Check for updates

ACM DIGITAL LIBRARY | Association for Computing Machinery | acm open

Latest updates: https://dl.acm.org/doi/10.1145/3292548

PDF Download
3292548.pdf
16 February 2026
Total Citations: 227
Total Downloads: 6193

Published: 28 January 2019
Accepted: 01 November 2018
Revised: 01 October 2018
Received: 01 November 2017

Citation in BibTeX format

SURVEY

Post-Quantum Lattice-Based Cryptography Implementations: A Survey

HAMID NEJATOLLAHI, University of California, Irvine, Irvine, CA, United States

NIKIL D. DUTT, University of California, Irvine, Irvine, CA, United States

SANDIP RAY, University of Florida, Gainesville, FL, United States

FRANCESCO REGAZZONI

INDRANIL BANERJEE, Qualcomm Incorporated, San Diego, CA, United States

ROSARIO CAMMAROTA, Qualcomm Incorporated, San Diego, CA, United States

Open Access Support provided by:

University of California, Irvine

Qualcomm Incorporated

University of Florida

Post-Quantum Lattice-Based Cryptography Implementations: A Survey

HAMID NEJATOLLAHI and NIKIL DUTT, University of California Irvine
SANDIP RAY, University of Florida
FRANCESCO REGAZZONI, ALaRi
INDRANIL BANERJEE and 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 in its application to both traditional and emerging security problems such as encryption, digital signature, key exchange, and homomorphic encryption. While such techniques provide guarantees, in theory, their realization on contemporary computing platforms requires careful design choices and tradeoffs 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 har dware.

CCS Concepts: • **Security and privacy → Public key (asymmetric) techniques**; *Hardware-based security protocols*;

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. 2019. Post-Quantum Lattice-Based Cryptography Implementations: A Survey. *ACM Comput. Surv.* 51, 6, Article 129 (January 2019), 41 pages.
https://doi.org/10.1145/3292548

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: H. Nejatollahi, University of California Irvine, Irvine, California, 92697-3435; email: hnejatol@uci.edu; N. Dutt, University of California Irvine, Irvine, California, 92697-3435; email: dutt@ics.uci.edu; S. Ray, University of Florida; email: sandip@ece.ufl.edu; F. Regazzoni, ALaRi; email: regazzoni@alari.ch; I. Banerjee, Qualcomm Technologies Inc. San Diego, CA, 92121-1714; email: ibanerje@qti.qualcomm.com; R. Cammarota, Qualcomm Technologies Inc. San Diego, CA, 92121-1714; email: ro.c@qti.qualcomm.com.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
0360-0300/2019/01-ART129 $15.00
https://doi.org/10.1145/3292548

1 INTRODUCTION

Advances in computing efficiency steadily erode computer security at its foundation, and this enables prospective attackers to use ever more powerful computing systems and the best cryptanalysis algorithms to increase attack speed. One aspect of this arises from Moore's Law, that predicts that traditional computing systems will 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 to 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. 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 the 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 capabilities ranging from high-performance (real-time) virtualized environments, such as cloud computing resources and software-defined networks, to highly resource-constrained Internet of Things (IoT) platforms to realize the vision of the 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 from cryptographic implementations. This calls for the development of programmable hardware capable of efficiently running not only individual cryptographic algorithms, but full protocols, 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 preceding examples, to name a few, form a compelling argument for innovation in the computer security space, including and beyond the foundations, down to the actual implementation and deployment of primitives and protocols to satisfy 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

[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/.

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. IoT ecosystems) as well as for diverse use cases, including the ability to perform computation on encrypted data by providing strong (much better understood than before) foundations for protocols based on asymmetric key cryptography against powerful attackers (using quantum computers and algorithms) and to offer protection beyond the span of traditional cryptography. Indeed, LBC 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 article, we review the foundations of LBC, 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 remainder of the article is organized as follows. In Section 2, we provide the relevant background to make the article 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 $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n \in \mathbb{R}^{d \times n}$ to be n-linearly independent vectors in $\mathbb{R}^d$. B is a $d \times n$ matrix in which ith column is $\mathbf{b}_i$ vector such that $B = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n]$. 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) form the lattice $\mathcal{L}$:

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

2.2 Ideal Lattices

An ideal lattice is defined over a ring $R = \mathbb{Z}_q[x]/(f(x))$, where $f(x) = x^n + f_n x^{n-1} + \cdots + f_1 \in Z[x]$ (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 than or equal to $n - 1$. 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 [SVP]) in the average case is as hard as solving the worst-case assumption. In order to solve the SVP problem, one should solve the problem for any input basis B (all instances of the SVP

problem should be solved). It is conjectured that there is no probabilistic polynomial time algorithm that can approximate certain computational problems on lattices to within polynomial factors [9]. This is the basis for the security of LBC schemes. The fastest algorithm to solve the SVP problem has the time and memory complexity of $2^{O(n)}$ [10–12]. We take the following definitions from Aggarwal et al. [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 SVP (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 SVP ($G_{AP}SVP_\gamma$) 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 CVP (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 (SIVP) is defined as finding n linearly independent lattice vectors $\{v = v_1, \ldots, v_n : v_i \in \mathcal{L}(B) for 1 \leq i \leq n\}$ that minimize $\|v\| = \max_i \|v_i\|$ [13]. Given an approximate factor $\gamma \geq 1$, the $\gamma - approximate$ ($SIVP_\gamma$) is defined as finding n-linearly independent vectors lattice vectors $\{v = v_1, \ldots, v_n : v_i \in \mathcal{L}(B)$ such that $\max_i \|v_i\| \leq \lambda_n(\mathcal{L}(B)$ where λ_n denotes the nth success minima. For an n-dimensional lattice, λ_i, ith 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 $a_1, a_2, \ldots, a_n \in \mathbb{Z}^{m \times n}$ 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 $x_1.a_1 + x_2.a_2 + \cdots + x_n.a_n = 0 \pmod q$. Short usually translates as $z_i \in \{-1, 0, +1\}$. Considering q-ary lattices, let $A = (a_1, a_2, \ldots, a_n)$ be a vector in $\mathbb{Z}^{m \times n}$ and $\Lambda_q^\perp(A) = \{z \in \mathbb{Z}^m : Az = 0 \pmod q\}$; the SIS problem is to find the shortest vector problem for the lattice $\Lambda_q^\perp$. Based on Ajtai's theorem, if polynomial time algorithm A solves the SIS problem, an efficient algorithm B exists that can solve the SVP (or SVIP) problem for any lattice of dimension n in polynomial time.

Ring-SIS. Let $A = (a_1, a_2, \ldots, a_n)$ be a vector with $a_i \in \mathbb{Z}_q[x]/(x^n + 1)$ and $\Lambda_q^\perp(A) = \{z \in \mathbb{Z}_q[x]/(x^n + 1) : Az = 0 \pmod q\}$; the 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 LWE) problem where e is the error that is sampled from

error distribution χ. The worst-case hardness assumption of the LWE problem is proved under the quantum [7] and classical [15] reductions. In addition, if secret s is sampled from the same error distribution as e, the hardness assumption of the LWE problem is still valid [16].

Ring-LWE. Let $R_q = \mathbb{Z}_q[x]/(x^n + 1)$ 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 the Ring-LWE problem, where $e \in R$ is the error with coefficients sampled from error distribution χ [17].

Learning with Rounding (LWR). The complexity and inefficiency of the decisional (LWE) problem prohibit its use for PRGs. The 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 conceals the lower order bits with rounding. Let a_i be sampled uniformly at random in Z_q^n and $\lfloor \cdot \rceil : Z_q \to Z_p$ 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 the 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 the 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 improves the security of the ideal lattices. With the same arithmetic foundation, *Module-LWE/Module-LWR* provides a tradeoff between security and cost (computation and bandwidth).

2.4 Arithmetic and Components of Lattices

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

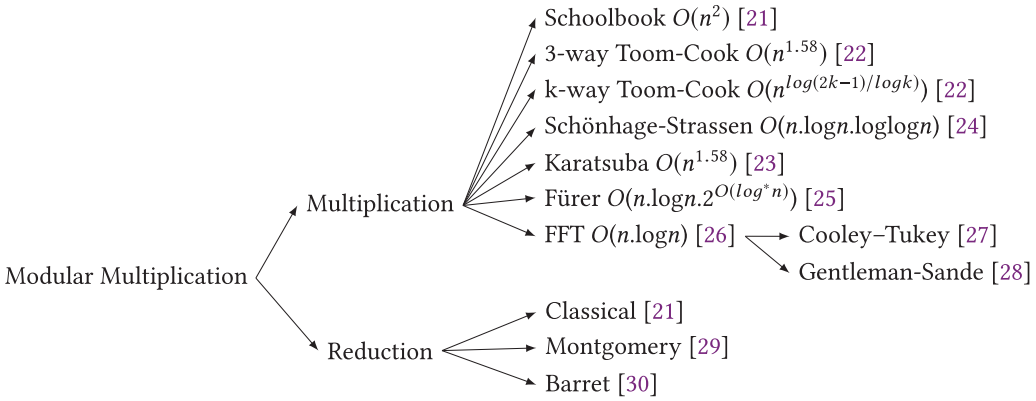

Fig. 1. Common modular multiplication algorithms.

Two main classes of lattice-based algorithms used in cryptography are NTRU and LWE. The security of NTRU is based on hardness not-provably reducible to solving the CVP in a lattice, whereas the security of LWE relies on provably reducible solving the 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 problems are resistant to quantum attacks, while their relatively inefficient nature led researchers to devise more efficient formulations (e.g., over rings, as in Ring-LWE).

Implementations are broadly classified into pure software, pure hardware, and hardware/software co-design cryptographic engines [20]. Implementation of 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 a large memory footprint due to the large key size—hundreds of kilobytes 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 the memory footprint can be traded-off with security assurance, which improves both efficiency and memory consumption.

Practical software implementations of standard lattices, encryption schemes [31], and key exchanges [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 the ASIC design of LBC are presented in Oder et al. [35].

2.4.1 The Multiplier Component. Matrix multiplication is used for 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 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 is defined as $R_p = Z_p/[x^{n+1}]$ in which coefficients of polynomials are in [0,p).

The 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 nth 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 $(a_0, \ldots, a_{n-1})$ as coefficients, we would have $(\hat{a_0}, \ldots, \hat{a_{n-1}}) = NTT(a_0, \ldots, a_{n-1})$ where

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

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

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

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

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

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

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

Two common algorithms to compute NTT are the Cooley-Tukey (CT) butterfly [27] and Gentleman-Sande (GS) butterfly [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 bit-reverse calculation [40]; however, the bit-reverse step can be avoided by using CT for NTT and GS for NTT^{-1} [39, 41].

Other popular multiplication algorithms in the 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 sizes, are presented in David et al. [42], in which the authors calculate the 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. The Karatsuba multiplier outperforms for operands greater than or equal to 32 bits. Schoolbook imposes a large memory footprint in order to store partial products, which negatively impact performance; this 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 LBC (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 of from 64 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, $\sigma = 3.33$ [46] is suggested. Digital signature sampling from a Gaussian sampler involves a large $\sigma = 215$ [33]. However, by employing Peikert's convolution lemma [47], the standard deviation can be reduced by an order of magnitude, which is a remarkable improvement on the precomputed table size. Speed and memory footprint are 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 $x = +\infty$ (considering only the positive side due to 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, the Rényi divergence technique [48] can be employed to measure the distance between two probability distributions (e.g., [40]). More precisely, in LBC, 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 a 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

Table 1. Comparison of Different Gaussian Samplers; Partially Extracted from Ducas et al. [33]

Sampler	Speed	FP exp()	Table Size	Table Lookup	Entropy	Features
Rejection	slow	10	0	0	45+10$log_2\sigma$	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

Gaussian sampler; hence, a λ-bit uniform random integer is used to approximate the discrete sampler. Fortunately, it is proved that the Gaussian sampler could be used to achieve a $\lambda/2$ security level (approximated sampler) instead of a 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 the 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 LBC schemes with a more mathematical outlook [54]. Gaussian samples are classified into six categories, and guidelines are provided for choosing the best candidate on different platforms for specific parameter ranges. However, we organize Gaussian samplers into the types discussed later. A summary of advantage and disadvantages of each sampler is given in Table 1.

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

Bernoulli Sampler. Bernoulli is an optimized version of rejection sampling that reduces the average number of required attempts for a successful sampler from 10 to around 1.47 with no need to calculate the `exp()` function or precomputed tables [58]. Bernoulli is introduced in Ducas et al. [33] for LBC 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 a Bernoulli sampler is shown here in five 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 the 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 $\sigma_2 = \sqrt{\frac{1}{2ln2}} \approx 0.849$ and 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], whereas 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]). The 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 [60]. Precomputed tables in Bernoulli sampling are small; binary Gaussian distribution (easy to sample intermediate sampler) is independent of σ; hence, the Peikert convolution lemma (smaller σ) does not have a considerable impact on the area. However, the convolution lemma reduces the area of precomputed tables in the 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 a 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 outputting $\sum_{i=1}^{k}(a_i, b_i)$ 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. A binomial sampler has been employed inside software implementations of NewHope [62–65], HILA5 [66], LAC [67], LIMA [68], Kyber [41, 69], and Titanium [70]. It also is used in the hardware implementation of NewHope [71].

Ziggurat Sampler. The Ziggurat sampler is a variation of the rejection sampler introduced in Marsaglia et al. [72] for a continuous Gaussian sampler.[3] A discrete version of Ziggurat sampler is proposed in Buchmann et al. [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). First, 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, a new random x should be sampled and the process is repeated.

More rectangles lead to a reduction in the number of rejections, better performance, and higher precision. Due to its flexibility, Ziggurat offers a tradeoff between memory consumption and

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

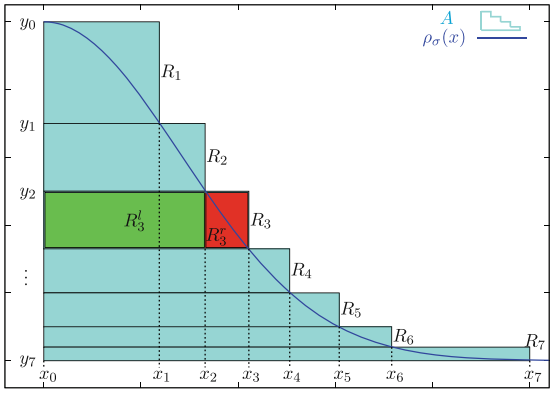

Fig. 2. A partition of Ziggurat [76].

performance, making it suitable for resource-constrained embedded devices. More precision and performance require more precomputed rectangles that impose heavy memory overhead, and the increase in the number of precomputed rectangles could drop 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 the tail-cut (τ) and Gaussian parameter (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 a binary search, on the CDF to find the corresponding x. A standard CDT needs a 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 precomputed tables of size 630 kbits and 370 kbits for 192- and 128-bit post-quantum security, respectively, which is not practical for resource-constrained embedded devices [33]. By employing Peikert's convolution lemma [47], the size of precomputed tables is dropped by a 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 Pöppelmann [58] by employing an adaptive mantissa size which halves the table size.

Pöppelmann et al. [78] proposed a hardware implementation of the CDT sampler which is further optimized in other works [60, 79, 80]. Du et al. [81] propose an efficient software implementation of CDT that is vulnerable to timing attacks; this is resolved in Khalid et al. [82], who suggest 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 rth row denotes the binary expansion of p_r. According to the

probability matrix, a discrete distribution generating the binary tree (DDG) is built whose ith level corresponds to the ith 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, the Knuth-Yao sampler needs a large memory to store the probability of the sample points with high precision, which is an issue on resource-constrained platforms. Combining 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 a small standard deviation that results in faster sampling; the same authors improved their implementation [86].

Based on the presented results in Buchmann et al. [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. The 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 a digital signature, the Knuth-Yao sampler is not suitable for digital signatures. Inside an encryption scheme with $\sigma_{LP} = 3.3$ [46], with the same security level ($\lambda = 64$), the Knuth-Yao sampler beats CDT in terms of the number of operations performed in one second per slice of FPGA (Op/s/S) for time-independent implementations [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 the Knuth-Yao implementation of Roy et al. [86] in term of Op/s/S.

The size of precomputed tables in a Bernoulli sampler is two orders of magnitude smaller than those of the CDT and Knuth-Yao samplers [33]; however, CDT has three times more throughput than Bernoulli for hardware the implementation of BLISS in Pöppelmann [60].

2.5 Lattice-Based Schemes

Security of the LBC schemes is based on the hardness of solving two average-case problems: SIS [87] and LWE [7].

Regev [7] proposed the LWE problem which can be reduced to a worst-case lattice problem like the SIVP. In the proposed scheme, the ciphertext is $O(n\log n)$ times bigger than plaintext; however, in Peikert et al. [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 Lindner and Peikert [46]. Another difference between Regev's encryption scheme and the LP scheme is that the former uses a discrete Gaussian sampler in the key generation step, while LP employs a 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: 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 another 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 key and Enc is the 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 a message to Bob. Encryption and decryption ciphers are one-way functions and are known publicly; hence the only secret data are the 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. A digital signature consists of three steps: key generation, Sign_{sk}, and Verify_{pk}. In the first step, the secret key (sk) and public key (pk) are generated; the signer keeps the secret key and all verifier parties have the public key of the signer. During the sign step, the signer applies the encryption algorithm on input message M with its private key and produces output S as $S = \text{Sign}_{sk}(M, sk_{signer})$. The 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 the 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 $S = Enc(D, sk_{signer})$. Afterward, signer sends the pair (M, S) to the verifier. Subsequently, the verifier uses a public key to decrypt S as $D' = Dec(S, pk_{signer})$. Then, the 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); 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: 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 first two works to propose lattice-based signatures which are not provably secure (due to the deterministic signing). Because of the high standard deviation of a Gaussian sampler, implementation of lattice-based digital signatures that use a Gaussian sampler is challenging. In addition, the need for hash function components and a 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, a public key 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 LWE (e.g., Frodo [32]), RLWE (e.g., BCNS [98] and NewHope [40]), and MLWE (e.g., Kyer[41]) that require less bandwidth compared 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

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

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[2] [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.

[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 that call for post-quantum algorithms [101] where authors first make an IND-CPA PKE and then build the CCA-KEM with the same parameter space by applying the KEM version of the transform [100].

Many submitted proposals to the NIST PQC standardization use a single proposal (e.g., LOTUS [102]) that supports both PKE (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 redundancy, in Section 3, we describe each scheme under only one of the categories of PKE (Section 3.2.1) or key exchange mechanism (Section 3.2.2). Similarly, to avoid redundancy, both software and hardware implementations are mentioned only once (software implementation in Section 3.2; hardware implementation in Section 3.3).

The contemporary LBC schemes extant in the literature are listed in Table 2. Details on the security level, public key, secret key, and ciphertext size of lattice-based PKE and key establishment

Table 3. Comparison of Contemporary Lattice-Based Public Key Encryption and Key Exchange Schemes

Scheme	PQ security		Failure Probablity[1]	Size (bytes)		
	SVP	CCA		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

(Continued)

Table 3. Continued

Scheme	PQ security		Failure Probablity[1]	Size (bytes)		
	SVP	CCA		Secret Key	Public Key	Ciphertext
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).

(KEX/KEM) schemes can be seen in Table 3. Details on the security level, secret key, public key, and signature size of the lattice-based digital signature schemes are listed in Table 4.

3 IMPLEMENTATION CHALLENGES

In this section, we consider various implementations of LBC schemes using software, hardware, software/hardware codesign, and DSP techniques. First we focus on the implementation of key arithmetic modules such as the Gaussian sampler and matrix/polynomial multiplication in Section 3.1. Software and hardware implementations of LBC 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 for computing the exponential function efficiently on resource-constrained devices regarding memory capacity. In order to decrease the memory footprint, pipelining the sampling algorithm is used, in which the 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 the memory (lookup table) footprint since lookup tables only store non-tail probabilities, 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.

Software Implementation. Buchmann et al. [76] design and implement, in C++, a flexible Gaussian sampler named Ziggurat which sets a tradeoff 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 an expensive exponential function. More rectangles (greater 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-constrained embedded devices (use few numbers of rectangles to minimize memory consumption). In order to increase the performance of the Ziggurat sampler, more rectangles are required, which imposes significant memory overhead since it needs to recompute all the rectangles and save them in memory. A better way to improve the Ziggurat sampler is to increase the number of approximation lines (by adding two extra points), which culminates in decreasing the number of exponential function calculations [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 are improved by employing more approximate lines.

Hardware Implementation. Roy et al. [85] implement the first high-precision and low-area hardware implementation of the Knuth-Yao sampler using a small standard deviation on a

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]	-	-

Xilinx Virtex5 FPGA. Knuth-Yao involves a random walk tree traversal which imposes expensive sequential bit scanning and a wide ROM footprint. To improve performance, the authors traverse the Discrete Distribution Generating (Ddg) tree by employing the 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. As presented, the Knuth-Yao sampler suffers from vulnerability to timing and power attacks due to the nonconstant time random walk, which is solved by a random shuffle approach to eliminate leaking timing information [86]. The authors offer the solution (random shuffle) but do not evaluate the 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 the 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. They reduce area occupation by employing piece-wise comparison (which results in 90% saving in the random bits) and avoiding a comparison of large numbers, which is an improvement on Pöppelmann and Güneysu [79]. Further improvement is achieved by employing small LUTs with high hit rates, which results in a 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 obstacle to improving 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 multilevel 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 nonconstant 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. The authors analyze different quantum secure parameters for a PKE scheme and digital signature. A CDT Gaussian sampler provides higher throughput with lower memory footprint when used for digital signature. Due to the inferior performance of the hardware Ziggurat implementation, the authors discourage the use of the Ziggurat sampler for digital signatures. Similarly, the CDT sampler achieves better balanced area and throughput for public key encryption, and allowing the use of BRAMs makes the Knuth-Yao variant a much superior design in terms of area and throughput. The authors use BRAMs to decrease occupied slices in the FPGA and to save precomputed values, which significantly improves performance.

Pöppelmann and Güneysu [59] propose an area-optimized hardware implementation of the 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 the proposed Gaussian sampler is suitable for encryption schemes, it would be challenging to employ inside digital signatures as the sampling component. Pöppelmann et al. [60] propose a hardware implementation of the Bernoulli sampler with binary Gaussian distribution for a `BLISS` scheme on Xilinx Spartan-6 FPGA.

Göttert et al. [56] propose the first hardware implementation of the discrete Gaussian sampler. They use rejection sampling in their software implementation; however, because of the obligatory floating point arithmetic, they prefer to employ lookup tables in their hardware implementation, in which the Gaussian distributed values in the stored array are indexed using a pseudo-random bit generator. The proposed sampler has unsatisfactory precision, far from the golden discrete Gaussian distribution due to its small tail bound. The authors employ a fully parallel architecture to design a polynomial multiplier which provides high throughput but makes the design extremely big; it 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 a reduction in fixed computation (4 NTT instead of 5 NTT) and in reducing the prescaling overhead. In addition, NTT memory access is minimized by storing two coefficients in a single word, processing two pairs of coefficients together, and eliminating idle cycles. The authors avoid using ROM to save twiddle factors and instead compute twiddle factors on demand. In addition, they reduce security by limiting coefficients of the secret key to be binary instead of Gaussian distributed, which allows multiplication to be replaced by addition operations. Small lookup tables are used in the Knuth-Yao discrete Gaussian sampler to avoid expensive bit scanning [85] and 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 the NTT algorithm (on an Nvidia GPU). Employing a 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 sparse polynomial multiplication for the first time, which improves the performance of the digital signature proposed in Güneysu [144] by 34%. The authors implement Schönhage-Strassen polynomial multiplication on an NVIDIA GPU and compare its performance with existing multiplication schemes, including iterative NTT, parallel NTT, and CUDA-based FFT (cuFFT) for different integer sizes.

Akleylek et al. [156] propose a software implementation (Intel Core i5-3210M) of sparse polynomial multiplication using a sliding window that results in around an 80% speed improvement compared to NTT [138]. The authors assume polynomials with coefficients with three possible values including -1,0 and +1. Multiplication by zero is 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 a high number of zeros and numerous identical patterns, which makes the system prone to timing attacks.

`FNLlib` [157] is a scalable, efficient, and open source C++ library containing optimized arithmetic operations on the polynomials for ideal LBC 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. The 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 1.9× for a C implementation (on a 64-bit platform) and 1.25× for an AVX2 vector implementation compared to NewHope (tolerant against timing attacks) is achieved. However, due to the lack of a 64-bit register, the proposed reduction technique does not provide speed-up on 32-bit microcontrollers [63]. Additionally, the 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 a standard lattice-based encryption scheme based on the LWE problem. These authors perform 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 the standard LWE-based encryption engine.

Pöppelmann et al. [78] propose the first hardware optimization of polynomial multiplication (NTT) for ideal lattice-based encryption schemes on a Xilinx Spartan-6 FPGA with the primary goal of minimizing the area. The 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 [79] for use in a Ring-LWE encryption engine. With acceptable runtime, Aysu et al. [37, 159] present an optimized hardware implementation of the NTT introduced in Pöppelmann et al. [78] to compute polynomial multiplication in a Ring-LWE on the smallest Spartan-3. The main idea is to compute twiddle factors on demand instead of storing them in the ROM. By replacing the modulus with a Fermat number, a shift operation can be used instead of polynomial exponentiation. However, the proposed optimizations cannot take advantage of the inherent parallelism in NTT. It should be mentioned that the authors [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. The 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; thus 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 multipliers (p has the adjustable length) which produce outputs (equivalent to two parallel NTT) that can be used to perform inverse-FFT.

Du et al. [161] propose a scalable and efficient polynomial multiplier architecture, implemented on the Xilinx Spartan-6 FPGA, that takes advantage of NTT's parallelism and provides a speed and area tradeoff. In several works ([78] and [37, 159]), one and two butterfly operators are employed, respectively; however, Du and Bai [161] use b (power of 2) butterfly operators to improve the speed of the polynomial multiplier, which performs 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 minimize the number of constant factors. The butterfly operation takes two coefficients (x, y) and one constant factor (ω) and makes two new coefficients ($[x+\omega y]$ mod p, $[x-\omega y]$ mod 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 are $(1.5n + 1.5n\log n)$, 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. The 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 runtime, throughput, area, and scalability. The first architecture yields the best performance using a pipelined modular butterfly-based FFT. The other two architectures are serially distributed arithmetic-based and nested multiplication, which occupy less area than the butterfly-based FFT.

Du et al. [163] achieve a 30% savings in time and space compared to Pöppelmann and Güneysu [78] on a Spartan-6 FPGA by performing an 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 $bit\text{-}reverse(i)^{th}$ memory location. Consequently, the bit-reversal step in the inverse-NTT is eliminated. The authors employ two Block RAMs on FPGA, which provide interleaving; hence, two parallel NTTs can be interleaved. They apply their optimization to the NTT of an RLWE-based public key cryptosystem [164]. Also, it is assumed that the uniformly random polynomial in the public key scheme is fixed, hence precomputing the NTT offline improves performance. In both papers, only one butterfly operator is used; however, by adapting bit-reversal and memory saving methods, the authors propose a fast polynomial multiplication architecture with four butterfly operators that achieve, on average, a 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 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 the ARM Cortex-M4F micro-controller. The main goal was maximizing the speed (using assembly level optimization) and minimizing the 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. They employ optimization [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 a Gaussian sampler based on the Knuth-Yao algorithm, which allows them to implement a Ring-LWE-based PKE scheme on a resource-constrained 8-bit ATxmega128 AVR processor. By applying sophisticated memory alignments for storing coefficients, about a 20% 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 PKE scheme on small 8-bit ATXmega128 and 32-bit Cortex-M0 microcontrollers 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 speed, it is beaten by the scheme in Liu et al. [134], with its slightly higher memory footprint. Similarly, the proposed design in Pöppelmann [38] uses NTT with precomputed twiddle factors and eliminates the bit reversal step, which results in a 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] are employed for NTRU schemes. 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. To be resilient to first-order side-channel attacks, 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.

Cheon et al. [103] exploit the 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 categories 1, 3, and 5). Sparse and small-secrets versions of LWE and LWR (RLWE and RLWR) are the security basis of the Lizard (RLizard) IND-CPA PKE. Besides an Intel Xeon CPU, the authors provide performance evaluation on a smartphone (Samsung Galaxy S7) for their recommended parameter of Lizard.CPA (128-bit quantum security). They claim that Lizard is suitable for smartphones (memory usage of 20 megabytes). The authors provide datapath and finite state machines 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 categories 1, 5, and 5, respectively). Based on the original NTRU scheme [89] and using parameters set by Hoffstein et al. [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 set at a power of 2 (2^{11}) to enhance the efficiency of the modulo arithmetic and integer multiplications. The 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 an 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 ciphertext and key size of around 1kB; these are designed to reduce an attacker's success probability by eliminating ring homomorphisms. Schemes are IND-CCA2, where a key can be used multiple times; hence, large key generation latency is tolerable. The 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 et 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. The authors report the performance results of the reference and AVX2 implementation on an 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 are concealed in the error without changing the target distribution. Consequently, more data are 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 categories 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). The authors adapt the FFT from Nussbaumer's method [26] for polynomial arithmetic in cyclotomic rings whose degree is a power of 2 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, the 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 a Gaussian sampler. BCNS caches keys, which can be very dangerous to the security of the protocol because of shared-key reused attacks [171], a problem that is solved in NewHope.

Alkim et al. [40] introduce NewHope, a portable C and highly optimized SIMD implementation (AVX2) of an 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 a smaller modulus, on-the-fly generation of the polynomial system parameter, efficient polynomial arithmetic (combining Montgomery and Barret reduction and employing polynomial encoding), and use of 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 NewHope with the same performance and security level. Simplicity is achieved by eliminating the error-reconciliation mechanism [97] with a 6% message size overhead. The authors discard the least significant bits of each coefficient due to their negligible impact on successful plaintext recovery. Additionally, the authors encode a single-key bit into four coefficients, which results in a 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 the ARM Cortex-M family, low-power Cortex-M0 and high-performance Cortex-M4, which is the first key exchange scheme with a quantum security level of 128 bits on constrained embedded devices. The authors optimize all hot regions of the protocol in assembly, including error reconciliation, the uniform noise generation by ChaCha20 stream cipher [173], and NTT/NTT^{-1}. For NTT, the authors set a memory-time tradeoff for precomputing the 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 NewHope by optimizing the pseudorandom generation part, which results in 1.5× better performance on the Intel Skylake processors. The 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 the additions of two polynomials, which results in speed improvements of 1.9 and 1.25 for C and AVX implementations (compared to the reference NewHope [40]), respectively.

Adapted from 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 performance comparable to the elliptic curve-based cryptosystems. The four mentioned KEMs are derived from NewHope-CPA-PKE (which does not support arbitrary length messages and thus cannot be used as a standalone encryption scheme) by applying a variant of the FO 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, the 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 the RLWE problem. At the same security level, Ding Key Exchange reduces communication cost (due to its rounding technique) compared to similar schemes (NewHope, NewHope-Simple, and Kyber). It provides security equivalent to AES-128, AES-192, and AES-256 (NIST security categories 1, 3, and 5) with

flexible parameter choices and a key size of n-bit where n can be 512 and 1024; as a result, it is more resistant to the Grover algorithm compared to NewHope, NewHope-Simple, and Kyber with their key size of 256 bits. The 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, has been tested on an Intel Core i7-6700 CPU and is integrated into OQS and OpenSSL. HILA5 uses SafeBits, an 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 a decryption failure of 2^{-128} compared to NewHope's failure rate of 2^{-64} (Frodo [32] and Kyber [41] report failure rates of $2^{-38.9}$ and $2^{-71.9}$, respectively) while 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 security equivalent 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. In addition, SHAKE-256 is used to sample from the uniform distribution.

Frodo (FrodoCCS) [32], the first practical implementation of a PKE 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 a matrix variant should be saved in memory. By generating and afterward discarding the matrix variant (on-the-fly), memory overhead is alleviated. The authors use an efficient inversion sampling method that uses precomputed tables. Based on the authors' claim, integration of Frodo into TLS halves server throughput. Consequently, NTT-friendly primes and polynomials 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 the FO transformation [100] to build FrodoKEM. The authors generate a 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 (generating the public matrix A and matrix arithmetic) are provided. The authors report the results of the implementation on a 64-bit ARM Cortex-A72 (with the best performance achieved by using an 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 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 that 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 a 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 software change; in addition, 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 key transport and encryption, while AKCN, asymmetric key consensus with noise, can only be employed for key transport. In the proposed scheme, the server sets the session key before starting the key exchange mechanism. Consequently, it provides the opportunity for offline message encryption, which provides higher security and better workload balance. Compared 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; the 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, a Ring-LWE-based version of OKCN, which adapts the same noise distribution and parameters as NewHope, provides a more computationally efficient scheme than NewHope. The 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%) compared to Frodo at the same security level (shared key size of 256 bits). The most efficient KEX with the shared key size of 512 bits is achieved by AKCN. The 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 a 765-bit shared key with less bandwidth than NewHope and NewHope-Simple (with a 256-bit shared key). The authors claim that they provide the most efficient lattice-based KEX with a shared key size of 256 bits by applying OKCN/AKCN to the MLWE-based key KEX. Additionally, they provide a new authenticated key exchange scheme named Concealed Nonmalleable Key-Exchange (CNKE).

Seo et al. [106] propose an error-blocked multibit 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; this is achieved 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 using 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 an Intel core-i7-7600, the authors implement schemes on the Zynq 7 FPGA platform.

Kyber [41] is a highly optimized IND-CCA KEM with post-quantum security based on the hardness of solving the (Modulue-LWE) problem [19]. Ideal lattices, with their ring structure, decrease the public key and ciphertext size of standard lattices schemes by sacrificing security assumptions. Module lattices are proposed to fill the gap, believing that a full ring structure is excessive [19]. The authors define an IND-CPA PKE scheme under the Module-LWE hardness assumption and apply a variant of the FO transform [100] to build an IND-CCA KEM. Employing IND-CCA KEM, they design IND-CCA KEX and AKEX under a 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) at the sacrifice of 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 as Kyber512, Kyber768, and Kyber1024 at 102-, 161-, and 218-bit security (NIST security categories 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 categories 1, 3, and 5).

Smart et al. [68] propose LIMA (**L**att**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 categories 1, 2, 3, and 5). The authors employ the FO and Dent transforms [176] to obtain IND-CCA PKE and IND-CCA KEM schemes. In addition to power-of-2 cyclotomic rings (LIMA-2p), the authors propose safe-prime cyclotomics (LIMA-sp) that reduce the probability of subfield attacks by sacrificing efficiency compared to the power-of-2 cyclotomics. In order to avoid decryption failure, the authors perform rejection sampling (from a centered binomial distribution) at the encryption stage, which makes the implementation in non-constant time. 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), an IND-CCA2 secure LWE-based PKE (LOTUS-PKE) and KEM (LOTUS-KEM) with 128-bit, 192-bit, and 256-bit security (NIST security categories 1, 3, and 5). The Knuth-Yao algorithm [83] is employed to sample the error term from the discrete Gaussian distribution. In order to reduce the sampling's overhead, a DDG tree is built online and a probability matrix is stored column-wise. In addition to the reference and optimized implementations, vectorized implementations (employing AVX2 vector instructions) of LOTUS-PKE and LOTUS-KEM are provided.

NTRU-KEM [125], an IND-CCA2-secure KEM based on the 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, and ciphertext size by 20%, 37%, 37%, respectively, and halves the required clock cycles for the encryption/encapsulation step. However, it increases required clock cycles for key generation and decryption/encapsulation by a factor of 3.47.

Plantard [108] presents Odd-Manhattan, an IND-CCA KEM by using the Dent transform on IND-CPA PKE, at 126-, 192-, and 256-bit security (NIST security categories 1, 3, and 5). Odd-Manhattan is based on the *α-Bounded Distance Parity Check (BDPCα)* [177], which imposes a considerable increase in time and the 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 a 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), and p (rounding modulus) $\in \mathbb{Z}^+$ where $q \leq p$ and $n \in 1, d$, if $n = 1$, this instantiated scheme is based on the LWR problem, while $n = d$ ($n + 1$ is prime) results in a RLWR-based scheme. Round2 provides two sets 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, compared to LWE, results in less random data generation due to avoiding sampling from nonuniform noise distribution; in addition, the required bandwidth is reduced since fewer bits are needed per coefficient. Secret terms can be either sparse-trinary (reduces the probability of error in decryption) or uniformly sampled in $\mathbb{Z}_q^d$. In order to have a unique implementation 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 public-key and ciphertext in total for NIST security category 5. Over the same ring as the NTRU-KEM scheme, Round2 shows a 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 categories 1, 3, and 5). Integers are chosen to be the power-of-2 modulus, which results in avoiding explicit modular reduction and relaxing complicated sampling methods (e.g., rejection) by efficient constant time sampling from a (modulo power 2) uniform distribution; however, a power-of-2 modulus prevents using NTT for 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 Alkim et al. [62].

Hamburg [91] introduces *ThreeBears*, a family of IND-CPA and IND-CCA KEM schemes adapted from Kyber [41] and based on the integer Module-LWE (ILWE) [178] problem. *ThreeBears* includes *BabyBear*, *MamaBear* (recommended), and *PapaBear* with NIST security categories 2, 4, and 5, respectively. For each scheme, deterministic CCA-secure (with FO transform) and ephemeral (without FO transform) implementations are presented. To reduce the memory footprint, the private key is rapidly generated by expanding a seed; similarly, the public key and large public matrix are generated in modulus N, where N is a large Mersenne prime. Sampling the noise is performed by expanding a seed to only 1B per digit. Although Saarinen's error correction [116, 117] can notably improve *ThreeBears* security, the author prefers to use a 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, particlularly on devices without vector unit support. 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* (respectively, *PapaBear*) is stronger than *Kyber-Paranoid* (respectively, 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 the 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 tradeoff 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 six suggested parameter sets, *Std*128, *Med*160, *Hi*192, and *Super*256 satisfy minimum security specified by NIST (categories 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, the authors provide performance analysis of avectorized implementation (AVX2) on Intel i7-7700K.

3.2.3 Digital Signature. Lyubashevsky [110] presents a Short Integer Solution (SIS) problem-based digital signature scheme adapted from the 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]. The 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 microcontrollers. 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 the bit reversal step during polynomial multiplication and applying the Gaussian sampling in a lazy manner. Compared to RLWE encryption, BLISS needs a 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 imposes a remarkable performance overhead. Consequently, for the Gaussian sampler, a 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 by 2.8x by employing the ternary representation of polynomials in order to shorten the length of the random numbers. During key generation, keys are rejected, which leads to a 5-10x 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 cannot be used in BLISS, BLISS-generated keys are compatible with BLISS-B.

Oder et al. [139] present an efficient software implementation of BLISS on the ARM Cortex-M4F to optimize the throughput along with minimizing the memory footprint. The 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 the 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 the GLP signature [61] and implement it on Intel's Sandy and Ivy Bridge processors. In the proposed scheme, the Gaussian sampler is replaced with uniform sampling from $\{-1, 0, +1\}$; to benefit from the AVX, each 512 double-precision floating-point array of coefficients is 32-byte aligned. In addition, 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 a 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× better speed than the matrix version for the 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. Only a 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 an efficient software implantation of lattice-based GLP and BLISS authentication protocols for resource-constrained smart cards and microcontrollers (ARM and AVR). The authors perform a design space exploration by choosing different parameter sets for FFT and the Gaussian sampler (Knuth-Yao and Bernoulli) along with the various PKE schemes. They conclude that LBC schemes are efficient enough to be implemented on constrained devices.

Alkim et al. [111] introduce TESLA (a tightly secure signature in random oracle model) 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 Dagdelen et al. [133] due to employing parallel matrix-vector multiplication and lazy reduction. The 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. A large public key size (about 1 MB) makes it impractical to implement. The same authors present a fast, small, and provably secure Ring-LWE-based software implementation of TESLA that uses uniform sampling on the same platform; this version reduces the key size by about three orders of magnitude [142]. The proposed Ring-TESLA benefits from AVX2 instructions, which has a one-cycle throughput for eight doubles integers. Instantiation from Ring-TESLA leads to the rejection of valid signatures in the verification stage due to a problem in parameter selection, which is solved in Chopra [121] by new parameter selection method and in [51] by altering the algorithm. Based on the claims in Barreto et al. [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 and 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 an 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 storage of all 32 bits of coefficients of the polynomial. Bindel et al. [124] propose qTESLA, a family of)provably, existentially unforgeable 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 security categories of 1, 3, and 5, respectively. 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 seeds in key generation and the 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 Bindel et al. [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 a costly signing step. The 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 leakage 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. The masking property of Gaussian samples is achieved by randomizing and combining multiple numbers of sample vectors.

Dilithium [131], a simple and efficient digital signature scheme resistant to lattice reduction attacks (with the same conservative security parameter as in Alkim et al. [40]), is adapted from designs [61], [94] that uses the Fiat-Shamir Abort Framework [110], which is secure in random oracle model (no security proof in quantum random oracle model is presented). The 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 a less than 5% increase in signature size. The authors set *total size* = *signature size* + *public key size* as their size parameter. Dilithium over a 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 a 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 highly secure to weakly secure schemes, respectively. However, Dilithium-G extract noise from the Gaussian sampler, which results in better security but vulnerable to timing attacks. Rejection sampling is the same for both schemes; if individual coefficients of a signature are not within a certain range, the signing procedure must be restarted. Dilithium employs standard NTT-based polynomial multiplication; however, in the 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 categories 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 the *MSIS* problem and hash function *H*). The signature size of Dilithium is about 2× bigger than that of BLISS ([33],[186]) (the smallest schemes among lattice-based digital signatures) that uses a discrete Gaussian sampler, which Dilithium avoids. However, compared to the most efficient lattice-based digital signature schemes that avoid Gaussian samplers, Dilithium achieves a 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 (to avoid reduction after addition and subtraction). A vectorized (AVX2 instruction set on the Intel Core i7-4770K) version of NTT gives a 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 categories 1, 3, and 5) with the primary goal of minimizing the *total size = signature size + public key size*. FALCON is the result of combining the 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 Ducas et al. [186], which can be transformed to FALCON by employing Fast Fourier sampling in private key operations. NTRU lattices with the capability of message recovery (entirely from the signature) result in the compactness of FALCON. The 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 devices without floating point units. Another downside of FALCON is its extensive use of discrete Gaussian sampling over integers, which is hard to protect against timing and side-channel attacks. FFT over 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 as the sampler, ChaCha20 as the PRNG and SHAKE-256 as XOF for all the security levels. FALCON can be easily transformed into an IBE scheme [186]. The authors only provide a performance evaluation of the reference implementation on an Intel Core i7-6567U CPU with 15% of the 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 of the same security level.

Chen et al. [74] present pqNTRUSign, a modular lattice-based hash-then-sign digital signature scheme (introduced in Hoffstein et al. [189]) at a 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 sets 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 the 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 studies of BLISS-BI [120] and NewHope [40]. The authors present a quick design flow that performs exploration and the design of programmable accelerators that lead to, on average, 35% and 50% improvement in the latency and energy-delay product, respectively. The authors create a programmable accelerator for NTT that can be employed in any scheme, with any set of parameters, and 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 PKE on the Xilinx Virtex-7 FPGA. Due to the large area occupation of the full Ring-LWE PKE scheme, only LWE-polynomial variants are chosen to be implemented. Proposed implementations are based on LP lattice-based encryption scheme [46], which achieves 316x higher

throughput compared to a 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 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. In addition, 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 a 3× 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. The authors suggest combining the precomputation 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 leads to more a compact and faster sampler than in Pöppelmann and Güneysu [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 Pöppelmann and Güneysu [79], this is one order of magnitude slower due to the nonapplicability of using NTT (using instead a DSP-enabled schoolbook polynomial multiplier). In addition, considerable area is saved by using a 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 the $exp()$ function.

Howe et al. [31] present the first and only hardware implementation of a lattice-based encryption engine based on the LWE 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 Göttert et al. [56]; in addition, it can closely compete with the encryption scheme of Pöppelmann and Güneysu [79]. To maximize performance, the authors use a larger Gaussian sampler, a 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 an evaluation and a 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. The 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. The authors remove the need for Gaussian noise sampling by using 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, the authors achieve high speed and small size for the implementation of the polynomial multiplier. Compared with BLISS, the proposed signature is suboptimal in terms of signature size and security level.

Pöppelmann et al. [60] achieve a high-throughput hardware implementation of BLISS [33] on Xilinx Spartan-6 FPGA. The authors improve and parallelize the column-wise Schoolbook multiplier presented in Güneysu et al. [61]. The authors employ an efficient CDT-based Gaussian sampler with large tables. To improve the performance of the CDT sampler, the authors 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. They provide enhanced CDT, which uses two smaller samples (Peikert convolution theorem [47]). A standard CDT needs a table of at least $\eta \times \tau \times \lambda = 215.73 \times 13.4 \times 128 = 370kb$ while enhanced CDT needs around a 23× smaller table. The authors evaluate the 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; in addition, enhanced CDT achieves 17.4 Million Operation per Seconds (`MOPS`), which is 2.3× more than that of the Bernoulli sampler. Based on the results, the 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 Güneysu et al. [61] but with major improvements. Compared with Güneysu et al. [61], instead of the 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. They develop a flexible processing core with VHDL that could be configured as either a signing or/and verification engine which does not impose any overhead to provide flexibility. The signing step breaks into three separate blocks, including a lattice processing engine, random oracle, and sparse multiplication along with compression unit; these run in parallel. The digital signature processor is based on the lattice processor for the PKE scheme in Pöppelmann and Güneysu [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 a decent performance level. With the same post-quantum security level as `NewHope` (128-bit), server and client work with clock frequencies of 125 and 117 MHz, respectively. The 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; this results in the lack of reusability as a disadvantage. For the sake of area optimization, 512 butterfly operations are performed serially, although 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; in addition, 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 generating 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 (e.g., 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 a 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. The signature generation scheme consists of two separate phases including offline and online phases, which are performed on the NIOS soft-core as software and on the Altera Cyclone-IV FPGA as hardware, respectively. Hardware is responsible for the 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 the 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

LBC 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 to tackle different tradeoffs, such as memory footprint, security, performance, and energy efficiency, 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. In addition, securing such platforms against physical attacks is a topic that needs to be researched.

In this article, we provided a review of LBC, 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.

REFERENCES

[1] Hamid Nejatollahi, Nikil Dutt, Sandip Ray, Francesco Regazzoni, Indranil Banerjee, and Rosario Cammarota. 2017. Software and hardware implementation of lattice-cased cryptography schemes. *University of California Irvine, CECS TR 17-04* (2017).

[2] Peter W. Shor. 1997. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal of Computing* (1997).

[3] Gui-Lu Long. 2001. Grover algorithm with zero theoretical failure rate. *Physical Review A* (2001).

[4] Ali Ansarmohammadi, Saeed Shahinfar, and Hamid Nejatollahi. 2015. Fast and area efficient implementation for chaotic image encryption algorithms. In *CADS*.
[5] Ali Ansarmohammadi, Hamid Nejatollahi, and Ghasemi Mehdi. 2013. A low-cost implementation of AES accelerator using HW/SW co-design technique. In *CADS*.
[6] Oscar Garcia-Morchon, Ronald Rietman, Sahil Sharma, Ludo Tolhuizen, and Jose Luis Torre-Arce. 2015. DTLS-HIMMO: Achieving DTLS certificate security with symmetric key overhead. In *ESORICS*.
[7] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. (2005).
[8] Miklós Ajtai. 1996. Generating hard instances of lattice problems (extended abstract). In *STOC*.
[9] Daniele Micciancio and Oded Regev. 2009. *Lattice-based Cryptography*.
[10] Miklós Ajtai, Ravi Kumar, and Dandapani Sivakumar. 2001. A sieve algorithm for the shortest lattice vector problem. In *STOC*.
[11] Daniele Micciancio and Panagiotis Voulgaris. 2010. Faster exponential time algorithms for the shortest vector problem. In *SODA*.
[12] Divesh Aggarwal, Daniel Dadush, Oded Regev, and Noah Stephens-Davidowitz. 2015. Solving the shortest vector problem in 2N time using discrete Gaussian sampling: Extended abstract. In *STOC*.
[13] Daniele Micciancio. 2010. *Cryptographic Functions from Worst-Case Complexity Assumptions*.
[14] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. 2009. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*.
[15] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. 2013. Classical hardness of learning with errors. In *STOC*.
[16] Benny Applebaum, et al. 2009. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*.
[17] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *EUROCRYPT'10*.
[18] Abhishek Banerjee, Chris Peikert, and Alon Rosen. 2012. Pseudorandom functions and lattices. In *Proceedings of the Annual International Conference on Theory and Applications of Cryptographic Techniques*.
[19] Adeline Langlois and Damien Stehlé. 2012. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive. (2012).
[20] Hamid Nejatollahi, Nikil Dutt, and Rosario Cammarota. 2017. Trends, challenges and needs for lattice-based cryptography implementations: Special session. In *CODES*.
[21] Donald E. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*.
[22] Stephen Cook, et al. 1969. On the minimum computation time of functions. *Ph.D. Dissertation, Harvard University* (1969).
[23] Anatolii Karatsuba and Yu Ofman. 1963. Multiplication of many-digital numbers by automatic computers. In *USSR Academy of Sciences*.
[24] Arnold Schönhage and Volker Strassen. 1971. Schnelle multiplikation Grosser Zahlen. *Computing* (1971).
[25] Martin Fürer. 2009. Faster integer multiplication. *SIAM Journal of Comput.* (2009).
[26] Henri Nussbaumer. 1980. Fast polynomial transform algorithms for digital convolution. *TASSP* (1980).
[27] James W. Cooley, et al. 1965. An algorithm for the machine calculation of complex journal = Mathematics of Computation, fourier booktitle. (1965).
[28] W Morven Gentleman, et al. 1966. Fast fourier transforms: For fun and profit. In *AFIPS'66*.
[29] Peter L Montgomery. 1985. Modular multiplication without trial division. *Mathematics of Computation* (1985).
[30] Paul Barrett. 1986. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *CRYPTO*.
[31] J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and K. Beeden. 2016. Lattice-based encryption over standard lattices in hardware. In *DAC*.
[32] Joppe Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. 2016. Frodo: Take off the ring! Practical, quantum-secure key exchange from LWE. In *CCS*.
[33] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013. Lattice signatures and bimodal Gaussians. In *CRYPTO*.
[34] James Howe, Thomas Pöppelmann, Máire O'Neill, Elizabeth O'Sullivan, and Tim Güneysu. 2015. Practical lattice-based digital signature schemes. *TECS* (2015).
[35] Tobias Oder, Tim Güneysu, Felipe Valencia, Ayesha Khalid, Maire O'Neill, and Francesco Regazzoni. 2016. Lattice-based cryptography: From reconfigurable hardware to ASIC. In *ISIC*.
[36] Franz Winkler. 1996. Polynomial algorithms in computer algebra. In *TMSC*.
[37] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. 2014. Compact ring-LWE cryptoprocessor. In *CHES'14*.

[38] Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. 2015. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In *LATINCRYPT*.
[39] Patrick Longa and Michael Naehrig. 2016. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. Cryptology ePrint Archive. (2016).
[40] Erdem Alkim, Léo Ducas, Thomas Pűppelmann, and Peter Schwabe. 2015. Post-quantum key exchange: -A new hope. Cryptology ePrint Archive. (2015).
[41] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, and Damien Stehlé. 2017. CRYSTALS: Kyber: A CCA-secure Module-Lattice-Based KEM. Cryptology ePrint Archive. (2017).
[42] Jean Pierre David, et al. 2007. Hardware complexity of modular multiplication and exponentiation. *TC* (2007).
[43] Donald Donglong Chen, Gavin Xiaoxu Yao, Ray C. C. Cheung, Derek Pao, and Cetin Kaya Koç. 2016. Parameter space for the architecture of FFT-based montgomery modular multiplication. *TC* (2016).
[44] Ciara Rafferty, Maire O'Neill, and Neil Hanley. 2017. Evaluation of large integer multiplication methods on hardware. *TC* (2017).
[45] Paul G. Comba. 1990. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal* (1990).
[46] Richard Lindner and Chris Peikert. 2011. Better key sizes (and attacks) for LWE-based encryption. In *CT-RSA'11*.
[47] Chris Peikert. 2010. An efficient and parallel Gaussian sampler for lattices. In *CRYPTO'10*.
[48] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. 2015. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In *ASIACRYPT*.
[49] Markku-Juhani O. Saarinen. 2015. Gaussian sampling precision in lattice cryptography. Cryptology ePrint Archive. (2015).
[50] Markku-Juhani O. Saarinen. 2017. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering* (2017).
[51] Paulo S. L. M. Barreto, Patrick Longa, Michael Naehrig, Jefferson E. Ricardini, and Gustavo Zanon. 2016. Sharper Ring-LWE signatures. Cryptology ePrint Archive. (2016).
[52] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill. 2016. On practical discrete Gaussian samplers for lattice-based cryptography. *TC* (2016).
[53] Daniele Micciancio and Michael Walter. 2017. Gaussian sampling over the integers: Efficient, generic, constant-time. Cryptology ePrint Archive. (2017).
[54] János Folláth. 2014. Gaussian sampling in lattice based cryptography. *Tatra Mountains Mathematical Publications* (2014).
[55] John Von Neumann. 1951. Various techniques used in connection with random digits. *National Bureau of Standards Applied Mathematics booktitle* (1951).
[56] Norman Göttert et al. 2012. On the design of hardware building blocks for modern lattice-based encryption schemes. In *CHES*.
[57] Léo Ducas and Phong Q. Nguyen. 2012. Faster Gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT*.
[58] Thomas Pöppelmann. 2016. *Efficient Implementation of Ideal Lattice-Based Cryptography*. Ruhr-Universität Bochum.
[59] Thomas Pöppelmann and Tim Güneysu. 2014. Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *ISCAS*.
[60] Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. 2014. Enhanced lattice-based signatures on reconfigurable hardware. In *CHES*.
[61] Tim Güneysu, et al. 2012. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES*.
[62] Erdem Alkim, Léo Ducas, Thomas Poppelmann, and Peter Schwabe. 2016. NewHope without reconciliation. Cryptology ePrint Archive. (2016).
[63] Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. 2016. NewHope on ARM cortex-M. In *SPACE*.
[64] Silvan Streit and Fabrizio De Santis. 2017. Post-quantum key exchange on ARMv8-A: A New Hope for NEON made simple. Cryptology ePrint Archive. (2017).
[65] Thomas Poppelmann, Erdem Alkim, Roberto Avanzi, Joppe Bos, Léo Ducas, Antonio de la Piedra, Peter Schwabe, and Douglas Stebila. 2017. *NewHope*. Technical Report. National Institute of Standards and Technology.
[66] Markku-Juhani O. Saarinen. 2017. *HILA5*. Technical Report. National Institute of Standards and Technology.
[67] Xianhui Lu, Yamin Liu, Dingding Jia, Haiyang Xue, Jingnan He, and Zhenfei Zhang. 2017. *LAC*. Technical Report. National Institute of Standards and Technology.
[68] Nigel P. Smart, Martin R. Albrecht, Yehuda Lindell, Emmanuela Orsini, Valery Osheter, Kenny Paterson, and Guy Peer. 2017. *LIMA*. Technical Report. National Institute of Standards and Technology.
[69] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2017. *CRYSTALS-KYBER*. Technical Report. National Institute of Standards and Technology.

[70] Ron Steinfeld, Amin Sakzad, and Raymond K. Zhao. 2017. *Titanium*. Technical Report. National Institute of Standards and Technology. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions.
[71] Oder Tobias and Güneysu Tim. 2017. Implementing the NewHope-simple key exchange on low-cost FPGAs. In *LATINCRYPT*.
[72] George Marsaglia, Wai Wan Tsang, et al. 2000. The Ziggurat method for generating random variables. *Journal of Statistical Software* (2000).
[73] George E. P. Box, Mervin E. Muller, et al. 1958. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics* (1958).
[74] Cong Chen, Jeffrey Hoffstein, William Whyte, and Zhenfei Zhang. 2017. *pqNTRUSign: A Modular Lattice Signature Scheme*. Technical Report. National Institute of Standards and Technology.
[75] Zhenfei Zhang, Cong Chen, Jeffrey Hoffstein, and William Whyte. 2017. *NTRUEncrypt*. Technical Report. National Institute of Standards and Technology.
[76] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. 2013. Discrete Ziggurat: A time-memory trade-off for sampling from a Gaussian distribution over the integers. In *SAC*.
[77] David B. Thomas, Wayne Luk, Philip H. W. Leong, and John D. Villasenor. 2007. Gaussian random number generators. *ACM CSUR* (2007).
[78] Thomas Pöppelmann and Tim Güneysu. 2012. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In *LATINCRYPT*.
[79] Thomas Pöppelmann and Tim Güneysu. 2013. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *SAC*.
[80] Chaohui Du and Guoqiang Bai. 2015. Towards efficient discrete Gaussian sampling for lattice-based cryptography. In *FPL*.
[81] C. Du and G. Ba. 2016. High-performance software implementation of discrete Gaussian sampling for lattice-based cryptography. In *ITNEACC*.
[82] A. Khalid, J. Howe, C. Rafferty, and M. O'Neill. 2016. Time-independent discrete Gaussian sampling for post-quantum cryptography. In *FPT*.
[83] Donald E. Knuth and Andrew C. Yao. 1976. The complexity of nonuniform random number generation. *Algorithms and Complexity: New Directions and Recent Results* (1976).
[84] Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. 2015. Efficient software implementation of ring-LWE encryption. In *DATE*.
[85] Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. 2013. High precision discrete Gaussian sampling on FPGAs. In *SAC*.
[86] Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. 2014. Compact and side channel secure discrete Gaussian sampling. Cryptology ePrint Archive. (2014).
[87] Daniele Micciancio and Oded Regev. 2007. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal of Computing* (2007).
[88] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A framework for efficient and composable oblivious transfer. In *CRYPTO*.
[89] Jeffrey Hoffstein, et al. 1998. NTRU: A ring-based public key cryptosystem. In *ANTS-III*.
[90] Damien Stehlé, et al. 2011. Making NTRU as secure as worst-case problems over ideal lattices. In *EUROCRYPT*.
[91] Mike Hamburg. 2017. *Three Bears*. Technical Report. National Institute of Standards and Technology.
[92] Miruna Rosca, Amin Sakzad, Ron Steinfeld, and Damien Stehlé. 2017. Middle-product learning with errors. Cryptology ePrint Archive. (2017).
[93] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. 2008. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*.
[94] Shi Bai and Steven D. Galbraith. 2014. An improved compression technique for signatures based on learning with errors. In *CT-RSA*.
[95] 1997. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*.
[96] Jeffrey Hoffstein, et al. 2003. NTRUSign: Digital signatures using the NTRU lattice. In *CT-RSA*.
[97] Jintai Ding, Xiang Xie, and Xiaodong Lin. 2012. A simple provably secure key exchange scheme based on the learning with errors problem. Cryptology ePrint Archive, (2012).
[98] Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. 2015. Post-quantum key exchange for the TLS protocol from the ring learning with errors problem. In *SP*.
[99] Eiichiro Fujisaki and Tatsuaki Okamoto. 1999. How to enhance the security of public-key encryption at minimum cost. In *PKC*.
[100] Dennis Hofheinz, Kathrin HÃűvelmanns, and Eike Kiltz. 2017. A modular analysis of the Fujisaki-Okamoto transformation. Cryptology ePrint Archive. (2017).

[101] 2017. NIST: National institute for standards and technology. Postquantum Crypto Project. (2017).
[102] Le Trieu Phong, Takuya Hayashi, Yoshinori Aono, and Shiho Moriai. 2017. *LOTUS*. Technical Report. National Institute of Standards and Technology.
[103] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. 2016. Lizard: Cut off the tail! Practical post-quantum public-key encryption from LWE and LWR. Cryptology ePrint Archive. (2016).
[104] Jung Hee Cheon, Sangjoon Park, Joohee Lee, Duhyeong Kim, Yongsoo Song, Seungwan Hong, Dongwoo Kim, Jinsu Kim, Seong-Min Hong, Aaram Yun, Jeongsu Kim, Haeryong Park, Eunyoung Choi, Kimoon kim, Jun-Sub Kim, and Jieun Lee. 2017. *Lizard*. Technical Report. National Institute of Standards and Technology.
[105] Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang. 2017. Choosing parameters for NTRUEncrypt. In *CT-RSA*.
[106] Minhye Seo, Jong Hwan Park, Dong Hoon Lee, Suhri Kim, and Seung-Joon Lee. 2017. *EMBLEM and R.EMBLEM*. Technical Report. National Institute of Standards and Technology.
[107] Michael Naehrig, Erdem Alkim, Joppe Bos, Léo Ducas, Karen Easterbrook, Brian LaMacchia, Patrick Longa, Ilya Mironov, Valeria Nikolaenko, Christopher Peikert, Ananth Raghunathan, and Douglas Stebila. 2017. *FrodoKEM*. Technical Report. National Institute of Standards and Technology.
[108] Thomas Plantard. 2017. *Odd Manhattan*. Technical Report. National Institute of Standards and Technology.
[109] Oscar Garcia-Morchon, Zhenfei Zhang, Sauvik Bhattacharya, Ronald Rietman, Ludo Tolhuizen, and Jose-Luis Torre-Arce. 2017. *Round2*. Technical Report. National Institute of Standards and Technology.
[110] Vadim Lyubashevsky. 2009. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*.
[111] Erdem Alkim, Nina Bindel, Johannes Buchmann, Ãzugijr Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krd'mer, and Filip Pawlega. 2015. Revisiting TESLA in the quantum random oracle model. Cryptology ePrint Archive. (2015).
[112] Sauvik Bhattacharya, Oscar Garcia-Morchon, Ronald Rietman, and Ludo Tolhuizen. 2017. spKEX: An optimized lattice-based key exchange. Cryptology ePrint Archive. (2017).
[113] Zhengzhong Jin and Yunlei Zhao. 2017. Optimal key consensus in presence of noise. Cryptology ePrint Archive. (2017).
[114] Yunlei Zhao, Zhengzhong jin, Boru Gong, and Guangye Sui. 2017. *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.
[115] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. 2016. NTRU Prime: Reducing attack surface at low cost. Cryptology ePrint Archive. (2016).
[116] Markku-Juhani Olavi Saarinen. 2017. Ring-LWE ciphertext compression and error correction: Tools for lightweight post-quantum cryptography. In *IoTPTS*.
[117] Markku-Juhani O. Saarinen. 2017. HILA5: On reliability, reconciliation, and error correction for Ring-LWE encryption. Cryptology ePrint Archive. (2017).
[118] Vadim Lyubashevsky. 2012. Lattice signatures without trapdoors. In *EUROCRYPT*.
[119] Rachid El Bansarkhani and Johannes Buchmann. 2013. Improvement and efficient implementation of a lattice-based signature scheme. In *SAC*.
[120] Léo Ducas. 2014. Accelerating Bliss: The geometry of ternary polynomials. Cryptology ePrint Archive. (2014).
[121] Arjun Chopra. 2016. Improved parameters for the Ring-TESLA digital signature scheme. Cryptology ePrint Archive. (2016).
[122] Arjun Chopra. 2017. GLYPH: A new instantiation of the GLP digital signature scheme. Cryptology ePrint Archive. (2017).
[123] Pierre-Alain Fouque et al. 2017. *FALCON: Fast-Fourier Lattice-based Compact Signatures over NTRU*. Technical Report. National Institute of Standards and Technology.
[124] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo S. L. M. Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Kramer, Patrick Longa, Harun Polat, Jefferson E. Ricardini, and Gustavo Zanon. 2017. *qTESLA*. Technical Report. National Institute of Standards and Technology.
[125] Andreas Hülsing, Joost Rijneveld, John Schanck, and Peter Schwabe. 2017. High-speed key encapsulation from NTRU. In *CHES*.
[126] Jintai Ding, Tsuyoshi Takagi, Xinwei Gao, and Yuntao Wang. 2017. *Ding Key Exchange*. Technical Report. National Institute of Standards and Technology.
[127] Andreas Hülsing, Joost Rijneveld, John M. Schanck, and Peter Schwabe. 2017. *NTRU-HRSS-KEM*. Technical Report. National Institute of Standards and Technology.
[128] Daniel J. Bernstein, Chitchanok Chuengsatiansup, Tanja Lange, and Christine van Vredendaal. 2017. *NTRU Prime*. Technical Report. National Institute of Standards and Technology.
[129] Rachid El Bansarkhani. 2017. *KINDI*. Technical Report. National Institute of Standards and Technology.

[130] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. 2017. *SABER: Mod-LWR Based KEM*. Technical Report. National Institute of Standards and Technology.
[131] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2017. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive. (2017).
[132] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2017. *CRYSTALS-Dilithium*. Technical Report. National Institute of Standards and Technology.
[133] Özgür Dagdelen, Rachid El Bansarkhani, Florian Göpfert, Tim Güneysu, Tobias Oder, Thomas Pöppelmann, Ana Helena Sánchez, and Peter Schwabe. 2014. High-speed signatures from standard lattices. In *LATINCRYPT*.
[134] Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. 2015. Efficient Ring-LWE encryption on 8-bit AVR processors. (2015).
[135] Oscar Reparaz, Sujoy Sinha Roy, Ruan de Clercq, Frederik Vercauteren, and Ingrid Verbauwhede. 2016. Masking ring-LWE. *Journal of Cryptographic Engineering* (2016).
[136] Johannes Buchmann, Florian Göpfert, Tim Güneysu, Tobias Oder, and Thomas Pöppelmann. 2016. High-performance and lightweight lattice-based public-key encryption. In *IoTPTS*.
[137] Ye Yuan, Chen-Mou Cheng, Shinsaku Kiyomoto, Yutaka Miyake, and Tsuyoshi Takagi. 2016. Portable implementation of lattice-based cryptography using JavaScript. In *CANDAR*.
[138] Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. 2013. Software speed records for lattice-based signatures. In *PQCrypto*.
[139] Tobias Oder, Thomas Pöppelmann, and Tim Güneysu. 2014. Beyond ECDSA and RSA: Lattice-based digital signatures on constrained devices. In *DAC*.
[140] Ahmad Boorghany, Siavash Bayat Sarmadi, and Rasool Jalili. 2015. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. (2015).
[141] Thomas Poppelmann, Tobias Oder, and Tim Gijneysu. 2015. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. Cryptology ePrint Archive. (2015).
[142] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson. 2016. An efficient lattice-based signature scheme with provably secure instantiation. In *AFRICACRYPT*.
[143] Shay Gueron and Fabian Schlieker. 2016. Speeding up R-LWE post-quantum key exchange. Cryptology ePrint Archive. (2016).
[144] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. 2015. Lattice-based signatures: Optimization and implementation on reconfigurable hardware. (2015).
[145] J. Howe, C. Rafferty, A. Khalid, and M. O'Neill. 2017. Compact and provably secure lattice-based signatures in hardware. (2017).
[146] Hamid Nejatollahi, Nikil Dutt, Indranil Banerjee, and Rosario Cammarota. 2018. Domain-specific accelerators for ideal lattice-based public key protocols. Cryptology ePrint Archive, Report 2018/608. (2018).
[147] Po-Chun Kuo, Wen-Ding Li, Yu-Wei Chen, Yuan-Che Hsu, Bo-Yuan Peng, Chen-Mou Cheng, and Bo-Yin Yang. 2017. High performance post-quantum key exchange on FPGAs. (2017).
[148] Aydin Aysu, Bilgiday Yuce, and Patrick Schaumont. 2015. The future of real-time security: Latency-optimized lattice-based digital signatures. (2015).
[149] A. Aysu and P. Schaumont. 2016. Precomputation methods for hash-based signatures on energy-harvesting platforms. *TC* (2016).
[150] Jeffrey Hoffstein, Jill Pipher, William Whyte, and Zhenfei Zhang. 2017. A signature scheme from learning with truncation. Cryptology ePrint Archive. (2017).
[151] Nagarjun C. Dwarakanath and Steven D. Galbraith. 2014. Sampling from discrete Gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing* (2014).
[152] Shruti More and Raj Katti. 2015. Discrete Gaussian sampling for low-power devices. In *PACRIM*.
[153] Pavel Emeliyanenko. 2009. Efficient multiplication of polynomials on graphics hardware. In *APPT*.
[154] Victor Shoup. 2016. NTL: A library for doing number theory. (2016).
[155] Sedat Akleylek, Özgur Dağdelen, and Zaliha Yüce Tok. 2015. On the efficiency of polynomial multiplication for lattice-based cryptography on GPUs using CUDA. In *ICCISB*.
[156] Sedat Akleylek, Erdem Alkım, and Zaliha Yüce Tok. 2016. Sparse polynomial multiplication for lattice-based cryptography with small complexity. *The Journal of Supercomputing* (2016).
[157] Carlos Aguilar-Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. 2016. NFLlib: NTT-based fast lattice library. In *CT-RSA*.
[158] Patrick Longa and Michael Naehrig. 2016. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *CANS*.
[159] Aydin Aysu, Cameron Patterson, and Patrick Schaumont. 2013. Low-cost and area-efficient FPGA implementations of lattice-based cryptography. In *HOST*.

[160] Donald Donglong Chen, Nele Mentens, Frederik Vercauteren, Sujoy Sinha Roy, Ray C. C. Cheung, Derek Pao, and Ingrid Verbauwhede. 2015. High-speed polynomial multiplication architecture for ring-LWE and SHE cryptosystems. *TCS* (2015).
[161] Chaohui Du and Guoqiang Bai. 2016. A family of scalable polynomial multiplier architectures for Ring-LWE based cryptosystems. (2016).
[162] Tamás Györfi, Octavian Cret, and Zalán Borsos. 2013. Implementing modular FFTs in FPGAs – A basic block for lattice-based cryptography. In *DSD*.
[163] Chaohui Du and Guoqiang Bai. 2016. Towards efficient polynomial multiplication for lattice-based cryptography. In *ISCAS*.
[164] Chaohui Du and Guoqiang Bai. 2016. Efficient polynomial multiplier architecture for Ring-LWE based public key cryptosystems. In *ISCAS*.
[165] Chaohui Du, Guoqiang Bai, and Xingjun Wu. 2016. High-speed polynomial multiplier architecture for ring-LWE based public key cryptosystems. In *GLSVLSI*.
[166] Nick Howgrave-Graham et al. 2003. NAEP: Provable security in the presence of decryption failures. Cryptology ePrint Archive. (2003).
[167] Daniel J. Bernstein. 2008. New stream cipher designs. Chapter The Salsa20 Family of Stream Ciphers.
[168] 2009. IEEE standard specification for public key cryptographic techniques based on hard problems over lattices. *IEEE Std 1363.1-2008* (2009).
[169] El Bansarkhani Rachid. 2017. LARA: A design concept for lattice-based encryption. Cryptology ePrint Archive. (2017).
[170] Chris Peikert. 2014. Lattice cryptography for the Internet. In *PQCrypto*.
[171] Scott Fluhrer. 2016. Cryptanalysis of Ring-LWE based key exchange with key share reuse. Cryptology ePrint Archive. (2016).
[172] Matt Braithwaite. 2016. Experimenting with post-quantum cryptography. (2016).
[173] Daniel J. Bernstein. 2008. ChaCha, a variant of Salsa20. In *SASC*.
[174] Morris J. Dworkin. 2015. *SHA-3 Standard: Permutation-based Hash and Extendable-output Functions*. Technical Report.
[175] Douglas Stebila and Michele Mosca. 2016. Post-quantum key exchange for the Internet and the Open Quantum Safe Project. Cryptology ePrint Archive. (2016).
[176] Alexander W. Dent. 2003. A designer's guide to KEMs. In *Cryptography and Coding*, Kenneth G. Paterson (Ed.).
[177] Vadim Lyubashevsky and Daniele Micciancio. 2009. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *CRYPTO*.
[178] Gu Chunsheng. 2017. Integer version of Ring-LWE and its applications. Cryptology ePrint Archive. (2017).
[179] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. 2013. Keccak. In *EUROCRYPT*.
[180] Ahmad Boorghany and Rasool Jalili. 2014. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. Cryptology ePrint Archive. (2014).
[181] Daniele Micciancio and Chris Peikert. 2012. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*.
[182] John Kelsey. 2016. SHA-3 derived functions: cSHAKE, KMAC, TupleHash, and ParallelHash. *NIST Special Publication* (2016).
[183] Nina Bindel, Johannes Buchmann, Juliane Kramer, Heiko Mantel, Johannes Schickel, and Alexandra Weber. 2017. Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. Cryptology ePrint Archive. (2017).
[184] Nina Bindel, Johannes Buchmann, and Juliane Kramer. 2016. Lattice-based signature schemes and their sensitivity to fault attacks. (2016).
[185] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. 2017. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. Cryptology ePrint Archive. (2017).
[186] Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. 2014. Efficient identity-based encryption over NTRU lattices. In *ASIACRYPT*.
[187] Léo Ducas and Thomas Prest. 2016. Fast Fourier orthogonalization. In *ISSAC*.
[188] Damien Stehlé and Ron Steinfeld. 2011. Making NTRU as secure as worst-case problems over ideal lattices. In *EUROCRYPT*.
[189] Jeff Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, and William Whyte. 2014. Transcript secure signatures based on modular lattices. In *PQCrypto*.

Received November 2017; revised October 2018; accepted November 2018