UNIVERSITY OF CALIFORNIA,
IRVINE

Programmable Accelerators for Lattice-based Cryptography

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Hamid Nejatollahi

Dissertation Committee:
Professor Nikil Dutt, Chair
Professor Rainer Döemer
Professor Ian G. Harris

2020

DEDICATION

To my parents, brothers, and my sister

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I am grateful to my advisor, professor Nikil Dutt for his patience, support, and guidance.

I would like to thank Professor Rainer Döemer and Professor Ian G. Harris for their participation in the thesis committee and providing helpful feedback.

I feel fortunate to receive mentorship from Dr. Rosario Cammarota, principal researcher at Intel Labs. He was always a source of inspiration and motivation for me.

I would like to shout out to my colleagues, Hossein Tajik, Majid Shoushtari, JurnGyu Park, Bryan Donyanavard, Tiago Mück, Chen-Ying Hsieh, Kasra Moazzemi, Sina Shahhosseini, Sina Labbaf, Biswadip Maity, Delaram Amiri, Emad Kasaeyan Naeini, Michael (Tao-Yi) Lee, Kenneth Stewart, Saehanseul Yi, Caio Batista de Melo, Jiyoung An, Khuong Vo, Amirhosein Mohaddesi, and Sujin Kang for their friendship and for sharing the pain.

Finally, my work would not have been possible without funding from the Electrical Engineering and Computer Science department as well as a gift from Qualcomm Technology Inc., or permission from the ACM and IEEE to include content from my previously published work in [160, 164, 163, 167, 165, 166].

CURRICULUM VITAE

Hamid Nejatollahi

EDUCATION

Doctor of Philosophy in Computer Science — **2020**
University of California, Irvine — *Irvine, CA*

Master of Science in Computer Engineering — **2014**
University of Tehran — *Tehran, Iran*

Bachelor of Science in Computer Engineering — **2012**
University of Tehran — *Tehran, Iran*

RESEARCH EXPERIENCE

Graduate Student Researcher — **2015–2020**
University of California, Irvine — *Irvine, CA*

TEACHING EXPERIENCE

Teaching Assistant — **2015–2020**
University of California, Irvine — *Irvine, CA*

INDUSTRY EXPERIENCE

Embedded Engineer — **Jul. 2014–Jul. 2015**
Iranian Embedded Systems (IES) — *Tehran, Iran*

Software Engineering Intern — **Sep. 2013–Jul. 2014**
Persian Architecture Systems — *Tehran, Iran*

REFEREED CONFERENCE PUBLICATIONS

CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware **2020**
ACM/IEEE Design Automation Conference (DAC)

Exploring Energy Efficient Quantum-resistant Signal Processing Using Array Processors **2020**
IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)

Flexible NTT Accelerators for RLWE Lattice-based Cryptography **2020**
IEEE International Conference on Computer Design (ICCD)

Domain-specific Accelerators for Ideal Lattice-based Public Key Protocols **2018**
Cryptology ePrint Archive, Report 2018/608

Trends, Challenges and Needs for Lattice-Based Cryptography Implementations **2018**
ACM/IEEE International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)

REFEREED JOURNAL PUBLICATIONS

Synthesis of Flexible Accelerators for Early Adoption of Ring-LWE Post-quantumCryptography," **2020**
ACM Transactions on Embedded Computing Systems (TECS)

Post-quantum Lattice-based Cryptography Implementations: A Survey **2019**
ACM Computing Surveys (CSUR)

ABSTRACT OF THE DISSERTATION

Programmable Accelerators for Lattice-based Cryptography

By

Hamid Nejatollahi

Doctor of Philosophy in Computer Science

University of California, Irvine, 2020

Professor Nikil Dutt, Chair

Advances in computing steadily erode computer security at its foundation, calling for fundamental innovations to strengthen the weakening cryptographic primitives and security protocols. While many alternatives have been proposed for symmetric key cryptography and related protocols (e.g., lightweight ciphers and authenticated encryption), the alternatives for public key cryptography are limited to post-quantum cryptography primitives and their protocols. In particular, lattice-based cryptography is a promising candidate, both in terms of foundational properties, as well as its application to traditional security problems such as key exchange, digital signature, and encryption/decryption. At the same time, the emergence of new computing paradigms, such as Cloud Computing and Internet of Everything, demand that innovations in security extend beyond their foundational aspects, to the actual design and deployment of these primitives and protocols while satisfying emerging design constraints such as latency, compactness, energy efficiency, and agility.

In this thesis, we propose a methodology to design programmable hardware accelerators for lattice-based algorithms and we use the proposed methodology to implement flexible and energy efficient post-quantum cache- and DMA-based accelerators for the most promising submissions to the NIST standardization contest. We validate our methodology by integrating our accelerators into an HLS-based SoC infrastructure based on the X86 processor

and evaluate overall performance. In addition, we adopt the systolic architecture to accelerate the polynomial multiplication, which is the heart of a subset of LBC algorithms (i.e., ideal LBC), on the field programmable gate arrays (FPGAs). Finally, we propose a high-throughput Processing In-Memory (`PIM`) accelerator for number theoretic transform (`NTT`-) based polynomial multiplier.

Chapter 1

Introduction

A worldwide research effort is currently pursuing a scalable quantum computer. Quantum computers, as envisioned by Richard Feynman [90], use quantum mechanics to carry out computation that can reach an exponential speedup in several applications, such as quantum chemistry [172]. Exponential speed up can also be reached in other problems, such as the search for the period of a function. As demonstrated by Peter Shor [208], it would allow us to factorize integers in polynomial time and ultimately render the public key infrastructure insecure. Shore's algorithm needs 6146 and 2330 Qubits to break ECC-256 and RSA-3072, respectively, in 3848 seconds computing the elliptic curve discrete logarithms in $E(F_p)$ and factoring an RSA modulus `N` needs 6146 and 2330 Qubits for RSA-3972 and ECC-256, respectively [193].

The scientific community is investing significant efforts to address the problem in a timely and effective way. Post-quantum cryptography is a vibrant area of research devoted to studying alternative public-key algorithms, executed on classical computers, but capable of withstanding quantum computational power. In December 2018, the US president signed H.R. 6227 to fund the National Quantum Initiative Act (NQI)[1]. The law authorizes 1.2 billion

[1] `https://fas.org/sgp/crs/misc/R45409.pdf`

to be invested in quantum information science over five years. NQI funding will go to the National Institute of Standards and Technology (NIST), National Science Foundation (NSF) Multidisciplinary Centers for Quantum Research and Education and to the Department of Energy Research and National Quantum Information Science Research Centers. The effort of governmental agencies in conducting the standardization of these novel algorithms endorses the urgency of the threat. The most relevant standardization effort is the NIST one, that started in 2017 with the submissions of potential candidates, and continues with the evaluation of the candidates for the next six to seven years, until the selection of the new standards. During the evaluation phase, the proposed algorithms will be analyzed and compared using various parameters, including the security and efficiency of software and hardware implementation.

Among the post-quantum cryptography families, the family of lattice-based cryptography (LBC) appears to be gaining acceptance. Its applications are proliferating for both traditional security problems (e.g., key exchange and digital signature), as well as emerging security problems (e.g., homomorphic schemes, identity-based encryption and even symmetric encryption). Lattice-based cryptographic primitives and protocols provides a rich set of primitives which can be used to tackle the challenges posed by deployment across diverse computing platforms, e.g., Cloud vs. Internet-of-Things (IoT) ecosystem, as well as for diverse use cases, including the ability to perform computation on encrypted data, providing strong (much better understood than before) foundations for protocols based on asymmetric key cryptography against powerful attackers (using Quantum computers and algorithms), and to offer protection beyond the span of traditional cryptography. Indeed, lattice-based cryptography promises to enhance security for long-lived systems, e.g., critical infrastructures, as well as for safety-critical devices such as smart medical implants [98].

1.1 Trends, Challenges and Needs for Lattice-based Cryptography Implementations

The emergence of new computing platforms, such as cloud Computing, software defined networks and Internet of Everything, demands the adoption of an increasing number of security standards, which in turn requires the implementation of a diverse set of cryptographic primitives, but this is only part of the story. At the computing platform level, we face a diversity of computing capability. On one end of the spectrum, in the cloud computing and software defined network space, applications demand high-performance, and energy efficiency of cryptographic implementations. This calls for the development of programmable hardware capable of running not only individual cryptographic algorithms, but full protocols efficiently, with the resulting challenge of designing for agility, e.g., designing computing engines that achieve the efficiency of Application-Specific Integrated Circuits (ASICs), while retaining some level of programmability. On the other end of the spectrum, in the IoT space, implementations of standardized cryptography to handle increased key sizes become too expensive in terms of cost, speed, and energy, but are necessary, e.g., in the case of long lived systems such as medical implants and image encryption algorithms [15]. In part, this demands the development of new and strong lightweight alternatives to symmetric key cryptography as well [14]. Furthermore, given the variety of applications and their interplay with the cloud, even in this case agility in the implementation becomes a key requirement. 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.

Figure 1.1 illustrate the taxonomy of the state-of-the-art accelerator architecture space. Compared to instruction-level accelerators, kernel-level accelerators implement large with more semantically rich functions of computation (e.g., Keccak and polynomial multiplica-

	Part of the Pipeline	Attached to Cache	Attached to the Memory Bus	Attached to the I/O Bus
Instruction-Level	FPU, SIMD, DySER	Hwacha, CHARM		
Kernel-Level	NPU, 10x10, H.264	DANA , SNNAP, C-Cores, **This Work: TECS'20**	AccStore, **This Work: 1-ICCD'19 2-ICASSP'20**	**This Work: DAC'20**
Application-Level	Oracle/Cavium **x86 AES, Crypto Acc,**	Key-Value Stores, Memcached	Sonic3D, IBM Power7+, HARP,	GPU, IBM Power, Catapult

Figure 1.1: Accelerator taxonomy [205].

tion) that leads to less flexibility [205]. Application level accelerators compute the whole application with the superior performance but the lowest flexibility. In the order dimension, accelerators can be tightly-coupled (i.e., part of the pipeline) or loosely-coupled to the processor. The more loose the connection to the CPU is, the more flexibility and lower performance are expected.

Cryptographic accelerators, such as X86 AES, are typically tightly-coupled application-level co-processors. However, the early adoption of quantum-resistant hardware is challenging because in this period of transition, during which candidates are still under the evaluation, a post-quantum standard is yet to be decided and also the parameters of the candidates themselves can change; subsequently, it is troublesome to commit to a specific algorithm and a specific set of parameters, knowing that they could be soon outdated [164].

The situation would change if the accelerators would be able to meet two requirements: 1) reach the required performance and 2) be flexible, thus "robust" to change in parameters

and algorithms (providing the so-called crypto-agility). To meet these goals, we propose to accelerate, instead of a complete algorithm, only several portions of the software routines, called micro-kernels. Compared to a whole dedicated accelerator, a micro-kernel accelerator does not reach the same level of performance, because the main processor executes parts of the computation. Subsequently, hardware acceleration based on micro-kernels is more flexible compared to the classical counterparts.

As shown in red , Figure 1.1, we design loosely-coupled kernel-level accelerators which are suitable for LBC schemes because the common resource-intensive kernels consumes most of the execution cycles and power. Figure 1.2 shows the profiling results for the most promising LBC schemes in the server-side (sign) client (verify). NewHope and Kyber are key encapsulation mechanisms (`KEM`) while Dilithium and qTESLA are digital signatures. `NTT_CT`, `NTT_GS`, and `Keccak` consume most of the cycles over different schemes for which we create flexible accelerators.

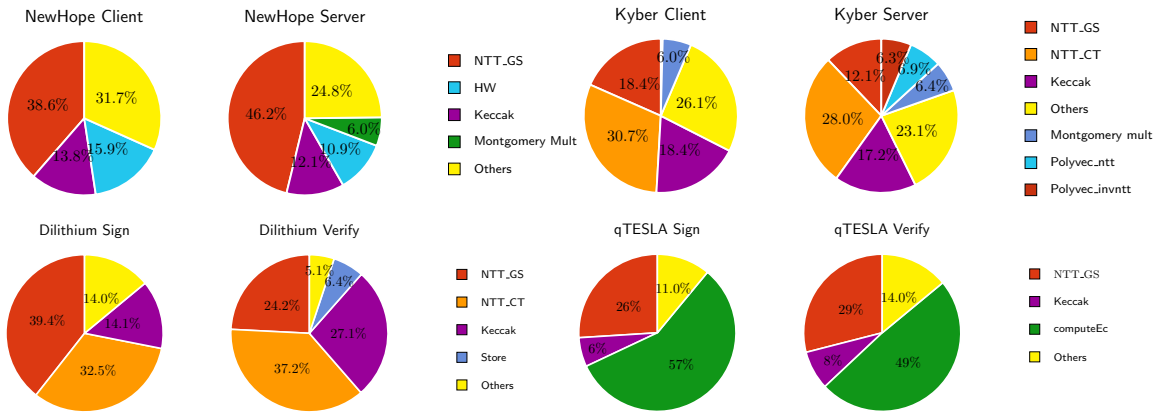


Figure 1.2: Profiling results for the most promising LBC schemes. Profiling results are based on the user-level CPU cycles

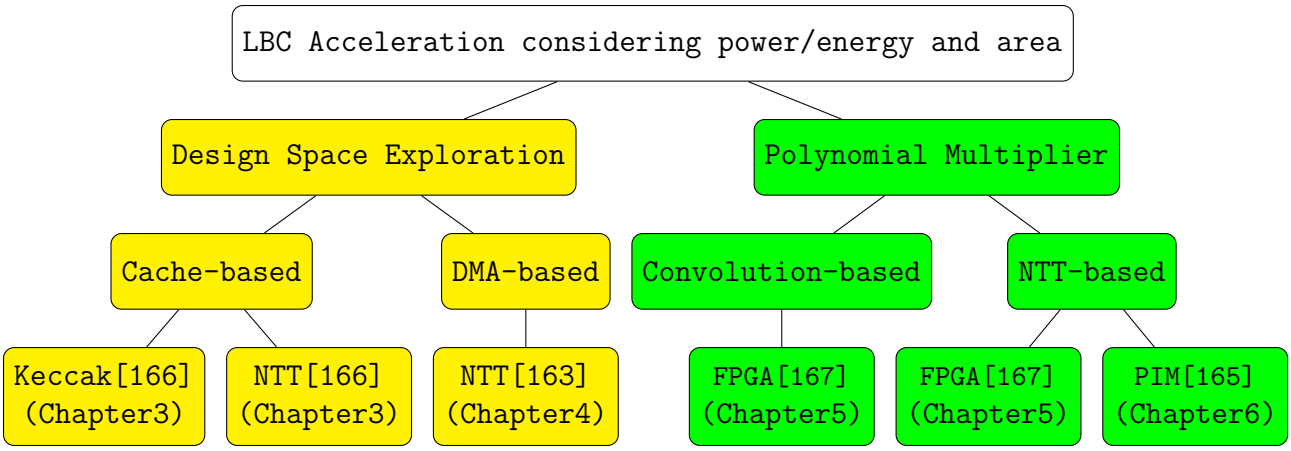


Figure 1.3: Thesis Contribution

1.2 Thesis Contributions

Figure 1.3 divides our contribution into two wide parts: 1) design space exploration (DSE) of accelerators and verify the design in system on chip (SoC) for numerous LBC schemes. 2) Hardware acceleration of polynomial multiplier.

We focus on domain-specific programmable hardware accelerators because they are the most suitable, at this stage, to adopt post-quantum algorithms in hardware. We concentrate on lattice-based algorithms, the most promising family of quantum-resistant algorithms thanks to their versatility and their excellent performance. Around half of the submission to the first round of the NIST PQC Standardization Process belong to this family. After a year of evaluation, 26 of the candidates are chosen for the second round; twelve candidates belong to lattices, and the remaining 14 candidates belong to code-based, hash-based, supersingular elliptic curve isogeny, and multivariate cryptography [164].

1.2.1 Design Space Exploration

As stated earlier, generation of domain specific accelerators involves considering the spectrum of the computing platforms which ranges from resource-constrained IoT devices to

performance-driven cloud servers. Therefore, explore the security, performance, area, power, and energy trade-offs of the target kernels seems to be essential. In this dissertation, we adopt a Pre-RTL simulation framework to explore the design space of micro-kernel-based accelerators for a subset of (`LBC`) algorithms (i.e., ideal LBC) and implement the identified accelerators. Based on Figure 1.2, we explore design space of the below resource-intensive kernels:

1. **Keccak**. Keccak is a family of cryptographic primitives primarily as a candidate for the NIST SHA-3 competition [34]. The main transformation in the family is the Keccak-f[1600] permutation which is used in the sponge mode in the specification for the SHA-3 hash function. The hash function operates on a state of 1600 bits and is designed for easy implementation in both software and hardware.

2. **Number theoretic transform (`NTT`)**. Polynomial multiplication is usually performed using the NTT, a variant of the Discrete Fourier Transform (DFT) for polynomial rings on finite arithmetic fields [160]. The process is slow due to the repeated use of expensive operations like multiplication and modulo reduction. Two common algorithms of performing NTT are Cooley-Tukey (`NTT_CT`) [61], produces the result in the bit-reverse order by receiving the input in the correct order, and Gentleman-Sande (`NTT_GS`) [100], receives the input in the reverse order and produces the output in the correct order. As can be seen in Figure 1.2, and `NTT_GS` and `NTT_CT` are the most resource-intensive kernels in the ideal LBC schemes.

We design cache-assisted [166] (Chapter3) and DMA-assisted [163] (Chapter4) accelerators and evaluate them into an HLS-based SoC infrastructure based on the X86 processor and evaluate overall performance, power, and energy.

1.2.2 Hardware Acceleration of Polynomial Multiplier

Polynomial multiplication is the most time-consuming routine int ideal LBC schemes. We design accelerators for two common methods to compute the polynomial multiplication.

1. **Convolution-based multiplier**. The easiest method to multiply two polynomials is to use the convolution (Schoolbook [135]) with the time complexity of $O(n^2)$. We adopt the systolic architecture [137] to accelerate the polynomial multiplication and implement on Zynq UltraScale+ FPGA [167] (Chapter5). Our design is 22x faster than the state-of-the-art FGPA implementation of the polynomial multiplier in the NewHope-Simple key exchange mechanism on a low-cost Artix7 FPGA [223] for polynomial degree n=1024.

2. **NTT-based multiplier**. Polynomial multiplier is usually implemented by the Number Theoretic Transform (NTT), which drops the time complexity of the polynomial multiplier from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot \log n)$. We present FPGA-based [167] and processing in memory (`PIM-`) based [165] in Chapter5 and Chapter6, respectively.

   - **FPGA-based NTT**. We design and implement two NTT-based polynomial multipliers on FPGAs.
     - `Seq_NTT`. Sequential NTT-based multiplier which consists of a single butterfly operator and perform the polynomial multiplication in serial. It achieves 3x speedup over the state-of-the-art on a low-cost Artix7 FPGA [223] for polynomial degree n=1024.
     - `SA_NTT`. When synthesized on a Zynq UltraScale+ FPGA, the *NTT-based* systolic array design (`SA_NTT`) achieve on average 1.7x speedup over our sequential NTT-based multiplier which means it is 5x faster than the state-of-the-art [223] for polynomial degree n=1024.

.

- **PIM-based NTT**. We design a high throughput PIM-based NTT that
  - achieves 31x throughput improvement with the same energy and 28% latency reduction compare to the our NTT on FPGA [167] (which itself is 3x faster than the state-of-the-art [223] for polynomial degree n=1024).
  - provides 5x throughput improvement with the 20x energy reduction, but 6x latency increase compare to Convolution-based multiplier on FPGA [167].
  - enables fast execution of the polynomial multiplication with the support of polynomials with degrees up to 32k, accommodating requirements for public key cryptographic systems for data at rest and in communication, and data in use, i.e., homomorphic encryption.

The rest of the thesis is organized as follows: Chapter 2 provides the background for `LBC` primitives and surveys the `LBC` implementations; Chapter 3 and Chapter 4 detail the designed cache- and DMA-based accelerators, respectively. Chapter 5 discusses the FPGA-based polynomial multipliers. Chapter 6 introduces the PIM-based polynomial multiplier. Finally, we conclude the thesis in Chapter 7.

Chapter 2

Background and Related Works

Lattice-based cryptographic primitives and protocols provides a rich set of primitives which can be used to tackle the challenges posed by deployment across diverse computing platforms, e.g., Cloud vs. Internet-of-Things (IoT) ecosystem, as well as for diverse use cases, including the ability to perform computation on encrypted data, providing strong (much better understood than before) foundations for protocols based on asymmetric key cryptography against powerful attackers (using Quantum computers and algorithms), and to offer protection beyond the span of traditional cryptography. Indeed, lattice-based cryptography promises to enhance security for long-lived systems, e.g., critical infrastructures, as well as for safety-critical devices such as smart medical implants [98]. In this chapter, we review the foundations of lattice-based cryptography, some of the more adopted instances of lattices in security, their implementations in software and hardware, and their applications to authentication, key exchange and digital signatures.

2.0.1 Arithmetic and Components of Lattices

In this section, we provide an evaluation of the components in a lattice-based cryptosystem that guides the actual implementation. To components are critical: (a) the Polynomial multiplication for ideal lattices, and matrix multiplication for standard lattice are the main speedup bottlenecks; (b) The discrete Gaussian sampling is used to sample noise in order to hide the secret information. There are various algorithms for the sampler and multiplier in the literature, by providing the designer with a specific goal [160]. We briefly review different algorithms and outline their practical implementations in Section 2.1.1

There exist two main classes of lattice-based algorithms used in cryptography, namely `NTRU` and Learning with Error (`LWE`). The security of `NTRU` is based on hardness not-provably reducible to solving the Closest Vector Problem (`CVP`) in a lattice, whereas the security of `LWE` relies on provably reducible solving the Shortest-Vector Problem (`SVP`) in a lattice. Consequently, `NTRU` suffers from security guarantees, but in practice provides more flexibility and efficiency in the implementation. On the contrary, `LWE` problems are resistant to quantum attacks, while their relatively inefficient nature led researchers to devise more efficient formulations, e.g., over rings - Ring Learning with Errors (`Ring-LWE`).

Implementations are broadly classified in pure software, pure hardware, and hardware/software co-design cryptographic engines [160]. Implementation of the modulo arithmetic (multiplication and addition of big numbers) is a bottleneck in `LBC`. For Standard `LWE` schemes, matrix multiplication algorithms are adopted, whereas number theoretic transform (`NTT`) is a better choice for polynomial multiplication in `Ring-LWE`. A summary of modular arithmetic is presented in Figure 2.1.

In addition to the arithmetic portion, a bottleneck in lattice-based schemes is the extraction of the random term, which usually is implemented with a discrete noise sampler (from a discrete Gaussian distribution) and can be done with rejection, inversion, Ziggurat, or

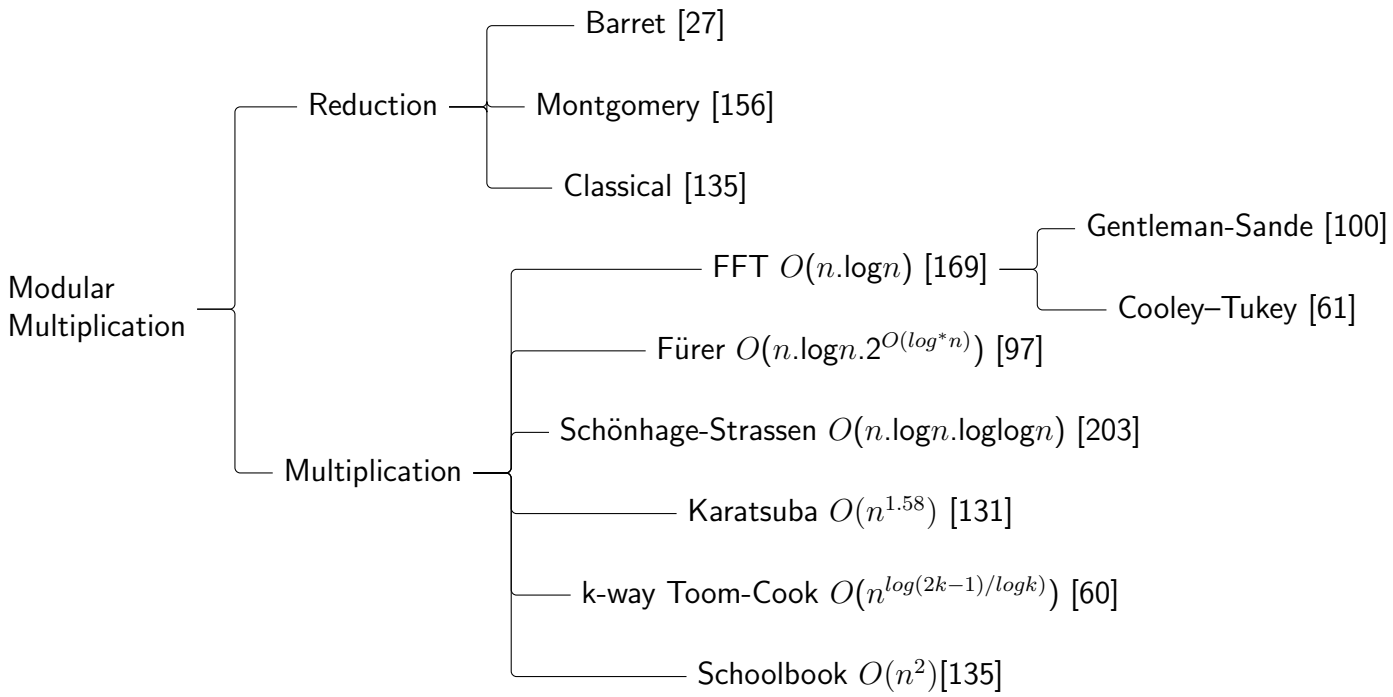


Figure 2.1: Common modular Multiplication algorithms

Knuth-Yao sampling with moderate standard deviation for key exchange and public key encryption, and small standard deviation for digital signature to achieve a compact and secure signature.

When implemented, Standard `LWE-based` schemes exhibit a relatively large memory footprint due to large key size - hundreds of kilobyte for the public key - which render 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` [151], making `Ring-LWE` an excellent candidate for resource-constrained devices, such as Wi-Fi capable smart devices, including medical implants. Another avenue to address resource constrained devices is that memory footprint can be traded off with security assurance, which improves both efficiency and memory consumption.

High-performance Intel/AMD processors, which are notoriously equipped with AVX vector instructions, and ARM/AVR micro-controllers are popular platforms for software im-

plementations, as we will see in more detail in the coming sections. Practical software implementations of standard lattices, encryption scheme [122] and key exchange [42], have been published recently. For hardware implementations, FPGAs provide flexibility and customization but not agility. Hardware implementations of lattice-based schemes (e.g., BLISS-I [79] with higher security level) are about an order of magnitude faster than the hardware implementation of RSA-2048 [123]. Furthermore, another candidate platform to implement lattice-based schemes is application specific integrated circuits (ASICs) of which there appear to be no such implementations in the literature at this time. However, the main advantages and challenges for ASIC design of lattice-based schemes are presented in [170].

2.0.1.1 The multiplier component

Matrix multiplication is used for the standard lattices, while polynomial multiplication is employed for ideal lattices. Arithmetic operations for a `Ring-LWE` based scheme are performed over a ring of polynomials. The most time and memory consuming part is the polynomial multiplication. The easiest way to multiply two polynomials is to use the Schoolbook algorithm with the time complexity of $O(n^2)$ [135]. Let n and p be degree of the lattice (n is a power of 2) and a prime number ($p = 1 \bmod 2n$), respectively. Z_p denotes the ring of integers modulo p and x^{n+1} is an irreducible degree n polynomial. The quotient ring R_p, contains all polynomials with the degree less than n in Z_p, that defines as $R_p = Z_p/[x^{n+1}]$ in which coefficients of polynomials are in the range [0,p).

The number theoretic transform (`NTT`) is a generalization of Fast Fourier Transform (`FFT`), which is carried out in a finite field instead of complex numbers. The latter could achieve time complexity of $O(n\log n)$. In other words, exp(-2πj/N) with n^{th} primitive root of unity ω_n which is defined as the smallest element in the ring that $\omega_n^n = 1 \bmod p$ and ${\omega_\mathrm{n}}^i \neq 1 \bmod p$ for i $\neq$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 $O(n\log n)$.

$$a(x) \times b(x) = NTT^{-1}(NTT(a) \odot NTT(b)) \tag{2.1}$$

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

$$(\hat{a_0}, ..., \hat{a_{n-1}}) = NTT(a_0, ..., a_{n-1}) \tag{2.2}$$

$$\hat{a_i} = \sum_{j=0}^{n-1} a_i \omega^{ij} mod(p), i = 0, 1, ..., n-1 \tag{2.3}$$

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

$$a_i = \sum_{j=0}^{n-1} \hat{a}_i \omega^{-ij} mod(p), i = 0, 1, ..., n-1 \tag{2.4}$$

In order to compute NTT(a), we pad the vector of coefficients with n zeros that leads to doubling the input size. Negative wrapped convolution technique [227], avoids doubling the input size.

To improve efficiency of polynomial multiplications with `NTT`, combining multiplications of powers of ω with powers of ψ and ψ^{-1} ($\psi^2 = \omega$) is beneficial which requires storage mem-

ory for precomputed powers of ω and ψ^{-1} in bit-reversed order [196, 185, 145]. NTT can be used only for the $p = 1 \pmod{2\ n}$ case where p is a prime integer. Suppose $a' = (a_0, \psi a_1, ..., \psi^{n-1} a_{n-1})$, $b' = (b_0, \psi b_1, ..., \psi^{n-1} b_{n-1})$, $c' = (c_0, \psi c_1, ..., \psi^{n-1} c_{n-1})$ to be coefficient vectors of the a, b, c that are multiplied component-wise by $(1, \psi^1, ..., \psi^{n-1})$. Based on the negative wrapped convolution theorem, modulo $(x^n + 1)$ is eliminated and the degree of NTT and NTT^{-1} is reduced from $2n$ to n.

$$c' = NTT^{-1}(NTT(a') \odot NTT(b')) \tag{2.5}$$

$$c = (\psi^0 {c'}_0, \psi^{-1} {c'}_1, ..., \psi^{-n+1} {c'}_{n-1}) \tag{2.6}$$

Two common algorithms to compute NTT are Cooley-Tukey butterfly (`CT`) [61] and Gentleman-Sande butterfly (`GS`) [100]. `CT`, decimation-in-time, outputs the result in the bit-reverse order by getting the input in the the correct order. `GS`, decimation-in-frequency, receives the input in the reverse order and produces the output in the correct order [145]. Employing `GS` to compute both NTT and NTT^{-1} involves in bit-reverse calculation [11]; however, bit-reverse step can be avoided by using `CT` for NTT and `GS` for NTT^{-1} [43, 145].

Other popular multiplication algorithms in literature are the Karatsuba algorithm (with time complexity of $O(n^{log3/log2})$ [131]), and subsequent variants of it [65]. Schönhage-Strassen with time complexity of $O(n.\log n.\log\log n)$ [203] outperforms the Karatsuba algorithm [52].

An extensive analysis and comparison for hardware complexity of various modular multiplications, including Schoolbook (classical), Karatsuba, and FFT, with different operand

size, are presented in [65]. Authors calculate hardware complexity of each multiplier by decomposing it into smaller units such as the full adder, half adder, multiplexer, and gate. Rafferty et al. [189] adopt the same approach to analyze large integer multiplications of both combined and individual multipliers. Karatsuba multiplier outperforms for operands greater or equal to 32 bits. Schoolbook imposes large memory footprint in order to store partial products which negatively impact performance that is mitigated by Comba [59] with the same time complexity but relaxing memory addressing by optimizing the sequence of partial products. Rafferty et al. compare hardware complexity, in terms of $+$, $-$, and $*$ units, of different combinations of classical (Comba), Karatsuba, and FFT (NTT for integers) for up to multipliers with 65536-bit operands. However, they evaluate the latency and clock frequency by implementing in hardware (Xilinx Virtex-7 FPGA) for up to 256-bits for the combination of NTT+Comba and 4096-bit for Karatsuba+Comba which are not in a good range for lattice-based cryptography (e.g., $1024 \times 14 = 14336$ bits are used for NewHope key exchange). Based on their (analytical) hardware complexity analysis, the combination of Karatsuba-Schoolbook is the best choice for operands under 64 bits. Karatsuba-Comba is preferable for operands for 64 bits to 256 bits. For larger operands, the lowest hardware complexity is achieved by combined multiplier NTT-Karatsuba-Schoolbook. It should be mentioned that results are for a single multiplication. Authors make some assumption for the sake of simplicity such as the shift operation is free and inputs are ready.

2.0.1.2 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 secu-

rity and efficiency of the sampler. For instance, a smaller standard deviation decreases the memory footprint required to store precomputed tables. For encryption/decryption schemes σ=3.33 [143] is suggested. Digital signature sampling from a Gaussian sampler involves large σ=215 [79]. However, by employing Peikert's convolution lemma [173], the standard deviation can be reduced by order of magnitude which is a remarkable improvement on the precomputed table size. Speed and memory footprint is λ dependent, i.e., higher λ results in more secure but a slower and bigger sampler. The tail of the Gaussian distribution touches the x-axis at x=$+\infty$ (considering only the positive side due to the symmetry) with negligible probability. Tail-cut parameter (τ) defines the amount of the distribution that we would like to ignore, hence random number e is sampled in $|e| \in \{0, \sigma \times \tau\}$ instead $|e| \in \{0, \infty\}$.

Instead of the statistical distance, Rényi divergence technique [21] can be employed to measure the distance between two probability distributions (e.g., [11]). More precisely, in lattice-based cryptography, Rényi divergence is used to generalized security reductions. Sampling the discrete Gaussian distribution is one the most time and memory hungry parts of lattice-based cryptosystems, due to the demands of high precision, many random bits, and huge lookup tables. To be more specific, the obligatory negligible statistical distance between the implementation of a Gaussian sampler (approximated) and the theoretical (perfect) discrete Gaussian distribution imposes expensive precise floating point arithmetic (to calculate the exponential function) or large memory footprint (to store precomputed probabilities). To keep statistical distance less than $2^{-\lambda}$, floating point precision with more than standard double-precision is obligatory which is not natively supported by the underlying platform; thus software libraries should be used to perform higher floating-point arithmetic. It is impractical to sample from a perfect Gaussian sampler; hence a λ-bit uniform random integer is used to approximate the discrete sampler. Fortunately, it is proven that the Gaussian sampler could be used to achieve $\lambda/2$ security level (approximated sampler) instead of λ level (perfect sampler), since (to date) there is no algorithm that can distinguish between a perfect sampler (λ bits) and an approximate sampler ($\lambda/2$ bits) [198]. In other words,

we can cut half of the bits in the sampler which results in a smaller and faster sampler [122, 199, 26]. Reduction in precision parameter (from λ to $\lambda/2$) changes tail-cut parameter (τ) as $\tau = \sqrt{\lambda \times 2.ln(2)}$ [121]. 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 [155]. Folláth provides a survey of different Gaussian samplers in lattice-based cryptography schemes with a more mathematical outlook [92]. Gaussian samples are classified into six categories and guidelines provided for choosing the best candidate on different platforms for specific parameter ranges. However, we organize Gaussian samplers into the following types discussed below. A Summary of advantage and disadvantages of each sampler are listed in Table 2.1.

Rejection Sampler

Firstly, x is sampled in (-$\tau\sigma$,$\tau\sigma$) uniformly at random where τ and σ are tail-cut and standard deviation of Gaussian distribution. After that, x is rejected with the probability proportional to 1-exp(-$x^2/2\sigma^2$). The high rejection rate of samples (on average eight trials to reach acceptance) along with expensive calculation of `exp()` are the main reasons for the inefficiency [225]. Göttert et al. [103] employ rejection sampler for the first time within `LBC`. Remarkable speed and area improvement could be achieved by accomplishing rejection operation using lazy floating-point arithmetic [83].

Bernoulli Sampler

Bernoulli is an optimized version of rejection sampling in order to reduce average required attempts for a successful sampler from 10 to around 1.47 with no need to calculate `exp()` function or precomputed tables [178]. Bernoulli is introduced in [79] for lattice-based cryptography and used widely in the research community [183, 180, 122]. The main idea be-

hind Bernoulli sampling is to approximate sampling from $D_{\mathbb{Z},k\sigma_2}$ using the distribution $k \cdot D_{\mathbb{Z}^+,\sigma_2} + \mathbb{U}(\{0,...,k-1\})$ where $\mathbb{U}$ is the uniform distribution. The procedure for Bernoulli sampler is shown below in 5 steps.

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

The standard deviation of the target Gaussian sampler $D_{\mathbb{Z},k\sigma_2}$, equals $k\sigma_2$ where $\sigma_2 = \sqrt{\frac{1}{2ln2}} \approx 0.849$ where standard deviation of the binary Gaussian sampler $D_{\mathbb{Z},\sigma_2}$ 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 [183], while for digital signatures with large standard deviation, using Gaussian distribution is mandatory [180]. Gaussian distribution can be replaced with other distributions (e.g., uniform distribution [105]). Bernoulli approach avoids long integer calculation and employs single bit operations that make it beneficial for hardware implementation. The time dependency of Bernoulli makes it vulnerable to timing attacks that are resolved in the hardware implementation of BLISS in [180]. Precomputed tables in Bernoulli sampling are small, besides binary Gaussian distribution (easy to sample intermediate sampler) is independent of σ hence the Peikert convolution lemma (smaller σ) does not have a considerable effect on the area. How-

ever, convolution lemma reduces the area of precomputed tables in CDT sampler by a factor of 23× for `BLISS` (reduces σ from 215 to 19.47).

Binomial Sampler

Centered Binomial distribution (ψ_k) is a close approximation of rounded Gaussian sampler (ξ_σ) that eliminates the need for computing $exp()$ and precomputed large tables. Let $\sigma = \sqrt{8}$ be the standard deviation of ξ_σ and a binomial distribution is parameterized with $k = 2\sigma^2$; choosing ψ_k as the sampling distribution has negligible statistical difference with rounded Gaussian sampler with $\sigma = \sqrt{8}$ [11]. Centered Binomial distribution(ψ_k) for integer $k \geq 0$ is defined as sampling $2 \cdot k$ random numbers uniformly from $\{0, 1\}$ as $(a_1, ..., a_k, b1, ..., b_k)$ and output $\sum_{i=1}^{k}(a_i, b_i)$ as the random sample [11]. Since k scales with power 2 of σ, it is not practical to use binomial sampling for digital signatures with large standard deviation. Binomial sampler has been employed inside software implementation of NewHope [12, 13, 216, 179], HILA5 [200], LAC [147], LIMA [211], Kyber [43, 16] and Titanium [215]. It also is used in hardware implementation of NewHope [223].

Ziggurat Sampler

Ziggurat sampler is a variation of rejection sampler introduced in [152] for a continuous Gaussian sampler [1]. The discrete version of Ziggurat sampler is proposed in [47] 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

[1] Another method to sample from a continuous Gaussian is **Box-Muller** [45]. Box-Muller method transforms two independent uniforms into two independent discrete Gaussian distributions. pqNTRUSign [50] and NTRUEncrypt [229] use Box-Muller based Gaussian sampler.

its lower right coordinates (Figure 2.2). Firstly, rectangle R_i and point x_i inside the rectangle is chosen uniformly at random. Since we are considering positive x_i, a random sign bit, s, is also required. If $x_i \leq x_{i-1}$, x_i resides below the curve and would be accepted. Otherwise, a uniformly random y_i is sampled and if $y_i \leq exp(x_i)$, the random point (x_i, y_i) is accepted; otherwise new random x should be sampled and the process is repeated.

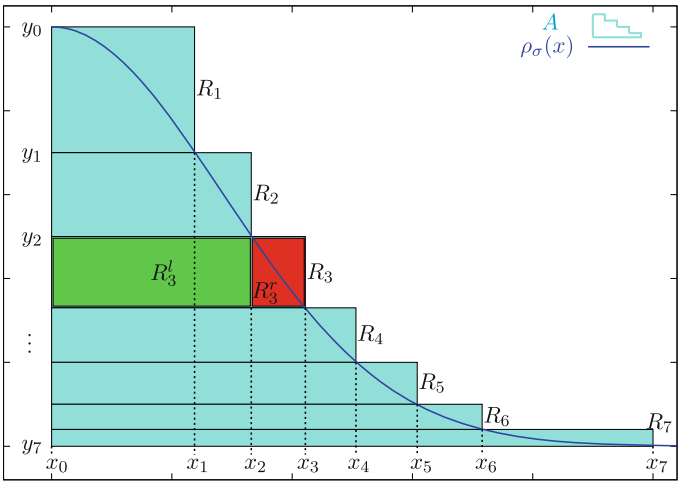


Figure 2.2: A partition of Ziggurat [47].

More rectangles reduce the number of rejections, better performance, and higher precision. Due to its flexibility

Cumulative Distribution Table (CDT) Sampling

CDT is also known as the inversion sampling method [173]. CDT is faster than rejection and Ziggurat samplers by avoiding the use of expensive floating-point arithmetic [222]. 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 [92]) for which their size is a function of distribution 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 binary search, on the CDFto find the corresponded x. A standard CDT needs table of size at least $\sigma\tau\lambda$ bits; for instance `BLISS-IV` ($\sigma = 13.4, \tau = 215, \lambda = 128$) and `BLISS-I` ($\sigma = 13.4, \tau = 19.54, \lambda = 128$) need at least pre-computed tables of size 630 kbits and 370 kbits for 192 and 128 bit post-quantum security, which is not practical for resource-constrained embedded devices [79]. By employing Peikert's convolution lemma [173], the size of precomputed tables are dropped by the factor of 11 by sampling twice from $\tau' = \frac{\tau}{\sqrt{1+k^2}} = 19.47$ instead of sampling from a $D_{Z,\tau}$ with $\tau = 215$. Further improvements on table size are presented in [178] by employing adaptive mantissa size which halves the table size.

Pöppelmann et al. [181] proposed a hardware implementation of CDT sampler which is further optimized in [182, 180, 73]. Du et al. [72] propose an efficient software implementation of CDT that is vulnerable to timing attack which is resolved in [133] by suggesting a time-independent CDT sampler. Pöppelmann et al. [183] combine the CDF and rejection sampling to achieve a compact and reasonably fast Gaussian sampler.

Knuth-Yao sampler

The Knuth-Yao sampler [136] provides a near optimal sampling (suitable for the high precision sampling) thanks to its near entropy consumption of the random bits required by the sampling algorithm. Assume n to be number of possible values for each random variable r with the probability of p_r. The probability matrix is constructed based on the binary expansion of each variable whose r^{th} row denotes the binary expansion of p_r. According to the probability matrix, a discrete distribution generating the binary tree (DDG) is built whose i^{th} level corresponds to the i^{th} column of the probability matrix. Sampling is the procedure of walking through the DDG tree until a reaching a leaf and returning its value as the sampling value. At each level, a uniformly random bit indicates whether the left child

or right child of the current node should be visited in the future. The Knuth-Yao sampler is suitable for schemes with small standard deviations; thus Knuth-Yao is not suitable for digital signature because of its slow sampling caused by a high number of random bits. In order to minimize the statistical distance between the approximated distribution and the true Gaussian distribution, Knuth-Yao sampler needs large memory to store probability of the sample points with high precision, which is an issue on resource-constrained platforms. Combination of Knuth-Yao and CDT results in about halving the table sizes, which is still prohibitively large; however, the Bernoulli sampler offers the best-precomputed table size [79].

De Clercq et al. [66] introduce an efficient software implementation of the Ring-LWE based cryptosystem by using Knuth-Yao as a Gaussian sampler. Using a column-wise method for sampling, Roy et al. [197] propose the first hardware implementation of the Knuth-Yao sampling with small standard deviation which results in faster sampling. Same authors improve their implementation in [195].

Based on the presented results on [47], with the same memory budget, CDT beats rejection sampling and discrete Ziggurat. The Ziggurat sampler outperforms CDT and rejection sampling for larger values of the standard deviation. Ziggurat sampler bears almost the same speedup as Knuth-Yao, while it improves the memory footprint by a factor of 400. As stated earlier, due to the large standard deviation necessary for digital signature, Knuth-Yao sampler is not suitable for digital signatures. Inside an encryption scheme with $\sigma_{LP} = 3.3$ [143], with the same security level ($\lambda = 64$), Knuth-Yao sampler beats CDT in terms of number of operations performed in one second per slice of FPGA (Op/s/S) for time-independent implementation [133]. However, for a time-dependent Gaussian sampler with the same security level ($\lambda = 94$), the CDT sampler proposed by Du and Bai [73] outperforms Knuth-Yao implementation of [195] in term of Op/s/S.

The Size of precomputed tables in a Bernoulli sampler is two orders of magnitude smaller than

Table 2.1: Comparison of different Gaussian samplers; partially extracted from [79].

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

that of CDT and Knuth-Yao sampler [79]; however, CDT has three times more throughput than Bernoulli for hardware implantation of `BLISS` in [180].

2.0.2 Lattice-based schemes

Security of the lattice-based cryptography schemes are based on hardness of solving two average-case problems, a.k.a Short Integer Solution (`SIS`) [154] and the Learning With Errors (`LWE`) problem [190].

Regev [190] proposed the Learning With Errors problem which can be reduced to a worst-case lattice problem like the Shortest Independent Vectors Problem (SIVP). In the proposed scheme, the ciphertext is $\mathcal{O}(n\log n)$ times bigger than plaintext; however, in [175] 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 [143]. Another difference between Regev's encryption scheme and LP scheme is that the former uses discrete Gaussian sampler in the key generation step, while LP employs Gaussian Sampler in encryption in addition to the key generation step.

2.0.2.1 Public Key Encryption

Public key encryption (PKE) is employed to encrypt and decrypt messages between two parties. Additionally, it is a basis for other cryptography schemes such as digital signatures. Generally, it consists of three steps including key generation, encryption, and decryption. The first party (Alice) generates two set of keys and keeps one of them private (sk_{Alice}) and distributes the other key (pk_{Alice}) to other party (Bob). In order to send the message M to Alice, Bob encrypts the message as $S = Enc(M, pk_{Alice})$ where pk_{Alice} is Alice's public and Enc is encryption cipher. Thereafter, Alice decrypts the message as $M = Dec(S, sk_{Alice})$ where Dec is the decryption cipher and sk_{Alice} is Alice's private key. Similarly, Alice should encrypt her message with Bob's public key (pk_{Bob}) if she wants to send message to Bob. Encryption and decryption ciphers are one-way functions and are known publicly; hence the only secret data are private keys of the recipients which should be impossible to uncover using their corresponding public keys. Practical lattice-based PKE schemes are either based on NTRU related [115, 213] or LWE [190] (and its variants including RLWE [151], MLWE [142], ILWE [114] and MPLWE [194]) assumptions.

2.0.2.2 Digital Signature

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

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

Lattice-based signature schemes belong to one of two classes including hash-and-sign (e.g. GPV [101]) and Fiat-Shamir signatures (e.g. BG [20], GLP [105], and BLISS [79]). GPV is a provably secure framework to obtain "hash-and-sign lattice-based signature schemes"; GGH [1] and NTRUSign [116] were the two first works that propose lattice-based signatures which are not provably secure (due to the deterministic signing). Because of the high standard deviation of Gaussian sampler, implementation of lattice-based digital signatures that use Gaussian sampler is challenging. Besides, the need for hash function components and rejection step makes digital signature more complex.

2.0.2.3 Key exchange mechanism

Key exchange is the process of exchanging keys between two parties in the presence of adversaries. If parties use symmetric keys, the same key is shared between them; otherwise, public key of parties should be exchanged. There are numerous public key exchange methods reported in the literature. For classical cryptography, Diffie–Hellman is a practical public key exchange that has been used widely. Establishing a shared key in a lattice-based key encapsulation (KEM) or key exchange (KEX) scheme can be done by either a *reconciliation-based* or *encryption-based* method [12]. Ding [70] propose the first lattice-based key agreement us-

ing the reconciliation-based method. There are plenty of reconciliation-based key agreement schemes based on the LWE (e.g., Frodo [42]), RLWE (e.g., BCNS [44] and NewHope [11]) and MLWE (e.g., Kyer[43]) that require less bandwidth compare to the simper encryption-based ones (e.g., NewHope-Simple [12] and NewHope-512/1024 [179]).

Fujisaki-Okamoto transform

In the random oracle model, an IND-CPA [2] public key encryption (PKE) can be transformed into a IND-CCA [3] PKE using the $Fujisaki - Okamoto$ (FO) transform [96]. Hofheinz et al. [120] introduce a variant of the FO transform that performs transformation from CPA-security into CCA-security in the quantum random oracle model. By applying this variant of FO on a CPA-secure PKE, an IND-CCA key encapsulation mechanism (KEM) is achieved. This transformation is widely used in submitted proposals to "NIST call for post-quantum algorithms" [3] where authors first make a IND-CPA PKE and then build the CCA-KEM, with the same parameter space, by applying the KEM version of the transform [120].
Note. There are plenty of submitted proposals to the NIST PQC standardization call that a single proposal (e.g., LOTUS [176]) supports both public key encryption (e.g., LOTUS-PKE) and key agreement (e.g., LOTUS-KEM); in Table 2.2 and Table 2.3, we list those proposals as two separate schemes. To avoid the redundancy, in section 2.1, we describe each scheme under only one of the categories of public key encryption (section **??**) or key exchange mechanism (section **??**).

Similarly, to avoid the redundancy, an article with both software and hardware implementations is mentioned only once (software implementation in section 2.1.2 or hardware implementation in section 2.1.3).

The contemporary lattice-based schemes existed in the literature are listed in Table 2.2.

[2]Indistinguishability under chosen plaintext attack
[3]Indistinguishability under chosen ciphertext attack

Details on the security level, public key, secret key and ciphertext size of lattice-based public key encryption (PKE), key establishment (KEX/KEM) schemes can be seen in Table 2.3. Furthermore, details on the security level, secret key, public key, and signature size of the lattice-based digital signature schemes are listed in Table 2.4.

Table 2.2: Contemporary lattice-based schemes.

Lattice Type	**Schemes**		
	Public Key Encryption	**Digital Signature**	**Key Exchange**
Standard	LP [143] Lizard [54, 55] NTRUEnrypt [118] EMBLEM.CPA [204] FrodoPKE [158] LOTUS-PKE [176] Odd-Manhattan [177] uRound2.PKE [99]	GGH [1] NTRUSign[1][116] GPV [101] Lyubashevsky [148] BG [20] TESLA [10]	Frodo [42] EMBLEM [204] FrodoKEM [158] spKEX[35] OKCN/AKCN-LWE/LWR[4][130, 230] Lizard [55] LOTUS-KEM [176] Odd-Manhattan [177] uRound2.PKE [99]
Ideal	NTRU [115] NTRU Prime[31] Ring-Lizard [54, 55] trunc8 [202] HILA5 [201, 200] R.EMBLEM.CPA [204] NewHope-CPA-PKE [179] ntru-pke, ss-ntru-pke [229] u/nRound2.PKE [99]	Lyubashevsky [149] GLP [105] GPV [88] BLISS [79] BLISS-B[78] Ring-TESLA [56] TESLA# [26] BLZZRD [199] GLYPH[2][57] FALCON [93] qTESLA [36]	JARJAR, NewHope [11] NewHope-Simple [12] BCNS [44] HILA5 [201, 200] NTRU KEM [126] Ding Key Exchange [69] R.EMBLEM [204] OKCN/AKCN-RLWE[4][130, 230] AKCN/OKCN-SEC[4][230] LIMA-sp/2p [211] RLizard [55] NewHope-CPA/CCA-KEM [179] ntru-kem, ss-ntru-kem [229] NTRU-HRSS-KEM [127] Streamlined-NTRU-Prime [32] NTRU-LPRime [32] u/nRound2.KEM [99]
Module	Kyber PKE [43, 16] AKCN-MLWE-CCA[4][230] KINDI$_{CPA}$ [25] SABER [64]	Dilithium [81, 80] pqNTRUSign [50]	Kyber KEM[43, 16] CNKE, OKCN/AKCN-MLWE[4][230] KINDI$_{CCA-KEM}$ [25] SABER [64] THREEBEARS[3][114]
Middle Product	Titanium-CPA [215]	-	Titanium-CCA [215]

[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 [230] family;

Table 2.3: Comparison of contemporary lattice-based public key encryption and key exchange schemes.

Scheme	PQ security		Failure	Size (bytes)		
	SVP	CCA	Prob.[1]	Secret	Public	Cipher
BCNS (KEX) [44]	78	-	?	4096	4096	4224
JarJar (KEX) [11]	118	-	55	896	928	1024
NewHope (KEX) [11]	255	-	61	1792	1824	2048
Frodo rec. (KEX) [42] [2]	130	-	36	1280	11296	11288
Kyber light (KEX) [43]	102	169	169	832	736	832
Kyber rec. (KEX)[43] [2]	161	142	142	1248	1088	1184
Kyber paranoid (KEX) [43]	218	145	145	1664	1440	1536
NTRU KEM [126]	123	∞	∞	1422	1140	1281
NTRU Prime (KEM) [31]	129	∞	∞	1417	1232	1141
HILA5 (KEM/PKE) [201]	255	135	135	1792	1824	2012
trunc8 [202] [§]	131	-	45	128	1024	1024
NTRU ees743ep1 (PKE) [115]	159	-	112	1120	1027	980
Ding Key Exchange [69] [4]	AES-256 [*]	-	60	3072	2064	2176
EMBLEM (KEM)[204]	AES-128	-	140	2039180	2036736	78368
R.EMBLEM (KEM) [204]	AES-128	-	140	6144	4096	3104
FrodoKEM (Frodo-976) [158]		AES-192	199	31272	15632	15762
FrodoKEM (Frodo-640) [158]		AES-128	148	19872	9616	9736
KCL (e.g., AKCN-RLWE) (KEM) [230] [4]		AES-256	40	1664	1,696	2083
KINDI (PKE/KEM) [25] [4+]		AES-256	276	2752	2368	3392
LAC (e.g., LAC256) [147] [4+]		AES-256	115	2080	1056	2048
LIMA (e.g., CCA.LIMA-2p2048) [211] [4+]		SHA-512	314	18433	12289	7299
LIMA (e.g., CCA.LIMA-sp2062)[211] [4+]		SHA-512	244	24745	16497	9787
Lizard.CCA (PKE) [55] [4]		AES-256	381	557056	6553600	3328
RLizard.CCA (PKE) [55] [4]		AES-256	305	513	8192	8512
Lizard.KEM [55] [4]		AES-256	381	34880	4587520	35904
RLizard.KEM [55] [4]		AES-256	305	769	8192	8256
LOTUS (PKE/KEM) [176] [4+]		AES-256	256	1630720	1470976	1768
NewHope1024 (KEM) [179] [5]		AES-256	216	3680	1824	2208
NewHope512 (KEM) [179] [5]		AES-128	213	1888	928	1220
Streamline-NTRU-Prime (KEM) [32]		AES-256	∞	1600	1218	1047
NTRU-LPRime (KEM) [32]		AES-256	∞	1238	1047	1175
NTRU-HRSS-KEM [127]		AES-128	∞	1418	1138	1278
NTRUEncrypt (e.g., ntru-kem-743) [229] [4]		AES-256	112	1173	1023	1023
Odd-Manhattan (KEM) [177] [4]		AES-256	?	4456650	4454241	616704
uRound2 (uround2_kem_nd_l5) [99] [4]		AES-256	65	169	709	868
nRound2 (nround2_kem_nd_l5) [99] [4]		AES-256	45	165	691	818
uRound2 (uround2_pke_nd_l5) [99] [4]		AES-256	137	1039	830	953
nRound2 (nround2_pke_nd_l5) [99] [4]		AES-256	164	1039	830	1017
SABER (PKE/KEM) [64] [4]		AES-256	165	3040	1312	1472
THREEBEARS (KEM) [114] [4]		AES-256	188	40	1584	1697
Titanium-CPA (PKE) [215] [4]		AES-256	30	32	23552	8320
Titanium-CCA (KEM) [215] [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);

Table 2.4: Comparison of popular lattice-based key digital signature schemes

Scheme	Security		Size		
	PreQ	PostQ	Secret Key	Public Key	Signature
GPV [101]	100	?	256 B	1.5 kB	1.186 kB
BG [20]	128	?	0.87 MB	1.54 MB	1.46 kB
TESLA–128 [10]	128	?	1.01 MB	1.33 MB	1.280 kB
TESLA–256 [10]	256	128	1.057 MB	2.2 MB	1.688 kB
GLP [105]	100	<80	256 B	1.5 kB	1.186 kB
BLISS-I [79] [2] BLISS-BI [78][1]	128	<66	256 B	896 B	700 B
TESLA#–I [26]	128	64	2.112 kB	3.328 kB	1.616 kB
TESLA#–II [26]	256	128	4.608 kB	7.168 kB	3.488 kB
Ring-TESLA-II [56]	118	64	1.92 kB	3.328 kB	1.568 kB
Dilithium rec. [81] [3]	138	125	3.504 kB	1.472 kB	2.701 kB
Dilithium high. [81][4]	176	160	3.856 KB	1.760 kB	3.366 kB
FALCON (falcon1024) [93]	AES256	128	8.193 kB	1.793 kB	1.233 kB
FALCON (falcon768) [93]	AES192	96	6.145 kB	1.441 kB	1.077 kB
FALCON (falcon512) [93]	AES128	64	4.097 kB	897 B	690B
pqNTRUsign [50] [5]	AES256	128	2.604 kB	2.065 kB	2.065 kB
qTESLA (qTesla_256) [36] [5]	AES256	128	8.256 kB	8.224 kB	6.176 kB
qTESLA (qTesla_192) [36] [5]	AES192	96	8.256 kB	8.224 kB	6.176 kB
qTESLA (qTesla_128) [36] [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;

2.1 Implementation challenges

In this section, we consider various implementations of lattice-based cryptographic schemes using software, hardware, software/hardware codesign and DSP techniques. First we focus on the implementation of key arithmetic modules such as Gaussian sampler and matrix/polynomial multiplication in Section 2.1.1. Software and hardware implementations of lattice-based cryptographic schemes are described in Section 2.1.2 and Section 2.1.3, respectively. Section 2.1.4 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 2.1.5.

Table 2.5 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.

Table 2.5: Popular implementation of lattice-based schemes.

Lattice Type	Schemes		
	Software	Hardware	Hardware/Software
Standard Lattices	PKE: [54, 55] [118] [204] [158] [176] [177] [99] DS: [20] [88][63] [10] KEX: [35] [42] [204] [158] [35] [130, 230] [55] [176] [177] [99]	PKE: [122] [204]	-
Ideal Lattices	PKE: [66] [144] [185] [192] [49] [228] [31] [54, 55] [202] [201, 200] [204] [179] [229] [99] DS: [105] [108] [88] [171] [180] [41] [185, 187] [7] [88] [79] [78] [56] [26] [199] [57] [93] [36] KEX: [70] [130] [11] [12] [13] [104] [126] [12] [44] [201, 200] [69] [204] [130, 230] [230] [211] [55] [179] [229] [127] [32] [99]	PKE: [103] [182] [196] [183] [204] [192] DS: [105] [180] [107] [124] [159] KEX: [204] [223] [139] [159]	DS: [19] [18]
Module Lattices	PKE: [43, 16] [64] [25] [230] DS: [81, 80] [119, 50] KEM: [43, 16] [114] [64] [25] [230]	-	-
Middle Product Lattices	PKE:[215] KEM:[215]	-	-

2.1.1 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 2.1.1.2.

2.1.1.1 Gaussian Sampler

Dwarakanath et al. [85] provide a survey of different algorithms of computing the exponential function efficiently on resource-constrained devices regarding memory capacity. In order to decrease memory footprint, pipelining the sampling algorithm is offered. To be more specific, authors divide the distribution curve into rectangles with the same probability and choose the rectangles according to the Knuth-Yao method which means the Knuth-Yao method is employed another round for rectangle itself. Tail probabilities in discrete distribution are relatively small which provide the chance of approximating them with lower precision arithmetic. Consequently, on the fly tail construction using standard double-precision floating-point precision is suggested. Although the offered idea could significantly reduce memory (lookup table) footprint since lookup tables only store non-tail probabilities; however, considerable floating point arithmetic overhead is imposed on the system.

Software Implementation. Buchmann et al. [47] design and implement, in C++, a flexible Gaussian sampler named Ziggurat which sets a trade-off between memory footprint, precision, and execution time. The area under the probability density function (PDF) is divided into rectangles with the same area that is employed to minimize calculation of expensive exponential function. More rectangles (more memory footprint) results in higher precision and better performance. The Ziggurat sampler is attractive because of the potential flexibility that makes it a good candidate to use in crypto-engines of either high-performance servers (allocate more memory to reach better performance and precision) or low-speed resource constraint embedded devices (use few numbers of rectangles to minimize memory consumption). In order to increase the performance of Ziggurat sampler, more rectangles are required which imposes significant memory overhead since it needs to re-compute all the rectangles and save them in the memory. A better way to improve Ziggurat sampler is to increase the number of approximation lines (by adding two extra points) which culminates in decreasing number of the exponential function calculation [157]. In other words, more approximation lines decrease the probability of rejection by providing a more precise approximation of the curve. Consequently, performance and memory occupation are improved by employing more approximate lines.

Hardware Implementation. Roy et al. [197] implement the first high precision and low area hardware implementation of Knuth-Yao sampler with small standard deviation on a Xilinx Virtex5 FPGA. Knuth-Yao involves a random walk tree traversal which imposes expensive sequential bit scanning and wide ROM footprint. To improve performance, authors traverse the discrete distribution generating (DDG) tree by employing relative distance of intermediate nodes. Column-wise storing of the samples probability in ROM improves the performance of the sampler. Numerous zeros in the probability matrix are well compressed by applying one-step compression which culminates in a near-optimal number of random bits to generate a sample point. Presented Knuth-Yao sampler suffers from vulnerability to timing and power attacks due to the non-constant time random walk which is solved

by a random shuffle approach in order to eliminate leaking the timing information [195]. Authors offer the solution (random shuffle) but do not evaluate hardware implementation of the shuffler. In the new implementation, the efficiency of the Knuth-Yao is enhanced by employing small LUTs with the input of the random bits and output of the sample point with high probability or an intermediate node positioned in the discrete distribution generation tree. Employing a lookup table with 8-bit input results in hitting a sample point (eliminate expensive bit-scanning procedure) with the probability of 97% by eight random bits. Additionally, a more compact sampler is achieved by reducing the width of ROM and random bit generator.

Du and Bai [73] implement a highly precise (large tail bound) and area efficient cumulative distribution function (CDF) inversion variant of the discrete Gaussian sampler on Xilinx Spartan-6 FPGA. Authors reduce area occupation by employing piecewise comparison (results in 90% saving of the random bits) and avoiding comparison of large numbers which is an improvement on [182]. Further improvement is achieved by employing small lookup tables with high hit rate which culminates in performance improvement. Performance of the proposed Gaussian sampler is improved twofold by the same authors with a software implementation (Intel Core i7-4771) [72]. The primary challenge to improve performance on a general purpose processor is a large number of random bits that are consumed by the sampler to generate a random number. This obstacle is alleviated (around 30%) by employing multi-level fast lookup table (using eight smaller lookup table instead of one). Further speed improvement of the sampler is gained by applying the multithreading technique. The main security drawback of both hardware and software implementation of the discrete Gaussian sampler by Du and Bai is its vulnerability to timing attacks because of the non-constant time traversal of the binary tree [121].

Howe et al. [121] propose a comprehensive evaluation of various practical hardware implementation of time-independent discrete Gaussian samplers, including Bernoulli, discrete

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

Pöppelmann and Güneysu [183] propose an area optimized hardware implementation of Bernoulli sampler that employs Bernoulli evaluation instead of evaluation of `exp()`. Rejection probability is high due to the absence of binary Gaussian distribution (easy to sample intermediate sampler) which results in increasing the entropy consumption and runtime. Although proposed Gaussian sampler is suitable for encryption schemes, it would be challenging to be employed inside digital signatures as the sampling component. Pöppelmann et al. [180] propose a hardware implementation of Bernoulli sampler with binary Gaussian distribution for `BLISS` scheme on Xilinx Spartan-6 FPGA.

Göttert et al. [103] propose the first hardware implementation of the discrete Gaussian sampler. Authors use rejection sampling in their software implementation; however, because of the obligatory floating point arithmetic, authors prefer to employ lookup tables in their hardware implementation of implement Gaussian which the Gaussian distributed values in the stored array are indexed using a pseudo-random bit generator. The proposed sampler has an unsatisfactory precision which has far distance from golden discrete Gaussian distribution due to the small tail bound. Authors employ a fully parallel architecture to design a

polynomial multiplier which provides high throughput but makes the design extremely big which cannot be fitted into the largest Virtex-7 FPGA family.

Roy et al. [196] propose a compact Ring-LWE cryptoprocessor to optimize NTT multiplication which is accomplished by the reduction in fixed computation (4 NTT instead of 5 NTT) and in reducing the pre-scaling overhead. Besides, NTT Memory access is minimized by storing two coefficients in a single word, processing two pairs of coefficients together, and eliminating idle cycles. Authors avoid using ROM to save the twiddle factors and instead compute the twiddle factors on demand. Besides, they reduce security by limiting coefficients of the secret key to be binary, instead of Gaussian distributed, which gives the opportunity to replace multiply with addition operations. Small lookup tables are used in the Knuth-Yao discrete Gaussian sampler to avoid expensive bit scanning [197] to improve speedup; additionally, ROM widths are reduced which results in a more compact and faster sampler than Bernoulli [183].

2.1.1.2 Multiplier

Software Implementation. Emeliyanenko [89] proposes an efficient 24-bit modular multiplication to achieve high throughput polynomial multiplications using NTT algorithm (on Nvidia GeForce GTX 280 GPU). Employing CUDA FFT kernel and Chinese Remainder Theorem (CRT), the proposed method provides better speedup compared to the GMP and NTL libraries for moderate coefficient bit-length.

Akleylek et al. [8] propose sparse polynomial multiplication for `LBC` for the first time which improves the performance of digital signature proposed in [107] by 34%. Authors implement the Schönhage-Strassen polynomial multiplication on NVIDIA GPU and compare its performance with existed multiplication schemes, including iterative NTT, parallel NTT and CUDA-based FFT (cuFFT) for different integer size.

Akleylek et al. [6] propose a software implementation (Intel Core i5-3210M processor) of the sparse polynomial multiplication using a sliding window that bears around 80% of speed improvement compared to NTT [108]. Authors assume to have polynomials with coefficients with three possible values including -1,0 and +1. Multiplications by zero are avoided, and for +1 and -1 cases addition and subtraction are used, respectively. The method can be used for polynomials with arbitrary coefficients by substituting a multiplication with a loop of additions. Performance of the proposed method depends on high number zeros and numerous identical patterns which make the system prone to timing attacks.

`FNLlib` [4] is a scalable, efficient, and open source C++ library contains optimized arithmetic operations on the polynomials for the ideal lattice-based cryptography schemes. Compared to the generic libraries for polynomial arithmetic, several orders of magnitude improvement in the speed is achieved by employing algorithm optimizations, including fixed sized Chinese Remainder Theorem (CRT), scalar modular multiplication and NTT algorithm, and programming-level optimizations, such as SSE and AVX2 SIMD. Authors use NFLlib for the RLWE encryption scheme and homomorphic encryption and compare their efficiency with classical cryptographic schemes (like RSA) and libraries (like NTL).

Longa et al. [146] 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, speed improvement of factor 1.9 for C implementation (on a 64-bit platform) and 1.25 for AVX2 vector implementation compared to NewHope (tolerant against timing attacks) is achieved. However, due to lack of 64-bit register, proposed reduction technique does not provide any speed up on 32-bit microcontrollers [13]. Additionally, authors use signed integer arithmetic which optimizes the number of add operations in both sampling and polynomial multiplication.

Hardware Implementation. Howe et al. [122] propose the only hardware implementation of standard lattice-based encryption scheme based on LWE problem. Authors perform the

multiply-accumulate (MAC) operations of matrices in the encryption scheme by utilizing a dedicated DSP48A1 unit of the Spartan-6 FPGA to achieve an area optimized hardware implementation of standard LWE based encryption engine.

Pöppelmann et al. [181] propose the first hardware optimization of polynomial multiplication (NTT) for the ideal lattice-based encryption schemes on a Xilinx Spartan-6 FPGA with the primary goal of minimizing the area. Authors design a sequential NTT (one butterfly operator) that stores twiddle factors in a dedicated ROM which imposes memory overhead but achieves decent performance. Twiddle factors refer to different powers of the multiplicative operands. An optimized version of presented NTT polynomial multiplier is employed in [182] to design in a Ring-LWE encryption engine. With acceptable runtime, authors of [17, 196] present an optimized hardware implementation of NTT introduced in [181] to compute polynomial multiplication in a Ring-LWE on the smallest Spartan-3. The main idea is to compute twiddle factors on the demand instead of storing them in the ROM. By replacing the modulus with a Fermat number, shift operation could be used instead to polynomial exponentiation. However, proposed optimizations cannot take advantage of inherent parallelism in NTT. It should be mentioned that authors of [17] do not provide the implementation of the whole crypto-engine.

Chen et al. [51] present a high-performance polynomial multiplication for Ring-LWE encryption cryptosystems in hardware on a Spartan-6 FPGA by exploiting the parallel property of the NTT. Authors provide a different secure set of parameters by which efficient Ring-LWE encryption and SHE could be achieved. They prove that polynomial multiplication can be done by computing the negative wrapped convolution by which there is no need to compute the modular reduction. Besides, size of FFT/inverse-FFT and point-wise multiplication is halved compared to the zero padding method. To be more specific, the proposed architecture for polynomial multiplication consists of two butterflies and two point-wise modulo p multiplier (p has the adjustable length) which produce outputs (equivalent to two parallel

NTT) that could be used to perform the inverse-FFT.

Du and Bai [75] propose a scalable and efficient polynomial multiplier architecture that take advantage of NTT's parallelism (implemented on Xilinx Spartan-6 FPGA) which provides a speed and area trade-off. In [181] and [196, 17] one and two butterfly operators are employed, respectively; however, [75] use b (power of 2) butterfly operators to improve speed of the polynomial multiplier which perform multiplication of two n-degree polynomials in $(1.5n + 1.5n\log n)/b$ cycles. To improve area (minimize required area), authors employ the cancellation lemma to minimized number of constant factors. Butterfly operation takes two coefficients (x,y) and one constant factor (ω) and makes two new coefficients ($[x+\omega y]$ mod p, $[x\text{-}\omega y]$ mod p). The butterfly can be used as a modulo p multiplier (by setting $x = 0$). Besides, in the first stage of inverse NTT the constant factor (ω) is 1, hence new coefficients are *[x-y] mod p* and *[x+y] mod p*. Consequently, necessary clock cycles to calculate a sequential polynomial multiplication is $(1.5n + 1.5n\log n)$ which reduces $3.5n$ of required cycles.

Györfi et al. [112] perform a thorough evaluation of various candidates to implement modular FFT on Xilinx Kintex-7 FPGA. Authors study three architectures in the diminished-one number system (computations over Z_{2k+1}) for different parameters in order to meet various factors such as run-time execution, throughput, occupation, and scalability. The first architecture yields the best performance, using pipelined modular butterfly-based FFT in which FFT core is throughput optimized. The other two architectures are serial distributed arithmetic-based and nested multiplication which occupy less area than butterfly-based FFT.

Du and Bai [76] demonstrate 30% savings in time and space (compared to [181]) on a Spartan-6 FPGA by performing on-the-fly performing bit-reversal step along with a new memory access scheme, (to load/store coefficients in calculating NTT). The idea is to load/store at address bit-reverse(i) instead of load/store at address i in memory, which means ith coefficient of NTT's output is located in the $bit - reverse(i)$th memory location. Consequently,

the bit-reversal step in the inverse-NTT is eliminated. Authors employ two Block RAMs on FPGA which provide interleaving, hence two parallel NTT can be interleaved. Authors apply their optimization to the NTT of an RLWE base public key cryptosystem [74]. Also, it is assumed uniformly random polynomial in the public key scheme to be fixed, hence precomputing the NTT of it offline improves the performance. In the above two articles, only one butterfly operator is used; however, by adapting bit-reversal and memory saving methods of above articles, authors propose a fast polynomial multiplication architecture with four butterfly operators that achieve on average 2.2 speedup improvement [77]. Two butterfly operators are used to calculate the ith level, and other two perform $(i+1)$th level calculations in the pipeline by using results of the ith stage.

2.1.2 Software Implementations of Lattice-based Cryptographic Schemes

Public Key Encryption. de Clercq et al. [66] propose an efficient software implementation of Ring-LWE based encryption for ARM Cortex-M4F micro-controller. The main goal was maximizing the speed (using assembly level optimization) and minimizing memory footprint (by storing two coefficients in one word). They employ the Knuth-Yao algorithm to achieve fast noise sampling and use the platform's True Random Number Generator (`TRNG`) to generate random numbers. Authors employ optimization of the paper [196] 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.

In [144], the authors use a byte-wise scanning method to improve the performance of the Gaussian sampler based on the Knuth-Yao algorithm. This allows them to implement a Ring-LWE based public key encryption scheme on a resource-constrained 8-bit ATxmega128 AVR

processor. By applying sophisticated memory alignments for storing coefficients, about a 20 percent decrease in RAM usage is achieved. For NTT computation a couple of optimization techniques are employed including approximation based reduction and negative wrapped convolution.

Buchmann et al. [49] implement a high performance and lightweight public key encryption scheme implemented on small and 8-bit ATXmega128 and 32-bit Cortex-M0 microcontrollers by replacing the Gaussian noise distribution with a uniform binary error distribution. The determine security of the scheme by evaluating hardness of binary LWE against hybrid attack.

The main advantage of this scheme over Lindner-Peikert's proposal (`LP`) [143] is its smaller key and ciphertext size. Regarding the speed, it is beaten by the scheme in [144] with slightly higher memory footprint. Similarly, proposed design in [185], uses NTT with precomputed twiddle factors and eliminating the bit reversal step which results in twofold performance improvement.

Yuan et al. [228] provide a portable JavaScript implementation of lattice-based cryptography 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 [131] is employed for NTRU schemes. In order to reduce the execution time, inverse transform sampling is employed in which possible values are precomputed and stored in a lookup table.

Reparaz et al. [192] implement a masked Ring-LWE scheme on a Virtex-II FPGA and 32-bit ARM Cortex-M4F which is Differential Power Analysis (DPA) resistant. In order to be resilient to first-order side-channel attacks, a constant time masked decoder with high success probability is implemented. Entire computation is done in the masked domain by employing a dedicated masked decoder which imposes considerable time and area overhead

compared with an unprotected design.

Cheon et al. [54] exploits learning with rounding (`LWR`) problem [23] and present `Lizard` and its ring variant (`Ring-Lizard`). Discrete Gaussian error distribution is replaced with an efficient rounding process with smaller modulus. Based on the results, 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. [55] submitted `Lizard` (IND-CPA/CCA PKE and IND-CCA2 KEM) and its ring variant, `RLizard`, to the NIST PQC standardization call (NIST security category of 1, 3 and 5). Sparse and small secrets version of LWE and LWR (RLWE and RLWR) are the security basis of the Lizard (RLizard) IND-CPA PKE.

Besides Intel Xeon E5-2620 CPU, authors provide performance evaluation on a smartphone (Samsung Galaxy S7) for their recommended parameter of Lizard.CPA (128-bit quantum security). Authors claim that Lizard is suitable for smartphones (memory usage of 20 megabytes). Authors provide datapath and finite state machine for hardware implementation of the Lizard PKE using Lizard.CPA and RLizard.CPA.

Chen et al. propose `NTRUEncrypt` [229] (at the NIST standardization call), a family of IND-CCA2 (resistant to subfield attacks) PKE and KEM schemes at 85, 159 and 198-bit post quantum security (NIST security category 1, 5 and 5). Based on the original NTRU scheme [115] and using parameters set of [118], `ntru-pke` and `ntru-kem` are achieved by applying NAEP transformation [125]. Using the same transformation, base on the provably secure NTRU encryption scheme [213] (based on RLWE problem), `ss-ntru-pke` and `ss-ntru-pke` are derived. Modulus is chosen to be a power of 2 (2^{11}) to enhance efficiency of the modulo arithmetic and integer multiplications. Authors adopt a PRNG from Salsa20 [30] to expand the seed. Box-Muller [45] is employed (only in ss-ntru-pke and ss-ntru-kem) to sample from the discrete Gaussian distribution. The performance results are reported only for Intel i7-6600U processor (AVX2 optimization of NTT is not performed).

Bernstein et al. [32] 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 the key size of around 1kB that are designed to reduce attacker's success probability by eliminating the ring homomorphisms. Schemes are IND-CCA2 where a key can be used multiple times; hence large key generation latency is tolerable. Authors implement the reference code on an Intel Xeon E3-1275 v3. `Streamlined-NTRU-Prime` is faster than `NTRU-LPRime` in terms of the encapsulation and decapsulation time with slower key generation.

Hülsing ea al. propose `NTRU-HRSS` [127], a One-way CPA secure (OW-CPA) PKE and `NTRU-HRSS-KEM`, a CCA2-secure KEM derived from NTRU [115] with 123-bit post-quantum security (NIST security category 1). In contrast to `NTRUEncrypt` [229] and standard NTRU [2], KEM is derived directly from NTRU-HRSS without using padding techniques (e.g., [125]). Contrary to Streamlined NTRUPrime [31] 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 use 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 and messages and large modulus in order to avoid decryption failure (in contrast to LWE-based schemes with a non-zero probability of failure) which leads to lower security and higher communication cost. Authors report the performance results of the reference and AVX2 implementation on Intel Core i7-4770K CPU.

Bansarkhani proposes `KINDI` [25] (at NIST PQC standardization call), a trapdoor-based encryption scheme based on LARA [188] in which data is concealed into the error without changing the target distribution. As a result, more data can be encrypted per ciphertext bit

which reduces the message expansion factor (beneficial in "sign-then-encrypt"). $\texttt{KINDI}_{\texttt{CPA}}$, (Module-LWE based IND-CPA PKE) has been proposed with five different parameter sets ranging from 164 to 330-bit security (NIST security category of 2, 4 and 5). By applying a variant of Fujisaki-Okamoto (FO) transformation [120] on the $\texttt{KINDI}_{\texttt{CPA}}$, $\texttt{KINDI}_{\texttt{CCA}-\texttt{KEM}}$ with the same parameter space, can be built.

Key Exchange. Ding et al. [70] 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 [174]. A practical constant-time software implementation of the Peikert's Ring-LWE key exchange protocol is proposed in [44], 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 key 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). For polynomial arithmetic, authors adapt the FFT from Nussbaumer's method [169], in cyclotomic rings whose degree is a power of two that provides efficient modular reduction. BCNS employs a fixed polynomial as the system parameter which could be a potential weak link of the protocol. Besides, selection of large modulus results in lower efficiency and security level, 78-bit quantum security, than expected from a Ring-LWE scheme. In contrast to the digital signature and encryption schemes, key exchange scheme does not need a high-quality Gaussian sampler [11], which BCNS uses; consequently, a simpler noise distribution is used in NewHope instead of Gaussian sampler. BCNS caches keys, which can be very dangerous to the security of the protocol because of the shared-key reused attacks [91], which is solved in the NewHope.

Alkim et al. [11] introduce NewHope, a portable C and highly optimized SIMD implementation (AVX2) of unauthenticated key exchange scheme, that solves the inefficiency (10 times better performance) and security drawbacks (increase quantum security level from 78-bit

to 128-bit) of BCNS by optimizing the key exchange algorithm and better parameter selection. A better analysis of failure probability which results in smaller modulus, on the fly generation of the polynomial system parameter, efficient polynomial arithmetic (combining Montgomery and Barret reduction and employing polynomial encoding), and using the centered binomial instead of the discrete Gaussian distribution are the main improvements of NewHope over BCNS. NewHope has attracted the attention of research and industry communities such that Google released the Chrome Canary which uses NewHope as the key exchange protocol along with elliptic curve Diffie–Hellman as the authentication protocol [46].

Alkim et al. [12] propose `NewHope-Simple`, a simpler variant of the Newhope with the same performance and security level. Simplicity is achieved by eliminating the error-reconciliation mechanism [70] with 6% message size overhead. Authors discard the least significant bits of each coefficient due to their negligible impact on the successful plaintext recovery. Additionally, authors encode a single key bit into four coefficients that results in the reduction of the ciphertext length. In `NewHope-Simple`, polynomial a can be fixed, while the original `NewHope` generates a on the fly for every single run of the scheme. Alkim et al. [13] present the software implementation of NewHope on ARM Cortex-M family, low power Cortex-M0 and high-performance Cortex-M4, which is the first key exchange scheme with the quantum security level of 128-bit on constrained embedded devices. Authors optimize all hot regions of protocol in assembly, including error reconciliation, the uniform noise generation by ChaCha20 stream cipher [29], and NTT/NTT^{-1}. For NTT, authors set a memory-time trade-off for precomputing powers of constants (design parameters) by which only a subset of the powers of constants are precomputed and stored in the table. Gueron and Schlieker [104] further optimize the NewHope by optimizing the pseudorandom generation part which results in 1.5× better performance on the Intel Skylake processors. Authors improve the sampling step by lowering the rejection rate (from 25% to 6%) and exploit the parallelism in the pseudorandom generation (replace SHAKE-128 with the parallelized SHA-256 or AES block

cipher) and rejection sampling (employing AVX vector instructions). Longa and Naehrig [146], employ a reduction technique (during the NTT calculation) that eliminates the modular reduction after additions of two polynomials which results in the speed improvement of 1.9 and 1.25 for C and AVX implementations (compared to the reference NewHope [11]), respectively.

Adopted from the NewHope-Simple, Alkim et al. [179] 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 target 101-bit security (NIST security category level 1) and NewHope1024-CPA-KEM and NewHope1024-CCA-KEM with 233-bit security (NIST category security category 5) with comparable performance as the elliptic curve based cryptosystems. Four mentioned KEMs are derived from NewHope-CPA-PKE (which does not support arbitrary length messages, hence can not be used as a standalone encryption scheme) by applying a variant of Fujisaki-Okamoto transform [120]. In order to generate the random number and shared secret, hash function `SHAKE256` [87] is used as a pseudorandom function; generation of the shared polynomial a is done by expanding a 32-byte seed using `SHAKE128` [87]. Besides the reference and vectorized (using AVX instructions) implementations on the Intel Core i7-4770K (Haswell) processor, authors provide implementation and optimization of KEMs on a 64-bit MIPS architecture (MIPS64).

Ding et al. [69] propose (at NIST PQC standardization call) *Ding Key Exchange*, an ephemeral IND-CPA secure error reconciliation-based key exchange protocol from RLWE problem. At the same security level, Ding Key Exchange reduces communication cost (due to its rounding technique) compare to the similar schemes (NewHope, NewHope-Simple, and Kyber). It provides equivalent security to AES-128, AES-192 and AES-256 (NIST security category 1,3 and 5) with flexible parameter choices and key size of n-bit where n can be 512 and 1024; as a result, it is more resistant to Grover algorithm compare to NewHope,

NewHope-Simple and Kyber with the key size of 256-bit. NTL library [209] and CDT sampler are used for polynomial multiplication and sampling from the Gaussian distribution.

`HILA5` [201], a Ring-LWE-based key exchange scheme (and also public key encryption) with the same security parameters ($n = 1024, q = 12289$) and sampler (binomial sampler ψ_{16}) as NewHope and has been tested on Intel Core i7-6700 CPU; also it has been integrated into OQS and OpenSSL. HILA5 uses SafeBits, improved version of the Peikert reconciliation mechanism [174], to reach slightly smaller messages than NewHope (36-byte which is 0.9%) at the same security level by generating unbiased secret bits and hence less randomness in secret bits. HILA5 employs an efficient constant time error correction block to correct 5 bits of error which results in decryption failure of 2^{-128} compared to NewHope's failure rate of 2^{-64} (Frodo [42] and Kyber [43] with failure rate of $2^{-38.9}$ and $2^{-71.9}$) by sacrificing less than 4% of performance. HILA5 can be employed as a PKE scheme due to its higher reliability. `HILA5` [200] is submitted to the NIST PQC standardization call as a famility of PKE and KEM schemes that provide equivalent security to AES-256 (NIST security category 5). Optimized polynomial multiplication and error sampling are performed by employing the Cooley-Tukey [61] method and binomial distribution. Besides, SHAKE-256 is used to sample from the uniform distribution.

`Frodo (FrodoCCS)` [42], the first practical implementation of public key exchange scheme based on standard lattices, original LWE problem [190], is secure against cache-timing attacks. Like BCNS, Frodo could 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 (4.7 times more than NewHope), throughput (1.2 less throughput than NewHope), and performance (8 times slower than NewHope). Massive memory overhead is imposed if matrix variant should be saved in the memory. By generating and afterward discarding the matrix variant (on-the-fly), memory overhead is alleviated. Besides, authors use an efficient Gaussian sampler, inversion

sampling method, which employs precomputed tables. Based on the authors' claim, integrating Frodo (LWE-based post-quantum key exchange protocol) into TLS halves the server throughput. To tackle the ring related attack, NTRUPrime is proposed as a more secure scheme by using a combination Karatsuba, schoolbook, and Toom's multiplier [31]. Consequently, NTT-friendly prime and polynomial are not crucial which results in a negligible drop in performance compared to NewHope [11].

`FrodoKEM` [158] is a family of IND-CCA secure KEMs based on the LWE problem with brute-force security of at least AES-128 (FrodoKEM-640) and AES-192 (FrodoKEM-640). FrodoPKE is transformed by a variant of `FO` transformation [120] to build the FrodoKEM. Authors generate public matrix A from a small seed using PRNG (AES128 or cSHAKE128) which results in a more balanced ciphertext and key sizes, but remarkable computational overhead. Timing and cache attacks are prevented by prohibiting the use of secret address accesses and branches. A portable C code reference and its optimized implementation (of generating the public matrix A and matrix arithmetic) are provided. Besides, authors report the results of the implementing on a 64-bit ARM Cortex-A72 (with the best performance achieved by using OpenSSL AES implementation, that benefits from the NEON engine) and an Intel Core i7-6700 (x64 implementation using AVX2 and AES-NI instructions). Employing modular arithmetic ($q \leq 2^{16}$) results in using efficient and easy to implement single-precision arithmetic. The sampling of the error term (16 bits per sample) is done by inversion sampling using a small lookup table corresponds to the discrete cumulative density functions (CDT sampling).

Open Quantum Safe (`OQS`) [212] software platform is designed to evaluate proposed quantum-resistant schemes which have an open-source library (contains C implementation of BCNS, NewHope, and Frodo) of post-quantum cryptographic schemes. More importantly, OQS offers the chance to integrate the quantum resistant schemes into classical applications and protocols with the goal of minimizing the software change; besides, OQS provide the op-

portunity to compare post-quantum schemes with each other or with classical cryptographic algorithms.

Jin and Zhao [130] present symmetric (OKCN) and asymmetric (AKCN) LWE and Ring-LWE based key exchange mechanisms (NewHope and Frodo are Ring-LWE and LWE-based symmetric key exchange schemes). OKCN, optimally-balanced key consensus with noise, can be used for key transport and encryption, while AKCN, asymmetric key consensus with noise, only can be employed for the key transport. In the proposed scheme, the server sets the session key before starting the key exchange mechanism. Consequently, it provides the opportunity to encrypt the message offline which provides higher security and better workload balance. Compares with Frodo, OKCN-LWE produces a much smaller matrix by eliminating the least significant bits of each LWE sample which results in less computation for matrix arithmetic; smaller matrix also results in the faster generation and sampling of the matrix. With the same set of parameters (same security level), OKCN-LWE consumes more bandwidth (30%) than Frodo, while its failure probability is remarkably lower. Employing the same optimization techniques, Ring-LWE based version of OKCN, which adopts the same noise distribution and parameters of NewHope, provides a more computationally efficient scheme than NewHope. Authors integrate the OKCN-LWE scheme into the open safe project platform [212].

Zhao et al. [230] extend [130] and present a generic construction of 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 exchanges require less bandwidth (18% and 28%) compare to Frodo at the same security level (shared key size of 256-bit). The most efficient key exchange mechanism with share the key size of 512-bit is achieved by AKCN. Authors prove that the errors in different positions in the shared key are independent and propose single-error correction (SEC) code to correct at least one bit error; using the SEC, with the same security

and error rate, OKCN/AKCN-RLWE based KEX schemes generate 765-bit shared key with less bandwidth than NewHope and NewHope-Simple (with 256-bit shared key). Authors claim that they provide the most efficient lattice-based key exchange scheme with share key size of 256-bit by applying OKCN/AKCN to MLWE-based key exchange mechanism. Additionally, the authors provide a new authenticated key exchange scheme named concealed non-malleable key-exchange (CNKE).

Seo et al. [204] propose error-blocked multi-bit key encapsulation mechanism named EMBLEM and (R.EMBLEM) which is (secure against adaptive chosen-ciphertext attack) based on the small secret LWE (RLWE) problem. During the decryption phase, the error does not affect the message by separating the message and error and concatenating each message block with the error-blocking bit. The secret key is sampled uniformly at random in $[-B, B]$ (where B is a positive integer smaller than σ) instead of Gaussian distribution. Consequently, the key size is notably reduced since the secret key can be generated by a 256-bit seed which eliminates the need for storing the whole matrix; however, it imposes computational overhead to generate the secret key from the seed using pseudorandom functions. For polynomial multiplication in R.EMBLEM, Cooly-Tukey butterfly and Gentleman-Sande butterfly are used in NTT and inverse NTT, respectively. Besides the software implementation on Intel core-i7-7600, authors implement schemes on the Zynq 7 FPGA platform.

`Kyber` [43] is a highly optimized IND-CCA KEM with the post quantum security based on the hardness of solving the module learning with error problem (`Modulue-LWE`) [142]. Ideal lattices with their ring structure decrease public key and ciphertext size of standard lattices schemes by sacrificing security assumption. Module lattices proposed to fill the gap by believing that full ring structure is excessive [142]. Authors define IND-CPA PKE scheme under Module-LWE hardness assumption and apply a variant of Fujisaki-Okamoto transform [120] to build a IND-CCA KEM. Employing IND-CCA KEM, they design IND-

CCA KEX and AKEX under hardness assumption in the classical and quantum random-oracle models. Kyber works over only one ring, $R_q = \mathbb{Z}_{7681}[x]/(x^{256}+1)$, which provides flexibility (e.g. performing polynomial multiplication) to sacrifice security (from 128-bit to 102-bit) to improve performance and communication size (33%) (by only changing k from 3 to 2). This flexibility is exclusive to Kyber (Module-LWE schemes); in Ring-LWE schemes, changing the security parameters results in building a new ring R_q and ring operations. Kyber has been submitted to the NIST PQC standardization call for as Kyber512, Kyber768, Kyber1024 at 102, 161 and 218-bit security (NIST security category 1, 3 and 5) [16].

Lu et al. [147] 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. However, 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 using the centered binomial distributions. LAC is proposed with three set of parameters that are much more expensive to break than AES128, AES192, and AES-256 (NIST security category of 1, 3 and 5).

Smart et al. [211] propose `LIMA` (**L**att**I**ce **MA**thematics), a family of IND-CCA and IND-CPA RLWE-based PKE (based on based on `LP` [143]) and KEM schemes, to the NIST PQC standardization call as 6 set of parameters with claimed post-quantum security from 143 to 274 (NIST security category of 1, 2, 3 and 5). Authors employ the Fujisaki-Okamoto [96] and Dent transform [68] to obtain IND-CCA PKE and IND-CCA KEM schemes. In addition to power-of-two cyclotomic rings (`LIMA-2p`), authors propose safe-prime cyclotomics (`LIMA-sp`) that reduces the probability of subfield attacks by scarifying the efficiency compare to the

power-of-two cyclotomics. In order to avoid decryption failure, authors perform rejection sampling (from a centered binomial distribution) at the encryption stage which makes the implementation to be non-constant time. In order to perform polynomial multiplication with FFT, large modulus should be selected for `LIMA-sp`.

Phong et al. [176] present `LOTUS` (**L**earning with err**O**rs based encryption with chosen ciphertex**T** for po**S**t quantum era) IND-CCA2 secure LWE-based PKE (`LOTUS-PKE`) and KEM (`LOTUS-KEM`) with 128-bit, 192-bit and 256-bit security (NIST security category of 1, 3 and 5). Knuth-Yao algorithm [136] is employed to sample the error term from discrete Gaussian distribution. In order to reduce sampling's overhead, DDG tree is built online and probability matrix is stored column-wised. Besides the reference and optimized implementations, vectorized implementation (employing AVX2 vector instructions) of LOTUS-PKE and LOTUS-KEM are provided.

`NTRU-KEM` [126], an IND-CCA2-secure KEM based on NTRU cryptosystem with 128-bit classical security, is the first timing attack resistant NTRU software thanks to its constant-time noise sampler. NTRU-based KEM has active security which allows parties to cache the ephemeral keys, however passive secure key exchange mechanisms like NewHope and Kyber must not use cached values. Compared to NewHope (255-bit PQ security), NTRU-KEM (123-bit PQ security) improves (secret key size, public key size,ciphertext size) by $(20\%, 37\%, 37\%)$ and halves the required clock cycles for encryption/encapsulation step. However, it increases required clock cycles for key generation and decryption/encapsulation by a factor of 3.47.

Plantard [177] presents `Odd-Manhattan`, a IND-CCA KEM by using Dent transform [68] on IND-CPA PKE, at 126,192 and 256-bit security (NIST security category 1,3 and 5). `Odd-Manhattan` is based on the α-*Bounded Distance Parity Check (BDPC*α) [150], which impose a considerable increase in time and size of key generation, encryption, and decryption. To mitigate the time overhead, computational reuse, i.e., store the results of the k consecutive

additions (constant time) in memory, with notable memory penalty has been employed.

Garcia-Morchon et al. [99] introduce `Round2`, a family of CCA-PKE (`Round2.PKE`) and CPA-KEM (`Round2.KEM`) based on the general learning with rounding (GLWR) problem. By having d (dimension), n (system parameter), q (large modulus), p (rounding modulus) $\in \mathbb{Z}^+$ where $q \leq p$ and $n \in 1, d$, if $n = 1$, instantiated scheme is based on the LWR problem, while $n = d$ ($n + 1$ is prime) results in a RLWR based scheme. Round2 provides two set of parameters including Unified-Round2 (`uRound2`, q is a power of 2) and NTT-Round2 (`nRound2`, q is prime, $n = d$ ($n + 1$ is prime)). With uRound2, schemes can be seamlessly instantiated from LWR or RLWR [23] (for all NIST security levels), both $n = 1$ and $n = d$, with the same code which provides agility (i.e., switching from RLWR-based schemes to LWR-based schemes without recompilation). GLWR, compare to LWE, results in decreasing random data generation due to avoiding sampling from the non-uniform noise distribution; besides, the required bandwidth is reduced since fewer bits are needed per coefficient. Secret terms can be either a sparse-trinary (reduces the probability of error in decryption) or uniformly sampled in $\mathbb{Z}_q^d$. In order to have a unique implementing for LWR and RLWR, a common multiplier that implements polynomial multiplication as the matrix multiplication is employed. Compare to NewHope [11] and Kyber [43], RLWR-based uRound2 requires smaller the public-key and ciphertext in total for NIST security category 5. Over the same ring as NTRU-KEM scheme, Round2 bears better speedup due to its faster key generation. Performance evaluation of the reference implementation is performed on Intel Core i7 2.6GHz.

D'Anvers et al. [64] propose `SABER`, a family of Module-LWR based IND-CPA PKE and IND-CCA KEM schemes including `LightSaber-KEM`, `Saber-KEM` and `FireSaber-KEM` with 115, 180 and 245-bit security (NIST security category 1, 3 and 5). Integers are chosen to be the power-of-two modulus which results in avoiding explicit modular reduction and relaxing complicated sampling methods (e.g., rejection) by efficiently, constant time, sampling from

a (modulo power 2) uniform distribution; however, power-of-two modulus prevents using the 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 [12].

Hamburg [114] introduces *ThreeBears*, a family of IND-CPA and IND-CCA KEM schemes adopted from Kyber [43] and based on integer Module-LWE (`ILWE`) [58] problem. *ThreeBears* includes *BabyBear*, *MamaBear* (recommended)and *PapaBear* with NIST security security category of 2, 4 and 5, respectively. For each scheme, deterministic CCA-secure (with `FO` transform) and ephemeral (without `FO` transform) implementations are presented. In order to reduce the memory footprint, the private key is rapidly generated by expanding a seed; similarly, public key and large public matrix (uniformly at random sample each element) are generated modulus N where N is a large Mersenne prime. Sampling the noise is performed by expanding a seed to only 1B per digit. Although the Saarinen's error correction [201, 202] can notably improve *ThreeBears* security, author prefers to use Melas BCH code as the two-error-correcting code to maintain the code simplicity. To preserve simplicity, NTT is not used which results in slower integer arithmetic, on devices without vector unit support in particular. Performance analysis of the *ThreeBears* implementations are provided on Intel Skylake, ARM Cortex-A8 and ARM Cortex-A53. With 15% smaller ciphertext and public key size, *MamaBear* (resp. *PapaBear*) is stronger than *Kyber-Paranoid* (resp. Hila5 [201, 202] and NewHope [11]).

Steinfeld et al. [215] 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 [194]. Schemes are tightly and provably secure based on the hardness of Polynomial-LWE problem over the polynomial ring $\mathbb{Z}[x]/f(x)$ where f is the member of a large group of ring polynomials. Titanium is a middle ground scheme that achieves a trade-off between

security and efficiency such that, in terms of ciphertext size and performance, it is superior to *Frodo* [42] but inferior to *Kyber* [43]. Among 6 suggested parameter sets, $Std128$, $Med160$, $Hi192$ and $Super256$ satisfy minimum security specified by NIST (1,1,3 and 5, respectively). However, the security analysis of Titanium assumes the classical random oracle model. NTT and binomial difference error distribution are used for polynomial multiplication and error sampling; secret key coordinates are sampled uniformly at random over $\mathbb{Z}_q$. It should be mentioned that error correction or reconciliation techniques are not employed in Titanium. In addition to the reference and optimized implementation, authors provide performance analysis of vectorized implementation (AVX2) on Intel i7-7700K.

Digital Signature. Lyubashevsky [148] presents Short Integer Solution (`SIS`) problem based digital signature schemes adopted from Fiat-Shamir transformation. Lyubashevsky improves this scheme by establishing it efficiently for Ring-SIS and Ring-LWE which culminates in a smaller signature and key size [149]. BLISS signature [79] 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 [20] 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. [185, 187] evaluate implementations of various lattice-based schemes, including RLWE-based public key encryption schemes and `BLISS` [79] on the 8-bit AVR micro-controllers. They review various NTT algorithms (written in C) and optimize (using assembly language and ignoring zero coefficients) polynomial multiplication (column-wise) for ideal lattice-based cryptography schemes. However, using precomputed twiddle factors in NTT computations requires more memory footprint. Official release code of Keccak [33] for AVR is used for the random oracle which is needed for signing and verification. Other

optimizations offered by the authors are removing bit reversal step during polynomial multiplication and applying the Gaussian sampling in a lazy manner. Compared to RLWE encryption, BLISS needs larger standard deviation for sampling from Gaussian distribution; candidates for Gaussian sampling are CDT sampling [40] with binary search, which results in large tables, and Bernoulli [41], that impose remarkable performance overhead. Consequently, for Gaussian sampler, KL-convolution sampler [180] is used which consume 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 through 2015.

BLISS-B [78] (with the same security level) improves the performance of original BLISS 2.8 times by employing the ternary representation of polynomials in order to shorten the length of the random numbers. During the key generation, keys will not be rejected which leads to 5-10 times enhancement of key generation step runtime. 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 could not be used in BLISS, BLISS generated keys are compatible with BLISS-B.

Oder et al. [171] present an efficient software implementation of BLISS on ARM Cortex-M4F to optimize the throughput along with minimizing memory footprint. Authors evaluate the efficiency of a variety of Gaussian samplers including the Bernoulli, Knuth-Yao, and Ziggurat [47]. In order to improve NTT computation, assembly level optimization along with precomputed coefficients are employed. They conclude that Knuth-Yao sampler is the best candidate for large devices, while for constrained devices Bernoulli is more favorable.

Güneysu et al. [108] present a highly optimized SIMD implementation of `GLP` signature [105] and implement it on Intel's Sandy and Ivy Bridge processors. In the proposed scheme, the Gaussian sampler is replaced with the uniform sampling from {-1,0,+1}; to benefit from the `AVX`, each 512 double-precision floating-point array of coefficients is 32-byte aligned. Besides,

modular reduction of the coefficients is performed in a lazy manner. However, the signature size and security level of the implemented scheme are inferior to BLISS.

El Bansarkhani and Buchmann [88] implement the first software implementation (space and speed optimized) of GPV signature scheme [101] by employing the Micciancio and Peikert (MP) trapdoors [153] on the Sun XFire 4400 server equipped with 16 Quad-Core AMD Opteron. Besides the matrix version, a much faster Ring-LWE based variant of the scheme is provided which has around 3-6 and 3-9 times better speed than matrix version for sign and verification steps, respectively. Due to the small number of stored entries, instead of rejection sampling, the inversion transform method is used for discrete Gaussian sampling during integer key generation; however, rejection sampling [101] is used in the randomized rounding. It worth mentioning that for random oracle SHA256 and a pseudo-random number generator from [48] are used.

Dagdelen et al. [63] propose a fast software implementation of the BG signature, with optimized rejection sampling, on an Intel processor with AVX and an ARMv7 with Neon vector instructions support. Small performance degradation is observed by employing 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. [40, 41] propose efficient software implantation of lattice-based GLP and BLISS authentication protocols for resource-constrained smart cards and micro-controllers (ARM and AVR). Authors perform a design space exploration by choosing different parameter sets for FFT and Gaussian Sampler (Knuth-Yao and Bernoulli) along with the various PKE schemes. They conclude that lattice-based schemes are efficient enough to be implemented on the constrained devices.

Alkim et al. [10] introduce `TESLA` (a tightly secure signature in random oracle model resulted) by a tight reduction to LWE-based problems on the standard lattices and implement

it on Intel Core-i7 4770K (Haswell). Their proposed design is adopted from BG signature [20] which is faster and smaller than the same scheme in [63] due to employing parallel matrix-vector multiplication and lazy reduction. Authors propose two variants TESLA-128 and TESLA-256; the former one, TELSA-I, is not quantum resistant, while the latter, TELSA-II, provides the first lattice-based digital signature with 128-bit security against quantum computers. Large public key size (about 1 MB) makes it impractical to implement. Same authors present a fast, small, and provably secure Ring-LWE based software implementation of TESLA, that uses uniform sampling, on the same platform which reduces the key size about three orders of magnitude [7]. The propose Ring-TESLA benefits from the AVX2 instructions, which has a one cycle throughput for eight doubles integers. Instantiation from the Ring-TESLA leads to the rejection of valid signatures in the verification stage due to a problem in the parameter selection which is solved in [56] by new parameter selection method and in [26] by altering the algorithm. Based on the claims in [26], TESLA and Ring-TESLA are using global parameters which results in employing a fixed lattice for all the signatures that could weaken the signature scheme. A recent version of TESLA [10] fixes the problem by adding a new condition to the signing step which results in dropping the speedup, creating a more complex signature scheme, with less success in signing. Barreto et al. [26] introduce TESLA#, a high-performance version of Ring-TESLA, on Intel Core i7-4770 Haswell processor, which resolves the security problems of TESLA. Further improvement is achieved by designing a more efficient Gaussian which accelerates the key generation step along with avoiding to store all 32 bits of coefficients of the polynomial. Bindel et al. [36] propose qTESLA a family of (provably existentially unforgeability under chosen-message attack (`EUF-CMA`) secure in the quantum random oracle model) Ring-LWE based digital signature schemes, including qTESLA-128, qTESLA-256 and qTESLA-192 with NIST's security categories of 1, 3 and 5. qTESLA adopts a simpler version [26] of the bimodal Gaussian sampler [79] that is only employed in key generation. Although qTESLA performs polynomial multiplication by NTT, it is compatible with the schoolbook algorithm. qTESLA uses cSHAKE

[132] to deterministically generate the random bits for driving the seeds in the key generation and generation of a new polynomial for every key pair. In the signing step, qTESLA employs SHA-3 as the hash function and cSHAKE as the pseudo-random function. Contrary to the Ring-TESLA, qTESLA is secure against cache side channel attacks by applying countermeasures introduced in [38]; however, qTESLA is vulnerable to fault attacks [37] similar to Ring-TESLA.

BLZZRD [199], a lattice-based signature based on BLISS-B [78] is implemented on an Intel Core-i7 Haswell processor with small signature size but with costly signing step. Authors achieve optimal compression for discrete Gaussian distribution by using Binary Arithmetic Coding (`BAC`) which leads to a more compact signature compared to advanced Huffman-based signature compressors. Further security improvement is gained by prohibiting leak of information about the execution time and power consumption of the arithmetic operation by applying randomization which makes the signature resistant to timing and power attacks. Additionally, masking properties of Gaussian samples is achieved by randomizing and combining multiple numbers the of sample vectors.

`Dilithium` [81], a simple and efficient digital signature scheme resistant to the lattice reduction attacks (with the same conservative security parameter in [11]) is adapted from the designs in [105] and [20] that uses Fiat-Shamir Abort Framework [148] which is secure in random oracle model (no security proof in quantum random oracle model is presented). Authors implement `Dilithium` and its variant `Dilithium-G` on Intel Core-i7 4770k with comparable efficiency to BLISS. In [105], *hints* are generated (by the signer to help verifier to verify the signature) to make the signature smaller; `Dilithium` improves hint-generation and halves the public key size with less than 5% increase in the signature size. In fact, authors set *total size* = *signature size* + *public key size* as their size parameter. `Dilithium` over ring of $Z_q[x]/[x^n+1]$ ($n = 256$,$q = 2^{23} - 2^{13} + 1 = 8380417$) has slightly bigger *total size* than BLISS (over ring of $Z_q[x]/[x^{1024}+1]$) with the same security level. `Dilithium` samples

polynomial noises from the uniform distribution S_η in $[-\eta, +\eta]$ where η is in the range of 3 to 7 for very high secure to weakly secure scheme, respectively. However, `Dilithium-G` extract noises from Gaussian sampler which results in better security but vulnerable to timing attacks. Rejection sampling is the same for both schemes as if individual coefficients of a signature is not within a certain range, signing procedure must be restarted. `Dilithium` employs the standard NTT-based polynomial multiplication, however in vectorized version, Dilithium uses integer instructions instead of floating point vector instructions [11].

`Dilithium` has been submitted to the NIST PQC standardization call at three security levels (all use the same ring) including *medium*, *recommended* and *very high* (NIST security category 1,2 and 3) [80]. Dilithium is tightly secure in the quantum random oracle model based on the *Module-LWE*, *Module-SIS* and *SelfTargetMSIS* [134] problem (adopted from combined security of $MSIS$ problem and hash function H). Signature size of Dilithium is about 2× bigger than that of BLISS [79] and [82] (smallest schemes among lattice-based digital signatures) that use discrete Gaussian sampler which Dilithium avoids. However, compare to the most efficient lattice-based digital signature schemes that avoid Gaussian sampler, Dilithium achieves 2.5× smaller public key size. Coole-Tukey and Gentleman-Sande butterflies are used in NTT and inverse NTT, respectively, to perform the polynomial multiplication in which Montgomery reduction is used after multiplication (avoid reduction after addition and subtraction). Vectorized (AVX2 instruction set on Intel Core i7-4770K) version of NTT gives 4.5× speed improvement over the (reference) integer NTT implementation which is 2× faster than floating point NTT [11]. 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 in parallel.

Fouque et al. [93] propose `FALCON`, a family of compact lattice-based hash-and-sign digital signature schemes with quantum security of 103, 172 and 230 bits (NIST security category

1,3 and 5) with the main goal of minimizing the *total size = signature size + public key size*. FALCON is the result of combining GPV framework [101], NTRU lattices [115] and Fast Fourier sampling [84]; provably secure NTRUSign is built by combining the GVP framework and NTRU lattices [214]. Instantiation of the GPV IBE over NTRU lattices is presented in [82] which can be transformed to FALCON by employing the Fast Fourier sampling in private key operations. NTRU lattices along with the capability of the message recovery (entirely from the signature) result in compactness of the FALCON. Verification step in FALCON is relatively fast and simple and can be performed by a hash function followed by NTT operations. FALCON uses double precision floating-point arithmetic in signing which can be challenging to implement on the devices without floating point units. Another downside of the FALCON is the extensive use of the discrete Gaussian sampling over the integers which is hard to protect against the timing and side-channel attacks. FFT over the complex numbers is used for private key operations, while public key operations and key generation are performed using NTT over $\mathbb{Z}_q$. FALCON uses bimodal Gaussian in the reject in the sampler, ChaCha20 as the PRNG and SHAKE-256 as XOF for all security levels. FALCON can be easily transformed into an IBE scheme [82]. Authors only provide the performance evaluation of the reference implementation on an Intel Core i7-6567U CPU with 15% of error margin due to not disabling the boosting feature of the processor. Falcon has the smallest *total size* among all the post-quantum digital signature schemes at the same security level.

Chen et al. [50] present `pqNTRUSign`, a modular lattice-based hash-then-sign digital signature scheme (introduced in [117]) at post-quantum security of 149-bit (NIST security category 5). pqNTRUSign provides sampler agility as the user can choose the sampler based on the design goal; constant time uniform sampling for the security and (bimodal) Gaussian sampling for the performance goals, respectively. The key generation is the same for both set of parameters (Gaussian-1024 and Uniform-1024); other steps have different implementations for various parameter sets. The public key, forgery and transcript security are provided by NTRU assumption, LWE problem over NTRU lattices and rejection sampler,

respectively. Like `NTRUEncrypt` [229], Box-Muller [45] 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.

2.1.3 Hardware Implementations of Lattice-based Cryptographic Schemes

Nejatollahi et al. [159] propose the first domain-specific accelerators for ideal lattice-based schemes with the case study of BLISS-BI [78] and NewHope [11]. Authors present a quick design flow that performs exploration and the design of programmable accelerators that leads to on average 35% and 50% improvement in the latency and energy-delay product. Authors create a programmable accelerator for NTT that can be employed in any scheme, with any set of parameters, that uses Gentleman-Sande butterfly; the `Keccak-f[1600]` accelerator is suitable for any classical and post-quantum cryptographic scheme as the heart of the SHA3.

Public Key Encryption. Göttert et al. [103] propose the first hardware implementation of Ring-LWE based public key encryption on the Xilinx Virtex-7 FPGA. Due to the large area occupation of the full Ring-LWE public key encryption scheme, only LWE-polynomial variants are chosen to be implemented. Proposed implementations are based on LP lattice-based encryption scheme [143] which achieve 316 times higher throughput compared to software implementation. Using a fully parallel architecture (which makes the design remarkably spacious), high throughput is achieved by minimizing required clock cycles in computing the NTT. The primary optimization metric is the performance for which they show speedups for encryption and decryption schemes by factors of 200 and 70, respectively, in comparison to the software implementation with the same level of security.

Pöppelmann and Güneysu [182] provide a flexible Ring-LWE encryption engine in which one core is used to perform key generation, encryption, and decryption steps with the primary

goal of optimizing throughput per unit of area. Besides, by applying optimizations including different encoding technique and removing some LSBs of the ciphertext coefficients, encryption engine, which is 60 times smaller than the design in [103], it could be fit on the Xilinx Spartan-6 FPGA with three times slower encryption step. They employ a Gaussian sampler with relatively low precision, 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. [196] implement a compact Ring-LWE crypto processor on Virtex 6 FPGA where they optimize NTT multiplication by reducing the fixed computation and pre-scaling overhead. Authors suggest combining pre-computation stage and NTT computation. Besides, NTT Memory access is minimized by storing two coefficients in a single word, processing two pairs of coefficients together, and eliminating idle cycles. Small lookup tables are used in the Knuth-Yao discrete Gaussian sampler [197] which leads to more compact and faster sampler than [183].

Pöppelmann and Güneysu [183] implement the smallest lattice-based encryption engine on Spartan-6 and Virtex-5. Compared with the high-speed implementation of [182], this is one order of magnitude slower due to non-applicability of using NTT (using DSP-enabled schoolbook polynomial multiplier). Besides, a considerable area is saved by using specific modulus, a power of 2, by which modular reduction is almost cost-free. Further area saving is achieved by using a Bernoulli distribution [79] with small precomputed tables in order to optimize simple rejection sampling by eliminating computing of $exp()$ function.

Howe et al. [122] present the first and the only, hardware implementation of lattice-based encryption engine based on learning with error problem over standard lattices on the lightweight Spartan-6 FPGA. The main concern is optimizing the area, while the scheme maintains the

balance between area and performance by using a larger Gaussian sampler. The proposed encryption engine is smaller in comparison to the design of [103]; besides, it can closely compete with the encryption scheme of [182]. To maximize the performance, authors use a larger Gaussian sampler, Bernoulli sampler, which generates samples in parallel with no adverse effect on the critical path.

Reparaz et al. [192] 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.

Digital Signature. Howe et al. [123] provide evaluation and summary of practical instantiations of digital signature schemes based on lattice problems on different platforms. Evaluation metrics are the secret key, public key, and signature size. Additionally, they give a survey of various implementations of basic blocks including NTT and sampling. Authors evaluate Bernoulli, Ziggurat, Knuth-Yao, and cumulative distribution table (CDT) variants of the Gaussian sampler.

Güneysu et al. [105] implement an efficient lattice-based signature scheme (GLP signature) on a Xilinx Virtex 6 FPGA which is the first practical lattice-based signature scheme that could resist transcript collision attacks. Author removes the need for Gaussian noise sampling by using the rejection sampling which leads to hiding the secret key contained in each signature. Security of the proposed scheme is lower than standard lattice-based signatures due to building the hardness assumption based on the Decisional Compact Knapsack problem. Because of the regular structure of the schoolbook algorithm, authors achieve high speed and small size for implementation of the polynomial multiplier. Compared with BLISS, the proposed signature is sub-optimal in terms of signature size and security level.

Pöppelmann et al. [180] implement a high throughput hardware implementation of BLISS [79] on Xilinx Spartan-6 FPGA. Authors improve and parallelize the column-wise schoolbook multiplier presented in [105]. Authors employ an efficient CDT based Gaussian sampler with large tables. To improve the performance of the CDT sampler, author deploy a decreasing number of comparisons by improving the binary search and reducing the size of precomputed large tables by using an optimized floating-point representation (adaptive mantissa size) with negligible effect on the performance. Authors provide enhanced CDT which uses two smaller samples (Peikert convolution theorem [173]). A standard CDT needs table of the size at least $\eta \times \tau \times \lambda = 215.73 \times 13.4 \times 128 = 370kb$ while enhanced CDT needs around $23\times$ smaller table. Authors evaluate performance and resource consumption of BLISS-I $(n = 512, q = 12289)$ by employing the CDT and two parallel Bernoulli samplers and conclude that CDT consumes less FPGA resources than Bernoulli; besides, enhanced CDT achieves 17.4 Million Operation per Seconds (`MOPS`) which is $2.3\times$ more than that of Bernoulli sampler. Based on the results, performance of enhanced CDT is almost the same for `BLISS-I` ($\sigma = 215$), `BLISS-III` ($\sigma = 250$) and `BLISS-IV` ($\sigma = 271$).

Güneysu et al. [107] 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 [105] but with major improvements. Compared with [105], instead of Schoolbook multiplier, authors employ a parallelized NTT for polynomial multiplications (the most time-consuming part of the digital signature), which leads to smaller and faster signing/verification engines. Authors develop a flexible processing core with VHDL that could be configured as either signing or/and verification engine which does not impose any overhead to provide flexibility. The signing step breaks into three separate blocks, including lattice processing engine, random oracle, and, sparse multiplication along with compression unit, which are running in parallel. The digital signature processor is based on the lattice processor for the public key encryption scheme in [182].

Key Exchange. Oder et al. [223] propose an area optimized constant time implementation of `NewHope-Simple` [12], on Xilinx Artix-7 FPGA, with decent performance level. With the same post-quantum security level as `NewHope` (128-bit), server and client work with clock frequency of 125 and 117 MHz, respectively. Authors design two separate modules for the client and server sides which forces an embedded system to be only either a server or a client, hence results in the lack of re-usability as a disadvantage. For the sake of the area optimization, 512 butterfly operations are performed serially, while they can be performed in parallel.

Kou et al. [139] provide a high performance pipelined implementation of NewHope [11] on Xilinx Artix-7 FPGA which is 19.1× (4 ×) faster (bigger) than the hardware implementation of the NewHope-Simple [223]. In order to improve the performance, NTT operations are computed using four butterfly units; besides, Longa-Naehrig modular reduction [145] is used instead of Barrett reduction.

2.1.4 Hardware/Software Implementations of Lattice-based Cryptographic Schemes

Because of the probabilistic inherent feature of rejection sampling in the lattice-based schemes, the probability of generation of an invalid signature exists that can be minimized by precomputation. Aysu et al. [18] divide the signature scheme in hash-based cryptographic signatures into two separate phases in order to minimize the energy and latency of signature generation. During the offline phase, input (message) independent computations, for instance, key and random number generation, are performed, and results are stored in a memory buffer as coupons. Subsequently, the output is generated using the precomputed coupons and the input (message) during the online phase. Employing the same idea, Aysu et al. [19] implement a latency optimized lattice-based signature with hardware/software

co-design technique. The main objective is to optimize latency which is achieved by focusing on the signature generation step on the embedded device. On the other hand, the verification step is performed on high-performance platform servers. Signature generation scheme consists of two separate phases including offline and online phases which are performed on NIOS soft-core as software, and Altera Cyclone-IV FPGA as hardware, respectively. Hardware is responsible for low latency hash function and polynomial multiplication; however, the software part computes and stores polynomials.

2.1.5 DSP Implementation

In order to perform the multiply-accumulate (`MAC`) operations of matrices in the encryption scheme, Howe et al. [122] utilize a dedicated DSP48A1 unit of the Spartan-6 FPGA to achieve an area optimized hardware implementation of standard LWE based encryption engine. The primary goal is optimizing area, while the scheme maintains the balance between area and performance by using a larger Gaussian sampler.

2.2 Conclusion

Lattice-based cryptographic algorithms promise to tackle the challenges posed by deployment across diverse computing platforms, as well as for diverse use cases within reasonable security, performance, and energy efficiency guarantees. Numerous schemes and implementations tackle different trade-offs, such as memory footprint, security, performance, and energy, are mapped on a variety of platforms and apply to specific use cases. However, current designs are still deficient in addressing the need for agility, which is paramount to tackle the needs of emerging business models at the computing platform level. Besides, securing such platforms against physical attacks is a topic that needs to be researched.

Chapter 3

Simulation-based Cache-assisted Polynomial Multiplier

3.1 Background

A lattice $L \subset \mathbb{R}^n$ is the set of all integer linear combinations of basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$. i.e., $L = \left\{ \sum a_i \mathbf{b}_i \ : \ a_i \in \mathbb{Z} \right\}$. In particular, L is a subgroup of $\mathbb{R}^n$ that is isomorphic to $\mathbb{Z}^n$. The cryptography based on lattices exploits the hardness of two problems: Short Integer Solution (SIS) and Learning With Errors (LWE). Cryptosystems based on the LWE problem, the most used one, have their foundation in the difficulty of finding the secret key sk, given (A, pk), where $pk = A * sk + e \bmod q$, being pk the public key, e an error vector with Gaussian distribution, and A a matrix of constants in $\mathbb{Z}_q^{r \times n}$ chosen randomly from a uniform distribution. In general, the parameters n, r and q are integers such that $n > 1$, $r > n$ and $q \geq 2$, but for some efficient implementations these numbers are constrained to a limited set of values. For more detailed information about the parameters we refer to the survey done by Oded Regev [190].

LWE, however, requires large keys that could be impractical; for instance, public and secret key size of Frodo [158], LWE-based KEM, at NIST security level 1 are 9KB and 19.8KB, respectively. To overcome this limitation, Lyubashevsky et al. [151] introduced Ring-LWE (RLWE), a derivation of LWE where A is implicitly defined as a vector a in the ring $\mathcal{R} \equiv \mathbb{Z}_q[x]/\langle x^n+1\rangle$. These lattices are called Ideal lattices, while other lattices are called Standard lattices. At the same security level as Frodo, NewHope [179], a RLWE-based KEM, provides 10X smaller public and secret keys. Since implementing a polynomial multiplication requires a notable amount of computational resources, it is implemented using the so-called Number Theoretic Transform (`NTT`), which is a Fast Fourier Transform implemented over a ring. In the rest of this section, we introduce a detailed definition of RLWE and number-theoretic transform.

Notations

In an RLWE encryption scheme, operations are performed on polynomials. We denote polynomials in the time domain by a lower case letter, e.g., x, and whenever we want to access one coefficient of the polynomial, we denote $x(i)$. Polynomials in the frequency domain are denoted by upper case letters. Multiplication of integers is described by $\cdot$ while we use $*$ for multiplication of polynomials. We omit $\cdot$ and $*$ where it is clear from the context.

3.1.1 Ring-LWE Encryption Scheme

The cryptosystems analyzed in this paper are mostly based on a ring-variant of the LWE problem [190, 151]. The hard problem to be solved on the ring $\mathcal{R} = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$ which defines as deciding whether the samples $(a_1, t_1), \ldots, (a_m, t_m) \in \mathcal{R} \times \mathcal{R}$ are chosen uniformly random or whether each $t_i = a_i s + e_i$, being $s, e_1, \ldots, e_m$ small coefficients from the (one-dimensional) discrete Gaussian distribution $\mathcal{D}_\sigma$ [151]. The distribution $\mathcal{D}_\sigma$ is defined such

that a value $x \in \mathbb{Z}$ is sampled from $\mathcal{D}_\sigma$ with the probability $\rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$ where $\rho_\sigma(x) = \exp\left(\frac{-x^2}{2\sigma^2}\right)$ and $\rho_\sigma\ (\mathbb{Z}) = \sum_{k=-\infty}^{\infty} \rho_\sigma(k)$.

The algorithms for key generation, encryption and decryption are presented in Algorithm 1, Algorithm 2, and Algorithm 3, respectively. The key generation uses a constant vector a for obtaining the public and secret key (pk, sk). The encryption procedure takes a binary string μ and releases the ciphers (c_1, c_2), which are used in the decryption to recover the message μ'. The vectors r_1, r_2, e_1, e_2, e_3 are error vectors sampled from a Gaussian distribution.

Algorithm 1: RLWE Key generation

Function $RLWE_{keyGen}(a)$**:**

```
/* Require: Access to a global constant 'a' that was uniformly
   chosen from R_q                                              */
```

$r_1, r_2 \leftarrow D_{\mathbb{Z}^n,\sigma}$
$p_k \leftarrow r_1 - ar_2$
$s_k \leftarrow r_2$
return (pk, sk)

Algorithm 2: RLWE Encryption

Function $RLWE_{enc}(pk, \mu)$**:**

```
/* Require: Access to a global constant a that was uniformly
   chosen from R_q and to a message μ ∈ {0,1}^n                 */
```

$e_1, e_2, e_3 \leftarrow D_{\mathbb{Z}^n,\sigma}$
$\bar{m} \leftarrow Encode(\mu)$
$c_1 \leftarrow ae_1 + e_2$
$c_2 \leftarrow pke_1 + e_3 + \bar{m}$
return (c_1, c_2)

Algorithm 3: RLWE Decryption

Function $RLWE_{dec}(c_1, c_2, sk)$**:**

$\mu' \leftarrow\ Decode(c_1 \cdot sk + c_2)$ **return** μ'

The Number-Theoretic Transform (NTT) transforms a finite field polynomial from the time to the frequency domain. Inverse Number-Theoretic Transform (NTT^{-1}) transform back the polynomial to the time domain. Polynomial multiplication in the frequency domain

has linear complexity instead of quadratic complexity. As the transformation itself has quasi-linear complexity, the overall effort for polynomial multiplication drops from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot \log n)$. The efficient computation of the NTT involves the coefficient ring to contain primitive roots of unity. Formally, a primitive root of unity is defined as follows.

Definition 3.1 (Primitive root of unity)**.** *Let $\mathcal{R}$ be a ring, $n \in \mathbb{N}_{\geq 1}$, and $\omega \in \mathcal{R}$. The value ω is an n-th root of unity if $\omega^n = 1$. The value ω is a primitive n-th root of unity (or root of unity of order n) if it is an n-th root of unity, $\omega^n \in \mathcal{R}$ is a unity in $\mathcal{R}$, and $\omega^{n/t} - 1$ is not a zero divisor for any prime divisor t of n.*

For a given primitive n-th root of unity ω in $\mathbb{Z}_q$, the NTT of a vector $a = (a(n-1), \ldots, a(0))$ is the vector $A = (A(n-1), \ldots, A(0))$ and it is computed as $A(i) = \sum_{j=0}^{n-1} a(j)\omega^{ij} \mod q, i = 0, 1, \ldots, n-1$. The inverse transformation is calculated as $a(i) = n^{-1} \sum_{j=0}^{n-1} A(j)\omega^{-ij} \mod q, i = 0, 1, \ldots, n-1$.

The idea is to transform two polynomials $a = a(n-1) \cdot x^{n-1} + \ldots + a(0)$ and $b = b(n-1) \cdot x^{n-1} + \ldots + b(0)$ into their NTT representations $A = A(n-1) \cdot x^{n-1} + \ldots + A(0)$ and $B = B(n-1) \cdot x^{n-1} + \ldots + B(0)$ and to compute the coefficient-wise multiplication as $C = \sum_{i=0}^{n-1} A(i) \cdot B(i) \cdot x^i$. The result $c = a * b$ is obtained after the computation of the inverse number theoretic transform (NTT^{-1}) to C. For $q = 1 \mod 2n$ the way the result has to be interpreted depends on the input. Two common algorithms of performing number theoretic transform are Cooley-Tukey (CT) [61], produces the result in the bit-reverse order by receiving the input in the correct order, and Gentleman-Sande (GS) [100], receives the input in the reverse order and produces the output in the correct order. We refer to Gentleman-Sande and Cooley-Tukey algorithms as `NTT_CT` and `NTT_GS`, respectively, in the rest of this paper. Employing `NTT_GS` to compute both NTT and NTT^{-1} involves in bit-reverse calculation (NewHope); however, bit-reverse step can be skipped by using `NTT_CT` for NTT and `NTT_GS` for NTT^{-1} (Kyber and Dilithium).

Keccak is a family of cryptographic primitives primarily as a candidate for the NIST SHA-3 competition [34]. The main transformation in the family is the Keccak-f[1600] permutation which is used in the sponge mode in the specification for the SHA-3 hash function. The hash function operates on a state of 1600 bits and is designed for easy implementation in both software and hardware. The state can be considered to be a $5 \times 5 \times 64$ array of bits. Let $a[i][j][k]$ be bit $(5i + j) \times w + k$ of the input, using a little-endian bit numbering convention and row-major indexing. Index arithmetic is performed modulo 5 for the first two dimensions and modulo w for the third. A set of 64 bits with the same x, y coordinates is called a lane. The basic block permutation function consists of 24 rounds of five steps (Figure 4.2).

- θ: In this step, we compute the parity of each of the 320 five bit columns and add it two nearby columns. Functionally it can be described as $a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x][y][z] + \sum_{y'=0}^{4} a[x+1][y][z-1]$
- ρ: In this step we rotate each of the lanes by a fixed number of bits. The transformation is given by $a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2]$ where $t = -1$, if $x = y = 0$ and otherwise $t \in [0, 24]$ satisfying $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 1 & 0 \end{pmatrix}^{t} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
- π: The mapping π is a transposition of the lanes. It is defined as $a[x][y][z] \leftarrow a[x'][y'][z]$, with $x = y',\ y = 2x' + 3y'$
- χ: This is the only non-linear step in the round function given by $a[x][y][z] \leftarrow a[x][y][z] + (a[x+1][y][z] + 1)a[x+2][y][z]$
- ι: This is the round constant addition step. A round constant is added only to the lane with co-ordinates (0,0).

The energy cost of implementation is crucial for constraint devices and servers; energy consumption of classical cryptography has been evaluated by many authors [226, 71, 224, 15].

However, for lattice-based cryptography, previous works focused mainly on limiting the area and improving the performance [22, 106]. However, energy has been left mostly unexplored, except for FPGA implementation [162], ASIC implementation [24], and simulation-based [163] accelerator for NTT, where none of them are cache-based accelerators. The early results of this work were presented in [161].

3.2 Design Flow Steps

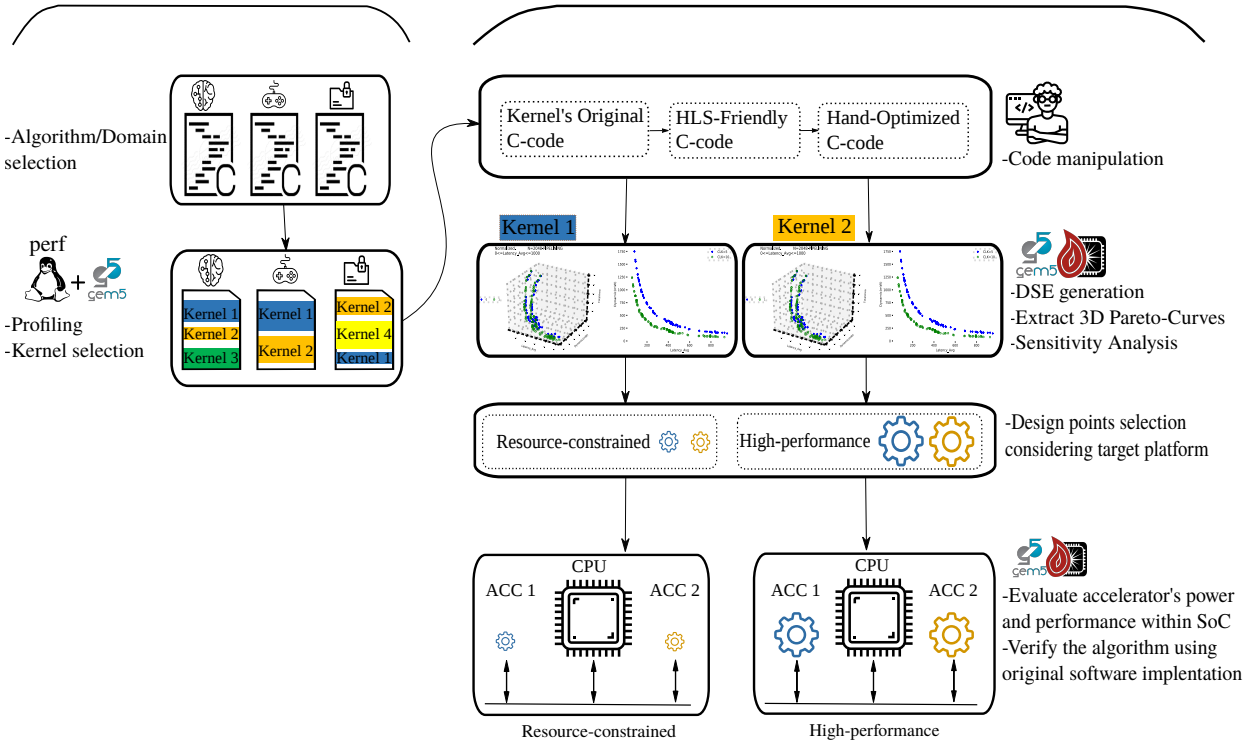


Figure 3.1: Design flow of the accelerator generation.

In this section, we detail the three main steps that compose our design space exploration flow: profiling, early exploration, and evaluation within the target architecture. Figure 4.1 illustrates the design flow steps to generate flexible accelerators for any domain or set of algorithms. The first step is the profiling of the implementations of the target lattice-based schemes. This process is needed to identify the bottlenecks and the kernel to be accelerated.

Profiling is executed by running the algorithm on a host arbitrary machine, e.g., X86 based system, and verify it using a microarchitectural simulator such as gem5 [39]. In the former case, profiling results correspond to collecting performance counters from the execution. In the latter case, profiling means collecting simulation statistics. In either case, the same hot-spots in the protocol are identified. We profile the server and the client portion of the protocol separately to accommodate the specific needs of each end-point. The identified hot-spot functions are candidates for acceleration and are input in our design exploration flow.

In the second step, we used gem5-Aladdin [207] to perform the design space exploration of the candidate functions. gem5-Aladdin is a fast design exploration framework that allows making decisions on the trade-off between performance, power, and area of the identified design points with high accuracy, even avoiding the explicit synthesis step. For standard linear algebra kernels, pre-RTL performance area and power estimation fall within a threshold of 10% compared to the actual synthesis. The results of the `DSE` are stored in an elastic search service, which we develop to keep track of the design points and to compare them in terms of area, power, performance, and complexity. The `DSE` itself consists on the extraction of instruction-level parallelism in each candidate function, and the mapping of such parallelism on a parameterizable set of finite constraints in the hardware, e.g., number of vector lanes, vector lane pipeline depth, ports to memory, and communication to memory (with cache or via streaming) [207]. The introduction of system-level requirements allows us to query the elastic search service and to prune down the set of possible designs further. We use 3D (latency, power, and area) Pareto-optimal fronts to eliminate non-optimal points. The design points that are not pruned will be evaluated in the whole system on the chip.

In the third step, we exploit the integration between gem5 and Aladdin [207], to evaluate the accelerator within the whole SoC. We used this approach to avoid Transaction Level Simulation, emulation or full-system HLD synthesis. The memory communication and the

host processor computation aspects are modeled in a pre-RTL environment. Thanks to the accuracy guaranteed by gem5 and Aladdin [207], this last step allows us to identify a restricted set of design points that quickly reach our design goals.

In this work, we accelerate the execution of lattice-based cryptography algorithms using co-processors with the small instruction set. Based on Figure 4.2, these accelerators are loosely coupled with the main processor in a System on Chip (SoC). Each accelerator has its private caches and operates in slave mode attached to the network o chip.

The accelerators are kernel-level accelerators, where the kernel to accelerate is the bottleneck functions that are common across different families of lattice-based cryptography schemes. For instance, any call to NTT_GS is transferred to the accelerator; if the size of the NTT_GS, i.e., polynomial degree, which determines the security strength, is bigger than the accelerator, we orchestrate the data stream via the cache hierarchy. In other words, the accelerator is a pipelined implementation of the kernel that arbitrary size of a call to the kernel can be offloaded to it.

The choice of this type of accelerator allows us to maintain high flexibility while at the same time, improving performance. This strategy ensures crypto-agility, which is fundamental for the early adoption of post-quantum algorithms. Additionally, it allows us to accelerate several lattice-based cryptography schemes with a small set of co-processors. Both advantages could not have been achieved using designs customized on each scheme.

3.3 Algorithms Profiling

We considered four key encapsulation mechanisms (NewHope, Kyber, R.EMBLEM, and KCL) and a digital signature algorithm (Dilithium). A KEM is a technique to encapsulate a symmetric key using an asymmetric cryptographic algorithm [168]. A digital signature

is a scheme that guarantees the integrity and authenticity of a message. In the rest of the section, we discuss their profiling results.

3.3.1 NewHope

NewHope cryptosystem is a suite of two KEMs based on NewHope-Simple [12], that bases its security on the hardness of RLWE problem. NewHope is the transformation of a Public Key Encryption (PKE) scheme into a KEM using the Targhi and Unruh transforms [220] that implies the use of hash functions, which in the case of NewHope is Shake256 [86]. Also, NTT and binomial samplers are employed to improve the performance of the algorithm. The current submission of NewHope to NIST competition presents two parameter sets, NewHope512cpa/cca, and NewHope1024cpa/cca, with security levels of 101 bits and 233 bits, depending on the length of ring polynomials [179]. Figure 4.4 presents the profile of NewHope1024cpa, referred to as NewHope in the rest of the paper, in the client and server-side. NewHope employs `NTT_GS` for both forward and inverse NTT.

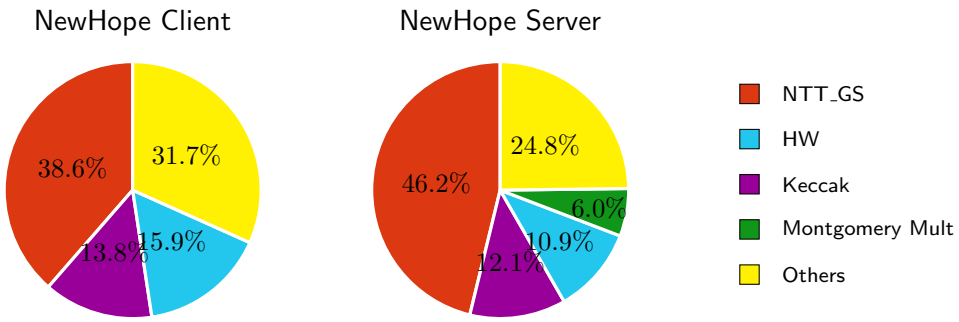


Figure 3.2: Profiling results for NewHope: Client - Server.

3.3.2 Kyber

Kyber is a KEM based on the LWE problem in module lattices [142] which is built using the Fujisaki-Okamoto transform [95]. Kyber uses `NTT_CT` and `NTT_GS` to compute forward and

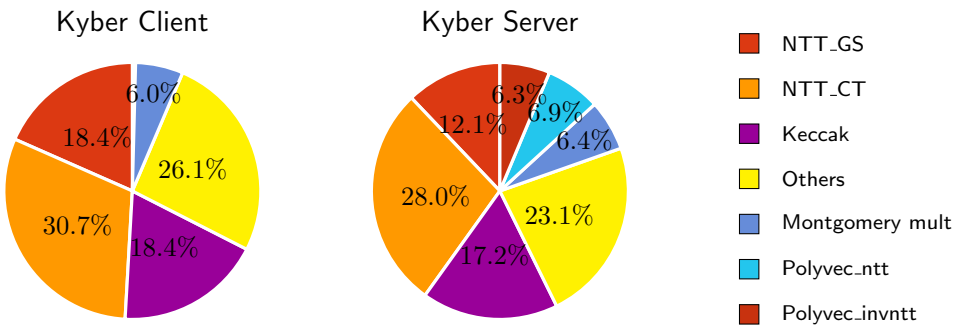


Figure 3.3: Profiling results for Kyber: Client - Server.

reverse NTT, respectively. Kyber is presented in three versions: Kyber512, Kyber768, and Kyber1024 with the quantum hardness 102, 161, and 241 bits, respectively[16]. Due to the similarity, we only analyze Kyber768 in this paper. The profile of Kyber768, referred to as Kyber in the rest of the paper, is shown in Figure 4.3.

3.3.3 Dilithium

Dilithium is a signature algorithm based on "Fiat-Shamir with Aborts" approach [148] that is designed to be easy to implement securely [80]. Its main characteristics are small signature and keys, conservative parameters and modular structure. The most relevant operations in Dilithium are eXtendable-Output Functions (XOF) and polynomial multiplication in a ring. SHAKE128 and Shake256 are used as XOF. Similar to Kyber, Dilithium uses `NTT_CT` and `NTT_GS` to compute forward and reverse NTT, respectively. The profile of Dilithium for signature and verification processes are reported in Figure 3.4.

3.3.4 R.EMBLEM

R.EMBLEM (and its standard lattice variant EMBLEM) is an error-blocked multi-bit RLWE-based (LWE-based) key encapsulation mechanism [204]. Small keys size is achieved by sep-

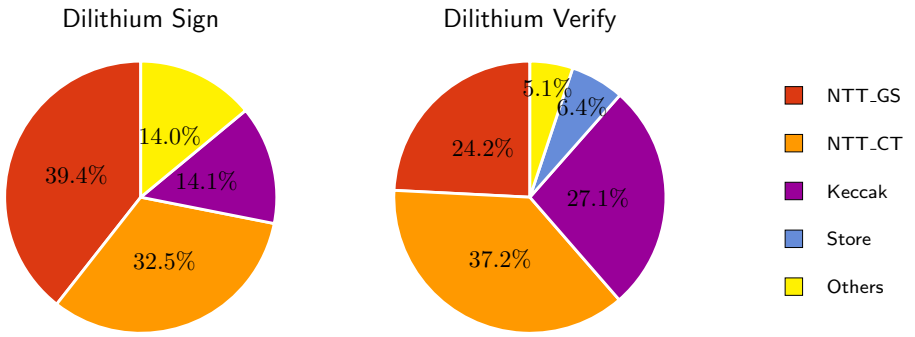


Figure 3.4: Profiling results for Dilithium: Sign - Verify.

arating the message from error and attach the error-blocking bit to each message block. Sampling the secret term is performed uniformly at random in $[-B, B]$. Similar to Kyber and Dilithium, R.EMBLEM uses `NTT_CT` and `NTT_GS` to compute forward and reverse NTT, respectively.

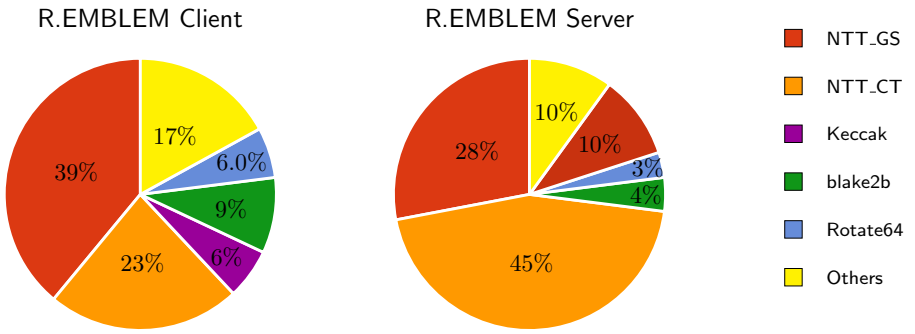


Figure 3.5: Profile R.EMBLEM: Client - Server.

3.3.5 KCL (Key Consensus from Lattice)

Zhao et al. [230] propose a generic construction of the authenticated key exchange, public-key encryption, and key encapsulation mechanism, including asymmetric key consensus (AKCN) and optimal asymmetric key consensus (OKCN). Proposed schemes are based on LWE and its variants over ideal and module lattices with different mathematical structures. Authors propose single error correction (SEC) code to correct at least one-bit error; using the SEC, at

Table 3.1: Size of loop and data types for NTT_GS and NNT_CT over different schemes. It should be mentioned that the data type for Keccak perform operations over `uint64_t` for al the schemes.

Scheme	Loop Iterations	Data type	
		Coeffcients	Twiddle factor
NewHope512cpa/cca	256	uint16_t	uint16_t
NewHope1024cpa/cca	512	uint16_t	uint16_t
Kyber	128	uint16_t	uint16_t
Dilithium	128	uint32_t	uint32_t
R.EMBLEM	512	int32_t	int32_t
AKCN_MLWE	128	int32_t	int16_t

the same security level and error rate, OKCN/AKCN-RLWE based KEX schemes generate 765-bit shared a key with less bandwidth than NewHope (256-bit shared key). Due to the similarity and to avoid the repetition, we provide the results for AKCN-MLWE. AKCN-MLWE uses `NTT_CT` to compute the forward and reverse NTT.

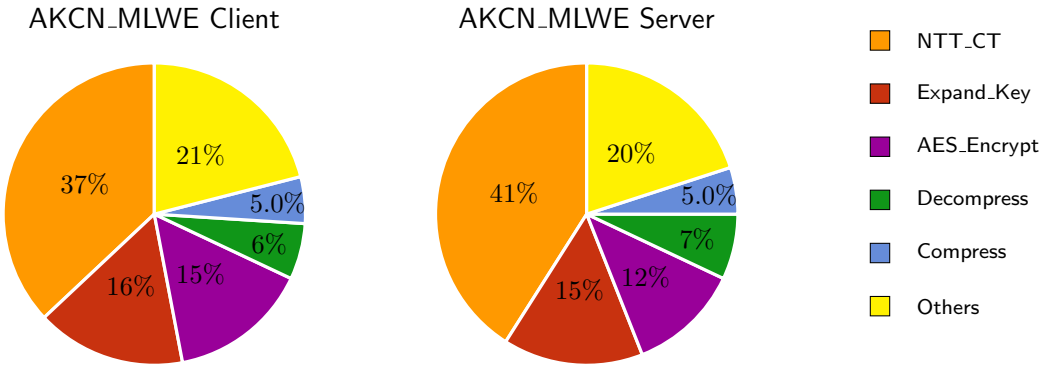


Figure 3.6: Profile AKCN_MLWE: Client - Server.

Table 3.1 shows summarized the size of the common kernels and the data types that they perform the computations. Keccak performs 25 iterations on unsigned 64-bit data. However, for NTT_GS and NTT_CT data types are different over different schemes. One 32-bit accelerator for NTT_GS and another 32-bit for NTT_CT are designed that can support calculation of forward and inverse number theoretic transform over the analyzed schemes with various data bitwidth, which is introduced in this work for the first time.

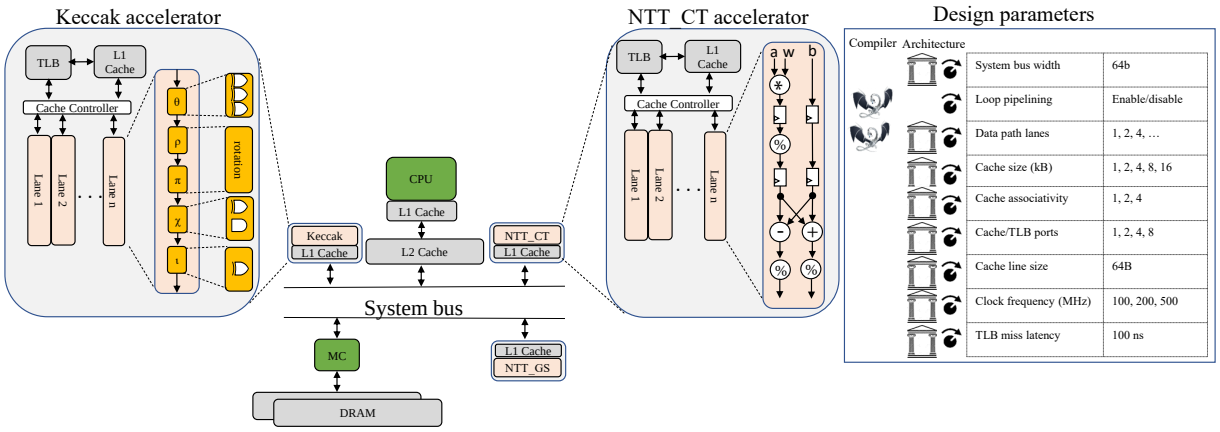


Figure 3.7: Architectural template for generated accelerators for `Keccak`, `NTT_CT` and `NTT_GS` kernels.

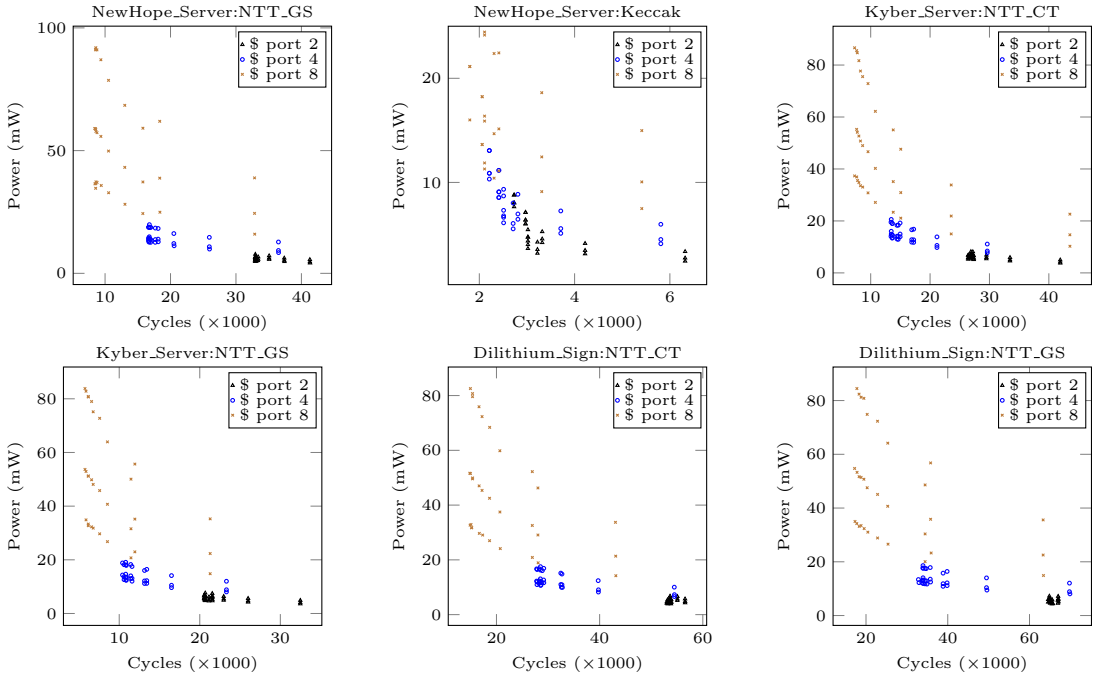


Figure 3.8: The sensitivity of the accelerators' performance and power to the number of cache ports.

3.4 Evaluation

The first step is to identify compute-intensive kernels, Section 4.3, and then search the space of the design and select the desired design points and evaluate them within an SoC. We use the original software implementation of the accelerated algorithms as a baseline for our

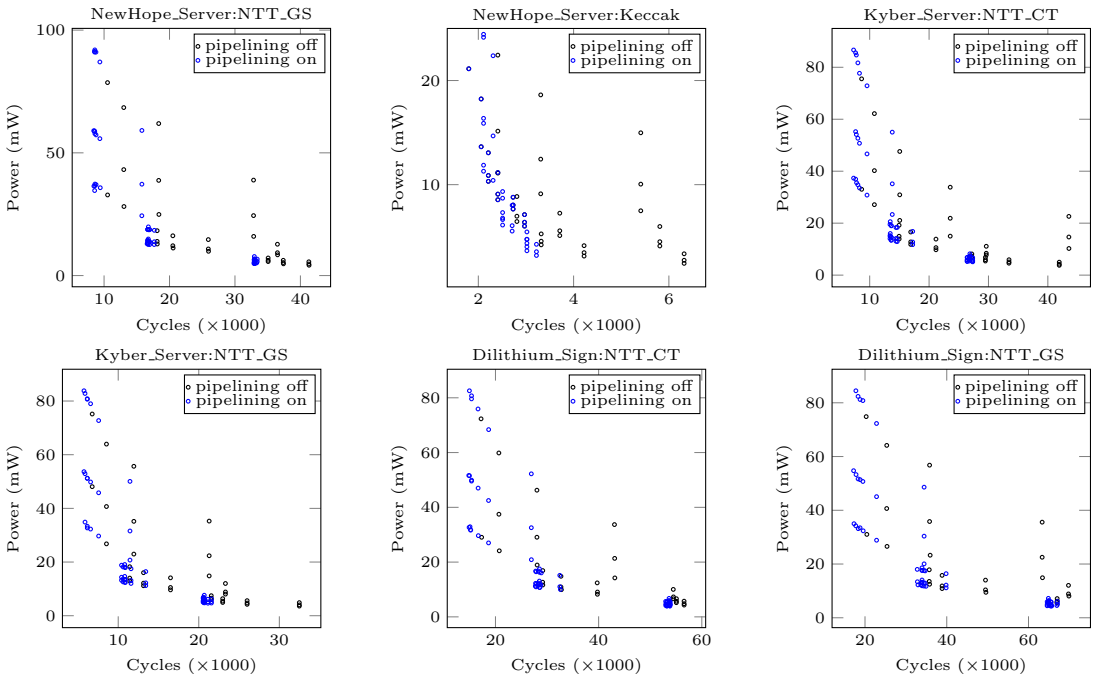


Figure 3.9: The sensitivity of the accelerators' performance and power to the software pipelining.

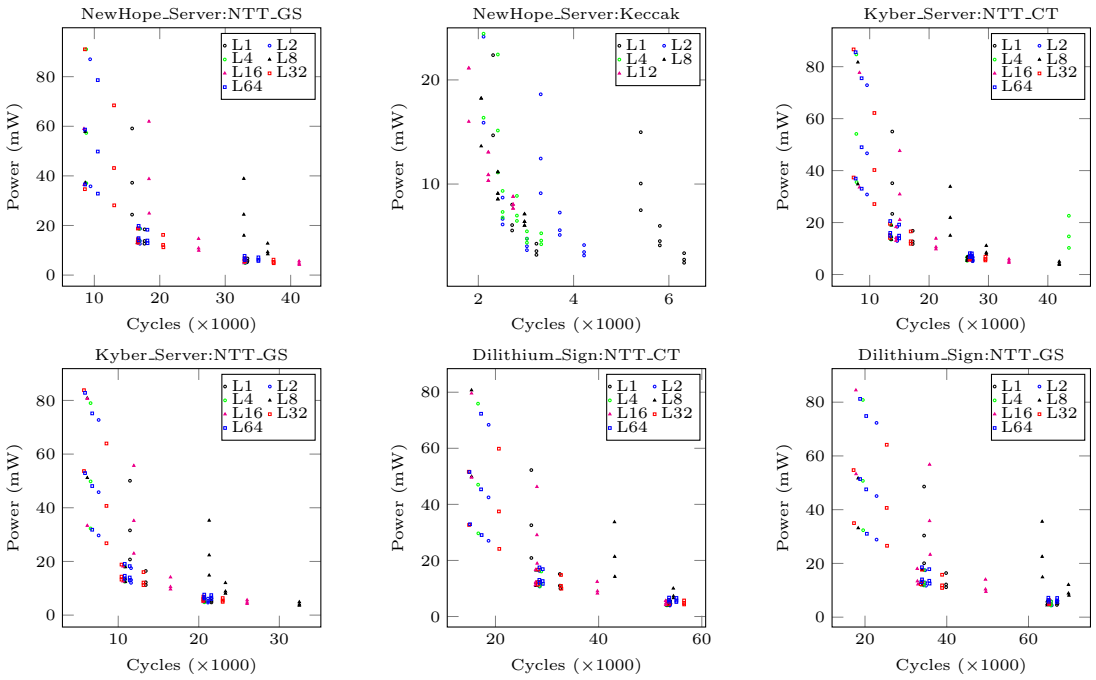


Figure 3.10: The sensitivity of the accelerators' performance and power to the number of lanes.

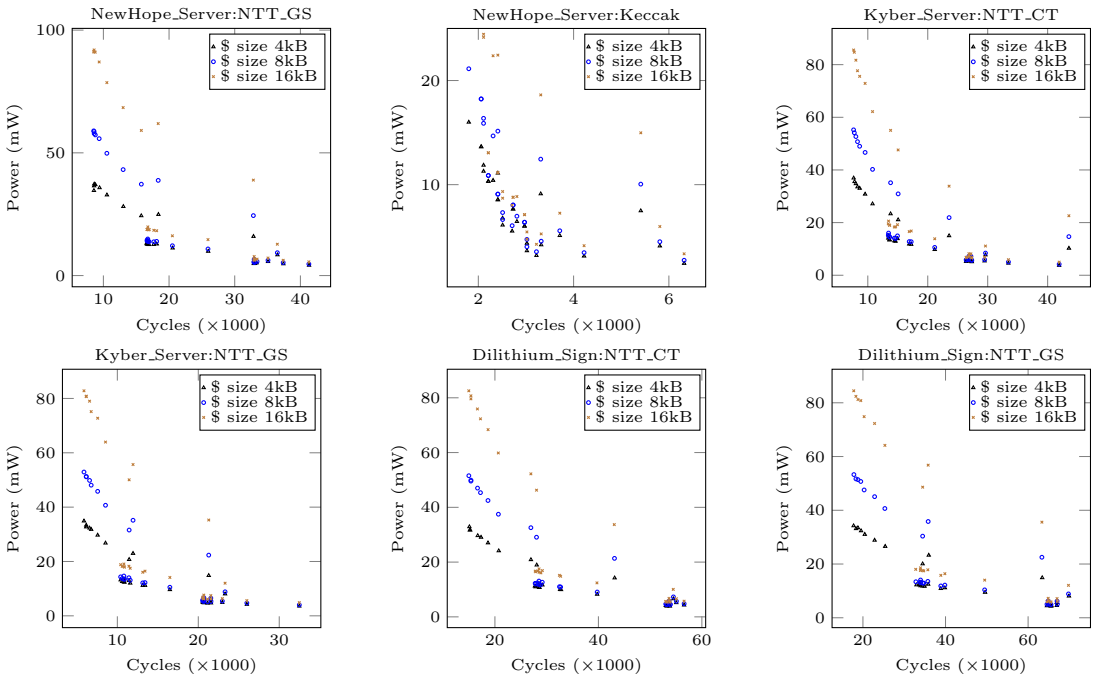


Figure 3.11: The sensitivity of the accelerators' performance and power to the cache size.

evaluation. These implementations are the references for generating the KAT files, which enables us to use them for verifying the accelerator's functionality. To apply our methodology in algorithms the usage of KAT files can be integrated into the tool flow..

Figure 4.2 shows the architectural template of the designed accelerators, which are composed of a set of lanes that implement the primitives of the kernel, a Translation Lookaside Buffer (TLB), one level of cache and the respective controller. Our tool allows analyzing the accelerators without digging into the details of the hardware implementations concentrating on the accelerator functionality.

Based on the profiling results (Figure 4.4-3.6) the top three common kernels that consume most of the cycles are `NTT_CT`, `NTT_GS`, and `Keccak`.

`Keccak` permutation consists of bit-wise arithmetic operations, and each datapath lane of the accelerator has basic components to compute the F[1600] permutation function. The designed accelerator for `Keccak` can be used in any cryptographic scheme to hash a message and

generate pseudo-random numbers. NTT_CT and NTT_GS accelerators are two programmable accelerators that can be employed inside any scheme with arbitrary security parameters to perform the number theoretic transform. Each datapath lane in NTT_CT and NTT_GS accelerators (Figure 4.2) executes the butterfly operation. To perform NTT and NTT^{-1} in NewHope, NTT_GS is used. However, AKCN_MLWE uses NTT_CT to compute both NTT and NTT^{-1}. Kyber, Dilithium, and R.EMBLEM perform NTT with NTT_CT and NTT^{-1} with NTT_GS.

Sensitivity analysis. SoC components share the same system bus width and cache line size; although these two are sweepable parameters, we fixed them at 8 and 64 bytes since they are the typical sizes of CPU and system bus. The clock frequency of the host processor and system bus are fixed at 2GHz and 1GHz, respectively. The frequency of the accelerator can be set to 100 MHz, 200 MHz, or 500 MHz.

Figure 3.8-3.11 shows the effect of number of cache ports (Figure 3.8), software pipelining (Figure 3.9), number of parallel lanes (Figure 3.10), and accelerator's cache size (Figure 3.11) on the power and latency of an accelerator in a scheme. *Loop pipelining* overlaps execution of instructions over different loop iterations by exploiting the loop level parallelism, and can be beneficial to improve the performance but increases the hardware complexity and the energy dissipation. Software pipelining is more effective if it is being used with loop unrolling. *Loop unrolling* determines the number of parallel lanes that are created inside the accelerator for each loop. Loop unrolling improves the performance by alleviating the branch penalty and memory latency with an increase in the code size and the hardware area. Memory reduction latency needs parallel access to memory banks from the lanes, which can be achieved by increasing the number of *cache ports*; this also incurs high power consumption and area overhead.

The DSE trends for each counterpart in a scheme are similar; hence we only present the results for the Server and the Signer Also, due to the resemblance of the DSE trends, e.g. ,

Kyber_Server: NTT_CT and *Dilithium_Sign: NTT_CT*, we just present `DSE` results of `NTT_CT` for Kyber and `NTT_GS` for Dilithium.

Selected design points. We carried out our selection focusing firstly on energy consumption, then the area occupation, and finally, the speed of each accelerator. Table 3.2 provides the improvement of the latency, energy, and EDP of the schemes when only one kernel is accelerated, and the rest of the scheme is executed on the CPU. After selecting the desired design points for each accelerator, they are coupled to the bus and communicate via the coherent memory hierarchy (Figure 4.2). Figure 4.8 summarizes speedup and EDP improvements of the schemes when all the calls inside of the schemes to `Keccak`, `NTT_CT`, and `NTT_GS` are offloaded to their corresponding optimized accelerators. The frequency of the accelerators 100 MHz, 200 MHz, and 500 MHz.

Based on the results we obtained, we can derive several general guidelines about accelerating lattice-based cryptography. Figure 3.9, suggests always to enable the loop pipelining since most of the points on the Pareto-frontier curves are the blue points (pipelining on). The iterative nature of the selected kernels is the main reason for the efficacy of loop pipelining. Considering energy as the main concern, independent of the scheme, after 64 times of loop unrolling the performance gain is saturated for `NTT_CT` and `NTT_GS` (Figure 3.10); saturation point for `Keccak` is 12 parallel lanes.

According to Figure 3.8, the highest speedup is achieved with 8 cache ports, considering the performance as the primary concern. However, considering the energy as the primary figure of merit as is our goal, cache port 4 is suggested because for all the kernels it achieves acceptable performance. Besides, the cache size of 4kB is proper for all the accelerators since with bigger cache sizes performance improvements cannot compensate for the energy penalty.

Another consideration is related to the size of the polynomial and the achieved acceleration.

Table 3.2: Latency and energy-delay product (EDP) improvement of the schemes (functional units power models are based on OpenPDK 45nm and SRAM model from CACTI 5.3); one accelerator (100MHz) at a time is attached to the host x86 CPU (2GHz) through the bus. To select a design point and attach the corresponding accelerator to the SoC, we focus firstly on energy consumption, then the area occupation and finally the speed of each accelerator.

Scheme	**Kernel**	**Cycle Count**		**Speedup (CPU+Accel)**	**Improvement (CPU+Accel)**		**Accelerator's area (mm^2)**
		CPU (Baseline)	**CPU+Accel**	**Speedup**	**Energy**	**EDP**	
NewHope1024cpa Server	**NTT_GS**	1283366	676297	1.89	1.36	2.6	1.42
	Keccak		1282400	1.01	1.06	1.07	1.64
NewHope1024cpa Client	**NTT_GS**	1353624	741725	1.82	1.35	2.46	1.43
	Keccak		1352716	1	1	1	1.64
Kyber768 Server	**NTT_CT**	2100227	1620437	1.29	1.11	1.44	0.42
	NTT_GS		1837164	1.14	1.19	1.54	1.43
	Keccak		2069412	1.1	1.05	1.07	1.64
Kyber768 Client	**NTT_CT**	1090235	843838	1.29	1.19	1.54	0.42
	NTT_GS		951567	1.15	1.13	1.29	1.43
	Keccak		1052613	1.03	1.13	1.17	1.64
Dilithium Sign	**NTT_CT**	2470672	2077058	1.18	1.07	1.45	1.36
	NTT_GS		1987848	1.24	1.09	1.28	1.46
	Keccak		2441352	1.01	1.03	1.25	1.64
Dilithium Verify	**NTT_CT**	1278898	1060998	1.20	1.20	1.28	1.36
	NTT_GS		1156888	1.10	1.16	1.35	1.46
	Keccak		1229706	1.04	1.20	1.05	1.64
R.EMBLEM Server	**NTT_CT**	4072534	2400767	1.7	1.5	2.5	1.36
	NTT_GS		3147634	1.3	1.3	1.7	1.42
R.EMBLEM Client	**NTT_CT**	1721429	1273543	1.34	1.26	1.7	1.36
	NTT_GS		1101269	1.56	1.43	2.24	1.42
AKCN_LWE Server	**NTT_CT**	1216360	933151	1.3	1.2	1.56	1.36
AKCN_LWE Client	**NTT_CT**	1087624	860891	1.26	1.1	1.33	1.36

The bigger the size of the polynomial, the more speedup we get. NewHope1024 achieves, on average, 20% more speedup than NewHope512 by using the same accelerators since polynomials in NewHope1024 is twice bigger than NewHope512, and hence, it gets more benefit from the software pipelining along with loop unrolling. In the case of `MLWE` schemes, Kyber and Dilithium, size of the polynomials are the same for all different version of the scheme; hence the speedup for different variants of `MLWE` schemes is similar.

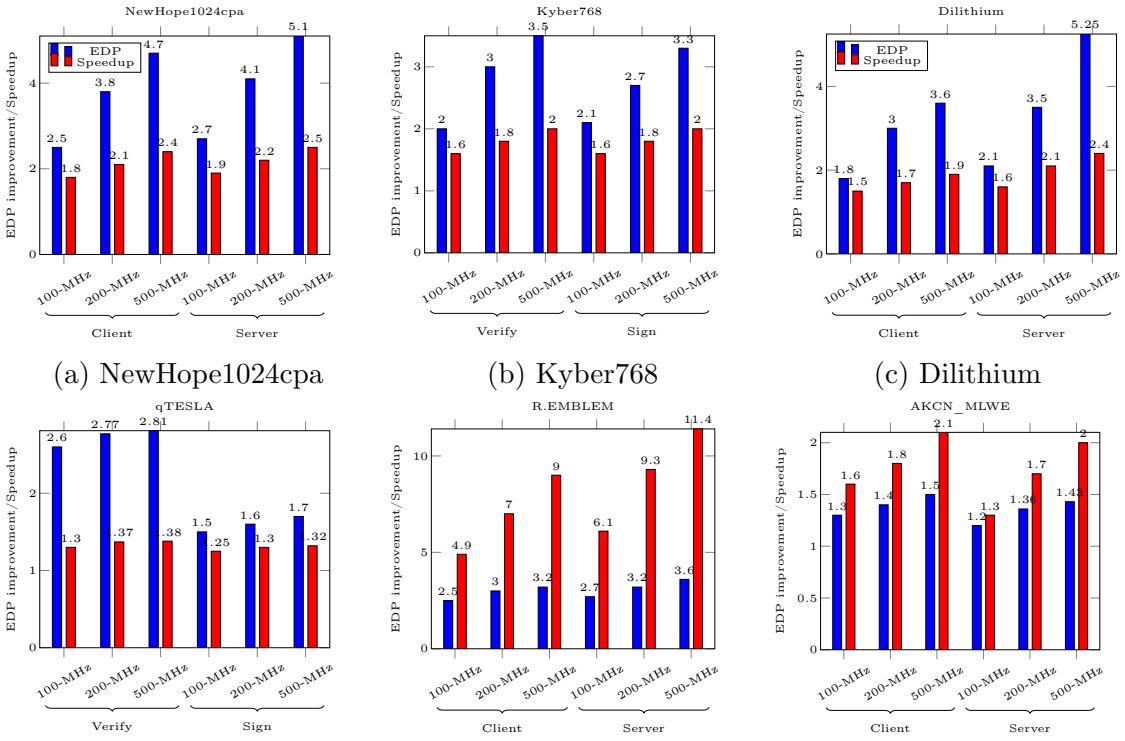
With the current fast CPU (2GHz) and energy-efficient accelerator (Figure 4.8) we suggest to allow the accelerators to operate at the maximum frequency, 500 MHz, specifically for `Keccak` since at 100 MHz (Table 3.2) it is not beneficial to use the accelerator. The power

consumption of the accelerators is 2 orders of magnitude less than that of the host CPU; consequently, faster accelerators result in better EDP improvement and higher speedup.

Comparing our acceleration with state of the art is challenging since the acceleration of these algorithms was not presented in literature before. We can, however, compare the achieved performance with results used with the same tool-chain in other domains.

Previous hardware implementations and accelerators. Hardware implementations for the analyzed algorithms are rather scarce, but proponents of R.EMBLEM designed and implemented in the Zynq-7000 board the algorithm EMBLEM.CPA (meaning without the ring structure) [204]. Their design only used 24 FFs and 48 LUTs, but this result can be misleading for evaluating the area of R.EMBLEM.CCA. The same board was used to implement NewHope for high performance. Four butterfly units were employed that use 6680 slices, 9412 FFs, 18756 LUTs, 8DSP, and 14 BRAMS; the implementation took 51.9 μs, 78.6 μs, 21.1 μs for the three phases of NewHope at a frequency of 131 MHz [138]. An existing co-processor for RSA/ECC was repurposed for (ideal) lattice-based cryptography. CCA-secure Kyber768 key generation was executed in 79.6 ms, encapsulation in 102.4 ms, and decapsulation in 132.7 ms [9]. Aside from the acceleration, this work confirmed that PQC is feasible on current smart card platforms.

Basu et al. [28] compared different hardware implementations of PQC algorithms. They used Xilinx HLS-based design space exploration to map C-code to FPGA and ASIC. Differing from us, they created an accelerator with the complete algorithms; instead, we create accelerators for the bottleneck functions. Therefore, we include in our methodology a profiling step to identify the most time-consuming functions, and hence, our accelerator implements kernels that can be used in different algorithms that result in smaller accelerator (even after a linear correction on the transistor length of the libraries). Additionally, we present the effect on energy, which is an overlooked metric in previous works.



(a) NewHope1024cpa (b) Kyber768 (c) Dilithium

(d) qTESLA (e) R.EMBLEM (f) AKCN_MLWE

Figure 3.12: Delay and EDP improvement of the schemes when all the accelerators (fastest design points) at different frequencies are simultaneously attached to the bus.

3.5 Conclusion

We present a design flow and demonstrate its suitability for accelerating *NewHope*, *Kyber*, *Dilithium*, *KCL* (Key Consensus from Lattice), and *R.EMBLEM* five submissions to the NIST standardization contest. Our experiments demonstrate the suitability of the approach for exploring the design space of hardware accelerator for lattice-based algorithms, allowing us to quickly identify hardware kernels capable of reducing the energy consumption up to 2.1x, EDP up to 5.2x and improve the speedup up to 2.5x. Thanks to the quick exploration and performance estimation, we discover compelling guidelines for manually designing more optimized hardware accelerators for lattice-based algorithms.

Chapter 4

Simulation-based DMA-assisted Polynomial Multiplier

4.1 Background

A lattice $L \subset \mathbb{R}^n$ is the set of all integer linear combinations of basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$. i.e., $L = \left\{ \sum a_i \mathbf{b}_i \ : \ a_i \in \mathbb{Z} \right\}$. LBC exploits the hardness of two problems: Short Integer Solution (SIS) and Learning With Errors (LWE) [190]. Cryptosystems based on the LWE problem, the most common one, have their foundation in the difficulty of finding the secret key sk, given (A, pk), where $pk = A * sk + e \bmod q$, given pk the public key, e an error vector with Gaussian distribution, and A a matrix of constants in $\mathbb{Z}_q^{r \times n}$ chosen randomly from a uniform distribution. Because LWE requires large keys (e.g., 11 KB for Frodo [42]), it could be impractical on the devices with limited on-chip memory. To overcome this limitation, Lyubashevsky et. al [151] introduced Ring-LWE (R-LWE), a derivation of LWE in which A is implicitly defined as a vector a in the ring $\mathcal{R} \equiv \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$. Because polynomial multiplication, the most frequently used kernel in RLWE schemes, is computationally inten-

sive, such multiplication is usually implemented by using the Number Theoretic Transform (NTT), which drops the time complexity of the polynomial multiplication from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot \log n)$ [164].

Polynomials $a = a(n-1) \cdot x^{n-1} + \ldots + a(0)$ and $b = b(n-1) \cdot x^{n-1} + \ldots + b(0)$ are transformed into their NTT representations $A = A(n-1) \cdot x^{n-1} + \ldots + A(0)$ and $B = B(n-1) \cdot x^{n-1} + \ldots + B(0)$, and their multiplication can be computed coefficient-wise multiplication as $C = \sum_{i=0}^{n-1} A(i) \cdot B(i) \cdot x^i$. The result $c = a * b$ is obtained after the computation of the inverse number theoretic transform (NTT^{-1}) of C. Two common algorithms for performing number theoretic transform are Cooley-Tukey (CT) [61], which produces the result in the bit-reverse order by receiving the input in the correct order, and Gentleman-Sande (GS) [100], which receives the input in the reverse order and produces the output in the correct order. We refer to Gentleman-Sande and Cooley-Tukey algorithms as `NTT_CT` and `NTT_GS`, respectively, in the rest of this paper. Employing `NTT_GS` to compute both NTT and NTT^{-1} involves bit-reverse calculation (NewHope); however, the bit-reverse step can be skipped by using `NTT_CT` for NTT and `NTT_GS` for NTT^{-1} (Kyber).

4.2 Design Space Exploration (DSE) Flow

Figure 4.1 shows the design method in a flow of three steps. The first step is profiling the implementations of the target lattice-based schemes. This process is needed to identify common bottlenecks and kernels to accelerate. Profiling is executed by running the protocol on a host arbitrary machine, e.g., x86 based system. Alternatively, the profiling can be executed in simulation, e.g., in the gem5 simulator [39]. In the former case, profiling results correspond to collecting performance counters from the execution. In the latter case, profiling means collecting simulation statistics. In either case, hot-spots in the protocol execution are identified. *We profile both the server and the client portion of the protocol separately to*

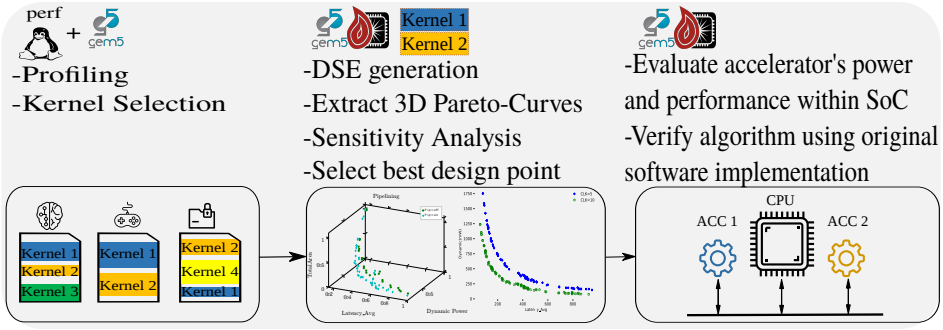


Figure 4.1: Design flow of the accelerator generation.

accommodate the specific needs of each end-point. The identified hot-spot functions are candidates for acceleration and are input in our design exploration flow. In the second step, we refer to Aladdin [206] to perform the design space exploration of the candidate functions, which takes a few seconds to generate a design point. Aladdin is a pre-RTL simulation tool whose performance, area, and power estimation for standard linear algebra kernels fall within a threshold of 10% with respect to actual HDL synthesis. The `DSE` proceeds by extracting the instruction-level parallelism in each kernel, and the mapping of such parallelism on a parameterizable set of finite constraints in the hardware, e.g., number of vector lanes, vector

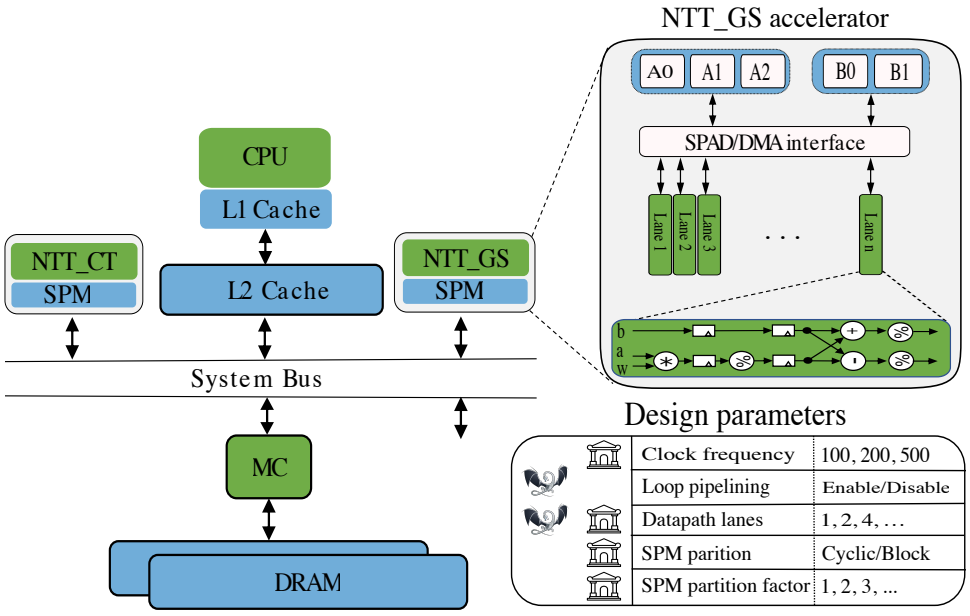


Figure 4.2: Architectural template for generated accelerators for (`NTT_CT`) and (`NTT_GS`) kernels. Configurable micro-architectural and compiler parameters are shown in the first column of the table; the next column is the searched spaced in this work.

lane pipeline depth, ports to memory, and communication to memory (with cache or via streaming). The design points that are not pruned during the multi-objective optimization of 3D-Pareto fronts will be evaluated in the whole system on the chip. In the third step, we use the integration of gem5 and Aladdin [207], to evaluate the accelerator within the whole SoC. Thanks to the accuracy guaranteed by gem5 and Aladdin, the last step allows us to identify a restricted set of designs that quickly reach our design goals.

Based on Figure 4.2, accelerators are loosely coupled with the main processor in an SoC. Each accelerator has its local memory and operates in the slave mode attached to the SoC. The accelerators are kernel-level accelerators, where the kernel to accelerate represents the bottleneck functions that are common across different families of lattice-based cryptography schemes. For instance, any call to `NTT_GS` and `NTT_CT` are transferred to their corresponding accelerator.

4.3 Algorithm Analysis & Profiling

We consider NewHope [179] and Kyber [16], two key encapsulation mechanisms (KEM) for acceleration. A KEM is defined as using an asymmetric cryptographic algorithm to encapsulate a symmetric key [168].

Kyber. Kyber is a KEM based on the LWE problem in module lattices [142]. Kyber uses `NTT_CT` and `NTT_GS` to compute forward and reverse NTT, respectively. Kyber is presented in three versions: Kyber512, Kyber768, and Kyber1024 [16]. Due to their similarity, we only analyze Kyber768 in this paper. The profile of Kyber768, referred to as Kyber in the rest of the paper, is shown in Figure 4.3. We introduce Kyber_Flex as a variant of Kyber in which both forward and reverse NTT are calculated using `NTT_GS`. This transformation involves adding the bit-reversal operation which is cheap in hardware. Kyber and Kyber_Flex work

on polynomials of degree 256.

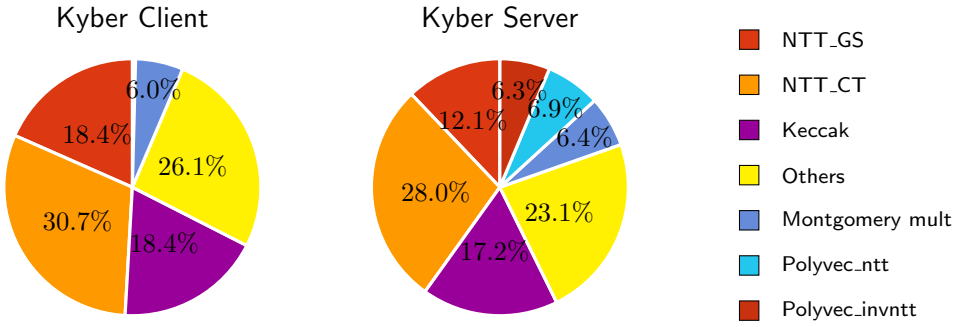


Figure 4.3: Profiling results for Kyber based on the user-level CPU cycles.

NewHope. NewHope cryptosystem is a suite of KEMs whose security is based on the hardness of the RLWE problem. The current submission of NewHope to the NIST competition presents two parameter sets, NewHope512cpa/cca and NewHope1024cpa/cca.NewHope1024cpa employs `NTT_GS` for both forward and inverse NTT. NewHope512cpa and NewHope1024cpa work on the polynomials with the degree of 512 and 1024, respectively. Figure 4.4 shows the profile of NewHope1024cpa in the client and server-side.

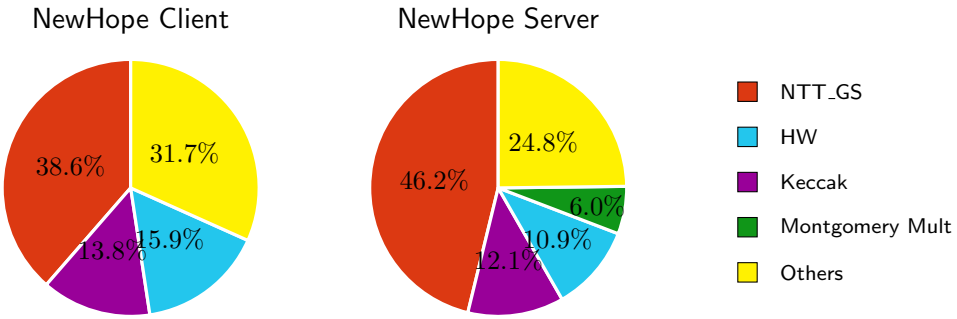


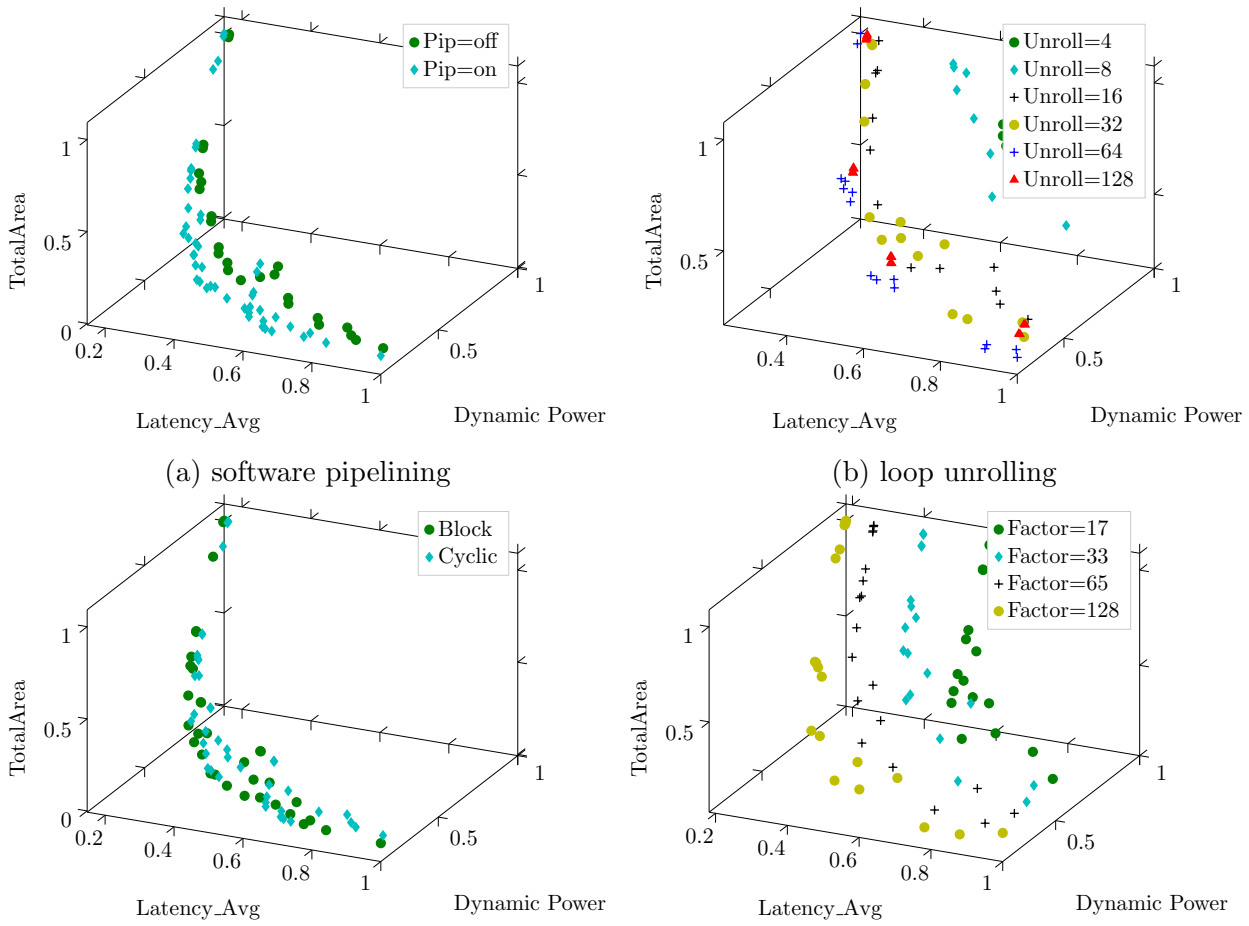
Figure 4.4: Profiling results for NewHope based on the user-level CPU cycles.

4.4 Design Space Exploration

The first step in design space exploration (DSE) is to identify compute-intensive kernels, and then search the space of the design and select the desired design points and evaluate them within an SoC. We use the original software implementation of the accelerated algorithms as a baseline for our evaluation. Based on the profiling results (Figures 4.3-4.4) we select `NTT_CT` and `NTT_GS` for acceleration. `NTT_CT` and `NTT_GS` are two flexible accelerators that can be employed inside any scheme with arbitrary security parameters to perform the number theoretic transform. Figure 4.2 shows the architectural template of the designed accelerators, which are composed of a set of lanes that implement the primitives of the kernel, local SRAM, one level of cache and the respective controller. Our tool allows analyzing the accelerators without digging into the details of the hardware implementations, concentrating on the accelerator functionality. To perform NTT and NTT^{-1} in NewHope, we use `NTT_GS`. Kyber performs NTT with `NTT_CT` and NTT^{-1} with `NTT_GS`.

Sensitivity analysis. SoC components share the same system bus width the size of 8 bytes. The clock frequency of the host processor and system bus are fixed at 2GHz and 1GHz, respectively. The frequency of the accelerator can be set to 100 MHz, 200 MHz, or 500 MHz. Design parameters of the accelerators which could have a significant impact on our figures of merit have been explored in more depth. Figure 4.5 shows the effect of software pipelining (Figure 4.5a), number of parallel lanes (Figure 4.5b), partition type/factor of the coefficient and twiddle factors (Figure 4.5c-4.5d) on the area, power, and latency of `NTT_GS` accelerator with polynomial degree of 256. Due to the similarity of results, we omit the results for polynomial degree of 512 and 1024 as well as the results for `NTT_CT` accelerator.

Selected design points. We perform a multi-objective optimization of 3D-Pareto fronts to eliminate non-optimal points from the selection process. Two steps of Pareto-frontier optimization are done; firstly, non-optimal points considering latency and power are pruned



(a) software pipelining

(b) loop unrolling

(c) SRAM partition type

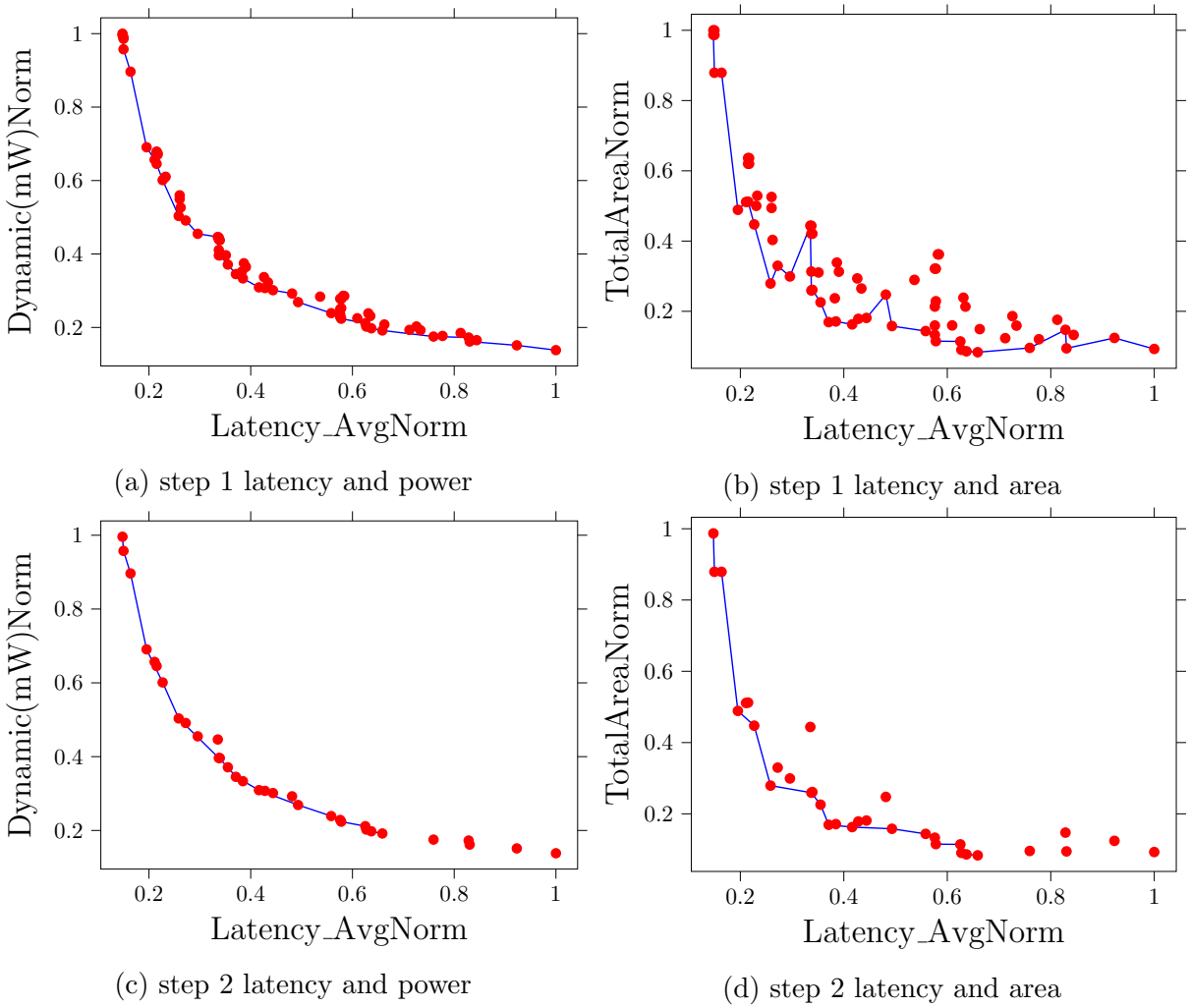
(d) SRAM partition factor

Figure 4.5: The normalized sensitivity of the `NTT_GS` performance, area, and power to different design parameters; software pipelining is either *off* or *on*; SRAM partition type is either *cyclic* or *block*.

(Figure 4.6a and Figure 4.6b); thereafter, we consider latency and area to prune the remaining non-optimal points (Figure 4.6c and Figure 4.6d).

After elimination of non-optimal points, the next step is to select the desired design point. We carried out our selection focusing firstly on energy consumption, then the speed of each accelerator. After selecting the desired design points for each accelerator, they are coupled to the bus and communicate via the coherent memory hierarchy (Figure 4.2). We generate the accelerators for `NTT_GS` and `NTT_CT` regarding the polynomial degree of 256 as in Kyber and Kyber_Flex to satisfy the programmability need. However, NewHope512cpa and

NewHope1024cpa have higher polynomial degrees and faster accelerators can be generated for them. We compare custom accelerators for `NTT_GS` to their programmable alternatives in Figure 4.7. We suggest to use the programmable accelerator for `NTT_GS`: we can achieve programmability as well as area and power reduction of the accelerator by employing a single smaller accelerator, at the cost of only 13% and 2% drop in the latency of the accelerators and the schemes. Table 4.1 provides the improvement of the latency, energy, and EDP of the schemes when the invocations to `NTT_CT` and `NTT_GS` are transferred to their corresponding programmable accelerators at 100 MHz. Kyber executes forward NTT and inverse NTT with `NTT_CT` and `NTT_GS`, respectively, while other schemes, including Kyber_Flex, run both



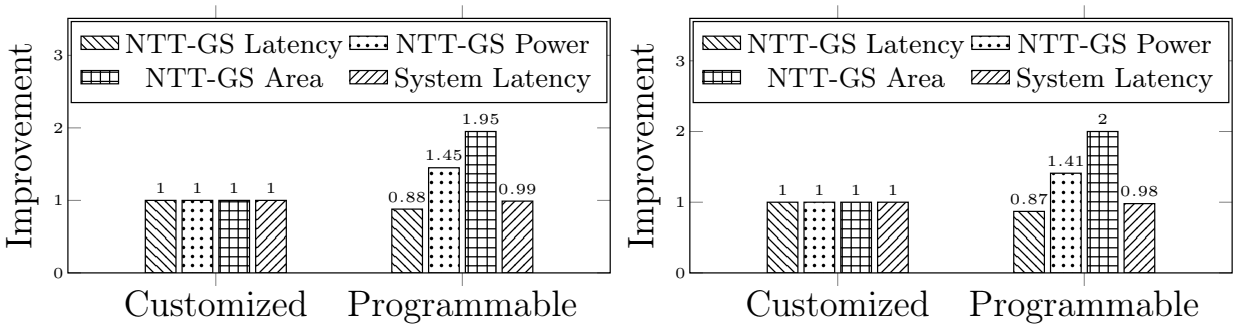
(a) step 1 latency and power

(b) step 1 latency and area

(c) step 2 latency and power

(d) step 2 latency and area

Figure 4.6: Multi-objective optimization of 3D-Pareto fronts. The points that do not lie on the blue curve are the non-optimal points .



(a) NewHope512 (b) NewHope1024

Figure 4.7: Comparison of programmable and customized `NTT_GS` at 100MHz for NewHope512cpa and NewHope1024cpa. Numbers are normalized to the customized accelerator.

forward and inverse NTT on `NTT_GS`. With Kyber_Flex we improve performance by 20% less than Kyber. Consequently, we suggest creating only one accelerator (i.e., `NTT_GS`) for all the ideal lattice-based schemes. As can be seen in Figure 4.8, an increase in the frequency of the accelerators has negligible improvements in performance and EDP. Figure 4.8 summarizes the effect of the accelerators' clock frequency on the speedup and EDP improvements of the schemes. Due to the negligible performance improvements, we suggest to employ the slowest accelerator (100 MHz) for the resource-constrained devices.

Table 4.1: Performance, energy, and EDP improvement of schemes with different polynomial degrees when the fastest `NTT_GS` and `NTT_CT` accelerators (100 MHz) are simultaneously connected to the CPU (2GHz) via DMA (1 GHz).

Scheme	**Degree**	**NTT**	**NTT^{-1}**	**Perf**	**Energy**	**EDP**
Kyber Client	256	NTT_CT	NTT_GS	1.71x	1.45x	2.49x
Kyber Server	256	NTT_CT	NTT_GS	1.74x	1.50x	2.62x
Kyber_Flex Client	256	NTT_GS	NTT_GS	1.53x	1.33x	2.05x
Kyber_Flex Server	256	NTT_GS	NTT_GS	1.57x	1.35x	2.10x
NewHope512 Client	512	NTT_GS	NTT_GS	1.67x	1.48x	2.48x
NewHope512 Server	512	NTT_GS	NTT_GS	1.68x	1.34x	2.26x
NewHope1024 Client	1024	NTT_GS	NTT_GS	1.81x	1.51x	2.74x
NewHope1024 Server	1024	NTT_GS	NTT_GS	1.88x	1.62x	3.06x

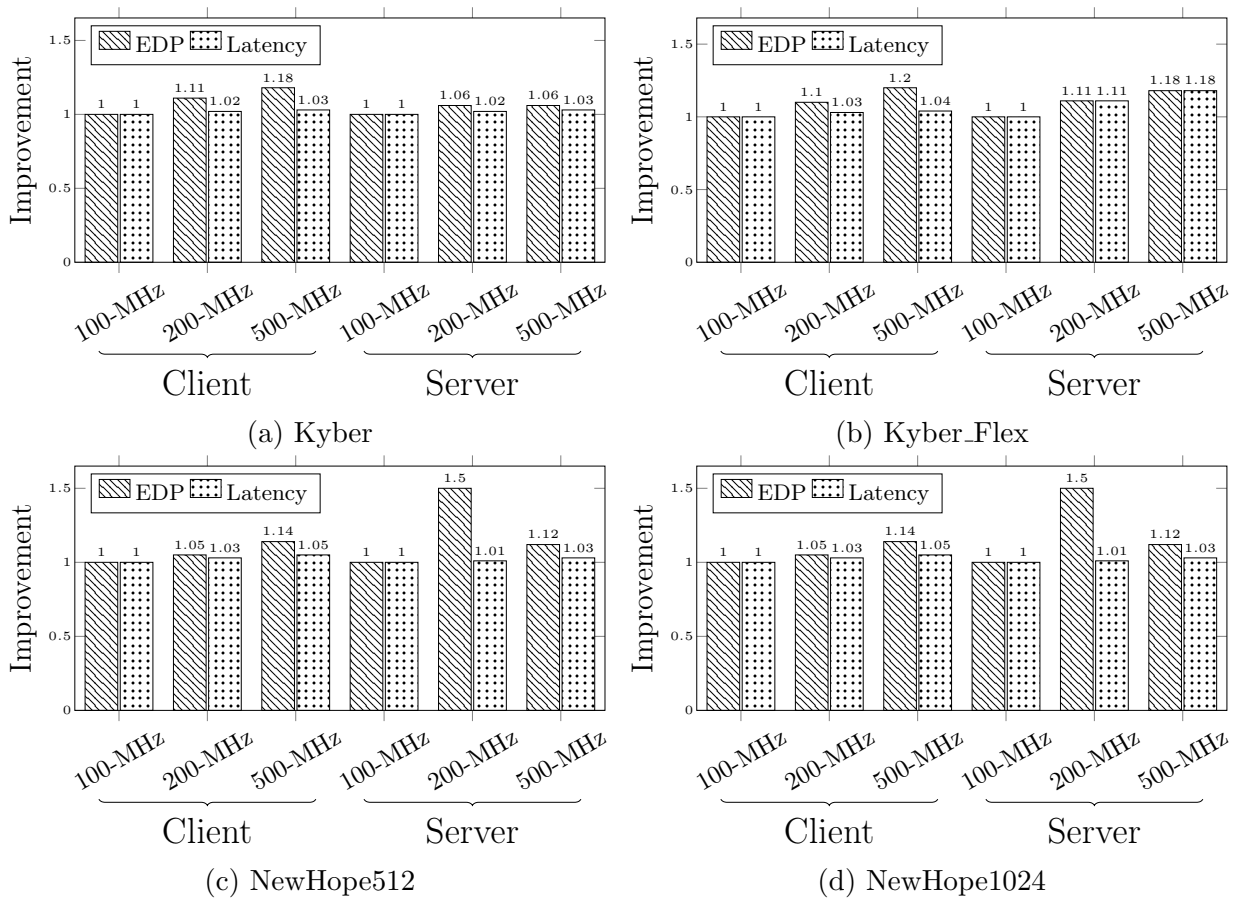


Figure 4.8: Normalized latency and EDP improvement of the schemes when the accelerators at different frequencies are simultaneously attached to the bus. Numbers are normalized to the execution case of 100MHz.

4.5 Conclusion

At this stage, it would be impractical to select a single scheme for hardware adoption due to unpredictable changes in both protocols and schemes. On the one hand, software-only implementations cannot satisfy the performance and energy requirements of the target platform; on the other hand, custom hardware implementations do not allow changes or upgradeability. To satisfy crypto-agility and efficiency challenges, we presented for the first time, a design flow for flexible DMA-based accelerators of hardware kernels, and demonstrated its suitability to accelerate ideal lattice-based schemes, including *NewHope* and *Kyber*. The presented approach provides the opportunity to quickly identify micro-kernels and generate a corresponding hardware accelerator that reduces energy consumption by 1.9x, EDP up to

3.9x and improve the performance up to 2x.

Chapter 5

Exploring Energy Efficient Quantum-resistant Signal Processing Using Array Processors

5.1 Background and Related Work

A lattice $L \subset \mathbb{R}^n$ is the set of all integer linear combinations of basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$. i.e., $L = \left\{ \sum a_i \mathbf{b}_i \ : \ a_i \in \mathbb{Z} \right\}$. LBC exploits the hardness of two problems: Short Integer Solution (SIS) and Learning With Errors (LWE) [190]. Cryptosystems based on the LWE problem, the most common one, have their foundation in the difficulty of finding the secret key sk, given (A, pk), where $pk = A * sk + e \bmod q$, given the public key pk, an error vector e with Gaussian distribution, and a matrix A of constants in $\mathbb{Z}_q^{r \times n}$ chosen randomly from a uniform distribution. Because LWE requires large keys (e.g., 11 KB for Frodo [42]), it can be impractical on devices with limited on-chip memory. To overcome this limitation, Lyubashevsky et. al [151] introduced Ring-LWE (RLWE), a derivation of LWE in which A

is implicitly defined as a vector a in the ring $\mathcal{R} \equiv \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. Arithmetic operations for a `Ring-LWE`-based scheme are performed over a ring of polynomials. Let n, a power of two, and p be the degree of the lattice and a prime number ($p = 1 \mod 2n$), 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 degree less than n in Z_p, that defines $R_p = Z_p/[x^{n+1}]$ in which coefficients of polynomials are in [0,p). Degrees of the polynomials in RLWE-based schemes vary between 256 [16] and 1024 [179].

In the following, we describe two common methods to compute the polynomial multiplication.

5.1.1 Convolution-based multiplier

The easiest way to multiply two polynomials is to use the convolution (Schoolbook [135]) with the time complexity of $O(n^2)$ as shown in Algorithm 4. A convolution-based multiplier can be seen as a discrete feed-forward finite impulse response (FIR) over the polynomials in $\mathcal{R} \equiv \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. Due to the inefficiency of the convolution-based multiplier, there is no notable work that uses it for the acceleration; however, by using its systolic architecture, we can achieve considerable performance gains.

5.1.2 NTT-based multiplier

Polynomial multiplier is usually implemented by using the Number Theoretic Transform (NTT), which drops the time complexity of the polynomial multiplier from $\mathcal{O}(n^2)$ to $\mathcal{O}(n \cdot \log n)$. Polynomials $a = a(n-1) \cdot x^{n-1} + \ldots + a(0)$ and $b = b(n-1) \cdot x^{n-1} + \ldots + b(0)$ are transformed into their NTT representations $A = A(n-1) \cdot x^{n-1} + \ldots + A(0)$ and $B = B(n-1) \cdot x^{n-1} + \ldots + B(0)$, and their multiplication can be computed coefficient-wise as $C = \sum_{i=0}^{n-1} A(i) \cdot B(i) \cdot x^i$. The result $c = a * b$ is obtained after the computation

Algorithm 4: Convolution (Schoolbook)-based Polynomial Multiplier

Function `POLY_CONV(`A, B`):`

`/* Initialization: Let` $a = \{a_0, a_1, a_2, ..., a_{n-1}\}$ `and` $b = \{b_0, b_1, b_2, ..., b_{n-1}\}$ $\in \mathbb{Z}_q[x]/<f(x)>$ `be two polynomials with the length of` n`, where` $f(x) = x^n + 1$ `is an irreducible polynomial with` n `a power of 2, and` $q \equiv 1 \, mod \, 2n$ `is a large prime number. */`

$c \leftarrow 0$
for $i = 0$ *to* $n - 1$ **do**
  for $j = 0$ *to* $j - 1$ **do**
    $sign \leftarrow (-1)^{\lfloor (i+j)/n \rfloor}$
    $index \leftarrow (i + j) \, mod \, n$
    $coeff \leftarrow a_i b_i \, mod \, q$
    $c_{index} \leftarrow integer(c_{index} + sign * coeff) \, mod \, q$
  end
end
return c

of the inverse number theoretic transform (NTT^{-1}) of C. Algorithm 5 describes the NTT-based polynomial multiplier. Figure 5.1 illustrates basic blocks of the NTT-based polynomial multiplier. One standard method to perform the number theoretic transform is Cooley-Tukey (CT) [61], which produces the result in the bit-reverse order by receiving the input in the correct order; the other method is Gentleman-Sande (GS) [100], which receives the input in the reverse order and produces the output in the correct order. Similar to [223], we use the Gentleman-Sande method to compute both NTT and NTT^{-1}, which needs bit-reverse calculation. As previously mentioned, NTT designs are known as *parallel NTT* due to the parallel use of butterfly processing elements. *Pipeline FFT* processor [210] is a high-throughput design of FFT that utilizes single-path delay feedback. Authors in [191] adopt *Pipeline FFT* to design a systolic array polynomial multiplier to develop a high throughput RLWE cryptoprocessor. In this work, we focus on the parallel NTT designs and leave the *Pipeline FFT* for future work.

Algorithm 5: NTT-based Polynomial Multiplier

Function `POLY_NTT(`A, B`):`

/* `Initialization: Let` $a = \{a_0, a_1, a_2, ..., a_{n-1}\}$ `and` $b = \{b_0, b_1, b_2, ..., b_{n-1}\}$ $\in \mathbb{Z}_q[x]/<f(x)>$ `be two polynomials with length of` n`, where` $f(x) = x^n + 1$ `is an irreducible polynomial with` n `a power of 2, and` $q \equiv 1 mod 2n$ `is a large prime number.` w `is the` n`-th root of unity and` ϕ `is the` $2n$`-th root of unity (`$\phi^2 = w$ `mod` q`);` w^{-1} `and` ϕ^{-1} `are the inverse of` w `mod`q `and` ϕ `mod`q`, respectively.` */

/* `Precompute:` $\{w^i, w^{-i}, \phi^i, \phi^{-i}\}$ `for` $i \in [0, n-1]$ */

$bitrev(A)$

$bitrev(B)$

for $i = 0$ *to* $n - 1$ **do**

  $\bar{a_i} \leftarrow a_i \phi^i$

  $\bar{b_i} \leftarrow b_i \phi^i$

end

$\bar{A} \leftarrow NTT\_GS(\bar{a}, w)$

$\bar{B} \leftarrow NTT\_GS(\bar{b}, w)$

$\bar{C} = \bar{A}.\bar{B}$

$bitrev(\bar{C})$

$\bar{c} \leftarrow NTT\_GS(\bar{C}, w^{-1})$

for $i = 0$ *to* $n - 1$ **do**

  $c_i \leftarrow \bar{c_i} \phi^{-i}$

end

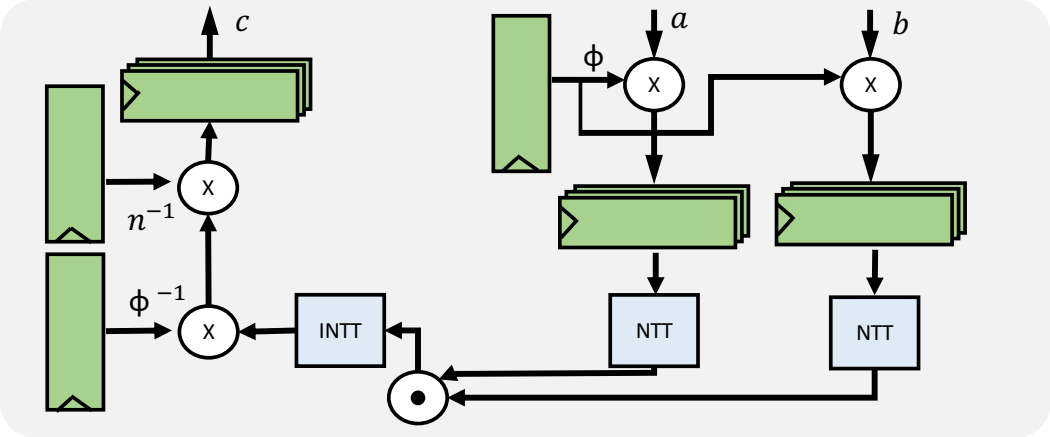
return C

5.1.3 Previous works

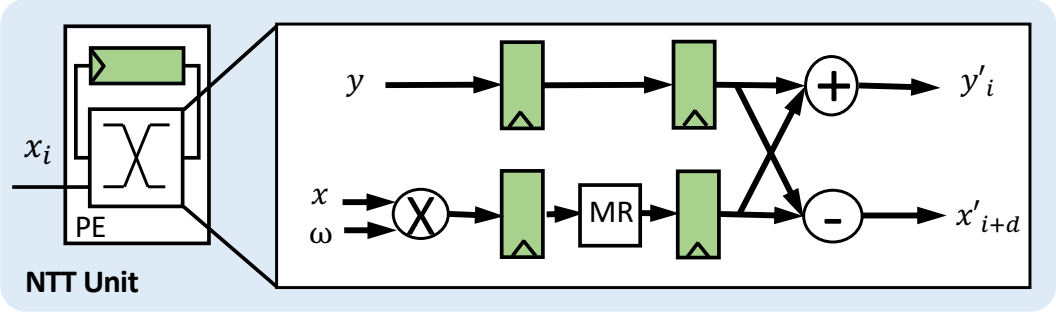
Previous efforts on the acceleration of the NTT mostly focus on the area and performance of the of the polynomial multiplier for LBC and leave the energy unexplored [184] [62]. Some efforts have evaluated the energy as well as the area and performance of NTT accelerators [166, 24, 163]. The only notable work that uses systolic arrays to accelerate polynomial multiplier reports only performance and area metrics for $n = 256$ and $n = 512$; however, we report energy as the primary goal as well as the performance and area for $n \in$ (128, 256, 512, 1024).

5.2 Systolic Array polynomial multiplier

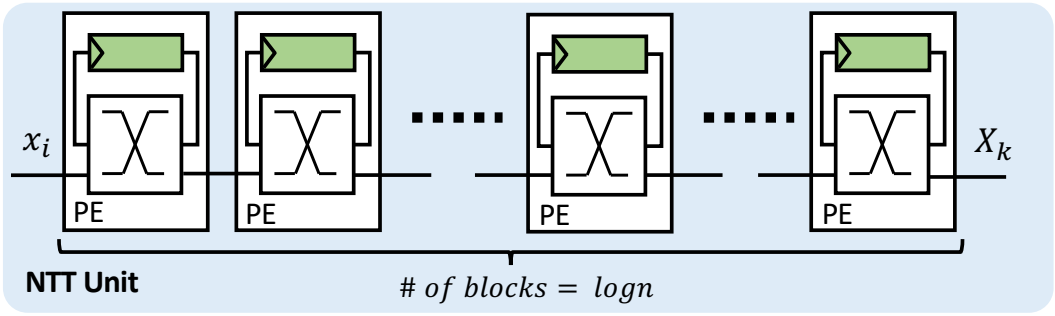
We use the concept of systolic array architecture to design polynomial multipliers.



(a) High level diagram for polynomial multiplication using NTT units



(b) NTT unit in the sequential NTT-based (`Seq_NTT`) polynomial multiplier



(c) NTT unit in the systolic array NTT-based (`SA_NTT`) polynomial multiplier

Figure 5.1: (a) Polynomial multiplication using NTT units (b) NTT unit based on only one PE which processes inputs sequentially in $(n/2)logn$ iterations. In order to perform polynomial multiplication NTT unit is executed three times (c) NTT-based systolic array polynomial multiplier encompass $logn$ PE blocks each performs butterfly operation in $n/2$ iterations.

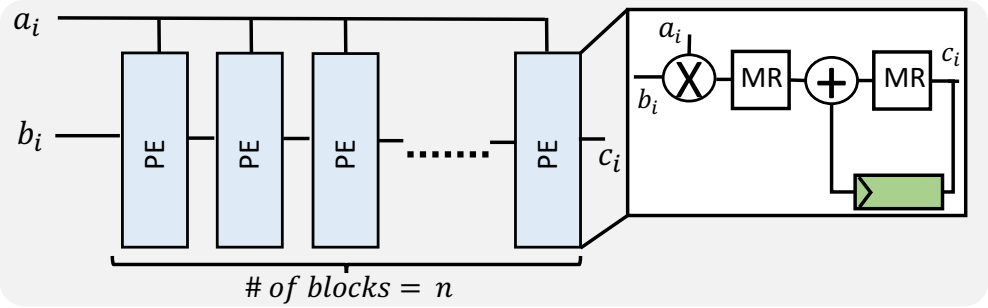


Figure 5.2: Convolution-based polynomial multiplier using the systolic array. Modular reduction (MR) is performed after each multiplication and addition.

5.2.1 Systolic arrays

Today's systems are intensely designed to move data for computation. Data movement is highly expensive in terms of energy consumption and latency compared to computation. Consequently, the movement of data is the critical bottleneck in computing systems as applications become more data-intensive. To address the bottleneck, we need an alternative architecture – such as a systolic array – to process data with less data movement. A systolic array consists of a set processing elements (PE), each capable of performing simple operations. Each PE is connected to its nearest PEs and performs operations on data. Data flows from the memory cells and passes through PEs before returning to the memory cells. Systolic architecture relieves the repeated memory access problem for general-purpose computing systems, which in turn helps to reduce latency.

We design and implement two systolic array polynomial multipliers including NTT-based and convolution-based systolic array and compare them to the sequential NTT-based multiplier.

5.2.2 Implementation of systolic array polynomial multiplier

5.2.2.1 NTT-based polynomial multiplier

The conventional hardware implementation for the NTT-based polynomial multiplier uses a processing element with only one butterfly block sequentially (Figure 5.1-a). We use sequential NTT-based multiplier (`Seq_NTT`) as the baseline in our experiments; `Seq_NTT`, our slowest design, provides 3x speedup to compute forward and inverse number theoretic transforms compared to the implementation of [223] on a low-cost Artix7 Xilinx FPGA.

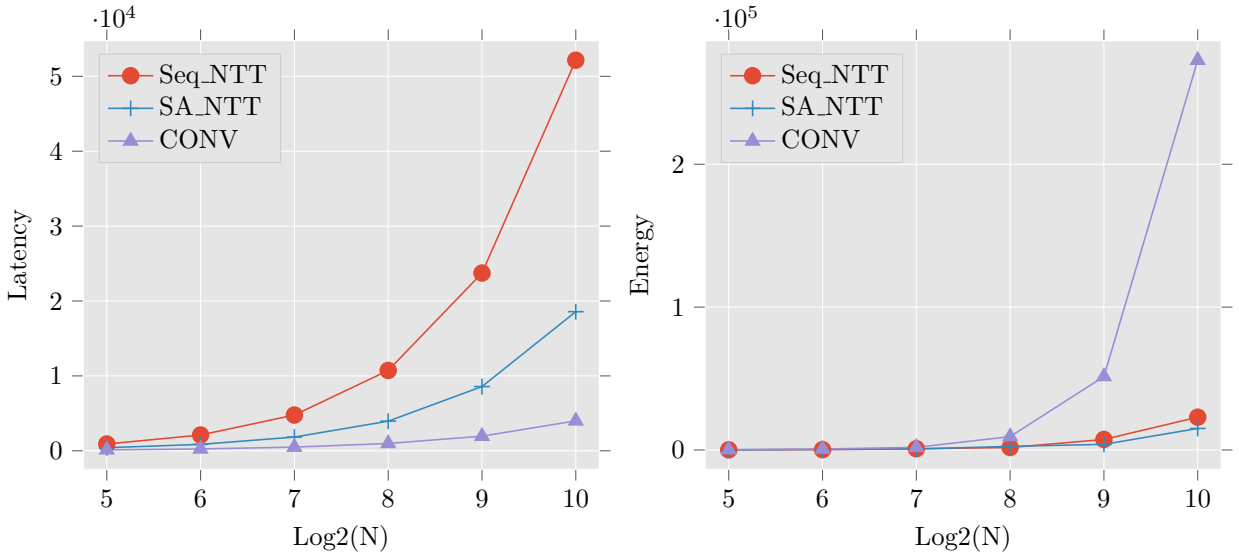
The NTT-based polynomial multiplier can be implemented using a systolic array (`SA_NTT`for the rest of the paper). Figure 5.1-b shows a *logn* array of processing elements, each executes $n/2$ butterfly operations on all coefficients of the input polynomial. In other words, we cascade all *logn* stages of the NTT-method and connect them using FIFO buffers. According to the Gentleman-Sande method, extracting parallelism between stages of the NTT is non-trivial; we can improve the performance of NTT by fusing multiple stages through the dataflow optimization of a high-level synthesis (HLS) tool.

5.2.2.2 Convolution-based polynomial multiplier

The time complexity of the convolution-based polynomial multiplier can be decreased to O(n) if we adjust the systolic architecture to use n-cascaded multiply-accumulators (MACs). As shown in Figure 5.2, each processing element in the convolution-based polynomial multiplier (`CONV`) performs modular reduction (MR) after each multiplication and addition. The significant performance improvement comes at the large area and energy overhead of performing n MACs. We use our optimized versions of Montgomery [156] and Barrett [27] reduction after each multiplication and addition, respectively, using only shifts and additions.

Table 5.1: High-level synthesis results on Zynq UltraScale++ for three different designs of polynomial multipliers. To multiply two polynomials using NTT-based multipliers, we need to execute NTT block three times, and each includes bit-reversal, forward NTT, and point-wised multiplication of the input vector with the precomputed twiddle factors. Although the degree of the polynomials in most of the RLWE-based schemes ranges from 256 to 1024, we also provide the results for other degrees to show the trends.

Design	N	Cycles	Latency (us)	Energy (uJ)	BRAM	CLB	DSP	FF	LUT	Freq. (MHz)
Seq_NTT	64	2112	7.47	0.16	0	221	10	1120	937	282.48
	128	4776	16.79	0.62	1	364	10	2085	1,520	284.33
	256	10728	36.37	1.27	2	667	10	4281	2999	294.89
	512	23736	83.93	4.36	2	1220	10	8374	5278	282.80
	1024	52152	169.28	13.37	2	2551	10	17091	10888	308.07
SA_NTT	64	897	4.86	0.49	14	366	38	1577	1760	184.39
	128	1878	10.19	0.95	17	426	43	1906	2063	184.16
	256	4008	21.56	2.15	24	547	48	2230	2372	185.83
	512	8634	47.63	5.28	27	605	53	2610	2772	181.25
	1024	18636	101.84	12.52	29	732	58	3007	3140	182.98
CONV	64	271	1.31	0.85	1	2226	322	8746	10223	205.88
	128	533	2.65	3.39	2	4398	642	19098	21501	200.68
	256	1058	5.32	8.33	2	8580	1282	38108	47332	198.57
	512	2107	10.75	64.03	2	19301	2562	135593	161924	195.84
	1024	3720	17.40	257.23	2	49714	4282	310247	322866	195.22



(a) Latency (b) Energy

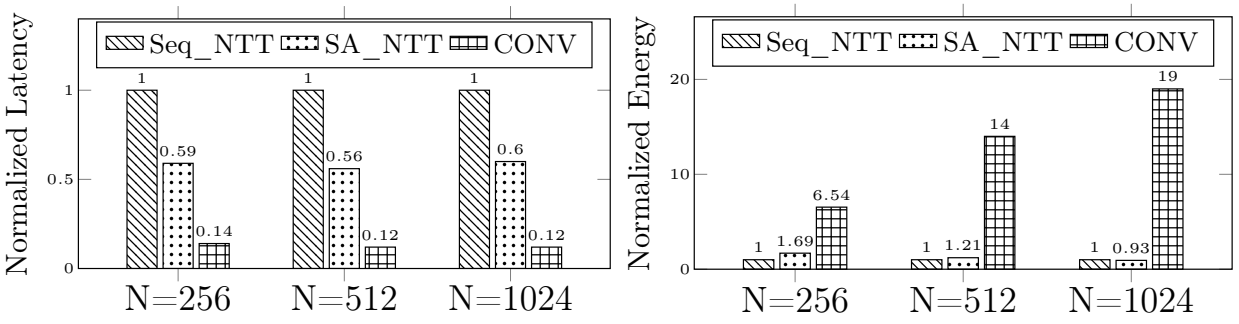
Figure 5.3: Increase in the latency and energy of polynomial multiplier by the increase in the size of the polynomials.

5.3 Evaluation

To perform the synthesis, we use Vivado high-level synthesis (HLS) 2018.2 and select Artix7 and Zynq UltraScale+ as the target FPGA devices from resource-constrained internet-of-things (IoT) settings and performance-driven real-time signal processing scenarios, respectively. We extract the numbers of reported resources (BRAM, CLB, DSP, FF, and LUT), maximum achieved frequency, and energy from the post-implementation process of the HLS tool. The latency of the polynomial multiplier is the number of execution cycles at the maximum achieved frequency.

For the polynomial-size of N=1024, `Seq_NTT` achieves 3x speedup to compute forward and inverse number theoretic transform compared to the implementation of [223] on a low-cost Artix7 FPGA. Table 6.2 shows the synthesis results of `Seq_NTT` as the smallest design along with the two systolic array polynomial multipliers, `SA_NTT` (NTT-based) and `CONV` (convolution-based on Zynq UltraScale++. `Seq_NTT` achieves the highest maximum frequency than `SA_NTT` and `CONV` because of its sequential architecture.

According to Figure 5.3, with the increase in the degree of the polynomial from 512 to 1024, the rise in the energy consumption of `CONV` is exponential due to the increase in the number of resources required to satisfy the timing constraints.



(a) Latency (b) Energy

Figure 5.4: Comparison of the different polynomial multiplier designs, normalized to NTT, for polynomials with degree 256, 512 and 1024.

Figure 5.4 shows the latency and energy consumption, extracted from Table 6.2, of the designs normalized to the `Seq_NTT`. For N=1024, e.g., NewHope scheme, `SA_NTT` reaches 3x speedup with 7% decrease in energy compared to the `Seq_NTT`; for N=256 and N=512, 1.7x improvement in latency is achieved with 69% and 21% increase in the energy, respectively. We suggest employing a systolic array NTT-based polynomial multiplier for resource-constrained devices to achieve plausible performance with low energy consumption to verify the digital signatures and/or perform encrypting and decryption. The convolution-based systolic array multiplier is suitable for high-performance servers that can tolerate higher energy consumption to generate 2x more signatures per second by CRYSTALS – Dilithium [80], a PQC digital signature scheme with a 7x speedup in the computing NTT. Forward and inverse NTT consumes around 65% of cycles in CRYSTALS – Dilithium to generate a signature.

5.4 Conclusion and future work

The advent of quantum computing threatens to render ineffective classical cryptographic schemes to secure signal processing applications. Emerging quantum-resistant cryptographic schemes show promise, but are hindered by the computational overhead of essential critical kernels such as a polynomial multiplication. This work explores, for the first time, the energy efficiency of array processors for implementing polynomial multipliers with degrees up to 1024.

We design and synthesize an NTT-based and a convolution-based systolic array polynomial multipliers and compare their performance, area, and energy with the sequential NTT-based counterpart. Our serial NTT-based design achieves 3x speedup on a low-cost Artix7 FPGA compare to the hardware implementation of NewHope-Simple.

On a Zynq UltraScale+ FPGA, NTT-based systolic array on average is 1.7x faster than

sequential NTT-based polynomial multiplier with less than 30% increase in the energy. The convolution-based systolic array on average is 7.5x faster than serial NTT-based polynomial multiplier with a 13.5x increase in the energy overhead; thus, convolution-based systolic array polynomial multipliers are suitable for high-performance servers that can tolerate higher energy consumption.

Our future work evaluates the energy efficiency of the *Pipeline FFT* processor to perform polynomial multiplication. Additionally, we make the systolic architecture of the `CONV`more area- and energy-efficient for securing quantum resistance of future signal processing applications.

Chapter 6

CryptoPIM: In-memory Acceleration for Lattice-based Cryptographic Hardware

6.1 Background and Related Work

A lattice $L \subset \mathbb{R}^n$ is defined as all the integer linear combinations of basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n \in \mathbb{R}^n$. The hardness of the lattice-based schemes are based on two mathematically hard problems: short integer solution (SIS) and, more commonly, learning With errors (LWE). Given the pair (A,pk) as a matrix of constants sampled uniformly at random in $\mathbb{Z}_q^n$ and the public key, the learning with error problem is defined as finding the secret key sk, where $pk = (A * sk + e) \bmod q$, and e is a small error vector that is sampled from a Gaussian distribution.

LWE-based schemes are impractical to be implemented on resource-constrained devices due to their large keys. At the same security level, Ring-LWE (RLWE) reduces the key size by

a factor of n, where n is the degree of the polynomial. In Ring-LWE (RLWE), a derivation of LWE in which A is implicitly defined as a vector a in the ring $\mathcal{R} \equiv \mathbb{Z}_q[x]/\langle x^n + 1\rangle$.

Algorithm 6: NTT-based Polynomial Multiplier

```
1  Function POLY_NTT(A, B):
       /* Initialization: w is the n-th root of unity and φ is the 2n-th
          root of unity (φ^2 = w mod q); w^-1 and φ^-1 are the inverse of w
          modq and φ modq, respectively.                                  */
       /* Precompute: {w^i, w^-i, φ^i, φ^-i} where w^i, w^-i are in reversed order,
          while φ^i, φ^-i are in normal order                            */
2      bitrev(A)
3      bitrev(B)
4      for i = 0 to n − 1 do
5          ā_i ← a_i φ^i
6          b̄_i ← b_i φ^i
7      end
8      Ā ← NTT_GS(ā, w)
9      B̄ ← NTT_GS(b̄, w)
10     C̄ = Ā.B̄
11     bitrev(C̄)
12     c̄ ← NTT_GS(C̄, w^-1)
13     for i = 0 to n − 1 do
14         c_i ← c̄_i φ^-i
15     end
16     return C
```

Arithmetic operations for a Ring-LWE-based scheme are performed over a Z_p, the ring of integers modulo p where n (degree of the polynomial) is a power of two, p is a large prime number, and x^{n+1} is an irreducible polynomial degree n. The quotient ring R_p includes polynomials with degree less than n in Z_p, that defines $R_p = Z_p/[x^{n+1}]$ in which coefficients of polynomials are in [0,p). Degrees of the polynomials in RLWE-based schemes vary between 256 [16] and 1024 [179] for public-key encryption and between 2k and 32k for homomorphic encryption [53].

Polynomial multiplication is commonly computed using the Number Theoretic Transform (NTT). Two polynomials ($a = a(n-1)\cdot x^{n-1} + \ldots + a(0)$ and $b = b(n-1)\cdot x^{n-1} + \ldots + b(0)$) are

transformed into the NTT domain ($A = A(n-1)\cdot x^{n-1} + \ldots + A(0)$ and $B = B(n-1)\cdot x^{n-1} + \ldots + B(0)$); multiplication of the two polynomial is computed as $C = \sum_{i=0}^{n-1} A(i) \cdot B(i) \cdot x^i$. The final result, $c = a*b$, is computed by applying the the inverse number theoretic transform (NTT^{-1}) on C. A common method to perform the number theoretic transform is Gentleman-Sande (GS) [100], which receives the input in the reverse order and produces the output in the normal order. Similar to [162], we employ the GS method to compute both forward and inverse number theoretic transforms. It involves changing the order of the coefficients in the vector representation (i.e., bit-reverse). Algorithm 5 describes the NTT-based polynomial multiplier using the GS method.

Algorithm 7: The Gentleman-Sande in-place NTT algorithm

```
1  Function ntt_gs(A,twiddle):
       /* To compute the NTT and NTT^-1, twiddle is set to {w^i} and {w^-i}
          for all i ∈ [0, n/2 − 1], respectively. Output is A in the
          frequency domain (bit-reversed order).                          */
2      for i=0 to log2n do
3          for idx=0 to n/2 do
4              st ← idx & ((1<<i)-1)
5              j ← ((idx & !(1<<i - 1) <<1)&(n-1) + st
6              j = j&(n − 1) + start
7              j' ← j + (1<<i)
8              k = j >> (i + 1)
9              W ← twiddle[j>>(i+1)]
10             T ← A[j]
11             A[j] ← (T + A[j'])mod q
12             A[j'] ← W*(T - A[j'])mod q
13         end
14     end
```

Numerous researches focus on implementation of classical cryptographic schemes for servers and constraint devices [224, 15]. In the scope of LBC schemes, researches on the acceleration of the NTT-based polynomial multiplications narrow the focus on the area and performance of the polynomial multiplier and leave the energy unexplored [184] [62].

Efforts have evaluated the energy as well as the area and performance of NTT accelerators

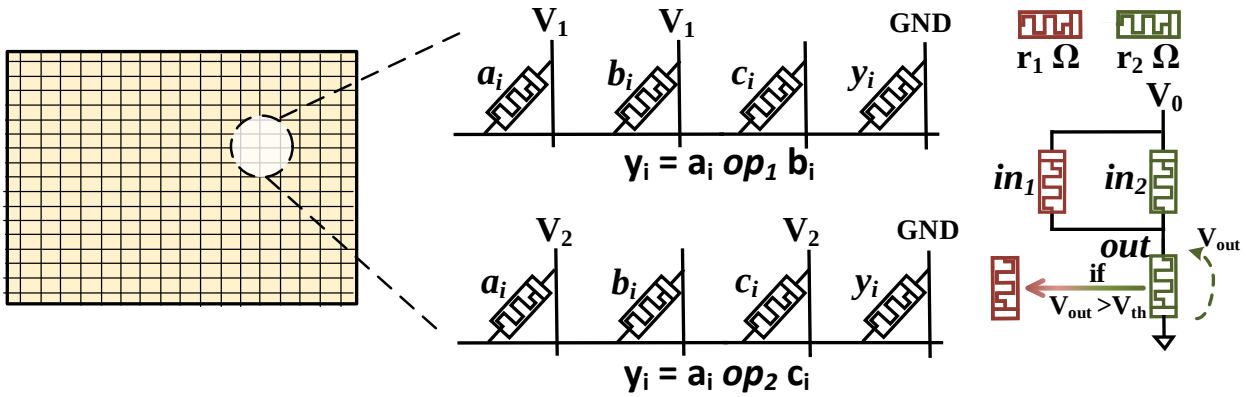


Figure 6.1: Digital processing in memory (PIM) overview.

[166][161][24][163]. We compare our results to the fastest FPGA implementation of the NTT-based multiplier in [162] for the vector sizes $n \in (256, 512, 1024)$ in terms of performance, energy, and throughput. Besides, we report the performance, energy, and throughput for vector sizes $n \in (256, 512, 1024, 2k, 4k, 8k, 16k, 32k)$.

While the works mentioned above try to accelerate NTT, they do not perform well for higher degree polynomials. Processing higher degree polynomial, even with NTT, involves a massive amount of computations. Hence, the application performance suffers due to (i) less than required on-chip memory and (ii) limited availability of complex cores. Prior work has proposed processing in memory (PIM), which is an architecture for doing in-situ computation [102, 94, 128].

Recent works in PIM enable highly efficient bitwise operations in memory [110, 94] and extend the operations to implement complex functions like floating-point arithmetic [128]. Besides, PIM has been shown to provide large dense memory and extensively parallel computing. Figure 6.1 shows high-level implementation of PIM based on bitwise computation in [110]. The memory block on the left is a crossbar of ReRAM (resistive RAM) cells, where each row of cells share a wordline, while those in the same column share a bitline. These cells have two possible states. The cells change state when the voltage across them crosses a device threshold. A memory cell is present at the intersection of a wordline and bitline. To implement bitwise functions, it applies a voltage V at the input bitlines and ground the

output bitline. The result of the computation is generated in the output cell. The operation performed is dependent on V. Also, the same operation can be executed in parallel over multiple rows of the block, enabling parallel vector-wide operations. Prior research designed PIM blocks for application acceleration, e.g., MapReduce based on 3D stacking [186], nearest neighbor search using computational logics beside DRAM [67], and parallel graph processing based on 3D DRAM [5]. Several SW interface designs have also been proposed for heterogeneous computing platforms to use the accelerators in system and coherently access host memory, e.g., IBM CAPI [217], Intel HARP [109], and HMC 2.0 [221].

Instead, in this work, we, for the first time, propose CryptoPIM, a high-throughput PIM accelerator for `NTT`-based polynomial multiplication. The proposed design optimizes the basic `NTT` operations, introduced new inter-block switches, and finally uses a configurable architecture to support polynomial multiplications for polynomials with degrees up to 32k.

6.2 CryptoPIM for RLWE Polynomial Multiplier

In this section, we detail CryptoPIM, a processing in-memory design for NTT-based polynomial multiplier. We start by analyzing the base algorithms to identify the basic operations involved in the execution. We then show how these operations can be mapped to PIM. In the end, we put together the individual PIM pieces to present a high-speed and highly efficient architecture for NTT-based polynomial multiplier.

6.2.1 Operational Breakdown of Polynomial Multiplication

As discussed in Section 6.1, Algorithm 16 contains multiple sequential operations. The computational operations involve element-wise multiplication between two vectors and NTT calculation over a vector. While `bitrev()` only changes the sequence of data reads and not

the data itself. Algorithm 7 shows that NTT is further composed of element-wise vector addition, subtraction, and multiplication operations. Hence, NTT-based polynomial multiplication essentially comprises of bit-reversal and element-wise addition, subtraction, and multiplication. Further, each data operation is followed by a modulo operation (`mod q`) to maintain consistent bitwidth.

6.2.2 Mapping Operations to PIM

Section 6.1 shows that PIM based designs [128, 94, 218, 111] can implement arithmetic functions with high parallelism and energy efficiency. Moreover, using such designs for integer arithmetic, as required in our case, can further increase their benefits. Integer operations do not involve tracking decimal point (for fixed-point) or iterative data-dependent shifts (for floating-point). This simple computation logic of integer operations, combined with the vector-based operations in polynomial multiplication, make it a suitable candidate for PIM.

6.2.2.1 Data organization in CryptoPIM

A memory block is an array of memory cells, where each memory cell represents a bit. N continuous memory cells in a row represent an N-bit number, with the first cell storing the MSB. For a block with r rows and c columns, each row stores c/N numbers, with the entire block having a capacity of $(c/N) \times r$ N-bit numbers.

In PIM, each row has some data columns and some processing columns. While data columns store inputs and other relevant data like pre-computed factors, processing columns are primarily used for performing intermediate operations and storing temporary results. However, the data and processing columns are physically indistinguishable and their roles can be changed on-the-fly. An input vector with m N-bit values is stored in data columns such that

each N-bit number occupies the same N columns in m rows. This is illustrated in Figure 6.2

6.2.2.2 Polynomial multiplication in CryptoPIM CryptoPIM

We implement the functions in polynomial multiplication to PIM as follows:

Bit-reversal: Bit-reversal changes the sequence of data read. In the case of PIM, where an input vector is stored over different rows in a memory block, a bit-reversal operation is equivalent to changing the row to which a value is written (shown in Figure 6.2. This can be easily achieved while writing the vector to the block. The arrangement can either be hard-coded or be flexible according to the target application.

Addition/Subtraction: The state-of-the-art PIM designs perform vector-wide addition. For subtraction, 2's complement is taken for one of the inputs (subtrahend) and then addition is performed. We use similar techniques where basic bitwise operations are cascaded to implement 1-bit adder. Then, these multiple such 1-bit additions are used to implement an N-bit operation. Although a single N-bit addition/subtraction may be a slow operation, r of such operations can be executed in parallel in a $r \times c$ memory block without any additional overhead, as shown in Figure 6.2. The latency of N-bit addition is $6N + 1$ cycles [110] and for subtraction is $7N + 1$.

Multiplication: An N-bit multiplication operation is broken into simple shift and addition of partial products. First, the partial products are generated using bitwise operations and stored in the same row as operands. Because CryptoPIM works with bit-level memory access, instead of explicit data shift, shifting operation is translated to selecting appropriate columns of memory block. Similar to addition/subtraction, r multiplication operations can also execute in parallel in a memory block which provides efficient vector-wide operations. The work in [113] proposed full-precision multipliers. However, the bitwise operations used by them are expensive. Instead, we combine the algorithm in [113] with the low latency

bitwise operations proposed in [110]. As a result, N-bit multiplication in CryptoPIM takes $6.5N^2 - 11.5N + 3$ cycles, significantly less than the $13N^2 - 14N + 6$ cycles of [113].

Modulo: While addition of two N-bit numbers can result in an $(N + 1)$-bit output, a multiplication may give an output with $2N$ bits. However, to maintain the consistency in number bitwidth with minimum effect on the algorithmic correctness, each computation undergoes a modulo operation. Modulo operations traditionally involve the expensive division operations. To enable modulo operations in memory, we use Barrett reduction [27] and Montgomery reduction [156] after each addition and multiplication, respectively. Further, in NTT, the modulo factor (q in our case) is generally fixed for a specific size of the vector or polynomial's degree. For instance, we set $q = 12289$ for vector sizes of 1024 and 512, according to NewHope [179], and $q = 7681$ for vector sizes of 256 and less according to Kyber [16]. For vector sizes of 2k, 4k, 8k, 16k, and 32k we set $q = 786433$ according to the Microsoft SEAL library [53]. We exploit this limited set of possible qs to make reduction more efficient in PIM. Instead of naively using multiple multiplications, we first convert these reduce operations into successive shift and add/subtract operations as shown in Algorithm 8. Since, we have bit-level access, we perform only the necessary bit-wise computations. For example, in line 15 of Algorithm 8, $\{v \leftarrow u$ & ((1<<18)-1)$\}$ resets to 0 all but the 17 LSBs of the output of $\{u \leftarrow$ (a<<13) - (a<<9) + a$\}$. Here, we compute only 17 LSBs of u.

Since q is the same for all the values in a vector, they require same shift (i.e. same column activations in CryptoPIM) and can hence be completely parallelized over the entire vector in CryptoPIM. For each reduction, latency is dependent on the total number of bitwise additions involved in it. Table 6.1 lists the latency of each reduction in CryptoPIM.

Algorithm 8: Reduction algorithms using shifts and adds. $q = 7681$ for $n \leq 256$, $q = 7681$ for n=512, 1024, and $q = 786433$ for n$\geq$ 2048

```
1  Function BARRET_REDUCE(f, a, b, ε):
2      if q == 12289 then
3          u ← ((a<<2)+(a)) >> 16; u ←(u<<13) + (u<<12) + u
4      if q == 768 then
5          u ← a>>13; u ← (u<<13) - (u<<9) - u
6      if q == 786433 then
7          u ← a>>20; u ← (u<<19) + (u<<18) + u
8      return a-u
9
10 Function MONTGOMERY_REDUCE(f, a, b, ε):
11     if q == 12289 then
12         u ← (a<<13) + (a<<12) - a; u ← u & ((1<<18)-1)
13         u ← (u<<13) + (u<<12) + u; u ← (u + a)>>18
14     if q == 768 then
15         u ← (a<<13) - (a<<9) + a; u ← u & ((1<<18)-1)
16         u ← (u<<13) - (u<<9) - u; u ← (u + a)>>18
17     if q == 786433 then
18         u ← (a<<19) - (a<<18) + a; u ← u & ((1<<32)-1)
19         u ← (u<<19) + (u<<18) - u; u ← (u + a)>>32
20     return u
```

Table 6.1: Execution time (cycles) for modulo operation

q	**7681**	**12289**	**786433**
Barrett Reduction	261	239	429
Montgomery Reduction	683	461	1083

6.2.3 CryptoPIM Building Blocks

The two basic building blocks of CryptoPIM are PIM-enabled memory block and fixed-function switch. A memory block implements one phase or stage of the polynomial multiplication. Two adjacent memory blocks are connected using the CryptoPIM fixed-function switches as shown in Figure 6.3 Each memory block is a PIM enabled an array of 512×512 memory cells and can process a vector of length 512 at a time. The fixed-function switches, unlike typical crossbar switches, enable only three types of connections: a direct rowA-to-

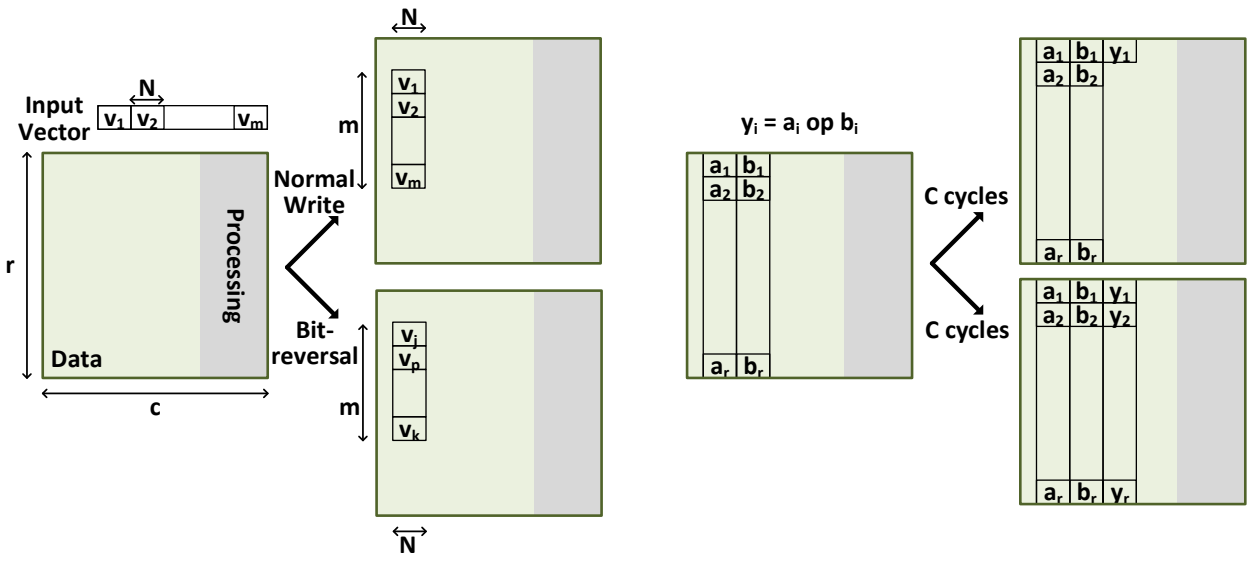


Figure 6.2: Data representation and parallel operations in CryptoPIM.

rowA, rowA-to-row(A+s), rowA-to-row(A-s). A fixed-function switch allows just one, the in-hardware-coded value of s. However, this 's' may be different (but fixed) for different instances of this fixed-function switch in the hardware. Instead, traditional crossbar switches provide a connection between any possible input/output combination leading to an exponential increase in logic requirement with an increase in inputs or outputs. Our fixed-function crossbar switch has just three logic switches per row. The number of switches per row is independent of the number of inputs/outputs. These switches can read up to one entire column of one block and write it to the next block in parallel, requiring as many cycles as the bitwidth of data. Hence, transferring data between two blocks in NTT requires only 3*bitwidth cycles, one each for A-to-A, A-to-(A+s), and A-to-(A-s) transfer.

All computation steps in Algorithm 5 and 7 are implemented using these building blocks. Each vector-wide data operation along with the involved modulo reduction is implemented in an independent memory block. Hence, we have a memory block each for $a_i\phi^i$, $b_i\phi^i$, $\bar{A}.\bar{B}$, $\bar{c_i}\phi^{-i}$. Then, the $log_2 n$ stages in NTT uses a memory block each. Each (NTT) stage block is connected to the next stage block using the switches.

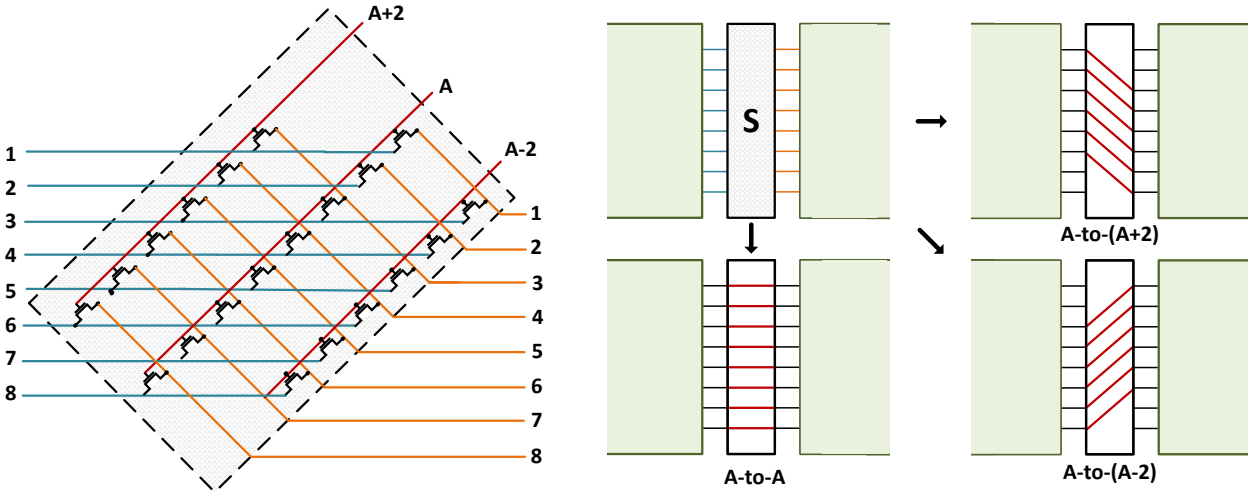


Figure 6.3: (Left) Detailed circuit of fixed-function switch with 8 inputs/outputs and shift factor of 2. (Right) All possible connections for s=2.

6.2.4 CryptoPIM Architecture

6.2.4.1 CryptoPIM Pipeline

The block-by-block modular architecture provides an opportunity to increase the throughput by pipelining. Figure 6.4a shows the most area-efficient NTT pipeline for the 16-bit datawidth. Here, the computation and its corresponding modulo operation are performed in the same block. For 16-bit datatype and n=256, this results in 2700 cycles per stage. Note that even though the logic performed in each stage have different latencies, the latency of the slowest stage determines the stage latency while pipelining. Now, the data computation and its modulo are completely independent operations and can be performed in separate blocks, leading to the pipeline shown in Figure 6.4b and stage latency of 1756 cycles. However, this comes at the cost of increasing the number of stages and hence, the total latency of one polynomial multiplication. We further optimized the pipeline by combining Montgomery reduction, addition/subtraction, and Barrett reduction in the same stage as shown in Figure 6.4c. We obtain the final stage latency of 1643 cycles.

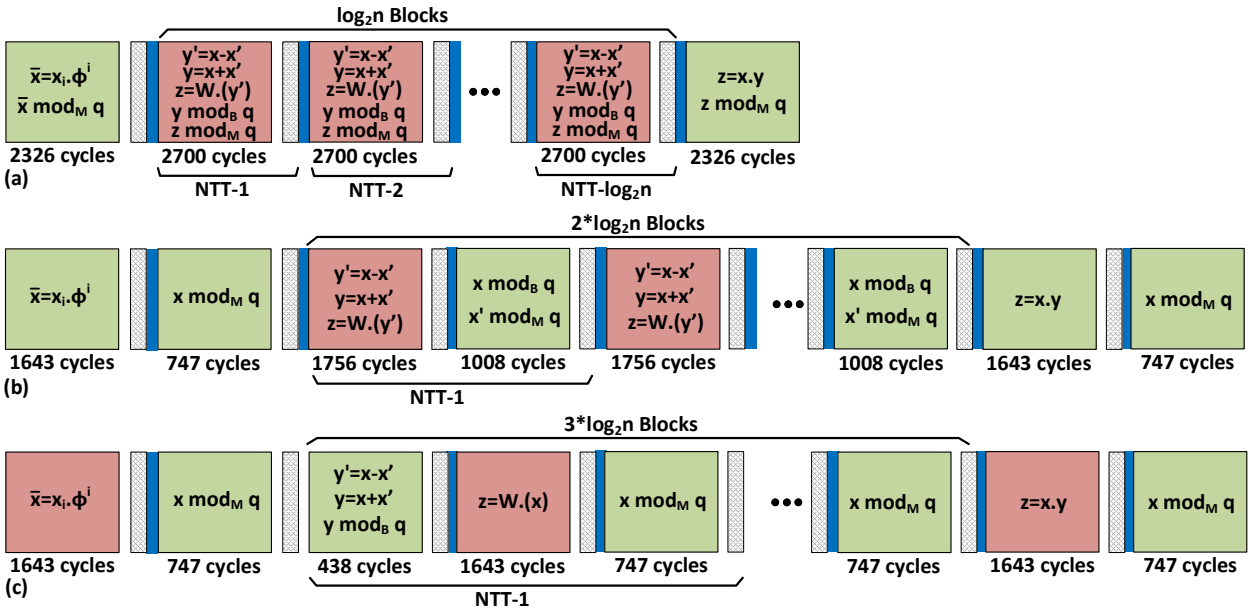


Figure 6.4: Detailed stage-by-stage breakdown of (a) Area-efficient pipeline, (b) naive pipeline, and (c) CryptoPIM pipeline. The region between two consecutive blue bars represents a pipeline stage. The slowest stage in a pipeline is colored red.

6.2.4.2 Configurable Architecture

CryptoPIM consists of a ReRAM memory chip with several memory banks. A set m cascaded memory blocks map to one memory bank. A memory bank takes in 512 parallel inputs in the first block and output 512-element wide vector. Hence, it can only process polynomials with degrees up to 512. However, the degree of the polynomials in RLWE/FHWE-based schemes generally ranges up to 32k. We design CryptoPIM architecture such that many of these banks can be dynamically arranged in the form of b_{soft} *softbanks*. A softbank consists of b_m parallel memory banks. Each softbank is responsible for processing vector-wide operations for a polynomial. Then, two softbanks dynamically form a *superbank* which completely processes multiplication between two input polynomials. To enable this configurability, CryptoPIM uses additional switches at the intersection of different banks and softbanks to allows data communication between them.

We optimize our hardware to support 32k degree polynomials in memory. A 32k NTT

pipeline has 49 blocks (from Figure 6.4). Hence, each bank has 49 memory blocks. We further need 64 such memory banks for each input polynomial, requiring 128 memory banks per 32k polynomial multiplication. If the degree of input polynomial is higher than 32k, CryptoPIM divides the inputs into segments of 32k and iteratively uses the hardware. On the other hand, if the input polynomial degree is less than 32k, we dynamically configure CryptoPIM into multiple superbanks to enable parallel multiplication of multiple polynomial pairs.

6.3 Evaluation

6.3.1 Evaluation Setup

We use an in-house cycle-accurate C++ simulator, which emulates CryptoPIM functionality. We used HSPICE for circuit-level simulations and calculate energy consumption and performance of all the CryptoPIM operations in the 45nm process node. We used System Verilog and Synopsys *Design Compiler* to implement and synthesize the CryptoPIM controller. The robustness of all proposed circuits, i.e., interconnect, has been verified by considering 10% process variations on the size and threshold voltage of transistors using 5000 Monte Carlo simulations. A maximum of 25.6% reduction in resistance noise margin was observed for RRAM devices. However, this did not affect the operations in CryptoPIM due to the high R_{OFF}/R_{ON}. We adopt a RRAM device with VTEAM model [141]. The device parameters [219] are chosen to fit practical devices [140] with switching delay of 1.1ns (= cycle time in CryptoPIM). The behavior of bitwise PIM operations has been demonstrated using a fabricated chip in [129].

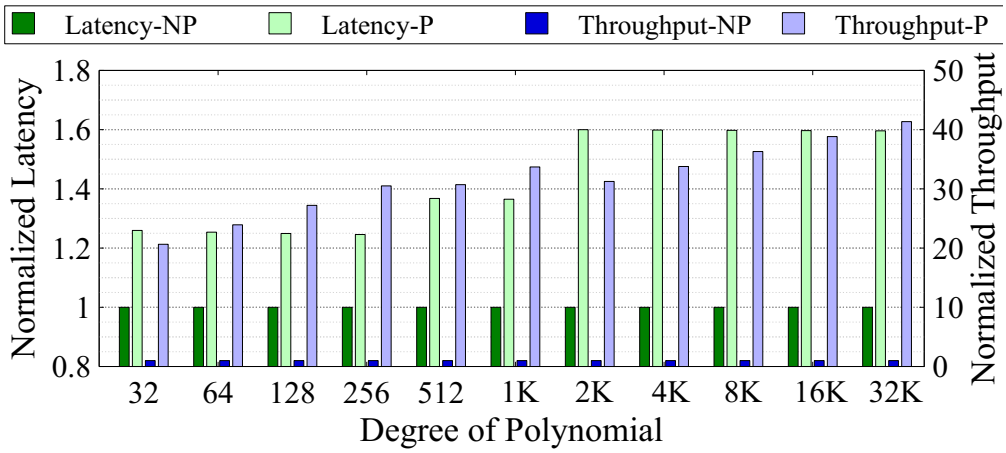


Figure 6.5: Normalized latency and throughput of CryptoPIM for different degrees of polynomial. NP and P represent non-pipelined and pipelined designs respectively.

6.3.2 Performance and Energy Consumption of CryptoPIM

Figure 6.5 shows the latency and throughput of non-pipelined and pipelined CryptoPIM over different degrees of polynomial multiplications (n). The latency of CryptoPIM increases with increase in n, primarily due to the increased number of NTT stages. However, the pipelined-throughput remains the same for the degrees processed in same bitwidth. This is because the latency of one stage in CryptoPIM depends on bitwidth and not n. The energy consumption of the design increases with n. This is due to increase in both the number of stages and well as the amount of parallel computations in each stage.

As evident from the results, pipelining increases the throughput tremendously with some latency overhead. For smaller degrees ($n \leq 1024$), average throughput improves by 27.8×, while incurring 29% latency overhead. When is $n > 1024$, the throughput improvement increases to 36.3×, with 59.7% increment in latency. This happens because the latency of multiplication increases exponentially with the bitwidth of inputs. For $n > 1024$ (32-bit inputs), the execution time of multiplication is 6.8× that of the second slowest operation. On the other hand, for $n \leq 1024$ (16-bit inputs), multiplication is just 2.3× slower than the second slowest operation. Hence, the pipelines is comparatively more balanced in 16-

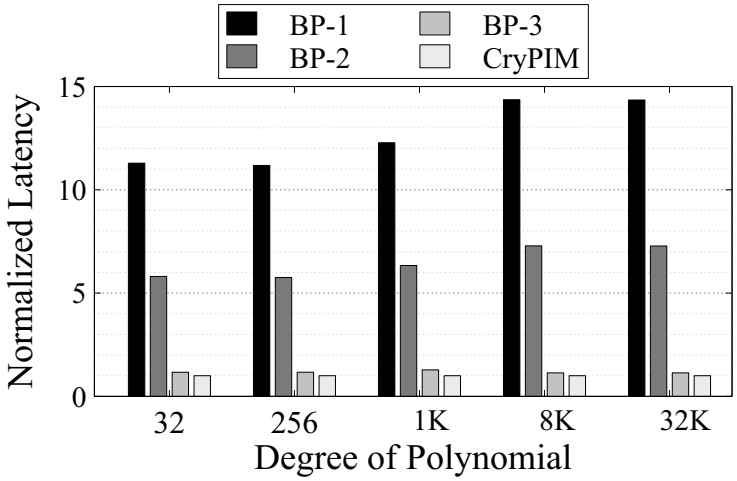


Figure 6.6: Comparison of CryptoPIM with PIM baselines.

bit than 32-bit. The energy consumption of CryptoPIM increases on average by just 1.6% for the pipelined design. While pipelining increases the number of stages, the underlying computations don't increase. Hence, the total amount of logic is same in both the pipelined and non-pipelined versions. The small increase in energy is due to increased block-to-block transfers.

6.3.3 Comparison with state-of-the-art PIM

To show the efficiency of our optimizations, we compare CryptoPIM with multiple PIM baselines. The first baseline PIM (BP-1) uses the operations proposed in [113], while utilizing the same building blocks and architecture as CryptoPIM. BP-2 is BP-1 with its N-bit multiplication replaced with the multiplication in CryptoPIM. BP-3 is BP-2 with the reduction operations converted to shift and adds. Figure 6.6 shows the latency of the three baselines and CryptoPIM for different degrees of polynomial multiplication. To have a fair comparison, we compare the baselines with the non-pipelined version of the design. We observe that BP-2 is on average 1.9× faster than BP-1. This shows that optimized CryptoPIM multiplications significantly improve CryptoPIM latency. Moreover, BP-3 is 5.5× faster than BP-2, which shows that the shift and add based reduction is more efficient than multiplication

based reduction. Finally, CryptoPIM is 1.2× faster than BP-3, showing that CryptoPIM modulo reductions are highly optimized. As a result, CryptoPIM is 12.7× faster than the state-of-the-art PIM (BP-1).

6.3.4 Comparison with CPU and FPGA

Table 6.2 shows the comparison of the pipelined-CryptoPIM in terms of latency, energy, and throughput to the FPGA (on Xilinx Zynq UltraScale+) and software (on an X86 CPU at 2GHz) implementations. Compare to the CPU implementation, CryptoPIM on average achieves 7.6x, 111x, and 226x improvement in the performance, throughput, and energy, respectively. For the sizes suitable for public-key encryption (256, 512, and 1024) CryptoPIM achieves on average 31x throughput improvement with the same energy and less than %30 reduction in the performance.

6.4 Conclusion

The (NTT) is the most time-consuming routine in ideal lattice-based cryptographic protocols. In this paper, we propose a high-throughput Processing-In-Memory accelerator for NTT-based polynomial multiplication. Our fast energy-efficient design, CryptoPIM, enables fast execution of the polynomial multiplication with the support of polynomials with degrees up to 32k, accommodating requirements for public key cryptographic systems for data at rest and in communication, and data in use (e.g., homomorphic encryption). CryptoPIM achieves on average 31x throughput improvement with the same energy consumption and 28% latency reduction compare to the fastest NTT-based polynomial multiplier implemented on FPGA.

Table 6.2: Comparison of the CryptoPIM to the FPGA and CPU implementation of the NTT-based polynomial multiplier. Throughput is defined as number of the polynomial multiplications per seconds. Energy is the require energy to multiply two polynomials.

Design	N	Bitwidth	Latency (us)	Energy (uJ)	Throughput
X86 (gem5)	256	16	84.81	570.60	11790
	512	16	168.96	1179.52	5918
	1k	16	349.41	2483.77	2861
	2k	32	736.92	5273.07	1365
	4k	32	1503.31	10864.64	665
	8k	32	3066.76	22385.51	326
	16k	32	6256.20	46123.84	159
	32k	32	12762.65	95032.33	78
NTT-based [162] (FPGA)	256	16	21.56	2.15	46382
	512	16	47.63	5.28	20995
	1k	16	101.84	12.52	9819
	2k-32k	-	-	-	-
CryptoPIM Pipelined	256	16	68.67	2.58	553311
	512	16	75.90	5.02	553311
	1k	16	83.12	11.04	553311
	2k	32	363.60	82.57	137511
	4k	32	392.69	178.62	137511
	8k	32	421.78	384.17	137511
	16k	32	450.87	822.21	137511
	32k	32	479.95	1752.15	137511

Chapter 7

Conclusions and Future Directions

Industry, academic, and government are working together to commercialize quantum computers – computers that can achieve unprecedented levels of performance in specific application domains, including biology and chemistry. With the compute power of quantum computers, the existence of quantum algorithms for solving problems such as the discrete logarithm problem promises to nullify the effectiveness of current public-key cryptography, as illustrated by in Shor's algorithm to factorize large integers in polynomial time.

Fortunately, Post-quantum cryptography (PQC) is a vibrant area of research devoted to studying alternative schemes for public-key cryptographic protocols, capable of withstanding quantum cryptanalysis attacks, and executable on classical computers. The relevance of the problem and the urgency of the threat is also demonstrated by the activity of government agencies, which are currently promoting the evaluation (first) and standardization process (next) of suites composed of such new algorithms. Lattice-based cryptography (`LBC`) schemes are the most promising family of quantum-resistant schemes due to their versatility and superior performance. In both the first and second rounds of the NIST PQC competition, about half of the candidates belong to the LBC family. We focus on aera, performance,

and energy efficiency as the primary optimization goals since such objectives are the most relevant for resourceful servers and cloud-based accelerators, as well as resource-constrained IoT devices.

We focus on domain-specific programmable hardware accelerators because they are the most suitable, at this stage, to adopt post-quantum algorithms in hardware. Generation of domain specific accelerators involves considering the spectrum of the computing platforms which ranges from resource-constrained IoT devices to performance-driven cloud servers.

Efficiency in terms of performance, energy, and sustainability of the designed accelerators (i.e., robustness to the changes in algorithms and parameters) are crucial for early adoption of such cryptographic protocols in hardware. We satisfy such goals by (a) analyzing and profiling schemes, and (b) targeting parts of the reference routines, which we address as micro-kernels. This is in contrast to the traditional approach of designing a dedicated co-processor for the entire routine. The employment of the micro-kernel accelerators provides more flexibility but has inferior performance compared to the dedicated co-processor because, in the micro-kernel approach, the main processor has to run portions of the protocol. Towards this goal, we design cache and DMA assisted accelerators for resource-intensive kernels of the LBC and verify them in We validate our methodology by integrating our accelerators into an HLS-based SoC infrastructure based on the X86 processor and evaluate overall performance.

Among the validated accelerators in the proposed simulation framework, we implement polynomial multipliers on FPGA and PIM platforms. We explore energy efficiency of different array-based polynomial multipliers. Then, we compare our FPGA implementation with our high-throughput PIM design.

Bibliography

[1] Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, 1997.

[2] Ieee standard specification for public key cryptographic techniques based on hard problems over lattices. *IEEE Std 1363.1-2008*, 2009.

[3] Nist: National institute for standards and technology. postquantum crypto project, 2017.

[4] C. Aguilar-Melchor, J. Barrier, S. Guelton, A. Guinet, M.-O. Killijian, and T. Lepoint. Nfllib: Ntt-based fast lattice library. In *CT-RSA*, 2016.

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News*, 2016.

[6] S. Akleylek, E. Alkım, and Z. Y. Tok. Sparse polynomial multiplication for lattice-based cryptography with small complexity. *The Journal of Supercomputing*, 2016.

[7] S. Akleylek, N. Bindel, J. Buchmann, J. Krämer, and G. A. Marson. An efficient lattice-based signature scheme with provably secure instantiation. In *AFRICACRYPT*, 2016.

[8] S. Akleylek, Ö. Dağdelen, and Z. Y. Tok. On the efficiency of polynomial multiplication for lattice-based cryptography on gpus using cuda. In *ICCISB*, 2015.

[9] M. R. Albrecht et al. Implementing rlwe-based schemes using an rsa co-processor. Cryptology ePrint Archive, Report 2018/425, 2018.

[10] E. Alkim, N. Bindel, J. Buchmann, Özgür Dagdelen, E. Eaton, G. Gutoski, J. Krämer, and F. Pawlega. Revisiting tesla in the quantum random oracle model. Cryptology ePrint Archive, 2015.

[11] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - a new hope. Cryptology ePrint Archive, 2015.

[12] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Newhope without reconciliation. Cryptology ePrint Archive, 2016.

[13] E. Alkim, P. Jakubeit, and P. Schwabe. Newhope on arm cortex-m. In *SPACE*, 2016.

[14] A. Ansarmohammadi, H. Nejatollahi, and G. Mehdi. A low-cost implementation of aes accelerator using hw/sw co-design technique. In *CADS*, 2013.

[15] A. Ansarmohammadi, S. Shahinfar, and H. Nejatollahi. Fast and area efficient implementation for chaotic image encryption algorithms. In *CADS*, 2015.

[16] R. Avanzi, J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-kyber. Technical report, National Institute of Standards and Technology, 2017.

[17] A. Aysu, C. Patterson, and P. Schaumont. Low-cost and area-efficient fpga implementations of lattice-based cryptography. In *HOST*, 2013.

[18] A. Aysu and P. Schaumont. Precomputation methods for hash-based signatures on energy-harvesting platforms. *TC*, 2016.

[19] A. Aysu, B. Yuce, and P. Schaumont. The future of real-time security: Latency-optimized lattice-based digital signatures. 2015.

[20] S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. In *CT-RSA*, 2014.

[21] S. Bai, A. Langlois, T. Lepoint, D. Stehlé, and R. Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In *ASIACRYPT*, 2015.

[22] T. Bai et al. Analysis and acceleration of NTRU lattice-based cryptographic system. In *SNPD '14*, 2014.

[23] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *Proceedings of the Annual International Conference on Theory and Applications of Cryptographic Techniques*, 2012.

[24] U. Banerjee et al. Sapphire: A configurable crypto-processor for post-quantumlattice-based protocols. *IACR TCHES*, 2019.

[25] R. E. Bansarkhani. Kindi. Technical report, National Institute of Standards and Technology, 2017.

[26] P. S. L. M. Barreto, P. Longa, M. Naehrig, J. E. Ricardini, and G. Zanon. Sharper ring-lwe signatures. Cryptology ePrint Archive, 2016.

[27] P. Barrett. Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor. In *CRYPTO*, 1986.

[28] K. Basu et al. Nist post-quantum cryptography- a hardware evaluation study. Cryptology ePrint Archive, Report 2019/047, 2019.

[29] D. J. Bernstein. Chacha, a variant of salsa20. In *SASC*, 2008.

[30] D. J. Bernstein. New stream cipher designs. chapter The Salsa20 Family of Stream Ciphers. 2008.

[31] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. Ntru prime: Reducing attack surface at low cost. Cryptology ePrint Archive, 2016.

[32] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. Ntru prime. Technical report, National Institute of Standards and Technology, 2017.

[33] G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Keccak. In *EUROCRYPT*, 2013.

[34] G. Bertoni et al. The keccak reference, 2011.

[35] S. Bhattacharya, O. Garcia-Morchon, R. Rietman, and L. Tolhuizen. spkex: An optimized lattice-based key exchange. Cryptology ePrint Archive, 2017.

[36] N. Bindel, S. Akleylek, E. Alkim, P. S. L. M. Barreto, J. Buchmann, E. Eaton, G. Gutoski, J. Kramer, P. Longa, H. Polat, J. E. Ricardini, and G. Zanon. qtesla. Technical report, National Institute of Standards and Technology, 2017.

[37] N. Bindel, J. Buchmann, and J. Krämer. Lattice-based signature schemes and their sensitivity to fault attacks, 2016.

[38] N. Bindel, J. Buchmann, J. Krämer, H. Mantel, J. Schickel, and A. Weber. Bounding the cache-side-channel leakage of lattice-based signature schemes using program semantics. Cryptology ePrint Archive, 2017.

[39] N. Binkert et al. The gem5 simulator. *SIGARCH*, 2011.

[40] A. Boorghany and R. Jalili. Implementation and comparison of lattice-based identification protocols on smart cards and microcontrollers. Cryptology ePrint Archive, 2014.

[41] A. Boorghany, S. B. Sarmadi, and R. Jalili. On constrained implementation of lattice-based cryptographic primitives and schemes on smart cards. 2015.

[42] J. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from lwe. In *CCS*, 2016.

[43] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. Crystals – kyber: a cca-secure module-lattice-based kem. Cryptology ePrint Archive, 2017.

[44] J. W. Bos, C. Costello, M. Naehrig, and D. Stebila. Post-quantum key exchange for the tls protocol from the ring learning with errors problem. In *SP*, 2015.

[45] G. E. Box, M. E. Muller, et al. A note on the generation of random normal deviates. *The annals of mathematical statistics*, 1958.

[46] M. Braithwaite. Experimenting with post-quantum cryptography, 2016.

[47] J. Buchmann, D. Cabarcas, F. Göpfert, A. Hülsing, and P. Weiden. Discrete ziggurat: A time-memory trade-off for sampling from a gaussian distribution over the integers. In *SAC*, 2013.

[48] J. Buchmann, E. Dahmen, E. Klintsevich, K. Okeya, and C. Vuillaume. Merkle signatures with virtually unlimited signature capacity. In *ACNS*, 2007.

[49] J. Buchmann, F. Göpfert, T. Güneysu, T. Oder, and T. Pöppelmann. High-performance and lightweight lattice-based public-key encryption. In *IoTPTS*, 2016.

[50] C. Chen, J. Hoffstein, W. Whyte, and Z. Zhang. pqntrusign: A modular lattice signature scheme. Technical report, National Institute of Standards and Technology, 2017.

[51] D. D. Chen, N. Mentens, F. Vercauteren, S. S. Roy, R. C. Cheung, D. Pao, and I. Verbauwhede. High-speed polynomial multiplication architecture for ring-lwe and she cryptosystems. *TCS*, 2015.

[52] D. D. Chen, G. X. Yao, R. C. Cheung, D. Pao, and C. K. Koç. Parameter space for the architecture of fft-based montgomery modular multiplication. *TC*, 2016.

[53] H. Chen, K. Laine, and R. Player. Simple encrypted arithmetic library - seal v2.1. Cryptology ePrint Archive, 2017.

[54] J. H. Cheon, D. Kim, J. Lee, and Y. Song. Lizard: Cut off the tail! practical post-quantum public-key encryption from lwe and lwr. Cryptology ePrint Archive, 2016.

[55] J. H. Cheon, S. Park, J. Lee, D. Kim, Y. Song, S. Hong, D. Kim, J. Kim, S.-M. Hong, A. Yun, J. Kim, H. Park, E. Choi, K. kim, J.-S. Kim, and J. Lee. Lizard. Technical report, National Institute of Standards and Technology, 2017.

[56] A. Chopra. Improved parameters for the ring-tesla digital signature scheme. Cryptology ePrint Archive, 2016.

[57] A. Chopra. Glyph: A new insantiation of the glp digital signature scheme. Cryptology ePrint Archive, 2017.

[58] G. Chunsheng. Integer version of ring-lwe and its applications. Cryptology ePrint Archive, 2017.

[59] P. G. Comba. Exponentiation cryptosystems on the ibm pc. *IBM systems journal*, 1990.

[60] S. Cook et al. On the minimum computation time of functions. *Ph.D. dissertation, Harvard University*, 1969.

[61] J. W. Cooley et al. An algorithm for the machine calculation of complex journal = Mathematics of Computation, fourier booktitle. 1965.

[62] D. B. Cousins, J. Golusky, K. Rohloff, and D. Sumorok. An fpga co-processor implementation of homomorphic encryption. In *HPEC*, 2014.

[63] Ö. Dagdelen, R. El Bansarkhani, F. Göpfert, T. Güneysu, T. Oder, T. Pöppelmann, A. H. Sánchez, and P. Schwabe. High-speed signatures from standard lattices. In *LATINCRYPT*, 2014.

[64] J.-P. D'Anvers, A. Karmakar, S. S. Roy, and F. Vercauteren. Saber: Mod-lwr based kem. Technical report, National Institute of Standards and Technology, 2017.

[65] J. P. David et al. Hardware complexity of modular multiplication and exponentiation. *TC*, 2007.

[66] R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Efficient software implementation of ring-lwe encryption. In *DATE*, 2015.

[67] C. C. del Mundo, V. T. Lee, L. Ceze, and M. Oskin. Ncam: Near-data processing for nearest neighbor search. In *MEMSYS*, 2015.

[68] A. W. Dent. A designer's guide to kems. In K. G. Paterson, editor, *Cryptography and Coding*, 2003.

[69] J. Ding, T. Takagi, X. Gao, and Y. Wang. Ding key exchange. Technical report, National Institute of Standards and Technology, 2017.

[70] J. Ding, X. Xie, and X. Lin. A simple provably secure key exchange scheme based on the learning with errors problem. 2012.

[71] M. R. Doomun et al. Energy consumption and computational analysis of rijndael-AES. In *International Conference in Central Asia on Internet*, 2007.

[72] C. Du and G. Ba. High-performance software implementation of discrete gaussian sampling for lattice-based cryptography. In *ITNEACC*, 2016.

[73] C. Du and G. Bai. Towards efficient discrete gaussian sampling for lattice-based cryptography. In *FPL*, 2015.

[74] C. Du and G. Bai. Efficient polynomial multiplier architecture for ring-lwe based public key cryptosystems. In *ISCAS*, 2016.

[75] C. Du and G. Bai. A family of scalable polynomial multiplier architectures for ring-lwe based cryptosystems, 2016.

[76] C. Du and G. Bai. Towards efficient polynomial multiplication for battice-based cryptography. In *ISCAS*, 2016.

[77] C. Du, G. Bai, and X. Wu. High-speed polynomial multiplier architecture for ring-lwe based public key cryptosystems. In *GLSVLSI*, 2016.

[78] L. Ducas. Accelerating bliss: The geometry of ternary polynomials. Cryptology ePrint Archive, 2014.

[79] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal gaussians. In *CRYPTO*. 2013.

[80] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. Crystals-dilithium. Technical report, National Institute of Standards and Technology, 2017.

[81] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, 2017.

[82] L. Ducas, V. Lyubashevsky, and T. Prest. Efficient identity-based encryption over ntru lattices. In *ASIACRYPT*, 2014.

[83] L. Ducas and P. Q. Nguyen. Faster gaussian lattice sampling using lazy floating-point arithmetic. In *ASIACRYPT*, 2012.

[84] L. Ducas and T. Prest. Fast fourier orthogonalization. In *ISSAC*, 2016.

[85] N. C. Dwarakanath and S. D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Applicable Algebra in Engineering, Communication and Computing*, 2014.

[86] M. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[87] M. J. Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.

[88] R. El Bansarkhani and J. Buchmann. Improvement and efficient implementation of a lattice-based signature scheme. In *SAC*, 2013.

[89] P. Emeliyanenko. Efficient multiplication of polynomials on graphics hardware. In *APPT*, 2009.

[90] R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 1982.

[91] S. Fluhrer. Cryptanalysis of ring-lwe based key exchange with key share reuse. 2016.

[92] J. Folláth. Gaussian sampling in lattice based cryptography. *Tatra Mountains Mathematical Publications*, 2014.

[93] P.-A. Fouque et al. Falcon: Fast-fourier lattice-based compact signatures over ntru. Technical report, National Institute of Standards and Technology, 2017.

[94] D. Fujiki, S. Mahlke, and R. Das. Duality cache for data parallel acceleration. In *ISCA*, 2019.

[95] E. Fujisaki et al. Secure integration of asymmetric and symmetric encryption schemes. *Journal of Cryptology*, 2013.

[96] E. Fujisaki and T. Okamoto. How to enhance the security of public-key encryption at minimum cost. In *PKC*, 1999.

[97] M. Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 2009.

[98] O. Garcia-Morchon, R. Rietman, S. Sharma, L. Tolhuizen, and J. L. Torre-Arce. Dtls-himmo: Achieving dtls certificate security with symmetric key overhead. In *ESORICS*, 2015.

[99] O. Garcia-Morchon, Z. Zhang, S. Bhattacharya, R. Rietman, L. Tolhuizen, and J.-L. Torre-Arce. Round2. Technical report, National Institute of Standards and Technology, 2017.

[100] W. M. Gentleman et al. Fast fourier transforms: For fun and profit. In *AFIPS '66*, 1966.

[101] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[102] M. Gokhale, B. Holmes, and K. Iobst. Processing in memory: The terasys massively parallel pim array. *Computer*, 1995.

[103] N. Göttert et al. On the design of hardware building blocks for modern lattice-based encryption schemes. In *CHES*, 2012.

[104] S. Gueron and F. Schlieker. Speeding up r-lwe post-quantum key exchange. Cryptology ePrint Archive, 2016.

[105] T. Güneysu et al. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES*, 2012.

[106] T. Guneysu et al. Towards lightweight identity-based encryption for the post-quantum-secure internet of things. In *ISQED '17*, 2017.

[107] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Lattice-sased signatures: Optimization and implementation on reconfigurable hardware. 2015.

[108] T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe. Software speed records for lattice-based signatures. In *PQCrypto*, 2013.

[109] P. K. Gupta. Xeon+ fpga platform for the data center. In *CARL*, 2015.

[110] S. Gupta, M. Imani, and T. Rosing. Felix: Fast and energy-efficient logic in memory. In *ICCAD*, 2018.

[111] S. Gupta, M. Imani, and T. Rosing. Exploring processing in-memory for different technologies. In *GSVLSI*, 2019.

[112] T. Györfi, O. Cret, and Z. Borsos. Implementing modular ffts in fpgas – a basic block for lattice-based cryptography. In *DSD*, 2013.

[113] A. Haj-Ali et al. Imaging-in-memory algorithms for image processing. *TCAS I*, 2018.

[114] M. Hamburg. Three bears. Technical report, National Institute of Standards and Technology, 2017.

[115] J. Hoffstein et al. Ntru: A ring-based public key cryptosystem. In *ANTS-III*, 1998.

[116] J. Hoffstein et al. Ntrusign: Digital signatures using the ntru lattice. In *CT-RSA*, 2003.

[117] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, and W. Whyte. Transcript secure signatures based on modular lattices. In *PQCrypto*, 2014.

[118] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, W. Whyte, and Z. Zhang. Choosing parameters for ntruencrypt. In *CT-RSA*, 2017.

[119] J. Hoffstein, J. Pipher, W. Whyte, and Z. Zhang. A signature scheme from learning with truncation. Cryptology ePrint Archive, 2017.

[120] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the fujisaki-okamoto transformation. Cryptology ePrint Archive, 2017.

[121] J. Howe, A. Khalid, C. Rafferty, F. Regazzoni, and M. O'Neill. On practical discrete gaussian samplers for lattice-based cryptography. *TC*, 2016.

[122] J. Howe, C. Moore, M. O'Neill, F. Regazzoni, T. Güneysu, and K. Beeden. Lattice-based encryption over standard lattices in hardware. In *DAC*, 2016.

[123] J. Howe, T. Pöppelmann, M. O'neill, E. O'sullivan, and T. Güneysu. Practical lattice-based digital signature schemes. *TECS*, 2015.

[124] J. Howe, C. Rafferty, A. Khalid, and M. O'Neill. Compact and provably secure lattice-based signatures in hardware, 2017.

[125] N. Howgrave-Graham et al. Naep: Provable security in the presence of decryption failures. Cryptology ePrint Archive, 2003.

[126] A. Hülsing, J. Rijneveld, J. Schanck, and P. Schwabe. High-speed key encapsulation from ntru. In *CHES*, 2017.

[127] A. Hülsing, J. Rijneveld, J. M. Schanck, and P. Schwabe. Ntru-hrss-kem. Technical report, National Institute of Standards and Technology, 2017.

[128] M. Imani et al. Floatpim: In-memory acceleration of deep neural network training with high precision. In *ISCA*, 2019.

[129] B. C. Jang, Y. Nam, B. J. Koo, J. Choi, S. G. Im, S.-H. K. Park, and S.-Y. Choi. Memristive logic-in-memory integrated circuits for energy-efficient flexible electronics. *Advanced Functional Materials*, 2018.

[130] Z. Jin and Y. Zhao. Optimal key consensus in presence of noise. Cryptology ePrint Archive, 2017.

[131] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. In *USSR Academy of Sciences*, 1963.

[132] J. Kelsey. Sha-3 derived functions: cshake, kmac, tuplehash, and parallelhash. *NIST special publication*, 2016.

[133] A. Khalid, J. Howe, C. Rafferty, and M. O'Neill. Time-independent discrete gaussian sampling for post-quantum cryptography. In *FPT*, 2016.

[134] E. Kiltz, V. Lyubashevsky, and C. Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. Cryptology ePrint Archive, 2017.

[135] D. E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. 1997.

[136] D. E. Knuth and A. C. Yao. The complexity of nonuniform random number generation. *Algorithms and complexity: new directions and recent results*, 1976.

[137] H.-T. Kung. Why systolic architectures? *Computer*, 1982.

[138] P. Kuo et al. High performance post-quantum key exchange on fpgas. Cryptology ePrint Archive, Report 2017/690, 2017. `https://eprint.iacr.org/2017/690`.

[139] P.-C. Kuo, W.-D. Li, Y.-W. Chen, Y.-C. Hsu, B.-Y. Peng, C.-M. Cheng, and B.-Y. Yang. High performance post-quantum key exchange on fpgas, 2017.

[140] S. Kvatinsky et al. Magic memristor aided logic. *TCAS II*, 2014.

[141] S. Kvatinsky et al. Vteam: A general model for voltage-controlled memristors. *TCAS II*, 2015.

[142] A. Langlois and D. Stehlé. Worst-case to average-case reductions for module lattices. Cryptology ePrint Archive, 2012.

[143] R. Lindner and C. Peikert. Better key sizes (and attacks) for lwe-based encryption. In *CT-RSA'11*, 2011.

[144] Z. Liu, H. Seo, S. S. Roy, J. Großschädl, H. Kim, and I. Verbauwhede. Efficient ring-lwe encryption on 8-bit avr processors, 2015.

[145] P. Longa and M. Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. Cryptology ePrint Archive, 2016.

[146] P. Longa and M. Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In *CANS*, 2016.

[147] X. Lu, Y. Liu, D. Jia, H. Xue, J. He, and Z. Zhang. Lac. Technical report, National Institute of Standards and Technology, 2017.

[148] V. Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *ASIACRYPT*, 2009.

[149] V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, 2012.

[150] V. Lyubashevsky and D. Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. In *CRYPTO*, 2009.

[151] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT'10*, 2010.

[152] G. Marsaglia, W. W. Tsang, et al. The ziggurat method for generating random variables. *Journal of statistical software*, 2000.

[153] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[154] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM Journal on Computing*, 2007.

[155] D. Micciancio and M. Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. Cryptology ePrint Archive, 2017.

[156] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 1985.

[157] S. More and R. Katti. Discrete gaussian sampling for low-power devices. In *PACRIM*, 2015.

[158] M. Naehrig, E. Alkim, J. Bos, L. Ducas, K. Easterbrook, B. LaMacchia, P. Longa, I. Mironov, V. Nikolaenko, C. Peikert, A. Raghunathan, and D. Stebila. Frodokem. Technical report, National Institute of Standards and Technology, 2017.

[159] H. Nejatollahi, N. Dutt, I. Banerjee, and R. Cammarota. Domain-specific accelerators for ideal lattice-based public key protocols. Cryptology ePrint Archive, Report 2018/608, 2018.

[160] H. Nejatollahi, N. Dutt, and R. Cammarota. Trends, challenges and needs for lattice-based cryptography implementations: Special session. In *CODES*, 2017.

[161] H. Nejatollahi et al. Domain-specific accelerators for ideal lattice-based public key protocols. Cryptology ePrint Archive, Report 2018/608, 2018.

[162] H. Nejatollahi et al. Exploring energy efficient quantum-resistant signal processing using array processors. Cryptology ePrint Archive, 2019.

[163] H. Nejatollahi et al. Flexible ntt accelerators for rlwe lattice-based cryptography. *ICCD*, 2019.

[164] H. Nejatollahi et al. Post-quantum lattice-based cryptography implementations: A survey. *ACM Comput. Surv.*, 2019.

[165] H. Nejatollahi et al. Cryptopim: In-memory acceleration forlattice-based cryptographic hardware. *DAC*, 2020.

[166] H. Nejatollahi et al. Synthesis of flexible accelerators for early adoption of ring-lwe post-quantum cryptography. *TECS*, 2020.

[167] H. Nejatollahi, S. Shahhosseini, R. Cammarota, and N. Dutt. Exploring energy efficient quantum-resistant signal processing using array processors. In *ICASSP*, 2020.

[168] NTTCorporation. Psec-kem specification, 2008.

[169] H. Nussbaumer. Fast polynomial transform algorithms for digital convolution. *TASSP*, 1980.

[170] T. Oder, T. Güneysu, F. Valencia, A. Khalid, M. O'Neill, and F. Regazzoni. Lattice-based cryptography: From reconfigurable hardware to asic. In *ISIC*, 2016.

[171] T. Oder, T. Pöppelmann, and T. Güneysu. Beyond ecdsa and rsa: Lattice-based digital signatures on constrained devices. In *DAC*, 2014.

[172] J. Olson et al. Quantum information and computation for chemistry. *arXiv preprint arXiv:1706.05413*, 2017.

[173] C. Peikert. An efficient and parallel gaussian sampler for lattices. In *CRYPTO'10*, 2010.

[174] C. Peikert. Lattice cryptography for the internet. In *PQCrypto*, 2014.

[175] C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[176] L. T. Phong, T. Hayashi, Y. Aono, and S. Moriai. Lotus. Technical report, National Institute of Standards and Technology, 2017.

[177] T. Plantard. Odd manhattan. Technical report, National Institute of Standards and Technology, 2017.

[178] T. Pöppelmann. *Efficient Implementation of Ideal Lattice-Based Cryptography*. Ruhr-Universität Bochum, 2016.

[179] T. Poppelmann, E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, P. Schwabe, and D. Stebila. Newhope. Technical report, National Institute of Standards and Technology, 2017.

[180] T. Pöppelmann, L. Ducas, and T. Güneysu. Enhanced lattice-based signatures on reconfigurable hardware. In *CHES*, 2014.

[181] T. Pöppelmann and T. Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In *LATINCRYPT*, 2012.

[182] T. Pöppelmann and T. Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In *SAC*, 2013.

[183] T. Pöppelmann and T. Güneysu. Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *ISCAS*, 2014.

[184] T. Pöppelmann, M. Naehrig, A. Putnam, and A. Macias. Accelerating homomorphic evaluation on reconfigurable hardware. In *Lecture Notes in Computer Science*. 2015.

[185] T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. In *LATINCRYPT*, 2015.

[186] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li. Ndc: Analyzing the impact of 3d-stacked memory+ logic devices on mapreduce workloads. In *ISPASS*, 2014.

[187] T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers. Cryptology ePrint Archive, 2015.

[188] E. B. Rachid. Lara - a design concept for lattice-based encryption. Cryptology ePrint Archive, 2017.

[189] C. Rafferty, M. O'Neill, and N. Hanley. Evaluation of large integer multiplication methods on hardware. *TC*, 2017.

[190] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. 2005.

[191] C. P. Rentería-Mejía and J. Velasco-Medina. High-throughput ring-lwe cryptoprocessors. *TVLSI*, 2017.

[192] O. Reparaz, S. S. Roy, R. de Clercq, F. Vercauteren, and I. Verbauwhede. Masking ring-lwe. *Journal of Cryptographic Engineering*, 2016.

[193] M. Roetteler et al. Quantum resource estimates for computing elliptic curve discrete logarithms. In *Proc. ASIACRYPT*, 2017.

[194] M. Rosca, A. Sakzad, R. Steinfeld, and D. Stehlé. Middle-product learning with errors. Cryptology ePrint Archive, 2017.

[195] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and side channel secure discrete gaussian sampling. Cryptology ePrint Archive, 2014.

[196] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede. Compact ring-lwe cryptoprocessor. In *CHES'14*, 2014.

[197] S. S. Roy, F. Vercauteren, and I. Verbauwhede. High precision discrete gaussian sampling on fpgas. In *SAC*, 2013.

[198] M.-J. O. Saarinen. Gaussian sampling precision in lattice cryptography. Cryptology ePrint Archive, 2015.

[199] M.-J. O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. *Journal of Cryptographic Engineering*, 2017.

[200] M.-J. O. Saarinen. Hila5. Technical report, National Institute of Standards and Technology, 2017.

[201] M.-J. O. Saarinen. Hila5: On reliability, reconciliation, and error correction for ring-lwe encryption. Cryptology ePrint Archive, 2017.

[202] M.-J. O. Saarinen. Ring-lwe ciphertext compression and error correction: Tools for lightweight post-quantum cryptography. In *IoTPTS*, 2017.

[203] A. Schönhage and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 1971.

[204] M. Seo, J. H. Park, D. H. Lee, S. Kim, and S.-J. Lee. Emblem and r.emblem. Technical report, National Institute of Standards and Technology, 2017.

[205] S. Shao. Design and modeling of specialized architectures. *Ph.D. dissertation, Harvard University*, 2016.

[206] S. Shao et al. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ISCA '14*, 2014.

[207] S. Shao et al. Co-designing accelerators and soc interfaces using gem5-aladdin. In *MICRO*, 2016.

[208] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 1997.

[209] V. Shoup. Ntl: a library for doing number theory. 2016.

[210] Shousheng He and M. Torkelson. A new approach to pipeline fft processor. In *ICPP*, 1996.

[211] N. P. Smart, M. R. Albrecht, Y. Lindell, E. Orsini, V. Osheter, K. Paterson, and G. Peer. Lima. Technical report, National Institute of Standards and Technology, 2017.

[212] D. Stebila and M. Mosca. Post-quantum key exchange for the internet and the open quantum safe project. Cryptology ePrint Archive, 2016.

[213] D. Stehlé et al. Making ntru as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, 2011.

[214] D. Stehlé and R. Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *EUROCRYPT*, 2011.

[215] R. Steinfeld, A. Sakzad, and R. K. Zhao. Titanium. Technical report, National Institute of Standards and Technology, 2017. available at `https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions`.

[216] S. Streit and F. D. Santis. Post-quantum key exchange on armv8-a – a new hope for neon made simple. Cryptology ePrint Archive, 2017.

[217] J. Stuecheli, B. Blaner, C. Johns, and M. Siegel. Capi: A coherent accelerator processor interface. *IBM Journal of Research and Development*, 2015.

[218] N. Talati, R. Ben-Hur, N. Wald, A. Haj-Ali, J. Reuben, and S. Kvatinsky. mmpu—a real processing-in-memory architecture to combat the von neumann bottleneck. In *Applications of Emerging Memory Technology*. 2020.

[219] N. Talati et al. Logic design within memristive memories using memristor-aided logic (magic). *NANO*, 2016.

[220] E. E. Targhi et al. Post-quantum security of the fujisaki-okamoto and oaep transforms. In *Theory of Cryptography*, 2016.

[221] M. Technology. Hybrid memory cube, 2017.

[222] D. B. Thomas, W. Luk, P. H. Leong, and J. D. Villasenor. Gaussian random number generators. *ACM CSUR*, 2007.

[223] O. Tobias and G. Tim. Implementing the newhope-simple key exchange on low-cost fpgas. In *LATINCRYPT*, 2017.

[224] J. Toldinas et al. Energy efficiency comparison with cipher strength of aes and rijndael cryptographic algorithms in mobile devices. 2011.

[225] J. Von Neumann. Various techniques used in connection with random digits. *National Bureau of Standards Applied Mathematics booktitle*, 1951.

[226] A. Wander et al. Energy analysis of public-key cryptography for wireless sensor networks. In *IEEE International Conference on Pervasive Computing and Communications*, 2005.

[227] F. Winkler. Polynomial algorithms in computer algebra. In *TMSC*. 1996.

[228] Y. Yuan, C.-M. Cheng, S. Kiyomoto, Y. Miyake, and T. Takagi. Portable implementation of lattice-based cryptography using javascript. In *CANDAR*, 2016.

[229] Z. Zhang, C. Chen, J. Hoffstein, and W. Whyte. Ntruencrypt. Technical report, National Institute of Standards and Technology, 2017.

[230] Y. Zhao, Z. jin, B. Gong, and G. Sui. A modular and systematic approach to key establishment and public-key encryption based on lwe and its variants. Technical report, National Institute of Standards and Technology, 2017.