TAL TECH

DOCTORAL THESIS

Hardware Realization of Lattice-based Post-Quantum Cryptography

Malik Imran

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2023

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
33/2023

Hardware Realization of Lattice-based Post-Quantum Cryptography

MALIK IMRAN

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Computer Systems

The dissertation was accepted for the defence of the degree of Doctor of Philosophy in Information and Communication Technologies on July 31 2023

Supervisor: Prof. Dr. Samuel Pagliarini,
Department of Computer Systems, Centre for Hardware Security,
Tallinn University of Technology,
Tallinn, Estonia

Opponents: Prof. Georg Sigl,
Technical University of Munich,
Munich, Germany

Prof. Francesco Regazzoni,
University of Amsterdam,
Amsterdam, The Netherlands

Defence of the thesis: August 29 2023, Tallinn

Declaration:
Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Malik Imran .....
signature

ISSN 2585–6898 (publication)
ISBN 978-9916-80-029-4 (publication)
ISSN 2585–6901 (PDF)
ISBN 978-9916-80-030-0 (PDF)
Printed by Koopia Niini & Rauam

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
33/2023

Võrel põhinev post-kvant-krüptograafia riistvaraline realisatsioon

MALIK IMRAN

Contents

List of Publications

The present Ph.D. thesis is based on the following publications that are referred to in the text by Roman numbers.

I M. Imran, Z. U. Abideen, and S. Pagliarini, "An experimental study of building blocks of lattice-based nist post-quantum cryptographic algorithms," *Electronics*, vol. 9, no. 11, 1953, 2020. DOI: https://doi.org/10.3390/electronics9111953

II M. Imran, Z. U. Abideen, and S. Pagliarini, "An open-source library of large integer polynomial multipliers," *in 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, Vienna, Austria, 2021, pp. 145–150. DOI: https://doi.org/10.1109/DDECS52668.2021.9417065

III M. Imran, Z. U. Abideen, and S. Pagliarini, "A versatile and flexible multiplier generator for large integer polynomials," *Journal of Hardware and Systems Security*, 2023. DOI: https://doi.org/10.1007/s41635-023-00134-2

IV M. Imran, F. Almeida, J. Raik, A. Basso, S. S. Roy, and S. Pagliarini, "Design space exploration of saber in 65nm asic," *in Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security, ASHES '21*, Virtual Event, Republic of Korea, 2021, pp. 85–90. DOI: https://doi.org/10.1145/3474376.3487278

V M. Imran, A. Aikata, S. S. Roy, and S. Pagliarini, "High-speed design of post-quantum cryptography with optimized hashing and multiplication," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023. DOI: https://doi.org/10.1109/TCSII.2023.3273821

VI M. Imran, F. Almeida, A. Basso, S. S. Roy, and S. Pagliarini, "High-speed SABER key encapsulation mechanism in 65nm CMOS." *Journal of Cryptographic Engineering*, 2023. DOI: https://doi.org/10.1007/s13389-023-00316-2

Other related publications

VII A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "Kali: A crystal for post-quantum security using kyber and dilithium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 2, pp. 747–758, 2023. DOI: https://doi.org/10.1109/TCSI.2022.3219555

VIII L. Aksoy, D. B. Roy, M. Imran, P. Karl, and S. Pagliarini, "Multiplierless design of very large constant multiplications in cryptography," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 11, pp. 4503–4507, 2022. DOI: https://doi.org/10.1109/TCSII.2022.3191662

IX F. Almeida, M. Imran, J. Raik, and S. Pagliarini, "Ransomware attack as hardware trojan: A feasibility and demonstration study," *IEEE Access*, vol. 10, pp. 44827–44839, 2022. DOI: https://doi.org/10.1109/ACCESS.2022.3168991

X T. Perez, M. Imran, P. Vaz, and S. Pagliarini, "Side-channel trojan insertion – a practical foundry-side attack via eco," *in 2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, Daegu, Korea, 2021, pp. 1–5. DOI: https://doi.org/10.1109/ISCAS51556.2021.9401481

XI M. Imran, S. Pagliarini, and M. Rashid, "An area aware accelerator for elliptic curve point multiplication," *in 2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Glasgow, UK, 2020, pp. 1–4. DOI: https://doi.org/10.1109/ICECS49266.2020.9294908

Abbreviations

ASIC	Application-specific Integrated Circuit
AES	Advanced Encryption Standard
CACR	Chinese Association for Cryptologic Research
CCA	Chosen Ciphertext Attack
CMOS	Complementary Metal Oxide Semiconductor
DSE	Design Space Exploration
DECAPS	Decapsulations
ECC	Elliptic Curve Cryptography
ENCAPS	Encapsulations
ENISA	European Union Agency for Cybersecurity
FPGA	Field-Programmable Gate Array
FOM	Figure-of-Merit
FFT	Fast Fourier Transform
FF	Flip-Flop
ITU	International Telecommunications Union
KEM	Key Encapsulation Mechanism
KEYGEN	Key Generation
LCM	Least Common Multiples
LWE	Learning With Errors
LWR	Learning With Rounding
LUT	Look-Up Table
NFS	Number Field Sieve
NIST	National Institute of Standards and Technology
NTT	Number Theoretic Transform
NWC	Negative Wrapped Convolution
PKC	Public-key Cryptography
PQC	Post-Quantum Cryptography
PAP	Power-Area-Performance
PCB	Printed Circuit Board
QC	Quantum Cryptography
RSA	Rivest, Shamir, and Adleman
ROM	Read Only Memory
RAM	Random Access Memory
RTL	Register-Transfer Level
SBM	Schoolbook Multiplier
TSMC	Taiwan Semiconductor Manufacturing Company

1 Introduction

The world is becoming increasingly digitized and connected, and ensuring the security and privacy of sensitive information has become a critical concern for individuals and organizations. The exponential growth of the internet has opened up many opportunities, but at the same time, it has also created new challenges. Recently, in [1], the International Telecommunications Union (ITU) announced that internet users increased from 400 million (in 2000) to 4.9 billion (in 2021). According to Snowden's report [2], in 2013, this growth rate is expected to be higher. The proliferation of internet users has increased numerous data breaches and cyber attacks in recent years, which have led to the theft of sensitive information, such as personal and financial data. These incidents have not only resulted in significant financial losses but have also damaged the reputation of organizations and eroded public trust. Various security measures have been developed and implemented to address these concerns, such as encryption/decryption [3], firewalls [4], and access controls [5]. However, despite these measures, the threat of cyber attacks remains a constant, and organizations must remain vigilant and take proactive measures to protect sensitive information. Hence, the increasing connectivity of the world has highlighted the importance of data security and privacy.

Cryptography is one of the techniques to protect sensitive information using mathematical problems [6]. It transforms original information/data into a format that humans cannot understand. The original information is called plaintext, while the text obtained after some mathematical operations is a ciphertext. The sequence of operations to obtain ciphertext from plaintext and vice versa determines a cryptographic algorithm/protocol. The current cryptographic schemes are categorized into symmetric and public-key cryptography (PKC). The sender and the recipient share a *common* key for encrypting and decrypting the message in symmetric key cryptography. The encryption is a transformation of plaintext into ciphertext, while the conversion back from ciphertext to plaintext is a decryption. Using a $common$ key in symmetric schemes makes symmetric cryptographic algorithms faster and more efficient for encrypting and decrypting large amounts of data. They are also more suitable for low-resource platforms such as wireless sensor nodes because they require less processing power and memory. However, the challenge with symmetric schemes is that the $common$ key needs to be shared over an unsecured channel between two parties (sender and receiver), which makes it potentially vulnerable to attacks.

On the other hand, PKC uses two keys, public and private. The public key is widely available and can be used by anyone to encrypt original information/data, while the private key is kept secret by the recipient and is used to decrypt the data. This makes PKC schemes secure, especially for applications that require longer-term security or when the parties involved have no prior relationship or secure communication channel. However, public-key schemes are typically slower and require more processing power and memory than symmetric schemes.

The choice of the cryptographic scheme depends on the specific requirements of the application or platform, such as the level of security needed. PKC-based cryptographic schemes are beneficial for achieving longer-term security, and their security strength depends on solving prime factorization and discrete logarithm problems. In number theory, integer factorization decomposes a composite number into a product of a smaller integer. The process is prime factorization if the roots are restricted to prime numbers. A composite number is a positive integer formed by multiplying two smaller positive

integers. In other words, it is a positive integer that has at least one divisor other than 1 and itself. Every positive integer is composite, prime, or unit, so the composite numbers are precisely those that are not prime and not a unit. For instance, integer 14 is a composite number because it is a product of the two smaller integers (i.e., 2×7). In contrast, the integers 2, 3, 5, and 7 are not composite numbers because they can divide only by 1 and themselves. Now, let us consider the following example to comprehend prime factorization. Take a prime P and let P be equal to 3240, and assume we need to find all prime roots/factors. The simplest way to do this is by finding the least common multiples (LCM), as factored in high school classes, and presented in Fig. 1 (left). The multiplication of the identified roots ensures the correctness of getting the original prime back. Note that the LCM method is effective only when the prime numbers are relatively small but for large primes, creating a tree diagram – as illustrated right side in Fig. 1 – is more beneficial.

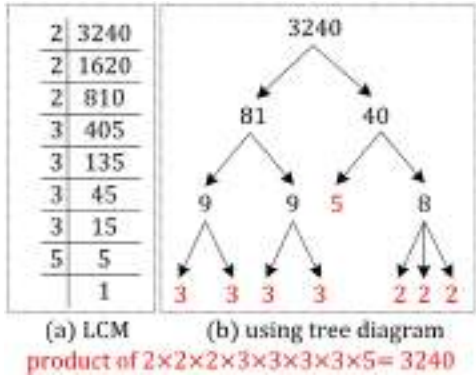


Figure 1: Methods for calculating prime factorization.

For discrete logarithms, we need to fix a prime P. Let a, b be nonzero integers $(mod\ P)$. The problem of finding x such that $a^x \equiv b\ (mod\ P)$ is called the discrete logarithm problem. Assume that n is the smallest integer such that $a^n \equiv (mod\ P)$. By assuming $0 \leq x < n$, we denote $x = L_{\mathsf{a}}(b)$ and call it the discrete logarithm of b with respect to $a\ (mod\ P)$. For example, let the prime $P = 11$, $a = 2$ and $b = 9$, then $x = L_2(9) = 6$.

Some open-source tools in the literature exist for factoring large primes and computing discrete logarithms. For example, an open-source CADO-NFS tool for integer factorization is available in [7], and it incorporates C/C++ implementations of the Number Field Sieve (NFS) algorithm [8] for factoring integers and computing discrete logarithms in finite fields. It is important to mention that not every integer is a prime, but for sufficiently large prime P, the literature demonstrates that the prime factorization and discrete logarithm problems are hard to solve on traditional computers and even on the fastest supercomputers because no efficient classical or non-quantum factorization algorithm is known.

The recent development in super-fast quantum computers [9, 10] raises issues in security and privacy. A quantum algorithm, named Shor's [11], provides a way to solve prime factorization and discrete logarithm problems exponentially faster than classical algorithms, making current PKC standards – Rivest, Shamir, and Adleman (RSA) [12] and elliptic curve cryptography (ECC) [13] – vulnerable to attacks by quantum computers. Therefore, two emerging directions such as quantum cryptography (QC)

and post-quantum cryptography (PQC) found in the literature to tackle these security concerns.

QC uses quantum mechanical properties to perform cryptographic tasks. At a very high level, the quantum cryptography model with the case of Alice, Bob, and Eve, is shown in Fig. 2. Alice and Bob want to send a secret to each other. Moreover, Alice sends Bob a series of polarized photons over a quantum channel (could be fiber optic cable), as shown in Fig. 2. If an eavesdropper, Eve, tries to listen in on the conversation, she must read each photon to read the secret. Then she must pass that photon on to Bob. By reading the photon, Eve alters the photon's quantum state, which introduces errors in the quantum key. This alerts Alice and Bob that someone is listening and the key has been compromised, so they discard it. Alice has to send Bob a new key that is not compromised, and then Bob can use that key to read the secret. The main advantage of quantum cryptography is that it allows the completion of many cryptographic tasks that are proven or presumed impossible using non-quantum communication. For instance, the data encoded by a quantum state is impossible to copy and modify. If someone tries to read the encoded data, the quantum state will be changed due to wave function collapse (no-cloning theorem [14]). This helps to detect eavesdropping in quantum key distribution.

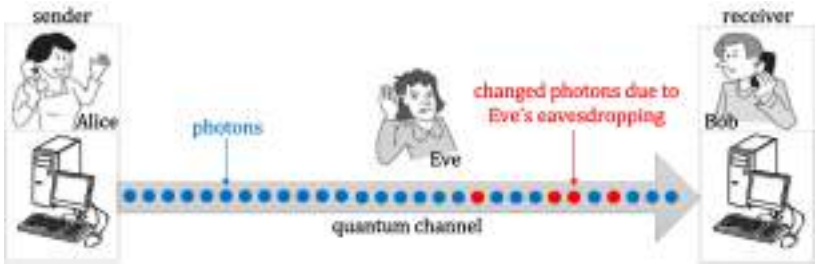


Figure 2: Quantum cryptography model with the case of Alice, Bob, and Eve.

On the other hand, PQC uses mathematical-based problems for constructing quantum-resilient algorithms or protocols to protect communications against quantum-computer attacks. Hence, the scientific community is constructing new reliable quantum-resistant cryptographic protocols, and standardization bodies and commercial organizations are also considering PQC alternatives. For example, in January 2020, the Chinese Association for Cryptologic Research (CACR) finished its PQC-standardization contest and selected LAC [15] as a winner for key establishment/agreement. Another example is an ongoing contest – initiated by the American National Institute of Standards and Technology (NIST) in 2017 – for post-quantum public-key cryptography standards. After the third round in 2022, NIST selected CRYSTALS-Kyber [16] and CRYSTALS-Dilithium [17] and stimulated the competition process in round four to investigate other protocols/algorithms. Note that quantum computers are still in their early stages of development, and only the big organizations like Google, IBM, etc., will have quantum computers soon; regular users wouldn't, and it may take a couple of years to come into the market. But some quantum computers have already been developed. In 2019, Google claimed to have the *Sycamore* – a 53 quantum bit (qubit) – quantum computer [9], which takes 200 seconds to sample one instance of a quantum circuit. The equivalent task on a supercomputer would take approximately 10,000 years. In 2021, IBM developed a 127 qubits processor, named *Eagle* [10]. According to [18], the *Eagle* chip is a step towards IBM's goal of creating a 433-qubit quantum processor

next year, followed by one with 1,121 qubits, named *Condor*, by 2023. Therefore, quantum-resistant cryptographic schemes are mandated to protect future and present communications.

The security strength of the NIST candidates for PQC standardization relies on several mathematical problems, including code, multivariate, isogeny, lattice, and hash. Amongst these, the lattice-based schemes are the most promising due to their computational efficiency, strong security assurance, and support for different applications; so from onward, this thesis discusses only lattice-based cryptography. Indeed, lattice-based cryptography has become a popular area of research in the last decade due to the introduction of the Learning With Errors (LWE) [19] and Learning With Rounding (LWR) [20] problems. The NIST selection of CRYSTALS-Kyber and CRYSTALS-Dilithium algorithms relies on LWE-based lattice cryptography, which confirms the increasing interest in this field. SABER [21], an LWR-based scheme, remained part of the NIST competition until round three [22] and is investigated as a case study in this thesis.

Despite the level of security needed, the choice of the cryptographic scheme (also) relies on the specific requirements of the application or platform, such as the available resources and the speed of encryption and decryption required. The applications related to the internet of things and wireless sensor nodes demand area- and power-constrained accelerators for cryptographic computations. High-speed cryptographic computations are always required for many applications, including wireless, telecom, cloud, data centers, enterprise systems, and network-related devices. For these applications, 8920 and 8955 families of Intel chipsets can process 5k, and 40k RSA decryption operations in one-second [23]. IBM 4769 hardware security module offers security services like key exchange and signature generation/verification using ECC and RSA standards [24]. Although these distinctive chips offer thousands of operations per second, they might become compromised since the security of ECC and RSA can be broken using Shor's algorithm [11] on a quantum computer. Hence, high-speed quantum-resistant cryptographic hardware accelerators are mandated to supersede ECC- and RSA-based devices.

The most commonly used platforms for implementing hardware accelerators are field programmable gate array (FPGA) and application-specific integrated circuit (ASIC). FPGAs are programmable hardware devices that can be configured and reconfigured to perform various tasks, including PQC algorithm acceleration. It offers several advantages: flexibility, reusability, and low development cost, and it can also be used to accelerate multiple PQC algorithms, making them a versatile choice. ASIC, on the other hand, are custom-built integrated circuits that are optimized for specific tasks or applications and offer higher performance and power efficiency than FPGA. However, ASICs are expensive to design and manufacture and are not reconfigurable. The choice between FPGA and ASIC for implementing PQC hardware accelerators will depend on factors such as the specific PQC algorithm(s) being accelerated, the required performance, and the available resources and budget. Keeping these factors in mind, some existing FPGA and ASIC hardware accelerators of quantum-resistant protocols (such as CRYSTALS-Kyber, CRYSTALS-Dilithium, and SABER) are implemented in [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36]. These implementations only provide the hardware demonstrations without the design optimizations for specific to certain parameters (such as low area, low power, high speed, etc.), hence posing a question: *how to further maximize the performance of PQC algorithms when demonstrated as hardware accelerators*?. This is the problem that this thesis explores.

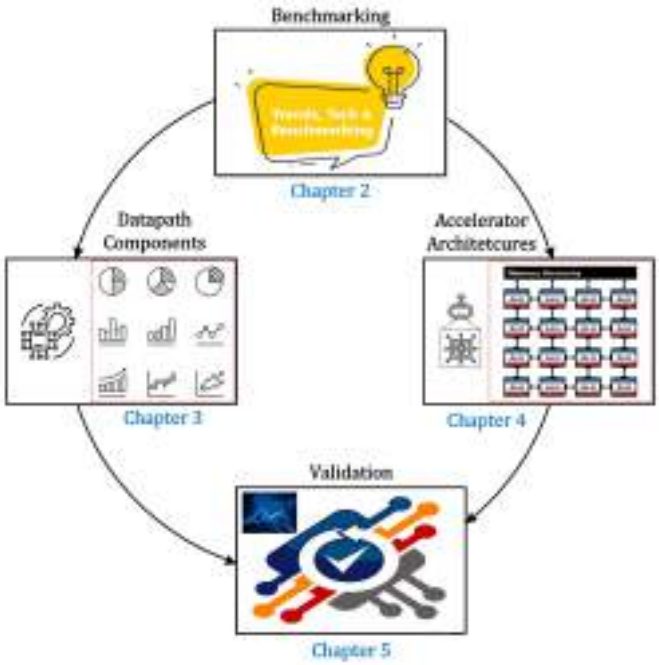


Figure 3: Structure of the thesis.

1.1 Novelty, Contributions & Summary of the Thesis

The thesis focuses on lattice-based PQC schemes and their performance improvement on the ASIC platform. Fig. 3 presents the overall structure of the thesis. Each chapter is a novel contribution to this thesis, and the corresponding details are as follows.

- **Chapter 2** This chapter gives a comprehensive overview of the concepts related to lattice-based PQC. Also, this chapter analyzes the building blocks of several lattice-based post-quantum algorithms, estimates their area and power on the ASIC platform, and concludes by selecting SABER [21] as the algorithm for hardware demonstrations and optimizations. Moreover, this chapter also describes the mathematical background for understanding the SABER algorithm and provides implementation platforms trade-off.

- **Chapter 3** The design of cryptographic hardware accelerators depends on polynomial arithmetic (addition, multiplication, inversion, sampling, hash, etc.) and logical operations in their datapath. However, polynomial multiplication is a computationally expensive operation in cryptographic schemes. Mostly the implementations of polynomial multipliers are specific to operands length and are not open-source for free use to everyone. Therefore, for the first time, I developed an open-source generator/tool for multiplying large integer polynomials to be used in conventional PKC algorithms (such as RSA and ECC) and PQC schemes. This chapter describes the structure/architecture of the developed multiplier generator tool. It offers flexibility, digitizing, pipelining, and also generates scripts for different ASIC synthesis tools, such as Cadence Genus and Design Compiler (DC) by Synopsis. Different figure-of-merits in Power-Area-Performance (PAP)

are defined to evaluate different polynomial multiplication architectures generated by the developed multiplier generator.

- **Chapter 4** The focus of this chapter is to provide a design space exploration (DSE) process of SABER for optimizing circuit frequency specific to the ASIC platform. The DSE process is initiated by setting a baseline architecture of SABER. Then, several memory types are utilized to evaluate the circuit frequency. Pipelining is incorporated to reduce the critical path of the SABER design. Parallel architectures are also proposed and implemented to reduce the clock cycle requirements for cryptographic computations, eventually improving the performance.

- **Chapter 5** In this chapter, a high-speed SABER chip is designed and fabricated on a 65nm process technology. It is important to mention that designing a Printed Circuit Board (PCB) is trivial for verification purposes. Therefore, I mount the fabricated chip on a PCB and interface it with a microcontroller, which helps to provide/collect inputs/outputs to/from the chip. All these details are described in this chapter. The fabricated chip is the fastest silicon demonstrated amongst state-of-the-art SABER chips regarding operating frequency.

- **Chapter 6** This chapter concludes the thesis. It provides future directions which indicate that the techniques studied in this thesis can be applied to other PQC algorithms, including CRYSTALS-Kyber and CRYSTALS-Dilithium, to improve their computation speed.

2 Background

This chapter describes the concepts related to lattice problems and the building blocks (i.e., multipliers, hash, samplers, etc.) needed for constructing lattice-based PQC algorithms in Sections 2.1 and 2.2, respectively. The SABER PQC protocol is described in Section 2.3. The existing hardware accelerators of lattice-based PQC algorithms are described in Section 2.4.

2.1 Lattice-Based Post-Quantum Cryptography

This section describes an overview of the hard problems defined over lattices. Such problems are a class of optimization problems and their conjectured intractability is the foundation of lattice-based public-key cryptography schemes [37]. Lattice problems have been studied for centuries and are considered hard to be solved. In 1996, Ajtai proposed the first significant public-key scheme using lattices in [38], which offered provable security, resistance to quantum computers, and worst-case hardness. Before defining the lattice problems, it is essential to define the elements (lattice, vector, and basis) on which the lattice problems depend.

- **Lattice.** A lattice $\mathcal{L} \in \mathbb{R}^m$ is a set of points in m-dimensional space with a periodic structure. An example of a two-dimensional lattice is shown in Fig. 4, where each box (filled with a black color) specifies the lattice point.

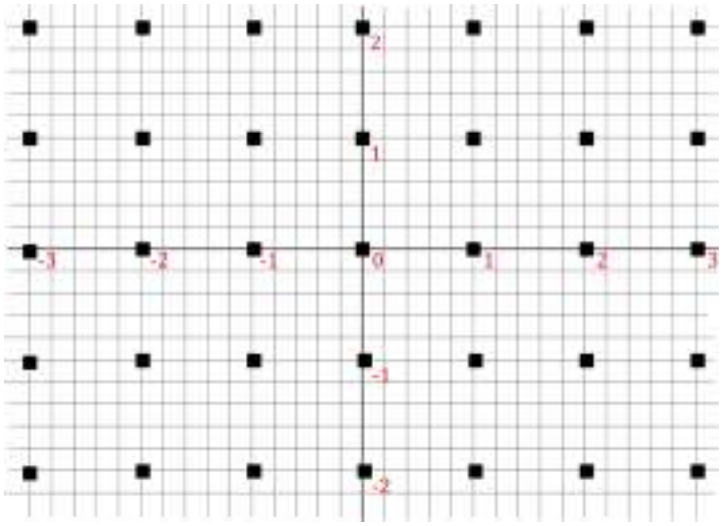


Figure 4: An example of a two-dimensional lattice over a set of all real numbers.

- **Vector.** A vector represents a quantity with magnitude (distance) and direction. Vectors can have different dimensions, however, the most intuitive is in two-dimensional or three-dimensional space. Below, Eq. 1 and Eq. 2 show the two-dimensional and three-dimensional vectors with their coordinates/elements.

$$\vec{v_1} = (2,1) \ \& \ \vec{v_2} = (2,8) \in \mathbb{R}^2 \tag{1}$$

$$\vec{v_1} = (2,1,4) \ \& \ \vec{v_2} = (2,8,5) \in \mathbb{R}^3 \tag{2}$$

- **Basis.** A basis is a collection of vectors to produce a point in a given space.

Definition 2.1.1. Lattice [39]. Let v be a set of n linearly independent vectors $v_0, v_1, \ldots, v_{n-1} \in \mathbb{R}^m$. The lattice $\mathcal{L}$ is the set of linear combinations of the vectors with coefficients in $\mathbb{Z}$, as shown in Eq. 3.

$$\mathcal{L} = \{a_0.v_0, + \cdots + a_{n-1}.v_{n-1}\} = \sum_{i=0}^{n-1} a_i.v_i \in \mathbb{Z} \tag{3}$$

In Eq. 3, v is a basis of $\mathcal{L}$, n specifies its rank and m determines its dimension. The lattice is a full-rank if $n = m$. Fig. 5 presents an example of a two-dimensional lattice with a basis of vectors v_1 and v_2. Any point in the lattice can be reached by an integer combination of vectors v_1 and v_2.

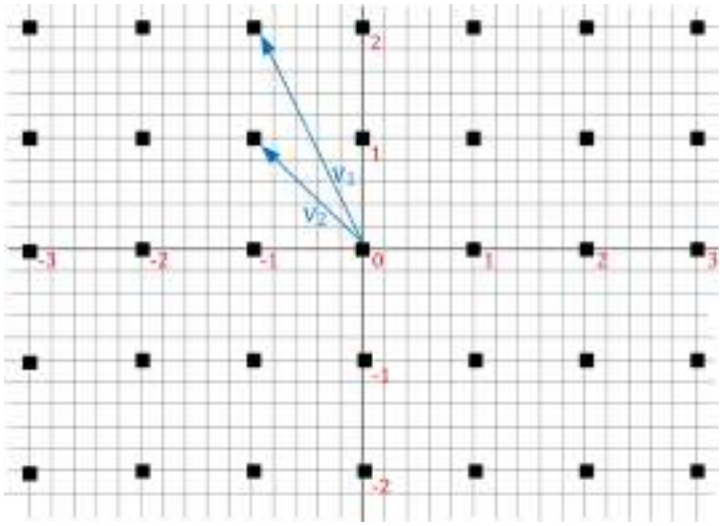


Figure 5: A two-dimensional lattice with two basis vectors v_1 and v_2. The coordinates of v_1 and v_2 are (-1, 2) and (-1, 1), respectively.

Lattice approximate problems. The shortest vector problem (SVP) and close vector problem (CVP) are two of the most important lattice approximate problems that play a significant role in the security of lattice-based cryptography [37]. These problems are presumed to be difficult to solve, which makes lattice-based cryptography secure. Therefore, the formal construction of SVP and CVP problems is described below.

Definition 2.1.2. SVP [40]. The SVP is finding the shortest non-zero vector in a lattice $\mathcal{L}$, which is defined by n linearly independent and randomly chosen basis vectors. In other words, find a non-zero vector v in a lattice $\mathcal{L}$ such that $\| v \| = \lambda_1(\mathcal{L})$, where $\| v \|$ is the Euclidean norm of the length of a vector v in $\mathcal{L}$, λ_1 is the shortest vector.

It shows in [38] that the SVP with Euclidean norm is NP-hard for randomized reductions. The SVP_γ is an γ-approximation version of the SVP where one has to find a vector v_γ in $\mathcal{L}$ such that $\| v_\gamma \| \leq \gamma \lambda_1(\mathcal{L})$.

Definition 2.1.3. CVP [39]. Given a target vector $t \in \mathbb{R}^m$ that is not necessarily in $\mathcal{L}$, find a vector $v \in \mathcal{L}$ that is closest to t. In other words, finding a vector $v \in \mathcal{L}$ reduces the Euclidean norm $\| t - v \|$.

Like SVP$_\gamma$, CVP$_\gamma$ is an γ-approximation of the CVP where one has to find a vector v_γ such that $\| t - v_\gamma \| \leq \| t - v \|$. Note that the CVP$_\gamma$ is the generalization of the SVP$_\gamma$. Thus, CVP is also known to be NP-hard [38].

The SVP or CVP or their approximate versions (SVP$_\gamma$ and CVP$_\gamma$) can be solved easily when a basis in a lattice consists of either orthogonal or near orthogonal vectors, also when short vectors are known. A set of orthogonal vectors describes a $good$ basis. Let us do examples to see the effect of bad and $good$ basis in lattice-based cryptography. The following examples are taken from [41]. Given a basis $B_{bad} = \{(6\ 14), (3\ 8)\}$ consisting of two vectors v_1 and v_2 with coordinates $(6\ 14)$ and $(3\ 8)$. Notice that v_1 and v_2 are not orthogonal to each other. Also, a target vector $t = (11.6\ 4.2)$ is given. Then the approximation problem asks for the nearest point of a given lattice to challenge the target point. The left portion in Fig. 6 describes the whole scenario, where a system of the equations for the given basis and target t must be solved to find the values of a and b. As seen in Fig. 6, the calculated values for a and b are real numbers (i.e., 13.4 and -22.9); these values cannot be used to calculate the lattice point (c), so the real values must be rounded up or down to get the integers (the closest value of 13.4 is 13 and -22.9 is -23) – this is the lattice approximation. After that, the values of a and b need to be used in the identical system of equations to calculate the lattice point. As shown green vector in Fig. 6, the calculated lattice point is (9 -2) and is far from the red vector, which is a target point $t = (11.6\ 4.2)$. The graphical visualization of the complete scenario is illustrated in the right part of Fig. 6, where the orange circle highlights that the target and calculated points are far from each other.

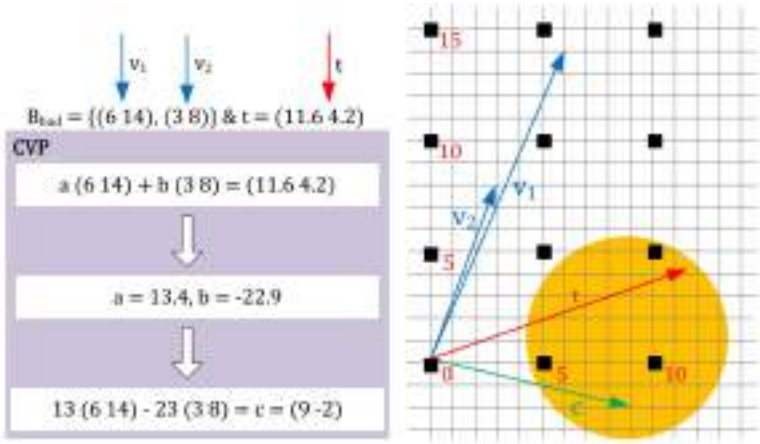


Figure 6: Example of a bad basis where the orange circle focuses on the target and calculated points far from each other. The purple portion solves the lattices for CVP.

Similarly, let us consider a basis $B_{good} = \{(3\ 0), (0\ 2)\}$ consisting of two vectors v_1 and v_2 with coordinates $(3\ 0)$ and $(0\ 2)$. Here, notice that the v_1 and v_2 are orthogonal. The same target vector $t = (11.6\ 4.2)$ is considered. The approximation problem asks for the closest point of a given lattice to challenge the target point. The left portion in Fig. 7 describes the whole scenario, where a system of equations for the given basis and target t must be solved to find the values of a and b. As seen in Fig. 7, the calculated values for a and b are real numbers (i.e., 3.86 and 2.1); these values cannot be used to calculate the lattice point c, so the real values must be rounded up or down to get the integers (the closest value of 3.86 is 4 and 2.1 is 2) – this is the lattice approximation.

After that, the values of a and b need to be used in the identical system of equations to calculate the lattice point. As shown by the green vector in Fig. 7, the calculated lattice point is (12 4) and is closest to the given target point $t = (11.6\ 4.2)$. The graphical visualization of the complete scenario is illustrated in the right part of Fig. 7, where the orange circle highlights that the target and calculated points are closer to each other.

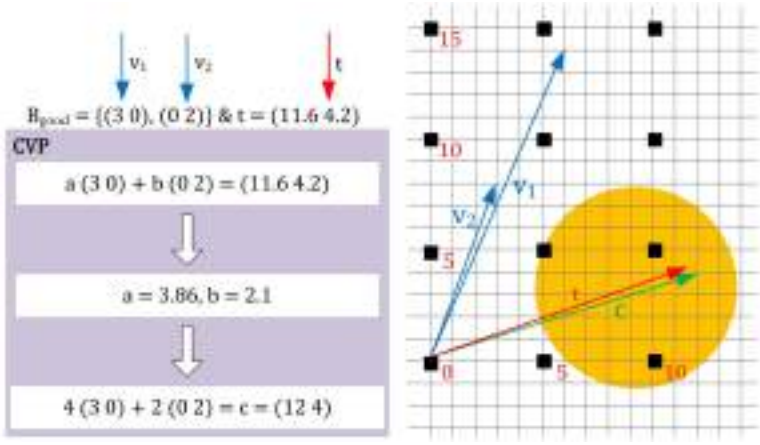


Figure 7: Example of a $good$ basis where the orange circle focuses on the target and calculated points closer to each other. The purple portion solves the lattices for CVP.

Consequently, lattice reduction algorithms in the literature aim to build a $good$ basis from any given basis for a lattice. For example, the LLL algorithm [42] outputs an LLL-reduced basis in a polynomial time but with the approximation factor of W^n, where W is a small constant. Hence, the LLL algorithm is effective in scenarios where the dimension n of the lattice is very small. The algorithms that achieve close approximation can run in approximation time. Examples of such algorithms are AKS [43], and BKZ [44]. The inability of the lattice reduction algorithms to find a good basis in polynomial time is used as the construction for lattice-based cryptography schemes.

LWE Problem. As reported earlier in this section, Ajtai described the first lattice-based public-key scheme in 1990 [38]. Later, in 2005, Regev [19] introduced a new lattice problem named LWE. Since its introduction, the LWE problem has become very popular for constructing various schemes such as public-key encryption, key exchange, digital signature generation/verification, and even homomorphic encryption schemes [37]. The LWE problem can be defined by a lattice with dimension n, an integer modulus q, and an error distribution χ over integers $\mathbb{Z}$. A secret vector $\mathbf{s}$ of dimension n is generated by choosing its coefficients uniformly in an n-dimensional ring $\mathbb{Z}_q^n$. Generate random vectors $\mathbf{a}_i$ by uniformly and error terms e_i from the error distribution χ. After that compute $b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i \in \mathbb{Z}_q$. Then the LWE distribution is denoted as A_s, χ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ and is the set of tuples (a_i, b_i). The lower bold characters show the vectors of dimension n. The decision and search are the two variants of LWE, defined below.

Definition 2.1.4. decision LWE problem [45]. Solving the decision LWE problem is to distinguish with non-negligible advantage between the samples drawn from LWE distribution A_s, χ and the same number of samples drawn uniformly from $\mathbb{Z}_q^n \times \mathbb{Z}_q$.

Definition 2.1.5. search LWE problem [45]. Find a secret $\mathbf{s}$ when a polynomial number of samples from the LWE distribution A_s, χ is given.

Note that the cryptosystems constructed on the security hardness of the original LWE

problem are slow because they need computations on larger matrices with coefficients from $\mathbb{Z}_q$. Hence, in literature, another computationally efficient variant of the LWE problem is defined over polynomial rings, called the ring-LWE problem [46].

Lyubashevsky, Peikert, and Regev initially introduced the ring-LWE problem [47]. Ring-LWE uses a particular class of lattices named "ideal lattices" to attain computational efficiency and reduce the key size. Therefore, the ring LWE problem is defined over a polynomial ring $R_q = \mathbb{Z}_q[x]/\langle f \rangle$, where $\langle f \rangle$ is an irreducible polynomial of degree n and coefficients of $\langle f \rangle$ contain modulus q. The problem is defined as follows: Sample a secret polynomial $s(x)$, and error polynomials $e_i(x) \in R_q$ with coefficients from χ. Next, generate polynomials $a_i(x)$ with coefficients chosen uniformly from $\mathbb{Z}_q$. Compute $b_i(x) = a_i(x).s(x) + e_i(x) \in R_q$. The ring-LWE distribution is the set of polynomial tuples $(a_i(x), b_i(x))$. As mentioned, e_i specifies the error polynomials with coefficients sampled from an n-dimensional error distribution χ. It is essential to highlight that the error distribution is a discrete Gaussian distribution except for some cases, e.g., for 2^k-power cyclotomics, where the error distribution is the product of n independent discrete Gaussians. Note that, in general, χ is more complicated to compute. One can form s by sampling the coefficients from χ rather than uniformly without any security implications [47].

Definition 2.1.6. decision ring-LWE problem [47]. Distinguish between the samples $(a_i(x), b_i(x))$ drawn from the ring-LWE distribution and the same number of samples generated by choosing the coefficients uniformly.

Definition 2.1.7. search ring-LWE problem [47]. Find a secret polynomial $s(x)$ given a polynomial number of samples constructed from the ring-LWE distribution.

Instead of the ring-LWE, another variant of LWE schemes is module-LWE. In contrast, ring-LWE uses polynomial ring elements, whereas module-LWR employs matrices of ring elements to define the problem. As summarized, there exist two cases. In the first case, when f specifies a cyclotomic polynomial [47], then the difficulty of the search ring-LWE problem is roughly equivalent to finding a short vector in an ideal lattice (composed of polynomials from R). A cyclotomic polynomial is a unique irreducible polynomial. In the second case, for the LWE problem, the security strength is related to solving the NP-hard SVP_γ over *general* lattices. These two cases are presumed to be equally difficult because no proof is known (to date) to show equivalence between the SVP_γ for general and ideal lattices. The computational efficiency using the ring-LWE problem is obtained at the cost of the above security assumption. The cryptographic schemes constructed on the ring-LWE problem are fast due to simple polynomial arithmetic [46].

LWR Problem. LWR is a variant of the LWE problem where random errors are replaced with deterministic rounding. Initially, the LWR problem was introduced in [48], and later, it was revisited in [49]. The LWR problem concerns the cryptographic properties of the function $f_s : \mathbb{Z}_q^n \to \mathbb{Z}_p$, given by $f_s(x) = \lfloor \langle x, s \rangle \rceil_p = \lfloor (P/q).\langle x, s \rangle \rceil$. Here, $s \in \mathbb{Z}_q^n$ and is a secret key. The term $\langle x, s \rangle$ determines the inner product of x and s mod q. The $\lfloor . \rceil$ denotes the closest integer. For mathematical derivations and more details, readers are referred to [38, 19] for the LWE problem and [48, 49, 50] for the LWR problem.

2.2 Building-Blocks for Lattice-Based Crypto Systems

This section deals with the building blocks of lattice-based PQC algorithms submitted to NIST for standardization. Currently, the NIST standardization process is in round four. I have started investigating the lattice-based PQC candidates submitted to NIST for standardization in 2020. At that time, the NIST competition was in round two. All

Table 1: Multiplication and hash methods for different PQC algorithms. These methods are obtained from their reference implementations, available at NIST sites [61] (after round-2) and [22] (after round-3).

PQC Algorithms	***Multiplication Methods***	***Hash Methods***
qTesla [51]	NTT and SBM	SHAKE-256 and cSHAKE-128/256
CRYSTALS-Dilithium [17]	NTT	SHAKE-128/256
NTRU-Prime [52]	SBM	SHA2-512
NewHope [53]	NTT	SHAKE-128/256
ThreeBears [54]	Karatsuba	cSHAKE-256
LAC [55]	SBM	–
Round5 [56]	SBM	cSHAKE-256 and AES-256
CRYSTALS-Kyber [16]	NTT	SHA3-256/512 and SHAKE-128/256
NTRU [57]	Karatsuba and Toom-Cook	SHA3-256
FrodoKEM [58]	SBM	SHAKE-128/256
Falcon [59]	SBM	SHAKE-256
SABER [21]	Karatsuba and Toom-Cook	SHAKE-128, SHA3-256/512

Note that the multiplication and hash methods in columns two and three have been considered from the reference C/C++ codes of PQC algorithms that were submitted to NIST for evaluation.

the lattice-based PQC algorithms that participated in the second and third rounds of the NIST contest are qTesla [51], CRYSTALS-Dilithium [17], NTRU-Prime [52], NewHope [53], ThreeBears [54], LAC [55], Round5 [56], CRYSTALS-Kyber [16], NTRU [57], FrodoKEM [58], Falcon [59], and SABER [21]. These PQC algorithms require various building blocks depending on the construction of the cryptographic protocol to perform cryptographic tasks. However, the polynomial multiplication and hash are the most critical operations to compute [60]. Table 1 lists different polynomial multiplication and hash operations, and the text below provides the implementation details of these multiplication and hash methods.

Polynomial multiplication involves multiplying two polynomials (i.e., a and b) and obtaining a resultant polynomial (i.e., c). The degree of the resulting polynomial is the sum of the degrees of the two input polynomials. The polynomial multipliers can be categorized into serial and parallel designs. In the case of bit-serial multipliers such as schoolbook (SBM) and Booth multipliers, the multiplication of polynomials is performed bit-by-bit, resulting in a sequence of partial products. These partial products are then added together to obtain the resultant polynomial. On the other hand, bit-parallel multipliers split the input polynomials into multiple parts and perform the multiplication of these parts in parallel. The inner product of the split portions is computed, and the resulting polynomial is generated using addition and subtraction operations. The 2-way Karatsuba multiplier is a famous bit-parallel multiplier that splits the input polynomials into two equal parts and uses three multiplications along with some additions and subtractions to compute the inner product. The 3-way and 4-way Toom-Cook multipliers split the input polynomials into three and four equal parts, respectively, and use a more complex algorithm to compute the inner product. Overall, bit-parallel multipliers are faster and more efficient than bit-serial multipliers, especially for larger input sizes. However, they also require more hardware resources and may not be practical for small input sizes.

SBM multiplier. SBM is the simplest way to multiply two input polynomials $a(x) \times b(x)$, as shown in Eq. 4. The resultant polynomial $c(x)$ is generated by performing bit-by-bit operations. Algorithm 1 shows the number of steps required to perform polynomial multiplication for the SBM multiplier, where polynomial a is multiplied with the shifted polynomial b to produce the resultant polynomial c. The latency associated with an SBM multiplier is $\lceil m \rceil$ clock cycles, whereas the operations to be computed

are $(m-1)$ additions and m multiplications (shifts).

$$c(x) = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_i b_j x^{i+j} \tag{4}$$

Algorithm 1: Traditional SBM multiplication

Input: a and b $(m-bit\ polynomial\ integers)$
Output: $c \leftarrow a \times b$
1 **for** *(j from* 0 *to* $m-1$*)* **do**
2 **if** $b_j = 1$ **then**
3 $c \leftarrow c + (a \times 2^j)$
4 **return** c

Booth multiplier. Similar to the SBM, the traditional Booth multiplier exploits add, subtract, and shift operations. Yet, unlike the SBM, it does not look at a bit at a time [62]. It observes two bits at a time and reduces the required addition and subtraction operations, ultimately reducing the multiplier's latency. The traditional Booth multiplication method is presented in Algorithm 2, where A keeps the generated partial product (initialized with 0). The $\overline{\mathrm{b}}$ shows the extended polynomial with the addition of a dummy 0-bit next to the least significant bit of the multiplier (b). It computes multiplication by inspecting the least significant two bits of the multiplier to match these four cases: 00, 01, 10, and 11. When the inspected bits are either 00 or 11, it means to do nothing or remain unchanged. For the remaining two cases, the multiplicand may be added (line 5) or subtracted (line 8) from the partial product (A). The $shift\_right\_add$ function of lines 6 and 9 in Algorithm 2 determines the multiplication of multiplicand by 2 with shift and add operations. For two operands of length m, Algorithm 2 takes $m/2$ clock cycles. Follow [62] for additional details.

Algorithm 2: Booth Multiplication

Input: a and b $(m-bit\ polynomial\ integers)$
Output: $c \leftarrow a \times b$
1 $A \leftarrow 0$ $(m-bit\ temporary\ integer)$
2 $\overline{\mathrm{b}} \leftarrow \{b, 0\}$
3 **for** *(j from* 0 *to* $m-1$*)* **do**
4 **if** $\overline{\mathrm{b}}_{j+1} \times \overline{\mathrm{b}}_j = 01$ **then**
5 $A \leftarrow A + a$
6 $c \leftarrow shift\_right\_add(A, \overline{\mathrm{b}}_{\mathrm{j}+1}, \overline{\mathrm{b}}_{\mathrm{j}})$
7 **if** $\overline{\mathrm{b}}_{j+1} \times \overline{\mathrm{b}}_j = 10$ **then**
8 $A \leftarrow A - a$
9 $c \leftarrow shift\_right\_add(A, \overline{\mathrm{b}}_{\mathrm{j}+1}, \overline{\mathrm{b}}_{\mathrm{j}})$
10 **return** c

Karatsuba multiplier. A generalized Karatsuba multiplier contains l number of levels to perform polynomial multiplication, where l depends on the user or designer to choose. For example, let us assume we have two input polynomials, z_1 and z_2. At the first level, z_1 and z_2 are divided into two smaller polynomials, $\frac{z_1}{2}$ and $\frac{z_2}{2}$. At the

second level, each split polynomial is further divided into two other polynomials, i.e., $\frac{z_1}{4}$ and $\frac{z_2}{4}$. The process of splitting polynomial repeats until the value l is reached. After splitting the input polynomials, the inner product can be computed, which is achieved using three inner multiplications, a few additions, and shift operations on small(er) operands. Eventually, the resulting polynomial is generated with the multiplications starting from the smaller polynomials to the larger one in a reverse order (meaning multiplications start from $\frac{z_1}{4}$ and $\frac{z_2}{4}$ to z_1 and z_2).

From Eq. 4, the split polynomial is derived in Eq. 5 where n shows the polynomial splits and k determines the index of the split polynomial. For a specific 2-way Karatsuba multiplier[1], the expanded version of Eq. 5 is shown in Eq. 6. It requires four multiplications for the execution of inner products (one to achieve the resulting polynomial $c_1(x)$, two multiplications for the execution of $c_2(x)$, and eventually one for the execution of $c_0(x)$). As presented in Eq. 7, the Karatsuba observation was to compute $c_2(x)$ with only one multiplication instead of two. The addition of inner products is required to generate the resultant polynomial $c(x)$, as presented in Eq. 8. Algorithm 3 provides the number of steps for the 2-way Karatsuba polynomial multiplication method. As the name implies, function *add_shift* in line 8 of Algorithm 3 applies the shift and add operations over the polynomials given in parentheses. In total, $\lceil \frac{m}{2} \rceil$ clock cycles are needed to implement one m-bit polynomial multiplication.

$$c(x) = \underbrace{\left(\sum_{i=\frac{k\times m}{n}}^{m-1} a_k(x) + \ldots + \sum_{i=0}^{\frac{k\times m}{n}-1} a_0(x) \right)}_{split\,polynomial\,a(x)} \times \underbrace{\left(\sum_{i=\frac{k\times m}{n}}^{m-1} b_k(x) + \ldots + \sum_{i=0}^{\frac{k\times m}{n}-1} b_0(x) \right)}_{split\,polynomial\,b(x)} \tag{5}$$

$$c(x) = \underbrace{a_1(x)b_1(x)}_{c_1(x)} + \underbrace{a_1(x)b_0(x) + a_0(x)b_1(x)}_{c_2(x)} + \underbrace{a_0(x)b_0(x)}_{c_0(x)} \tag{6}$$

$$c_2(x) = (a_1(x) + a_0(x)) \times (b_1(x) + b_0(x)) - c_1(x) - c_0(x) \tag{7}$$

$$c(x) = c_0(x) + c_1(x) + c_2(x) \tag{8}$$

Algorithm 3: 2-way Karatsuba Multiplication

Input: a and b $(m-bit\ polynomial\ integers)$
Output: $c \leftarrow a \times b$

1 $[b_1, b_0, a_1, a_0] \leftarrow \frac{[a,b]}{2}$
2 $c_0 \leftarrow a_0 \times b_1$
3 $c_1 \leftarrow a_1 \times b_1$
4 $c_{01} \leftarrow a_1 + a_0$
5 $c_{10} \leftarrow b_1 + b_0$
6 $c_2 \leftarrow c_{10} \times c_{01} - c_1 - c_0$
7 **for** *(j from* 0 *to* $\frac{m-1}{2}$*)* **do**
8 $\quad c \leftarrow c_0 + add\_shift(c_1, c_2)$
9 **return** c

Toom-Cook multiplier. The Toom-Cook multiplication method is the advanced and extended form of Karatsuba multiplication. The difference is in dividing input

[1] 2-way Karatsuba means that the splitting of input polynomials for Karatsuba multiplication is applied only once.

polynomials into 3 and 4 parts instead of 2 (as in 2-way Karatsuba). With index k of the split input polynomials, the values for $n = 3$ and $n = 4$ in Eq. 5 determine the equations of 3-way and 4-way Toom-Cook multipliers. The expanded version of Eq. 5 produces nine and sixteen inner multiplications for 3-way and 4-way Toom-Cook multipliers, respectively. Using a process identical to the 2-way Karatsuba, the required nine and sixteen inner multiplications can be reduced to five and seven. The equations for variants of the Toom-Cook multiplier are not shown as it requires an identical procedure to the 2-way Karatsuba. However, Algorithm 4 presents a complete understanding of the Took-Coom multiplication method when the split input polynomials are three smaller polynomials. As the name implies, function add_shift in line 8 of Algorithm 4 applies the shift and add operations over the polynomials given in parentheses. In total, $\lceil \frac{m}{3} \rceil$ and $\lceil \frac{m}{4} \rceil$ clock cycles are required to execute one m-bit polynomial multiplication.

Algorithm 4: 3-way Toom-Cook Multiplier

Input: a and b $(m-bit\ polynomial\ integers)$
Output: $c \leftarrow a \times b$

1 $[b_2, b_1, b_0, a_2, a_1, a_0] \leftarrow \frac{[a,b]}{3}$
2 $c_0 \leftarrow a_0 \times b_0$
3 $c_1 \leftarrow a_0 \times b_1 + a_1 \times b_0$
4 $c_2 \leftarrow a_0 \times b_2 + a_1 \times b_1 + a_2 \times b_0$
5 $c_3 \leftarrow a_1 \times b_2 + a_2 \times b_1$
6 $c_4 \leftarrow a_2 \times b_2$
7 **for** *(j from* 0 *to* $\frac{m-1}{3}$*)* **do**
8 $\quad c \leftarrow c_0 + add\_shift(c_1, c_2, c_3, c_4)$
9 **return** c

Multipliers based on Number Theoretic Transformation (NTT). The NTT-based polynomial multiplication is an efficient way to multiply two polynomials over ring $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$, where $\mathbb{Z}_q[X]/\langle X^n + 1\rangle$ represent the polynomial ring reduced with cyclotomic polynomial $(X^n + 1)$ over $\mathbb{Z}_q[X]$. It is a generalization of the Fast Fourier Transform (FFT). Let we have a polynomial f with degree n, where $f = \sum_{i=0}^{n-1} f_i X^i$ and $f_i \in \mathbb{Z}_q$ and ω_n be the n-th primitive root of unity such that $\omega_n^n = 1 \mod q$. Then the forward NTT can be defined by $\hat{f} = NTT(f)$, such that $\hat{f}_i = \sum_{j=0}^{n-1} f_j \omega_n^{ij} \mod q$. Similarly, the inverse NTT can be computed by $f = INTT(\hat{f})$, such that $f_i = n^{-1} \sum_{j=0}^{n-1} \hat{f}_j \omega_n^{-ij} \mod q$. Based on these definitions, an NTT-based polynomial multiplication between a and b can be performed such that $a.b = INTT(NTT(a) \circ NTT(b))$.

The NTT-based multiplication computes on convolution, which transforms the input polynomials of length n to $2n$ with zeros padding, resulting in more computation time. Therefore, to avoid applying the NTT of length $2n$ with n zero padding of inputs, a negative wrapped convolution (NWC) [63] method is introduced at the cost of pre-processing of NTT and post-processing of INTT. Let us say $\psi = \sqrt{\omega_n}$; it is a primitive $2n$-th root of unity. The pre-processing cost includes the multiplication between the coefficients of the input polynomials and ψ^i. In contrast, the post-processing cost includes the multiplication between the coefficients of the output polynomials and ψ^{-i}.

The CooleyTukey (CT) and Gentleman-Sande (GS) butterfly configurations are the most frequently employed in literature on NTT-based implementations. Using these configurations reduces the bit-reverse operation in NTT, which is the bit-wise reversal

of the binary representation of the coefficient index. For more insight details at the algorithmic level, interested readers are referred to [64], and to follow some recent NTT-based hardware accelerators, readers are referred to [65, 66, 67].

The last column of Table 1 shows the hash methods implemented in different PQC algorithms for various purposes, such as binomial sampling. The PQC schemes of column one of Table 1 contributed in rounds two and three of the NIST competition process and mainly depended on variants of the SHA2, SHA3, and SHAKE-128/256 hash functions. This thesis is not describing the inner structures of these hash functions; however, the only objective is to highlight the complexity of the PQC schemes when realized as hardware accelerators. NIST standardizes the most recent SHA3 and its variants in [68] and is mainly used in all PQC schemes of Table 1, including the NIST selected CRYSTALS-Kyber and CRYSTALS-Dilithium algorithms to be standardized in the near future.

Also, the computation time of polynomial multiplications and hash operations of the PQC algorithms depends on their security parameters. NIST has defined five security levels (1 to 5) for investigating PQC algorithms. Security levels 1, 3, and 5 are equivalent to AES-128, AES-192, and AES-256 bit key search. The remaining security levels (2 and 4) are equivalent to SHA-256/SHA3-256 and SHA-384/SHA3-384 bit collision search. Implementing all security levels in one hardware design requires large memory utilization. In other words, despite the polynomial multiplications and hash operations, large memory utilization is also the key characteristic of the PQC algorithms when demonstrated as hardware accelerators. Therefore, in [69], I have evaluated the memory, hash, and multiplier building blocks of PQC algorithms of Table 1, where I have targeted the highest security parameters shown in Fig. 8. The detailed outcomes appear in [69] while the major findings are repeated in Fig. 9.

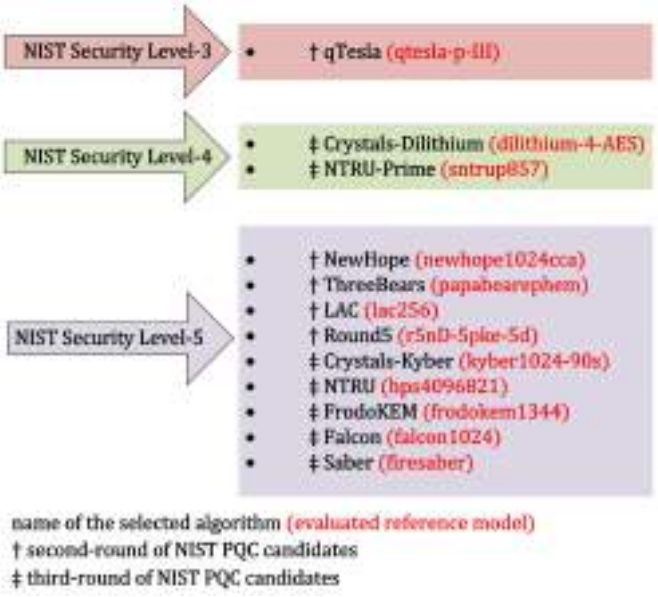


Figure 8: Selected lattice-based PQC algorithms and the corresponding implementations utilized in this study. Red-colored text inside the parenthesis specifies selected security parameters.

To evaluate the area and power results in Fig. 9, I have added the area and power of

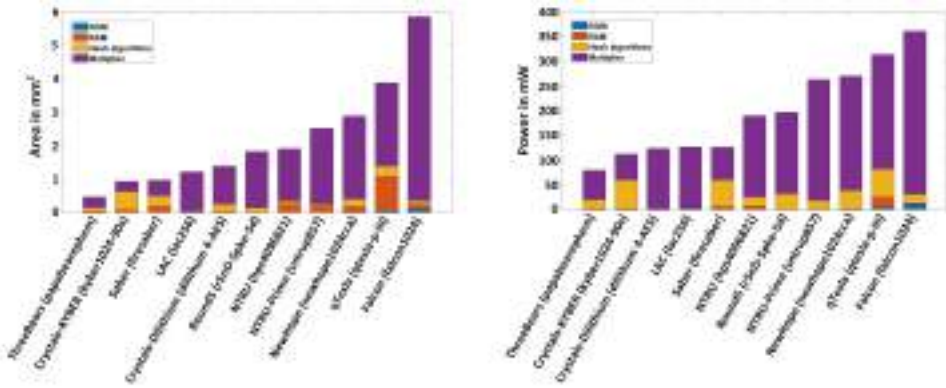


Figure 9: Total area and power of the studied NIST lattice-based PQC algorithms on 65nm process technology.

memory, multiplier, and hash operations together. The area for memory is investigated in terms of read-only (ROM) and read-access (RAM) memories. The instances of the required ROM and RAMs are generated using a commercial memory compiler from a partner foundry. For the corresponding PQC algorithm of Table 1, I have implemented Algorithms 1 to 4 for polynomials multiplication. The hash algorithms of column three of Table 1 are also implemented. For details about polynomials' input and output lengths, required memory sizes for ROM and RAM, and input and output of hash functions, readers can follow [69]. Consequently, the CRYSTALS-Kyber algorithm utilizes lower resources and consumes less power than other NIST round three candidates. On the other hand, the CRYSTALS-Dilithium takes higher resources but consumes lower power than the SABER algorithm, as shown in Fig. 9. Therefore, due to its simple mathematical structure, I selected SABER for further investigations in this thesis. Hence, the following text overviews SABER, including its building blocks.

2.3 SABER PQC KEM Protocol

SABER [21] provides security against Chosen-Ciphertext Attacks (IND-CCA), and its security hardness depends upon solving the module variant of the LWR problem (mod-LWR) [48]. A mod-LWR sample is defined by $(a, b = \lfloor \frac{p}{q}(a^T s) \rceil) \in \mathcal{R}^{l \times 1} \times \mathcal{R}_p$. Here, a denotes a vector of randomly generated polynomials in $\mathcal{R}_q$, s determines a secret vector of polynomials in $\mathcal{R}_q$ whose polynomial coefficients are sampled from a binomial distribution, and the modulus p is less than q. The decisional variant of the problem is about finding a way to distinguish between two types of samples (mod-LWR and uniformly random) in $\mathcal{R}_q^{l \times 1} \times \mathcal{R}_p$. Moreover, SABER uses the Mod-LWR problem with p and q being power-of-two to construct a public-key encryption (PKE) scheme that is secure against Chosen Plaintext Attacks (IND-CPA). The PKE scheme supports the following cryptographic operations: (i) generation of a pair of public and private keys (PKE.KEYGEN), (ii) encryption (PKE.ENC), and (iii) decryption (PKE.DEC). The related algorithms to execute these operations are described in algorithms 5, 6, and 7. Similarly, for the KEM operations, the following are supported: (i) generation of a pair of public and private keys (KEM.KEYGEN), (ii) encapsulation (KEM.ENCAPS), and (iii) decapsulation (KEM.DECAPS). The algorithms for these operations are described in algorithms 8, 9, and 10.

Algorithm 5: SABER.PKE.KEYGEN() [30]

Input: SABER Parameter Lengths
Output: $pk \Leftarrow (seed_A, b), sk \Leftarrow (s)$

$seed_A \Leftarrow \mathcal{U}(\{0,1\}^{256})$
$A \Leftarrow gen(seed_A) \in \mathcal{R}_q^{l \times l}$
$r \Leftarrow \mathcal{U}(\{0,1\}^{256})$
$s \Leftarrow \beta_\mu(\mathcal{R}_q^{l \times l};\ r)$
$b \Leftarrow ((A^T s + h)\ mod\ q) \gg (\epsilon_q - \epsilon_p) \in \mathcal{R}_p^{l \times l}$
return $pk \Leftarrow (seed_A, b), sk \Leftarrow (s)$

Algorithm 6: SABER.PKE.ENC() [30]

Input: $pk \Leftarrow (seed_A, b), m \in R_2;\ r)$
Output: $c \Leftarrow (c_m, b')$

$A \Leftarrow gen(seed_A) \in \mathcal{R}_q^{l \times l}$
if $r\ is\ not\ specified$ **then**
  $r \Leftarrow \mathcal{U}(\{0,1\}^{256})$
$s' \Leftarrow \beta_\mu(\mathcal{R}_q^{l \times l};\ r)$
$b' \Leftarrow ((As' + h)\ mod\ q) \gg (\epsilon_q - \epsilon_p) \in \mathcal{R}_p^{l \times 1}$
$v' \Leftarrow b^T(s'\ mod\ p) \in R_p$
$c_m \Leftarrow (v' + h_1 - 2^{\epsilon_p - 1} m\ mod\ p) \gg (\epsilon_p - \epsilon_T) \in \mathcal{R}_T$
return $c \Leftarrow (c_m, b')$

Algorithm 7: SABER.PKE.DEC() [30]

Input: $sk \Leftarrow s, c \Leftarrow (c_m, b')$
Output: m'

$v \Leftarrow b'^T(s\ mod\ p) \in \mathcal{R}_p$
$m' \Leftarrow ((v - 2^{\epsilon_p - \epsilon_T} c_m + h_2)\ mod\ p) \gg (\epsilon_p - 1) \in \mathcal{R}_2$
return $c \Leftarrow (c_m, b')$

Algorithm 8: SABER.KEM.KEYGEN() [30]

Input: SABER.PKE.KEYGEN()
Output: $pk \Leftarrow (seed_A, b), sk \Leftarrow (s, z, pkh)$

$pk \Leftarrow (seed_A, b)$
$pkh \Leftarrow \mathcal{F}(pk)$
$z \Leftarrow \mathcal{U}(\{0,1\}^{256})$
return $pk \Leftarrow (seed_A, b), sk \Leftarrow (s, z, pkh)$

Algorithm 9: SABER.KEM.ENCAPS() [30]

Input: $pk \Leftarrow (seed_A, b)$
Output: c, K

$m \Leftarrow \mathcal{U}(\{0,1\}^{256})$
$(\hat{K}, r) \Leftarrow \mathcal{G}(\mathcal{F}(pk), m)$
$c \Leftarrow$ SABER.PKE.ENC($pk, m;\ r$)
$K \Leftarrow \mathcal{F}(\hat{K}, c)$
return $pk \Leftarrow (seed_A, b), sk \Leftarrow (s, z, pkh)$

Algorithm 10: SABER.KEM.DECAPS() [30]

Input: $sk \Leftarrow (s, z, pkh), pk \Leftarrow (seed_A, b), c$
Output: K

$m' \Leftarrow$ SABER.PKE.DEC(s, c)
$(\hat{K}', r') \Leftarrow \mathcal{G}(pkh, m')$
$c' \Leftarrow$ SABER.PKE.ENC$(pk, m'; r')$
if $c = c'$ **then**
    $K \Leftarrow \mathcal{H}(\hat{K}', c)$
else
    $K \Leftarrow \mathcal{H}(z, c)$
return $pk \Leftarrow (seed_A, b), sk \Leftarrow (s, z, pkh)$

In algorithms 5 to 10, the coefficients of the secret vectors s and s' are sampled from a centered binomial distribution $\beta_\mu(\mathcal{R}_q^{l \times 1})$ with a parameter μ, where $\mu < p$. The hash functions used in the SABER protocol are determined by $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{H}$. $\mathcal{F}$ and $\mathcal{H}$ are implemented using SHA3-256, while $\mathcal{G}$ is implemented using SHA3-512. A variant of SABER, $\mathcal{U}$, samples the secret vectors s and s' from a centered uniform distribution instead of the binomial distribution. This makes the secret generation more efficient, as sampling from $\mathcal{U}$ is simpler than sampling from β_μ. The constant polynomials used in SABER are h_1 and h_2. The implementation constants l, ϵ_q, ϵ_p, and ϵ_T have values of 3, 13, 10, and 4 for SABER. Different operations of SABER are further described in the following points.

- PKE.KEYGEN begins by randomly generating a seed that defines an $l \times l$ matrix A comprising l^2 polynomials in $\mathcal{R}_q$. A function gen of Algorithm 5 is used to generating a matrix from the seed based on SHAKE-128. A secret vector s of polynomials is also generated. These polynomials are sampled from a centered binomial distribution. The generated public key contains a matrix seed and rounded product $A^T s$, while the secret key contains a secret vector s. KEM.KEYGEN follows the same steps as used for the PKE.KEYGEN, except that it appends a secret key with a hash of the public key and a randomly generated string z.

- The PKE.ENC operation consists of generating a new secret s' and adding a message to the inner product between the public key and the new secret s'. This forms the first part of the ciphertext while the second part contains the rounded product As'. The KEM.ENCAPS operation starts by randomly generating a message m and obtaining from that the public key. The ciphertext c contains the encrypted message and a value achieved from the message and public key.

- PKE.DEC requires the secret key s to extract the original message from the inner product between the public and secret keys. It is the counterpart to PKE.ENC. KEM.DECAPS re-encrypts the obtained message with the randomness associated with it and checks whether the ciphertext corresponds to the one received.

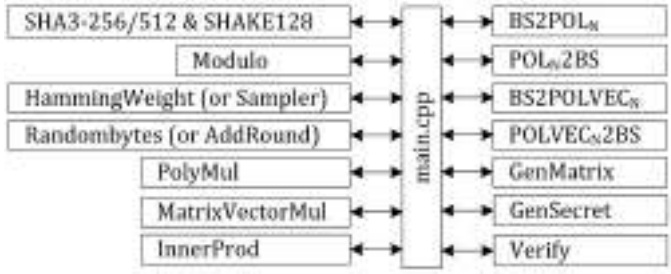
SABER offers three variants to target different security levels: LightSABER, SABER, and Fire SABER. The supported parameters to implement variants of SABER are given in Table 2. The values of the implementation constants used in algorithms 5 to 10 can be chosen from the SABER reference document [21]. Table 2 shows that, for the same

parameter size, three variants of SABER differ only in the secret key size. Moreover, the required building blocks are shown in Fig 10 to implement three variants of SABER.

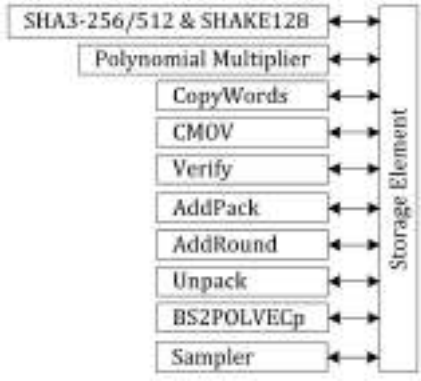
Table 2: Security parameters of SABER for PKE and KEM operations (taken from [21])

SL_i	Public-key (B)	Secret-key (B)	Cipher-text (B)
Light SABER (PKE & KEM): $l = 2$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^3$, $\mu = 10$			
SL_1	672	832 (for PKE) & 1568 (for KEM)	736
SABER (PKE & KEM): $l = 3$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^4$, $\mu = 8$			
SL_3	992	1248 (for PKE) & 2304 (for KEM)	1088
Fire SABER (PKE & KEM): $l = 4$, $n = 256$, $q = 2^{13}$, $p = 2^{10}$, $T = 2^6$, $\mu = 6$			
SL_5	1312	1664 (for PKE) & 3040 (for KEM)	1472

SL_i: Security levels, **SL_1:** equivalent to AES-128, **SL_3:** equivalent to AES-192 **SL_5:** equivalent to AES-256.



(a) Concerning SABER specification document [21].



(b) Regarding FPGA-based hardware design of [30].

Figure 10: SABER building blocks.

Fig. 10(a) provides the SABER building blocks concerning its specification document of [21], where all the blocks are implemented in C/C++ and called in a main file to execute the sequence of SABER operations. The building blocks, shown in Fig. 10(b), are regarding FPGA-based reference SABER implementation of [30], where blocks on

the left are the arithmetic and logical units and these blocks shares storage element amongst them to keep intermediate and the final results after the computations.

The blocks of Fig. 10(a) and Fig. 10(b) operate identically with some additional logic. For example, the SHA3-256/512 & SHAKE128 implemented as a wrapper to operate variants of SHA3, GenMatrix and GenSecret blocks of Fig. 10(b). Similarly, a polynomial multiplier is also implemented as a wrapper in Fig. 10(b), and it implements PolyMul, MatrixVectorMul, and InnerProd blocks of Fig. 10(a). The HammingWeight and Randombytes blocks of Fig. 10(a) correspond to the sampler and AddRound blocks of Fig. 10(b). Moreover, the BS2POL$_N$ and POL$_N$2BS blocks of Fig. 10(a) correspond to Unpack and AddPack units. The additional CMOV and CopyWords blocks in Fig. 10(b) need to compute matrix transpose by shifting rows with the columns and vice versa. In short, the strategies to implement these building blocks are described in the text below.

SABER requires several hash functions such as variants of SHA3 (256/512) and an extended output function, i.e., SHAKE128, for different purposes such as binomial sampling. All these functions are standardized in FIPS-202 [68]. SHA3-256 takes the input byte string from the byte array of length l and generates the output byte string of length 32. Similarly, SHA3-512 takes the input byte string from the byte array of length l and generates the output byte string of length 64. SHAKE128 receives the input byte string from the byte array of length l and generates the output byte string of length L. The execution of all these hash functions is based on a KECCAK sponge function [68] to compute the permutations. The building blocks of KECCAK are $theta$, pi, rho, chi, and $iota$. To understand KECCAK building blocks, interested readers are referred to follow the KECCAK specification document [68].

SABER involves polynomial-to-polynomial multiplications and matrix-to-vector multiplication. In polynomial-to-polynomial multiplications, the corresponding inputs and produced output are in polynomials. In matrix-to-vector multiplication, the first input to the multiplier is a matrix that belongs to $R_q^{l\times 1}$ while the second input is a vector v, and it returns the products in a vector. Several approaches exist in the literature to operate these (polynomial-to-polynomial and matrix-to-vector) multiplications. These approaches include schoolbook [70, 71], Karatsuba [69, 72, 73], Toom-Cook [74, 75], NTT [65], Booth [62], etc. Since SABER uses a power-of-two moduli $p = 2^{10}$ and $q = 2^{13}$ [30]; therefore NTT-based multiplication could be applied but has no benefit and even, it worse the performance of the SABER [76] – while the remaining methods can be applied to perform polynomial coefficient multiplications. The mathematical structures of these polynomial multiplication methods are already described in Section 2.2.

As mentioned before, SABER uses a power-of-two moduli $p = 2^{10}$ and $q = 2^{13}$ [21], therefore algorithms 1 and 2 can be applied in their present form to perform SABER polynomial coefficient multiplication with clock cycles overhead compared to SABER-specific SBM multipliers of [30, 34, 36]. More precisely, SABER has 256 public and secret polynomial coefficients with a length of 13 bits and 4 bits each, respectively. Therefore, to perform multiplication over a 13-bit public polynomial coefficient with a 4-bit secret polynomial coefficient, algorithms 1 and 2 take 4 and 2 clock cycles as these multipliers are not only specific to SABER but also feasible for other cryptographic algorithms such as ECC. Using SBM multiplication designs of [30, 34, 36], one polynomial coefficient can be multiplied in one clock cycle; hence for 256 coefficients, only 256 cycles are required. In the case of other multipliers of algorithms 3 and 4, some special considerations are needed to multiply SABER polynomial coefficients so that readers can follow [31] (for Karatsuba implementation specific to SABER) and [32] (for Toom-Cook implementation specific to SABER).

The remaining building blocks are CopyWords, Constant-time Move (CMOV), Verify, AddPack, AddRound, Unpack, and BS2POLVECp. These have a low computational complexity of $\mathcal{O}(n)$ [30]. The Copy-Words block is used to copy data-block from one location to another. In SABER, this block is only utilized during the key generation to compute the transpose of a matrix. The Verify block is a key component of cryptographic protocols that aim to ensure the authenticity of the computed/generated data. In the decapsulation operation, the received ciphertext must be compared with the re-encrypted ciphertext. If they match, the result of this comparison is stored in a flag register, which is used by the CMOV instruction to either copy the decrypted session key to a specified location or a pseudo-random string.

The AddPack block performs coefficient-wise addition between a constant value and a message. This operation is used in various cryptographic algorithms to transform the message in a controlled and predictable way. Adding a constant value transforms the message into a new value that is less predictable to an attacker. In addition, AddPack is also responsible for packing the result bits into a byte string; the functions for this transformation are described in the SABER specification document [21]. The AddRound block performs two tasks: coefficient-wise addition and coefficient-wise rounding. The coefficient-wise addition involves adding a constant value h to each coefficient of the input data. This helps to mix the input data and add randomness to the result. The coefficient-wise rounding involves rounding each coefficient of the result (obtained after addition) to the nearest integer value. This helps reduce the number of possible output values and therefore increases the cipher's security. Overall, the AddRound block plays an important role in designing secure cryptographic ciphers by adding randomness and reducing the number of possible outputs, making it harder for an attacker to predict the output or reverse the cipher. The conversion from byte into bit strings is the responsibility of Unpack unit; the functions for this transformation are described in the SABER specification document [21]. A BS2POLVECp block transforms the byte strings into polynomial vectors. For more specific details, and corresponding algorithms for different transformations, readers can follow the SABER specification document [21].

2.4 Implementation Platforms and Hardware Accelerators

This section summarizes different implementations of SABER on various platforms, including RISC-V processors [77], general-purpose-processors (GPUs) [78], ARM platforms [79, 80, 81], software implementations with side-channel protection [82, 83], a side-channel protected hardware implementation [84], an embedded microcontroller [85], FPGAs [76, 30, 86] and ASICs [34, 36, 31, 32, 33]. Below I am describing these accelerator architectures along with their limitations and advantages.

A RISC-V architecture is modified in [77] to integrate a tightly coupled hardware accelerator for performance improvement of lattice-based PQC. The aim was to reuse the RISC-V processor resources to reduce memory access efficiently, significantly increasing performance and keeping low area overhead. This was achieved with three steps: (i) initially, the authors proposed hardware accelerators (one for NewHope, CRYSTALS-Kyber, and SABER) and integrated them into the RISC-V pipeline design, (ii) then they extended the RISC-V Instruction Set Architecture (ISA) to include twenty-nine additional instructions to execute operations for lattice-based cryptography efficiently, and (iii) finally the authors implemented the extended RISC-V architecture on ASIC and FPGA platforms. Compared to only software implementation on RISC-V, the co-design of [77] shows a speedup of 11.4, 9.6, and 2.7 for NewHope, Crystals-Kyber, and SABER. Compared to ASIC, the consumed energy reduces by 9.5, 7.7, and 2.1

times for NewHope, Crystals-Kyber, and SABER.

The experiments in [78] reveal that the dot-product instruction, introduced by NVIDIA in modern GPU architectures, can effectively accelerate matrix multiplication and polynomial convolution operations commonly found in post-quantum lattice-based cryptographic schemes.

NIST has recommended ARM microcontrollers as an important benchmarking platform for its PQC standardization process, and hence several implementations reported performance improvements [79, 80, 81]. The use of polynomial multiplication styles (such as Toom-Cook, Karatsuba, and NTT) on embedded vector architectures is explored in [79] where implementations were performed on Arm Cortex-M4 CPU as well as the newer Cortex-M55 processor architectures. Through careful register management and instruction scheduling, they show a significant performance improvement (3-5 times faster) compared to highly optimized implementations on the Cortex-M4 architecture while maintaining a low area and energy profile suitable for use in the embedded market. The focus on low area and energy consumption is particularly important for embedded systems, which often have limited resources and power constraints. The design space of SABER on Cortex-M3 and Cortex-M4 processors is explored in [80]. Postquantum cryptography schemes' speed and memory optimizations are crucial for practical deployment in resource-constrained microcontrollers, specifically to ensure secure communication in IoT-related applications. This is addressed by the authors of [81] where they have leveraged digital signal processing instructions and efficient memory access to optimize the polynomial multiplication operation of SABER on the Cortex-M4 processor, which is a critical part of the scheme. Additionally, they have employed the Karatsuba algorithm and just-in-time strategy to generate the module lattice's public matrix, which helps to reduce the memory footprint.

SABER is very efficient for masking because of the two specific design preferences: (i) power-of-two moduli and (ii) limited noise sampling with LWR. Therefore, in [82], the SABER design includes a novel primitive for masked logical shifting on arithmetic shares, and adapts an existing masked binomial sampler to provide side-channel resistant implementation on the ARM Cortex-M4 microcontroller. In [83], authors claimed to have the first masked software-hardware co-design for PQC with SABER and CRYSTALS-Kyber algorithms (as a case study) where they devise a masked ciphertext compression protocol for non-power-of-two moduli PQC schemes. To accelerate the performance of the linear operations such as a multiplier, they implement a generic NTT-based multiplier suitable for schemes those not allowing the NTT operations (such as SABER). For the required non-linear operations, they have developed masked hardware accelerators that allow secure instructions execution using RISC-V instruction set extensions.

An efficient implementation of SABER on ESP32[2] microcontroller is implemented in [85], where a big integer RSA-based co-processor is utilized for computing the polynomial multiplications of SABER.

At register-transfer-level (RTL), a SABER hardware accelerator is designed to be a fast co-processor for lattice-based cryptography in [30]. The co-processor is optimized for polynomial multiplication and includes various design decisions and architectural optimizations to reduce overall cycle counts and improve resource utilization. For key generation, encapsulation, and decapsulation operations, the accelerator requires 5453, 6618, and 8034 cycles for a module dimension of 3 (which provides security similar to AES-192). It runs at a maximum frequency of $250MHz$ on a Xilinx UltraScale+ FPGA

[2]ESP32 is an embedded microcontroller explicitly designed for an IoT environment with WiFi and bluetooth support. Its manual can be accessed at [87].

and consumes 23686 look-up-tables (LUTs), 9805 flip-flops (FFs), and 2 BRAM tiles.

An NTT-based polynomial multiplier has been shared between two quantum-resistant cryptographic protocols, i.e., SABER and Dilithium, in [76]. The authors estimate that this can lead to a 4% increase in LUT count for existing Dilithium implementations. Their NTT-based multiplier has a minor trade-off of producing inexact results in some limited inputs, but the authors conduct a thorough analysis and prove that the probability of these events occurring is near zero and does not affect the security of the implementation. They also implement the NTT multiplier in hardware and obtain a design with competitive performance/area trade-offs. The implementation has a latency of 519 cycles and consumes 2012 LUTs and 331 FFs when implemented on an Artix-7 FPGA. A shuffling-based method is (also) offered to provide side-channel protection with low overhead during polynomial multiplication. Furthermore, the side-channel security of the design is evaluated on a Sakura-X FPGA board. It is important to note that only the NTT-based multiplication core is described in [76] without providing the complete implementation of the SABER and CRYSTALS-Dilithium PQC algorithms.

Design and implementation of a domain-specific co-processor to accelerate the performance of SABER are considered in [86] where authors run the building blocks on an ARM core and the most computationally intensive operations are offloaded to the co-processor, leveraging the idea of distributed computing at the micro-architectural level and incorporating algorithmic optimizations. The results show that the co-processor provides approximately a 6 times speedup compared to optimized software implementation, with a small area cost. The design was demonstrated on a Zynq-7000 ARM/FPGA System-on-Chip (SoC) platform. Hence, the co-processor accelerators of [30] and [86] demonstrate the potential for hardware acceleration of PQC algorithms and highlight the benefits of a hardware-software co-design approach for efficient and compact implementations.

Another co-processor PQC accelerator is presented in [88]. The authors have implemented three lattice-based PQC algorithms (FrodoKEM, Round5, and SABER) on an Ultrascale+ FPGA using a software/hardware codesign approach.

Using various optimization techniques such as pipelining, resource sharing, and efficient memory arrangements, a design space exploration of SABER is presented in [34]. These optimizations (after synthesis) resulted in a clock frequency of $1GHz$. However, when the full-optimized SABER architecture was fabricated on a 65nm process technology, the maximum operating frequency was only $715MHz$ (details are described in [36]). This decrease in operating frequency is common when transitioning from simulation to actual physical implementation due to manufacturing process variability, power constraints, and limitations in on-chip interconnect. Notably, the decrease in operating frequency does not necessarily mean that the optimized architecture is ineffective. In [36], a proof-of-concept of the optimized SABER architecture is demonstrated, showing the adopted approach's viability.

The Energy-efficient crypto processor architecture of [31] for SABER employs the hierarchical Karatsuba method to optimize the processor's energy consumption. Implementing the processor on 40nm process technology reveals an area consumption of $0.38mm^2$ and a maximum frequency of $400MHz$. A Toom-Cook multiplier with a striding of 4 for SABER 256-degree polynomial multiplications is also a significant contribution to the field; this optimization approach is investigated on 65nm process technology, and the relevant details are described in [32]. These optimizations (of the Karatsuba and Toom-Cook) help to make the processor more efficient and practical for real-world applications.

It is impressive that a flexible crypto processor has been fabricated in [33] for several hard mathematical problems using a 28nm process technology. The support for various cryptographic algorithms such as SABER, NTRU, CRYSTALS-Dilithium, Rainbow, CRYSTALS-Kyber, and McEliece makes the design very versatile and suitable for a wide range of cryptographic applications. The fact that it can operate at a maximum frequency of 500MHz while consuming low power at a 0.9V supply voltage is also noteworthy. The large chip size of 3.6mm^2 is used because it supports multiple algorithms. These features make the design an excellent choice for implementing secure cryptographic operations for various applications.

Most FPGA and ASIC SABER hardware accelerators execute the polynomial multiplications based on the sign-magnitude format. Recently, in [89], SABER multiplication design was presented where authors emphasized two's complement representation system to multiply SABER polynomial coefficients.

In summary, hash and polynomial multiplications are the critical building blocks of implementing lattice-based cryptography. In addition, large memory size is also a requirement of the lattice-based PQC algorithms to keep the initial, intermediate, and final results. The existing hardware accelerators are mostly obtained after the polynomial multiplications' optimizations. Indeed, NIST is investigating the security aspects of PQC algorithms primarily at the software level only, and the performance of PQC algorithms on different platforms is a crucial factor to consider. The choice of platform for implementing PQC algorithms is not straightforward and depends on the system's specific requirements. For example, pure software implementations of PQC algorithms are often more flexible and can be easily updated, but they may not provide the same level of security as hardware implementations. On the other hand, hardware implementations, such as those implemented on FPGAs or ASICs, can provide a higher level of security, but they are often more expensive and may not be as flexible as software implementations. The combined software-hardware approach provides a balance between performance, security, and cost. It allows for the benefits of hardware-based security to be combined with the flexibility of software implementations.

3 A Generator of Large Integer Polynomial Multipliers

This chapter focuses on the open-source polynomial multiplier generator that I have developed. The chapter highlights the critical features and describes the multiplier generator architecture in sections 3.1 and 3.2, respectively. Section 3.3 provides the implementation results in various design parameters, including area, latency, clock frequency, and power, of the generated multipliers on ASIC and FPGA platforms. After providing the values of these design parameters, more than one design parameters are utilized simultaneously to define the figures-of-merit (FoM) and design trade-offs to evaluate the performance of the multiplier generator, the subsequent details are shown in Section 3.4. Section 3.5 compares the generated multipliers to existing multiplier accelerators.

Indeed, cryptographic systems rely on arithmetic and logical operations for secure communication and data exchange. Multiplication is often considered the most computationally intensive operation in cryptographic circuits, and it can become a bottleneck for efficient implementation of cryptographic schemes [90, 91, 92, 93, 94]. This is especially true for public-key cryptosystems like RSA and ECC [95, 61], which require efficient polynomial multiplications. Post-quantum cryptography algorithms also require efficient polynomial multiplications. Additionally, fully homomorphic encryption enables multi-party communications on the cloud and requires large integer polynomial multipliers [96]. Therefore, there is a need for efficient polynomial multipliers to ensure the security and efficiency of cryptographic systems.

Multiple multiplication techniques are available in the literature for multiplying polynomial coefficients, and each technique has its own advantages and disadvantages. Some commonly used techniques include the traditional SBM, Karatsuba, Toom-Cook, Montgomery, Booth, and NTT. These techniques can also be utilized in a digitized form, where the polynomial is split into smaller parts to reduce the complexity of the multiplication at the expense of additional control logic to drive and unite the small products. The choice of multiplication technique depends on the application's specific requirements and the target hardware platform. The reference implementations of various PQC algorithms, available in [61], suggest using different multiplication techniques for different algorithms. For example, (i) SBM is used in FrodoKEM and NTRU-Prime, (ii) Karatsuba and Toom-Cook are used in SABER and NTRU, (iii) an NTT is used in CRYSTALS-Kyber, and (iv) Montgomery and SBM are used in Falcon.

Examples of recent works employing non-digitized and digitized polynomial multiplication methods are given in [90, 93, 92, 65, 97, 98, 99, 100, 101, 102, 103, 104], and [105, 91, 106, 94], respectively. Even if several implementations of different multiplication approaches are available in the literature, these dedicated implementations are optimized for a specific operand size and a given target (e.g., high speed or low area or low power). The matter is that this trade-off space exploration is difficult to drive without automation. Therefore, there is a real need for access to (many) multiplications approaches where designers can select an appropriate multiplier architecture combined with their choice of operand lengths.

3.1 Supported Features

Concerning the gap mentioned above and the requirement for automation, this section describes the features of the proposed generator of several polynomial multipliers, named **TTech-LIB**. TTech-LIB is an open-source repository [107] of several large

integer polynomial multipliers whose initial results on Artix-7 FPGA and 65nm ASIC platforms appeared in [108] and more detailed results on Artix-7 FPGA, 15nm, and 65nm ASIC technologies are published in [109]. The critical features of the multiplier generator are as follows:

(i) **Flexibility:** The developed multiplier generator supports five multiplication approaches: (i) SBM, (ii) Booth, (iii) 2-way Karatsuba, (iv) 3-way Toom-Cook, and (v) 4-way Toom-Cook.

(ii) **Pipelining:** The proposed generator supports pipelining to reduce the critical paths (which therefore improves the clock frequency) of the multiplier circuits.

(iii) **Digitizing:** The developed multiplier generator offers a parameterized digit-serial multiplier wrapper to multiply polynomial coefficients. By default, the wrapper instantiates a singular SBM multiplier. It can be replaced by any other multiplier method (from the proposed TTech-LIB or otherwise) as the input/output interfaces are compatible.

(iv) **Agnostic RTL:** The codes generated by the multiplier generator tool are technology- and device-agnostic, thus being synthesizable for both FPGA and ASIC platforms. ASIC designers can additionally generate synthesis scripts for one of two synthesis tools, either Synopsis Design Compiler or Cadence Genus. The user is not bound to generate only a single architecture at a time; the generator can produce multiple solutions if asked, which will appear as separate Verilog (.v) files.

The generated multipliers by TTech-LIB take two m-bit polynomials (a and b) as input and result in an output of polynomial (c) with $2 \times m$ bit. The algorithmic details of the supported multipliers (SBM, Booth, 2-way Karatsuba, 3-way Toom-Cook, and 4-way Toom-Cook) are already described in Section 2.2. Similarly, a supported digit-serial wrapper takes two m-bit polynomials $a(x)$ and $b(x)$ as input and produces $c(x)$ as an output. The digits of polynomial $b(x)$ are created with different lengths, which depend on the user choice as follows: $d = \frac{m}{n}$, where d denotes the total number of digits, m is the length of $b(x)$, and users can choose n that determines the length of each digit. After digitization, the multiplication of each digit is computed serially with the polynomial $a(x)$. Finally, the resultant polynomial $c(x)$ is constructed using shift and add operations. For one-digit serial multiplication, n cycles are needed. Thus, the total digits are d, and the total clock cycles for one m-bit polynomial multiplication with n bit digit take $\lceil d \times n \rceil$. It is important to note that the respective users/designers can select any multiplication method inside the proposed digit-serial wrapper. For experiments in this work, an SBM multiplication method is used.

Since the proposed library is aimed at large polynomials, the 2-way Karatsuba, 3-way Toom-Cook, 4-way Toom-Cook, and Booth multipliers, generated in the proposed TTech-LIB, actually implement the SBM strategy. The implementation of SBM, Booth and our digit-serial wrapper produces resultant polynomial $c(x)$ serially while 2-way Karatsuba, 3-way Toom-Cook and 4-way Toom-Cook multipliers use a *hybrid* approach (as they utilize a combination of both serial and parallel execution of SBM for the computations).

The proposed generator architecture in TTech-LIB provides only the polynomial multiplication without modular reduction. For modular reduction over prime and binary elliptic curves, NIST-specified reduction routines [110] can be employed after the

multiplier circuit generated by TTech-LIB. Similarly, in the case of PQC algorithms, an additional m-bit subtractor is required after the multiplier circuit for modular reduction when polynomial coefficients need to multiply iteratively. The size of the polynomial coefficient (m) depends on the specific PQC algorithm being used.

3.2 Proposed Multiplier Generator Architecture

Fig. 11 shows the architecture of the multiplier generator that supports TTech-LIB. It shows that the generator engine takes inputs from a simple XML file structured around a few keywords. The descriptions are given below.

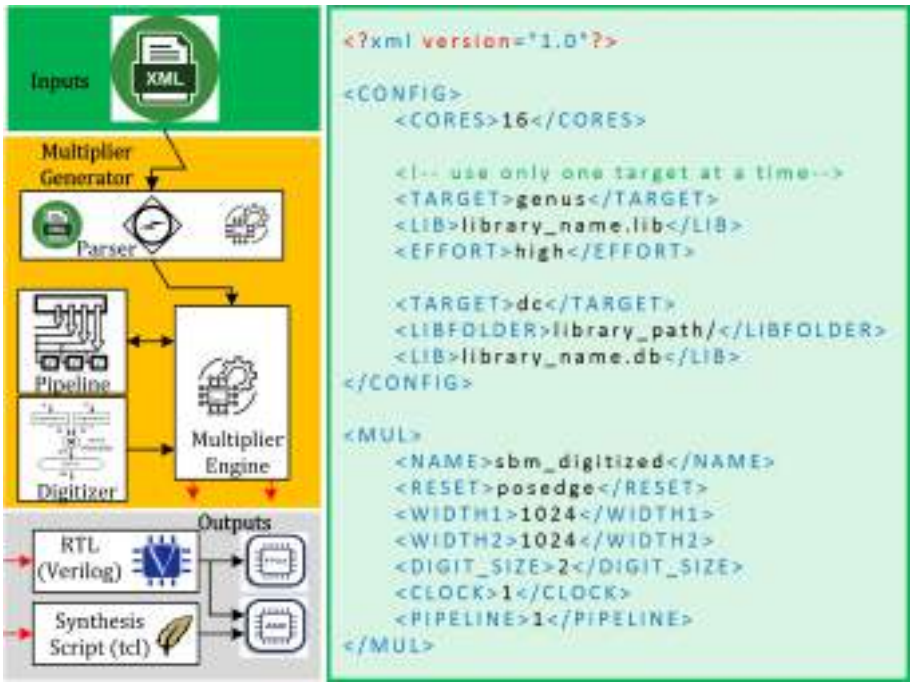


Figure 11: Structure of the proposed multiplier generator. Green, orange, and gray portions identify the input parameters, multiplier generator, and generated scripts and RTL files as output.

The "target", "lib", and "effort" keywords are used to generate script files for the ASIC platform, where the "target" keyword specifies the name of the commercial synthesis tool (genus or dc), the "lib" keyword specifies the used library for the targeted synthesis tools, and "effort" keyword determines the level of synthesis effort and can take one of three values: low, medium, and high.

The "multiplier" keyword specifies the name of the multiplication method to generate and TTech-LIB takes the following names of the multiplication methods as input: (i) schoolbook (for SBM), (ii) booth (for Booth), (iii) 2_way_karatsuba (for Karatsuba), (iv) 3_way_toom_cook (for Toom-Cook with splitting levels of three), (v) 4_way_toom_cook (for Toom-Cook with splitting levels of four), and (vi) sbm_digitized (for digitized multiplication).

The "reset" keyword determines the reset behavior (rising or falling edge of the clock) for the generated multiplier circuit. The "width1" and "width2" keywords provide

the length of the polynomials as input operands to the multiplier. The "clock" keyword defines the timing constraint. Users can use "digit_size" and "pipeline" keywords to target different digit sizes and pipeline stages based on their application needs. To generate non-pipelined multiplication circuits, the value for "pipeline" must be set to one. The multiplier generator (orange portion in Fig. 11) takes all the parameters as input using the parser and generates the corresponding Verilog HDL and script files in respective directories. The generated code is pure RTL, therefore platform and technology agnostic.

The structure of the proposed TTech-LIB is relatively simple and includes five directories, i.e., (i) bin, (ii) run, (iii) src, (iv) synth, and (v) vlog. As the name specifies, bin and run directories contain the essential files to compile and execute the project. The src directory contains the library source files. The synth and vlog directories keep the generated scripts and Verilog files, respectively. All the multipliers use an identical interface, meaning the inputs are always clk, rst, a, and b while the output is always c.

The complete project files (written in C++) are freely available to everyone on a GitHub repository [107]. To compile and execute the project source files, the current directories should be /TMlib/src and /TMlib/run, respectively. Moreover, source compile.sh and ../bin/libgen.exe commands can use to compile and run the source files.

3.3 Implementation Results

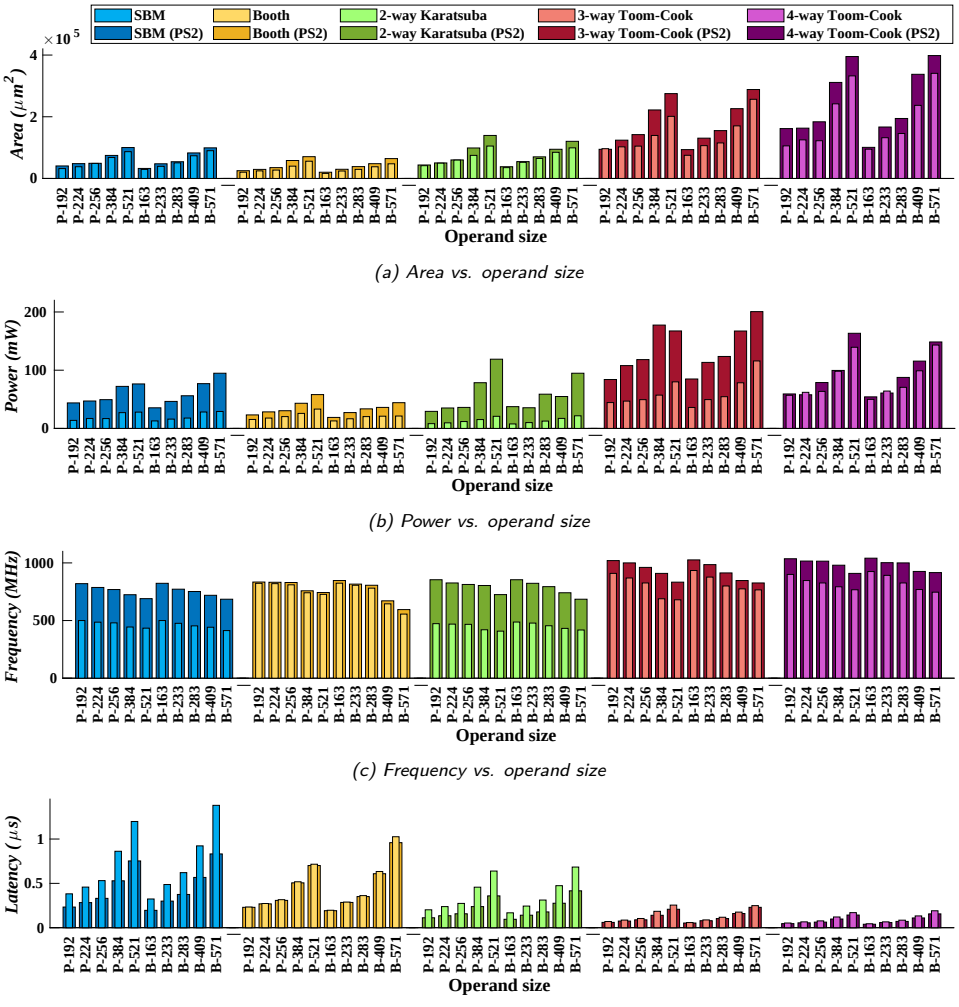
The proposed multiplier generator supports different multiplication architectures: non-digitized and digitized. Also, the implementation results are given on distinct platforms (FPGA and ASIC). A 15nm [111] and a 65nm technology are used for logic synthesis on the ASIC platform, while an Artix-7 device is used for synthesis on FPGA. The tools used for logic synthesis on ASIC and FPGA platforms are Cadence Genus and Vivado IDE. The NIST-recommended prime (192, 224, 256, 384, and 521) and binary (163, 233, 283, 409, and 571) elliptic curve fields are used for the performance evaluation of the supported non-digitized multipliers. To assess the performance of the digitized wrapper, different digit sizes are considered for the operand lengths 521, 571, and 1024. The performance of our generated multipliers is evaluated in terms of various design parameters, i.e., clock frequency, latency, area, and power. The frequency, area, and power values are obtained directly from the tools for both FPGA and ASIC evaluations. At the same time, latency is calculated using Eq. 13.

$$latency\,(\mu s) = \underbrace{\underbrace{\left(\frac{clock\,cycles}{frequency\,(MHz)}\right)}_{non-digitized} \times total\,digits}_{digitized} \tag{9}$$

Non-digitized multipliers on ASIC and FPGA platforms. Figures 12 and 13 show the implementation results for non-digitized polynomial multiplication methods (including non-pipelined and pipelined) over the NIST-recommended prime (P-192 to P-521) and binary (B-163 to B-571) fields utilized in ECC-based public-key cryptosystems on ASIC (65nm technology) and Artix-7 FPGA[3]. Moreover, Fig. 12a to 12d and Fig. 13a to 13d indicate the operand size and design feature (area in μm^2 for ASIC and slices for FPGA, power in mW, frequency in MHz and latency in μs) on horizontal and vertical

[3]This FPGA is designed in modern 28nm technology.

axis. The area of an FPGA implementation can be estimated in terms of LUTs, slices, Regs, DSP, and carry blocks. The implemented multipliers utilize LUTs, slices, Regs, and several F7 & F8 muxes. DSP and carry blocks are not utilized. Therefore, Fig. 13a shows slices as an area of the implemented multipliers because later slices are also utilized to define figures of merit.



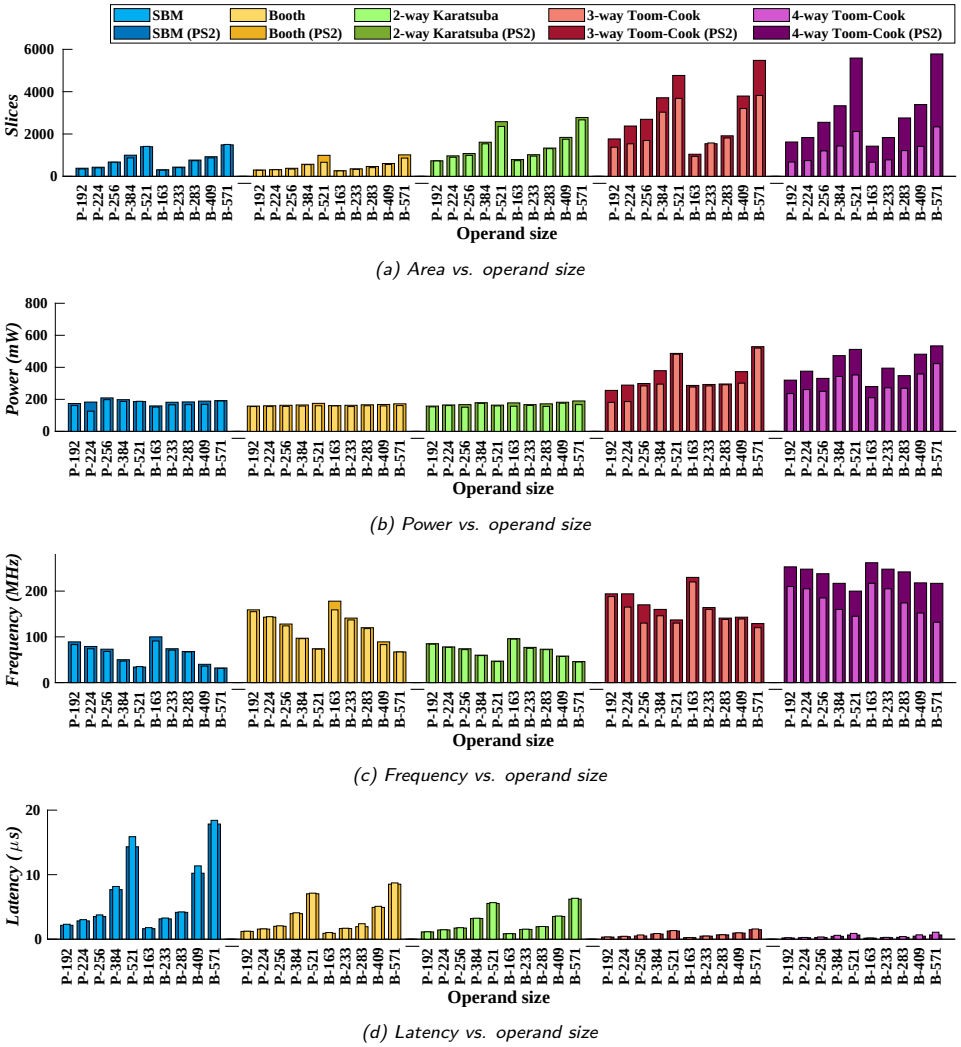
(a) Area vs. operand size

(b) Power vs. operand size

(c) Frequency vs. operand size

(d) Latency vs. operand size

Figure 12: Results for the non-pipelined and pipelined variants of several non-digitized multipliers on 65nm ASIC over NIST recommended prime and binary elliptic curves



(a) Area vs. operand size

(b) Power vs. operand size

(c) Frequency vs. operand size

(d) Latency vs. operand size

Figure 13: Results for the non-pipelined and pipelined variants of several non-digitized multipliers on Artix-7 FPGA over NIST recommended prime and binary elliptic curves

To comprehend Fig. 12a to Fig. 12d and Fig. 13a to Fig. 13d, assume the P-192-labeled left-first bar from the area (Fig. 12) and slices (Fig. 12) panels. Here, the field is determined by the first letter (P for prime, B for binary), and the integer specifies the multiplier input length. Furthermore, in figures 12 and 13, the results for five distinct multiplication methods are shown from left to right in the following sequence: (i) SBM; (ii) Booth; (iii) 2-way Karatsuba; (iv) 3-way Toom-Cook; and (v) 4-way Toom-Cook. For the color scheme of implemented non-pipelined and pipelined multiplier variants, see the legend of Fig. 12 and Fig. 13. The results for 2-stage pipelining are provided for the pipelined multiplier variants, which are annotated with the label 'PS2' in Fig. 12 and Fig. 13. The highest possible frequency is obtained by increasing pipeline stages until saturation occurs, and adding more stages is no longer beneficial. For the studied circuits, saturation occurs if more than 2 pipeline stages are added. A third stage brings

a minor increase in the clock frequency at a significant cost in area and power. As a consequence, this thesis shows results only for PS2.

Concerning the non-pipelined and pipelined multipliers on ASIC 65nm technology, as shown in Fig. 12, there is an increase in area, power, and latency characteristics with the increase in operand length. On the other hand, there is a decrease in clock frequency with the increase in operand length. Then, pipelining improves the performance (clock frequency) at the cost of area and power. It is important to note that the pipelined variant of the Booth multiplier results in minor improvements. Moreover, for every studied multiplier, the power of the pipelined variants is always higher than the non-pipelined ones.

The Booth multiplier uses less area than the other evaluated multipliers, as shown in Fig. 12, for pipelined and non-pipelined versions. Additionally, the 2-way Karatsuba variant without pipelines gets lower power values than other chosen multipliers. The Booth multiplier uses less power for pipelined variants. The rationale is that Booth has the simplest datapath among the multipliers under study. For example, in our implemented architectures, SBM needs a $2m+2m$ bit adder, Booth requires an m bit adder and subtractor, 2-way Karatsuba requires $m+m+m$ bit adder and subtractor, 3-way Toom-Cook requires $\frac{m}{4}$ bit incrementer, and 4-way Toom-Cook requires sixteen $\frac{m}{4}$ bit incrementers. For non-pipelined and pipelined implementations, variants of Toom-Cook multipliers report higher clock frequency and lower latency values.

In contrast to ASIC evaluations, the performance of the non-pipelined and pipelined multipliers over Artix-7 FPGA is different because the implementation platforms are relatively different. For both non-pipelined and pipelined multipliers, as shown in Fig. 13, there is an increase in area, power, and latency characteristics with the increase in operand length. On the other hand, there is a decrease in clock frequency with the increase in operand length. Alike in ASIC implementations, pipelining improves the performance (clock frequency) with an excess of both area and consumed power. The latency trend is opposite to the clock frequency, it increases as the operand size increases, but the pipeline stages decrease the latency. As shown in Fig. 13, the Booth multiplier uses fewer FPGA slices than the other evaluated multipliers. Moreover, the non-pipelined and pipelined variants of 2-way Karatsuba achieve lower power values than other selected multipliers. Similar to the ASIC implementations, non-pipelined and pipelined variants of a Toom-Cook multiplier result in higher clock frequency and lower latency values.

In summary, the results obtained from ASIC and FPGA analysis of non-digitized multipliers demonstrate that multiple design parameters, including area, power, frequency, and latency, are subject to trade-offs. The findings also indicate that the choice of a multiplier architecture depends on the application's specific requirements. For applications prioritizing reduced hardware resource utilization, bit-serial multiplication approaches like SBM and Booth are more practical. Contrarily, for high-speed applications, utilizing bit-parallel multiplication approaches, including 2-way Karatsuba and variants of Toom-Cook, offers more significant benefits.

Digitized SBM multiplier on ASIC and FPGA platforms. The experimental results for the non-pipelined digitized multiplier wrapper on ASIC 65nm technology are shown on the left portion of Table 3. Similarly, the right part of Table 3 provides implementation results of the non-pipelined digitized multiplier wrapper on Artix-7 FPGA. For synthesis on both ASIC and FPFA platforms, the selected lengths of the input operands are 521, 571, and 1024, as given in column one of Table 3. The selected digit sizes (n) for input lengths 521 and 571 are 32, 41, 53, and 81. For an input length

Table 3: ASIC and FPGA results for digitized multipliers of various input sizes

m	n	d	ASIC (65nm)				FPGA (Artix-7)					
			Freq MHz	Lat μs	Area μm^2	Pow mW	Freq MHz	Lat μs	LUTs	Regs	CBs	Pow mW
521×521	32	17	505	1.07	106956.7	30.9	33.11	16.43	6369	1692	408	184
	41	13	377	1.41	101538.7	26.1	29.15	18.28	7995	1681	416	192
	53	10	340	1.55	94752.7	20.0	28.32	22.72	8079	1732	417	191
	81	7	336	1.68	84321.0	15.4	34.48	15.12	6095	1758	408	220
571×571	32	18	487	1.18	114999.8	36.7	30.12	18.06	6397	1847	447	194
	41	14	369	1.55	116010.3	28.9	27.17	19.62	8750	1834	455	192
	53	11	312	1.86	91393.9	18.1	26.04	20.35	9053	1880	449	187
	81	8	291	2.22	76146.8	14.1	28.01	23.13	8958	1951	452	226
1024×1024	2	512	363	2.82	196131.2	38.0	14.22	72.11	10993	3634	1085	173
	4	256	357	2.86	178581.2	35.1	15.89	64.48	10824	3384	928	172
	8	128	353	2.90	167536.4	31.5	16.86	60.66	11074	3261	849	180
	16	64	343	2.98	166533.1	30.2	17.51	58.48	10634	3248	811	185
	32	32	313	3.27	148489.5	23.0	17.89	57.28	11371	3267	791	190
	64	16	285	3.59	122257.8	20.8	17.89	57.04	11947	3330	792	195
	128	8	268	3.82	123164.6	19.9	18.57	55.14	12207	3450	800	221
	256	4	263	3.89	129542.4	19.5	18.93	54.09	11367	3740	832	247
	512	2	261	3.92	136292.4	23.1	19.12	53.55	10385	4295	896	226
	1024	1	259	3.95	177834.2	24.1	18.46	55.50	11462	5303	1024	235

m: is the field size or length of the inputs (in bits), **n:** is the digit size, **d:** shows total digits.

of 1024 bits, digit sizes are selected in powers of two, for $n = 2 \dots 1024$ where the values for digit size n and total digits d are shown in columns two and three of Table 3.

Regarding the implementation results for ASIC 65nm technology, it shows that the increase in digit size leads to a decrease in clock frequency, as given in column four of Table 3. The increase in digit size increases latency, as shown in column five of Table 3. With an increase in the digit size n, the achieved results for power and area parameters indicate behavior akin to a parabolic curve, as provided in Table 3 (see columns six and seven). For extreme cases of too small or too large digits, the wrapper logic becomes inefficient and may even become the bottleneck for timing. Therefore, shorter digit lengths are more valuable for an application that demands high speed. On the other hand, the reported results on Artix-7 reveal that the increase in digit size increases clock frequency, as shown in column eight of Table 3. This increase in clock frequency occurs until a saturation point is reached. Once the saturation point is reached, clock frequency decreases with the increase in digit size. Therefore, in this particular experiment, saturation occurs when the value for $n = 512$. Yet, before saturation is achieved, tiny increments in frequency are already observed, implying that selecting the number of digits based on frequency alone is not a good strategy. Other reported characteristics, i.e., latency, LUTs, and power, show a non-linear behavior (see columns nine, ten, and thirteen of Table 3). As summarized, the implementation results achieved after synthesis (clock frequency, area in terms of LUTs, Regs and Carry blocks, latency, and power) for FPGA are different compared to ASIC as the implementation platforms are relatively different.

The results for various digit sizes of 1024×1024 SBM multiplication method on 15nm technology are presented in Table 4. The selected length of input operands is 1024, as shown in column one of Table 4. For an input length of 1024 bits, digit sizes

Table 4: Synthesis results for 1024×1024 digitized multiplier on ASIC 15nm

m	***n***	***d***	***Freq*** ***(MHz)***	***Lat (μs)***	***Area*** ***(μm^2)***	***Pow*** ***(mW)***
1024×1024	2	512	909	1.12	19182.7	21.0
	4	256	884	1.15	19059.8	19.9
	8	128	862	1.18	18367.2	21.2
	16	64	840	1.21	17398.7	20.9
	32	32	829	1.23	17105.5	20.8
	64	16	826	1.23	17523.4	20.5
	128	8	822	1.24	17460.4	19.9
	256	4	819	1.25	18594.0	23.5
	512	2	813	1.25	19719.6	25.4
	1024	1	806	1.27	22979.3	30.2

m: specifies the inputs length (in bits), **n:** shows the digit size, **d:** is the total digits.

are selected again in powers of two, for $n = 2 \ldots 1024$ where the values for digit size n and total digits d are shown in columns two and three of Table 4. Columns four to seven provide the frequency (Freq in MHz), latency (Lat in μs), area (in μm^2), and power (in mW). The results show that the increase in digit size reduces clock frequency, as given in column four of Table 4. On the other hand, the increased digit size increases latency, as presented in column five of Table 4. With an increase in the digit size n, the achieved power and area parameters results indicate behavior similar to a parabolic curve, as shown in the last two columns of Table 4. Similar to the results obtained on 65nm technology, for extreme cases of too small or too large digits, the wrapper logic becomes inefficient and may even become the bottleneck for timing. Therefore, shorter digit lengths are more useful for an application that demands high speed.

The implementation results for identical values of m, n, and d in Table 3 and Table 4 achieved on 15nm technology outperform the results obtained on 65nm technology, as expected. More specifically, the 15nm technology allows for a threefold increase in clock frequency with a significant reduction in area and power.

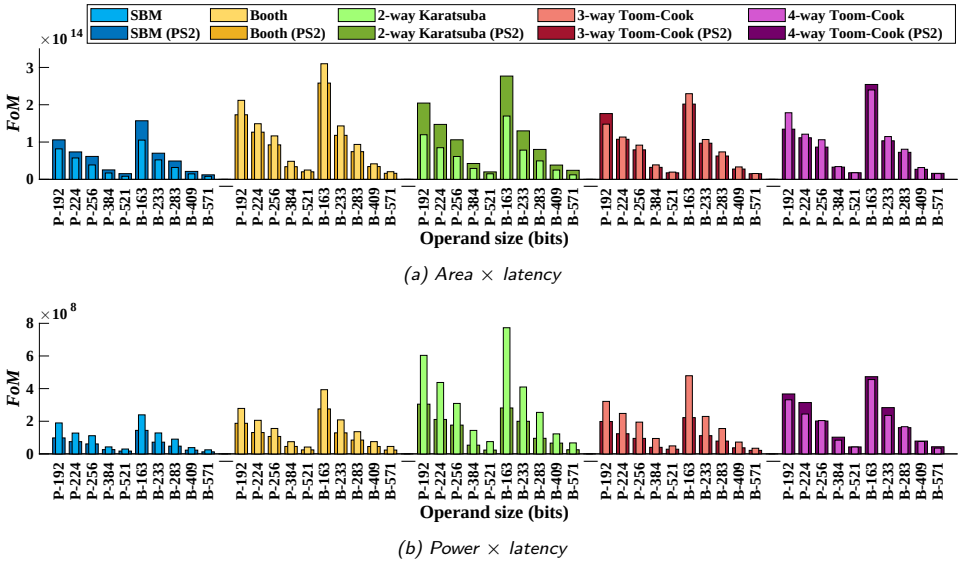
3.4 Figures of Merit and Trade-offs

The previous section has presented only the implementation results for non-digitized and digitized multiplier wrapper. However, FoMs are defined to analyze the performance of non-digitized and digitized multipliers using the combined effect of their characteristics simultaneously. Thus, an FoM to evaluate the area and performance for both ASIC and FPGA platforms is defined using Eq. 10. For FPGA, the number of slices is utilized as area in Eq. 10. The higher the FoM values, the better performance of the multiplier. Similarly, an FoM is calculated using Eq. 11 to evaluate the power and latency parameters.

$$FoM = \frac{1}{area\ (\mu m^2) \times latency\ (\mu s)} \tag{10}$$

$$FoM = \frac{1}{power\ (mW) \times latency\ (\mu s)} \tag{11}$$

FoM for non-digitized multipliers on ASIC and FPGA platforms. The calculated values of defined FoMs for both non-pipelined and pipelined multipliers on ASIC and



(a) Area × latency

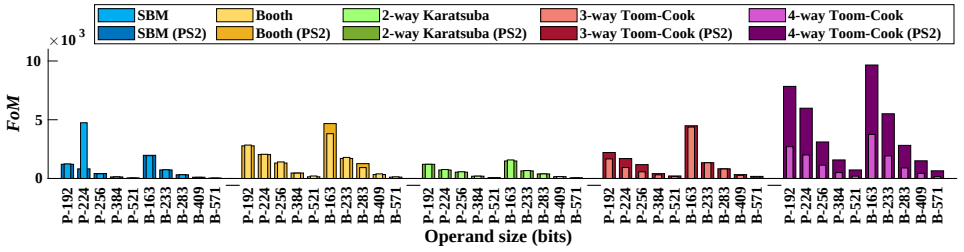
(b) Power × latency

Figure 14: FoMs regarding area vs. latency and power vs. latency for various non-digitized multipliers on ASIC.

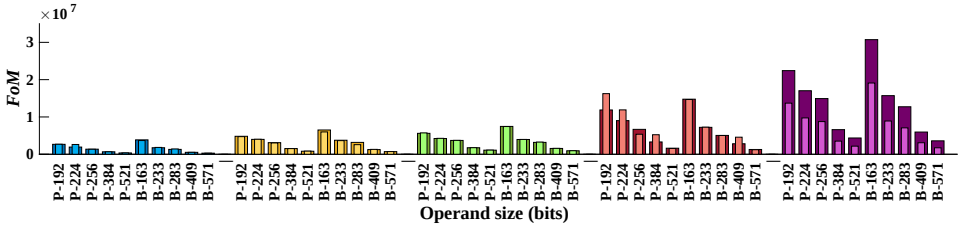
FPGA platforms are illustrated in figures 14 and 15, respectively – where PS2 (a 2-stage pipeline) shows the pipelined variants for different multipliers. For each panel in figures 14 and 15, the multipliers are shown from left to right in the following order: (i) SBM, (ii) Booth, (iii) 2-way Karatsuba, (iv) 3-Way Toom-Cook and (v) 4-way Toom-Cook.

For the ASIC platform (Fig. 14), the trend shows a decrease in the FoM values with increased operand size. Concerning Fig 14a, the value of the non-pipelined multiplier is lower than the pipelined multiplier except for the Booth and variants of Toom-Cook multipliers. For pipelined multipliers, the highest value of FoM for Eq. 10 is achieved for the 2-way Karatsuba multiplier. The performance (latency) versus area trade-off for non-pipelined multipliers could be graded, from highest to lowest, as (i) Booth, (ii) 4-way Toom-Cook, (iii) 3-way Toom-Cook, (iv) 2-way Karatsuba and (v) SBM. For similar performance versus area trade-off, the possible grading from highest to lowest for the pipelined multipliers is (i) 2-way Karatusuba, (ii) 4-way Toom-Cook, (iii) Booth, (iv) 3-way Toom-Cook and (v) SBM. As far as the trend from Fig. 14b is concerned, the value of the FoM for non-pipelined multipliers is higher than pipelined variants except for the 4-way Toom-Coom multiplier. For non-pipelined and pipelined variants, the highest FoM value for Eq. 11 is achieved for a 2-way Karatsuba and 4-way Toom-Cook multiplier. Based on Fig. 14b, the latency versus power trade-off of the non-pipelined multipliers could be graded as (i) 2-way Karatsuba, (ii) 3-way Toom-Cook, (iii) 4-way Toom-Cook, (iv) Booth and (v) SBM. Furthermore, for similar performance versus power trade-off, the possible grading from highest to lowest for pipelined multipliers is (i) 4-way Toom-Cook, (ii) 2-way Karatusuba, (iii) Booth, (iv) 3-way Toom-Cook and (v) SBM.

Similarly, for the FPGA platform (Fig. 15), the values for the non-pipelined multipliers are lower than the pipelined variants, except for the SBM and 3-way Toom-Cook multipliers. For pipelined multipliers in Fig. 15a, the highest FoM is achieved for the



(a) Area (Slices) × latency



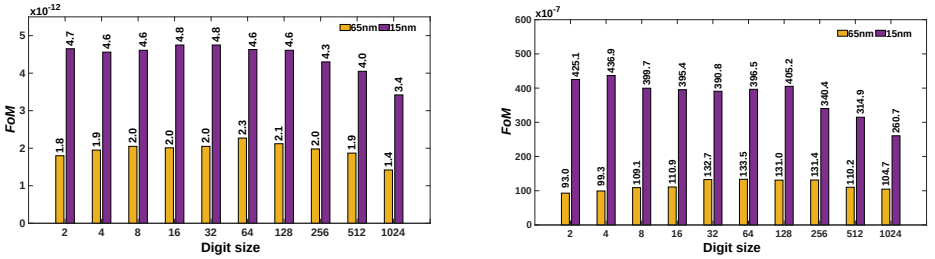
(b) Power × latency

Figure 15: FoMs regarding area vs. latency and power vs. latency for various non-digitized multipliers on FPGA.

4-way Toom-Cook. The performance (latency) versus area trade-off for non-pipelined multipliers could be ranked, from highest to lowest, as (i) Booth, (ii) 4-way Toom-Cook, (iii) 3-way Toom-Cook, (iv) 2-way Karatsuba and (v) SBM. For equivalent performance versus area trade-off, the possible ranking from highest to lowest for the pipelined multipliers is (i) 4-way Toom-Cook, (ii) Booth, (iii) 3-way Toom-Cook, (iv) 2-way Karatusuba and (v) SBM. Notice that SBM is the least preferred multiplier according to the defined FoMs. Concerning Fig. 15b, for non-pipelined and pipelined variants, the highest FoM value is achieved for 3-way and 4-way Toom-Cook multipliers, respectively. Moreover, the performance (latency) versus power trade-off of the non-pipelined multipliers could be ranked as (i) 3-way Toom-Cook, (ii) 4-way Toom-Cook, (iii) 2-way Karatsuba, (iv) Booth, and (v) SBM. For identical performance versus power trade-off, the ranking from highest to lowest for pipelined multipliers is (i) 4-way Toom-Cook, (ii) 3-way Toom-Cook, (iii) 2-way Karatusuba, (iv) Booth, and (v) SBM.

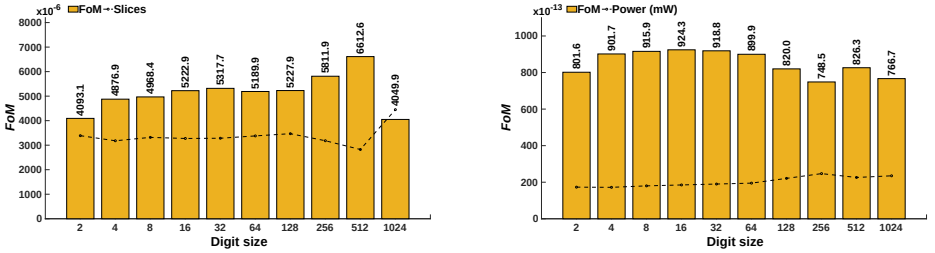
Figures 14 and 15 assist the designer in selecting a suitable multiplier architecture according to application requirements. From an area perspective, SBM is the best candidate. However, even if SBM has a relatively small footprint and relatively small power consumption, this comes at the expense of performance. The pipelined variant of the Booth multiplier is also a good candidate with the least power and optimal performance compared to the non-pipelined version of a 4-way Toom-Cook multiplier. These examples show the PPA trade-offs that are considered based on the FoMs in this study.

FoM for digitized SBM multiplier on ASIC and FPGA platforms. A 1024×1024 multiplier is considered with various digit sizes to calculate FoM for evaluation on ASIC and FPGA platforms. The calculated FoM results for ASIC on 15 and 65nm technologies are shown in Fig. 16, while the calculated values of FoM in terms of area$\times$latency and power$\times$latency for FPGA are shown in figures 17a and 17b, respectively.



(a) Area × latency *(b) Power × latency*

Figure 16: FoMs in terms of area × latency and power × latency for digitized wrapper with SBM multiplier on ASIC



(a) Area (slices) × latency FoM for FPGA *(b) Power × latency FoM for FPGA*

Figure 17: FPGA FoMs in terms of area × latency and power × latency for digitized wrapper with SBM

Let us consider only the FoM results from Fig. 16 for evaluations; it becomes clear that the extreme cases lead to suboptimal results for both FoMs (area×latency and power×latency) on 65nm technology (presented in figures 16a and 16b). This is not evident for the FoMs calculated on the 15nm technology where longer digit cases lead to suboptimal results. For the studied 1024 × 1024 multiplier, the variant with $n = 64$ and $d = 16$ presents an optimal solution on 65nm technology. Similar values, such as $n = 32$ and $n = 128$, also give very close to optimal solutions. On 15nm technology, the optimal solutions for area × latency are achieved for $n = 16$ and $n = 32$. Additional closer values to optimal solutions are achieved for digit sizes $2, 4, 8, 64$, and 128. Similarly, a digit size for $n = 4$ provides the best power × latency solution.

There are multiple approaches to evaluating FoM results on FPGA. For example, the number of FPGA basic building blocks for area evaluation are slices, LUTs, flip-flops, and carry units. However, the FoM in Eq. 10 can be calculated using different metrics of interest (such as slices, LUTs, flip-flops, or carry blocks). This study substitutes FPGA slices for the area in Eq. 10. Therefore, Fig. 17a shows that the FoM values for $n = 512$ and $d = 2$ result in an optimal solution. Fig. 17b reveals that the optimal solution is achieved for $n = 2$ and $d = 512$. Hence, they are all very close.

3.5 Comparison and Discussion

To perform a realistic and reasonable comparison with state-of-the-art, we have used similar operand lengths, digit sizes, and implementation platforms as presented in Table 5. Column one presents the reference design (Ref). The implemented multiplier,

utilized platform (device), and targeted operands length (m) are given in columns two to four. Different values of m are considered in the existing implementations to present results for polynomial multiplications. However, for our comparison, we have considered only the larger operands. The implemented circuit's clock frequency ($Freq$ in MHz) is given in column five of Table 5. The last two columns (six and seven) provide the latency (Lat in μs) and the hardware resources (in μm^2 for ASIC and in LUTs for FPGA), respectively. In Table 5, 'N/A' is utilized to denote values that are not provided.

Table 5: Comparison with state-of-the-art multipliers

Ref	***Multiplier***	***Device***	m	***Freq (MHz)***	***Lat (μs)***	***Area (μm^2)/LUTs***
[97]	BL-PIPO	65nm	163	N/A	N/A	5328 GE
[112]	LCHMA	65nm	163	68.49	N/A	321692
		Virtex-4	163	33.78	N/A	34118 (19030 slices)
[92]	Radix-2 Montgomery	Virtex-6	1024	53.23	19.26	2566
[98]	Systolic Montgomery	90nm	13	100	0.91	4782
[113]	Montgomery	Virtex-5	1024	400	0.88	6105 slices
[104]	PCA approach	Virtex-II	163	177.8	0.91	225 slices
			128	104.3	0.61	3499
[90]	2-way Karatsuba	Virtex-7	256	74.5	1.71	7452
			512	51.6	4.96	20474
[91]	DSM	Virtex-6	571	258.5	0.03	10983 (when ds=64)
			2048	N/A	N/A	18067 (when ds=2)
[106]	DSMM	Virtex-7	2048	N/A	N/A	33734 (when ds=4)
			2048	N/A	N/A	62023 (when ds=8)
			571	540/CCs=571	1.05	1731 (when ds=1)
			571	550/CCs=286	0.52	1730 (when ds=2)
			571	572/CCs=143	0.25	2302 (when ds=4)
[100]	SBM (digit serial)	Virtex-5	571	450/CCs=72	0.16	3451 (when ds=8)
			571	400/CCs=36	0.09	5754 (when ds=16)
			571	400/CCs=24	0.06	8051 (when ds=24)
			571	360/CCs=18	0.05	10350 (when ds=32)
TW	SBM	65nm	163	500	0.326	29341 (11727 GE)
		Virtex-4	163	65.68	2.48	1934 (987 slices)
	Booth	Virtex-4	163	131	1.24	565 slices
		Virtex-6	1024	71.5	14.32	2429
		65nm	163	824	0.19	20258.6
		Virtex-5	1024	39.35	13.01	4113 slices
	2-way Karatsuba	Virtex-7	128	167.4	0.38	2110
			256	119.9	1.06	4318
			512	63.8	4.01	9582
	SBM Wrapper	Virtex-6	571	46.4	1.74	6181 (when ds=64)
			2048	15.03	69760	25559 (when ds=2)
		Virtex-7	2048	16.6	15790	22040 (when ds=4)
			2048	17.4	3760	23315 (when ds=8)
			571	23/CCs=571	24.82	11803 (when ds=1)
			571	27.1/CCs=286	10.55	10353 (when ds=2)
			571	30/CCs=143	4.76	9209 (when ds=4)
		Virtex-5	571	32/CCs=72	2.25	9399 (when ds=8)
			571	33/CCs=36	1.09	8713 (when ds=16)
			571	30/CCs=24	0.80	16536 (when ds=24)
			571	34/CCs=18	0.52	8767 (when ds=32)

BL-PIPO: Bit level parallel in parallel out multiplier using SBM multiplication method, **PCA:** programmable cellular automata, **DSM:** Digit Serial Montgomery multiplier based wrapper, **ds:** digit size, **DSMM:** Digit Serial modular multiplier, **GE:** Gate equivalent, **LCHMA:** Low-complexity hybrid multiplier architecture, **TW:** this work, latency reported for design [98] is in milli-second.

Bit-serial architectures. FPGA results for operand length of 1024 are reported in [92] where the authors have utilized a Virtex-6 device. A Radix-2 Montgomery multiplier architecture [92] results in 25% higher clock frequency and latency than the Booth multiplier generated by TTech-LIB. The excessive use of LUTs in their implementation is noticeable (see the last column of Table 5). On the Virtex-5

device, FPGA implementations for 1024-bit operand lengths are reported in [113]. The Montgomery multiplier architecture of [113] results in 9.83 times higher clock frequency when compared to the Booth multiplier generated by TTech-LIB. Due to higher frequency, they have achieved a latency value of $0.88\mu s$ that is comparatively 2.81 times lower than the TTech-LIB generated Booth multiplier circuit ($2.48\mu s$). On the other hand, there is a trade-off since the generated Booth multiplier utilizes 1.48 times fewer FPGA slices.

The comparison to systolic Montgomery multiplier architecture of [98] can be a little unfair as this study uses a 65nm technology for logic synthesis while a 90nm technology is considered in [98]. However, the TTech-LIB-generated SBM and Booth serial multipliers have been compared. For operands length of 13-bit over elliptic curve binary $GF(2^{13})$ field, their architecture achieves 5 times lower clock frequency when compared to 163-bit generated (by TTech-LIB) SBM and Booth multiplier implementations. The generated SBM and Booth implementations utilize the higher area and take more computational time as the length of the operands is 12.5 times higher than [98].

For 163-bit operands size on 65nm ASIC and Virtex-4 FPGA platforms, the low-complexity hybrid multiplier architecture of [112] is 7.30 and 1.94 times slower in clock frequency as compared to SBM generated multiplier by TTech-LIB. As shown in Table 5, the latency comparison is hard as the related information is not described in [112]. Moreover, the generated SBM multiplier by TTech-LIB utilizes 10.96 and 17.64 times lower hardware resources on similar ASIC and FPGA platforms.

In [104], for 163-bit operands length, a programmable cellular automata-based bit-serial multiplier design is reported on Xilinx Virtex-II Pro FPGA[4]. Therefore, this study uses a Virtex-4 device built on a 90nm technology to provide a comparison that is not disproportionately unfair. As shown in Table 5, the dedicated architecture of [104] results in lower hardware resources (225 slices whereas the generated Booth multiplier by TTech-LIB used 565) and achieves higher clock frequency ($177.8MHz$ while the generated Booth multiplier design in this study operates at $131MHz$). This comparison shows that there is always a trade-off between flexibility and performance (area, clock frequency, latency, etc.).

Bit-parallel designs. A bit-parallel 2-way Karatsuba multiplier is reported in [90] for a Virtex-7 FPGA. In terms of latency, it is 38% (for operand size of 128 bit), 39% (for operand size of 256 bit), and 20% (for operand size of 512 bit) slower when compared to 2-way Karatsuba multiplier generated by TTech-LIB, as shown in Table 5. Additionally, the proposed 2-way Karatsuba multiplier requires fewer FPGA LUTs (see column seven in Table 5) as compared to [90]. The BL-PIPO multiplier of [97] on 65nm technology utilizes 55% lower gate counts compared to the SBM multiplier generated by TTech-LIB. However, the multiplier given in [97] shares resources with a reduction unit specific for the 163-bit operand. The proposed multiplier generates a $2 \times m - 1$ bit output, whereas their solution generates an m bit output.

Digitized solutions. The digit-serial Montgomery multiplier wrapper of [91] results in 83% higher clock frequency and 58% lower latency than the proposed digitized solution based on SBM multiplier architecture. This is valid when the digitized flavor of polynomials multiplication is considered for comparison over different digit sizes. Contrarily, the generated digit serial wrapper by TTech-LIB results in 56% lower hardware resources over Virtex-6 FPGA. Another digit serial modular multiplication wrapper of [106] results in 14% (for ds=2) lower FPGA LUTs, while for remaining digit sizes of 4 and 8, it utilizes 35% and 63% higher FPGA LUTs as compared to SBM wrapper

[4]The Xilinx Virtex-II Pro devices are built on a 90nm technology.

generated by TTech-LIB. The frequency and latency parameters cannot be compared because the relevant information is unavailable in the reference designs.

In [100], a digit-serial multiplier for the operand length of 571 bits over Virtex-5 is described, as shown in Table 5. With the increase in digit sizes (i.e., 1, 2, 4, 8, 16, 24, and 32), the digit-serial multiplier of [100] results in an increase in the hardware resources (LUTs) and a decrease in clock cycles (CCs) and latency. For clock frequency, it shows behavior like a parabolic curve. This is not the case for TTech-LIB offered digit-serial wrapper as it considers the flexibility which is not tackled in the design of [100]. With a similar clock cycle requirement, a digit-serial wrapper generated by TTech-LIB takes more computational time and achieves lower clock frequency than [100]. Moreover, the wrapper generated by TTech-LIB utilizes more hardware resources for ds $=$ 1, 2, 4, 8, 16, and 24. For a digit size of 32 (see the last column of Table 5), the proposed wrapper utilizes 1.18 times lower hardware resources with an overhead in latency. Therefore, the wrapper generated by TTech-LIB outperforms in terms of hardware resources (LUTs) for larger digit sizes compared to [100].

In summary, the comparisons and discussion reveal that the versatile and flexible TTech-LIB multiplier generator provides, in general, a *realistic* and *reasonable* comparison to many existing multiplier architectures [98, 92, 104, 97, 90, 91, 106, 100]. It is essential to highlight that some of the compared architectures also contain reduction routines in their implementation, a feature that is not currently supported by TTech-LIB generator but could be considered in future. Based on the results, it has been evaluated that designers can explore various design parameters within TTech-LIB-supported multiplier architectures and benefit from competitive implementations concerning the existing literature on polynomial multipliers. Since the TTech-LIB generator produces RTL code that is technology- and platform-agnostic, users can (also) take the code as a starting point for their design, develop the optimized ones, and define other FoMs for further evaluation.

4 Design Space Exploration of SABER

This chapter provides the design space exploration of PQC algorithms for performance improvement on the ASIC platform. The DSE process is accomplished by adopting several memory configurations and employing wider datapaths. A SABER PQC algorithm/protocol is considered as a case study in this thesis to perform the DSE process. Section 4.1 describes the DSE process, including the SABER architectural details. Section 4.2 describes the area, timing, and power results. The comparison to existing state-of-the-art hardware accelerators and discussions are provided in Section 4.3.

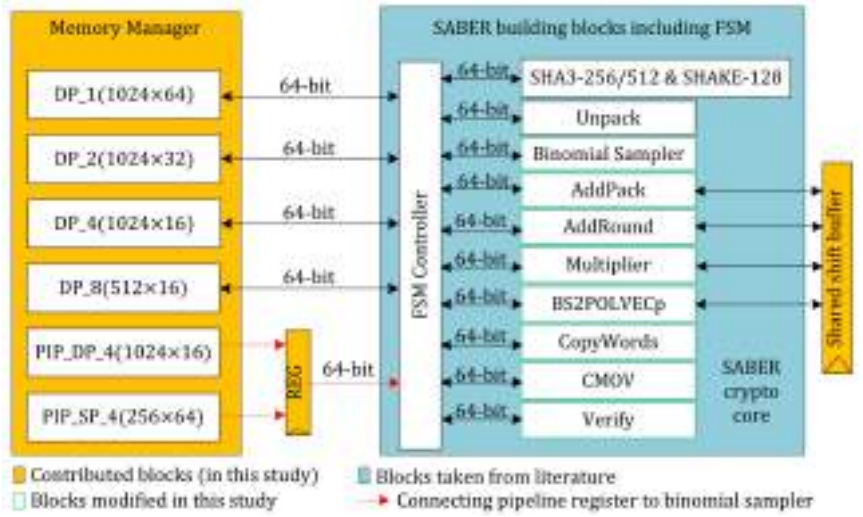
The design space exploration, in this study, determines the adaption of various architectural elements such as distinct memory configurations, pipelining, logic sharing, and different data path widths with an emphasis on optimizing the design for a specific 65nm ASIC technology. Therefore, an open-source implementation of SABER is selected to initiate the DSE process. This open-source SABER code is modeled as an instruction set co-processor architecture, and the code is written in Verilog HDL at Register Transfer Level. The Verilog code of the SABER co-processor can be accessed directly from [114] and the corresponding architectural details and implementation results appeared in [30]. The top-level block diagram of the SABER architecture of [30] consists of four units: (i) a data memory; (ii) a program memory; (iii) a dedicated finite state machine (FSM) controller for efficient control functionalities; and (iv) SABER building blocks. The building blocks of SABER are (i) a polynomial Vector-Vector multiplier wrapper; (ii) variants of secure hash algorithms, i.e., SHA3-256, SHA3-512, and SHAKE-128; (iii) a binomial sampler; (iv) AddPack; (v) AddRound; (vi) Verify; (vii) CMOV; (viii) Unpack; (ix) CopyWords; and (x) BS2POLVEC$_p$.

Note that the open-source SABER code of [114] was developed specifically for an FPGA platform, but in this study, the ASIC platform is targeted. Therefore, a baseline ASIC architecture is developed to evaluate SABER on a 65nm commercial technology to fulfill the DSE premise. The strategy employed in this thesis differs from the approach of [30], as BRAM is replaced with an SRAM in the baseline design. Furthermore, a commercial memory compiler from a partner foundry is used in this work to generate the same size SRAM memory. The next section will show multiple variants where several memory instances have been used with different sizes. It is essential to mention that the baseline design in this study is still a co-processor architecture and assumes that the program memory resides outside the SABER accelerator. The other building blocks are considered as implemented in the open-source SABER accelerator of [114], but most of them are modified during the DSE process, which will be detailed in the upcoming sections.

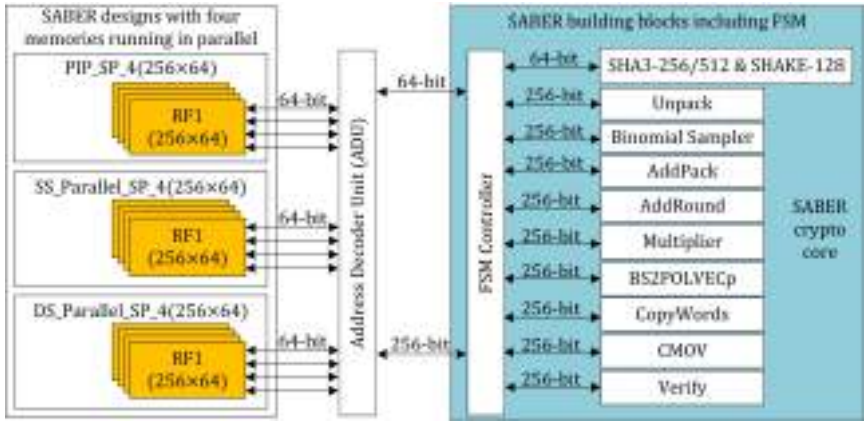
The DSE process is initiated with a SABER serial architecture and completed with parallel designs. Therefore, a total of eight SABER designs include in the DSE, one corresponds to the baseline and the remaining seven are optimized ones, including serial and parallel techniques. The details of the corresponding serial and parallel SABER designs are described in the upcoming section.

4.1 Serial and Parallel SABER Architectures

A summary of the DSE process is shown in Fig. 18, where Fig. 18(a) describes the serial SABER design by using several optimization approaches (pipelining, resource sharing and different memory configurations). In contrast, Fig. 18(b) provides the parallel SABER architecture using a wider 256-bit data path strategy instead of a 64-bit data path utilized in the baseline serial SABER design.



(a) Serial SABER architecture by utilizing different memory configurations, pipelining, and resource sharing.



(b) Parallel SABER architecture by employing wider data path strategy.

Figure 18: Block diagrams of the designs generated during the design space exploration.

Hence, Fig. 18(a) and Fig. 18(b) show different SABER designs with different names to differentiate the studied architectures from one another. For example, the prefixes DP and SP mean that the architecture employs a dual-port or a single-port memory. Similarly, the PIP prefix implies that the architecture is pipelined. Moreover, the prefixes SS and DS show that the design uses single-sponge and double-sponge KECCAK functions inside the SHA3 variants for hash operations of parallel SABER designs (of Fig. 18(b)). A prefix Parallel determines that the SABER design supports a 256-bit data path instead of a 64 bit. Note that in the serial SABER architectures of Fig. 18(a), single-sponge is used in the SHA3 variants, and the architectures differ in different memory sizes and number of memory instances used. On the other hand, in the parallel SABER implementations of Fig. 18(b), memory instances and size are fixed, but the architectures vary with single- and double-sponge functions. Therefore, based on this terminology, the following architectures have been considered:

- **Baseline**
  - DP_1(1024x64)
- **Optimized**
  - DP_2(1024x32)
  - DP_4(1024x16)
  - DP_8(512x16)
  - PIP_DP_4(1024x16)
  - PIP_SP_4(256x64)
  - SS_Parallel_SP_4(256x64)
  - DS_Parallel_SP_4(256x64)

Subsequently, seven optimized designs have been presented; the first five optimized designs have been generated from the baseline design from Fig. 18(a), and the last two-optimized designs have been presented from Fig. 18(b). The memory is structured as $i(m \times n)$, where i shows the number of memory instances, m determines the number of memory addresses, and n implies the data width of each address.

In contrast to the open-source design of [114], the DSE process led to the creation of new units in addition to the FSM controller and SABER building blocks shown in Fig. 18(a): (i) memory manager; (ii) pipeline register; and (iii) shared shift buffer. All these units are common to all of the studied serial SABER architectures except for the pipeline register, which is considered only in pipeline architectures, i.e., PIP_DP and PIP_SP. It is essential to highlight that several modifications have been performed in the SABER building blocks to synchronize their inputs/outputs with the memory timing requirements. The modified SABER building blocks are outlined with green color lines in Fig. 18(a).

On the other hand, the parallel SABER designs constructed from Fig. 18(b) utilize four smaller SRAM-based RegFile memories having a size of 256×64 each. The difference is that in Fig. 18(a), all the memory instances operate serially. This means that at one time, only one 64-bit word can read/write on one memory. But in Fig. 18(b), four smaller memories operate in parallel. Each memory can read/write one 64-bit word in one clock cycle. So, four memory instances in parallel can read/write one 256-bit word in one cycle. To deal with 256-bit words in Fig. 18(b), an additional address decoder unit is required. This address decoder unit prepares the corresponding 64-bit or 256-bit word for the SABER controller to operate SABER building blocks.

The following text will describe the design blocks of Fig. 18(a) and Fig. 18(b) in detail. Moreover, from now on, the baseline and optimized SABER architectures will be referred to using abbreviated forms. For example, the optimized SABER PIP_SP_4(256×64), SS_Parallel_SP_4(256×64), and DS_Parallel_SP_4(256×64) designs will be abbreviated as PIP_SP, SS_Parallel, and DS_Parallel.

4.1.1 Memory Manager

In Fig. 18(a), the name 'memory manager' comes from the smart memory synthesis [115] process concept. The smart memory synthesis is the observation that smaller, more distributed memories can benefit an ASIC design because the smaller memories need simpler address decoder units which are faster. This, combined with the fact that part of the address decoding is now described as logic and can be co-optimized with the

remainder of the design, leads to performance improvements with a sometimes marginal increase in area. Hence, in this study, a smart memory synthesis strategy is explored within the limitations of a commercial memory compiler. For the key encapsulation mechanism, when security is equivalent to AES-192, SABER needs 992, 1344, and 1088 bytes for generating a single public-key, secret-key, and the cipher text [21]. This need confirms that a relatively large memory size is needed. Hence, a dual-port memory size of 1024×64 is employed in the FPGA design of SABER in [30] and the baseline SABER design in this work incorporates the same memory size of 1024×64.

The DSE process is initiated where the word size of the employed large memory of size 1024×64 is divided into smaller chunks (32 and 16), and then the number of memory instances is increased accordingly. With this division, the memory structure becomes DP_2(1024×32) and DP_4(1024×16). More precisely, DP_2(1024×32) means that two instances of a dual-port memory are employed where the total number of addresses is 1024, and the data stored on each address is 32-bit. For the memory structure of DP_4(1024×16), four instances of dual-port memory are used where the total number of addresses is 1024, and the word size is 16. As expected, these memory choices increase clock frequency but at the expense of area and power. Afterward, from DP_4(1024×16) memory structure, another architecture is constructed where the number of addresses is (also) divided to take half per memory (i.e., from 1024 to 512). In this case, the memory structure becomes DP_8(512×16), and this means that eight instances of dual-port memory are used where the total number of addresses is 512, and the word size is 16. Later in the results section, I will show that this design choice increases area and power with a minor gain in operating frequency. Hence, at this stage, it has been realized that further dividing memories into smaller chunks is no longer beneficial, and other optimization approaches must be explored.

Hence, pipelining is utilized after dividing the number of addresses and data widths for memories. In the first pipelined design, i.e., PIP_DP, the same 4(1024×16) memory structure is chosen as in DP_4(1024×16). The second pipelined architecture, however, utilizes compiled RegFiles[5]. In this DSE process, one of the limitations of using a RegFile is that the IP available to the author was for a single-port instead of a dual-port. Thus, converting the design from a dual-port memory to a single-port requires several modifications in the building blocks to generate their correct functionalities. Therefore, using single-port memory increases the overall clock cycle count, but this will show later in the results section that this increase is beneficial since the improved clock frequency still reduces the overall latency for all SABER operations. Hence, the memory structure is PIP_SP_4(256×64).

4.1.2 Pipelining

Finding an appropriate location for the placement of pipeline registers in a digital design is a critical task. Therefore, a pipeline register is placed at the memory output in the DSE process because evaluating the critical path of several architectures (in Section 4.2) shows that memory is the performance bottleneck of the design. Therefore, in the PIP_DP and PIP_SP architectures in Fig. 18(a), the input to the pipeline register is from the memory while the output is connected to the binomial sampler. The red dotted lines in Fig. 18(a) mean that the pipeline register is connected with the binomial sampler through the FSM controller.

[5]RegFiles are not flip-flops. This vendor-specific terminology for a compiled 6T SRAM memory is advantageous when bit density can be traded-off with performance. Its vendor also calls it a "high-speed" variant of SRAM.

4.1.3 Shared Shift Buffer

Several building blocks of SABER, i.e., AddRound, AddPack, BS2POLVECp, and multiplier, require a shift register to read from many memory addresses and accumulate (hundreds of) bits into local registers. For example, a 320-bit long register is required in AddPack and BS2POLVECp, while a 64 and 676-bit register is required in AddPack and Multiplier blocks, respectively. Note that the SABER building blocks produce outputs serially, so the shift buffer can be shared as there are no concerns with concurrent access. Therefore, a 676-bit register is shared across AddRound, AddPack, BS2POLVECp, and Multiplier blocks. Using a shared shift buffer results in a 10.3% decrease in the total area with no impact on performance. This shared buffer is common in all the optimized designs of Fig. 18(a).

4.1.4 Address Decoder Unit (ADU)

The address decoder unit is only involved in the SABER design of Fig. 18(b), where four instances of smaller memories are utilized and each memory can read/write one 64-bit word in one clock cycle. Recalling again, four memory instances in parallel perform one 256-bit word as read/write in one cycle. Therefore, the ADU selects an appropriate memory to read/write a 64-bit word. Moreover, it also communicates to the SABER controller to pass/collect 64-bit (for SHA3 variants) or 256-bit (for other SABER blocks) data as input/output to/from the SABER core.

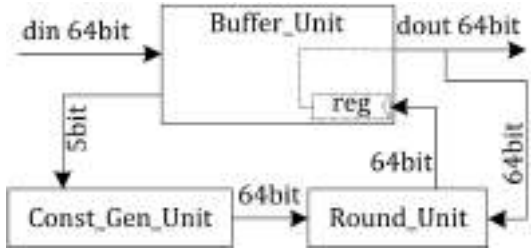
4.1.5 SABER Building Blocks

The blue portion in Fig. 18(a) and Fig. 18(b) shows the SABER building blocks, including the FSM controller to drive the SABER operations (i.e., key generation, encapsulation, and decapsulation). These SABER building blocks can be implemented using different approaches. However, serial and parallel implementations of these building blocks are described below.

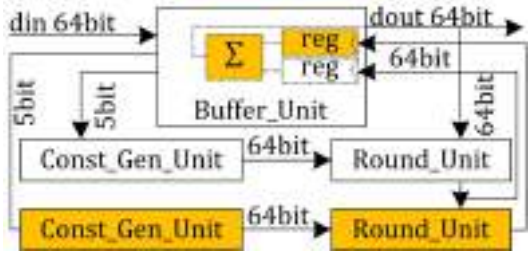
SHA3-256/512 & SHAKE-128. Fig. 18(a) and Fig. 18(b) show that SABER uses SHA3-256 and SHA3-512 hash functions. Moreover, it also uses an EoF (SHAKE-128). These hash and EoF functions use the KECCAK sponge function to compute the 'state permutations'. Hence, in Fig. 18(a) and Fig. 18(b), these hash functions are implemented in a wrapper across a single KECCAK core like implemented in [30]. For Fig. 18(a), an open-source high-speed implementation of the KECCAK core is selected, originally developed by the KECCAK team in [116]. This high-speed KECCAK core computes 'state permutations' iteratively (or in a serial fashion) after every 28 clock cycles; generating 1,344 bits of pseudo-random string in 28 cycles. The serial implementation of the KECCAK core is illustrated in Fig. 19(a). For the parallel SABER architecture of Fig. 18(b), a serial KECCAK core of [116] is modified with orange additional blocks to half the clock cycles, and the update block design of the KECCAK core is shown in Fig. 19(b).

The serial implementation of the KECCAK core of Fig. 19(a) needs an instance each of (i) Buffer_Unit, (ii) Const_Gen_Unit, and (iii) Round_Unit. The initial vectors, intermediate, and final results are kept in the Buffer_Unit. Also, Buffer_Unit holds different counter values for generating KECCAK round vectors. Therefore, Const_Gen_Unit generates 64-bit round vectors based on a 5-bit counter value coming from Buffer_Unit. The Round_Unit specifies the KECCAK sponge function, and its implementation relies on implementing five KECCAK building blocks, i.e., $theta$, pi, rho, chi, and $iota$[6]. Moreover, the Round_Unit takes two 64-bit inputs, one from the

[6]The $theta$, pi, rho, chi, and $iota$ KECCAK building blocks operate on 64-bit width, and



(a) Serial design of KECCAK, implemented in [116].



(b) Parallel design of KECCAK.

Figure 19: KECCAK cores.

Const_Gen_Unit, and another from the Buffer_Unit. It generates a 64-bit vector as output which is further connected as an input to a register inside the Buffer_Unit. This technique takes 28 cycles to serve 24 KECCAK rounds iteratively: 24 cycles are for 24 KECCAK rounds, and an additional 4 cycles determine the 'wait' until the registers in the datapath are free. This serial KECCAK implementation architecture can also be named KECCAK with a single-sponge function.

On the other hand, the parallel implementation of the KECCAK core of Fig. 19(b) can also be named KECCAK with a double-sponge function. It details how the number of clock cycles of the KECCAK core can be reduced by using additional orange-colored boxes. It includes a Buffer_Unit, two blocks of the Const_Gen_Unit and Round_Unit. As the name implies, the Buffer_Unit keeps the initial, intermediate, and final KECCAK results. Like serial (or single-sponge) KECCAK architecture, it also holds counter values for generating round vectors. Therefore, the Buffer_Unit is modified by adding a register and an accumulator. Each register takes a 64-bit vector from the corresponding Round_Unit block, while an accumulator is mandated to produce the final result for the next KECCAK round. Similarly, each block of Const_Gen_Unit takes a 5-bit counter value as input from Buffer_Unit and produces a 64-bit constant vector as an output. Each instance of the Round_Unit (or sponge function) takes two 64-bit inputs and produces a single 64-bit output for the registers in the Buffer_Unit. The first 64-bit input to the corresponding sponge function is from the round constants block. The second 64-bit input to the first sponge function is from the KECCAK buffer (after the accumulation) and its output goes as an input to the second sponge function. This means the sponge functions are connected serially, one after another. The outputs of the first and second sponge functions are connected as inputs to the KECCAK buffer to accumulate the results. Employing double-sponge KECCAK functions, 14 clock cycles

the related mathematical functions to implement these KECCAK building blocks are described in [68].

are (only) required to operate 24 KECCAK rounds. Compared to the SABER FPGA design of [30], the double-sponge function divides the cycle counts by two with area and power overhead.

Binomial sampler. A binomial sampler operates on parameter μ and computes a sample from a μ-bit pseudo-random input string. Let us assume $r[\mu-1:0]$ is a pseudo-random string. Then the sample is computed by subtracting the Hamming weight of the most-significant $\mu/2$ bits from the Hamming weight of the least-significant $\mu/2$ bits, i.e., by computing HW$(r[\mu/2-1:0])-$HW$(r[\mu-1:\mu/2])$, where HW() specifies the Hamming weight. In the SABER PQC protocol, the secret polynomial coefficients are computed from centered binomial distribution using parameters $\mu=10$, $\mu=8$, and $\mu=16$ for LightSABER, SABER, and FireSABER. Hence the secret polynomial coefficients in SABER PQC KEM must be in a range [-5, 5], [-4, 4], and [-3, 3] for LightSABER, SABER, and FireSABER. As the parameter μ is very small in all variants of SABER, it is very simple to implement a binomial sampler using bit manipulations. Therefore, in Fig. 19(a) and Fig. 19(b), the binomial sampler is a combinational block that directly maps the input string into a sample value based on the parameter value μ. A sample is represented as a 4-bit signed-magnitude number for all variants of SABER. In the reference C/C++ implementations of SABER, a sample is represented using 2's complement number system. However, it has been reported in [30] that using a signed-magnitude number system reduces hardware complexity. Therefore, this study also uses a signed-magnitude number system to represent a sample.

In the binomial sampler of Fig. 19(a), two 64-bit words are loaded from the data memory and stored in a 128-bit buffer. After that, for SABER where parameter $\mu=8$, 16 ($128/\mu=8$) samples are generated in parallel and stored in a 64-bit output buffer (samples $\times$ 4, where 4 represent a sample in a signed-magnitude number system for $\mu=8$). Finally, the 64-bit word from the output buffer is written back into data memory. On the other hand, in the binomial sampler of Fig. 19(b), eight 64-bit words are loaded from the data memory and stored in a long 512-bit buffer. Then, for SABER where parameter $\mu=8$, 64 ($512/\mu=8$) samples are generated in parallel and stored in a 256-bit output buffer (samples $\times$ 4, where 4 represent a sample in a signed-magnitude number system for $\mu=8$). Finally, the 256-bit word from the output buffer is written back into data memory. Generating 64 samples in parallel in Fig. 19(b) is beneficial to reduce clock cycles with area and power overhead.

Multiplier. In ideal and module lattice-based cryptosystems, the performance of the polynomial multiplier plays a critical role. Hence, SABER uses power-of-two moduli $p=2^{10}$ and $q=2^{13}$, the fastest native NTT-based polynomial multiplier is not beneficial for SABER. The reference C/C++ implementation of SABER uses Toom-Cook polynomial multiplier, the second fastest multiplier after NTT. The structure of the Toom-Cook multiplier is recursive, and it is difficult to transform into an iterative algorithm. In [86], a hardware implementation of the Toom-Cook multiplier is described for lattice-based cryptosystems where several challenges have been identified when implementing the recursive function calls of the Toom-Cook.

In this thesis, an SBM multiplier is realized for SABER PQC KEM. Recalling once more, SABER involves public and secret polynomials of degree 256. Therefore, a schoolbook multiplier of degree 256 is shown in Algorithm 11.

Algorithm 11: Traditional integer polynomial SBM multiplier for SABER [30].

Input: Polynomial $a(x)$ and $b(x)$ of degree 256
Output: The product of $a(x) \cdot b(x)$ of degree 256

1 $acc(x) \leftarrow 0$
2 **for** $i = 0$; $i < 256$; $i = i + 1$ **do**
3 **for** $j = 0$; $j < 256$; $j = j + 1$ **do**
4 $acc[j] = acc[j] + b[j].a[i]$ mod $\mathbb{Z}_q$
5 $b = b.x$ mod $\mathcal{R}_q$
6 **return** acc

Line one of Algorithm 11 initializes an accumulator buffer with 0. Moreover, this accumulator buffer stores the multiplication result. Line four of Algorithm 11 multiplies the i-th coefficient of $a(x)$ with the j-th coefficient of $b(x)$ followed by modular addition and reduction operations. This is the integer polynomial multiplication of two polynomials of degree 256. After executing the inner for loop, line five of Algorithm 11 multiplies the rotated polynomial $b(x)$ with x in $\mathcal{R}_q$, where $\mathcal{R}_q$ is a ring of the polynomial.

Algorithm 11 describes the traditional way to multiply public and secret polynomials of degree 256; specific to SABER, some optimizations can be made to reduce the complexity of the SBM multiplier. The public polynomial is represented with $a(x)$, and the secret polynomial is shown with $s(x)$. Moreover, the public polynomial multiplications are computed in $\mathcal{R}_p$ and $\mathcal{R}_q$. The coefficients of a secret polynomial are generated from the centered binomial distribution, and depending on the variant of SABER, the secret coefficients are contained in the intervals [-3,3], [-4,4], and [-5,5]. In addition, the modular reduction in p and q is free in SABER as these are in power-of-two. Hence, reduction-free modular multiplication to multiply SABER polynomial coefficients is shown in Algorithm 12 where the coefficient-wise polynomial multiplication is shown using shift and add operations rather than a true integer multiplier. It is important to note that the coefficient of secret polynomial s is in a signed-magnitude form, and the multiplications need to perform only with their absolute values. The accumulator buffer must update by adding or subtracting the results based on the sign-bit of the coefficient of s. As the modulus q is a power of 2 and the coefficients of a are represented as 13-bit numbers, modulus reduction is implicit and requires no additional operation.

Algorithm 12: Coefficient wise shift and add multiplier [30].

Input: $a(i)$ (a 13-bit number) and $s(i)$ (a 3-bit number with $0 \leq s_j \leq 5$)
Output: The product of $a(i) \cdot s(j)$ modulo $q = 2^{13}$

1 $n_0 \leftarrow 0$
2 $n_1 \leftarrow a_i$
3 $n_2 \leftarrow a_i \ll 1$
4 $n_3 \leftarrow a_i + (a_i \ll 1)$
5 $n_4 \leftarrow a_i \ll 2$
6 $n_5 \leftarrow a_i + (a_i \ll 2)$
7 **return** n_k, where $k = s_j$

The overall cost of the multiplier for SABER depends on the computation of the following matrix, where a, s, and r show the coefficients of public, secret, and resultant polynomials. Each row of matrix a contains 256 13-bit polynomial coefficients. Each row

of the matrix s contains 256 4-bit polynomial coefficients. Therefore, 768 coefficients are in three rows of a matrix a and a matrix s.

$$\begin{bmatrix} a_{(0,0)} & a_{(0,1)} & \cdots & a_{(0,255)} \\ a_{(1,0)} & a_{(1,1)} & \cdots & a_{(1,255)} \\ a_{(2,0)} & a_{(2,1)} & \cdots & a_{(2,255)} \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ r_2 \end{bmatrix} \tag{12}$$

In Fig. 18(a), a serial multiplier is implemented using an SBM architecture to compute Eq. 12 for coefficient multiplications of SABER. The serial architecture of the SBM multiplier is illustrated in Fig. 20 where 256 MAC units are employed to implement the matrix of Eq. 12 for coefficient multiplications. Moreover, two buffers (sbuff and abuff) load the corresponding secret and public polynomial coefficients from the external data memory for multiplication. An additional buffer (i.e., accbuff) accumulates the multiplication result. Furthermore, each MAC unit performs coefficient multiplication using Algorithm 12. The multiplication starts with loading 256 secret polynomial coefficients from the first row of Eq. 12 into the corresponding buffer. Then, 256 public polynomial coefficients will be loaded into the corresponding buffer from the first row of Eq. 12. After that, the multiplication will be computed, and the result will be stored in the accumulator buffer (i.e., accbuff). This process repeats twice for the second and third rows multiplication of Eq. 12. Note 256 MAC units are running in parallel in this iterative approach. Each MAC unit takes one clock cycle for singular 13-bit and 4-bit polynomial coefficient multiplication. Thus, the architecture of Fig. 20 takes 768 clock cycles to implement the SABER matrix multiplication of Eq. 12.

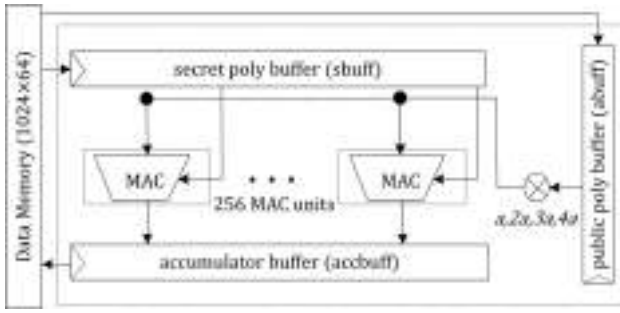


Figure 20: Serial SBM multiplier architecture for SABER coefficients multiplication [86].

Based on the serial SBM multiplier of Fig. 20, a fully parallel SBM multiplier is proposed and the block diagram is shown in Fig. 21, which is utilized for SABER coefficient multiplications in Fig. 18(b).

As shown in Fig. 21, the fully parallelized polynomial multiplication design includes two long polynomial buffers (LPPB and LSPB) and three copies of a schoolbook multiplier, that is, SBM1, SBM2, and SBM3. The length of LPPB and LSPB is proportional to the size of the matrix a and matrix s, respectively. Recalling again that each row of matrix a contains 256 13-bit public polynomial coefficients and each row of the matrix s contains 256 4-bit secret polynomial coefficients. Therefore, matrix a and s contain 768 coefficients in three rows (256 in one row). Then, the length of LPPB is 9984 bits (768×13) and the length of LSPB is 3072 bits (768×4). Multiplication starts with loading 768 polynomial coefficients into LPPB and LSPB buffers.

When loading all the 768 polynomial coefficients into LPPB and LSPB buffers is finished, the corresponding 256 public and secret polynomial coefficients are forwarded

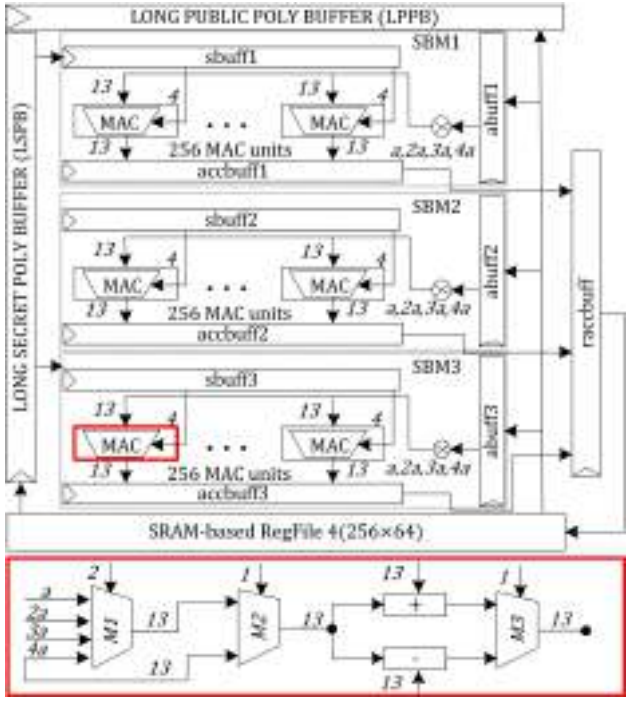


Figure 21: Parallel SBM multiplier architecture for SABER coefficients multiplication.

to multipliers SBM1, SBM2, and SBM3. The design of the SBM1 multiplier is shown in Fig. 21; it contains three buffers (i.e., pbuff1, sbuff1, and accbuff1) and 256 MAC units. The pbuff1 and sbuff1 contain 256 coefficients of the first row of the matrix a and matrix s from Eq. 12 for multiplication. The multiplication computation takes 256 clock cycles having 256 MAC units. Each MAC unit takes two inputs, the size of the first input is 13-bit (public polynomial coefficient), and the size of the second input is 4-bit (secret polynomial coefficient), resulting in a 13-bit polynomial as output, as shown in Fig. 21. A 13-bit output polynomial from each MAC relies on the 4-bit secret polynomial. Two bits from the LSB side of a secret polynomial decide between shifted 13-bit public polynomial coefficients (a, $2a$, $3a$, $4a$) using a multiplexer $M1$. The next (third) bit from the LSB side is a sign bit. The last bit of a secret polynomial coefficient determines the modular addition or subtraction operation to generate a 13-bit multiplication result. Finally, accbuff1 accumulates the multiplication results for the SBM1 multiplier.

Similarly, for SBM2 and SBM3 multipliers, the identical strategy of the SBM1 multiplier is utilized, as shown in Fig. 21. But, in the SBM2 multiplier, pbuff2 and sbuff2 keep the public and secret polynomial coefficients of the second row of the matrix a and matrix s of Eq. 12. Similarly, pbuff3 and sbuff3 hold the public and secret polynomial coefficients from the third row of matrix a and matrix s of Eq. 12. It is essential to note that an additional buffer is also required to accumulate multiplication results from three copies of the used SBM multipliers. Therefore, Fig. 21 shows that an additional 'raccbuff' buffer accumulates the multiplication results from SBM1, SBM2, and SBM3 before writing back on the employed data memory.

In a nutshell, the computational cost of the serial and parallel SBM multipliers of

Fig. 20 and Fig. 21 is 768 and 256 clock cycles, respectively. In Fig. 20, 256 MAC units have been used and these MACs operated iteratively (or serially) to compute the polynomial multiplications of Eq. 12 in 768 clock cycles. The identical strategy is also utilized in schoolbook multipliers of [30, 34, 36] for SABER polynomial coefficients. On the other hand, the parallel SBM multiplier of Fig. 21 utilizes 768 MAC units (running all in parallel) and takes 256 clock cycles to compute the polynomial multiplications of Eq. 12. Despite the computational cost of the coefficients multiplication, the use of a long buffer (i.e., LPPB and LPSB) approach is beneficial to avoid frequent memory access for read/write operations in Fig. 21 because the SABER architecture deals with 256-bit data bus instead of the typical 64-bit size found in the literature (and in Fig. 20). The total clock cycle cost of loading public and secret polynomials from data memory is 156 and 48 for the serial SBM design of Fig. 20. The fully-parallelized architecture of Fig. 21 reduces these costs to 39 and 12 cycles. Apart from the computation cost, the area utilization of the multiplier of Fig. 21 is 3 times the area consumed by the SBM multiplier of Fig. 20.

Other SABER building blocks. The UnPack, AddPack, AddRound, BS2POLVECp, CopyWords, CMOV, and Verify blocks are implemented to deal with the corresponding 64-bit and 256-bit SABER architectures of figures 20 and 21. The objective of the UnPack block in SABER is to transform a byte string into a bit string. The AddPack block performs coefficient-wise addition of a constant with a generated message, which is subsequently packed into a byte string. Similarly, the AddRound block executes coefficient-wise addition of a constant with coefficient-wise rounding. The BS2POLVECp block transforms the byte string into a polynomial vector. CopyWords block in SABER is incorporated in figures 20 and 21 to perform matrix transpose by copying rows in columns and vice versa. The Verify block in the SABER compares two-byte strings of the same length. The output of the Verify block enables the CMOV block to either copy the decrypted session key or a pseudo-random string at a specified memory location.

4.2 Implementation Results

Table 6 provides the implementation results on a 65nm commercial technology for the baseline and optimized architectures. These results are obtained after logic synthesis using Cadence Genus. Column one of Table 6 shows the baseline and optimized designs constructed from variants of Fig. 18. The area information is provided in columns two and three. Similarly, the timing information is presented in columns four and five. From columns six to eleven, power information is provided.

Area and Power Evaluations. The serial SABER architectures of Fig. 18(a) concurrently using compiled memories in a "smart synthesis" fashion with logic sharing to several SABER building blocks and pipelining allows maximizing the clock frequency. Column five of Table 6 shows that this approach enables obtaining a clock frequency of up to $1GHz$ on 65nm process technology with area (column two) and power (columns six to eleven) overheads. Optimizing from the baseline (DP_1(1024×64)) to the PIP_DP architectures revealed that the memory is the primary bottleneck in the SABER implementation. For example, in the case of the baseline design, the total dynamic power consumption of the utilized memory is 44%, while the combinational logic accounts for only 19%. Moreover, an increase in the number of memory instances increases the power consumption of the designs, as noticed in the last column of Table 6 for the PIP_DP_4(1024×16) architecture where four memory instances account for 72% of the total dynamic power and the logic consumes only 10%. Therefore, one approach to overcoming this bottleneck involves using faster memory instances, as

demonstrated by the PIP_SP_4(256×64) architecture, where the combinational logic and the memory account for 23% and 27% of the dynamic power consumption of the SABER architecture. Despite the timing and power results, column two of Table 6 shows that the increase in memory instances also increases the area.

Similarly, for parallel SABER architectures of Fig. 18(b) where four memory instances are running in parallel, the PIP_SP_4(256×64) and SS_Parallel_SP_4(256×64) designs obtain $1GHz$ clock frequency, as shown in column five of Table 6. On the other hand, DS_Parallel_SP_4(256×64) design can operate on a maximum clock frequency of $936MHz$. In addition, implementing SS_Parallel and DS_Parallel designs reveals that the logic consumes more dynamic power than the four memory instances running all in parallel; see columns nine and eleven of Table 6. The potential reasons include the 256-bit data path where all SABER building blocks operate on 256-bit words instead of the SHA3-256/512 and SAHKE wrapper. Another reason is implementing the SABER building blocks using several buffers as utilized in Fig. 19(b) for KECCAK hash computations and Fig. 21 for fully parallelized SBM multiplier. The use of several buffers in SABER building blocks causes to increase in the dynamic power consumption of the sequential logic instead of the combinational, as can see that the SS_Parallel design consumes 76% (for sequential logic or flip-flops), 17% (for combinational logic), and 7% (for four instances of RegFiles running all in parallel) of the total dynamic power consumption. On the other hand, the DS_Parallel design consumes 53% (for sequential logic or flip-flops), 41% (for combinational logic), and 5% (for four instances of RegFiles running all in parallel) of the total dynamic power consumption.

Critical path analysis. On 65nm process technology, the critical paths of the designs of Fig. 18(a) and Fig. 18(b) are presented in Fig. 22. It shows that the memories containing longer access time to read/write one operation result in longer critical paths. Moreover, as seen from DP_1 to PIP_DP designs in Fig. 22, the memory is the real bottleneck, while the use of faster SRAM-based RegFiles results in a shorter critical path, as can be seen for last three designs. In other words, as shown in Fig. 22, the critical path of the baseline architecture (i.e., DP_1) depends on the memory and some amount of combinational logic (to a lesser degree). However, this is not the case for the last three optimized architectures (PIP_SP, SS_Parallel, and DS_Parallel), where the critical path is mostly combinational logic. Also, the critical path of the PIP_SP design contains the setup time of the destination flip-flop, as this design contains a pipeline register between the memories and a binomial sampler.

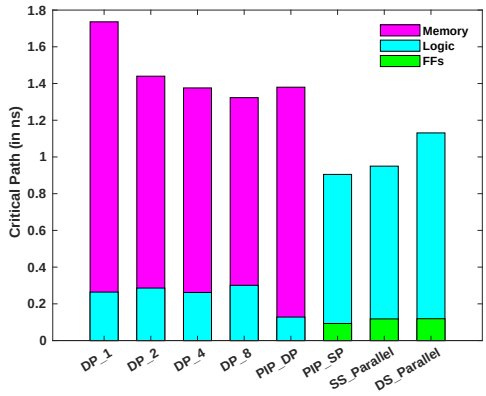


Figure 22: Critical path evaluations of serial and parallel SABER architectures.

Table 6: Results after logic synthesis for serial and parallel SABER PQC KEM on 65nm process technology.

Designs	***Area Results***		***Timing Results***		***Power Information (in mW)***					
	Area (mm^2)	***Gates***	***Clk. P*** (ns)	***Freq*** (MHz)	***Crypto core***		***Comb logic***		***Memory***	
					Lkg	***Dyn***	***Lkg***	***Dyn***	***Lkg***	***Dyn***
Serial SABER designs with 64-bit data path + single-sponge → see Fig. 18(a)										
DP_1(1024×64)	0.299	43336	2.000	500	0.090	86.844	0.059	16.235 (19%)	0.003	38.001 (44%)
DP_2(1024×32)	0.308	45319	1.718	582	0.091	104.835	0.059	18.499 (18%)	0.004	48.322 (46%)
DP_4(1024×16)	0.340	39981	1.638	610	0.082	135.342	0.051	18.762 (14%)	0.006	81.368 (60%)
DP_8(512×16)	0.478	45979	1.624	615	0.099	220.410	0.062	21.691 (10%)	0.010	157.490 (71%)
PIP_DP_4(1024×16)	0.365	46217	1.508	663	0.097	233.361	0.063	20.890 (10%)	0.006	168.476 (72%)
PIP_SP_4(256×64)	0.314	64230	0.998	1002	0.111	142.413	0.074	32.925 (23%)	0.006	39.060 (27%)
Parallel SABER designs with 256-bit data path + single/double-sponge → see Fig. 18(b)										
SS_Parallel_SP_4(256×64)	0.944	199288	0.998	1002	0.412	646.880	0.241	106.457 (17%)	0.006	45.376 (7%)
DS_Parallel_SP_4(256×64)	1.026	237761	1.068	936	0.461	860.504	0.289	354.028 (41%)	0.006	43.020 (5%)

Clk. P: clock period, **Freq:** operating frequency, **Lkg:** leakage power, **Dyn:** dynamic power, **Comb logic:** combinational logic, The flip-flops in SS_Parallel design consume 493.072 dynamic power, which is 76% of the total dynamic power, The flip-flops in DS_Parallel_SP_4(256×64) design consumes 461.628 dynamic power, which is 53% of the total dynamic power.

As observed that the critical paths of the last two designs in Fig. 22 are a bit higher than the critical path of the most-optimized PIP_SP design. One reason is the pipelining in PIP_SP design; another is 64-bit words in the data path, while SS_Parallel and DS_Parallel designs operate on 256-bit words. One more thing is observed that using a single-sponge function (as used in SS_Parallel design) results in a shorter critical path than the DS_Parallel design where two sponge functions are employed in the wrapper of SHA3-256/512 and SHAKE-128 functions. As the critical path of the SS_Parallel and DP_Parallel designs is a bit higher than the PIP_SP design, the SS_Parallel and DP_Parallel designs are beneficial to reduce the clock cycles requirement, which will be discussed next. Overall, the critical path evaluations of SABER designs with different characteristics imply that the last three optimized architectures are saturating the memory bandwidth thanks to the optimization strategies at the architecture and circuit levels.

Clock cycle count. The clock cycles have been calculated from end to end of each operation (KEYGEN, ENCAPS, and DECAPS). Moreover, the computation time needed to perform one cryptographic operation determines the latency, measured in μs, and is calculated using Eq. 13. Therefore, the total clock cycles and latency to compute KEYGEN, ENCAPS, and DECAPS operations of the baseline and optimized designs are provided in Table 7. Column one provides the implemented design, and columns two to four provide the total clock cycles for KEYGEN, ENCAPS, and DECAPS operations. Finally, the last three columns show the latency values.

$$Latency\,(\mu s) = \frac{Total\ Clock\ Cycles}{Clock\ Frequency\ (MHz)} \tag{13}$$

Table 7: Total clock cycles and latency for CCA-secure KEM SABER on a 65nm commercial technology.

Designs	Total Clock Cycles			Latency (μs)		
	KEYGEN	ENCAPS	DECAPS	KEYGEN	ENCAPS	DECAPS
DP_1	5644	6990	8664	11.2	13.9	17.3
DP_2	5644	6990	8664	9.6	12.0	14.8
DP_4	5644	6990	8664	9.2	11.4	14.2
DP_8	5644	6990	8664	9.1	11.3	14.0
PIP_DP	5741	7087	8761	8.6	10.6	13.1
PIP_SP	7154	7136	9359	7.1	7.1	9.3
SS_Parallel	4166	4917	5249	4.1	4.9	5.2
DS_Parallel	3836	4554	4908	4.0	4.8	5.2

For the first six designs, Table 7 shows that the increase in both clock cycles and clock frequency (values given in column five of Table 6) results in a decrease in the computation time, i.e., latency. On the other hand, for the last two designs, Table 7 shows a decrease in both clock cycles and clock frequency (values provided in column five of Table 6), resulting in a decrease in the computation time.

Apart from the total clock cycle information in Table 7, the clock cycle distribution amongst the building blocks of the SABER PQC algorithm is further shown in Fig. 23. Note, this information is only provided for the last three optimized designs: (i) PIP_SP_4(256×64), (ii) SS_Parallel_SP_4(256×64), and (iii) DS_Parallel_SP_4(256×64). Therefore, from left to right, the first row with three

panels in Fig. 23 specifies the KEYGEN, ENCAPS, and DECAPS operations for a serial SABER architecture, i.e., PIP_SP_4(256×64). Similarly, the second row includes three panels for the same three operations on a parallel SABER architecture – SS_Parallel_SP_4(256×64) – with a single sponge in its KECCAK block. DS_Parallel_SP_4(256×64) design in the third row of Fig. 23 has the double-sponge functions. The total clock cycles from the last three rows of Table 7 are presented again in the bottom panel of Fig. 23. In addition, in Fig. 23, hash shows the variants of SHA3 (256/512) and SHAKE-128.

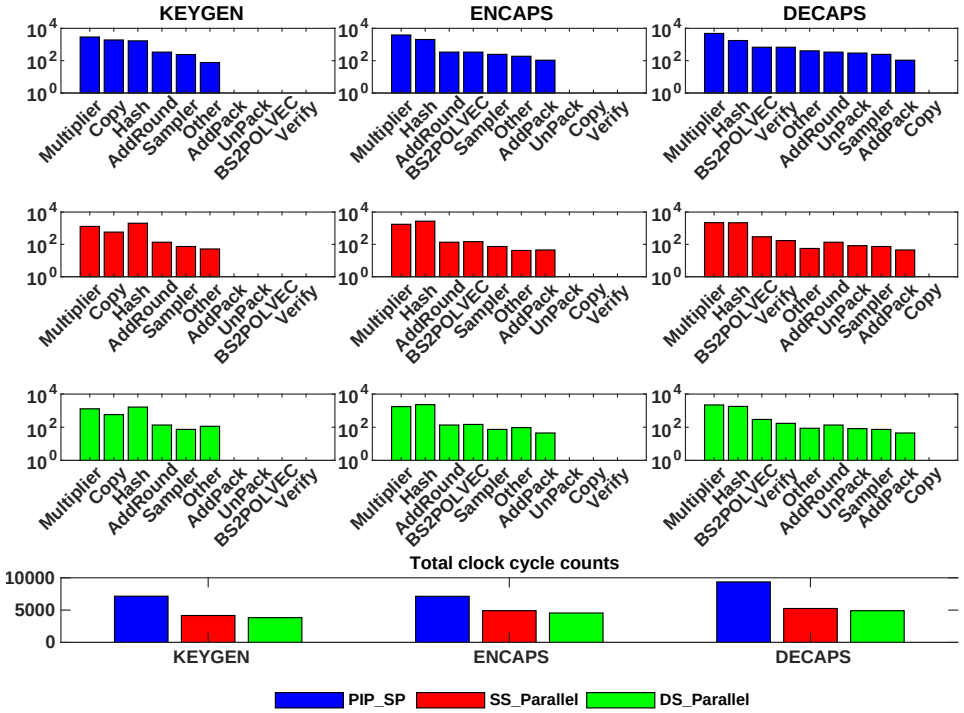


Figure 23: Clock cycle distribution for PIP_SP, SS_Parallel, and DS_Parallel designs.

As expected, Fig. 23 shows a decrease in clock cycles for KEYGEN, ENCAPS, and DECAPS operations when moving from a serial (PIP_SP) to a parallel design with a single-sponge function (SP_Parallel) – see blue and red bars. Similarly, there is a decrease in clock cycles for hash operation when comparing two parallel SABER designs (SP_Parallel and DS_Parallel) with single- and double-sponge functions (see red and green bars). The last panel in Fig. 23 highlights the total cycle count for each operation on the last three optimized architectures. The average number of clock cycles required to execute KEYGEN, ENCAPS, and DECAPS operations using an SS_Parallel accelerator is 1.65× lower than the serial SABER architectures of [34, 36]. The DS_Parallel accelerator design further reduces the clock cycle requirement by 1.07× compared to SS_Parallel architecture.

Until now, the implementation results are presented on a commercial 65nm process technology. Next, the last two optimized (SS_Parallel, and DS_Parallel) parallel designs have also been evaluated on a modern 28nm process technology to investigate the clock frequency, latency, area, power, and energy parameters. After the logic synthesis, the implementation results are given in Table 8. Column one provides the implementation

Table 8: Results of the optimized SABER accelerators on 28nm technology.

Implementation details	***SS_Parallel***	***DS_Parallel***
Maximum Frequency (MHz)	2500	2500
Latency (μs)	1.66/1.96/2.09	1.53/1.82/1.96
Utilized Area (mm^2)	0.251	0.255
Power (Lkg/Dyn) (mW)	10.96/556.25	11.49/597.05
Energy (μJ)	0.923/1.090/1.162	0.913/1.086/1.170

details regarding clock frequency, latency, area, power, and energy. Columns two and three show the corresponding values for SS_Parallel and DS_Parallel designs. By separating with a '/' character, the latency, power, and energy values are given for KEYGEN, ENCAPS, and DECAPS operations of SABER. Similarly, Lkg and Dyn are the leakage and dynamic power consumption. The Vivado IDE tool is used for simulations, while Cadence Genus is used for logic synthesis. In addition, the area and power values are reported directly from the synthesis tool; latency values are calculated using Eq. 13; energy values are calculated using Eq. 14.

$$Energy(\mu J) = Dynamic\ Power\ (mW) \times Latency\ (\mu s) \tag{14}$$

Table 8 shows that the SP_Parallel and DP_Parallel designs can operate on 2500MHz clock frequency. DS_Parallel design reduces the computation time for the SABER's KEYGEN, ENCAPS, and DECAPS operations compared to SS_Parallel design. The reason is the single-sponge function in SS_Parallel design while double-sponge functions in DS_Paralel design, comparatively, the double-sponge functions minimize the clock cycle. Although DS_Parallel design reduces the computation time, on the other hand, it increases by $+4.63\%$ and $+6.84\%$ for leakage and dynamic power, and $+1.57\%$ increase in area. The max frequency is obtained by pushing the timing constraint until the slack is close to zero. Apart from the area and power increase, the DL_Parallel design has higher merit as it consumes nearly the same energy as the SS_Parallel design.

In summary, a significant improvement in clock cycles, latency, area, power, and energy when moving from a serialized design to parallel architectures reveals that the realized approaches (of Fig. 18, Fig. 19 and Fig. 21) can be utilized in other PQC algorithms for optimizations.

4.3 Comparison and Discussion

The previous section describes the DSE process, where one baseline and seven optimized SABER designs have been constructed. Therefore, comparing all the baseline and optimized designs is challenging with existing SABER architectures; however, the most optimized (three) designs are selected to compare to state-of-the-art architectures: PIP_SP, SS_Parallel, and DS_Parallel. Table 9 shows the comparison after logic synthesis to existing FPGA and ASIC SABER implementations. Also, the NIST-selected CRYSTALS-Kyber PQC KEM to be standardized in the near future is compared. Column one provides the reference design, while the implementation platform (FPGA/ASIC) is shown in column two. The computational cost in latency for KEYGEN, ENCAPS, and DECAPS operations is presented in column three. In addition, the latency values are separated by the character '/'. The operating frequency in MHz is reported in column four. Finally, the last column shows the area (LUT/FF for FPGA and ASIC in mm^2).

A symbol '–' is placed in Table 9 where the relevant information is not reported in the reference designs.

Table 9: ASIC and FPGA comparison to existing PQC KEM SABER and CRYSTALS-Kyber hardware accelerators after logic synthesis. All implementation results are for security equivalent to AES-192.

Ref. #	***FPGA/ASIC***	***Latency (μs)***	***Freq. (MHz)***	***Area LUT/FF (or) mm^2***
SABER Implementations				
[30]	Ultrascale+	21.8/26.5/32.1	250	23.6K/9.8K
[31]	40nm	2.66/3.64/4.25	400	0.38
[84]	Artix-7	–/373.1/422.1	125	6.7K/7.3K
[86]	Artix-7	3.2K/4.1K/3.8K	125	7.4K/7.3K
[88]	Ultrascale+	–/60/65	322	–/–
CRYSTALS-Kyber Accelerators (Kyber-768)				
[26]	28nm	4.5/5.6/6.9	2000	0.263
[117]	Artix-7	209	115	16K/6K
[118]	Artix-7	499.8 (ENCAPS)	155	97K/153K
[118]	Artix-7	658.7 (DECAPS)	155	110K/167K
[119]	Virtex-7	39 (KEYGEN)	217	22K/12K
[119]	Virtex-7	57.5 (ENC+DEC)	226	29K/22K
[120]	65nm	35/50/70	200	372KGE
PIP_SP	65nm	7.1/7.1/9.3	1000	0.314
SS_Parallel	65nm	4.1/4.9/5.2	1002	0.944
DS_Parallel	65nm	4.0/4.8/5.2	936	1.026
SS_Parallel	40nm	2.4/2.9/3.0	1694	0.846
DS_Parallel	40nm	3.4/4.1/4.4	1095	0.767
SS_Parallel	28nm	1.6/1.9/2.0	2500	0.251
DS_Parallel	28nm	1.5/1.8/1.9	2500	0.255

ENC and DEC: represents encryption and decryption operations, **ENCAPS and DECAPS:** shows encapsulation and decapsulation operations, **GE:** specifies the gate equivalents, **Ref [117]:** reports the latency for ENCAPS + DECAPS and KEYGEN can be executed offline, **Ref [118]:** instead of FFs, slices are reported as area, **Ref [119]:** the reported results are for Kyber-1024 (security equivalent to AES-256), **Ref [120]:** the latency values are calculated using the ratio of clock cycles (mentioned in the reference paper) with frequency $200MHz$.

Comparison to SABER hardware accelerators. As shown in Table 9, the FPGA implementations are reported in [30, 84, 86, 88] while an ASIC SABER implementation after logic synthesis is described in [31]. As mentioned before, the objective of the DSE process was to improve the operating frequency of the lattice-based PQC hardware accelerators specific to the ASIC platform. As can see in column four of Table 9, the highest operating frequency on FPGA devices is $322MHz$ which is obtained in [88]; on the other hand, the frequency obtained for ASIC on 40nm technology is $400MHz$ and is achieved in [31]; comparatively, on different ASIC platforms (65, 40, 28nm), the operating frequency reported in column four of Table 9 for PIP_SP, SS_Parallel and DS_Parallel designs is very high. More precisely, on an identical 28nm process technology, the implemented SS_Parallel and DS_Parallel designs are 4.23× and 2.73× faster than the ASIC implementation of [31]. Similarly, the implemented SS_Parallel and DS_Parallel designs are 7.76× faster than the fastest FPGA implementation of [88].

Instead of the modern 40 and 28nm process technologies, more insight comparisons are given below on 65nm technology, highlighting the significance of the DSE process.

Let us consider only the SABER FPGA designs for comparison, but before comparing results, it is essential to highlight that a realistic comparison to FPGA devices is difficult as the implementation platforms differ. In terms of computation time or latency (shown in column three of Table 9), the most efficient implementation of SABER on FPGA is described in [30], where the design takes 5453, 6618 and 8034 clock cycles for one KEYGEN, ENCAPS, and DECAPS operations computation. Their design utilizes a co-processor architecture style where all building blocks of SABER operate serially. The PIP_SP architecture uses the same serial strategy for execution; therefore, the PIP_SP accelerator takes 7154, 7136, and 9359 clock cycles to compute one KEYGEN, ENCAPS, and DECAPS operation. Indeed, the PIP_SP design utilizes more clock cycles; on the other hand, the PIP_SP accelerator on 65nm process technology requires 3.07× (for KEYGEN), 3.73× (for ENCAPS), and 3.45× (for DECAPS) lower latency than [30]. This is due to the higher operating frequency of $1GHz$ in PIP_SP, which is obtained by employing a pipeline register in the data path of the SABER crypto core, which reduces the critical path. Another reason is using four smaller distributed single-port RegFile memories in PIP_SP accelerator architecture while a singular dual-port BRAM of size 64KB is used in [30]. In PIP_SP accelerator architecture, the same memory size is utilized. The area comparison is hard as the implementation platforms are different.

In [84], a lightweight hardware implementation of SABER uses a masking technique to protect against side-channel attacks. Initially, a lightweight hardware architecture is designed as a baseline, and after that, countermeasures are incorporated for side-channel attack protection. The authors claimed to have the first secure hardware-protected implementation of SABER, outperforming previously reported secure software and software/hardware co-design implementations. The area and latency results of the unprotected SABER implementation of [84] are shown in Table 9, where the utilized LUT and FF are 6713 and 7363. In addition, the utilized slices and DSP blocks (not shown in Table 9) are 2631 and 32, respectively. If considering the protected SABER implementation, the values for LUT, FF, slices, and DSP blocks are 19299, 21977, 7036, and 64. The unprotected and protected SABER designs do not utilize the BRAMs and operate on identical $125MHz$ frequency. Despite the area, the computational cost (latency) of the unprotected SABER designs is 373.1μs (for ENCAPS) and 422.1μs (for DECAPS). Similarly, the computational cost of the protected SABER designs is 576.0μs (for DECAPS). On modern Artix-7 FPGA device, adding side-channel countermeasures to unprotected SABER design of [84] results in a 2.87× (ratio of 19299 with 6713) increase in the number of LUT and a 1.4× increase in latency. When comparing the unprotected SABER design of [84] with PIP_SP architecture on 65nm process technology, the PIP_SP design takes 52.54 and 45.38 times lower latency for ENCAPS and DECAPS operations. The cause is the higher operating frequency of $1GHz$ for PIP_SP, while in the reference design, the obtained circuit frequency is $125MHz$ on Artix-7 FPGA. Also, this comparison shows that the PIP_SP design is 8× faster (in operating frequency) compared to unprotected and protected SABER implementations of [84].

Similar to other SABER designs, in [86], a hardware-software co-design approach is utilized to implement SABER design. The authors have operated SABER operations on an ARM core, and only the most computationally intensive polynomial multiplication operation is tasked to the coprocessor, resulting in a compact design. They utilized a distributed computing concept at the micro-architectural level, where different algo-

rithmic optimizations have been performed, resulting in a speedup of approximately six times compared to optimized-only software implementation with a minor increase in hardware cost. The Zynq-7000 FPGA SoC is used for hardware deployments. On 65nm technology, for KEYGEN, ENCAPS, and DECAPS operations, the PIP_SP architecture utilizes 450.7×, 577.4×, and 408.6× lower latency, when compared to [86]. Moreover, like [84], the PIP_SP architecture is 8× faster in clock frequency. The area comparison is challenging as the implementation platform in this study is ASIC, while FPGA implementation is provided in the reference design.

Another co-processor-based SABER design is described in [88], where the implementation and benchmarking of three lattice-based KEM algorithms, including SABER, have been presented. Compared to pure-software-based implementations, the SABER co-processor on the Ultrascale+ platform results in 28× (for ENCAPS) and 20× (for DECAPS) speed-ups. Therefore, compared to [88], the PIP_SP architecture on 65nm technology results in a 3.10× speed-up in clock frequency. As for as the latency is concerned for comparison, the PIP_SP design is 8.45× and 6.98× faster than [88]. The optimized latency in the PIP_SP accelerator is achieved due to pipelining and by employing variants of four smaller SRAM-based RegFile memory instances.

Now let us see the ASIC SABER implementation of [31]. It describes an energy-efficient configurable crypto-processor architecture supporting multiple security levels of SABER. It incorporates an 8-level Karatsuba multiplier to perform coefficient-wise multiplications. Moreover, an optimized hardware-efficient Karatsuba scheduling strategy is implemented for a pre-/post-processing structure of the Karatsuba, which reduces the area overheads. The SABER design of [31] takes 1066, 1456, and 1701 clock cycles for KEYGEN, ENCAPS, and DECAPS operations. On TSMC 40nm process technology, the utilized area of the SABER design of [31] is $0.38mm^2$ (shown in the last column of Table 9). Comparatively, on 65, 40, and 28nm process technologies, the PIP_SP, SS_Parallel, and DS_Parallel designs are faster than [31] in operating frequency because in the serial PIP_SP architecture pipelining reduces the critical path which eventually improves the operating frequency. The SS_parallel and DS_Parallel designs utilized a wider 256-bit data path strategy, reducing the clock cycles. As shown in column three of Table 9, the serial and parallel (PIP_SP, SS_Parallel, and DS_Parallel) designs on 65nm technology take higher computation time for KEYGEN, ENCAPS, and DECAPS operations because the design of [31] uses two-sponge functions in the KECCAK core to reduce the clock cycles and another reason is the use of several smaller memories specific to SABER operations. On the other hand, the SS_Parallel and DS_Parallel designs on identical 40nm and even on modern 28nm technologies take much lower computation time than [31]. The reason is four smaller memories running all in parallel allows to deal with a wider 256-bit data path in SS_Parallel and DS_Parallel designs, whereas in [31], a 64-bit data path is implemented even with several smaller memories. Hence, using four smaller memories running all in parallel in SS_Parallel and DS_Parallel design is more efficient than [31] as this strategy reduces the clock cycles, improves the operating frequency, and the ratio of clock cycles with the operating frequency allows to obtain lower computation time.

Comparison to CRYSTALS-Kyber hardware accelerators. The point-to-point comparison is challenging as the PQC schemes are different. Moreover, the area comparison to FPGA devices is difficult as the implementation platforms differ. Hence, this comparison shows the significance of several optimization techniques utilized in executing the DSE process for optimizing the performance of the SABER PQC algorithm. In addition, it highlights the importance of the techniques used in SABER

for optimizing other lattice-based NIST-selected PQC algorithms such as CRYSTALS-Kyber, CRYSTALS-Dilithium, etc. Note that these optimization techniques are not only specific to lattice-based PQC algorithms but can be used for other cryptographic applications for different purposes. For instance, one-time data loading from memories and a wider data path of 256-bit strategies are beneficial to reduce clock cycles. Below, the comparison to some CRYSTALS-Kyber hardware accelerators is compared to the last three optimized SABER designs.

As for as the ASIC implementations are concerned for comparison, in [26], a unified architecture (named KaLi) is described to perform KEYGEN, ENCAPS, DECAPS, SIGNGEN (signature-generation), and SIGNVRF (signature-verify) for all three security levels of CRYSTALS-Kyber, and CRYSTALS-Dilithium PQC algorithms. On modern 28nm ASIC technology, KaLi can operate at a maximum of $2GHz$ clock frequency. Comparatively, on identical 28nm ASIC technology, the SS_Parallel and DS_Parallel designs can operate on $2.5GHz$. As shown in column three of Table 9, the PIP_SP design on 65nm technology takes more computation time than the design of [26] on 28nm technology. On the other hand, the parallel SABER designs take less computation time than the KaLi design of [26]. The area reported for KaLi is higher than this work because KaLi utilizes unified accelerator architecture for CRYSTALS-Kyber and CRYSTALS-Dilithium algorithms. On 65nm process technology, an ASIC implementation of CRYSTALS-Kyber is described in [120] where a maximum $200MHz$ operating frequency is achieved. On the same 65nm technology, the PIP_SP, SS_Parallel, and DS_Parallel designs can operate on higher $1GHz$, $1GHz$, and $936MHz$ clock frequencies. As can be observed from column three of Table 9, the optimized SABER designs (in this study) take much less time in latency than CRYSTALS-Kyber implementation of [120]. Similarly, in column five of Table 9, the area for reference design is given in gate equivalents (which is 372K); however, the gate equivalents for implemented SABER optimized PIP_SP, SS_Parallel, and DS_Parallel designs is 64.2K, 199.2K, and 237.7K. These values are much lower than the 372K. Hence, the parallel designs of this study are more efficient in operating frequency and latency compared to ASIC accelerators of CRYSTALS-Kyber of [26, 120].

The FPGA-implemented designs of CRYSTALS-Kyber are reported in [117, 118, 119]. The designs of [117] and [118] generate the KEYGEN offline while the remaining two operations, i.e., ENCAPS, & DECAPS, are executed on an FPGA device. Moreover, the implementation of these two designs is different than each other. For example, in [117], a unified design is presented for ENCAPS and DECAPS operations. The design of [118] includes two implementations, one for each ENCAPS and DECAPS; these two designs operate on identical $155MHz$ clock frequency (see column four of Table 9), but the hardware utilization costs are different (97K LUTs for ENCAPS and 110K LUTs for DECAPS – see column five of Table 9). The design of [119] is different than the accelerators of [117, 118], as it implements all three operations (KEYGEN, ENC, and DEC) on hardware; the KEYGEN design is dedicated while a unified implementation is presented for ENC (encryption) and DEC (decryption) operations. Due to different platforms, the latency and area parameters are difficult to compare with the ASIC-implemented designs of this study. However, it can be seen from column four of Table 9 that the optimized SABER implementations of this study are faster in clock frequency as compared to recent CRYSTALS-Kyber implementations. The reasons are the smaller memories running in parallel and the larger data path size of 256-bit (for parallel designs). Apart from these techniques, pipelining, shared shift buffers across several building blocks, and one-time data loading from the memories are the additional

approaches that help to obtain higher circuit frequency and reduce clock cycles. The studied approaches in the DSE process are not specific to SABER and PQC algorithms; these can be utilized in other related applications to optimize the operating frequency.

In summary, the premise of the DSE process was to optimize the operating frequency of the PQC algorithms when demonstrated on the ASIC platform. Consequently, after applying several optimization techniques, column four of Table 9 shows that the serial and parallel designs (of this study) are faster in operating frequency compared to state-of-the-art PQC accelerators with an additional area overhead.

5 High-Speed SABER Chip Design

This chapter concentrates on the silicon implementation of SABER designs from the ones studied in the Chapter 4. Hence, an optimized PIP_SP serial design is considered for fabrication. Initially, the chapter describes the chip architecture, including the top wrapper, serial input/output interface, and SABER crypto core, along with the building blocks in Section 5.1. Section 5.2 details the measurement results, including the chip layouts, experimental setup, and the range of operations the fabricated chip supports. In Section 5.3, the performance of the fabricated SABER chip is compared to existing SABER silicon-proven implementations.

5.1 Chip Architecture

Fig. 24 shows the top-level architecture of the fabricated SABER chip. It includes a wrapper, a serial-in/out interface, and the SABER crypto core. The wrapper acts as a controller to operate the required cryptographic operations. As the name implies, serial-in/out bears inputs serially from outside to the chip and also results in a serialized output. The SABER crypto core is responsible for the computations of corresponding operations such as KEYGEN, ENCAPS, and DECAPS. The upcoming sections provide the architectural details of the wrapper, serial-in/out interface, and SABER crypto core.

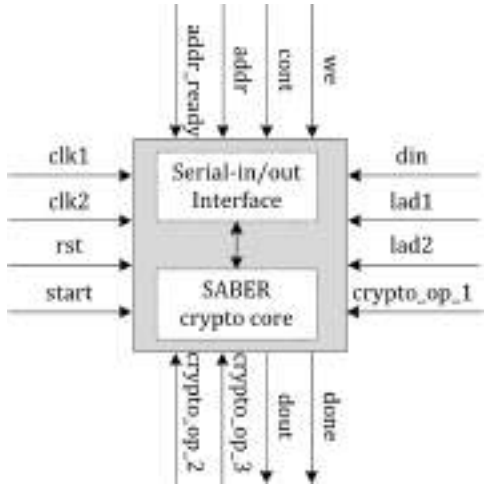


Figure 24: Top-level architecture of the SABER chip, where gray portion specifies the wrapper.

5.1.1 Wrapper

Fig. 24 illustrates that the chip's interface comprises 16 I/O pins, each capable of handling a single bit. The input pins consist of $clk1$, $clk2$, rst, $start$, we, $cont$, $addr$, $addr\_ready$, din, $lad1$, $lad2$, $crypto\_op\_1$, $crypto\_op\_2$, and $crypto\_op\_3$, whereas the output pins are $dout$ and $done$.

As the objective of this study is to operate the SABER crypto core at a high frequency, it becomes difficult to communicate with the outside environment at a high frequency. Therefore, two different clocks (named $clk1$ and $clk2$) are utilized. The $clk1$ pin drives a slower clock that feeds the serial I/O interface of the chip. Similarly, $clk2$ drives the faster clock connected to the inner SABER crypto core. The names of various other I/O pins are intuitive: rst is a reset signal, $start$ is a trigger signal for starting

cryptographic operations, we is a write-enable, din is data in, $dout$ is data out, $addr$ specifies read/write address. The pins $addr\_ready$ and $done$ inform when operations are finalized, either loading an address or an entire crypto operation.

The objective of using a $cont$ pin is to measure the chip's power consumption when the KEYGEN, ENCAPS, and DECAPS operations are executed continuously (i.e., in an infinite loop). Doing so will ensure that the power measurement is not affected by I/O limitations.

The combined use of $lad1$ and $lad2$ allows to drive four possible combinations: (i) 2'b00 means "no-operation", (ii) 2'b01 means load read/write address on the chip using $addr$, (iii) 2'b10 means load input data vector from outside on the chip using din, and (iv) 2'b11 means reading data back from the chip on $dout$. The $crypto\_op\_1$, $crypto\_op\_2$, and $crypto\_op\_3$ signals are used to select the crypto operation, either KEYGEN, ENCAPS, or DECAPS.

The wrapper of the chip is an FSM-based dedicated controller. It is responsible to execute the KEYGEN, ENCAPS, and DECAPS operations by properly orchestrating the sequential use of the SABER blocks. The chip remains in an IDLE mode until the $start$ signal is asserted. Next, based on the values of $crypto\_op\_1$, $crypto\_op\_2$, and $crypto\_op\_3$, the FSM begins to execute the corresponding sequence of instructions for computation of KEYGEN, ENCAPS, and DECAPS operations. When the required KEM operation completes its execution, the FSM returns the chip into an IDLE mode (if $cont$ is 0, otherwise the operation is continuously executed non-stop when $cont$ is 1).

5.1.2 Serial-in/out interface

Fig. 25 depicts the architecture of the serial-in/out interface, which bridges the external environment and the SABER crypto core. This interface helps data loading via serial communication and serves two purposes: loading user-defined inputs din and $addr$ & chip debugging. The serial interface can access the entire memory addressing space (1024×64) to store or retrieve data from the memories. The incoming bits are accumulated into vector lengths of 10 bits for read/write addresses and 64 bits for read/write data to load user-defined inputs. Three shift registers are used: (i) one for read/write address, (ii) one for data input, and (iii) one for data output, as shown inside the highlighted circle in Fig. 25 to accumulate $addr$ and read/write data bits. In addition, an 8-bit $count$ register counts up to 10 for loading read/write addresses and up to 64 for read/write data. All these shift registers operate based on the values of $lad1$ and $lad2$ signals.

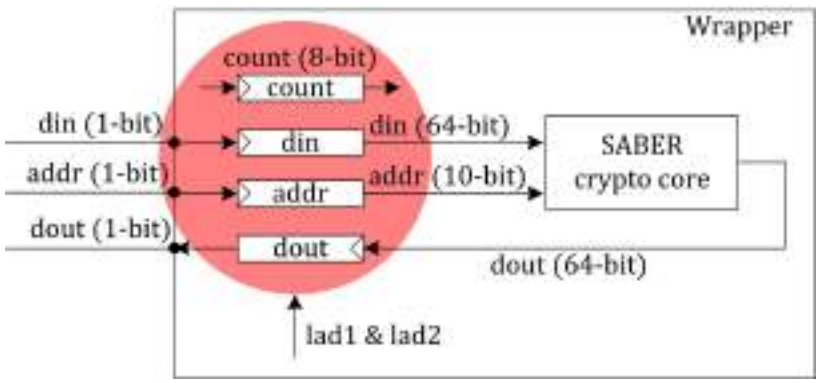


Figure 25: Design for serial-in/out interface.

The $addr$, din, and $dout$ pins of the fabricated chip are linked to the corresponding read/write address, data input, and data output shift registers, respectively. Recalling again, the values on $lad1$ and $lad2$ are used to route the corresponding shift register bits to the appropriate pins.

5.1.3 SABER crypto core

The SABER crypto core corresponds to the PIP_SP architecture and is capable of computing KEYGEN, ENCAPS, and DECAPS operations. As shown in Fig. 26, it consists of several blocks, i.e., a data memory, a routing network, a pipeline register, a shared shift buffer, building blocks, and a dedicated controller. The corresponding details of these blocks are given below.

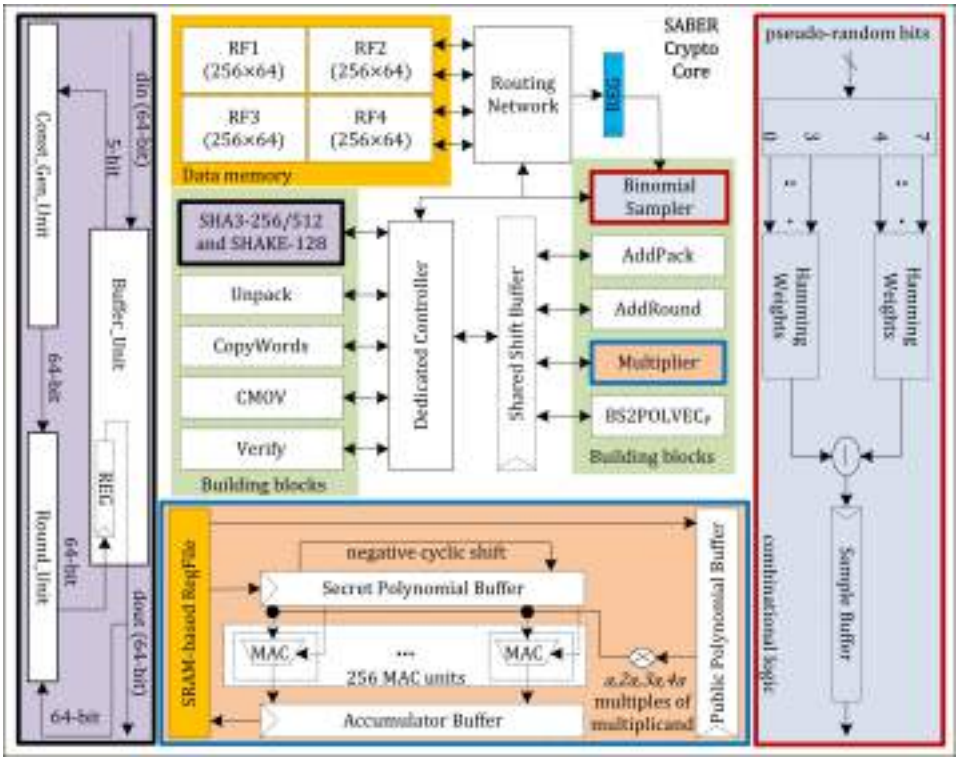


Figure 26: SABER crypto core.

To implement the SABER as a hardware accelerator, the FPGA-based SABER design of [30] utilizes a BRAM-based dual-port data memory of size 1024×64. In the previous chapter, a DSE process was presented. It has been determined that the smaller and distributed memories in an ASIC design are more beneficial as the smaller memories require simpler address decoder units which are faster and lead to performance improvements with area and power overheads. Therefore, as shown in Fig. 26, the SABER crypto core utilizes four instances of 256× 4 size of a single-port SRAM-based RegFile as a data memory to retain initial, intermediate, and final results during and after the computations. The total size of the four memory instances is (256× 4) × 4 = 65Kbits.

The proposed SABER crypto core splits the memory address space into multi-

ple blocks. However, each memory block requires different signals for write enable, read/write address, and input/output data. Hence, a unified routing network is essential for communication between the SABER building blocks and memory instances. Therefore, the routing network of the SABER crypto core includes several multiplexers that handle the corresponding memory instances during read and write operations. In the DSE process, it has been described the use of smaller memories in serial and parallel fashions. Note that the SABER crypto core routing network in fabricated chip handles four memory instances sequentially. This means only one memory instance can read/write at one time.

The hardware implementations of the building blocks of the PQC protocols require several shift registers for different purposes, such as shift and accumulation. For example, many SABER building blocks need shift registers with different lengths to acquire data from many memory addresses and then accumulate into local registers/buffers for computations. For instance, a 320-bit register is required in AddRound and BS2POLVECp, while a 64 and 676-bit register is needed in AddPack and multiplier, respectively. Using different buffers in different building blocks results in higher hardware resources and consumes more power. Therefore, a better solution is to use a single shared buffer. The difficulty is in determining an appropriate length for such a buffer. As SABER requires polynomial multiplications over 256 13-bit coefficients, a serialized architecture is more beneficial to load some partial coefficients for multiplication and then load the subsequent coefficients. In the SABER crypto core, 52 13-bit polynomial coefficients are loaded at first in a 676-bit buffer for multiplications. After that, the next coefficients are loaded for multiplications, and so on, until the completion of 256 coefficients. Consequently, a single 676-bit register is shared across AddRound, AddPack, BS2POLVECp, and multiplier blocks in the SABER crypto core to save area without degrading the performance of the crypto core.

The green portion in Fig. 26 highlights the SABER building blocks: (i) variants of secure hash algorithms (i.e., SHA3-256, SHA3-512, and SHAKE-128); (ii) Unpack; (iii) CopyWords; (iv) CMOV; (v) Verify; (vi) Binomial sampler, (vii) AddPack, (viii) AddRound, (ix) Multiplier, and (x) BS2POLVECp. In Chapter 4, the serial and parallel implementation of these SABER building blocks is already described. Recalling again, PIP_SP architecture is fabricated on 65nm technology. Therefore, all the SABER building blocks in the fabricated chip support serial implementation.

Despite the SABER building blocks, the crypto core has a dedicated controller. Therefore, based on the instructions from the wrapper for the computation of KEYGEN, ENCAPS, and DECAPS, the controller generates the corresponding control signals to execute the SABER building blocks one at a time; this means that the fabricated chip incorporates a serial architecture instead of the parallel design. Moreover, it controls the use of the shared shift buffer and the routing network. As shown in Fig. 26, the binomial sampler is connected through a pipeline register; this creates NOP (no-operation) or execution bubbles. Hence, the controller also manages the synchronization between the building blocks.

5.2 Measurement Results

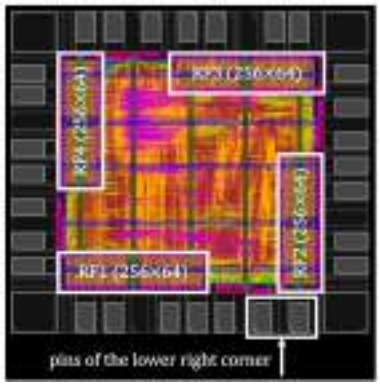
A $65nm$ CMOS (Complimentary Metal Oxide Semiconductor) technology is used for silicon demonstration of the proposed SABER architecture. The RTL code is written in Verilog HDL. To generate the netlist, the top-level SABER design was synthesized using Cadence Genus and a foundry-provided $65nm$ standard cell library. After that, the generated netlist was loaded for physical implementation in Cadence Innovus. For

physical verification (DRC and LVS), Calibre was used. Later on, the GDSII file was submitted to the foundry for fabrication. A total of one hundred chips were fabricated and twenty-five were packaged in a Dual-In-Line-28 (DIP-28) form factor. It is important to provide that the design implementation was completed in August 2021, the chip underwent fabrication in the time frame from September–November, and fabricated parts were delivered in December 2021. Finally, the testing and measurement results were finished in February 2022.

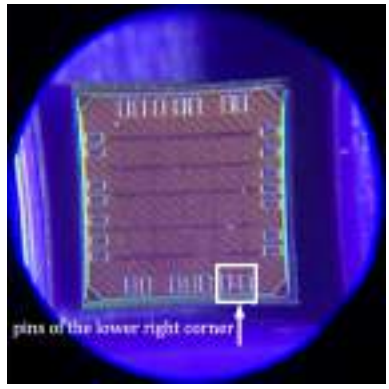
The chip layouts and the experimental setup, including the leakage current measurement, area, timing & power results, and an operational range of the chip, are described in the following sections.

5.2.1 Chip Layouts and Experimental Setup

The chip layout is shown in Fig. 27(a), where the four memory instances are highlighted around the corners across the core. All metals between M2 and M7 were used for signal routing. M7 is also used for creating a power ring around the core. Moreover, the power is distributed across the core using horizontal and vertical stripes in M8 and M9 (and these stripes are visible in Fig. 27(a)). The die size is $960\mu m \times 960\mu m$. The SABER design barely fits in this size. The placement density of the core area is 93.4%, with the remaining 6.6% occupied by decap and filler cells. This high density made the SABER design very challenging for timing closure. Moreover, the I/O pins (seven on each side of the chip) and power stripes routed across the entire chip, horizontally and vertically, are visible. Similarly, a micrograph of an unpackaged chip taken by a microscope is illustrated in Fig. 27(b). It is possible to recognize the same power routing stripes and IOs as in the physical layout. Fig. 27(a) and Fig. 27(b) highlight pins of the chip's lower right corner for orientation.



(a) Physical layout of the SABER chip taken from Cadence Innovus.



(b) Microscope view of an unpackaged die where it is possible to identify the IOs (7 on each side) and horizontal & vertical power stripes on the top metal layers.

Figure 27: Physical layout and microscope view of the fabricated SABER chip.

The utilized testing setup to validate the chip is shown in Fig. 28. A printed circuit board (PCB) was designed using KiCAD and fabricated to facilitate the test and enable measurements. A DIP-28 socket is mounted on the PCB. Therefore, the packaged chip is placed on the DIP-28 socket. Two power sources are connected to the PCB through BNC connectors, where the core logic (1.2V) and IO cells (2.5V) are powered

through the PCB. Small decoupling capacitors are manually mounted on the PCB for both VDDs. The STM32F446RE [121] microcontroller is integrated with the PCB to drive all the input signals except the faster clock (i.e., $clk2$). The microcontroller also collects the outputs of the chip. To generate the fast $clk2$, a high-frequency generator (shown between the two power sources in Fig. 28) is used. Note that the fabricated SABER chip does not contain an internal clock generator.

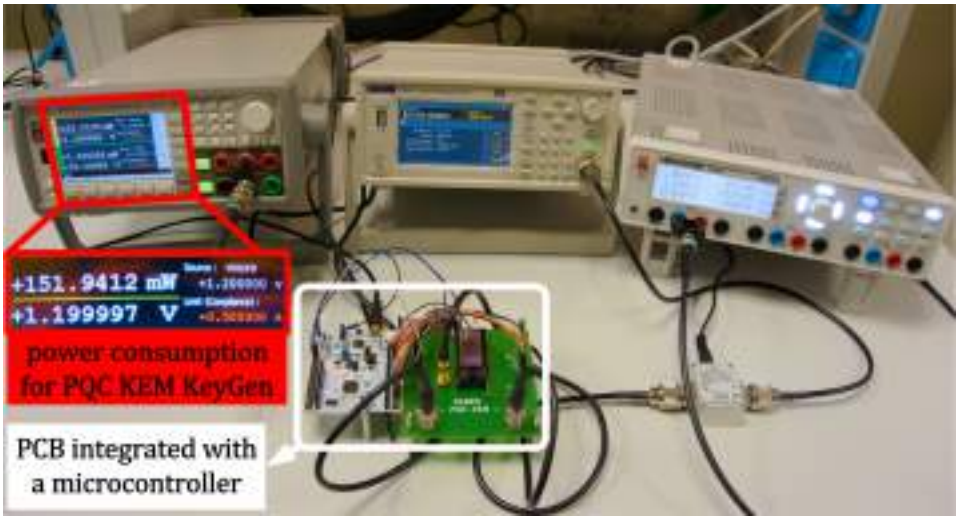


Figure 28: Testing setup used to validate the fabricated SABER chip.

5.2.2 Leakage Current Measurement

The plot of a normal distribution of the average leakage current measurements of the twenty-five packaged chips is shown in Fig. 29. Reminding that the leakage (or state-off) current is the current that flows through a device even when the device is not actively computing. The average leakage current is 0.2099mA and the standard deviation is 0.0409. The measured data points are plotted as red circles over the normal distribution (black line). The pre-silicon leakage current results (obtained from Innovus) for three different corners, i.e., typical, worst, and best, are 0.164mA (on 1.2V), 0.450mA (on 1.08V) and 3.20mA (on 1.32V) and these values are relative to temperatures of $25°C$, $125°C$ and $0°C$, respectively. The blue vertical line in Fig. 29 shows a typical corner's pre-silicon leakage current value. The measurement results appear slightly more pessimistic than the simulated value predicted but within the expected range. The best and the worst measured data points are also highlighted in Fig. 29.

5.2.3 Area, Timing and Power Results

The identified 'best case' sample is placed on the PCB to identify the chip's highest possible operating frequency and power consumption. All the KEM operations of SABER (i.e., KEYGEN, ENCAPS, and DECAPS) can be executed on 770, 715, and 840MHz at 1.2V supply voltage. The corresponding power values on identical operating conditions for KEYGEN, ENCAPS, and DECAPS are 151, 158, and 157mW. These power values are obtained using a high-precision measurement unit. Here, for the KEYGEN operation of SABER, the obtained 151mW value is highlighted with a red portion in Fig. 28.

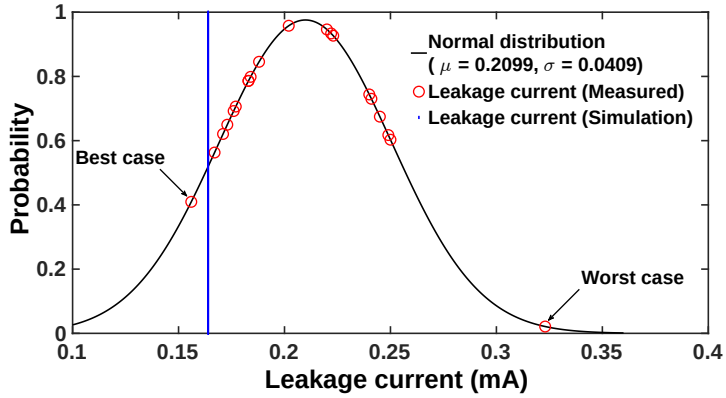


Figure 29: Average leakage current measurement plotted as a normal distribution. Each red circle corresponds to a single sample or chip. The best and worst values are also highlighted.

Therefore, it has been identified that the 715MHz is the optimal clock frequency where all the KEYGEN, ENCAPS, and DECAPS operations of SABER can perform correctly. In short, on a supply voltage of 1.2V, the optimal operating frequency is 715MHz, and the consumed power of the SABER chip is 151mW (for KEYGEN), 158mW (for ENCAPS) and 152mW (for DECAPS). Therefore, the average power consumption at 715MHz is 153.6mW.

The timing results in clock cycles and latency for KEM-supported operations are given in Table 10. Column one provides the design parameter for clock cycles and latency (μs). The corresponding values of clock cycles and latency for KEYGEN, ENCAPS, and DECAPS operations of SABER are shown in columns two to four. The latency value @ optimal 715MHz is calculated using Eq. 15. The DSE process in chapter 4 briefly describes the clock cycle information.

$$latency(in\ \mu s) = \frac{Clock\ cycles}{Frequency\ (in\ MHz)} = \frac{Clock\ cycles}{715MHz} \tag{15}$$

Table 10: Timing results for SABER after physical measurements at nominal 1.2V @ 715MHz.

Operation	***KEYGEN***	***ENCAPS***	***DECAPS***
Clock cycles	7154	7136	9359
Latency (in μs)	10.00	9.98	13.08

Instead of the power and timing results, the top-level area breakdown of the fabricated SABER design is presented in Table 11 where column one provides the design units and column two shows the utilized area. It shows that the I/O placement, serial-in/out interface, SABER crypto core, and four instances of small memories utilize 0.350, 0.041, 0.232, and 0.104mm^2 area out of the total 1mm^2 chip size. The sum of the area of these blocks is 0.727mm^2. The remaining area (1mm^2–0.727mm^2) is wasted with mandatory empty spaces between the IO cells and the seal ring, the IO cells and the core, and power rings.

After the area, timing, and power results, it is essential to show the fabricated SABER chip's response in different operating conditions (or supply voltage values).

Table 11: Top level area breakdown of the SABER chip.

Design unit(s)	***Utilized area (mm^2)***
Pads and I/O ring	0.350
Wrapper + Serial interface	0.041
SABER crypto core	0.232
Memories	0.104

Generally, a Shmoo plot provides the graphical representation of the response of the component (or) system varying over a range of conditions or inputs. Therefore, for only the DECAPS operation of a SABER, the complete range that fabricated SABER chip supports can be visualized in Fig. 30 where the horizontal axis shows the operating frequency (in MHz), and each tick represents an increment of $10MHz$. On the other hand, the vertical axis shows the supplied voltage (in V) in steps of 0.05V. It shows that the fabricated SABER chip is fully operational at a very small clock frequency of $10MHz$ with a supplied voltage of 0.65V. The increase in VDD (from 0.65 to 1.4) increases the operational frequency (from $10MHz$ to a bit more than $800MHz$). Note that the fabricated chip is functional in the green portion of Fig. 30 while the red area determines that the chip is not-operational.

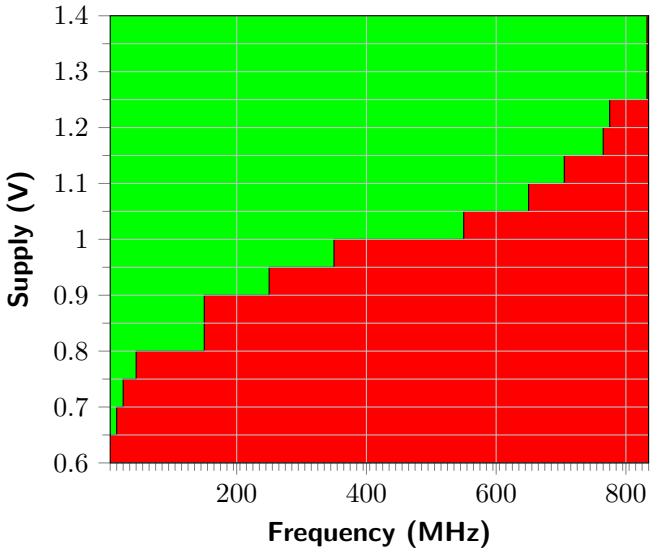


Figure 30: Graphical representation of the entire range of operation that the chip supports (Shmoo plot).

5.3 Comparison and Discussion

Several FPGA and ASIC SABER implementations are available in the literature. Therefore, for a realistic comparison, Table 12 compares only the existing fabricated SABER

chips with the demonstrated chip in this study[7]. Column one of Table 12 provides the reference designs, whereas the implementation technology is given in column two. The area in chip size is illustrated in column three. Column four provides the clock cycle utilization for KEYGEN, ENCAPS, and DECAPS operations. The operating frequency in MHz @ supply voltage is reported in column five. Similar to clock cycles, the computation time in latency (in μs) for KEYGEN, ENCAPS, and DECAPS operations is given in column six. Finally, the last column of Table 12 shows the fabricated chips' power consumption (in mW).

Table 12: Comparison of the fabricated SABER chip with existing PQC ASIC chips. All implementation results are for security equivalent to AES-192.

Ref	*Tech*	*Chip size*	*Clock cyles*	*Frequency (MHz)*	*Latency (in μs)*	*Power (mW)*
[32]	65nm	1.6	14336/18704/23376	160 @ 1.1V	89.6/116.9/146.1	–
[32]	65nm	1.6	14336/18704/23376	10 @ 0.7V	1433.6/1870.4/2337.6	0.334
[33]	28nm	3.6	–/–/–	500 @ 0.9V	–/–/–	39–368
PIP_SP	65nm	1	7154/7136/9359	160 @ 1.2V	44.7/44.6/58.4	43.5
	65nm	1	7154/7136/9359	10 @ 0.7V	715.4/713.6/935.9	0.855
	65nm	1	7154/7136/9359	715 @ 1.2V	10/9.9/13	153.6

Ref [32]: the clock cycles have been calculated by multiplying the corresponding latency values with 160MHz clock frequency, **PIP_SP:** SABER design fabricated in this study.

A realistic comparison is made by operating the fabricated SABER chip on the same conditions employed in [32] for measurement purposes, as presented in Table 12. More precisely, in [32], the KEYGEN, ENCAPS, and DECAPS operations of SABER were computed at 160MHz @ a nominal voltage of 1.1V, and a frequency of 10MHz @ supply voltage of 0.7V.

For two conditions when operating frequency $=$ 160MHz @ 1.1 supply voltage and operating frequency $=$ 10MHz @ 0.7 supply voltage, the fabricated chip (in this study) is 2, 2.62, and 2.50 times faster in terms of clock cycles and computational time (latency) for KEYGEN, ENCAPS, and DECAPS operations of SABER, respectively, in comparison to 65nm demonstrated SABER chip of [32]. The reason is a centralized schoolbook multiplier utilized in this study from [122] for multiplying two 256-degree polynomials in SABER. On the other hand, the SABER fabricated chip of [32] employs a Toom-Cook multiplication method with a striding factor of 4, which reduces memory requirements by half but takes significantly more clock cycles and utilizes more hardware

[7]Before comparing the chip results, it is essential to provide that in chapter 4, the DSE process emphasizes using smaller and distributed memories to achieve high-speed PQC algorithms implementation on the ASIC platform, which is (also) used in the high-speed SABER chip design. More precisely, the chip contains four SRAM-based inferred memories, each with a size of 256×64 and an addressing range of [0-255], [256-511], [512-767], and [768-1023]. However, a logic bug provoked the first address of memories 2, 3, and 4 to be incorrectly decoded, resulting in data overwrite and a few flipped bits in the chip's output compared to the expected results. However, this issue can be bypassed by not using these memory addresses in LightSABER. While on the other hand, the SABER and FireSABER variants will still be affected. Nevertheless, this issue does not impact the computational blocks of SABER or change the number of memory accesses. Therefore, we have the assurance that the reported power values in this thesis are representative.

resources. A comprehensive comparison over various multiplication architectures in [108] reports that the Toom-Cook multiplier is known to be more hardware-intensive than the schoolbook multiplier. Therefore, the use of a schoolbook multiplier along with a shared shift buffer across several building blocks of SABER results in $1mm^2$ chip size, which is comparatively 1.6 times lower compared to [32]. As shown in Table 12, instead of the area and latency parameters, the power comparison is only possible for operating frequency $= 10MHz$ @ 0.7 supply voltage. Comparatively, the fabricated chip in this study consumes 2.55 times more power because the objective of the fabricated SABER chip was to obtain higher clock frequency. In contrast, the objective in [32] was low area and power reduction.

Regardless of the operating conditions considered in [32], a comparison between the clock frequency of $715MHz$ @ 1.2 nominal voltage (considered in this study) and the maximum clock frequency achieved in [32] of $160MHz$ @ 1.1 supply voltage reveals that the chip fabricated in this study performs 8.96, 11.80, and 11.23 times faster for the computation of KEYGEN, ENCAPS, and DECAPS operations, respectively.

Table 12 shows that a realistic and reasonable comparison to [33] regarding the area, timing, and power parameters is not feasible because the implementation technologies are different. This study considers a 65nm process technology for silicon demonstrations while a modern 28nm technology is utilized in [33]. In addition, the fabricated chip is specific to SABER. At the same time, a silicon-implemented design of [33] considers several cryptographic primitives (i.e., SABER, NTRU, CRYSTALS-Dilithium, Rainbow, CRYSTALS-Kyber and McEliece). Depending on the execution of a specific cryptographic protocol, the power values are 39–$368mW$. Hence, this comparison is also not possible.

6 Conclusions and Future Directions

The critical findings of this study are summarized in the following. **Open-source libraries/tools are always in the community's interest.** The polynomial multipliers are essential for multiplying polynomial coefficients in cryptographic algorithms, including PQC and homomorphic encryption schemes. At the onset of my research in 2020, it was found that no open-source tool existed for generating these multipliers. Hence, I developed the first open-source library for large integer polynomial multiplications to address this gap. I believe open-source libraries and tools are always in the community's interest because they promote collaboration, innovation, and knowledge sharing. When researchers and developers share their works on open-source platforms such as GitHub, the respective community can benefit from their expertise and build upon their ideas, leading to rapid outputs in research and development cycles.

Sometimes memory becomes a bottleneck in PQC accelerators. The performance of the PQC algorithm as a hardware accelerator depends on the computation of building blocks (i.e., multipliers, hash, samplers, etc.). Despite these blocks, memory is essential in the hardware accelerators. In PQC algorithms, sometimes, it becomes a real bottleneck when the high-speed designs are in the designer's interest. One of the approaches to overcome this bottleneck is to use faster SRAM-based RegFile memories. **Small and distributed memories (in parallel) improve throughput.** Several instances of small and distributed memories when running (all) in parallel are advantageous to reduce clock cycles and critical paths and improve the operating frequency. In addition, the parallel use of several smaller memories is more beneficial to reduce frequent read/write access from the data memory. Overall, these advantages help to maximize the throughput or performance of the PQC hardware accelerator.

One-time data loading and buffering benefits for designing efficient polynomial multipliers. One of the approaches for performance improvement of polynomial coefficient multiplications is to load data from memory only once and store it in long polynomial buffers. This approach reduces the number of clock cycles required and is particularly helpful in designing parallel multipliers. For instance, in the SS_Parallel and DS_Parallel SABER designs, the schoolbook multiplier benefits from this one-time loading approach, resulting in a lower clock cycles count. Similarly, this approach benefits the design of a compact and parallel NTT-based multiplier for the CRYSTALS-Kyber and CRYSTALS-Dilithium PQC algorithms, which are expected to be standardized in 2024. It is worth noting that NTT-based polynomial multiplications are (also) required for lattice-based homomorphic encryption schemes, in addition to PQC algorithms. **Wider data path benefits to high-speed crypto applications.** Indeed, PQC algorithms require variants of SHA3 and SHAKE hash functions, and these hashes operate on 64-bit for permutation computations. Hence, all state-of-the-art PQC hardware accelerators adopted a 64-bit data path in their designs. Instead of 64-bit, a 256-bit data path is utilized in this thesis, concluding that the wider data path strategy reduces clock cycles and allows for obtaining 2.5GHz on modern 28nm process technology but, on the other hand, increases critical path delay.

Next, some future directions to extend this thesis work are provided. **Design of hardware accelerators.** Several optimization techniques, including pipelining, resource sharing, and wider data path strategy, have been employed to maximize the performance of the PQC algorithm, specifically SABER, which remained a participant in the NIST competition until round three. Even if the optimized approaches are applied to SABER, they could be used in other PQC KEM and digital signature algorithms, such as CRYSTALS-Kyber, LAC, CRYSTALS-Dilithium, and SPHINCS+, to improve their

processing speed and performance. Apart from improving the performance of PQC algorithms, the same optimization techniques with minor adaptations can also be used to realize hardware resources and power consumption for a wide range of cryptographic applications, including IoT and cloud computing. **Unified ECC + PQC accelerators.** The ENISA report from 2021 [123] highlights the transition from the pre-quantum era to the post-quantum one, which requires the combination of pre-and post-quantum cryptography algorithms in a single cryptosystem to ensure security even for today's computers. Hence, this could be an attractive choice in the future.

Protection against physical attacks. This thesis explores only the hardware realization of the lattice-based PQC schemes on the ASIC platform without focusing on the side-channel resistance and other related attacks such as timing, fault, etc. Hence, designing countermeasures against side-channel and fault attacks would be very interesting for future research.

List of Figures

List of Tables

References

[1] ITU, "Measuring digital development facts and figures," last accessed on February 12, 2023. [Online] available at: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2021.pdf.

[2] E. Snowden's, "Nsa collecting phone records of millions of verizon customers daily," last accessed on February 22, 2023. [Online] available at: `https://www.theguardian.com/us-news/the-nsa-files`.

[3] J. Daemen and V. Rijmen, "Aes proposal: Rijndael," last accessed on February 12, 2023. [Online] available at: `https://web.archive.org/web/20070203204845/https://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael.pdf`.

[4] NORTON, "What is a firewall? firewalls explained and why you need one," last accessed on February 11, 2023. [Online] available at: `https://us.norton.com/blog/emerging-threats/what-is-firewall#`.

[5] Fortinet, "What is access control?," last accessed on February 16, 2023. [Online] available at: `https://www.fortinet.com/resources/cyberglossary/access-control`.

[6] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd ed., 2014. `https://dl.acm.org/doi/book/10.5555/2700550`.

[7] CADO-NFS, "Cado-nfs an implementation of the number field sieve algorithm," last accessed on February 17, 2023. [Online] available at: `https://cado-nfs.gitlabpages.inria.fr`.

[8] C. Pomerance and P. Erdös, "A tale of two sieves," last accessed on February 21, 2023. [Online] available at: `https://www.ams.org/notices/199612/pomerance.pdf`.

[9] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Neven, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, p. 505–510, 2019.

[10] IBM, "Ibm unveils breakthrough 127-qubit quantum processor," last accessed on November 22, 2022. [Online] available at: `https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor`.

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

[12] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, 1978.

[13] V. S. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology — CRYPTO '85 Proceedings* (H. C. Williams, ed.), (Berlin, Heidelberg), pp. 417–426, Springer Berlin Heidelberg, 1986.

[14] W. H. Wootters and W. H. Zurek, "A single quantum cannot be cloned," *Nature*, vol. 299, no. 5886, pp. 802–803, 1982.

[15] X. Lu, Y. Liu, Z. Zhang, D. Jia, H. Xue, J. He, B. Li, and K. Wang, "Lac: Practical ring-lwe based public-key encryption with byte-level modulus." Cryptology ePrint Archive, Paper 2018/1009, 2018. `https://eprint.iacr.org/2018/1009`.

[16] P. Schwabe and J. Mann, "Crystals-kyber: Cryptographic suite for algebraic lattices," last accessed on February 11, 2023. [Online] available at: `https://pq-crystals.org/kyber/`.

[17] P. Schwabe and J. Mann, "Crystals-dilithium: Cryptographic suite for algebraic lattices," last accessed on February 11, 2023. [Online] available at: `https://pq-crystals.org/dilithium/`.

[18] P. Ball, "First quantum computer to pack 100 qubits enters crowded race," *Nature*, vol. 599, p. 542, 2021.

[19] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *J. ACM*, vol. 56, sep 2009.

[20] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices." Cryptology ePrint Archive, Paper 2011/401, 2011. `https://eprint.iacr.org/2011/401`.

[21] A. Basso, J. M. B. Mera, J.-P. D'Anvers, A. Karmakar, S. S. Roy, M. V. Beirendonck, and F. Vercauteren, "Saber: Mod-lwr based kem (round 3 submission)," last accessed on March 23, 2022. [Online] available at: `https://www.esat.kuleuven.be/cosic/pqcrypto/saber/files/saberspecround3.pdf`.

[22] NIST, "Computer security resource centre: Pqc standardization process, third round candidate announcement," 2020. [Online] available at: `https://csrc.nist.gov/news/2020/pqc-third-round-candidate-announcement`.

[23] Intel, "Integrated cryptographic and compression accelerators on intel architecture platforms," last accessed on September 29, 2022. [Online] available at: `https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/integrated-cryptographic-compression-accelerators-brief.pdf`.

[24] IBM, "Ibm cex7s / 4769 pcie cryptographic coprocessor (hsm)," last accessed on October 20, 2022. [Online] available at: `https://public.dhe.ibm.com/security/cryptocards/pciecc4/docs/4769_Data_Sheet.pdf`.

[25] A. Jati, N. Gupta, A. Chattopadhyay, and S. K. Sanadhya, "A configurable crystals-kyber hardware implementation with side-channel protection." Cryptology ePrint Archive, Paper 2021/1189, 2021. https://eprint.iacr.org/2021/1189.

[26] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "Kali: A crystal for post-quantum security using kyber and dilithium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 2, pp. 747–758, 2023.

[27] L. Beckwith, D. T. Nguyen, and K. Gaj, "High-performance hardware implementation of crystals-dilithium." Cryptology ePrint Archive, Paper 2021/1451, 2021. https://eprint.iacr.org/2021/1451.

[28] G. Land, P. Sasdrich, and T. Güneysu, "A hard crystal - implementing dilithium on reconfigurable hardware," in *Smart Card Research and Advanced Applications: 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11–12, 2021, Revised Selected Papers*, (Berlin, Heidelberg), p. 210–230, Springer-Verlag, 2021.

[29] Z. Zhou, D. He, Z. Liu, M. Luo, and K.-K. R. Choo, "A software/hardware co-design of crystals-dilithium signature scheme," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 14, jun 2021.

[30] S. Sinha Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, p. 443–466, 2020.

[31] Y. Zhu, M. Zhu, B. Yang, W. Zhu, C. Deng, C. Chen, S. Wei, and L. Liu, "Lwrpro: An energy-efficient configurable crypto-processor for module-lwr," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 3, pp. 1146–1159, 2021.

[32] A. Ghosh, J. Mera, A. Karmakar, D. Das, S. Ghosh, I. Verbauwhede, and S. Sen, "A 334μw 0.158mm^2 saber learning with rounding based post-quantum crypto accelerator," in *2022 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–2, 2022.

[33] Y. Zhu, W. Zhu, M. Zhu, C. Li, C. Deng, C. Chen, S. Yin, S. Yin, S. Wei, and L. Liu, "A 28nm 48kops 3.4µj/op agile crypto-processor for post-quantum cryptography on multi-mathematical problems," 2022. IEEE International Solid State Circuits Conference (ISSCC), San Francisco, CA, USA, p. 514–516, February 20-26, 2022.

[34] M. Imran, F. Almeida, J. Raik, A. Basso, S. S. Roy, and S. Pagliarini, "Design space exploration of saber in 65nm asic," in *Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security*, ASHES '21, (New York, NY, USA), p. 85–90, Association for Computing Machinery, 2021.

[35] M. Imran, A. Aikata, S. S. Roy, and S. Pagliarini, "High-speed design of post quantum cryptography with optimized hashing and multiplication," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. –, no. –, 2023.

[36] M. Imran, F. Almeida, A. Basso, S. S. Roy, and S. Pagliarini, "High-speed saber key encapsulation mechanism in 65nm cmos," *Journal of Cryptographic Engineering (JCEN)*, vol. –, no. –, pp. –, 2023.

[37] S. S. Roy, *Public Key Cryptography on Hardware Platforms: Design and Analysis of Elliptic Curve and Lattice-based Cryptoprocessors*. PhD thesis, KU LEUVEN, Belgium, 2017.

[38] M. Ajtai, "Generating hard instances of lattice problems (extended abstract)," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, (New York, NY, USA), p. 99–108, Association for Computing Machinery, 1996.

[39] J. Silverman, J. Pipher, and J. Hoffstein, "An introduction to mathematical cryptography," 2008. [Online] available at: https://link.springer.com/book/10.1007/978-0-387-77993-5.

[40] O. Regev, "Lattices in computer science," Fall 2009. [Online] available at: https://cims.nyu.edu/ regev/teaching/lattices_fall_2009/.

[41] N. Körtge, "The idea behind lattice-based cryptography or how can lattices be useful for cryptography?," last accessed on February 26, 2023. [Online] available at: `https://medium.com/nerd-for-tech/the-idea-behind-lattice-based-cryptography-5e623fa2532b`.

[42] A. K. Lenstra, H. W. L. Jr., and L. Lovász, "Factoring polynomials with rational coefficients," *Mathematische Annalen*, vol. 261, p. 515–534, 1982.

[43] M. Ajtai, R. Kumar, and D. Sivakumar, "A sieve algorithm for the shortest lattice vector problem," in *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, STOC '01, (New York, NY, USA), p. 601–610, Association for Computing Machinery, 2001.

[44] C. Schnorr and M. Euchner, "Lattice basis reduction: Improved practical algorithms and solving subset sum problems," *Mathematical Programming*, vol. 66, p. 181–199, 1994.

[45] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," in *In Proceedings of the thirty-seventh annual ACM symposium on Theory of computing, STOC '05*, pp. 84–93, 2005.

[46] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the design of hardware building blocks for modern lattice-based encryption schemes," in *Cryptographic Hardware and Embedded Systems – CHES 2012* (E. Prouff and P. Schaumont, eds.), (Berlin, Heidelberg), pp. 512–529, Springer Berlin Heidelberg, 2012.

[47] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology – EUROCRYPT 2010* (H. Gilbert, ed.), (Berlin, Heidelberg), pp. 1–23, Springer Berlin Heidelberg, 2010.

[48] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices." Cryptology ePrint Archive, Paper 2011/401, 2011. `https://eprint.iacr.org/2011/401`.

[49] J. Alwen, S. Krenn, K. Pietrzak, and D. Wichs, "Learning with rounding, revisited: New reduction, properties and applications." Cryptology ePrint Archive, Paper 2013/098, 2013. `https://eprint.iacr.org/2013/098`.

[50] A. Bogdanov, S. Guo, D. Masny, S. Richelson, and A. Rosen, "On the hardness of learning with rounding over small modulus," in *Theory of Cryptography* (E. Kushilevitz and T. Malkin, eds.), (Berlin, Heidelberg), pp. 209–224, Springer Berlin Heidelberg, 2016.

[51] S. Akleylek, E. Alkim, P. S. L. M, N. Bindel, J. Buchmann, E. Eaton, G. Gutoski, J. Krämer, P. Longa, H. Polat, J. E. Ricardini, and G. Zanon, "Submission to nist's post-quantum project (2nd round): lattice-based digital signature scheme qtesla," last accessed on March 20, 2020. [Online] available at: `https://qtesla.org`.

[52] D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal, "Ntru prime: round 2–20190330," last accessed on April 17, 2020. [Online] available at: `https://ntruprime.cr.yp.to`.

[53] E. Alkim, R. Avanzi, J. Bos, L. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, D. Stebila, M. R. Albrecht, E. Orsini, V. Osheter, K. G. Paterson, G. Peer, and N. P. Smart, "Newhope," last accessed on April 17, 2020. [Online] available at: `https://newhopecrypto.org`.

[54] M. Hamburg, "Post-quantum cryptography proposal: Threebears," last accessed on May 23, 2020. [Online] available at: `https://sourceforge.net/projects/threebears/`.

[55] X. Lu, Y. Liu, D. Jia, H. Xue, J. He, Z. Zhang, Z. Liu, H. Yang, B. Li, and K. Wang, "Lac: Lattice-based cryptosystems," last accessed on May 11, 2020. [Online] available at: `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.

[56] H. Baan, S. Bhattacharya, S. Fluhrer, O. Garcia-Morchon, T. Laarhoven, R. Player, R. Rietman, M.-J. O. Saarinen, L. Tolhuizen, J. L. Torre-Arce, and Z. Zhang, "Round5: Kem and pke based on (ring) learning with rounding," last accessed on April 28, 2020. [Online] available at: `https://round5.org`.

[57] C. Chen, O. Danba, J. Hoffstein, A. Hülsing, J. Rijneveld, T. Saito, J. M. Schanck, P. Schwabe, W. Whyte, K. Xagawa, T. Yamakawa, and Z. Zhang, "Ntru," last accessed on April 16, 2020. [Online] available at: `https://ntru.org`.

[58] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, D. Stebila, K. Easterbrook, and B. LaMacchia, "Frodokem learning with errors key encapsulation algorithm," last accessed on April 17, 2020. [Online] available at: `https://frodokem.org`.

[59] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: fast-fourier lattice-based compact signatures over ntru specifications v1.1," last accessed on April 20, 2020. [Online] available at: `https://falcon-sign.info`.

[60] D. Soni and R. Karri, "Efficient hardware implementation of pqc primitives and pqc algorithms using high-level synthesis," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 296–301, 2021.

[61] NIST, "Computer security resource centre: Post-quantum cryptography, round 2 submissions," 2020.

[62] S. Venkatachalam, E. Adams, H. J. Lee, and S.-B. Ko, "Design and analysis of area and power efficient approximate booth multipliers," *IEEE Transactions on Computers*, vol. 68, no. 11, pp. 1697–1703, 2019.

[63] T. Pöppelmann and T. Güneysu, "Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware," in *Progress in Cryptology – LATINCRYPT 2012* (A. Hevia and G. Neven, eds.), (Berlin, Heidelberg), pp. 139–158, Springer Berlin Heidelberg, 2012.

[64] Z. Liang and Y. Zhao, "Number theoretic transform and its applications in lattice-based cryptosystems: A survey," 2022.

[65] A. C. Mert, E. Öztürk, and E. Savaş, "FPGA implementation of a run-time configurable ntt-based polynomial multiplication hardware," *Microprocessors and Microsystems*, vol. 78, p. 103219, 2020.

[66] K. Koleci, P. Mazzetti, M. Martina, and G. Masera, "A flexible ntt-based multiplier for post-quantum cryptography," *IEEE Access*, vol. 11, pp. 3338–3351, 2023.

[67] K. Derya, A. C. Mert, E. Öztürk, and E. Savaş, "Coha-ntt: A configurable hardware accelerator for ntt-based polynomial multiplication," *Microprocessors and Microsystems*, vol. 89, p. 104451, 2022.

[68] NIST, "Sha-3 standard: Permutation-based hash and extendable-output functions." FIPS PUB 202, last accessed on January 9, 2023. Available at `https://doi.org/10.6028/NIST.FIPS.202`.

[69] M. Imran, Z. U. Abideen, and S. Pagliarini, "An experimental study of building blocks of lattice-based nist post-quantum cryptographic algorithms," *Electronics*, vol. 9, no. 11, 2020.

[70] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 10, pp. 2459–2463, 2019.

[71] Y. Zhang, Y. Cui, Z. Ni, D.-E.-S. Kundi, D. Liu, and W. Liu, "A lightweight and efficient schoolbook polynomial multiplier for saber," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2251–2255, 2022.

[72] M. Kashif, I. Cicek, and M. Imran, "A hardware efficient elliptic curve accelerator for fpga based cryptographic applications," in *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*, pp. 362–366, 2019.

[73] S. Jahani, A. Samsudin, and K. G. Subramanian, "Efficient big integer multiplication and squaring algorithms for cryptographic applications," *Journal of Applied Mathematics*, vol. 2014, no. 107109, pp. 1–9, 2014.

[74] M. Bodrato, "Towards optimal toom-cook multiplication for univariate and multivariate polynomials in characteristic 2 and 0," in *Arithmetic of Finite Fields* (C. Carlet and B. Sunar, eds.), (Berlin, Heidelberg), pp. 116–133, Springer Berlin Heidelberg, 2007.

[75] M. Bodrato, "Notes on low degree toom-cook multiplication with small characteristic," 2020. Online] available at: http://www.bodrato.it/papers/#CIVV2007.

[76] A. Basso, F. Aydin, D. Dinu, J. Friel, A. Varna, M. Sastry, and S. Ghosh, "Where star wars meets star trek: Saber and dilithium on the same polynomial multiplier." Cryptology ePrint Archive, Report 2021/1697, 2021. https://ia.cr/2021/1697.

[77] T. Fritzmann, G. Sigl, and J. Sepúlveda, "Risq-v: Tightly coupled risc-v accelerators for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, p. 239–280, Aug. 2020.

[78] W.-K. Lee, H. Seo, S. O. Hwang, A. Karmakar, J. M. B. Mera, and R. Achar, "Dpcrypto: Acceleration of post-quantum cryptographic algorithms using dot-product instruction on gpus." Cryptology ePrint Archive, Report 2021/1389, 2021. https://ia.cr/2021/1389.

[79] H. Becker, J. M. Bermudo Mera, A. Karmakar, J. Yiu, and I. Verbauwhede, "Polynomial multiplication on embedded vector architectures," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, p. 482–505, 2021.

[80] A. Abdulrahman, J.-P. Chen, Y.-J. Chen, V. Hwang, M. J. Kannwischer, and B.-Y. Yang, "Multi-moduli ntts for saber on cortex-m3 and cortex-m4," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, p. 127–151, Nov. 2021.

[81] A. Karmakar, J. M. Bermudo Mera, S. Sinha Roy, and I. Verbauwhede, "Saber on arm: Cca-secure module lattice-based key encapsulation on arm," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, p. 243–266, Aug. 2018.

[82] M. V. Beirendonck, J.-P. D'anvers, A. Karmakar, J. Balasch, and I. Verbauwhede, "A side-channel-resistant implementation of saber," *J. Emerg. Technol. Comput. Syst.*, vol. 17, no. 2, p. 1–26, 2021.

[83] T. Fritzmann, M. Van Beirendonck, D. Basu Roy, P. Karl, T. Schamberger, I. Verbauwhede, and G. Sigl, "Masked accelerators and instruction set extensions for post-quantum cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, p. 414–460, Nov. 2021.

[84] A. Abdulgadir, K. Mohajerani, V. B. Dang, J.-P. Kaps, and K. Gaj, "A lightweight implementation of saber resistant against side-channel attacks," 2021. In: Adhikari, A., Küsters, R., Preneel, B. (eds) Progress in Cryptology – INDOCRYPT 2021. INDOCRYPT 2021. Lecture Notes in Computer Science(), vol 13143. Springer, Cham. https://doi.org/10.1007/978-3-030-92518-5_11.

[85] B. Wang, X. Gu, and Y. Yang, "Saber on esp32." Cryptology ePrint Archive, Report 2019/1453, 2019. https://ia.cr/2019/1453.

[86] J. Maria Bermudo Mera, F. Turan, A. Karmakar, S. Sinha Roy, and I. Verbauwhede, "Compact domain-specific co-processor for accelerating module lattice-based kem," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[87] ESPRESSIF, "Esp32 series," last accessed on February 17, 2023. [Online] available at: https://www.alldatasheet.com/view_datasheet.jsp?Searchword=ESP32.

[88] V. B. Dang, F. Farahmand, M. Andrzejczak, and K. Gaj, "Implementing and benchmarking three lattice-based post-quantum cryptography algorithms using software/hardware codesign," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, pp. 206–214, 2019.

[89] Y. Tu, P. He, C.-Y. Lee, D. Chasaki, and J. Xie, "Hardware implementation of high-performance polynomial multiplication for kem saber," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1160–1164, 2022.

[90] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," *IEEE Transactions on Computers*, vol. 66, no. 8, pp. 1369–1382, 2017.

[91] M. Morales-Sandoval, C. Feregrino-Uribe, P. Kitsos, and R. Cumplido, "Area/performance trade-off analysis of an fpga digit-serial gf(2m) montgomery multiplier based on lfsr," *Computers & Electrical Engineering*, vol. 39, no. 2, pp. 542 – 549, 2013.

[92] A. A. Abd-Elkader, M. Rashdan, E.-S. A. Hasaneen, and H. F. Hamed, "Advanced implementation of montgomery modular multiplier," *Microelectronics Journal*, vol. 106, p. 104927, 2020.

[93] M. Imran, Z. U. Abideen, and S. Pagliarini, "An experimental study of building blocks of lattice-based nist post-quantum cryptographic algorithms," *Electronics*, vol. 9, no. 11, p. 1953, 2020.

[94] B. Rashidi, "Throughput/area efficient implementation of scalable polynomial basis multiplication," *Journal of Hardware and Systems Security*, vol. 4, no. 2, pp. 120–135, 2020.

[95] H. Eberle, N. Gura, S. Shantz, V. Gupta, L. Rarick, and S. Sundaram, "A public-key cryptographic processor for rsa and ecc," in *Proceedings. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004.*, pp. 98–110, IEEE, 2004.

[96] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, (New York, NY, USA), p. 1219–1234, Association for Computing Machinery, 2012.

[97] R. Azarderakhsh, K. U. Järvinen, and M. Mozaffari-Kermani, "Efficient algorithm and architecture for elliptic curve cryptography for extremely constrained secure applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 4, pp. 1144–1155, 2014.

[98] J. Xie, J. j. He, and P. K. Meher, "Low latency systolic montgomery multiplier for finite field $gf(2^m)$ based on pentanomials," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 2, pp. 385–389, 2013.

[99] Y. Doröz, E. Öztürk, and B. Sunar, "Accelerating fully homomorphic encryption in hardware," *IEEE Transactions on Computers*, vol. 64, no. 6, pp. 1509–1521, 2015.

[100] G. D. Sutter, J.-P. Deschamps, and J. L. Imana, "Efficient elliptic curve point multiplication using digit-serial binary field operations," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, pp. 217–225, 2013.

[101] S. Venkatachalam, H. J. Lee, and S.-B. Ko, "Power efficient approximate booth multiplier," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2018.

[102] P. K. Somayajulu and S. Ramesh, "Area and power efficient 64-bit booth multiplier," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pp. 721–724, 2020.

[103] A. Mrabet, N. El-Mrabet, R. Lashermes, J.-B. Rigaud, B. Bouallegue, S. Mesnager, and M. Machhout, "A scalable and systolic architectures of montgomery modular multiplication for public key cryptosystems based on dsps," *Journal of Hardware and Systems Security*, vol. 1, no. 3, pp. 219–236, 2017.

[104] M. Machhout, Z. Guitouni, K. Torki, L. Khriji, and R. Tourki, "Coupled fpga/asic implementation of elliptic curve crypto-processor," *International Journal of Network Security & Its Applications*, vol. 2, no. 2, pp. 100–112, 2010.

[105] J. Xie, P. K. Meher, X. Zhou, and C. Lee, "Low register-complexity systolic digit-serial multiplier over $gf(2^m)$ based on trinomials," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 4, pp. 773–783, 2018.

[106] J. Pan, P. Song, and C. Yang, "Efficient digit-serial modular multiplication algorithm on fpga," *IET Circuits, Devices Systems*, vol. 12, no. 5, pp. 662–668, 2018.

[107] M. Imran, Z. U. Abideen, and S. Pagliarini, "TTech-LIB: Center for hardware security," 2020. `https://github.com/Centre-for-Hardware-Security/TTech-LIB`.

[108] M. Imran, Z. U. Abideen, and S. Pagliarini, "An open-source library of large integer polynomial multipliers," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 145–150, 2021.

[109] M. Imran, Z. U. Abideen, and S. Pagliarini, "A versatile and flexible multiplier generator for large integer polynomials," *Journal of Hardware and Systems Security (HASS)*, vol. –, no. –, 2022.

[110] NIST, "Recommended Elliptic Curves for Federal Government Use (1999)." `https://csrc.nist.gov/csrc/media/publications/fips/186/2/archive/2000-01-27/documents/fips186-2.pdf`.

[111] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, (New York, NY, USA), p. 171–178, Association for Computing Machinery, 2015.

[112] R. Azarderakhsh and A. Reyhani-Masoleh, "Low-complexity multiplier architectures for single and hybrid-double multiplications in gaussian normal bases," *IEEE Transactions on Computers*, vol. 62, no. 4, pp. 744–757, 2013.

[113] A. Rezai and P. Keshavarzi, "High-throughput modular multiplication and exponentiation algorithms using multibit-scan–multibit-shift technique," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1710–1719, 2015.

[114] S. S. Roy and A. Basso, "Hardware implementation of saber," last accessed on March 19, 2023. [Online] available at: `https://github.com/sujoyetc/SABER_HW`.

[115] H. E. Sumbul, K. Vaidyanathan, Q. Zhu, F. Franchetti, and L. Pileggi, "A synthesis methodology for application-specific logic-in-memory designs," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[116] K. Team, "Keccak in vhdl: High-speed core," last accessed on March 16, 2023. [Online] available at: `https://keccak.team/hardware.html`.

[117] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "Instruction-set accelerated implementation of crystals-kyber," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 11, pp. 4648–4659, 2021.

[118] Y. Huang, M. Huang, Z. Lei, and J. Wu, "A pure hardware implementation of crystals-kyber pqc algorithm through resource reuse," *IEICE Electronics Express*, vol. advpub, p. 17.20200234, 2020.

[119] T. T. Nguyen, S. Kim, Y. Eom, and H. Lee, "Area-time efficient hardware architecture for crystals-kyber," *Applied Sciences*, vol. 12, no. 11, p. 10 pages, 2022.

[120] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "A monolithic hardware implementation of kyber: Comparing apples to apples in pqc candidates," in *Progress in Cryptology – LATINCRYPT 2021* (P. Longa and C. Ràfols, eds.), (Cham), pp. 108–126, Springer International Publishing, 2021.

[121] STM32, "Nucleo-64 development board with stm32f446re mcu," last accessed on March 19, 2023. [Online] available at `https://www.st.com/en/evaluation-tools/nucleo-f446re.html`.

[122] A. Basso and S. S. Roy, "Optimized polynomial multiplier architectures for post-quantum kem saber," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1285–1290, 2021.

[123] W. Beullens, J.-P. D'Anvers, A. Hülsing, T. Lange, L. Panny, C. de Saint Guilhem, and N. P. Smart., "Post-Quantum Cryptography: Current State and Quantum Mitigation." Available at: `https://www.enisa.europa.eu/publications/post-quantum-cryptography-current-state-and-quantum-mitigation`, last accessed on March 13, 2023.