Spartan-3. The main idea is to compute twiddle factors on the demand instead of storing them in the ROM. By replacing the modulus with a Fermat number, shift operation can be used instead of polynomial exponentiation. However, proposed optimizations cannot take advantage of inherent parallelism in NTT. It should be mentioned that authors of [159] do not provide the implementation of the whole crypto-engine.

Chen et al. [160] present a high-performance polynomial multiplication for Ring-LWE encryption cryptosystems in hardware on a Spartan-6 FPGA by exploiting the parallel property of the NTT. Authors provide a different secure set of parameters by which efficient Ring-LWE encryption is achieved. They prove that polynomial multiplication can be done by computing the negative wrapped convolution by which there is no need to compute the modular reduction. To be more specific, the proposed architecture for polynomial multiplication consists of two butterflies and two point-wise modulo p multiplier (p has the adjustable length) which produce outputs (equivalent to two parallel NTT) that can be used to perform the inverse-FFT.

Du et al. [161] propose a scalable and efficient polynomial multiplier architecture, implement on the Xilinx Spartan-6 FPGA, that takes advantage of NTT's parallelism which provides a speed and area trade-off. In [78] and [37, 159] one and two butterfly operators are employed, respectively; however, [161] use b (power of 2) butterfly operators to improve speed of the polynomial multiplier which perform multiplication of two n-degree polynomials in $(1.5n + 1.5n\log n)/b$ cycles. To minimize the area, authors employ the cancellation lemma to minimized number of constant factors. Butterfly operation takes two coefficients (x,y) and one constant factor (ω) and makes two new coefficients $([x+\omega y] \bmod p, [x-\omega y] \bmod p)$. The butterfly can be used as a modulo p multiplier (by setting $x = 0$). In the first stage of the inverse NTT, the constant factor (ω) is one, new coefficients are $[x-y] \bmod p$ and $[x+y] \bmod p$. Consequently, necessary clock cycles to calculate a sequential polynomial multiplication is $(1.5n + 1.5n\log n)$ which reduces $3.5n$ of required cycles.

Györfi et al. [162] perform a thorough evaluation of various candidates to implement modular FFT on Xilinx Kintex-7 FPGA. Authors study three architectures in the diminished-one number system (computations over Z_{2k+1}) for different parameters in order to meet various factors such as run-time, throughput, area, and scalability. The first architecture yields the best performance, by using pipelined modular butterfly-based FFT. The other two architectures are serial distributed arithmetic-based and nested multiplication which occupy less area than the butterfly-based FFT.

Du et al. [163] achieve 30% savings in the time and space compared to [78] on a Spartan-6 FPGA by performing on-the-fly bit-reversal step along with a new memory access scheme, (to load/store coefficients in calculating NTT). The idea is to load/store at address bit-reverse(i) instead of load/store at address i in memory, which means ith coefficient of NTT's output is located in the *bit-reverse(i)*th memory location. Consequently, the bit-reversal step in the inverse-NTT is eliminated. Authors employ two Block RAMs on FPGA which provide interleaving, hence two parallel NTT can be interleaved. Authors apply their optimization to the NTT of an RLWE base public key cryptosystem [164]. Also, it is assumed uniformly random polynomial in the public key scheme to be fixed, hence precomputing the NTT of it offline improves the performance. In both papers, only one butterfly operator is used; however, by adapting bit-reversal and memory saving methods of above articles, authors propose a fast polynomial multiplication architecture with four butterfly operators that achieve on average 2.2 speedup improvement [165]. Two butterfly operators are used to calculate the ith level, and other two perform $(i + 1)$th level calculations in the pipeline by using results of the ith stage.

3.2 Software Implementations of Lattice-based Cryptographic Schemes

3.2.1 Public Key Encryption. de Clercq et al. [84] propose an efficient software implementation of Ring-LWE based encryption for ARM Cortex-M4F micro-controller. The main goal was maximizing the speed (using assembly level optimization) and minimizing memory footprint (by storing two coefficients in one word). They employ the Knuth-Yao algorithm to achieve fast noise sampling and use the platform's True Random Number Generator (TRNG) to generate random numbers. Authors employ optimization of the paper [37] including instruction-level parallelization. Additionally, polynomial multiplication is optimized by integrating multiple coefficients into one large word allowing load/store operations to be performed with a single instruction.

Liu et al. [134] employ a byte-wise scanning method to improve the performance of Gaussian sampler based on the Knuth-Yao algorithm which allows them to implement a Ring-LWE based public key encryption scheme on a resource-constrained 8-bit ATxmega128 AVR processor. By applying sophisticated memory alignments for storing coefficients, about 20 percent decrease in memory usage is achieved. For NTT computation a couple of optimization techniques are employed including approximation based reduction and negative wrapped convolution.

Buchmann et al. [136] implement a high performance and lightweight public key encryption scheme on a small and 8-bit ATXmega128 and 32-bit Cortex-M0 micro-controllers by replacing the Gaussian noise distribution with a uniform binary error distribution. The main advantage of the scheme over Lindner-Peikert's proposal (LP) [46] is the smaller key and ciphertext size. Regarding the speed, it is beaten by the scheme in [134] with slightly higher memory footprint. Similarly, proposed design in [38], uses NTT with precomputed twiddle factors and eliminating the bit reversal step which results in twofold performance improvement.

Yuan et al. [137] provide a portable JavaScript implementation of LBC schemes on PC web browsers, Tessel (an embedded system for IoT applications), and Android devices. To compute polynomial multiplication in Ring-LWE schemes NTT is used, while Karatsuba algorithms [23] is employed for NTRU schemes. In order to reduce the execution time, inverse transform sampling is employed in which possible values are precomputed and stored in a LUT.

Reparaz et al. [135] implement a masked Ring-LWE scheme on a Virtex-II FPGA and 32-bit ARM Cortex-M4F which is Differential Power Analysis (DPA) resistant. In order to be resilient to first-order side-channel attacks, a constant time masked decoder with high success probability is implemented. Entire computation is done in the masked domain by employing a dedicated masked decoder which imposes considerable time and area overhead compared with an unprotected design.

Cheon et al. [103] exploits LWR problem [18] and present Lizard and its ring variant (Ring-Lizard). Discrete Gaussian error distribution is replaced with an efficient rounding process with smaller modulus. Lizard beats NTRU and RSA encryption schemes by factors of 3 and 5, respectively. The main idea behind the Lizard is to eliminate the least significant bits of the ciphertext rather than integrating the message with some error.

Cheon et al. [104] submit Lizard (IND-CPA/CCA PKE and IND-CCA2 KEM) and its ring variant, RLizard, to the NIST PQC standardization call (NIST security category of 1, 3 and 5). Sparse and small secrets version of LWE and LWR (RLWE and RLWR) are the security basis of the Lizard (RLizard) IND-CPA PKE. Besides an Intel Xeon CPU, authors provide performance evaluation on a smartphone (Samsung Galaxy S7) for their recommended parameter of Lizard.CPA (128-bit quantum security). Authors claim that Lizard is suitable for smartphones (memory usage of 20 megabytes). Authors provide datapath and finite state machine for hardware implementation of the Lizard PKE using Lizard.CPA and RLizard.CPA.

Chen et al. propose NTRUEncrypt [75] (NIST standardization call), a family of IND-CCA2 (resistant to subfield attacks) PKE and KEM schemes at 85, 159 and 198-bit post quantum security (NIST security category 1, 5 and 5). Based on the original NTRU scheme [89] and using parameters set of [105], ntru-pke and ntru-kem are achieved by applying NAEP transformation [166]. Using the same transformation, based on the provably secure NTRU encryption scheme [90], ss-ntru-pke and ss-ntru-pke are derived. Modulus is chosen to be a power of 2 (2^{11}) to enhance efficiency of the modulo arithmetic and integer multiplications. Authors adapt a PRNG from Salsa20 [167] to expand the seed. Box-Muller [73] is employed (only in ss-ntru-pke and ss-ntru-kem) to sample from the discrete Gaussian distribution. The performance results are reported only for Intel i7-6600U processor (AVX2 optimization for NTT is not performed).

Bernstein et al. [128] introduce two ideal lattice-based KEMs named Streamlined-NTRU-Prime and NTRU-LPRime with 248-bit and 225-bit security (NIST security category 5), respectively, with the ciphertext and the key size of around 1kB that are designed to reduce attacker's success probability by eliminating the ring homomorphisms. Schemes are IND-CCA2 where a key can be used multiple times; hence large key generation latency is tolerable. Authors implement the reference code on an Intel Xeon. Streamlined-NTRU-Prime is faster than NTRU-LPRime in terms of the encapsulation and decapsulation time with slower key generation.

Hülsing ea al. propose NTRU-HRSS [127], a one-way CPA secure (OW-CPA) PKE and NTRU-HRSS-KEM, a CCA2-secure KEM derived from NTRU [89] with 123-bit post-quantum security (NIST security category 1). In contrast to NTRUEncrypt [75] and standard NTRU [168], KEM is derived directly from NTRU-HRSS without using padding techniques (e.g., [166]). Contrary to Streamlined NTRUPrime [115] and standard NTRU, the correctness of NTRU-HRSS does not rely on the fixed weight distinctions. NTRU-HRSS is designed based on the worrisome algebraic structure of cyclotomic rings (in contrast to Streamlined NTRUPrime). Additionally, NTRU-HRSS has probabilistic encryption, while Streamlined NTRUPrime has deterministic encryption. NTRU-HRSS uses the power of 2 modulus rather than the prime modulus (used in Streamlined NTRUPrime) which leads to faster arithmetic computation. NTRU-HRSS employs trinary secret key/message and large modulus in order to avoid decryption failure (in contrast to LWE-based schemes with a non-zero probability of failure) which leads to lower security and higher communication cost. Authors report the performance results of the reference and AVX2 implementation on Intel Core i7-4770K CPU.

Bansarkhani proposes KINDI [129] (at NIST PQC standardization call), a trapdoor-based encryption scheme based on LARA [169] in which data is concealed into the error without changing the target distribution. Consequently, more data is encrypted per ciphertext bit which reduces the message expansion factor (beneficial in "sign-then-encrypt"). $\text{KINDI}_{\text{CPA}}$, (Module-LWE based IND-CPA PKE) has been proposed with five different parameter sets ranging from 164 to 330-bit security (NIST security category of 2, 4 and 5). By applying a variant of Fujisaki-Okamoto (FO) transformation [100] on the $\text{KINDI}_{\text{CPA}}$, $\text{KINDI}_{\text{CCA-KEM}}$ with the same parameter space, can be built.

3.2.2 Key Exchange. Ding et al. [97] propose a provably secure Ring-LWE key exchange mechanism which is not passively secure since it produces biased keys. Peikert improves the protocol by using a new reconciliation method which generates unbiased keys [170]. Bos et al. [98] propose a practical constant-time software implementation of the Peikert's Ring-LWE key exchange protocol, namely BCNS, which can be added as the key exchange protocol to the transport layer security (TLS) protocol in OpenSSL along with RSA as the authentication and SHA-256 as the hashing method. The most time-consuming part of the protocol is the Gaussian sampler, which is done by employing a constant-time search on the Cumulative Distribution Table (CDT). Authors adapt the FFT from Nussbaumer's method [26] for polynomial arithmetic in cyclotomic rings whose degree is a power of two that provides efficient modular reduction. BCNS employs a fixed polynomial as the system parameter which can be a potential weak link of the protocol. Selection of a large modulus results in lower efficiency and security level, 78-bit quantum security, than expected from a Ring-LWE scheme. In contrast to the digital signature and encryption schemes, key exchange scheme does not need a high-quality Gaussian sampler [40], which BCNS uses; consequently, a simpler noise distribution is used in NewHope instead of Gaussian sampler. BCNS caches keys, which can be very dangerous to the security of the protocol because of the shared-key reused attacks [171], which is solved in the NewHope.

Alkim et al. [40] introduce NewHope, a portable C and highly optimized SIMD implementation (AVX2) of unauthenticated key exchange scheme, that solves the inefficiency (10 times better performance) and security drawbacks (increase quantum security level from 78-bit to 128-bit) of BCNS by optimizing the key exchange algorithm and better parameter selection. A better analysis of failure probability which results in smaller modulus, on the fly generation of the polynomial system parameter, efficient polynomial arithmetic (combining Montgomery and Barret reduction and employing polynomial encoding), and using the centered binomial instead of the discrete Gaussian distribution are the main improvements of NewHope over BCNS. NewHope has attracted the attention of research and industry communities such that Google released the Chrome Canary which uses NewHope as the key exchange protocol along with elliptic curve Diffie–Hellman as the authentication protocol [172].

Alkim et al. [62] propose NewHope-Simple, a simpler variant of the Newhope with the same performance and security level. Simplicity is achieved by eliminating the error-reconciliation mechanism [97] with 6% message size overhead. Authors discard the least significant bits of each coefficient due to their negligible impact on the successful plaintext recovery. Additionally, authors encode a single key bit into four coefficients that results in the reduction of the ciphertext length. In NewHope-Simple, polynomial a can be fixed, while the original NewHope generates a on the fly for every single run of the scheme. Alkim et al. [63] present the software implementation of NewHope on ARM Cortex-M family, low power Cortex-M0 and high-performance Cortex-M4, which is the first key exchange scheme with the quantum security level of 128-bit on constrained embedded devices. Authors optimize all hot regions of protocol in assembly, including error reconciliation, the uniform noise generation by ChaCha20 stream cipher [173], and NTT/NTT^{-1}. For NTT, authors set a memory-time trade-off for precomputing powers of constants (design parameters) by which only a

subset of the powers of constants are precomputed and stored in the table. Gueron and Schlieker [143] further optimize the NewHope by optimizing the pseudorandom generation part which results in 1.5× better performance on the Intel Skylake processors. Authors improve the sampling step by lowering the rejection rate (from 25% to 6%) and exploit the parallelism in the pseudorandom generation (replace SHAKE-128 with the parallelized SHA-256 or AES block cipher) and rejection sampling (employing AVX vector instructions). Longa and Naehrig [158], employ a reduction technique (during the NTT calculation) that eliminates the modular reduction after additions of two polynomials which results in the speed improvement of 1.9 and 1.25 for C and AVX implementations (compared to the reference NewHope [40]), respectively.

Adapted from the NewHope-Simple, Alkim et al. [65] propose NewHope as a family of KEMs, at NIST PQC standardization call. The submitted proposal includes NewHope512-CPA-KEM and NewHope512-CCA-KEM ($n = 1024, q = 12289$) which targets 101-bit security (NIST security category level 1) and NewHope1024-CPA-KEM and NewHope1024-CCA-KEM with 233-bit security (NIST category security category 5) with comparable performance as the elliptic curve based cryptosystems. Four mentioned KEMs are derived from NewHope-CPA-PKE (which does not support arbitrary length messages, hence can not be used as a standalone encryption scheme) by applying a variant of Fujisaki-Okamoto transform [100]. In order to generate the random number and shared secret, hash function `SHAKE256` [174] is used as a pseudorandom function; generation of the shared polynomial a is done by expanding a 32-byte seed using `SHAKE128` [174]. Besides the reference and vectorized (using AVX instructions) implementations on the Intel Core i7-4770K (Haswell) processor, authors provide implementation and optimization of KEMs on a 64-bit MIPS architecture (MIPS64).

Ding et al. [126] propose (at NIST PQC standardization call) *Ding Key Exchange*, an ephemeral IND-CPA secure error reconciliation-based key exchange protocol from RLWE problem. At the same security level, Ding Key Exchange reduces communication cost (due to its rounding technique) compare to the similar schemes (NewHope, NewHope-Simple, and Kyber). It provides equivalent security to AES-128, AES-192 and AES-256 (NIST security category 1,3 and 5) with flexible parameter choices and key size of n-bit where n can be 512 and 1024; as a result, it is more resistant to Grover algorithm compare to NewHope, NewHope-Simple and Kyber with the key size of 256-bit. NTL library [154] and CDT sampler are used for polynomial multiplication and sampling from the Gaussian distribution.

`HILA5` [117], a Ring-LWE-based KEX (and also PKE) with the same security parameters ($n = 1024, q = 12289$) and sampler (binomial sampler ψ_{16}) as NewHope and has been tested on Intel Core i7-6700 CPU and is integrated into OQS and OpenSSL. HILA5 uses SafeBits, improved version of the Peikert reconciliation mechanism [170], to reach slightly smaller messages than NewHope (36-byte which is 0.9%) at the same security level by generating unbiased secret bits and hence less randomness in secret bits. HILA5 employs an efficient constant time error correction block to correct 5 bits of error which results in decryption failure of 2^{-128} compared to NewHope's failure rate of 2^{-64} (Frodo [32] and Kyber [41] with failure rate of $2^{-38.9}$ and $2^{-71.9}$) by sacrificing less than 4% of performance. HILA5 can be employed as a PKE scheme due to its higher reliability. `HILA5` [66] is submitted to the NIST PQC standardization call as a famility of PKE and KEM schemes that provide equivalent security to AES-256 (NIST security category 5). Optimized polynomial multiplication and error sampling are performed by employing the Cooley-Tukey [27] method and binomial distribution. Besides, SHAKE-256 is used to sample from the uniform distribution.

`Frodo (FrodoCCS)` [32], the first practical implementation of public key exchange scheme based on standard lattices, the original LWE problem [7], is secure against cache-timing attacks. Like BCNS, Frodo can be integrated into OpenSSL such that Google has announced that Frodo is used in 1% of Chrome web browsers. Matrix arithmetic compared to polynomial arithmetic imposes considerable overheads on the bandwidth (4.7x more than NewHope), throughput (1.2 less throughput than

NewHope), and performance (8x slower than NewHope). Massive memory overhead is imposed if matrix variant should be saved in the memory. By generating and afterward discarding the matrix variant (on-the-fly), memory overhead is alleviated. Authors use an efficient inversion sampling method that uses precomputed tables. Based on the authors' claim, integration of Frodo into TLS halves the server throughput. Consequently, NTT-friendly prime and polynomial are not crucial which results in a negligible drop in performance compared to NewHope [40].

FrodoKEM [107] is a family of IND-CCA secure KEMs based on the LWE problem with brute-force security of at least AES-128 (FrodoKEM-640) and AES-192 (FrodoKEM-640). FrodoPKE is transformed by a variant of FO transformation [100] to build the FrodoKEM. Authors generate public matrix A from a small seed using PRNG (AES128 or cSHAKE128) which results in a more balanced ciphertext and key sizes, but remarkable computational overhead. Timing and cache attacks are prevented by prohibiting the use of secret address accesses and branches. A portable C code reference and its optimized implementation (of generating the public matrix A and matrix arithmetic) are provided. Authors report the results of the implementation on a 64-bit ARM Cortex-A72 (with the best performance achieved by using OpenSSL AES implementation, that benefits from the NEON engine) and an Intel Core i7-6700 (x64 implementation using AVX2 and AES-NI instructions). Employing modular arithmetic ($q \leq 2^{16}$) results in using efficient and easy to implement single-precision arithmetic. The sampling of the error term (16 bits per sample) is done by inversion sampling using a small LUT corresponds to the discrete cumulative density functions (CDT sampling).

Open Quantum Safe (OQS) [175] software platform is designed to evaluate quantum-resistant schemes which have an open-source library (contains C implementation of BCNS, NewHope, Frodo, etc.) of PQC schemes. OQS offers the chance to integrate the quantum resistant schemes into classical applications and protocols with the goal of minimizing the software change; besides, OQS provide the opportunity to compare PQC schemes with each other or with classical cryptographic schemes.

Jin et al. [113] present symmetric (OKCN) and asymmetric (AKCN) LWE and Ring-LWE based key exchange mechanisms. OKCN, optimally-balanced key consensus with noise, can be used for the key transport and encryption, while AKCN, asymmetric key consensus with noise, only can be employed for the key transport. In the proposed scheme, the server sets the session key before starting the key exchange mechanism. Consequently, it provides the opportunity to offline encryption of the message which provides higher security and better workload balance. Compares with Frodo, OKCN-LWE produces a much smaller matrix by eliminating the least significant bits of each LWE sample which results in less computation for matrix arithmetic; smaller matrix also results in the faster generation and sampling of the matrix. With the same set of parameters, OKCN-LWE consumes more bandwidth (30%) than Frodo, while its failure probability is remarkably lower. Employing the same optimization techniques, Ring-LWE based version of OKCN, which adapts the same noise distribution and parameters of NewHope, provides a more computationally efficient scheme than NewHope. Authors integrate the OKCN-LWE scheme into the open safe project platform [175].

Zhao et al. [114] extend [113] and present a generic construction of the authenticated key exchange, PKE and KEM schemes based on LWE/RLWE, LWR, and MLWE problems (submitted to NIST PQC standardization call as KCL (pka OKCN/AKCN/CNKE)). OKCN-LWE and OKCN-LWR key exchange mechanism require less bandwidth (18% and 28%) compare to Frodo at the same security level (shared key size of 256-bit). The most efficient KEX with the share key size of 512-bit is achieved by AKCN. Authors prove that the errors in different positions in the shared key are independent and propose single-error correction (SEC) code to correct at least one bit error; using the SEC, with the same security and error rate, OKCN/AKCN-RLWE based KEX schemes generate 765-bit shared key with less bandwidth than NewHope and NewHope-Simple (with 256-bit shared key). Authors claim that they provide the most efficient lattice-based KEX with the share key size of 256-bit by applying

OKCN/AKCN to MLWE-based key KEX. Additionally, they provide a new authenticated key exchange scheme named concealed non-malleable key-exchange (CNKE).

Seo et al. [106] propose error-blocked multi-bit key encapsulation mechanism named EMBLEM and (R.EMBLEM) which is (secure against adaptive chosen-ciphertext attack) based on the small secret LWE (RLWE) problem. During the decryption phase, the error does not affect the message by separating the message and error and concatenating each message block with the error-blocking bit. The secret key is sampled uniformly at random in $[-B, B]$ (where B is a positive integer smaller than σ) instead of Gaussian distribution. Consequently, the key size is notably reduced since the secret key can be generated by a 256-bit seed which eliminates the need for storing the whole matrix; however, it imposes computational overhead to generate the secret key from the seed using pseudorandom functions. For polynomial multiplication in R.EMBLEM, Cooly-Tukey butterfly and Gentleman-Sande butterfly are used in NTT and inverse NTT, respectively. Besides the software implementation on Intel core-i7-7600, authors implement schemes on the Zynq 7 FPGA platform.

`Kyber` [41] is a highly optimized IND-CCA KEM with the post quantum security based on the hardness of solving the (`Modulue-LWE`) problem [19]. Ideal lattices with their ring structure decrease public key and ciphertext size of standard lattices schemes by sacrificing security assumption. Module lattices proposed to fill the gap by believing that full ring structure is excessive [19]. Authors define IND-CPA PKE scheme under Module-LWE hardness assumption and apply a variant of Fujisaki-Okamoto transform [100] to build a IND-CCA KEM. Employing IND-CCA KEM, they design IND-CCA KEX and AKEX under hardness assumption in the classical and quantum random-oracle models. Kyber works over only one ring, $R_q = \mathbb{Z}_{7681}[x]/(x^{256} + 1)$, which provides flexibility (e.g. performing polynomial multiplication) to sacrifice security (from 128-bit to 102-bit) to improve performance and communication size (33%) (by only changing k from 3 to 2). This flexibility is exclusive to Kyber (Module-LWE schemes); in Ring-LWE schemes, changing the security parameters results in building a new ring R_q and ring operations. Kyber has been submitted to the NIST PQC standardization call for as Kyber512, Kyber768, Kyber1024 at 102, 161 and 218-bit security (NIST security category 1, 3 and 5) [69].

Lu et al. [67] present `LAC` (**LA**ttice-based **C**ryptosystems) that includes an IND-CPA PKE (`LAC.CPA`), a passively secure KEX (`LAC.KE`), an IND-CCA KEM (`LAC.CCA`) and an AKEX (`LAC.AKE`) all of which are based on the RLWE problem. The main design concern is to enhance bandwidth efficiency (reduce key and ciphertext size) by setting modulus q to be small ($q = 251$) which prevents the direct use of NTT in LAC. Although employing AVX2 vector instructions improves the performance of the polynomial multiplication by a factor of 30, polynomial multiplication imposes remarkable computational pressure on the systems without the support of vector instructions. Sampling the secret and error term is done by employing the centered binomial distributions. LAC is proposed with three set of parameters that are much more expensive to break than AES128, AES192, and AES-256 (NIST security category of 1, 3 and 5).

Smart et al. [68] propose `LIMA` (**Latt**I**ce** **MA**thematics), a family of IND-CCA and IND-CPA RLWE-based PKE (based on `LP` [46]) and KEM schemes, to the NIST PQC standardization call as a set of parameters with claimed post-quantum security from 143 to 274 (NIST security category of 1, 2, 3 and 5). Authors employ the Fujisaki-Okamoto and Dent transform [176] to obtain IND-CCA PKE and IND-CCA KEM schemes. In addition to power-of-two cyclotomic rings (`LIMA-2p`), authors propose safe-prime cyclotomics (`LIMA-sp`) that reduces the probability of subfield attacks by scarifying the efficiency compare to the power-of-two cyclotomics. In order to avoid decryption failure, authors perform rejection sampling (from a centered binomial distribution) at the encryption stage which makes the implementation to be non-constant time. In order to perform the polynomial multiplication with FFT, a large modulus should be selected for `LIMA-sp`.

Phong et al. [102] present LOTUS (**L**earning with err**O**rs based encryption with chosen ciphertex**T** for po**S**t quantum era) IND-CCA2 secure LWE-based PKE (LOTUS-PKE) and KEM (LOTUS-KEM) with 128-bit, 192-bit and 256-bit security (NIST security category of 1, 3 and 5). Knuth-Yao algorithm [83] is employed to sample the error term from the discrete Gaussian distribution. In order to reduce the sampling's overhead, DDG tree is built online and probability matrix is stored column-wised. Besides the reference and optimized implementations, vectorized implementation (employing AVX2 vector instructions) of LOTUS-PKE and LOTUS-KEM are provided.

NTRU-KEM [125], an IND-CCA2-secure KEM based on NTRU cryptosystem with 128-bit classical security, is the first timing attack resistant NTRU software thanks to its constant-time noise sampler. NTRU-based KEM has active security which allows parties to cache the ephemeral keys; however, passive secure key exchange mechanisms like NewHope and Kyber should not use cached values. Compared to NewHope (255-bit PQ security), NTRU-KEM (123-bit PQ security) improves (secret key size, public key size,ciphertext size) by (20%, 37%, 37%) and halves the required clock cycles for encryption/encapsulation step. However, it increases required clock cycles for the key generation and decryption/encapsulation by a factor of 3.47.

Plantard [108] presents Odd-Manhattan, a IND-CCA KEM by using Dent transform on IND-CPA PKE, at 126,192 and 256-bit security (NIST security category 1,3 and 5). Odd-Manhattan is based on the *α-Bounded Distance Parity Check (BDPCα)* [177], which impose a considerable increase in time and size of key generation, encryption, and decryption. To mitigate the timing overhead, computational reuse, i.e., store the results of the k consecutive additions (constant time) in memory, with notable memory penalty has been employed.

Garcia-Morchon et al. [109] introduce Round2, a family of CCA-PKE (Round2.PKE) and CPA-KEM (Round2.KEM) based on the general learning with rounding (GLWR) problem. By having d (dimension), n (system parameter), q (large modulus), p (rounding modulus) $\in \mathbb{Z}^+$ where $q \leq p$ and $n \in 1, d$, if $n = 1$, instantiated scheme is based on the LWR problem, while $n = d$ ($n + 1$ is prime) results in a RLWR based scheme. Round2 provides two set of parameters including Unified-Round2 (uRound2, q is a power of 2) and NTT-Round2 (nRound2, q is prime, $n = d$ ($n + 1$ is prime)). With uRound2, schemes can be seamlessly instantiated from LWR or RLWR [18] (for all NIST security levels), both $n = 1$ and $n = d$, with the same code which provides agility (i.e., switching from RLWR-based schemes to LWR-based schemes without recompilation). GLWR, compare to LWE, results in less random data generation due to avoiding the sampling from the non-uniform noise distribution; besides, the required bandwidth is reduced since fewer bits are needed per coefficient. Secret terms can be either a sparse-trinary (reduces the probability of error in decryption) or uniformly sampled in $\mathbb{Z}_q^d$. In order to have a unique implementing for LWR and RLWR, a common multiplier that implements polynomial multiplication as the matrix multiplication is employed. Compare to NewHope [40] and Kyber [41], RLWR-based uRound2 requires smaller the public-key and ciphertext in total for NIST security category 5. Over the same ring as NTRU-KEM scheme, Round2 bears better speedup due to its faster key generation. Performance evaluation of the reference implementation is performed on Intel Core i7 2.6GHz.

D'Anvers et al. [130] propose SABER, a family of Module-LWR based IND-CPA PKE and IND-CCA KEM schemes including LightSaber-KEM, Saber-KEM and FireSaber-KEM with 115, 180 and 245-bit security (NIST security category 1, 3 and 5). Integers are chosen to be the power-of-two modulus which results in avoiding explicit modular reduction and relaxing complicated sampling methods (e.g., rejection) by efficiently, constant time, sampling from a (modulo power 2) uniform distribution; however, power-of-two modulus prevents using NTT for the polynomial multiplication (Karatsuba and Toom-Cook algorithms are used instead). Switching among SABER schemes is accomplished by

choosing a modulus of higher rank in the fixed polynomial ring $\mathbb{Z}_{2^{13}}[x]/(x^{256}+1)$. The failure rate is reduced by using a reconciliation method introduced in [62].

Hamburg [91] introduces *ThreeBears*, a family of IND-CPA and IND-CCA KEM schemes adapted from Kyber [41] and based on integer Module-LWE (ILWE) [178] problem. *ThreeBears* includes *BabyBear*, *MamaBear* (recommended)and *PapaBear* with NIST security security category of 2, 4 and 5, respectively. For each scheme, deterministic CCA-secure (with FO transform) and ephemeral (without FO transform) implementations are presented. In order to reduce the memory footprint, the private key is rapidly generated by expanding a seed; similarly, public key and large public matrix (uniformly at random sample each element) are generated modulus N where N is a large Mersenne prime. Sampling the noise is performed by expanding a seed to only 1B per digit. Although the Saarinen's error correction [116, 117] can notably improve *ThreeBears* security, author prefers to use Melas BCH code as the two-error-correcting code to maintain the code simplicity. To preserve simplicity, NTT is not used which results in slower integer arithmetic, on devices without vector unit support in particular. Performance analysis of the *ThreeBears* implementations are provided on Intel Skylake, ARM Cortex-A8 and ARM Cortex-A53. With 15% smaller ciphertext and public key size, *MamaBear* (resp. *PapaBear*) is stronger than *Kyber-Paranoid* (resp. Hila5 [116, 117] and NewHope [40]).

Steinfeld et al. [70] present Titanium, a family of IND-CPA PKE (Titanium-CPA) and IND-CCA KEM (Titanium-CCA) schemes based on the middle product LWE (MPLWE) problem [92]. Schemes are tightly and provably secure based on the hardness of Polynomial-LWE problem over the polynomial ring $\mathbb{Z}[x]/f(x)$ where f is the member of a large group of ring polynomials. Titanium is a middle ground scheme that achieves a trade-off between security and efficiency such that, in terms of ciphertext size and performance, it is superior to *Frodo* [32] but inferior to *Kyber* [41]. Among 6 suggested parameter sets, *Std*128, *Med*160, *Hi*192 and *Super*256 satisfy minimum security specified by NIST (1,1,3 and 5, respectively). However, the security analysis of Titanium assumes the classical random oracle model. NTT and binomial difference error distribution are used for polynomial multiplication and error sampling; secret key coordinates are sampled uniformly at random over $\mathbb{Z}_q$. It should be mentioned that error correction or reconciliation techniques are not employed in Titanium. In addition to the reference and optimized implementation, authors provide performance analysis of vectorized implementation (AVX2) on Intel i7-7700K.

3.2.3 Digital Signature. Lyubashevsky [110] presents Short Integer Solution (SIS) problem based digital signature schemes adapted from Fiat-Shamir transformation. Lyubashevsky improves this scheme by establishing it efficiently for Ring-SIS and Ring-LWE which culminates in a smaller signature and key size [118]. BLISS signature [33] is an optimized version of Lyubashevsky's signature where a binomial Gaussian sampler is used in the rejection sampler component, resulting in a remarkable reduction in the standard deviation of the Gaussian distribution. Bai and Galbraith [94] propose a provably secure small signature (BG signature) based on the standard LWE and SIS problems that can be implemented using uniform distributions. Standard worst-case computational assumptions on lattices are the basis of security for the BG signature scheme.

Pöppelmann et al. [38, 141] evaluate implementations of various LBC schemes, including RLWE-based PKE schemes and BLISS [33] on the 8-bit AVR micro-controllers. They review various NTT algorithms (written in C) and optimize (using assembly language and ignoring zero coefficients) polynomial multiplication (column-wise) for ideal LBC schemes. However, using precomputed twiddle factors in NTT computations requires more memory footprint. Official release code of Keccak [179] for AVR is used for the random oracle which is needed for signing and verification. Other optimizations offered by the authors are removing bit reversal step during polynomial multiplication and applying the Gaussian sampling in a lazy manner. Compared to RLWE encryption, BLISS needs larger standard deviation for sampling from Gaussian distribution; candidates for Gaussian sampling

are CDT sampling [180] with binary search, which results in large tables, and Bernoulli [140], that impose remarkable performance overhead. Consequently, for Gaussian sampler, KL-convolution sampler [60] is used which consumes less flash memory compared with the CDT and Bernoulli samplers. The BLISS implementation consumes the least flash footprint on AVR and has the lowest published runtime by 2015.

BLISS-B [120] (with the same security level) improves the performance of original BLISS 2.8 times by employing the ternary representation of polynomials in order to shorten the length of the random numbers. During the key generation, keys are rejected which leads to 5-10 times reduction in the runtime of the key generation step. Generated signatures by BLISS and BLISS-B are compatible with each other, allowing signatures generated by one to be valid for the other. Although generated keys of BLISS-B can not be used in BLISS, BLISS generated keys are compatible with BLISS-B.

Oder et al. [139] present an efficient software implementation of BLISS on ARM Cortex-M4F to optimize the throughput along with minimizing memory footprint. Authors evaluate the efficiency of a variety of Gaussian samplers including the Bernoulli, Knuth-Yao, and Ziggurat [76]. In order to improve NTT computation, assembly level optimization along with precomputed coefficients are employed. They conclude that Knuth-Yao sampler is the best candidate for large devices, while for constrained devices Bernoulli is more favorable.

Güneysu et al. [138] present a highly optimized SIMD implementation of GLP signature [61] and implement it on Intel's Sandy and Ivy Bridge processors. In the proposed scheme, the Gaussian sampler is replaced with the uniform sampling from {-1,0,+1}; to benefit from the AVX, each 512 double-precision floating-point array of coefficients is 32-byte aligned. Besides, modular reduction of the coefficients is performed in a lazy manner. However, the signature size and security level of the implemented scheme are inferior to BLISS.

El Bansarkhani and Buchmann [119] implement the first software implementation (space and speed optimized) of GPV signature scheme [93] by employing the Micciancio and Peikert (MP) trapdoors [181] on the Sun XFire 4400 server equipped with 16 Quad-Core AMD Opteron. Besides the matrix version, a much faster Ring-LWE based variant of the scheme is provided which has around 3-6 and 3-9 times better speed than matrix version for sign and verification steps, respectively. Due to the small number of stored entries, instead of rejection sampling, the inversion transform method is used for discrete Gaussian sampling during integer key generation; however, rejection sampling [93] is used in the randomized rounding.

Dagdelen et al. [133] propose a fast software implementation of the BG signature, with optimized rejection sampling, on an Intel processor with AVX and an ARMv7 with Neon vector instructions support. Small performance degradation is observed by employing the standard lattices instead of ideal lattices which is a great achievement because there is no quasi-logarithmic arithmetic scheme like NTT for standard lattices.

Boorghany et al. [140, 180] propose efficient software implantation of lattice-based GLP and BLISS authentication protocols for resource-constrained smart cards and micro-controllers (ARM and AVR). Authors perform a design space exploration by choosing different parameter sets for FFT and Gaussian Sampler (Knuth-Yao and Bernoulli) along with the various PKE schemes. They conclude that LBC schemes are efficient enough to be implemented on the constrained devices.

Alkim et al. [111] introduce TESLA (a tightly secure signature in random oracle model resulted) by a tight reduction to LWE-based problems on the standard lattices and implement it on Intel Core-i7 4770K (Haswell). They adapt the design from BG signature [94] which is faster and smaller than the same scheme in [133] due to employing parallel matrix-vector multiplication and lazy reduction. Authors propose two variants TESLA-128 and TESLA-256; the former one, TELSA-I, is not quantum resistant, while the latter, TELSA-II, provides the first lattice-based digital signature with 128-bit security against quantum computers. Large public key size (about 1 MB) makes it impractical to

implement. Same authors present a fast, small, and provably secure Ring-LWE based software implementation of TESLA, that uses uniform sampling, on the same platform which reduces the key size about three orders of magnitude [142]. The propose Ring-TESLA benefits from the AVX2 instructions, which has a one cycle throughput for eight doubles integers. Instantiation from the Ring-TESLA leads to the rejection of valid signatures in the verification stage due to a problem in the parameter selection which is solved in [121] by new parameter selection method and in [51] by altering the algorithm. Based on the claims in [51], TESLA and Ring-TESLA use global parameters which results in employing a fixed lattice for all the signatures that could weaken the signature scheme. A recent version of TESLA [111] fixes the problem by adding a new condition to the signing step which results in dropping the speedup, creating a more complex signature scheme, with less success in signing. Barreto et al. [51] introduce TESLA#, a high-performance version of Ring-TESLA, on Intel Core i7-4770 Haswell processor, which resolves the security problems of TESLA. Further improvement is achieved by designing a more efficient Gaussian which accelerates the key generation step along with avoiding to store all 32 bits of coefficients of the polynomial. Bindel et al. [124] propose qTESLA a family of (provably existentially unforgeability under chosen-message attack (EUF-CMA) secure in the quantum random oracle model) Ring-LWE based digital signature schemes, including qTESLA-128, qTESLA-256 and qTESLA-192 with NIST's security categories of 1, 3 and 5. qTESLA adapts a simpler version [51] of the bimodal Gaussian sampler [33] that is only employed in key generation. Although qTESLA performs polynomial multiplication by NTT, it is compatible with the schoolbook algorithm. qTESLA uses cSHAKE [182] to deterministically generate the random bits for driving the seeds in the key generation and generation of a new polynomial for every key pair. In the signing step, qTESLA employs SHA-3 as the hash function and cSHAKE as the pseudo-random function. Contrary to the Ring-TESLA, qTESLA is secure against cache side channel attacks by applying countermeasures introduced in [183]; however, qTESLA is vulnerable to fault attacks [184] similar to Ring-TESLA.

BLZZRD [50], a lattice-based signature based on BLISS-B [120] is implemented on an Intel Core-i7 Haswell processor with small signature size but with costly signing step. Authors achieve optimal compression for discrete Gaussian distribution by using Binary Arithmetic Coding (BAC) which leads to a more compact signature compared to the advanced Huffman-based signature compressors. Further security improvement is gained by prohibiting leak of information about the execution time and power consumption of the arithmetic operation by applying randomization which makes the signature resistant to timing and power attacks. Masking property of Gaussian samples is achieved by randomizing and combining multiple numbers the of sample vectors.

Dilithium [131], a simple and efficient digital signature scheme resistant to the lattice reduction attacks (with the same conservative security parameter in [40]) is adapted from the designs in [61] and [94] that uses Fiat-Shamir Abort Framework [110] which is secure in random oracle model (no security proof in quantum random oracle model is presented). Authors implement Dilithium and its variant Dilithium-G on Intel Core-i7 4770k with comparable efficiency to BLISS. In [61], *hints* are generated (by the signer to help the verifier to verify the signature) to make the signature smaller; Dilithium improves the hint-generation and halves the public key size with less than 5% increase in the signature size. Authors set *total size* = *signature size* + *public key size* as their size parameter. Dilithium over ring of $Z_q[x]/[x^n + 1]$ ($n = 256, q = 2^{23} - 2^{13} + 1 = 8380417$) has slightly bigger *total size* than BLISS (over ring of $Z_q[x]/[x^{1024} + 1]$) with the same security level. Dilithium samples polynomial noises from the uniform distribution S_η in $[-\eta, +\eta]$ where η is in the range of [3-7] for very high secure to weakly secure scheme, respectively. However, Dilithium-G extract noises from Gaussian sampler which results in better security but vulnerable to timing attacks. Rejection sampling is the same for both schemes as if individual coefficients of a signature is not within a

certain range, signing procedure must be restarted. Dilithium employs the standard NTT-based polynomial multiplication, however in vectorized version, Dilithium uses integer instructions instead of floating point vector instructions [40].

Dilithium has been submitted to the NIST PQC standardization call at three security levels (all use the same ring) including *medium*, *recommended* and *very high* (NIST security category 1,2 and 3) [132]. Dilithium is tightly secure in the quantum random oracle model based on the *Module-LWE*, *Module-SIS* and *SelfTargetMSIS* [185] problem (adapted from combined security of *MSIS* problem and hash function H). The signature size of Dilithium is about 2× bigger than that of BLISS [33] and [186] (smallest schemes among lattice-based digital signatures) that use discrete Gaussian sampler which Dilithium avoids. However, compare to the most efficient lattice-based digital signature schemes that avoid Gaussian sampler, Dilithium achieves 2.5× smaller public key size. Coole-Tukey and Gentleman-Sande butterflies are used in NTT and inverse NTT, respectively, to perform the polynomial multiplication in which Montgomery reduction is used after multiplication (avoid reduction after addition and subtraction). Vectorized (AVX2 instruction set on Intel Core i7-4770K) version of NTT gives 4.5× speed improvement over the (reference) integer NTT implementation which is 2× faster than floating point NTT [40]. Dilithium uses SHAKE128 and SHAKE256 to drive the matrix ($A \in R_q^{k \times l}$ in NTT domain) and vectors. Vectorization improves speedup of the matrix and vector expansion by sampling four coefficients simultaneously.

Fouque et al. [123] propose FALCON, a family of compact lattice-based hash-and-sign digital signature schemes with quantum security of 103, 172 and 230 bits (NIST security category 1,3 and 5) with the primary goal of minimizing the *total size = signature size + public key size*. FALCON is the result of combining GPV framework [93], NTRU lattices [89] and Fast Fourier sampling [187]; provably secure NTRUSign is built by combining the GVP framework and NTRU lattices [188]. Instantiating the GPV IBE over NTRU lattices is presented in [186] which can be transformed to FALCON by employing the Fast Fourier sampling in private key operations. NTRU lattices along with the capability of the message recovery (entirely from the signature) result in the compactness of FALCON. Verification step in FALCON is relatively fast and simple and can be performed by a hash function followed by NTT operations. FALCON uses double precision floating-point arithmetic in signing which can be challenging to implement on the devices without floating point units. Another downside of the FALCON is the extensive use of the discrete Gaussian sampling over the integers which is hard to protect against the timing and side-channel attacks. FFT over the complex numbers is used for private key operations, while public key operations and key generation are performed using NTT over $\mathbb{Z}_q$. FALCON uses bimodal Gaussian in the reject in the sampler, ChaCha20 as the PRNG and SHAKE-256 as XOF for all security levels. FALCON can be easily transformed into an IBE scheme [186]. Authors only provide the performance evaluation of the reference implementation on an Intel Core i7-6567U CPU with 15% of error margin due to not disabling the boosting feature of the processor. Falcon has the smallest *total size* among all the post-quantum digital signature schemes at the same security level.

Chen et al. [74] present pqNTRUSign, a modular lattice-based hash-then-sign digital signature scheme (introduced in [189]) at post-quantum security of 149-bit (NIST security category 5). pqNTRUSign provides sampler agility as the user can choose the sampler based on the design goal; constant time uniform sampling for the security and (bimodal) Gaussian sampling for the performance goals, respectively. The key generation is the same for both set of parameters (Gaussian-1024 and Uniform-1024); other steps have different implementations for various parameter sets. The public key, forgery and transcript security are provided by NTRU assumption, LWE problem over NTRU lattices and rejection sampler, respectively. Like NTRUEncrypt [75], Box-Muller [73] is employed to sample

from the discrete Gaussian distribution. AVX optimization for polynomial multiplication is not included in pqNTRUSign; a naive NTT with the time complexity of $O(N^2/2)$ is used.

3.3 Hardware Implementations of Lattice-based Cryptographic Schemes

Nejatollahi et al. [146] propose the first domain-specific accelerators for ideal lattice-based schemes with the case study of BLISS-BI [120] and NewHope [40]. Authors present a quick design flow that performs exploration and the design of programmable accelerators that leads to on average 35% and 50% improvement in the latency and energy-delay product. Authors create a programmable accelerator for NTT that can be employed in any scheme, with any set of parameters, that uses Gentleman-Sande butterfly; the `Keccak-f[1600]` accelerator is suitable for any classical and post-quantum cryptographic scheme as the heart of the SHA3.

3.3.1 Public Key Encryption. Göttert et al. [56] propose the first hardware implementation of Ring-LWE based public key encryption on the Xilinx Virtex-7 FPGA. Due to the large area occupation of the full Ring-LWE public key encryption scheme, only LWE-polynomial variants are chosen to be implemented. Proposed implementations are based on LP lattice-based encryption scheme [46] which achieve 316 times higher throughput compared to software implementation. Using a fully parallel architecture (which makes the design remarkably spacious), high throughput is achieved by minimizing required clock cycles in computing the NTT. The primary optimization metric is the performance for which they show speedups for encryption and decryption schemes by factors of 200 and 70, respectively, in comparison to the software implementation with the same level of security.

Pöppelmann et al. [79] provide a flexible Ring-LWE encryption engine in which one core is used to perform the key generation, encryption, and decryption steps with the primary goal of optimizing throughput per unit of area. Besides, by applying optimizations including different encoding technique and removing some LSBs of the ciphertext coefficients, and encryption engine, it can be fit in the Xilinx Spartan-6 FPGA with three times slower encryption step. They employ a Gaussian sampler with relatively low precision by using the CDT sampling method that compares random probabilities with a cumulative distribution table. The proposed Gaussian Sampler is fast (one sampler per cycle at 60 MHz) with the cost of numerous random bits (85) to produce a single random number. The Gaussian sampler is time independent with the cost of an array of parallel comparators, one per each word of the table.

Roy et al. [37] implement a compact Ring-LWE crypto processor on Virtex 6 FPGA where they optimize NTT multiplication by reducing the fixed computation and pre-scaling overheads. Authors suggest to combine pre-computation stage and NTT computation. NTT Memory access is minimized by storing two coefficients in a single word, processing two pairs of coefficients together, and eliminating the idle cycles. Small LUTs are used in the Knuth-Yao discrete Gaussian sampler [85] which lead to more compact and faster sampler than [59].

Pöppelmann and Güneysu [59] implement the smallest lattice-based encryption engine on Spartan-6 and Virtex-5. Compared with the high-speed implementation of [79], this is one order of magnitude slower due to non-applicability of using NTT (using DSP-enabled schoolbook polynomial multiplier). Besides, a considerable area is saved by using specific modulus, a power of 2, by which modular reduction is almost cost-free. Further area saving is achieved by using a Bernoulli distribution [33] with small precomputed tables in order to optimize simple rejection sampling by eliminating computing of $exp()$ function.

Howe et al. [31] present the first and the only, hardware implementation of lattice-based encryption engine based on learning with error problem over standard lattices on the lightweight Spartan-6 FPGA. The main concern is optimizing the area, while the scheme maintains the balance between area and performance by using a larger Gaussian sampler. The proposed encryption engine is smaller

in comparison to the design of [56]; besides, it can closely compete with the encryption scheme of [79]. To maximize the performance, authors use a larger Gaussian sampler, Bernoulli sampler, which generates samples in parallel with no adverse effect on the critical path.

Reparaz et al. [135] implement a masked Ring-LWE scheme on a 32-bit ARM Cortex-M4F and Virtex-II FPGA which is Differential Power Analysis (DPA) resistant. In order to be first-order side-channel attack resilient, a constant time masked decoder with high success probability is implemented. The entire computation is done in the masked domain by employing a dedicated masked decoder which imposes considerable time and area overhead compared with an unprotected design.

3.3.2 Digital Signature. Howe et al. [34] provide evaluation and summary of practical instantiations of digital signature schemes based on lattice problems on different platforms. Evaluation metrics are the secret key, public key, and signature size. Additionally, they give a survey of various implementations of basic blocks including NTT and sampling. Authors evaluate Bernoulli, Ziggurat, Knuth-Yao, and cumulative distribution table (CDT) variants of the Gaussian sampler.

Güneysu et al. [61] implement an efficient lattice-based signature scheme (GLP signature) on a Xilinx Virtex 6 FPGA which is the first practical lattice-based signature scheme that could resist transcript collision attacks. Author removes the need for Gaussian noise sampling by using the rejection sampling which leads to hiding the secret key contained in each signature. Security of the proposed scheme is lower than standard lattice-based signatures due to building the hardness assumption based on the Decisional Compact Knapsack problem. Because of the regular structure of the schoolbook algorithm, authors achieve high speed and small size for implementation of the polynomial multiplier. Compared with BLISS, the proposed signature is sub-optimal in terms of signature size and security level.

Pöppelmann et al. [60] implement a high throughput hardware implementation of BLISS [33] on Xilinx Spartan-6 FPGA. Authors improve and parallelize the column-wise schoolbook multiplier presented in [61]. Authors employ an efficient CDT based Gaussian sampler with large tables. To improve the performance of the CDT sampler, author deploy a decreasing number of comparisons by improving the binary search and reducing the size of precomputed large tables by using an optimized floating-point representation (adaptive mantissa size) with negligible effect on the performance. Authors provide enhanced CDT which uses two smaller samples (Peikert convolution theorem [47]). A standard CDT needs table of the size at least $\eta \times \tau \times \lambda = 215.73 \times 13.4 \times 128 = 370kb$ while enhanced CDT needs around 23× smaller table. Authors evaluate performance and resource consumption of BLISS-I ($n = 512, q = 12289$) by employing the CDT and two parallel Bernoulli samplers and conclude that CDT consumes less FPGA resources than Bernoulli; besides, enhanced CDT achieves 17.4 Million Operation per Seconds (`MOPS`) which is 2.3× more than that of Bernoulli sampler. Based on the results, performance of enhanced CDT is almost the same for `BLISS-I` ($\sigma = 215$), `BLISS-III` ($\sigma = 250$) and `BLISS-IV` ($\sigma = 271$).

Güneysu et al. [144] propose an optimized and flexible implementation of a lattice-based digital signature on Xilinx Spartan-6 and Virtex-6 FPGAs which has the same theoretical basis as [61] but with major improvements. Compared with [61], instead of Schoolbook multiplier, authors employ a parallelized NTT for polynomial multiplications (the most time-consuming part of the digital signature), which leads to smaller and faster signing/verification engines. Authors develop a flexible processing core with VHDL that could be configured as either signing or/and verification engine which does not impose any overhead to provide flexibility. The signing step breaks into three separate blocks, including lattice processing engine, random oracle, and, sparse multiplication along with compression unit, which are running in parallel. The digital signature processor is based on the lattice processor for the public key encryption scheme in [79].

3.3.3 Key Exchange. Oder et al. [71] propose an area optimized constant time implementation of NewHope-Simple [62], on Xilinx Artix-7 FPGA, with decent performance level. With the same post-quantum security level as NewHope (128-bit), server and client work with clock frequency of 125 and 117 MHz, respectively. Authors design two separate modules for the client and server sides which forces an embedded system to be only either a server or a client, hence results in the lack of re-usability as a disadvantage. For the sake of the area optimization, 512 butterfly operations are performed serially, while they can be performed in parallel.

Kou et al. [147] provide a high performance pipelined implementation of NewHope [40] on Xilinx Artix-7 FPGA which is 19.1× (4 ×) faster (bigger) than the hardware implementation of the NewHope-Simple [71]. In order to improve the performance, NTT operations are computed using four butterfly units; besides, Longa-Naehrig modular reduction [39] is used instead of Barrett reduction.

3.4 Hardware/Software Implementations of Lattice-based Cryptographic Schemes

Because of the probabilistic inherent feature of rejection sampling in the lattice-based schemes, the probability of generation of an invalid signature exists that can be minimized by precomputation. Aysu et al. [149] divide the signature scheme in hash-based cryptographic signatures into two separate phases in order to minimize the energy and latency of signature generation. During the offline phase, input (message) independent computations, for instance, key and random number generation, are performed, and results are stored in a memory buffer as coupons. Subsequently, the output is generated using the precomputed coupons and the input (message) during the online phase. Employing the same idea, Aysu et al. [148] implement a latency optimized lattice-based signature with hardware/software co-design technique. The main objective is to optimize latency which is achieved by focusing on the signature generation step on the embedded device. On the other hand, the verification step is performed on high-performance platform servers. Signature generation scheme consists of two separate phases including offline and online phases which are performed on NIOS soft-core as software, and Altera Cyclone-IV FPGA as hardware, respectively. Hardware is responsible for low latency hash function and polynomial multiplication; however, the software part computes and stores polynomials.

3.5 DSP Implementation

In order to perform the multiply-accumulate (MAC) operations of matrices in the encryption scheme, Howe et al. [31] utilize a dedicated DSP48A1 unit of the Spartan-6 FPGA to achieve an area optimized hardware implementation of standard LWE based encryption engine. The primary goal is optimizing area, while the scheme maintains the balance between area and performance by using a larger Gaussian sampler.

4 CONCLUSION

Lattice-based cryptographic algorithms and protocols promise to tackle the challenges posed by deployment across diverse computing platforms, as well as for diverse use cases within reasonable security, performance, and energy efficiency guarantees.

Numerous schemes and implementations tackle different trade-offs, such as memory footprint, security, performance, and energy, are mapped on a variety of platforms and apply to specific use cases. However, current designs are still deficient in addressing the need for agility, which is paramount to tackle the needs of emerging business models at the computing platform level. Besides, securing such platforms against physical attacks is a topic that needs to be researched.

In this manuscript, we provided a review of lattice-based cryptography, some of the proposals for lattices in computer security, their implementations in software and hardware, and their applications to key exchange/encapsulation and digital signatures.

5 ACKNOWLEDGEMENT

This work was supported in part with a gift from Qualcomm Technology Inc.

REFERENCES

[1] H. Nejatollahi et al. Software and hardware implementation of lattice-cased cryptography schemes. *University of California Irvine, CECS TR 17-04*, 2017.
[2] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 1997.
[3] G.-L. Long. Grover algorithm with zero theoretical failure rate. *Physical Review A*, 2001.
[4] A. Ansarmohammadi et al. Fast and area efficient implementation for chaotic image encryption algorithms. In *CADS*. 2015.
[5] A. Ansarmohammadi et al. A low-cost implementation of aes accelerator using hw/sw co-design technique. In *CADS*. 2013.
[6] O. Garcia-Morchon et al. Dtls-himmo: Achieving dtls certificate security with symmetric key overhead. In *ESORICS*. 2015.
[7] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. 2005.
[8] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*. 1996.
[9] D. Micciancio et al. *Lattice-based cryptography*. 2009.
[10] M. Ajtai et al. A sieve algorithm for the shortest lattice vector problem. In *STOC*. 2001.
[11] D. Micciancio et al. Faster exponential time algorithms for the shortest vector problem. In *SODA*. 2010.
[12] D. Aggarwal et al. Solving the shortest vector problem in 2n time using discrete gaussian sampling: Extended abstract. In *STOC*. 2015.
[13] D. Micciancio. *Cryptographic Functions from Worst-Case Complexity Assumptions*. 2010.
[14] D. Stehlé et al. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*. 2009.
[15] Z. Brakerski et al. Classical hardness of learning with errors. In *STOC*. 2013.
[16] B. Applebaum et al. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*. 2009.
[17] V. Lyubashevsky et al. On ideal lattices and learning with errors over rings. In *EUROCRYPT'10*. 2010.
[18] A. Banerjee et al. Pseudorandom functions and lattices. In *Proceedings of the Annual International Conference on Theory and Applications of Cryptographic Techniques*. 2012.
[19] A. Langlois et al. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive, 2012.
[20] H. Nejatollahi et al. Trends, challenges and needs for lattice-based cryptography implementations: Special session. In *CODES*. 2017.
[21] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. 1997.
[22] S. Cook et al. On the minimum computation time of functions. *Ph.D. dissertation, Harvard University*, 1969.
[23] A. Karatsuba et al. Multiplication of many-digital numbers by automatic computers. In *USSR Academy of Sciences*. 1963.
[24] A. Schönhage et al. Schnelle multiplikation grosser zahlen. *Computing*, 1971.
[25] M. Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 2009.
[26] H. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *TASSP*, 1980.
[27] J. W. Cooley et al. An algorithm for the machine calculation of complex journal = Mathematics of Computation, fourier booktitle. 1965.
[28] W. M. Gentleman et al. Fast fourier transforms: For fun and profit. In *AFIPS '66*. 1966.
[29] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 1985.
[30] P. Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *CRYPTO*. 1986.
[31] J. Howe et al. Lattice-based encryption over standard lattices in hardware. In *DAC*. 2016.
[32] J. Bos et al. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *CCS*. 2016.
[33] L. Ducas et al. Lattice signatures and bimodal gaussians. In *CRYPTO*. 2013.
[34] J. Howe et al. Practical lattice-based digital signature schemes. *TECS*, 2015.
[35] T. Oder et al. Lattice-based cryptography: From reconfigurable hardware to asic. In *ISIC*. 2016.
[36] F. Winkler. Polynomial algorithms in computer algebra. In *TMSC*. 1996.
[37] S. S. Roy et al. Compact ring-lwe cryptoprocessor. In *CHES'14*. 2014.
[38] T. Pöppelmann et al. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In *LATINCRYPT*. 2015.
[39] P. Longa et al. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. Cryptology ePrint Archive, 2016.

[40] E. Alkim et al. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, 2015.
[41] J. Bos et al. Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, 2017.
[42] J. P. David et al. Hardware complexity of modular multiplication and exponentiation. *TC*, 2007.
[43] D. D. Chen et al. Parameter space for the architecture of fft-based montgomery modular multiplication. *TC*, 2016.
[44] C. Rafferty et al. Evaluation of large integer multiplication methods on hardware. *TC*, 2017.
[45] P. G. Comba. Exponentiation cryptosystems on the ibm pc. *IBM systems journal*, 1990.
[46] R. Lindner et al. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA'11*. 2011.
[47] C. Peikert. An efficient and parallel gaussian sampler for lattices. In *CRYPTO'10*. 2010.
[48] S. Bai et al. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In *ASIACRYPT*. 2015.
[49] M.-J. O. Saarinen. Gaussian sampling precision in lattice cryptography. Cryptology ePrint Archive, 2015.
[50] M.-J. O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 2017.
[51] P. S. L. M. Barreto et al. Sharper ring-lwe signatures. Cryptology ePrint Archive, 2016.
[52] J. Howe et al. On practical discrete gaussian samplers for lattice-based cryptography. *TC*, 2016.
[53] D. Micciancio et al. Gaussian sampling over the integers: Efficient, generic, constant-time. Cryptology ePrint Archive, 2017.
[54] J. Folláth. Gaussian sampling in lattice based cryptography. *Tatra Mountains Mathematical Publications*, 2014.
[55] J. Von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards Applied Mathematics booktitle*, 1951.
[56] N. Göttert et al. On the design of hardware building blocks for modern lattice-based encryption schemes. In *CHES*. 2012.
[57] L. Ducas et al. Faster gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT*. 2012.
[58] T. Pöppelmann. *Efficient Implementation of Ideal Lattice-Based Cryptography*. Ruhr-Universität Bochum, 2016.
[59] T. Pöppelmann et al. Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *ISCAS*. 2014.
[60] T. Pöppelmann et al. Enhanced lattice-based signatures on reconfigurable hardware. In *CHES*. 2014.
[61] T. Güneysu et al. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES*. 2012.
[62] E. Alkim et al. Newhope without reconciliation. Cryptology ePrint Archive, 2016.
[63] E. Alkim et al. Newhope on arm cortex-m. In *SPACE*. 2016.
[64] S. Streit et al. Post-quantum key exchange on armv8-a – a new hope for neon made simple. Cryptology ePrint Archive, 2017.
[65] T. Poppelmann et al. Newhope. Technical report, National Institute of Standards and Technology, 2017.
[66] M.-J. O. Saarinen. Hila5. Technical report, National Institute of Standards and Technology, 2017.
[67] X. Lu et al. Lac. Technical report, National Institute of Standards and Technology, 2017.
[68] N. P. Smart et al. Lima. Technical report, National Institute of Standards and Technology, 2017.
[69] R. Avanzi et al. Crystals-kyber. Technical report, National Institute of Standards and Technology, 2017.
[70] R. Steinfeld et al. Titanium. Technical report, National Institute of Standards and Technology, 2017. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.
[71] O. Tobias et al. Implementing the newhope-simple key exchange on low-cost fpgas. In *LATINCRYPT*. 2017.
[72] G. Marsaglia et al. The ziggurat method for generating random variables. *Journal of statistical software*, 2000.
[73] G. E. Box et al. A note on the generation of random normal deviates. *The annals of mathematical statistics*, 1958.
[74] C. Chen et al. pqntrusign: A modular lattice signature scheme. Technical report, National Institute of Standards and Technology, 2017.
[75] Z. Zhang et al. Ntruencrypt. Technical report, National Institute of Standards and Technology, 2017.
[76] J. Buchmann et al. Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers. In *SAC*. 2013.
[77] D. B. Thomas et al. Gaussian random number generators. *ACM CSUR*, 2007.
[78] T. Pöppelmann et al. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In *LATINCRYPT*. 2012.
[79] T. Pöppelmann et al. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *SAC*. 2013.
[80] C. Du et al. Towards efficient discrete gaussian sampling for lattice-based cryptography. In *FPL*. 2015.
[81] C. Du et al. High-performance software implementation of discrete gaussian sampling for lattice-based cryptography. In *ITNEACC*. 2016.
[82] A. Khalid et al. Time-independent discrete gaussian sampling for post-quantum cryptography. In *FPT*. 2016.
[83] D. E. Knuth et al. The complexity of nonuniform random number generation. *Algorithms and complexity: new directions and recent results*, 1976.
[84] R. de Clercq et al. Efficient software implementation of ring-lwe encryption. In *DATE*. 2015.

[85] S. S. Roy et al. High precision discrete gaussian sampling on fpgas. In *SAC*. 2013.
[86] S. S. Roy et al. Compact and side channel secure discrete gaussian sampling. Cryptology ePrint Archive, 2014.
[87] D. Micciancio et al. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 2007.
[88] C. Peikert et al. A framework for efficient and composable oblivious transfer. In *CRYPTO*. 2008.
[89] J. Hoffstein et al. Ntru: A ring-based public key cryptosystem. In *ANTS-III*. 1998.
[90] D. Stehlé et al. Making ntru as secure as worst-case problems over ideal lattices. In *EUROCRYPT*. 2011.
[91] M. Hamburg. Three bears. Technical report, National Institute of Standards and Technology, 2017.
[92] M. Rosca et al. Middle-product learning with errors. Cryptology ePrint Archive, 2017.
[93] C. Gentry et al. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*. 2008.
[94] S. Bai et al. An improved compression technique for signatures based on learning with errors. In *CT-RSA*. 2014.
[95] Public-key cryptosystems from lattice reduction problems. In *CRYPTO*. 1997.
[96] J. Hoffstein et al. Ntrusign: Digital signatures using the ntru lattice. In *CT-RSA*. 2003.
[97] J. Ding et al. A simple provably secure key exchange scheme based on the learning with errors problem. 2012.
[98] J. W. Bos et al. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *SP*. 2015.
[99] E. Fujisaki et al. How to enhance the security of public-key encryption at minimum cost. In *PKC*. 1999.
[100] D. Hofheinz et al. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, 2017.
[101] Nist: National institute for standards and technology. postquantum crypto project. 2017.
[102] L. T. Phong et al. Lotus. Technical report, National Institute of Standards and Technology, 2017.
[103] J. H. Cheon et al. Lizard: Cut off the tail! practical post-quantum public-key encryption from lwe and lwr. Cryptology ePrint Archive, 2016.
[104] J. H. Cheon et al. Lizard. Technical report, National Institute of Standards and Technology, 2017.
[105] J. Hoffstein et al. Choosing parameters for ntruencrypt. In *CT-RSA*. 2017.
[106] M. Seo et al. Emblem and r.emblem. Technical report, National Institute of Standards and Technology, 2017.
[107] M. Naehrig et al. Frodokem. Technical report, National Institute of Standards and Technology, 2017.
[108] T. Plantard. Odd manhattan. Technical report, National Institute of Standards and Technology, 2017.
[109] O. Garcia-Morchon et al. Round2. Technical report, National Institute of Standards and Technology, 2017.
[110] V. Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*. 2009.
[111] E. Alkim et al. Revisiting tesla in the quantum random oracle model. Cryptology ePrint Archive, 2015.
[112] S. Bhattacharya et al. spkex: An optimized lattice-based key exchange. Cryptology ePrint Archive, 2017.
[113] Z. Jin et al. Optimal key consensus in presence of noise. Cryptology ePrint Archive, 2017.
[114] Y. Zhao et al. A modular and systematic approach to key establishment and public-key encryption based on lwe and its variants. Technical report, National Institute of Standards and Technology, 2017.
[115] D. J. Bernstein et al. Ntru prime: Reducing attack surface at low cost. Cryptology ePrint Archive, 2016.
[116] M.-J. O. Saarinen. Ring-lwe ciphertext compression and error correction: Tools for lightweight post-quantum cryptography. In *IoTPTS*. 2017.
[117] M.-J. O. Saarinen. Hila5: On reliability, reconciliation, and error correction for ring-lwe encryption. Cryptology ePrint Archive, 2017.
[118] V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*. 2012.
[119] R. El Bansarkhani et al. Improvement and efficient implementation of a lattice-based signature scheme. In *SAC*. 2013.
[120] L. Ducas. Accelerating bliss: The geometry of ternary polynomials. Cryptology ePrint Archive, 2014.
[121] A. Chopra. Improved parameters for the ring-tesla digital signature scheme. Cryptology ePrint Archive, 2016.
[122] A. Chopra. Glyph: A new insantiation of the glp digital signature scheme. Cryptology ePrint Archive, 2017.
[123] P.-A. Fouque et al. Falcon: Fast-fourier lattice-based compact signatures over ntru. Technical report, National Institute of Standards and Technology, 2017.
[124] N. Bindel et al. qtesla. Technical report, National Institute of Standards and Technology, 2017.
[125] A. Hülsing et al. High-speed key encapsulation from ntru. In *CHES*. 2017.
[126] J. Ding et al. Ding key exchange. Technical report, National Institute of Standards and Technology, 2017.
[127] A. Hülsing et al. Ntru-hrss-kem. Technical report, National Institute of Standards and Technology, 2017.
[128] D. J. Bernstein et al. Ntru prime. Technical report, National Institute of Standards and Technology, 2017.
[129] R. E. Bansarkhani. Kindi. Technical report, National Institute of Standards and Technology, 2017.
[130] J.-P. D'Anvers et al. Saber: Mod-lwr based kem. Technical report, National Institute of Standards and Technology, 2017.
[131] L. Ducas et al. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, 2017.
[132] L. Ducas et al. Crystals-dilithium. Technical report, National Institute of Standards and Technology, 2017.
[133] Ö. Dagdelen et al. High-speed signatures from standard lattices. In *LATINCRYPT*. 2014.
[134] Z. Liu et al. Efficient ring-lwe encryption on 8-bit avr processors. 2015.
[135] O. Reparaz et al. Masking ring-lwe. *Journal of Cryptographic Engineering*, 2016.

[136] J. Buchmann et al. High-performance and lightweight lattice-based public-key encryption. In *IoTPTS*. 2016.
[137] Y. Yuan et al. Portable implementation of lattice-based cryptography using javascript. In *CANDAR*. 2016.
[138] T. Güneysu et al. Software speed records for lattice-based signatures. In *PQCrypto*. 2013.
[139] T. Oder et al. Beyond ecdsa and rsa: Lattice-based digital signatures on constrained devices. In *DAC*. 2014.
[140] A. Boorghany et al. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. 2015.
[141] T. Pöppelmann et al. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. Cryptology ePrint Archive, 2015.
[142] S. Akleylek et al. An efficient lattice-based signature scheme with provably secure instantiation. In *AFRICACRYPT*. 2016.
[143] S. Gueron et al. Speeding up r-lwe post-quantum key exchange. Cryptology ePrint Archive, 2016.
[144] T. Güneysu et al. Lattice-sased signatures: Optimization and implementation on reconfigurable hardware. 2015.
[145] J. Howe et al. Compact and provably secure lattice-based signatures in hardware. 2017.
[146] H. Nejatollahi et al. Domain-specific accelerators for ideal lattice-based public key protocols. Cryptology ePrint Archive, Report 2018/608, 2018.
[147] P.-C. Kuo et al. High performance post-quantum key exchange on fpgas. 2017.
[148] A. Aysu et al. The future of real-time security: Latency-optimized lattice-based digital signatures. 2015.
[149] A. Aysu et al. Precomputation methods for hash-based signatures on energy-harvesting platforms. *TC*, 2016.
[150] J. Hoffstein et al. A signature scheme from learning with truncation. Cryptology ePrint Archive, 2017.
[151] N. C. Dwarakanath et al. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 2014.
[152] S. More et al. Discrete gaussian sampling for low-power devices. In *PACRIM*. 2015.
[153] P. Emeliyanenko. Efficient multiplication of polynomials on graphics hardware. In *APPT*. 2009.
[154] V. Shoup. Ntl: a library for doing number theory. 2016.
[155] S. Akleylek et al. On the efficiency of polynomial multiplication for lattice-based cryptography on gpus using cuda. In *ICCISB*. 2015.
[156] S. Akleylek et al. Sparse polynomial multiplication for lattice-based cryptography with small complexity. *The Journal of Supercomputing*, 2016.
[157] C. Aguilar-Melchor et al. Nfllib: Ntt-based fast lattice library. In *CT-RSA*. 2016.
[158] P. Longa et al. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *CANS*. 2016.
[159] A. Aysu et al. Low-cost and area-efficient fpga implementations of lattice-based cryptography. In *HOST*. 2013.
[160] D. D. Chen et al. High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems. *TCS*, 2015.
[161] C. Du et al. A family of scalable polynomial multiplier architectures for ring-lwe based cryptosystems. 2016.
[162] T. Györfi et al. Implementing modular ffts in fpgas – a basic block for lattice-based cryptography. In *DSD*. 2013.
[163] C. Du et al. Towards efficient polynomial multiplication for battice-based cryptography. In *ISCAS*. 2016.
[164] C. Du et al. Efficient polynomial multiplier architecture for ring-lwe based public key cryptosystems. In *ISCAS*. 2016.
[165] C. Du et al. High-speed polynomial multiplier architecture for ring-lwe based public key cryptosystems. In *GLSVLSI*. 2016.
[166] N. Howgrave-Graham et al. Naep: Provable security in the presence of decryption failures. Cryptology ePrint Archive, 2003.
[167] D. J. Bernstein. New stream cipher designs. chapter The Salsa20 Family of Stream Ciphers. 2008.
[168] Ieee standard specification for public key cryptographic techniques based on hard problems over lattices. *IEEE Std 1363.1-2008*, 2009.
[169] E. B. Rachid. Lara - a design concept for lattice-based encryption. Cryptology ePrint Archive, 2017.
[170] C. Peikert. Lattice cryptography for the internet. In *PQCrypto*. 2014.
[171] S. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. 2016.
[172] M. Braithwaite. Experimenting with post-quantum cryptography. 2016.
[173] D. J. Bernstein. Chacha, a variant of salsa20. In *SASC*. 2008.
[174] M. J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.
[175] D. Stebila et al. Post-quantum key exchange for the internet and the open quantum safe project. Cryptology ePrint Archive, 2016.
[176] A. W. Dent. A designer's guide to kems. In K. G. Paterson (editor), *Cryptography and Coding*. 2003.
[177] V. Lyubashevsky et al. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *CRYPTO*. 2009.
[178] G. Chunsheng. Integer version of ring-lwe and its applications. Cryptology ePrint Archive, 2017.
[179] G. Bertoni et al. Keccak. In *EUROCRYPT*. 2013.
[180] A. Boorghany et al. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. Cryptology ePrint Archive, 2014.

[181] D. Micciancio et al. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*. 2012.
[182] J. Kelsey. Sha-3 derived functions: cshake, kmac, tuplehash, and parallelhash. *NIST special publication*, 2016.
[183] N. Bindel et al. Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. Cryptology ePrint Archive, 2017.
[184] N. Bindel et al. Lattice-based signature schemes and their sensitivity to fault attacks. 2016.
[185] E. Kiltz et al. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. Cryptology ePrint Archive, 2017.
[186] L. Ducas et al. Efficient identity-based encryption over ntru lattices. In *ASIACRYPT*. 2014.
[187] L. Ducas et al. Fast fourier orthogonalization. In *ISSAC*. 2016.
[188] D. Stehlé et al. Making ntru as secure as worst-case problems over ideal lattices. In *EUROCRYPT*. 2011.
[189] J. Hoffstein et al. Transcript secure signatures based on modular lattices. In *PQCrypto*. 2014.