Rudraksh: A compact and lightweight post-quantum key-encapsulation mechanism

Suparna Kundu[1], Archisman Ghosh[2], Angshuman Karmakar[3], Shreyas Sen[2] and Ingrid Verbauwhede[1]

[1] COSIC, KU Leuven, Belgium
[2] Purdue University, USA
[3] Indian Institute of Technology Kanpur, India

suparna.kundu@esat.kuleuven.be, ghosh69@purdue.edu, angshuman@cse.iitk.ac.in, shreyas.sen@gmail.com, ingrid.verbauwhede@esat.kuleuven.be

Abstract. Resource-constrained devices such as wireless sensors and Internet of Things (IoT) devices have become ubiquitous in our digital ecosystem. These devices generate and handle a major part of our digital data. In the face of the impending threat of quantum computers on our public-key infrastructure, it is impossible to imagine the security and privacy of our digital world without integrating post-quantum cryptography (PQC) into these devices. Usually, due to the resource constraints of these devices, the cryptographic schemes in these devices have to operate with very small memory and consume very little power. Therefore, we must provide a lightweight implementation of existing PQC schemes by possibly trading off the efficiency. The other option that can potentially provide the most optimal result is by designing PQC schemes suitable for lightweight and low-power-consuming implementation. Unfortunately, the latter method has been largely ignored in PQC research.

In this work, we first provide a lightweight CCA-secure PQ key-encapsulation mechanism (KEM) design based on hard lattice problems. We have done a scrupulous and extensive analysis and evaluation of different design elements, such as polynomial size, field modulus structure, reduction algorithm, secret and error distribution, etc., of a lattice-based KEM. We have optimized each of them to obtain a lightweight design. Our design provides a 100 bit of PQ security and shows $\sim$3x improvement in terms of area with respect to the state-of-the-art Kyber KEM, a PQ standard.

Keywords: Post-quantum cryptography, Key-encapsulation mechanism, Lightweight cryptography, Lattice-based cryptography, Hardware implementation, FPGA.

1 Introduction

Lightweight cryptography (LWC) is a niche research area in cryptography that studies methods to incorporate secure cryptographic protocols into devices with minimal resources due to their operational requirements. An example is the Internet of Things (IoT) devices. These devices are ubiquitous in our current hyper-connected world and have vast applications in smart cities, autonomous driving, supply chain management, telemedicine, smart homes, etc. It is projected that there will be close to 55.7 billion connected (IoT) devices generating around 80 Zettabytes of data, which is around 50% of all data generated by the end of 2025 [Int, Tao]. Apart from IoT devices, there are other resource constraint devices such as wearable electronics, sensors, and actuators used in industrial automation, environmental monitoring, and microcontrollers in embedded devices. These devices often possess and communicate private and critical information that should be protected from misuse by malicious entities. However, due to the minimal resources of these devices, it is not trivial to integrate cryptographic or secure communication protocols in these devices. Hence, all these devices and their applications are direct beneficiaries of LWC schemes.

There are two major avenues in the research and development of LWC. First, to implement existing cryptographic protocols which are not specifically designed as LWC in a *lightweight manner* such as the lightweight implementations of symmetric-key ciphers such as AES (Advanced Encryption Standard) [Can05, BMR+13], Keccak [KY10, KYS+11], or public-key cryptographic (PKC) algorithms such as Elliptic curve cryptography (ECC) [HWF09, BGK+06]. Second, designing cryptographic schemes that are suitable for lightweight implementation such as symmetric-key cipher ASCON [DEMS12], which is the winner of the National Institute of Standards and Technology (NIST) lightweight cryptography competition [NIS23a] and also selected as the 'primary choice' for lightweight authenticated encryption in the final portfolio of the CAESAR [cae19] competition. Quark [AHMN13] is a lightweight hash function designed particularly for low-power devices such as Radio Frequency Identification (RFID) devices. Similarly, for PKC, Brainpool curve Brainpool224r1 [SS04] or Koblitz curves [Kob87] like curve25519 [Ber06] were proposed for lightweight implementation of ECC.

Recently, NIST also standardized post-quantum cryptography (PQC) [AAC+22] key-encapsulation mechanism (KEM) Crystals-Kyber [ABD+21] and digital signature schemes SPHINCS+ [ABB+18], Crystals-Dilithium [DKL+18], and Falcon [FHK+18] as a contingency plan in anticipation of the arrival of large-scale quantum computers and their detrimental effect on our existing PKC schemes. Naturally, we will also have to equip resource-constrained IoT and embedded devices with quantum-secure cryptographic schemes to secure them for the foreseeable future. There exist some LW implementations of the standardized schemes such as compact implementation Kyber [XL21] or Dilithium [ZZW+21, LSG22], but to the best of our knowledge, there does not exist any LW design of PQC.

Apart from LW design, there is another subtle issue in the context of IoT devices. Consider a typical IoT ecosystem, as shown in Fig. 1. Here, the IoT peripheral devices connect to the public internet through an IoT gateway server. The IoT Gateway architecture has several layers, such as a security layer, device layer, data management layer, etc. As part of connecting IoT peripheral devices, the IoT gateway servers perform data filtering and processing, protocol translation, authorization and authentication, etc. Whenever a user or device wants to connect to a peripheral device or vice-versa, the gateway servers have to run proper authentication and authorization protocols to make this happen. The gateway servers are usually powerful servers serving numerous IoT peripheral devices simultaneously. So, for them, high throughput is a more important operational metric than resource consumption. Meanwhile, the reverse is more important for IoT peripheral devices, which connect sporadically to the IoT gateway servers. Therefore, a *flexible* cryptographic scheme that can be instantiated either in a high latency and low resource consumption mode or in a low latency and high resource consumption mode is highly suitable in this scenario.

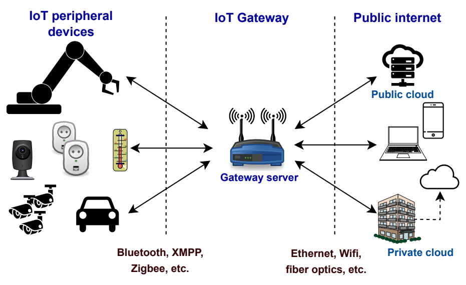

Figure 1: An illustrative example of a typical IoT gateway architecture.

From a very broad perspective, this work aims to close these gaps in research. For the rest of this work, we will use the term lattice-based cryptography (LBC) to denote the cryptographic schemes based on the learning with errors (LWE) [Reg09] problem or its variants such as ring-learning with errors [LPR10], learning with rounding [BPR12], etc. We also want to delineate the term *lightweight* implementation here. For software implementation on resource constraint devices like Cortex-M0/M4 we use the term lightweight implementation for implementations with low memory footprint such as [BKS19, KMRV18, GKS20]. Lightweight design also implies low area and low-power or energy solutions for hardware devices such as field-programmable gate arrays (FPGA) or application-specific integrated circuits (ASIC); however, low-power or energy implementation cannot be demonstrated without custom ASIC design as FPGA defaults to high power-consuming interfaces. We expect that the ASIC version of our design will also reflect relatively lower memory as we can custom-make memory as per our requirement instead of using entire Block RAMs. We have demonstrated a low-area implementation in FPGA in this work. We have kept ASIC-related optimization as part of future work. Below, we briefly summarize the salient contributions of this work.

PQ KEM suitable for resource-constrained devices: We propose a lightweight post-quantum chosen-ciphertext attack (CCA) secure module-learning with errors-based key-encapsulation mechanism (MLWE-KEM). To the best of our knowledge, only the work by Buchmann et al. [BGG$^+$16] deliberated on a lightweight lattice-based key-exchange scheme suitable for IoT devices before us. However, this scheme was only secure under chosen plaintext attacks. This is often inadequate as Hermans et al. [HPVP11] have pointed out that a CPA-secure scheme can only provide security against a narrow range of adversaries. Therefore, in this work, we focus on designing a CCA-secure KEM. Also, one of the most prominent issues among the standardized PQC signatures is their large signature size compared to classical signature schemes. This has a very detrimental effect on some protocols, such as transport layer security (TLS), where the increase in the size of certificates in the chain-of-trust model leads to serious performance degradation [SKD20] due to the congestion control mechanism of the transmission control protocol (TCP). To address such problems, NIST has called for another standardization [CML] for PQ signatures with small signature sizes and fast verification time. Also, some proposals have been made for replacing the TLS handshake with a CCA-secure KEM, such as KEMTLS [SSW20] or TLS-PDK [SSW21], for better performance. Therefore, we believe that lightweight KEMs profoundly impact the transition from classical PKC to PQC. Further, the techniques developed in this work can also be used to create a lightweight PQ digital signature scheme.

Practical design strategy: We adopt a new design style that is strongly coupled with hardware implementation. Our design decisions have been strongly driven by their potential advantage for lightweight hardware implementation. We explored the parameter space of LBC to propose optimum parameters that satisfy our design objectives. The NIST PQ standardization procedure witnessed a collective effort from researchers around the world for a thorough and rigorous analysis of different design elements of LBC. Therefore, to reap the benefits of the procedure and to bolster confidence in our designed KEM, we have kept the design very Kyber-*esque*. We refrained from making aggressive design decisions such as using non-constant time error correcting codes or unusual secret and error distributions such as narrow centered binomial distribution, fixed weight binary vectors, etc. Such designs were shown to be vulnerable to different types of attacks in the past and, therefore, considered insecure. We also explored and analyzed different implementations of number theoretic transform (NTT)-based polynomial multiplication. We use a modular reduction algorithm for lightweight hardware implementation. We provide optimized hardware implementation using a Xilinx Virtex-7 Field Programmable Gate Array (FPGA) to demonstrate the efficacy and justify the design decisions. We have also comprehensively compared our scheme with other state-of-the-art compact implementations of LBC.

Lightweight implementation: We demonstrated our design using a lightweight implementation in FPGA. We observe that a major source of hardware overhead for lattice-based

KEMs such as Kyber mainly comes from the bulky Keccak [BDPVA13] modules, storing the twiddle factors and other memory requirements. We want to make this KEM design suitable for lightweight CCA-secure KEMs, so we focus on minimizing the area in terms of FPGA resources such as LUT, DSP, and flip-flops, and memory with reasonable latency (100us). ASCON [DEMS12] is a family of lightweight authenticated encryption and hashing algorithms. We are the first to replace Keccak using ASCON [DEMS12] to reduce the overhead of Keccak. This is also the first demonstration of the efficiency and benefits of using ASCON in a cryptographic scheme. Our results show that area and memory can be reduced approximately 3$\times$ with respect to the most compact design of Kyber [XL21], the current NIST standard for KEM in terms of LUTs, flip-flops, and DSP. Our design requires no slices and only $\sim$11Kb cumulative memory, which is 2$\times$ with respect to the state-of-the-art.

2 Background and related works

We denote the set of integers modulo q as $\mathbb{Z}_q$ and the quotient ring $\mathbb{Z}_q/(x^n+1)$ as R_q^n $(n \geq 1)$. The ring containing the vectors with l elements and the matrix with $l \times l$ elements from R_q^n are represented as $R_q^{n(l)}$ $R_q^{n(l\times l)}$ respectively. Lowercase letters indicate polynomials ($v \in R_q^n$), and bold lowercase letters denote vectors of polynomials ($\boldsymbol{s} \in R_q^{n(l)}$). Bold uppercase letters represent matrices of polynomials ($\boldsymbol{A} \in R_q^{n(l\times l)}$). Multiplication of two polynomials $a \in R_q^n$ and $b \in R_q^n$ is denoted by $a \cdot b \in R_q^n$, and the dot product between those two polynomials is presented by $(a \circ b) \in R_q^n$. The number theoretic transform (NTT) representation of a polynomial $a \in R_q^n$ is denoted by $\hat{a}$. When NTT is applied to each constituent element of $\boldsymbol{a} \in R_q^{n(l)}$ and $\boldsymbol{A} \in R_q^{n(l\times l)}$), it is denoted as $\hat{\boldsymbol{a}}$ and $\hat{\boldsymbol{A}}$ respectively. $a \leftarrow \chi(S)$ represents that a is sampled from the set S according to the distribution χ, and we use $\leftarrow$ to denote probabilistic output. $a := \chi(S;\, seed_a)$ indicates that $a \in S$ is generated from the $seed_a$ and follows the distribution χ, and we use $:=$ to denote deterministic output. We use $\mathcal{U}$ to denote uniform distribution and β_μ to denote the centered binomial distribution (CBD) with standard deviation $\sqrt{\mu/2}$. We denote the hamming weight function by `HW`. $|x|$ denotes bit length of the bitstring x.

2.1 Learning with Errors Problem

The learning with errors (LWE) problem was introduced by Regev [Reg09] and is as hard as standard worst-case lattice problems [Pei09]. Given $\boldsymbol{A} \leftarrow \mathcal{U}(\mathbb{Z}_q^{(m\times n)})$, $m = O(poly(n))$ $\boldsymbol{s} \leftarrow \chi_1(\mathbb{Z}_q^{(n)})$, and $\boldsymbol{e} \leftarrow \chi_2(\mathbb{Z}_q^{(m)})$, where χ_1 and χ_2 are two narrow distributions. The LWE instance consists of the pair $(\boldsymbol{A},\, \boldsymbol{A}\cdot\boldsymbol{s}+\boldsymbol{e}) \in \mathbb{Z}_q^{(m\times n)} \times \mathbb{Z}_q^{(m)}$. The LWE problem states that for $b \leftarrow \mathcal{U}(\mathbb{Z}_q^{(m)})$, it is hard to distinguish between the following pairs $(\boldsymbol{A},\, \boldsymbol{A}\cdot\boldsymbol{s}+\boldsymbol{e})$ and $(\boldsymbol{A},\, \boldsymbol{b})$. The hardness depends on the distributions χ_1,χ_2 and the parameters q, n. The Ring-LWE (RLWE) [LPR10] and the Module-LWE (MLWE) [LS15] problems are algebraically structured variants of the LWE problem. $\boldsymbol{A}$, $\boldsymbol{s}$, $\boldsymbol{e}$ are polynomials sampled from the ring $R_q^n = \mathbb{Z}_q/(x^n+1)$ in the RLWE problem. In the MLWE problem, $\boldsymbol{A}$ is a matrix of polynomials sampled uniformly from $R_q^{n(l\times l)}$, and $\boldsymbol{s}$, $\boldsymbol{e}$ are vectors of polynomials samples from the set $R_q^{n(l)}$.

2.2 MLWE-based Public-key Encryption

A generic MLWE-based public-key encryption (PKE) is shown in Fig. 2. It consists of three algorithms: (i) key-generation (`PKE.KeyGen`) generates public-key pk and secret-key sk, (ii) encryption (`PKE.Enc`) takes inputs as public-key pk and message m and generates ciphertext c, and (iii) decryption (`PKE.Dec`) takes inputs as ciphertext c and secret-key sk and recovers the encrypted message. This PKE scheme is indistinguishable under chosen plaintext attacks (IND-CPA) based on the assumption of the hardness of the MLWE problem. Here, q is a prime

```
PKE.KeyGen()
1. $seed_{\boldsymbol{A}},\, seed_{\boldsymbol{se}} \leftarrow \mathcal{U}(\{0,1\}^{\text{len}_K})$
2. $\hat{\boldsymbol{A}} := \texttt{PRF}(R_q^{n(l\times l)};\, seed_{\boldsymbol{A}})$   $\triangleright$ $(\hat{\boldsymbol{A}} = \texttt{NTT}(\boldsymbol{A}))$
3. $\boldsymbol{s} := \beta_\eta(R_q^{n(l)};\, seed_{\boldsymbol{se}})$
4. $\boldsymbol{e} := \beta_\eta(R_q^{n(l)};\, seed_{\boldsymbol{se}})$
5. $\hat{\boldsymbol{s}} := \texttt{NTT}(\boldsymbol{s}) \in R_q^{n(l)},\, \hat{\boldsymbol{e}} := \texttt{NTT}(\boldsymbol{e}) \in R_q^{n(l)}$
6. $\hat{\boldsymbol{b}} := (\hat{\boldsymbol{A}} \circ \hat{\boldsymbol{s}} + \hat{\boldsymbol{e}}) \in R_q^{n(l)}$
7. return $(pk := (seed_{\boldsymbol{A}},\, \hat{\boldsymbol{b}}),\, sk := (\hat{\boldsymbol{s}}))$

PKE.Dec($sk := \boldsymbol{s},\, c := (\boldsymbol{u},\, v)$)
1. $\boldsymbol{u'} := \texttt{Decompress}(\boldsymbol{u},\, p) \in R_q^{n(l)}$
2. $v' := \texttt{Decompress}(v,\, t+2^B) \in R_q$
3. $\hat{\boldsymbol{u'}} := \texttt{NTT}(\boldsymbol{u'}) \in R_q^{n(l)}$
4. $\hat{u''} := \hat{\boldsymbol{u'}} \circ \hat{\boldsymbol{s'}} \in R_q^n$
5. $m'' := v' - \texttt{INTT}(\hat{u''}) \in R_q^n$
6. $m' := \texttt{Decode}(m'') \in R_{2^B}$
7. return $m'$

PKE.Enc($pk := (seed_{\boldsymbol{A}},\, \boldsymbol{b}),\, m \in R_2;\, r$)
1. $\hat{\boldsymbol{A}} \leftarrow \texttt{PRF}(R_q^{n(l\times l)};\, \text{seed}_{\boldsymbol{A}})$
2. if $r$ is not specified then $r \leftarrow \mathcal{U}(\{0,1\}^{256})$
3. $\boldsymbol{s'} := \beta_\eta(R_q^{n(l)};\, r)$
4. $\boldsymbol{e'} := \beta_\eta(R_q^{n(l)};\, r),\, e'' := \beta_\eta(R_q^n;\, r)$
5. $\hat{\boldsymbol{s'}} := \texttt{NTT}(\boldsymbol{s'}) \in R_q^{n(l)}$
6. $\hat{\boldsymbol{b'}} := \hat{\boldsymbol{A}}^T \circ \hat{\boldsymbol{s'}}$
7. $\boldsymbol{b'} := (\texttt{INTT}(\hat{\boldsymbol{b'}}) + \boldsymbol{e'}) \in R_q^{n(l)}$
8. $\hat{c_m} := \hat{\boldsymbol{b}}^T \circ \hat{\boldsymbol{s'}}$
9. $c_m := \texttt{INTT}(\hat{c_m}) + e'' + \texttt{Encode}(m) \in R_q^n$
10. $\boldsymbol{u} := \texttt{Compress}(\boldsymbol{b'},\, p) \in R_p^{n(l)}$
11. $v := \texttt{Compress}(c_m,\, t+2^B) \in R_{t+2^B}$
12. return $c := (\boldsymbol{u},\, v)$
```

Figure 2: MLWE based IND-CPA secure PKE using NTT

modulus, and p, t are power-of-two moduli. These algorithms use NTT to perform polynomial multiplication efficiently [LN16]. The $\texttt{Encode}: R_{2^B}^n \longrightarrow R_q^n$ is defined as $\texttt{Encode}(m) = \lfloor \frac{q}{2^B} \rceil m$ and the $\texttt{Decode}: R_q^n \longrightarrow R_{2^B}^n$ is defined as $\texttt{Decode}(m'') = \frac{2^B m'' + \lfloor q/2 \rfloor}{q} \& (2^B - 1)$.

2.3 MLWE-based Key Encapsulation Mechanism

```
KEM.KeyGen()
1. $(pk := (seed_{\boldsymbol{A}},\, \hat{\boldsymbol{b}})$
2. $sk := (\hat{\boldsymbol{s}})) := \texttt{PKE.KeyGen}()$
3. $pkh := \mathcal{H}(pk)$
4. $z \leftarrow \mathcal{U}(\{0,1\}^{\text{len}_K})$
5. return $(\overline{pk} := (seed_{\boldsymbol{A}},\, \hat{\boldsymbol{b}}),\, \overline{sk} := (\hat{\boldsymbol{s}},\, z,\, pkh))$

KEM.Decaps($\overline{sk} := (\hat{\boldsymbol{s}},\, z,\, pkh),\, \overline{pk} := (seed_{\boldsymbol{A}},\, \boldsymbol{b}), c$)
1. $m' := \texttt{PKE.Dec}(\boldsymbol{s},\, c)$
2. $(K',\, r') := \mathcal{G}(pkh,\, m')$
3. $c_* := \texttt{PKE.Enc}(pk,\, m';\, r')$
4. $K'' := \mathcal{H}(c,\, z)$
5. if $c = c_*$ then return $K := K'$
6. else return $K := K''$

KEM.Encaps($\overline{pk} := (seed_{\boldsymbol{A}},\, \boldsymbol{b})$)
1. $m \leftarrow \mathcal{U}(\{0,1\}^{\text{len}_K})$
2. $(K,\, r) := \mathcal{G}(\mathcal{H}(pk),\, m)$
3. $c := \texttt{PKE.Enc}(pk,\, m;\, r)$
4. return $(c,\, K)$
```

Figure 3: MLWE based IND-CCA secure KEM using NTT

The PKE scheme described in Sec. 2.2 is IND-CPA. Indistinguishability under adaptive chosen ciphertext attack (IND-CCA) is a stronger security notion than IND-CPA and is desired to construct a KEM. The IND-CPA PKE Fig. 2 is converted to IND-CCA KEM by applying a variant of Fujisaki–Okamoto (FO) transformation [HHK17]. As the PKE scheme is based on the MLWE problem, the PKE scheme is not perfectly correct (when the decryption of the encrypted message does not return the original message). If the underlying PKE is $(1-\delta)$ correct then the KEM based on the PKE is also $(1-\delta)$ correct [HHK17]. Jiang *et al.* [JZC+17] also showed that given underlying PKE is $(1-\delta)$ correct, then the KEM based on it is S-bit post-quantum secure if $\delta \leq 2^{-S}$. This KEM closely follows the

Frodo-KEM [BCD+16] construction. The IND-CCA MLWE-based KEM consists of three algorithms (i) key-generation (`KEM.KeyGen`), (ii) encapsulation (`KEM.Encaps`), and (iii) decapsulation (`KEM.Decaps`), as shown in Fig. 3. These algorithms use two hash functions namely $\mathcal{G}: \{0,1\}^* \longrightarrow \{0,1\}^{2*\text{len}_K}$ and $\mathcal{H}: \{0,1\}^* \longrightarrow \{0,1\}^{\text{len}_K}$.

2.4 Related works

As described in Sec. 1, lightweight cryptography is essential for devices with limited computational resources such as memory, processing power, energy consumption, etc. Most works on lightweight PKC schemes are based on ECC [HB10, BMS+06, HWF09, BGK+06, SS04] which are not secure against quantum adversaries. Lattice-based constructions are promising candidates for designing lightweight PQC schemes. The NIST standard lattice-based KEMs (e.g. Kyber [BDK+17]) or the finalists of NIST standardization (e.g. Saber [DKRV18]) are mainly designed keeping security and performance in mind. Afterward, LW versions of these schemes were implemented. For reference, Huang *et al.* [HHLW20] Xing *et al.* [XL21], Ni *et al.* [NKLO23] proposed optimized implementations of Kyber in various hardware platforms. Roy *et al.* [RB20] presented an implementation of Saber on FPGA hardware, and Ghosh *et al.* [GMK+22, GMK+23] proposed an area and energy-optimized implementation of Saber in ASIC. There are several hardware or hardware/software co-designs of Kyber available [BSNK19, DFA+20, BUC19].

Buchmann *et al.* [BGG+16] proposed an RLWE-based encryption scheme with binary secret and errors (called Ring-BinLWE scheme) suitable for lightweight PKC applications. Subsequently, more efficient variants of this scheme have been published in the following works [LAM+22, XHW21, EBSMB19, HGX21]. However, these schemes are only IND-CPA and hence vulnerable to the chosen ciphertext attacks (CCA). Later on, Ebrahimi *et al.* [EBS20] propose a CCA secure version of the IND-CPA Ring-BinLWE scheme, but the quantum bit security provided by this scheme is < 75. This is relatively lower in comparison to Saber and Kyber, which provide at least 100-bit PQ security even in their lowest security versions. There is always a trade-off between efficient implementation in the resource-constraint platform and security, and not much work has been dedicated to designing lightweight PQC without compromising security.

During the NIST PQC standardization procedure, a suite consisting of three learning with rounding (a variant of LWE problem) based PQC KEMs, Scabbard, was proposed by Mera *et al.* [MKKV21]. This work explored new design choices, such as a small polynomial size $n = 64$ for one of the schemes (Espada) to reduce the memory footprint of the implementation in the resource constraint Cortex-M4 device. Before that, the smallest polynomial size n used in LWE-based KEM was 256. In addition to this, several new designs of LWE-based KEMs, such as Smaug [CCHY23], TiGER [PJP+22], etc., have been submitted in the ongoing Korean PQC competition [Kpq]. Although the aforementioned works improved the state-of-the-art of LBC with different design choices and implementations, none of them explored all the possible design choices of LWE-based KEMs from the perspective of lightweight hardware implementations.

3 Rudraksh: Design Space Exploration

Designing a cryptographic scheme is fundamentally solving an optimization problem where the major objective functions are attaining a particular level of security, reducing latency, and reducing bandwidth (the size of the public key, ciphertext, and secret key). However, for our lightweight design, we additionally impose new constraints, such as low memory, low energy, and area requirements, to execute the scheme with reasonable latency. Our LBC design is influenced primarily by 3 parameters, the structure of the module *i.e* the rank of the matrix l and the size of the constituent polynomials n, the prime modulus q, and the standard deviation σ of the secret or error distribution. In this section, we discuss our design decisions and the rationale behind them in choosing these variables to achieve our design objective of a lightweight KEM.

3.1 Module space exploration

Keeping the q and σ fixed, the security of standard, module, or ideal lattice-based cryptographic scheme is dependent on the rank n' of the underlying matrix $\boldsymbol{A}$ only. Module lattices present a convenient and generic representation of different lattices; therefore, in this section, we will use the modules to describe different types of lattices. Let's consider a module lattice $\boldsymbol{A} \in R_q^{n(l\times l)}$. The rank of the underlying matrix is $n' = l \times n$. Fixing $n = 1$ and $l = n'$, we get a standard lattice. On the other hand, if we fix $l = 1$ and $n' = n$ *i.e.* our lattice consists of a single polynomial, and we get an ideal lattice (ring lattice). Indeed, for a long time before the proposal [LS15] of module lattices, these two extremities were the only two choices available to design lattice-based cryptosystems, as shown in Fig. 4. Although Kyber [BDK+17], Saber [DKRV18], and Dilithium [DKL+18] are prominent examples of cryptographic schemes based on module lattices, a vast spectrum of lattice configurations with different values of l or n has been left unexplored. This is shown in Fig. 4 as a grey-shaded region. We explore this region to find optimal choices for n and l to design a lightweight KEM. It should be noted that this is not trivial. Intuitively, one might think that choosing a small n would immediately lead to a lightweight design as it reduces the size of the multiplier. However, to maintain the n' for the security, decreasing n increases l. This implies more multiplications, more random numbers, larger moduli, etc. Similarly, just decreasing l is also not useful for LW designs. We have to strike a delicate balance between l and n and other metrics that influence the scheme's suitability for small resource-constrained devices. We discuss these different metrics and how they are affected by different values of n and l below.
Memory consumption: The matrix-vector multiplication is performed in `PKE.KeyGen` (therefore in `KEM.KeyGen`) and `PKE.Enc` (therefore in `KEM.Encaps` and `KEM.Decaps`) algorithm (shown in Fig. 2). The storage requirement for the public-matrix $\boldsymbol{A}$ is one of the most memory-expensive operations for the LWE schemes that use module lattice structure. It requires to store $l \times l$ polynomials of degree $n-1$. Currently, the *de-facto* standard of lattice-based implementation is to generate this matrix using the *just-in-time* [KMRV18] strategy. This method generates the matrix $\boldsymbol{A}$ one polynomial at a time by utilizing the sponge-based '..squeeze-absorb-squeeze..' operation of the extended output function (XOF) function such as Keccak [BDPVA13]. Therefore, the memory requirement to perform the matrix-vector multiplication is proportional to the size of one single polynomial. As we move towards the left of Fig. 4, polynomial size n decreases. So, although l has to be increased to maintain the security level, the memory requirement in this configuration is smaller. We store a single polynomial for all the polynomial multiplications in hardware platforms. Of course, one can take extreme measures such as generating a single coefficient at a time and performing a single integer multiplication to reduce memory. However, it would drastically deteriorate performance. Therefore, we explore possible MLWE schemes with smaller polynomial sizes, which can be implemented with low resources without much performance degradation. We primarily target schemes where n is power-of-2 and less than 256, such that 128, 64, and 32.
Multiplier size: In the case of a standard lattice-based scheme with matrix rank n', one of the most computation-heavy operations is multiplications between $n' \times n'$ matrix and n' length vector. For the ring lattice-based scheme, we have to perform polynomial multiplications between two $n'-1$ degree polynomials to achieve a similar level of security. This can be done using quasi-linear NTT multiplication. Hence, for a particular security level, the ring lattice-based schemes are more efficient than the standard lattice-based schemes in terms of computational cost. However, the resource consumption in that case is relatively huge as two $n'-1$ degree polynomials must be stored to perform the polynomial multiplication. Therefore, for a specific security level, due to *just-in-time* strategy, the module lattice-based schemes are more beneficial to reducing resource consumption than the ring lattice-based schemes. Although we have to perform multiple polynomial multiplications

due to the use of module structure, module lattice-based schemes perform better than the standard lattice-based schemes. The choice of the hard problem *i.e.* MLWE or MLWR, and polynomial size determines the size of the multiplier in hardware. Nevertheless, the resource consumption is proportional to the size of a single polynomial for module lattice-based schemes. Therefore, choosing the hard problem and polynomial size is one of the leading factors when designing an efficient scheme for resource-constrained devices.

Generally, the size of the polynomial n is chosen in multiple with the size of the secret message bit-length len_K [ADPS16, BDK+17, CKLS18]. If $n = B' * \text{len}_K$ for an integer $B' > 0$, we can use B' coefficients of ciphertext polynomial v (generated during from encryption algorithm shown in Fig. 2) to hide a single message bit by replicating a single message-bit B' times. But if the polynomial-size n is smaller than the size of the message bit-length len_K *i.e.* $n = (1/B) * \text{len}_K$ for an integer $B > 1$, then we have to encode B message bits into a single coefficient of ciphertext polynomial v [BCD+16] as displayed in Fig. 2 (line 8, in the `PKE.Enc` algorithm). This would increase the requirement of the reconciliation bits ($\log_2 t$) and eventually the modulus of a coefficient of v ($= \log_2 t + B$). This will require a larger modulus q, which reduces the security. We will discuss this phenomenon in more detail in Sec. 3.2.

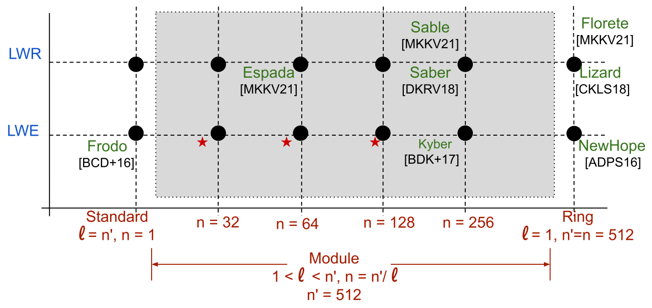

Figure 4: Design space of lattice-based KEMs depending on the different underlying matrix sizes. Our explored module spaces are marked with red stars.

Flexible design: Module lattices present an opportunity to design cryptographic schemes that benefit the IoT architecture as discussed in Sec. 1. The structure of module lattices can be utilized to instantiate such a flexible scheme. Let's say a cryptographic scheme uses a module lattice of size $l \times l$. An extremely low latency and high resource-consuming implementation can be realized by implementing l^2 multipliers in parallel, and an extremely lightweight and high latency version can be realized by implementing a single multiplier repeatedly for l^2 times. Here, of course, the XOF has to be implemented accordingly to match the latency of the multiplier. As discussed before, polynomial arithmetic, specifically polynomial multiplication, is one of the major bottlenecks in LBC's performance and resource consumption. The polynomial size in both Kyber and Dilithium is 256. Even in the most lightweight instantiation, an IoT peripheral device has to use a 256×256 polynomial multiplier. This is still very expensive for an IoT peripheral device. A smaller polynomial size is more suitable with some sacrifice in efficiency. Therefore, a balance has to be struck between these two metrics for a suitable lattice-based PQ scheme for IoT.

In conclusion, ring lattice-based schemes are especially advantageous for achieving better performance but demand much hardware area. Meanwhile, standard lattice-based schemes theoretically can be implemented with lesser memory and area at the expense of substantial computation costs. Module lattice-based schemes with polynomial-size n and underlying lattice's matrix rank $n' = l \times n$ provide a trade-off between them. If we keep n' constant, increments of l decrease n, which reduces the memory requirements for a single polynomial. However, this increases the generation cost of the matrix $\boldsymbol{A}$, as the matrix consists of $l \times l \times n \times \lceil \log_2 q \rceil$ pseudo-random bits. These pseudo-random numbers are generated by using the XOF function, which is another computationally expensive operation as described later in Sec. 3.5. If memory is not a concern, then increments of l can be used to increase parallelism in hardware implementations. There are some module LWR-based designs that have been proposed in recent years [DKRV18, MKKV21]. However, the module-space for designing different MLWE-based schemes remained mostly unexplored. Therefore, we choose to explore the module space for the LWE problem denoted by red colored stars in Fig. 4 to construct a lightweight MLWE-based KEM with optimal parameters.

3.2 Choice of moduli

It is clear from the discussion in the previous section that we want to explore the module lattice space to design lattice-based KEM. LWE-based schemes use reconciliation mechanisms by sending some extra bits [BDK+17]. These bits help to recover the encrypted message during the decryption algorithm by reducing the noise introduced during the encryption procedure called decryption noise (as LWE-based encryption schemes are not perfect). Increased decryption noise induces an increment in the failure probability, which can cause a decryption failure attack [DGJ+19]. As discussed earlier, a smaller polynomial size n reduces the memory consumed by a single polynomial and also the area required to implement single polynomial multiplication in hardware. But, if we reduce the size of the polynomial n, then we have to encode multiple message bits in a coefficient of v. This will increase the failure probability. This can be compensated by more reconciliation bits which in turn increases q.

The security of a lattice-based cryptosystem increases with the increase in the error-to-modulus ratio, *i.e.* keeping the error distribution fixed, the security will reduce with the increase in the value of q, and vice versa. Therefore, a smaller value of q helps to increase efficiency, reduce computational and storage resources, and also reduce the bandwidth (size of the public-key, secret-key, and ciphertext), but increases the failure probability. Hence, we concentrated on finding an optimal value of the modulus q for which the decryption failure would be minimal. We also proposed implementations with minimal hardware resources.

The type of the modulus *i.e* prime vs. power-of-2 modulus also has a significant impact on the performance and resource utilization of the scheme. We have deferred this discussion till Sec. 3.4.

3.3 Secret and error distribution

In LWE-based schemes, coefficients of secret and error are usually sampled from a narrow distribution. There are several (M/R)LWE-based KEMs that have utilized discrete Gaussian distribution as secret and error distribution [BCD+16]. Unfortunately, it is hard to implement a Gaussian sampler efficiently and securely against timing attacks. For KEMs, Alkim *et al.* [ADPS16] showed that this Gaussian distribution can be replaced with a centered binomial distribution (CBD) whose standard deviation is the same as the Gaussian distribution. The sampling from a CBD is much simpler and easier to protect against side-channel attacks.

Several other distributions have been explored in the design of LWE-based schemes to gain efficiency, such as binary distribution [BGG+16], fixed weight distribution [BBF+19, CCHY23], etc. In the binary distribution, the secret and error have only two possible values $\{0, 1\}$. If we use this binary distribution for secret, then the standard deviation of the secret is low. This implies security will be reduced, and to diminish that, we need to increase the rank of the underlying lattice matrix. In the fixed weight distribution, the hamming weight of the secret/error polynomial is fixed. The security of these distributions is yet to be thoroughly investigated and does not elicit enough confidence. Due to these reasons, we refrained from aggressive design choices such as the small CBD β_1, binary distribution, or fixed weight distribution as the secret or error distribution to attain better efficiency. We have used a CBD β_μ with $\mu \geq 2$ to alleviate these issues in our design.

The sampling from a CBD β_μ is accomplished by performing $\texttt{HW}(a) - \texttt{HW}(b)$, where a, b are μ bit pseudo-random numbers. The parameter μ of CBD is crucial in deciding the scheme's efficiency. This also impacts the security parameters of a CCA secure scheme, failure probability, and bit security. The CBD sampler uses pseudo-random numbers, and the bigger the CBD parameter μ, the more pseudo-random numbers generation will be required. Pseudo-random numbers are generated using some extendable-output function (XOF). The XOF is one of the costliest operations in terms of computation and resources (Details about XOF have been provided in the next subsection). Finding a smaller μ is necessary for better performance, less resource utilization, and lower failure probability. However, a larger μ increases the bit security. As we are aiming for a CCA-secure KEM scheme with 100-bit PQ security using Hofheinz *et al.*-version of FO transform [HHK17], the failure probability must be $\leq 2^{-100}$.

Hence, in our design, we have analyzed these aspects with a wide range of values of μ to find an optimal choice to strike a balance between the security and efficiency of our scheme.

3.4 Choice of polynomial multiplication

For LBC schemes, polynomial multiplication is one of the major bottlenecks with respect to efficiency and resource consumption. In literature, there exist mainly two types of polynomial multiplication algorithms for implementing LBC schemes: i) Toom-Cook multiplication [Too63, Coo66], and (ii) NTT multiplication [LN16]. Toom-Cook multiplication is relatively simpler and can be used for any modulus. However, the time complexity of the Toom-Cook polynomial multiplication is asymptotically slower $O(n^{1+\epsilon})$, where $0<\epsilon<1$. On the other hand, NTT multiplication is the most used polynomial multiplication for implementing LBC schemes due to its faster quasi-linear time complexity ($O(n\log_2 n)$). However, for the NTT multiplication, the modulus q needs to *NTT friendly i.e.* a prime number with the primitive $2n$-th root of unity in the prime field $\mathbb{Z}_q$. Please note that, for small values of n, the efficiency of Toom-Cook polynomial multiplication with a power-of-2 modulus (as prime reduction is free in a power-of-2 ring) and NTT-based polynomial multiplication on an appropriate prime modulus is comparable when performed the full multiplication. However, for LBC schemes, we can sample the public matrix $\texttt{NTT}(\boldsymbol{A})=\hat{\boldsymbol{A}}$ from $R_q^{n(l\times l)}$ using $seed_{\boldsymbol{A}}$ instead of sampling $\boldsymbol{A}$ and perform NTT on $\boldsymbol{A}$. It can save cycles while computing $\boldsymbol{A}\cdot\boldsymbol{s}$ as $\boldsymbol{A}$ is random implies $NTT(\boldsymbol{A})$ is random and vice-versa. We also can save execution time on the NTT-based polynomial multiplication by omitting the INTT operation and keeping the multiplication result in the NTT domain (e.g., Line (5) in the PKE.KeyGen() in Fig. 2). This makes the LBC scheme with NTT multiplication more efficient than Toom-Cook multiplication. Therefore, we choose to use NTT-based polynomial multiplication over a prime moduli q.

NTT multiplication between two polynomials a and b from R_q^n is performed by $a\cdot b=\texttt{INTT}(\texttt{NTT}(a)\circ\texttt{NTT}(b))$. We denote point-wise multiplication (PWM) with $\circ$. Given ζ is the $2n$-th primitive root of unity and $\omega=\zeta^2$, the $\texttt{NTT}(x)=\hat{x}=(\hat{x_0},\ \hat{x_1},\ \ldots,\hat{x_{n-1}})$ and $\texttt{INTT}(\hat{x})=x=(x_0,\ x_1,\ \ldots,x_{n-1})$ are denoted by the following Eq. 1 & 2.

$$\hat{x_i}=\sum_{j=1}^{n-1}x_j\zeta^{(2i+1)j}=\sum_{j=1}^{n-1}(x_j\zeta^j)\omega^{ij} \bmod q,\ 0\leq i\leq n-1. \tag{1}$$

$$x_j=1/n\sum_{i=1}^{n-1}\hat{x}_i\zeta^{-(2i+1)j}=\zeta^j/n\sum_{i=1}^{n-1}\hat{x}_i\omega^{-ij} \bmod q,\ 0\leq j\leq n-1. \tag{2}$$

In this procedure of multiplication, we have to store the pre-computed values of $\zeta^j \bmod q$ (for $1\leq j\leq n-1$) along with the coefficients of two participated polynomials for improving the performance. Therefore, the total memory requirement to perform multiplication depends on polynomial size n.

In the literature, there is another type of NTT called incomplete NTT multiplication, where the NTT multiplication between two n size polynomials is replaced by two separate NTT multiplications between two $n/2$ size polynomials. Karatsuba multiplication is performed on the last 1 degree polynomials. Therefore, incomplete NTT multiplication requires more modular multiplications than complete NTT multiplication. Yet, the incomplete NTT multiplications outperform complete NTT multiplications [AABCG20] on software by omitting several reduction steps after modular multiplication. Incomplete NTT multiplication is also used in Kyber. We employ a single butterfly module shown in Fig. 8 that includes a reduction step for performing NTT, INTT, and PWM in our KEM. Therefore, incomplete NTT multiplication increases the latency in our KEM implementation. Consequently, we choose the complete NTT multiplication over the incomplete one for polynomial multiplication. More details regarding NTT multiplication are provided in Sec 4.2.

3.5 ASCON based hash and XOF functions

The implementations of LBC exhibit a unique and interesting phenomenon. Lattice-based cryptographic schemes use a lot of pseudorandom numbers to generate the matrix $\hat{\boldsymbol{A}}$ and the secret $\boldsymbol{s}$ and error $\boldsymbol{e}$ vectors. From a designer's perspective, generating pseudorandom numbers is considered an auxiliary function that does not impose major overhead on executing the whole scheme. More focus is given to optimizing the core functions of the cryptographic scheme as they consume the majority of the time and resources in the implementation. This is true for classical PKC (and symmetric-key ciphers also) schemes such as RSA [RSA78] and ECC [Mil86], where most of the time and resources are spent on making the multi-precision multiplications and scalar point multiplication, respectively. However, for LBC, the standard approach of generating pseudorandom numbers is using an XOF such as Keccak [BDPVA13]. This process takes close to or, in some cases, more than 50% of total time and/or area [XL21]. As numerous works have been done to optimize the core operation of LBC, which is polynomial multiplication in software and hardware platforms [RB20, MTK$^+$20], the process of random number generation has become the bottleneck.

To alleviate this problem, designers have proposed alternate versions of their schemes, such as Kyber-90s [ABD$^+$21] or Saber-90s [BMD$^+$21] where they proposed to generate the random numbers using a block cipher (such as AES [Can05]) in counter mode. While this could be a good solution for software and hardware platforms with dedicated support for these block ciphers, such as the AES-NI instruction set, for standalone hardware with minimal software support, this is not a good solution. As shown in Fig. 3 (for $\mathcal{G}$, $\mathcal{H}$), we need to use the hash function SHA3 or Keccak for a CCA-secure KEM using FO transformation. Therefore, we cannot completely remove the Keccak module from the hardware platform. Moreover, we must include another module implementing the block cipher algorithm.

Therefore, the best possible solution in this scenario is to replace the *bulky* Keccak module with some lightweight alternative. Incidentally, NIST concluded its lightweight cryptography competition [NIS23a] in February 2023 and selected the ASCON [DEMS12] family of lightweight ciphers. Like the Keccak, ASCON is also based on the sponge construction [BDPA11] and can be used as a Hash function and XOF. Moreover, ASCON is specifically designed for lightweight implementation on resource-constraint devices. This makes ASCON an ideal choice for replacing the Keccak function in our design. However, this is not very straightforward. The biggest hurdle is the difference in the state size of these two ciphers, which are 320 and 1600 for ASCON and Keccak, respectively. Therefore, each ASCON-squeeze outputs a fraction of pseudorandom bits compared to a Keccak-squeeze. Hence, to utilize ASCON's full potential, we have carefully designed our architecture exploiting the lightweight sub-layer and linear layer of ASCON, as well as meticulous scheduling and memory organization in the FPGA implementation, whose throughput does not become the operation bottleneck (explained in Sec. 4.3). Another hurdle in replacing Keccak with ASCON is that the current version of ASCON provides maximum 128-bit security; therefore, it is unsuited to replace SHAKE-256 or SHA3-512, which has been used in Kyber. However, it is fine for our lightweight design. Due to these issues, although the use of ASCON in place of Keccak has been deliberated for a long time, we have not seen any hardware/software implementation yet. To the best of our knowledge, we are the first to use and demonstrate the efficiency and utility of ASCON in any cryptographic implementation.

Table 1: Parameter set of all the explored designs of KEMs together with Kyber and NewHope for NIST-I security level

NIST security level	Module Parameter		Primary modulus		Compression modulus		CBD parameter		Encoding	Security	Failure probability
	l	n	q	$\lceil \log_2 q \rceil$	$\lceil \log_2 p \rceil$	$\lceil \log_2 t \rceil$	η_1	η_2	B	(Quantum, Classical)	
KEM-poly32	21	32	31873	15	12	3	2	2	4	(105, 116)	-113
KEM-poly64	9	64	7681	13	10	3	2	2	2	(104, 114)	-128
KEM-poly128	4	128	3329	12	10	2	2	2	1	(101, 111)	-179
Kyber [ABD+21]	2	256	3329	12	10	3	3	2	1	(107, 118)	-139
NewHope [ADPS16]	1	512	12289	14	14	2	4	4	1	(101, 112)	-213

3.6 Parameters of our scheme

We target to attain at least 100-bit post-quantum security for our lightweight KEM. It will provide equivalent security with AES-128 [AAC+22] and belong to the NIST-I security category. Therefore, the current version of ASCON with 128 bit security is enough for us. In this section, we discuss the process of finding parameters for Rudraksh.

Dachman-Soled *et al.*'s leaky-LWE estimator [DSDGR20] is the state-of-the-art tool to estimate the hardness of the underlying LWE problem. It uses the best-known lattice reduction algorithm Block Korkine-Zolotarev (BKZ) [SE94, CN11] algorithm. BKZ algorithm primarily estimates the difficulty of solving the shortest vector problem (SVP) in a smaller lattice. This is known as core-SVP hardness. The security of the overall LBC scheme is the hardness of this core-SVP problem with some polynomial overhead. Usually, we ignore this polynomial overhead for a pessimistic estimate of security. The leaky-LWE estimator tool takes the underlying base matrix rank $n' = l \times n$, the modulus q, and the standard deviation of secret or error distributions of a scheme as input. It returns both post-quantum and classical bit security of the corresponding scheme. As discussed earlier, while designing a module lattice-based scheme for resource constraint devices, the two most important parameters are the polynomial-size n and the length of the vector l. We have viewed finding the optimal parameter set for our lightweight KEM as a multi-dimensional optimization problem. First, we fixed the polynomial-size n and exhaustively searched all the possible values of other parameters such as the vector length l, modulus q, and CBD parameter η_1, η_2, etc. This is followed by calculating the resource consumption for these parameters. We have repeated the process for all the power-of-2 polynomial sizes to maintain the efficiency of the scheme as it is beneficial for the implementation of several primary building blocks, such as NTT multiplication, `Encode`, `Decode` functions, etc. We also did not explore the polynomial-size below $n = 32$, as the failure probability increases in these cases drastically. We have to increase the length of the vector quite a lot to counteract the high failure probability, which affects the scheme's efficiency. We provide optimal parameter sets for three configurations (i) `KEM-poly32`: with polynomial-size 32, (ii) `KEM-poly64`: with polynomial-size 64, (iii) `KEM-poly128`: with polynomial-size 128. The parameters of all these configurations are shown in Tab. 1. This table also includes the parameters of Kyber [ABD+21], where the $n = 256$, and NewHope [ADPS16], where $n = 512 = n'$. We also provide the process to find parameters for `KEM-poly64` in Fig. 5.

Memory and area are two primary benchmarks for hardware resource consumption. The memory possesses all the resources used for data usage, which includes all the on-chip memory structures such as Block-RAM (BRAM), distributed RAM, etc. The area contains the configuration logic resources, including look-up tables (LUTs) and logical elements. We discuss the estimated hardware resource usage of all the configurations proposed in Tab. 1 in terms of memory and select the one that can be operated with optimal resources. Each polynomial of the secret-key vector $\boldsymbol{s}$ can be generated from $\texttt{seed}_{\boldsymbol{s}}$. These secret polynomials generate the public-key vector $\hat{\boldsymbol{b}}$ during the key-generation procedure or used in the `PKE.Dec` (Fig 2, line (3) in `PKE.KeyGen`) during the decapsulation algorithm. One polynomial length ($n \times \lceil \log_2 q \rceil$ bits) memory storage is required for the secret polynomial (for $\boldsymbol{s}$ in `PKE.KeyGen`, for $\boldsymbol{s'}$ in `PKE.Enc`), and for the runtime calculation of single polynomial in the public matrix $\hat{\boldsymbol{A}}$. For efficient implementation, another polynomial storage is required for the n roots of unity (or twiddle

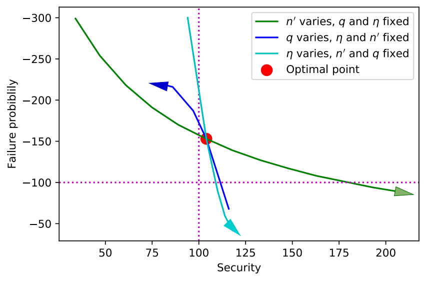

Figure 5: Relation between n', q, and η (η_1/η_2) when $n = 64$ is fixed (arrows indicate the direction of increase in values). The parameter set of the optimal point is selected for `KEM-poly64`.

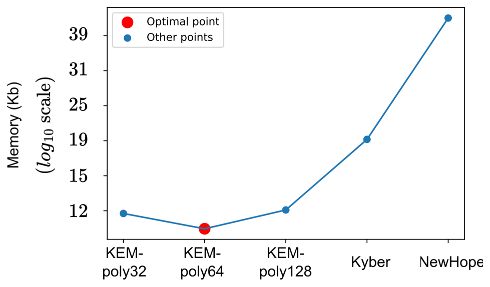

Figure 6: Memory consumption of the KEM depending on the polynomial size

factors). We use one more polynomial space to save the ciphertext v as well as a vector of polynomial space ($l \times n \times \lceil \log_2 q \rceil$ bits) to store the public vector $\hat{\boldsymbol{b}}$ in the key-generation algorithm, and that same space is used to store the ciphertext vector $\boldsymbol{u}$ during `PKE.Enc`. We also need 320 bit ASCON state register for `KEM-poly32`, `KEM-poly64`, `KEM-poly128`, and 1600 bit state register of Keccak for Kyber and NewHope. Extra buffer is often used for post-processing for hash (for K in Encapsulation and K', K'' in decapsulation) and the pseudorandom number z generated during key generation and used in decapsulation algorithm (for the cases of decryption failure). This buffer size is equivalent to the state register. Therefore, we need memory for four polynomials, one vector of polynomials, states of ASCON or Keccak, and storage for hash output and z. We calculate the storage requirement for each of the configurations of Tab. 1 and present them with the help of Fig. 6. It is evident from Fig. 6 that `KEM-poly64` uses the least storage compared to other configurations. Therefore, we select `KEM-poly64` as our lightweight KEM Rudraksh and present a lightweight (low resource) implementation on hardware.

4 Hardware design

After exploring the theoretical design decisions for developing a lightweight lattice-based KEM, we efficiently implement the scheme and show the scheme's area and latency requirements in hardware. We have chosen Xilinx Virtex-7 FPGA as our target device. Our hardware design consists of several components. First, we discuss the system architecture; second, we discuss the re-configurable butterfly architecture; and finally, we discuss the datapath of the ASCON-based XOF function. We also discuss the other computational units, such as the CBD sampler, rejection sampling unit, etc. We also discuss our efficient memory organization, as careful memory organization is crucial for memory reduction. Note that reducing memory is important for lightweight design as often use cases are memory-constrained IoT devices. They will often share the memory with other systems, and careful memory-reduced design is of utmost importance. Efficient NTT memory access removes the need to reorganize memory, which increases latency overhead. Finally, we describe the scheduling during all the operations.

4.1 System architecture

The full system architecture is shown in Fig. 7. The seeds of matrix $\hat{\boldsymbol{A}}$, secret $\boldsymbol{s}$, and error $\boldsymbol{e}$ are taken from an external True Random Number Generator for demonstration. A controller FSM controls ASCON-XOF and butterfly units synchronously. It enables/gates the CBD/rejection sampler when not required. For example, sampling $\hat{\boldsymbol{A}}$ does not require CBD and hence is gated by the controller. The controller also provides an address offset to avoid memory collision with the public/secret key if we want to store them. The ASCON permutation is used for the $\mathcal{H}$, $\mathcal{G}$, and `PRF` functions. It is the key function to generate the public matrix of the polynomials $\hat{\boldsymbol{A}}$. Each coefficient of the polynomials is $\lceil \log_2 q \rceil$ bit and less than prime modulo q. Therefore, an additional rejection sampler is needed to discard coefficients, $\geq q$, while generating $\boldsymbol{A}$. The ASCON core is also used to generate the pseudo-random number for the CBD sampler module to construct the secret $\boldsymbol{s}$ (or $\boldsymbol{s'}$). The secret is directly stored in 2 NTT memories. The reconfigurable butterfly unit performs the NTT/INTT/point-wise multiplication(PWM) operation. Synchronous memory access for NTT and INTT are ensured in the design. We discuss the details in Sec. 4.4. PWM is a vector multiplication as the polynomial-size is small (64). We use complete NTT multiplication, unlike Kyber, where PWM is complex as it uses incomplete NTT multiplication along with Karatsuba multiplication in the last step as part of PWM.

4.2 Reconfigurable butterfly unit

We introduce a reconfigurable butterfly unit for NTT, INTT, PWM, compress, decompress, encode, and decode functions. The butterfly unit is configured by the 3-bit mode signal provided by the controller (Fig. 8). A single DSP unit is used for multiplication. Notably, multiplication is a key operation in all the previous computations and takes a significant

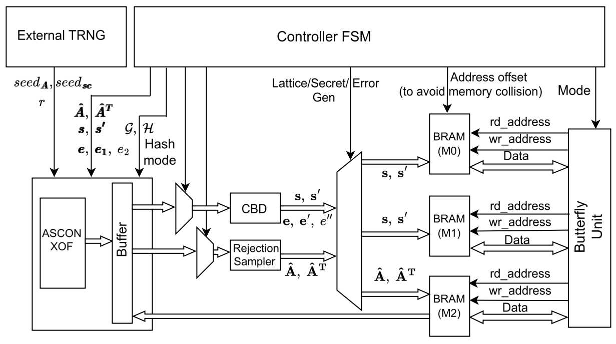

Figure 7: Full system architecture

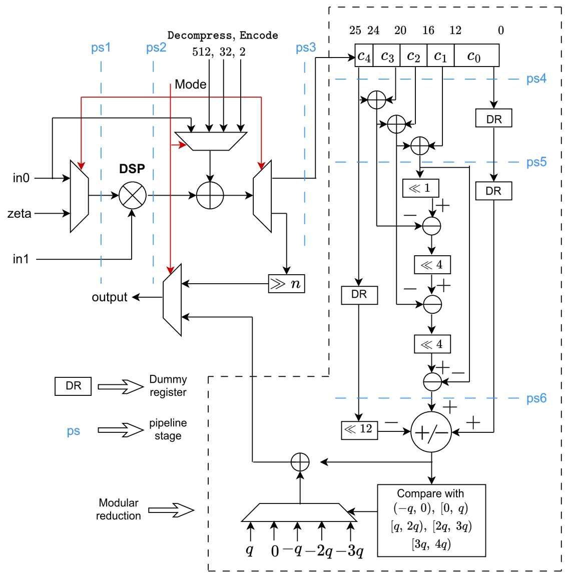

Figure 8: Architecture of the butterfly module

area. The DSP unit performs the multiplication with the twiddle factor ($\zeta^j \bmod q$ where $1 \leq j \leq n-1$ and `zeta`) in NTT/INTT operations. We adapted Zhang et al. [ZYC+20] technique for INTT. Here, the multiplication with $(1/n) \bmod q$ of Eq. 2 is replaced by the $(1/2) \bmod q$ in each butterfly operation. This step eliminated complex multiplication by 1-bit left shift operation at negligible hardware cost. The butterfly unit also consists of an adder/subtractor, as shown in Fig. 8. Finally, it includes a three-stage pipelined shift-and-add modular reduction, ensuring a very low critical path for the design. It is important to note that ASCON-XOF is extremely lightweight, and its substitution layer consists of a few 'xor' and 'and' gates. Hence, we use 6-stage pipelines in the butterfly to maintain a low critical path, resulting in a high-frequency design.

Only the multiplication and the modular reduction are enabled, while the butterfly operates in PWM mode. We employ the butterfly unit to compute $\texttt{compress}(\boldsymbol{b'}, 1024)$, defined by $\frac{(\boldsymbol{b'} \ll 10)+q/2}{q}$ followed by keeping lower 10 bits (similar to [ABD+20]). The division by q is replaced by multiplication with an approximate value of $1/q$. This procedure includes multiplication with $(2^{32}/q+1)$ followed by 32 bit right shift. We use a similar technique to compute $\texttt{compress}(c_m, 32)$. Here, we substitute the division by q with multiplication with $(2^{27}/q+1)$ followed by 27 bit right shift. While computing $\texttt{Decode}(m)$, we replace the division by q with multiplication with $(2^{30}/q+1)$ and followed by 30 bit right shift. $\texttt{decompress}(\boldsymbol{u}, 1024) = (q \cdot \boldsymbol{u} + 512) \gg 10$, $\texttt{decompress}(v, 32) = (q \cdot v + 16) \gg 5$, and $\texttt{encode}(m) = (q \cdot m + 2) \gg 2$ involve a multiplication with q. All the right shift operations are implemented using a configurable barrel shifter. We describe modular reduction hardware in detail below.

Modular reduction: Modular reduction is one of the crucial parts of the butterfly core. Some of the primarily utilized modulus reduction algorithms are Montgomery reduction [BDK+17] and Barrett reduction [XL21]. Later, for primes like $q = 2^m \times k + 1$, where k is an odd number, the k-reduction [NDG19] algorithm has been proposed. This reduction algorithm performs better and consumes less area on hardware than the Montgomery or Barrett reduction. Subsequently, the k^2-reduction algorithm is introduced by Bisheh-Niasar et al. [BAK21], which is more efficient than the k-reduction algorithm. k-reduction algorithm takes input c and outputs $d \equiv k \times c \bmod q$, and k^2-reduction algorithm takes input c and outputs $d \equiv k^2 \times c \bmod q$. We can eliminate the extra k^s, $s \in \{1,\ 2\}$, by replacing the pre-computed factor `zeta` by $k^{-s} \times \texttt{zeta}$ during NTT or INTT. However, while using the k^s-reduction during point-wise multiplication, we have to perform one extra multiplication followed by the reduction to discard the extra k^s factor. It either increases the number of DSPs (containing one multiplication unit) or the latency. However, this extra step is unnecessary if we use the shift-and-add modular reduction technique. We used this technique for our prime $q = 7681$ and presented it in Alg. 1. The detailed implementation is shown on the right side of the butterfly unit (Fig. 4.2). The modular reduction only needs addition/subtraction and bit shift operation, making it extremely lightweight and suitable for high-frequency operations.

Algorithm 1: Shift-and-add modular reduction

```
Input  : $c$ is an integer $\in [0,\ (q-1)^2]$
Output : $d$, where $d \equiv c \bmod q$
$c = c_4||c_3||c_2||c_1||c_0$                                   $\triangleright |c_0| = 13,\ |c_1| = |c_2| = |c_3| = 4,\ |c_4| = 1$
$temp_0 = c_4 + c_3$; $temp_1 = temp_0 + c_2$; $temp_2 = temp_1 + c_1$
$temp_3 = (temp_2 \ll 1) - temp_0$; $temp_4 = (temp_3 \ll 4) - temp_1$
$temp_5 = (temp_4 \ll 4) - temp_2$; $temp_6 = temp_5 + c_0$
$res = (-c_4 \ll 12) + temp_6$
if $((d[15]) == 1)$ then $d += q$
if $(d > q)$ then $d -= q$
if $(d > q)$ then $d -= q$
if $(d > q)$ then $d -= q$
return $d$
```

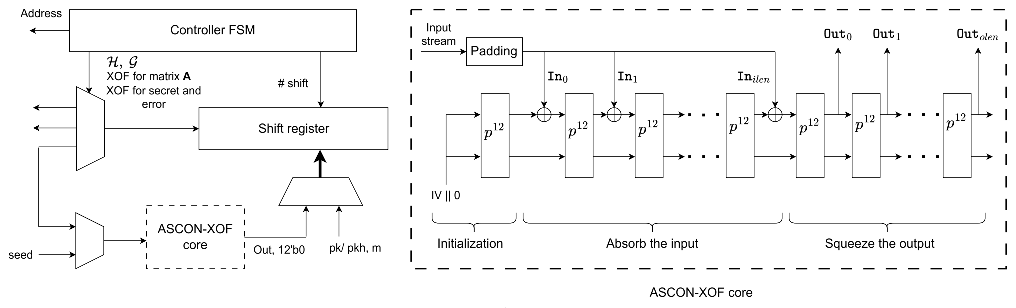

Figure 9: Structure of ASCON XOF hardware

4.3 ASCON core as XOF function

ASCON-XOF takes an input of arbitrary size in chunks of 64 bits and generates an output with variable lengths (in chunks of 64 bits). ASCON is also a sponge-based construction like Keccak. It offers a smaller state size of 320 bits (64-bit rate and 256-bit capacity), whereas the Keccak state register size is 1600 bits. ASCON-XOF can be implemented with low-area (Fig. 9). It supports several functional modes which are `PRF` to generate the public-matrix $\hat{\boldsymbol{A}}$, the CBD sampler to generate the secret $\boldsymbol{s}$ (or $\boldsymbol{s'}$), error $\boldsymbol{e}$ (or $\boldsymbol{e'}$,e''), and $\mathcal{G}$, $\mathcal{H}$. The global controller fixes the mode for this block.

ASCON-XOF function has three steps: a) initialization, b) absorb, and c) squeeze. The initial state register is pre-computed from `IV` in our design to save latency. The second step is to absorb the input stream in the block of 64 bits. In Fig. 9, `ilen` denotes $\lceil \frac{\text{input length}+1}{64} \rceil$. During absorb, 64 bits input block is XORed with the first 64 bits of the state register followed by p^{12}. The third step is to squeeze the output bits. This process continues until the required length of output is extracted. We denote $\lceil \frac{\text{output length}}{64} \rceil$ by `olen` in the figure. ASCON permutation p^{12} is the primary building block of the ASCON-XOF function. It is used during all the three steps. This permutation consists of three steps: (i) addition of constant round, (ii) substitution layer, and (iii) linear diffusion layer [DEMS12].

The rate of input and output block of ASCON is only 64-bit. ASCON's permutation p^{12} includes only bit-wise XOR, circular shift, and bit-wise AND operations. Therefore, single permutation takes considerably less number of gates than Keccak. Moreover, this implies that the critical path of the ASCON permutation is small and, hence, increases the maximum frequency. The execution of the ASCON permutation at a higher frequency compensates for high clock cycle consumption during absorb and squeeze functions. It helps this design compute with a similar order of latency as Kyber. This design decision assists in attaining an efficient performance with reduced area usage and makes it especially suitable for lightweight designs.

This ASCON-XOF hardware also contains a 76-bit buffer/shift register. This buffer stores the input of the absorb while computing $\mathcal{H}(pk)$. Each coefficient of the vector of polynomials is 13 bits, and we save the pk in coefficient format one by one. Once a minimum of 64 bits is stored, those bits are used for absorption while new coefficients are introduced in the buffer. The ASCON absorb's input block size is 64 bits (determined by ASCON rate). We load the first 5 coefficients of pk (65 bits) to the shift register for the first absorb. The input block of the first absorb step consists of the first 4 coefficient and 12 bits from the lowest significant bits (LSB) of the 5th coefficients. One bit of the 5th coefficient remains in the shift register. Then, we load the next 5 coefficients of the pk to the shift register. Now, the shift register holds 66 bits of input. We use 64 bits from the LSB (including the remaining 1 bit of the 1st 5 coefficients) as the second input block. This process continues until the whole pk is absorbed. As the longest common factor between 13 and 64 is 1, the minimum size of the shift register needs to be $64+12=76$ to accommodate extra bits of the input stream for all possible cases.

The same buffer temporarily stores the output blocks when the ASCON block works in PRF mode, producing the public matrix $\hat{\boldsymbol{A}}$. ASCON squeeze generates 64-bit output after a 12 round permutation p^{12}, and each coefficient size of $\boldsymbol{A}$ is 13 bits. Then, these coefficients are fed to the rejection sampler. It accepts if the coefficients are less than q. So, only four coefficients can be constructed after a single squeeze. There will be 12 bits remaining in the shift register. After the next squeeze, another 64 bits output is added to the shift register. To accommodate all the bits, the same 76-bit shift register is used. From these 76 bits, 65 bits from the LSB are utilized to construct the next 5 coefficients of the matrix $\hat{\boldsymbol{A}}$, and 11 bits will remain in the shift register. This process will continue until the whole matrix is generated.

The ASCON block is required to sample the secret $\boldsymbol{s}$ (or $\boldsymbol{s'}$) and error $\boldsymbol{e}$ (or $\boldsymbol{e'}$,e''). The pseudorandom bits created by ASCOn permutation are fed to the CBD sampler to construct the final secret and error coefficients. Here, the 16 bytes seed (+1 byte of nonce) works as the input of the absorb step. Three absorb steps are required as the input block size is 64 bits. To construct a coefficient of the secret or error, 4 bits of XOF output are needed. Therefore, 4-times squeeze is required to generate a single polynomial.

4.4 Memory organization

One of the key design aspects of our design is to reduce memory as much as possible to make it resource-constraint device-friendly. We took two key approaches to this. First, we reduce the total memory by carefully choosing lattice $\hat{\boldsymbol{A}}$ and secret $\boldsymbol{s}$ generation. Second, NTT memory organization is done carefully to accommodate minimum BRAM usage for NTT. We use just two 18K BRAMs for NTT/INTT operations and one 18K BRAM for run time lattice generation and public key storage. While we are generating $\hat{\boldsymbol{A}}$, the careful design choice of 64-point polynomials gives us the perfect opportunity to synchronize ASCON-XOF-Based $\hat{\boldsymbol{A}}$ generation and $\texttt{NTT}(\boldsymbol{s})$ operation. For example, generation of $\boldsymbol{A}$(13*64 bit) consumes 192(= 3*12 for absorb + 13*12 for squeeze) cycles. However, we often need to squeeze more to accommodate more coefficients, as some are rejected. This takes 12-24 cycles more on average. Our NTT is a single butterfly design; hence, $\texttt{NTT}(\boldsymbol{s})$ takes 32*6 = 192 cycles. This careful design ensures that both hardware pieces work synchronously. This gives us the perfect opportunity for a runtime secret generation, which may not be preferred for NIST standard Kyber as NTT takes significantly more cycles for Kyber. As we need to generate lattice $\boldsymbol{A}$, $\texttt{NTT}(\boldsymbol{s})$ can be done within that time. This ensures that we can store the secret seeds and generate them every time, reducing the memory requirement for the secret by a significant amount. For example, we need to keep storage of 1 polynomial generation related to the secret generation contrary to 1 vector of a polynomial of the most optimized ML-KEM design [NIS23b]. If we choose 128-/256-point NTT, $\texttt{NTT}(\boldsymbol{s})$ takes more cycles, causing run time secret generation to be infeasible. We also use trivial ML-KEM optimization [NIS23b], such as run-time $\boldsymbol{A}$ generation. We have a separate memory for the public key. However, that is not necessary if it is integrated with IoT devices. IoT devices often have extra memory, which can be utilized to communicate with another party.

NTT memory is implemented with 2 separate memory as shown in Fig. 10. Once the secret is generated, 1st half is written in one BRAM, say M0, whereas 2nd half is written in M1 as coeff[0], and coeff[32] is required in the first stage. At every level, writing is swapped

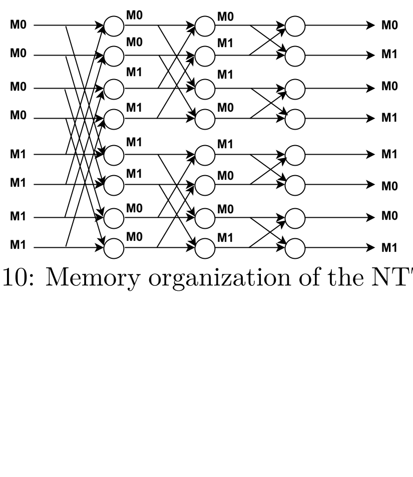

Figure 10: Memory organization of the NTT module

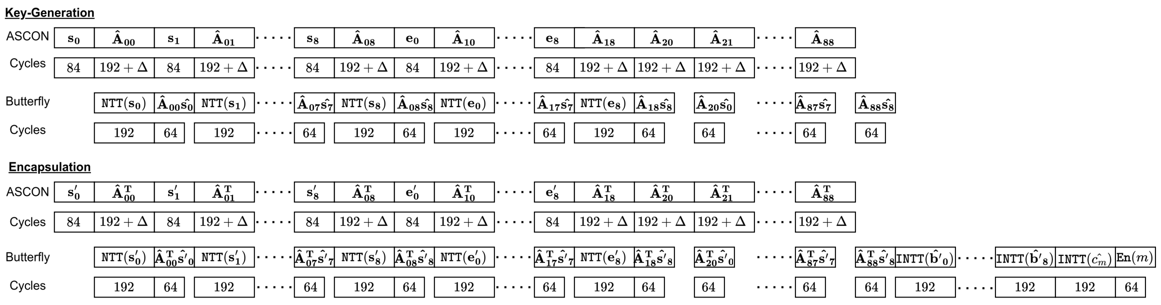

Figure 11: Scheduling with ASCON and butterfly module

to ensure the next stage data is available from 2 different memory. This strategy ensures streamlined dataflow even with a single port write-enabled memory. We are using 3, 18K BRAMs for this implementation. However, total memory is not used. For example, M0 & M1 need just 32*13-bit memory each (416 bit, 2.3% of 18K memory).

4.5 Scheduling

Scheduling is an important aspect of KEM hardware design. There are 2 parallel components of the KEM data path: (i) butterfly unit, which computes NTT, INTT, and PWM as well as encode, compress, and decompress, and (ii) Hash/PRF functions (ASCON-XOF for our case, SHA3 and SHAKE for Kyber). These 2 components are independent. This allows us to schedule synchronously, as shown in Fig. 11. Note that both components are specially required for the keygen and encrypt phases. Decrypt only needs butterfly, whereas FO-related functions require ASCON-XOF only. We sample using ASCON-XOF and CBD, which needs 84 cycles; then, one polynomial is sampled using rejection sampling followed by ASCON XOF. It is important to note that rejection sampling, in this case, takes at least 192 cycles, which is enough to calculate 64-point NTT. Then, while sampling the next secret, we can multiply and accumulate it as that takes fewer cycles. A lightweight ASCON core takes multiple cycles to create a lattice, even if the secret is stored. We have the option to store the entire secret in memory. However, runtime generation of the secret polynomial costs only 160 cycles of latency in the key generation/encryption function, which is negligible. As we are targeting lightweight design for energy and area-constrained IoT devices, we have taken this approach to reduce memory further.

4.6 Other computational units

We require two more components: rejection sampling and CBD sampler. Rejection sampling is a part of matrix $\hat{\boldsymbol{A}}$ or $\hat{\boldsymbol{A}}^{\boldsymbol{T}}$ generation. It checks and accepts if the ASCON-XOF generated 13 bits output is less than q. Otherwise, it rejects those 13 bits and proceeds with the next 13 bits. The implementation of this component is not constant time. However, it does not affect the security as the matrix $\hat{\boldsymbol{A}}$ is a public matrix. The CBD sampler is used to sample coefficients of the secret polynomials $\boldsymbol{s}$, $\boldsymbol{s'}$, and the error polynomials $\boldsymbol{e}$, $\boldsymbol{e'}$, e''. Each coefficient of these polynomials is constructed from 4 bits output of the ASCON-XOF. These 4 bits output can be denoted as a[0:3]. The coefficient is implemented by the following operation $b = \mathtt{HW}(a[0:1]) - \mathtt{HW}(a[2:3])$. Then the coefficient value $b \in [-2, 2]$. In other words, secret/error is sampled by calculating the Hamming Distance of two 2-bit numbers.

5 Results

In this section, we will discuss the implementation results and compare them with the state-of-the-art KEM designs.

Table 2: Resource requirements of various modular reductions

Modular reduction	Area	
	DSP	LUT
Shift-and-add	**0**	**102**
Montgomery+Barrett* [ABD+21]	2	13
k^2-reduction* [NKLO23]	0	132
k^2-reduction* [BAK21]	0	80

* one extra polynomial multiplication is required

Table 3: Submodules area requirements in client/server format

Submodules	Area			
	LUT	FF	BRAM	DSP
Butterfly	514/514	325/325	0/0	1/1
| Reduction	102/102	27/27	0/0	0/0
ASCON-XOF	688/688	326/326	0/0	0/0
| Permutation p^{12}	684/684	321/321	0/0	0/0
| Round counter	4/4	5/5	0/0	0/0
CBD(x2)	10/10	0/0	0/0	0/0
Rejection sampling	17/17	13/13	0/0	0/0
Top logic (ASCON buffer +FSM controller +verify)	1542/1640	922/841	1.5/1.5	0/0
Total	2771/2869	1586/1505	1.5/1.5	1/1

5.1 Resource consumption of submodules

Resource consumption for each component, followed by full hardware, is presented in Tab. 3. Our butterfly design requires multiplication, which is realized by a DSP unit. The reconfigurable butterfly consumes only 514 LUT and 325 Flip-flops in addition to 1 DSP unit. Modular reduction is one of the key components of the butterfly unit. We explored multiple strategies for $q = 7681$ and concluded that the shift-and-add modular reduction is the least hardware-intensive, as shown in Tab. 2. However, it is important to note that while using Montgomery and Barrett reduction following [ABD+21], one extra multiplication followed by the reduction converts a polynomial to the Montgomery domain. Returning from the Montogomery domain often requires one extra multiplication, costing extra latency for an extra DSP unit. Similarly, using k^2-reduction [BAK21, NKLO23], one extra multiplication followed by the reduction is performed during PWM to discard the extra k^2 factor. But, for the Shift-and-add modular reduction in Algorithm. 1, the extra multiplication and reduction step is not required as no domain change is performed.
Another key component of the datapath is ASCON-XOF hardware. Permutation consumes maximum area with 684 LUT and 321 flip-flops. The top includes an FSM controller, a 76-bit buffer/shift register for ASCON-XOF and verify logic, and 3-block RAMs. Overall, the controller is the key contributor in terms of area. This exploration indicates that in case of low latency requirement, an HW-SW codesign approach can also be taken to minimize the area further. We have used three 18K BRAMs in total. 18K BRAMs are considered 0.5 BRAM in FPGA architecture. 2 BRAMs (M0, M1) are used for NTT/INTT operations, and another BRAM (M2) has been used for public key storage. However, we do not use the entire memory storage. For M0, M1, only 2.3% of the BRAM has been used, whereas M2 uses $\sim$40% of the BRAM.

Using ASCON-XOF instead of Keccak comes with a trade-off in clock cycles due to its lightweight state registers. However, the extremely lightweight datapath paves the path to higher-frequency operations. A 6-stage pipelined architecture is also used in the butterfly to keep the critical path low. The server (runs key-generation and decapsulation algorithm) and client (runs only encapsulation algorithm) can operate at 318MHz and 323MHz, respectively. However, key generation, encapsulation, and decapsulation require 73 μs, 87 μs, and 110 μs, respectively. Latency is often not the utmost priority in resource/energy-constraint IoT devices, though latency overhead is reasonable due to the careful design of the datapath.

5.2 Comparison with the state-of-the-art

Comparison with the state-of-the-art hardware implementations of the notable candidates, including Kyber, is demonstrated in Tab. 4. Recently, a few designs based on Ring-LWE have been explored that are only CPA-secure. Notably, Ring-LWE often raises questions in terms

Table 4: Comparison of implementation of Rudraksh (KEM-poly64) with the state-of-the-art schemes. Freq. represents frequency, Exec time represents execution time. KG, Enc, and Dec represent key-generation, encapsulation, and decapsulation, respectively.

Scheme		Area					Freq.	kCycle counts	Exec time(μs)
		LUT	FF	Slice	BRAM	DSP	(MHz)	KG/Enc/Dec	KG/Enc/Dec
This work	client	**2771**	**1586**	**0**	**1.5**	**1**	**322**	**23/28/35**	**73.2/87.15/110**
	server	**2869**	**1505**				**318**		
Kyber[HLLM24]	client	4777	2661	1395	2.5	0	244	-/-/-	278/416/552
	server	4993	2765	1452					
Kyber[ZLZ+22]		8966	9173	3186	10.5	6	204	5/5/5	11.5/17.3/23.5
Kyber[XL21]	client	6785	3981	1899	3	2	161	4/5/7	23.4/30.5/41.3
	server	7412	4644	2126					
Kyber[DFA+20]		13745	11107	-	14	8	245	2/3/4	8.8/12.2/17.9
Kyber[BUC19]		14975	2539	4173	14	11	25	745/132/142	2980/5268/5692
Frodo[HOKG18]	client	6745	3528	1855	1	11	167	3276/3317/3358	19621/19866/20732
	server	7220	3549	1992		16	162		
Saber[RB20]		23686	9805	0	2	0	150	3/4/5	18.41/26.89/33.58
NewHope[ZQY+20]		6780	4026	-	7	2	200	4/7/3	21/33/12.5
InvRBLWE* [EBSMB19]	client	5000	5000	1292	0	0	443	.5/1/.5	1.13/2.32/1.13
	server						455		
RLWE*[PG13]		5595	4760	1887	7	1	251	15/14/9	57.90/54.86/35.39

* This schemes only provides CPA security. All other schemes are CCA secure

of security due to its structural lattices. Our KEM, for the first time, shows a lightweight CCA-secure design. Our design consumes only 2771/2869 LUT and 1586/1505 Flip-flops with a single DSP. It takes 2.5x less LUT/flip-flops and 2x less DSP and BRAM with respect to the most compact version of Kyber [XL21]. A recent implementation of Kyber [HLLM24] shows an improvement in Kyber hardware at the cost of latency. Our design is $\sim$5x times faster compared to [HLLM24] and consumes $\sim 50\%$ of area. A RISC-V-based softcore is used for Kyber implementation in [BUC19]. It offers flexibility and re-usability but has a significantly high area and latency overhead. Our design takes $40-60$x less execution time with respect to [BUC19] while consuming almost 5x less hardware and 10x less BRAM. Ring-LWE-based CPA-secure lightweight schemes have been proposed in this context before in [PG13, EBSMB19]. However, our design still consumes at least approximately 2x less area while providing the CCA-security. In brief, this KEM, for the first time, shows a lightweight CCA-secure design that can be adopted easily in resource-constraint edge devices.

6 Conclusion and future work

In this work, we proposed a lightweight design of a PQ-secure KEM with practical security and efficiency. We have designed the scheme from scratch and provided an optimized implementation. Our design strategy involves optimizing the scheme's parameters and other design elements with continuous feedback from the implementation. The final design results from multiple iterations and refinement of this process. The use of ASCON as a lightweight XOF to replace is also the first of its kind. Our immediate next plan is to design an ASIC of our PQ KEM and compare the results. Also, in the future, we would like to use the same strategies to design a lightweight digital signature scheme.

On another note, side-channel attack (SCA) protection is necessary for widely deployed algorithms. The implementation of Rudraksh is constant-time. Therefore, it is already timing side-channel attack (SCA) secure. One widely used provably secure countermeasure against other SCA is masking. We need some additional components for a masked version of Rudraksh, namely masked ASCON, masked CBD, arithmetic-to-Boolean (A2B), and Boolean-to-arithmetic (B2A) conversion algorithms [HKL+22]. ASCON is more side-channel resilient than other lightweight schemes, and the area overhead of SCA-protected ASCON with masking will be comparatively less than Keccak [DEMS12]. It will benefit our scheme, Rudraksh, by reducing the area cost of side-channel protection with masking. The cost of the area consumption of masked CBD, A2B, and B2A will be approximately the same for Rudraksh and Kyber. Therefore, the overall area consumption of masked Rudraksh should be lower than Kyber. However, it needs formal and experimental verification, which we have left as future work.

Acknowledgements

This work was partially supported by Horizon 2020 ERC Advanced Grant (101020005 Belfort), CyberSecurity Research Flanders with reference number VR20192203, BE QCI: Belgian-QCI (3E230370) (see beqci.eu), and Intel Corporation. Archisman Ghosh is partially supported by Intel Corporation and NSF (CNS 17-19235).

References

[AABCG20] Erdem Alkim, Yusuf Alper Bilgin, Murat Cenk, and François Gérard. Cortex-M4 optimizations for R,M lwe schemes. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):336–357, Jun. 2020.

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process. Online. Accessed 26th June, 2023, 2022.

[ABB+18] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Florian Mendel Martin M. Lauridsen, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. SPHINCS+ Submission to the NIST post-quantum project, v.3.1, 2018. [Online; accessed 10-January-2024].

[ABD+20] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Kyber. `https://github.com/pq-crystals/kyber/tree/main`, 2020.

[ABD+21] Roberto Avanzi, Joppe Bos, Leo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 3.02). Online, 2021.

[ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum Key Exchange - A New Hope. In Thorsten Holz and Stefan Savage, editors, *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*, pages 327–343. USENIX Association, 2016.

[AHMN13] Jean-Philippe Aumasson, Luca Henzen, Willi Meier, and María Naya-Plasencia. Quark: A lightweight hash. *Journal of Cryptology*, 26(2):313–339, 2013.

[BAK21] Mojtaba Bisheh-Niasar, Reza Azarderakhsh, and Mehran Mozaffari Kermani. High-speed ntt-based polynomial multiplication accelerator for post-quantum cryptography. In *28th IEEE Symposium on Computer Arithmetic, ARITH 2021, Lyngby, Denmark, June 14-16, 2021*, pages 94–101. IEEE, 2021.

[BBF$^+$19] Hayo Baan, Sauvik Bhattacharya, Scott R. Fluhrer, Óscar García-Morchón, Thijs Laarhoven, Ronald Rietman, Markku-Juhani O. Saarinen, Ludo Tolhuizen, and Zhenfei Zhang. Round5: Compact and fast post-quantum public-key encryption. In Jintai Ding and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019, Chongqing, China, May 8-10, 2019 Revised Selected Papers*, volume 11505 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2019.

[BCD$^+$16] Joppe W. Bos, Craig Costello, Léo Ducas, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Ananth Raghunathan, and Douglas Stebila. Frodo: Take off the ring! practical, quantum-secure key exchange from LWE. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1006–1018. ACM, 2016.

[BDK$^+$17] Joppe Bos, Leo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM. Cryptology ePrint Archive, Report 2017/634, 2017. https://ia.cr/2017/634.

[BDPA11] Guido Bertoni, Joan Daemen, Micha el Peeters, and Gilles Vam Assche. Cryptographic sponge functions. Online. Accessed 08-01-2024, 2011.

[BDPVA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, pages 313–314, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[Ber06] Daniel J Bernstein. Curve25519: New diffie-hellman speed records. https://cr.yp.to/ecdh/curve25519-20060209.pdf, 2006. Accessed: 2024-01-08.

[BGG$^+$16] Johannes Buchmann, Florian Göpfert, Tim Güneysu, Tobias Oder, and Thomas Pöppelmann. High-performance and lightweight lattice-based public-key encryption. In Richard Chow and Gökay Saldamli, editors, *Proceedings of the 2nd ACM International Workshop on IoT Privacy, Trust, and Security, IoTPTS@AsiaCCS, Xi'an, China, May 30, 2016*, pages 2–9. ACM, 2016.

[BGK$^+$06] Lejla Batina, Jorge Guajardo, Tim Kerins, Nele Mentens, Pim Tuyls, and Ingrid Verbauwhede. An elliptic curve processor suitable for rfid-tags. *IACR Cryptol. ePrint Arch.*, page 227, 2006.

[BKS19] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. Memory-efficient high-speed implementation of kyber on cortex-m4. In Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology – AFRICACRYPT 2019*, pages 209–228, Cham, 2019. Springer International Publishing.

[BMD$^+$21] Andrea Basso, Jose Maria Bermudo Mera, Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Michiel Van Beirendonck, and Frederik Vercauteren. Saber: Mod-lwr based kem (round 3 submission). Online. Accessed 08-01-2024, 2021.

[BMR$^+$13] Andrey Bogdanov, Florian Mendel, Francesco Regazzoni, Vincent Rijmen, and Elmar Tischhauser. ALE: aes-based lightweight authenticated encryption. In *FSE*, volume 8424 of *Lecture Notes in Computer Science*, pages 447–466. Springer, 2013.

[BMS+06] Lejla Batina, Nele Mentens, Kazuo Sakiyama, Bart Preneel, and Ingrid Verbauwhede. Low-cost elliptic curve cryptography for wireless sensor networks. pages 6–17, 09 2006.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012.

[BSNK19] Kanad Basu, Deepraj Soni, Mohammed Nabeel, and Ramesh Karri. NIST Post-Quantum Cryptography- A Hardware Evaluation Study. *IACR Cryptol. ePrint Arch.*, page 47, 2019.

[BUC19] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols (Extended Version). *IACR Cryptol. ePrint Arch.*, page 1140, 2019.

[cae19] The Competition for Authenticated Encryption: Security, Applicability, and Robustness. https://competitions.cr.yp.to/caesar-submissions.html, 2019.

[Can05] D. Canright. A Very Compact S-Box for AES. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 441–455, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[CCHY23] Jung Hee Cheon, Hyeongmin Choe, Dongyeon Hong, and MinJune Yi. SMAUG: Pushing Lattice-based Key Encapsulation Mechanisms to the Limits. *IACR Cryptol. ePrint Arch.*, page 739, 2023.

[CKLS18] Jung Hee Cheon, Duhyeong Kim, Joohee Lee, and Yongsoo Song. Lizard: Cut off the tail! A practical post-quantum public-key encryption from LWE and LWR. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 160–177. Springer, 2018.

[CML] Lily Chen, Dustin Moody, and Yi-Kai Liu. Post-quantum cryptography: Digital signature schemes. round 1 additional signatures.

[CN11] Yuanmi Chen and Phong Q. Nguyen. Bkz 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Coo66] S. A. Cook. *On the Minimum Computation Time of Functions.* PhD thesis, Harvard University, 1966. pp. 51-77.

[DEMS12] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. ASCON: Lightweight Authenticated Encryption & Hashing. https://ascon.iaik.tugraz.at/files/asconv12-nist.pdf, 2012. Accessed: 2024-01-08.

[DFA+20] Viet Ba Dang, Farnoud Farahmand, Michal Andrzejczak, Kamyar Mohajerani, Duc Tri Nguyen, and Kris Gaj. Implementation and Benchmarking of Round 2 Candidates in the NIST Post-Quantum Cryptography Standardization Process Using Hardware and Software/Hardware Co-design Approaches. *IACR Cryptol. ePrint Arch.*, page 795, 2020.

[DGJ$^+$19] Jan-Pieter D'Anvers, Qian Guo, Thomas Johansson, Alexander Nilsson, Frederik Vercauteren, and Ingrid Verbauwhede. Decryption failure attacks on ind-cca secure lattice-based schemes. In *Public-Key Cryptography – PKC 2019*, volume 11443 of *Lecture Notes in Computer Science*, pages 565–598. Springer, 2019.

[DKL$^+$18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, Feb. 2018.

[DKRV18] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. Cryptology ePrint Archive, Paper 2018/230, 2018. https://eprint.iacr.org/2018/230.

[DSDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with side information: Attacks and concrete security estimation. Cryptology ePrint Archive, Report 2020/292, 2020. https://eprint.iacr.org/2020/292.

[EBS20] Shahriar Ebrahimi and Siavash Bayat-Sarmadi. Lightweight and fault-resilient implementations of binary ring-lwe for iot devices. *IEEE Internet of Things Journal*, 7(8):6970–6978, 2020.

[EBSMB19] Shahriar Ebrahimi, Siavash Bayat-Sarmadi, and Hatameh Mosanaei-Boorani. Post-quantum cryptoprocessors optimized for edge and resource-constrained devices in iot. *IEEE Internet of Things Journal*, 6(3):5500–5507, 2019.

[FHK$^+$18] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over ntru, 2018. [Online; accessed 10-June-2023].

[GKS20] Denisa O. C. Greconici, Matthias J. Kannwischer, and Amber Sprenkels. Compact dilithium implementations on cortex-m3 and cortex-m4. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):1–24, Dec. 2020.

[GMK$^+$22] Archisman Ghosh, Jose Maria Bermudo Mera, Angshuman Karmakar, Debayan Das, Santosh Ghosh, Ingrid Verbauwhede, and Shreyas Sen. A 334uw 0.158mm2 saber learning with rounding based post-quantum crypto accelerator. In *IEEE Custom Integrated Circuits Conference, CICC 2022, Newport Beach, CA, USA, April 24-27, 2022*, pages 1–2. IEEE, 2022.

[GMK$^+$23] Archisman Ghosh, Jose Maria Bermudo Mera, Angshuman Karmakar, Debayan Das, Santosh Ghosh, Ingrid Verbauwhede, and Shreyas Sen. A 334 μw 0.158 mm^2 ASIC for post-quantum key-encapsulation mechanism saber with low-latency striding toom-cook multiplication. *IEEE J. Solid State Circuits*, 58(8):2383–2398, 2023.

[HB10] Mohamed N. Hassan and Mohammed Benaissa. A scalable hardware/software co-design for elliptic curve cryptography on picoblaze microcontroller. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 2111–2114, 2010.

[HGX21] Pengzhou He, Ujjwal Guin, and Jiafeng Xie. Novel low-complexity polynomial multiplication over hybrid fields for efficient implementation of binary ring-lwe post-quantum cryptography. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(2):383–394, 2021.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 341–371. Springer, 2017.

[HHLW20] Yiming Huang, Miaoqing Huang, Zhongkui Lei, and Jiaxuan Wu. A pure hardware implementation of CRYSTALS-KYBER PQC algorithm through resource reuse. *IEICE Electron. Express*, 17(17):20200234, 2020.

[HKL$^+$22] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Daan Sprenkels. First-order masked kyber on ARM cortex-m4. *IACR Cryptol. ePrint Arch.*, page 58, 2022.

[HLLM24] Shiyang He, Hui Li, Fenghua Li, and Ruhui Ma. A lightweight hardware implementation of crystals-kyber. *Journal of Information and Intelligence*, 2(2):167–176, 2024.

[HOKG18] James Howe, Tobias Oder, Markus Krausz, and Tim Güneysu. Standard lattice-based key encapsulation on embedded devices. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):372–393, 2018.

[HPVP11] Jens Hermans, Andreas Pashalidis, Frederik Vercauteren, and Bart Preneel. A new rfid privacy model. In Vijay Atluri and Claudia Diaz, editors, *Computer Security – ESORICS 2011*, pages 568–587, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[HWF09] Daniel Hein, Johannes Wolkerstorfer, and Norbert Felber. Ecc is ready for rfid – a proof in silicon. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography*, pages 401–413, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Int] International Data Corporation (IDC). Worldwide Semiannual Internet of Things (IoT) Sensor Forecast, 2021-2025.

[JZC$^+$17] Haodong Jiang, Zhenfeng Zhang, Long Chen, Hong Wang, and Zhi Ma. Post-quantum ind-cca-secure kem without additional hash. Cryptology ePrint Archive, Report 2017/1096, 2017. `https://eprint.iacr.org/2017/1096`.

[KMRV18] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. Saber on ARM CCA-secure module lattice-based key encapsulation on ARM. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):243–266, 2018.

[Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[Kpq] KpqC. Korean pqc competition. `https://www.kpqc.or.kr/competition.html`. [Online; accessed 10-January-2024].

[KY10] Elif Bilge Kavun and Tolga Yalcin. A Lightweight Implementation of Keccak Hash Function for Radio-Frequency Identification Applications. In *Proceedings of the 6th International Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec'10, page 258–269, Berlin, Heidelberg, 2010. Springer-Verlag.

[KYS+11] Jens-Peter Kaps, Panasayya Yalla, Kishore Kumar Surapathi, Bilal Habib, Susheel Vadlamudi, Smriti Gurung, and John Pham. Lightweight Implementations of SHA-3 Candidates on FPGAs. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology – INDOCRYPT 2011*, pages 270–289, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[LAM+22] Benjamin J. Lucas, Ali Alwan, Marion Murzello, Yazheng Tu, Pengzhou He, Andrew J. Schwartz, David Guevara, Ujjwal Guin, Kyle Juretus, and Jiafeng Xie. Lightweight hardware implementation of binary ring-lwe pqc accelerator. *IEEE Computer Architecture Letters*, 21(1):17–20, 2022.

[LN16] Patrick Longa and Michael Naehrig. Speeding up the number theoretic transform for faster ideal lattice-based cryptography. In Sara Foresti and Giuseppe Persiano, editors, *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings*, volume 10052 of *Lecture Notes in Computer Science*, pages 124–139, 2016.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[LSG22] Georg Land, Pascal Sasdrich, and Tim Güneysu. A hard crystal - implementing dilithium on reconfigurable hardware. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications*, pages 210–230, Cham, 2022. Springer International Publishing.

[Mil86] Victor S. Miller. Use of elliptic curves in cryptography. In Hugh C. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, pages 417–426, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[MKKV21] Jose Maria Bermudo Mera, Angshuman Karmakar, Suparna Kundu, and Ingrid Verbauwhede. Scabbard: a suite of efficient learning with rounding key-encapsulation mechanisms. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):474–509, 2021.

[MTK+20] Jose Maria Bermudo Mera, Furkan Turan, Angshuman Karmakar, Sujoy Sinha Roy, and Ingrid Verbauwhede. Compact domain-specific co-processor for accelerating module lattice-based KEM. In *57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, July 20-24, 2020*, pages 1–6. IEEE, 2020.

[NDG19] Duc Tri Nguyen, Viet Ba Dang, and Kris Gaj. A high-level synthesis approach to the software/hardware codesign of ntt-based post-quantum cryptography algorithms. In *International Conference on Field-Programmable Technology, FPT 2019, Tianjin, China, December 9-13, 2019*, pages 371–374. IEEE, 2019.

[NIS23a] Lightweight Cryptography Project. https://csrc.nist.gov/projects/lightweight-cryptography, 2023.

[NIS23b] NIST. Module-Lattice-based Key-Encapsulation Mechanism Standard. Online, 2023.

[NKLO23] Ziying Ni, Ayesha Khalid, Weiqiang Liu, and Máire O'Neill. Towards a lightweight crystals-kyber in fpgas: an ultra-lightweight bram-free NTT core. In *IEEE International Symposium on Circuits and Systems, ISCAS 2023, Monterey, CA, USA, May 21-25, 2023*, pages 1–5. IEEE, 2023.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009.

[PG13] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 2013.

[PJP$^+$22] Seunghwan Park, Chi-Gon Jung, Aesun Park, Joongeun Choi, and Honggoo Kang. Tiger: Tiny bandwidth key encapsulation mechanism for easy migration based on rlwe(r). Cryptology ePrint Archive, Paper 2022/1651, 2022. https://eprint.iacr.org/2022/1651.

[RB20] Sujoy Sinha Roy and Andrea Basso. High-speed Instruction-set Coprocessor for Lattice-based Key Encapsulation Mechanism: Saber in Hardware. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):443–466, 2020.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):34:1–34:40, 2009.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[SKD20] Dimitrios Sikeridis, Panos Kampanakis, and Michael Devetsikiotis. Assessing the overhead of post-quantum cryptography in tls 1.3 and ssh. In *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '20, page 149–156, New York, NY, USA, 2020. Association for Computing Machinery.

[SS04] Robert Schiefer and Rene Steinwandt. Designing efficient ecc hardware–the p and r coordinate approach. In Pil Joong Lee and Sang Hyuk Kim, editors, *ASIACRYPT 2004*, pages 263–280. Springer Berlin Heidelberg, 2004.

[SSW20] Peter Schwabe, Douglas Stebila, and Thom Wiggers. Post-quantum tls without handshake signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1461–1480, New York, NY, USA, 2020. Association for Computing Machinery.

[SSW21] Peter Schwabe, Douglas Stebila, and Thom Wiggers. More efficient post-quantum kemtls with pre-distributed public keys. In Elisa Bertino, Haya Shulman, and Michael Waidner, editors, *Computer Security – ESORICS 2021*, pages 3–22, Cham, 2021. Springer International Publishing.

[Tao] Jeff Tao. The key to intelligent connectivity in a world awash with iot data? making decisions at the edge. *Forbes*.

[Too63] A.L Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. In *Soviet Mathematics-Doklady*, volume 7, pages 714–716, 1963. http://toomandre.com/my-articles/engmat/MULT-E.PDF.

[XHW21] Jiafeng Xie, Pengzhou He, and Wujie Wen. Efficient implementation of finite field arithmetic for binary ring-lwe post-quantum cryptography through a novel lookup-table-like method. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 1279–1284, 2021.

[XL21] Yufei Xing and Shuguo Li. A compact hardware implementation of cca-secure key exchange mechanism CRYSTALS-KYBER on FPGA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):328–356, 2021.

[ZLZ$^+$22] Qingru Zeng, Quanxin Li, Baoze Zhao, Han Jiao, and Yihua Huang. Hardware design and implementation of post-quantum cryptography kyber. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, 2022.

[ZQY$^+$20] Neng Zhang, Qiao Qin, Hang Yuan, Chenggao Zhou, Shouyi Yin, Shaojun Wei, and Leibo Liu. NTTU: an area-efficient low-power ntt-uncoupled architecture for ntt-based multiplication. *IEEE Trans. Computers*, 69(4):520–533, 2020.

[ZYC$^+$20] Neng Zhang, Bohan Yang, Chen Chen, Shouyi Yin, Shaojun Wei, and Leibo Liu. Highly efficient architecture of newhope-nist on FPGA using low-complexity NTT/INTT. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):49–72, 2020.

[ZZW$^+$21] Cankun Zhao, Neng Zhang, Hanning Wang, Bohan Yang, Wenping Zhu, Zhengdong Li, Min Zhu, Shouyi Yin, Shaojun Wei, and Leibo Liu. A compact and high-performance hardware architecture for crystals-dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):270–295, Nov. 2021.