A $334\mu W$ $0.158mm^2$ ASIC for Post-Quantum Key-Encapsulation Mechanism Saber with Low-latency Striding Toom-Cook Multiplication Extended Version

Archisman Ghosh*, Jose Maria Bermudo Mera‡§, Angshuman Karmakar‡¶, Debayan Das*, Santosh Ghosh†, Ingrid Verbauwhede‡, and Shreyas Sen*
*School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, USA ‡COSIC, KU Leuven, Belgium †Intel Labs, Intel Corporation, Hillsboro, OR, USA §PQShield Ltd, UK ¶Indian Institute of Technology Kanpur, India

Abstract

The hard mathematical problems that assure the security of our current public-key cryptography (RSA, ECC) are broken if and when a quantum computer appears rendering them ineffective for use in the quantum era. Lattice based cryptography is a novel approach to public key cryptography, of which the mathematical investigation (so far) resists attacks from quantum computers. By choosing a module learning with errors (MLWE) algorithm as the next standard, National Institute of Standard & Technology (NIST) follows this approach. The multiplication of polynomials is the central bottleneck in the computation of lattice based cryptography. Because public key cryptography is mostly used to establish common secret keys, focus is on compact area, power and energy budget and to a lesser extent on throughput or latency. While most other work focuses on optimizing number theoretic transform (NTT) based multiplications, in this paper we highly optimize a Toom-Cook based multiplier. We demonstrate that a memory-efficient striding Toom-Cook with lazy interpolation, results in a highly compact, low power implementation, which on top enables a very regular memory access scheme. To demonstrate the efficiency, we integrate this multiplier into a Saber post-quantum accelerator, one of the four NIST finalists. Algorithmic innovation to reduce active memory, timely clock gating and shift-add multiplier has helped to achieve 38% less power than state-of-the art PQC core, 4 $\times$ less memory, 36.8% reduction in multiplier energy and 118$\times$ reduction in active power with respect to state-of-the-art Saber accelerator (not silicon verified). This accelerator consumes $0.158mm^2$ active area which is lowest reported till date despite process disadvantages of the state-of-the-art designs.

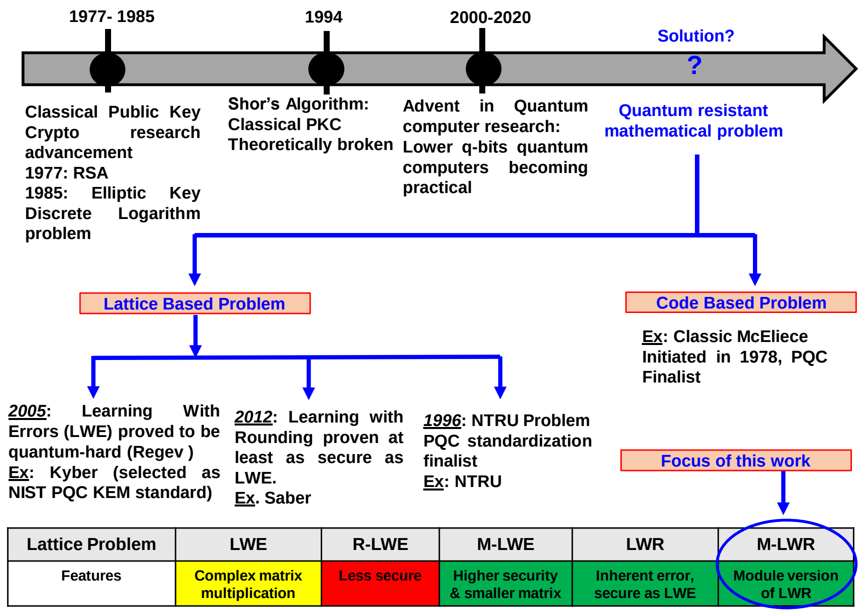
Index Terms

Post-quantum cryptography, first accelerator, striding Toom-Cook, lazy interpolation, memory-efficient, energy-efficient architecture, compact design

I. Introduction

In the past few decades, we have witnessed an unparalleled expansion of digital technologies in to all parts of our life. Considering the huge convenience and cost-effectiveness facilitated by these technologies in different sectors such as commerce, education, banking, communications, legality, etc, this penetration is expected to grow further in the future. However, if we look closely such proliferation of these technologies would never have been possible if there were no *assurances* such as the ability to verify the identity of the other communicating entity, mechanisms to prevent and detect tampering of data, the assertion of privacy between two or more communicating entities, methods to store one's secret and sensitive data, etc, while using these technologies. Cryptography is the discipline that studies different techniques to provide such assurance. Cryptography primarily emanated from the need for the secret exchange of information between allies and friendly entities in a hostile environment and until the end of the second world war, cryptography found little use outside military applications. Nevertheless, with the advancement in the field of computer science and the increase in digital technologies the field of cryptography flourished and ramified to provide solutions for many different problems.

Broadly, cryptography can be classified into two ways, symmetric-key cryptography and asymmetric-key or public-key cryptography (PKC). Symmetric-key cryptographic schemes are fast, low resource-hungry and can be used to encrypt data in bulk. However, they assume that a common *secret-key* has been shared among communicating parties beforehand. This requirement is often difficult to satisfy on real-world applications. On the other hand, PKC schemes, do not have such pre-requisite of shared secret-keys among communicating parties. Although they are typically slower, more resource-hungry, and can encrypt smaller data relative to the symmetric-key cryptographic schemes. Due to this complementary nature, symmetric-key and asymmetric-key cryptographic schemes are often used in tandem in many real-world applications. For example, in transport layer security (TLS) [1] protocol PKC schemes or specifically key-exchange schemes are used to establish the secret session key which is then used to encrypt ensuing communication using symmetric-key schemes. Another example is blockchain which is primarily constructed by combining digital-signatures [2] (asymmetric-key) and hash functions [3] (symmetric-key). The security assurances of both of these schemes arise from the computational intractability of some underlying hard problems i.e. it is infeasible to solve even one instance of such problems with practical parameters using the best-known algorithms to solve them which are running on a classical computer. However, in the context of quantum computers, this assumption does not always hold. For example, large integer-factorization and elliptic-curve discrete logarithm problems are two such hard problems that are used to construct two very well-known PKC schemes Rivest-Shamir-Adleman (RSA) [4] and elliptic-curve cryptography (ECC) [5], [6] respectively. However, there are quantum algorithms such as Shor's [7] and Proos-Zalka's [8] algorithm that can solve these two problems *easily* in polynomial time using a *large* quantum computer. Considering that our current public-key infrastructure is based mostly on RSA- and ECC-based schemes, we need to replace these schemes with quantum-resistant PKC to maintain the security of our digital world in the future. For symmetric-key, this problem is much less severe. The best-known quantum algorithm that can be used against symmetric-key cryptography is Grover's search [9] algorithm. This algorithm gives a quadratic speed-up in searching an unordered list. Hence, the adverse effect of Grover's search algorithm can be easily overcome by doubling the key lengths of symmetric-key cryptography.


1977- 1985
1994
2000-2020
Solution?
?
Classical Public Key Crypto research advancement
1977: RSA
1985: Elliptic Key Discrete Logarithm problem
Shor's Algorithm: Classical PKC Theoretically broken
Advent in Quantum computer research: Lower q-bits quantum computers becoming practical
Quantum resistant mathematical problem
Lattice Based Problem
Code Based Problem
Ex: Classic McEliece Initiated in 1978, PQC Finalist
2005: Learning With Errors (LWE) proved to be quantum-hard (Regev)
Ex: Kyber (selected as NIST PQC KEM standard)
2012: Learning with Rounding proven at least as secure as LWE.
Ex. Saber
1996: NTRU Problem PQC standardization finalist
Ex: NTRU
Focus of this work

Lattice Problem	LWE	R-LWE	M-LWE	LWR	M-LWR
Features	Complex matrix multiplication	Less secure	Higher security & smaller matrix	Inherent error, secure as LWE	Module version of LWR

Fig. 1: Timeline of PKC. Quantum computing research has motivated the use of quantum-hard mathematical problems.

Post-quantum cryptography (PQC) studies hard problems that remain hard to solve even in the presence of large quantum computers. The post-quantum standardization initiative by the National Institute of Standards and Technology (NIST) [10] in 2017 was a prudent step towards developing PQC. Initially, 59 and 23 schemes were submitted in key-encapsulation mechanism (KEM) and digital signature categories, respectively. 3 lattice-based (Saber, Kyber, and NTRU) and 1 code-based algorithm (Classic McEliece) were selected after three rounds of research and analysis by NIST for final consideration.

A historic timeline of PKC research along with the pros and cons of different lattice constructions is summarized in Figure 1. NIST has recently mentioned Kyber as the new standard. However, the NIST report on finalist candidates [11] mentioned that all the different experiments suggest Saber has strong security at par with Kyber.

A. *Lattice-based Cryptography*

The standard hard problem used to build lattice-based cryptographic (LBC) schemes is the Learning with errors (LWE) problem [12]. It states that it is hard to distinguish between n uniformly random samples $a \in \mathbb{Z}_q^n$ and n samples drawn as $(a_i, b_i = \langle a_i \cdot s \rangle + e_i)$ without having information about the secret s or the random error terms e_i. The LWE problem can be used to build public-key cryptography considering the matrix A and vector b (formed by the samples a_i and b_i, respectively) as the public key and s as the secret key. The main drawback of LWE schemes is the expensive matrix-vector multiplication $A \cdot s$ with a large range n.

The variant ring-Learning with errors (R-LWE) [13] was proposed to improve the efficiency of LWE schemes. In R-LWE, there is a single sample $(a, t = a * s + e) \in R_q \times R_q$ where each of the elements is now a polynomial belonging to the ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. R-LWE can be connected to LWE by looking at the new public matrix A as a matrix formed by rotations of the polynomial a. However, the central operation is now a polynomial convolution for which there exist algorithms with better time complexity.

While R-LWE schemes are more efficient, there is less trust in their security due to the extra algebraic structure conferred to the mathematical problem, which might be exploitable by an attacker. Module Learning with errors (M-LWE) [14] was proposed as an intermediate solution between LWE and R-LWE. In M-LWE the public matrix A has a much smaller dimension than in LWE, but each of the elements is a polynomial rather than an integer. The secret s and the error term e are also short vectors of polynomials. It offers higher security confidence than R-LWE while still benefiting from the better efficiency of polynomial arithmetic.

In this work, we work with Saber key-encapsulation mechanism [15] which is based on the hard problem known as Learning with rounding (LWR). LWR is a variant of the aforementioned LWE where the error term is introduced implicitly by *rounding* unlike the explicit addition of error terms in LWE. Therefore, for a given matrix $A \in \mathbb{Z}_q^{m\times n}$, random secret and errors $s, e \in \mathbb{Z}_q^n$, the LWE and LWR samples are given as $(A, b = A \cdot s + e)$ and $(A, b = \lfloor A \cdot s \rceil)$ respectively. Here, we want to point that $A \cdot s$ is a very good *entropy-diffuser*. In fact, $A \cdot s$ itself can be used as a pseudo-random function. However, since A is invertible with a very high probability, given the samples $b = A \cdot s$, the secret can be recovered trivially. Therefore, this cannot be used in cryptography. In Regev's [12] seminal LWE paper, it was shown that even after adding small error terms e the samples $b = A \cdot s + e$ remain computationally indistinguishable from uniformly generated random samples $\mathbf{u}$ *i.e.* $\mathbf{u} \overset{\text{comp}}{\approx} b$. This removes the possibility of any correlation attack on LWE. In fact, the decisional problem of LWE *i.e.* distinguishing samples b from u can be shown to be equivalent to the computational LWE problem mentioned above [12]. In his paper, Regev also showed that given samples b, recovering the secret s is at least as hard as solving GAP shortest vector problems (SVP) in random lattices in the worst case [16]. As no known quantum algorithms can solve GapSVP problem with an overwhelming advantage over their classical counterparts, this reduction from GapSVP to LWE engenders the quantum hardness of LWE-based schemes.

Banerjee et al. [17] first introduced the LWR problem which removes the necessity of adding the explicit error terms and introducing the error implicitly by *chopping* the lower order bits or rounding to a smaller field $\mathbb{Z}_p$. They showed that the LWR problem is at least as hard as LWE problem. One of their main results was to show that if q/p is sufficiently big then the $\lfloor A \cdot s \rceil \overset{stat}{\approx} \lfloor A \cdot s + e \rceil$. So combining this with the Regev's [12], [13] result mentioned earlier, one can show that $\lfloor A \cdot s \rceil \overset{\text{stat}}{\approx} \lfloor A \cdot s + e \rceil \overset{\text{comp}}{\approx} \lfloor \mathbf{u} \rceil$. Therefore, as before it is difficult to distinguish between LWR and uniform random distribution. This implies that the correlation between the LWR sample and the secret is not more than the correlation between an LWE sample and its respective secret. Later works by Bogdanov et al. [18], Alperin-Sheriff et al. [19], and Alwen et al. [20] further reduced the required value of q/p for these reductions to be held. Quite unfortunately, delving deeper into the security analysis of LWE or LWR is out of the scope of this work. We kindly refer interested readers to the original articles for more details. For a detailed discussion on deriving the security of Saber from LWR or specifically Module-LWR, we refer to the NIST specification document [15] of Saber submission.

B. Saber

Saber [15] is a lattice-based post-quantum KEM. It is one of the 4 finalist schemes of the NIST standardization procedure in the KEM category. Therefore, it has gone through extensive and rigorous scrutiny by the cryptographic community. This is a testimony of Saber for efficiency, theoretical security, and resilience to physical attacks.

Saber's security relies on the hardness of solving the module Learning with rounding (**M-LWR**) problem. **Key generation**, described in Algorithm 1, starts by generating a truly random 256-bit seed. This seed is expanded with a function based on the extendable output function SHAKE-128 to generate the public matrix A. A similar approach is followed to generate the secret s, but the coefficients of s follow a discrete binomial distribution β_μ with parameter μ rather than being uniformly distributed. The sample b is computed as the product $A^T \cdot s$ followed by the addition of a constant value h and a rounding operation. The public key is composed of the seed to generate A and the sample b. The secret key is s.

The **encryption**, described in Algorithm 2, starts by regenerating the public matrix A. Then, it proceeds in the same way as the key generation to generate a new secret s' and a new sample b'. Additionally, a sample v' is computed as the product of the other part of the public key with the new secret $b^T \cdot s'$. The message m is encoded in this vector c_m, which together with b' constitutes the ciphertext. The **decryption**, described in Algorithm 3, computes a new sample v in an analogous way to encryption by multiplying $b'^T \cdot s$. The message is recovered by decoding the difference between v and a scaled version of the other part of the ciphertext c_m.

Algorithm 1 `Saber.PKE.KeyGen()` [15]

```
1: $seed_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\boldsymbol{A} = \text{gen}(\text{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$
3: $r = \mathcal{U}(\{0,1\}^{256})$
4: $\boldsymbol{s} = \beta_\mu(R_q^{l\times 1}; r)$
5: $\boldsymbol{b} = ((\boldsymbol{A}^T\boldsymbol{s} + \boldsymbol{h}) \bmod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l\times 1}$
6: return $(pk := (seed_{\boldsymbol{A}}, \boldsymbol{b}), \boldsymbol{s})$
```

Algorithm 2 Saber.PKE.Enc$(pk = (\boldsymbol{b}, seed_{\boldsymbol{A}}), m \in R_2; r)$ [15]

```
1: $\boldsymbol{A} = \texttt{gen}(\text{seed}_{\boldsymbol{A}}) \in R_q^{l \times l}$
2: if r is not specified then
3:     $r = \mathcal{U}(\{0,1\}^{256})$
4: $\boldsymbol{s'} = \beta_\mu(R_q^{l \times 1}; r)$
5: $\boldsymbol{b}' = ((\boldsymbol{A}\boldsymbol{s}' + \boldsymbol{h}) \bmod q) \gg (\epsilon_q - \epsilon_p) \in R_p^{l \times 1}$
6: $v' = \boldsymbol{b}^T(\boldsymbol{s}' \bmod p) \in R_p$
7: $c_m = (v' + h_1 - 2^{\epsilon_p - 1} m \bmod p) \gg (\epsilon_p - \epsilon_T) \in R_T$
8: return $c := (c_m, \boldsymbol{b'})$
```

Algorithm 3 Saber.PKE.Dec$(\boldsymbol{s}, c = (c_m, \boldsymbol{b'}))$ [15]

```
1: $v = \boldsymbol{b}'^T(\boldsymbol{s} \bmod p) \in R_p$
2: $m' = ((v - 2^{\epsilon_p - \epsilon_T} c_m + h_2) \bmod p) \gg (\epsilon_p - 1) \in R_2$
3: return $m'$
```

Algorithm 4 Saber.KEM.KeyGen()

```
$(seed_{\boldsymbol{A}}, \boldsymbol{b}, \boldsymbol{s}) = \texttt{Saber.PKE.KeyGen}()$
$pk = (seed_{\boldsymbol{A}}, \boldsymbol{b})$
$pkh = \mathcal{F}(pk)$
$z = \mathcal{U}(\{0,1\}^{256})$
return $(pk := (seed_{\boldsymbol{A}}, \boldsymbol{b}), sk := (z, pkh, pk, \boldsymbol{s}))$
```

Algorithm 5 Saber.KEM.Encaps$(pk := (seed_{\boldsymbol{A}}, \boldsymbol{b}))$

```
1: $m \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $(r, \hat{K}) = \mathcal{G}(\mathcal{H}(pk), m)$
3: $c = \texttt{Saber.PKE.Enc}(pk, m, r)$
4: $K = \mathcal{H}(\mathcal{H}(c), \hat{K})$
5: return $(c, K)$
```

Algorithm 6 Saber.KEM.Decaps$(c, sk := (z, pkh, pk, \boldsymbol{s}))$

```
1: $m' = \texttt{Saber.PKE.Dec}(\boldsymbol{s}, c)$
2: $(r', \hat{K}') = \mathcal{G}(\mathcal{H}(pk), m')$
3: $c' = \texttt{Saber.PKE.Enc}(pk, m'; r')$
4: if $c = c'$ then
5:     return $K = \mathcal{H}(\mathcal{H}(c), \hat{K}')$
6: else
7:     return $K = \mathcal{H}(\mathcal{H}(c), z)$
```

The Chosen Ciphertext Attack-secure (CCA) version of Saber is achieved by applying the Fujisaki-Okamoto transform to the encryption scheme. Such construction requires two additional hash functions $\mathcal{G}$ and $\mathcal{H}$, which are SHA3-512 and SHA3-256, respectively. In its KEM setting, the **key-generation** algorithm utilizes the public-key key-generation algorithm as shown in Algorithm 1. Additionally, it adds the hash of public-key, the public-key, and 256-bit random number (z) for CCA security . The message is randomly generated during **encapsulation** as shown in Algorithm 5. The hash functions are used to generate randomness for the encryption as well as to generate the established session key. During **decapsulation**, the encryption is recalculated to check for potentially maliciously crafted ciphertexts. If the ciphertext match, the session key is computed in the same way using the hash functions as shown in Algorithm 6.

A sample KEM is shown in Fig. 2 between 2 parties, namely Alice & Bob. Alice creates the public key and secret key using algorithm 4. She sends public key to Bob. Upon receiving that, bob calculates the encapsulation and sends the ciphertext of a message m using algorithm 5. Alice decapsulates using her secret key and received ciphertext c using algorithm 6. It should be noted that our IC can work as both Alice & Bob here.

C. Features of Saber

However, compared to the other lattice-based finalist schemes of NIST standardization procedure *i.e.* Kyber [21] and NTRU [22], Saber is relatively less studied scheme. Further, the design of Saber is also quite unique and unconventional

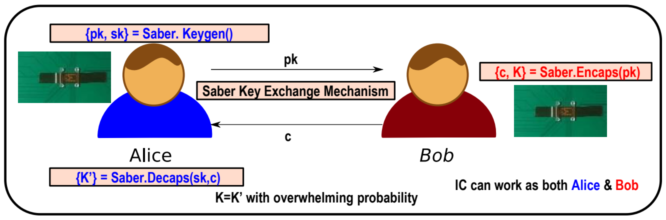

{pk, sk} = Saber. Keygen()
pk
Saber Key Exchange Mechanism
{c, K} = Saber.Encaps(pk)
c
Alice
Bob
{K'} = Saber.Decaps(sk,c)
K=K' with overwhelming probability
IC can work as both Alice & Bob

Fig. 2: A sample KEM mechanism between 2 communicating parties. Note that our IC can act as both parties. Keygen(), Encaps() and Decaps() are presented in Algorithm 4, 5 & 6 respectively.

compared to the other lattice-based schemes. Thus studying implementation aspects of Saber on dedicated hardware platforms ASIC is an appropriate and attractive field of investigation for the benefit of PQC. We describe some of the salient features of Saber below. Here are 3 salient features of Saber as PQC-KEM scheme:

- *Power-of-2 moduli:* Polynomial multiplication is one of the most computationally expensive components of LBC. The *de-facto* standard modulus for LBC is a prime that facilitates usage of fast quasi-linear number theoretic transform (NTT) [23] based polynomial multiplication. However, Saber uses an unorthodox choice of power-of-two modulus. It has been shown that Saber performance is at par with similar schemes which use NTT based polynomial multiplication on different platforms [24]–[26] by combining sub-quadratic polynomial multiplication algorithms Toom-Cook [27], [28] and Karatsuba [29].
- *Flexibility:* Due to the use of module-lattices, the security of Saber can be easily upgraded or downgraded by appropriately increasing or decreasing the number of polynomials in the public matrix. This requires little to no change in other parameters of the scheme. Therefore, ASICs and software libraries designed for one level of security can be easily adapted to other security levels with small changes which results in great flexibility.
- *Resistance to side-channel attacks:* Saber uses constant-time algorithms to prevent side-channel attacks like timing or simple-power analysis attacks. Generating noise inherently by rounding and using centered binomial distributions instead of more complex discrete Gaussian distributions [30], [31] reduces the side-channel attack surface of Saber greatly. Also, it has been recently shown that masking, which is a provably secure countermeasure against powerful side-channel attacks, can be integrated much more efficiently on Saber than other lattice-based KEMs such as Kyber [32], [33]. This efficiency comes from the usage of power-of-2-moduli rather than prime moduli. In this work, we have followed constant-time implementations and avoided conditional branching on secret values similar to previous works on Saber [15]. Therefore, our implementation is resistant to simple power analysis (SPA) or simple electromagnetic analysis (SEMA) attacks. Recently, physical countermeasures have been very popular in this context as they are architecture-agnostic [34]–[37]. Integrating these countermeasures (masking or physical countermeasures) along with attack detection mechanism (e.g. [38], [39]) with this Saber architecture will help in more sophisticated DPA/CPA/CEMA security. These countermeasures can be integrated in the future versions of the ASIC for SCA-resilient implementation of Saber.

Though NIST has recommended Kyber for standardization for a few reasons, some of which are not strictly technical per se. Their final report [11] (Sec. 4.3.4) mentions that the security, performance, bandwidth usage, etc. of Saber is at par with Kyber. Therefore Saber is perfectly suitable to be used as a PQC KEM in the future. Please note that another NIST finalist NTRU [22] is already present in many popular products like OpenSSH (version 9.0 onwards), GPL, WolfSSL, etc.

Furthermore, as Saber and Kyber are both based on module lattices and share a large number of individual blocks, we firmly believe that many of the *techniques* developed here for Saber can also be used for Kyber. For example, the polynomial sizes of Saber and Kyber are same and hence our strategies can be even used to realize a low-power and area instantiation of Kyber on ASIC. We should note that Kyber in its current form cannot be executed directly on our ASIC. To realize this, we need to make suitable changes such as the incorporation of a suitable modular reduction strategy, updating data-paths to suit the parameters of Kyber, deciding how the matrix A is generated, etc. These can be chosen according to the final goal of the ASIC design. However, this is an intriguing research question that needs more attention and due deliberation. Therefore, we leave this as future work.

D. PQC Hardware Implementations: State-of-the-Art

In this section, we summarize the state-of-the-art of hardware implementations of post-quantum lattice-based KEMs. A more detailed analysis of performance, area and power figures is deferred to Section IV. The most frequent hardware implementations that can be found in the literature are FPGA-based implementations. The reason is that lattice-based cryptography has undergone a quick and recent development and the shortest design cycle of FPGA implementations is more adequate to develop a proof of concepts or to demonstrate algorithmic optimizations for existing schemes. Particularly, all most relevant lattice-based KEMs have FPGA implementations as can be seen for Frodo [40], Kyber [41], NTRU [42], NTRUPrime [43] and Saber [44] [45].

Despite the variety of schemes, the differences between distinct hardware implementations lie in the optimizations performed on the multiplier to suit better the parameters of the corresponding scheme. Focusing on Saber hardware implementations, we can distinguish three different approaches toward multiplication. *First*, the Toom-Cook algorithm is used to accelerate the multiplication on hardware, together with an HW-SW co-design strategy to achieve a compact implementation on a heterogeneous ARM+FPGA System-on-Chip (SoC) [44]. *Second*, high performance can be achieved with a fully parallel multiplier where the full polynomials are loaded and shifted after every multiplication [45]. *Third*, a combination of the Karatsuba algorithm and parallel multipliers has been proposed to achieve the high-performance operation of Saber while reducing the area requirements with respect to the fully parallel approach [46]. Both the second and the third approach have been used to implement Saber on full custom hardware [47] [48]. Though [48] proposes another hardware for Saber, as mentioned in the paper [48], due to a logic bug, the address offsets of 3 of the 4 distributed memories are incorrectly decoded and data is overwritten. This logical error results in a few flipped bits on the output of the chip when compared with the expected results. Due to this issue, we refrain from comparing performance directly with this work [48] in the comparison table as the table includes only solid-state circuit literatures with functional ICs.

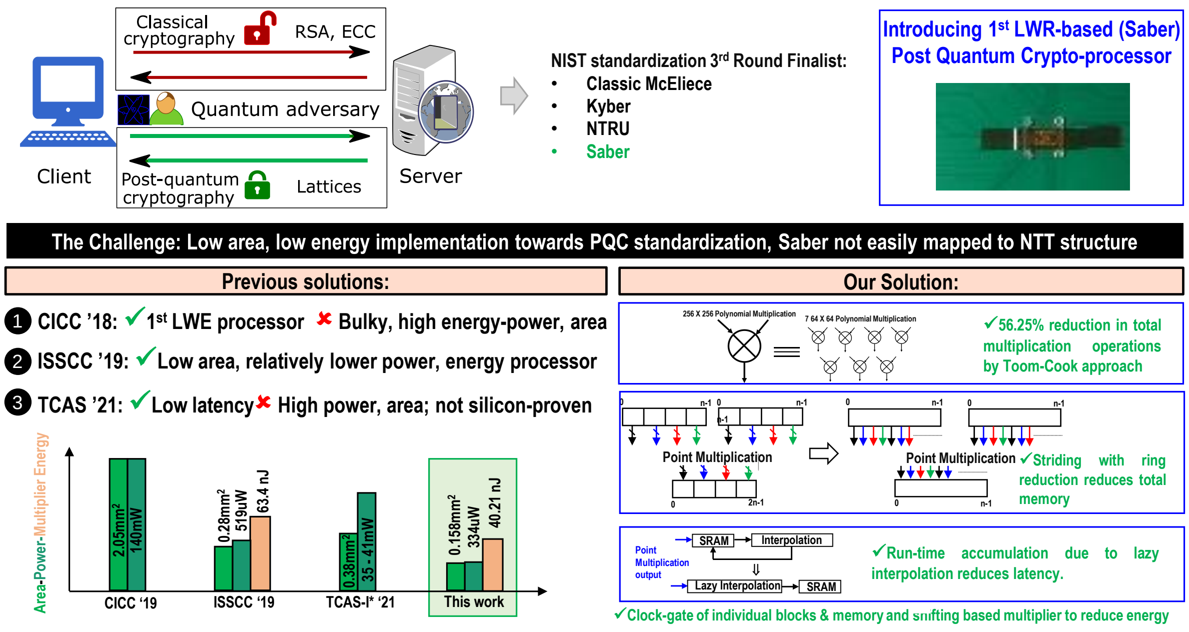

Classical cryptography
RSA, ECC
Quantum adversary
Post-quantum cryptography
Lattices
Client
Server
NIST standardization 3rd Round Finalist:
Classic McEliece
Kyber
NTRU
Saber
Introducing 1st LWR-based (Saber) Post Quantum Crypto-processor
The Challenge: Low area, low energy implementation towards PQC standardization, Saber not easily mapped to NTT structure
Previous solutions:
1 CICC '18: ✓ 1st LWE processor ✗ Bulky, high energy-power, area
2 ISSCC '19: ✓ Low area, relatively lower power, energy processor
3 TCAS '21: ✓ Low latency ✗ High power, area; not silicon-proven
Area-Power-Multiplier Energy
2.05mm²
140mW
0.28mm²
519uW
63.4 nJ
0.38mm²
35 - 41mW
0.158mm²
334uW
40.21 nJ
CICC '19
ISSCC '19
TCAS-I* '21
This work
Our Solution:
256 X 256 Polynomial Multiplication
7 64 X 64 Polynomial Multiplication
✓56.25% reduction in total multiplication operations by Toom-Cook approach
Point Multiplication
0
n-1
2n-1
✓Striding with ring reduction reduces total memory
Point Multiplication output
SRAM
Interpolation
Lazy Interpolation
SRAM
✓Run-time accumulation due to lazy interpolation reduces latency.
✓Clock-gate of individual blocks & memory and shifting based multiplier to reduce energy

Fig. 3: Classical PKC is broken by quantum computers. This work introduces the first silicon-verified M-LWR-based post-quantum crypto-accelerator. Fabricated IC uses a striding Toom-Cook-based polynomial multiplier with lazy interpolation to reduce latency and memory. Clock-gating of individual blocks and shift-based unit multiplier achieve the lowest power design.

E. Motivation

KEM schemes are used to establish a common secret key between two or more communicating parties. A very well-known usage of KEM is inside transport layer security (TLS) protocol which is a newer version of the secure socket layer (SSL) protocol [49]. In TLS the *handshake* protocol uses KEM to establish the common secret *session* key. A very well-known application that uses TLS or by extension KEM scheme is HTTPS or Hypertext Transfer Protocol (HTTP) over SSL. In HTTPS the client (browser) first authenticates the server (web server) using the help of certificate authority and digital signature algorithms. The client also obtains the SSL/TLS certificate in this process. This certificate among other things contains the KEM scheme name and the public key of the server. The client uses this KEM public key to initiate the handshake protocol which finishes with a common secret key on both sides of the client and server. This secret key is now used by a previously agreed symmetric-key algorithm like the advanced encryption standard (AES) to encrypt all the data communication between the client and the server.

This secret key derived using the handshake protocol is also called *master* secret. Whenever a client opens a connection with the server there might not be a full handshake involving the KEM protocol. Instead, the client and server can reuse the master secret key established during some previous connection. In this case, the secret key for the symmetric-key algorithm is derived using the master secret and a random value that the client and the server sent each other during their initial messages. This is called *abbreviated handshake*. Multiple connections which share the same master secret constitute a session. Depending on the security settings of the client and server this master secret is kept alive for a long duration. Therefore removing the necessity of invoking the KEM algorithm for a long time. Hence, the KEM schemes are short-lived and invoked only a few times during the lifetime of the application.

However, the accelerator is there for the lifetime of the system. Due to the short-lived nature of KEMs in typical applications, the latency of KEM operations becomes less relevant compared to other metrics such as small area of the processor, low power, and low energy. High performance is arguably more important for symmetric key primitives that are constantly used once the communication has been established. A smart design of a KEM accelerator seeks to reduce the cost in the main system by achieving low area as well as to reduce the operational costs by achieving low power and low energy.

In this paper, we also aim at filling this gap in the state-of-the-art by bringing the first approach in silicon. Moreover, fundamental constraints such as low area and low energy when implementing key exchange operations makes the *first approach*, i.e., Toom-Cook, more attractive for a dedicated IC. In Section IV we show different metrics to compare our design [50], [51] to other Saber ASIC designs as well as to ASIC designs [52] that accelerate other lattice-based schemes such as NewHope [52] and Kyber [53].

F. Contributions

Figure 3 shows the key contributions of this work which we categorically describe below.

- **First silicon-verified implementation of Saber:** We have presented the first Saber core and till now one of the few *fully functional* PQC ASIC. Therefore this work is a cornerstone in the development of PQC and in the transition from classical PKC to PQC.
- **Multiplier optimization with striding Toom-Cook and lazy interpolation to reduce energy and power of the accelerator:** Multiplication of Saber cannot be directly mapped with NTT structure, unlike Kyber. Different approaches have been taken to circumvent this problem. One approach is to use an alternative multiplication strategy. The second approach is to tweak the Saber algorithm so that it can be mapped to NTT structure [26]. First approach is taken here. Several works are exploring classical Toom-Cook-based architecture. In this work, we have further optimized Toom-Cook multiplication with striding and lazy interpolation to make it memory efficient. It should be noted that this architecture helps us to reduce total SRAM used in the IC. Significant amount of power/energy consumption comes from the memory block which is leakage dominated. Reducing total memory size significantly helps to reduce final energy consumption.
  It should be noted that previously lazy interpolation is explored only in software [24] in Toom-Cook multiplication context. [24] focuses on a software Cortex-M4 based latency vs memory optimization problem for Toom-Cook multiplication. Additionally, our work merges a striding memory-access approach with lazy interpolation to provide a unified low-power, low-energy, and reduced memory ASIC for Saber with comparable latency for the multipliers.
- **Re-configurable instruction-based multi-purpose architecture:** This architecture is re-configurable and can be controlled by micro-instruction provided from outside. Hence, the same architecture can be used to calculate encapsulation, decapsulation, and key generation removing the requirement for duplicate hardware.
- **Lowest area & power implementation among the PQC cores and accelerators:** This work provides 36% power improvement from silicon-verified PQC core and 118X improvement with respect to state-of-the art Saber accelerator (not silicon verified) [47]. Saber core takes 0.158 mm^2 of the area and 10.1875KB memory which is the lowest among PQC cores to date.

G. System Overview & Paper Organization

This IC has dedicated blocks for each micro-operations namely polynomial multiplication, binomial sampler, addpack, addround, cmov, verify etc. Polynomial multiplication is the most computationally complex, hence most area & power hungry block. This is chosen for algorithmic & architectural optimization in order to achieve low area and low energy. Algorithmic level optimizations of the multiplier are discussed in Section II. Section III discusses hardware optimizations along with a brief discussion about all the circuit components. Silicon results are presented in Section IV before we conclude the work in Section V.

II. Multiplier Optimizations

The core operation of Saber as a M-LWR scheme is the matrix-vector multiplication between the public matrix A and the secret vector s. Since the dimension of the module lattice is $l = 3$, the public matrix A is composed by 3×3 polynomials with 256 coefficients each. Therefore, the operation that needs to be optimized is actually the polynomial multiplication, which is defined in the algebraic ring $R_q = \mathbb{Z}_q[x]/(x^n + 1)$. Mathematically, a multiplication in R_q is a standard polynomial multiplication followed by the modular reduction by (x^n+1). The rule of thumb to compute the modular reduction is to equalize the modulus to zero $(x^n + 1)$ and, therefore, to apply the change of variable $x^n = -1$ to the product of two polynomials. In other words, the multiplication in R_q is equivalent to a negatively wrapped polynomial multiplication. Later n this section, we explain how to implement the multiplier to exploit this structure. Next, we discuss the algorithmic choices for polynomial multiplication.

There are two approaches to implement polynomial multiplication on hardware. First, one can use the straightforward algorithm with quadratic complexity referred to as **schoolbook** [45]. While this approach leads to poor performance figures on software, on hardware we can take advantage of its simpler structure to parallelize the arithmetic operations. The degree of parallelization can be increased to trade off area for higher performance and research shows that the highest performance for polynomial multiplication can be achieved with the fully parallel multiplier [42].

Second, one can optimize the polynomial multiplication algorithm to reduce the computational complexity. Then, the implementation of the chosen algorithm can still be optimized at platform level to take advantage of the available resources. This approach is the most popular for software implementations [44] [26] but it can also be followed with successful results on hardware [44].

The polynomial multiplication module is implemented following the second approach. Moreover, we use Toom-Cook 4-way to reduce the complexity of the polynomial multiplication and parallelize it as in [44]. Thus, for Saber parameters a single 256×256 polynomial multiplication is broken down into seven 64×64 polynomial multiplications, which are parallelized at hardware level. We propose additional optimizations to the multiplication, a **strided algorithm** and **lazy interpolation**, both of which are explained next in this section. There are three benefits of this approach:

- Using Toom-Cook multiplication reduces total number of multiplication from 256×256 into seven 64×64 polynomial multiplications hence saving 56.25% of total multiplication operation.
- Striding Toom-Cook reduces the memory requirement, hence helps in reducing energy consumption and total area.
- Lazy interpolation helps in reducing number of iteration and helps in latency improvement.

A. *Classical Toom-Cook vs Striding Toom-Cook*

Algorithm 7 Evaluation of classical and striding Toom-Cook 4-way with vertical scanning

```
Input: Two arrays $A$ and $B$ with the $n = 256$ coefficients of the polynomials
Output: Seven arrays $w_1$ to $w_7$ with 127 / 64 coefficients of the intermediate products each
1: for $j = 0$ to 63 do
2:     $r_0 = A_0[j]$;
3:     $r_1 = A_{64}[j]$;      $r_1 = A_1[j]$;
4:     $r_2 = A_{128}[j]$;     $r_2 = A_2[j]$;
5:     $r_3 = A_{192}[j]$;     $r_3 = A_3[j]$;
6:     $r_4 = r_0 + r_2$;     $r_5 = r_1 + r_3$;
7:     $r_6 = r_4 + r_5$;     $r_7 = r_4 - r_5$;
8:     $aws_3[j] = r_6$;     $aws_4[j] = r_7$;
9:     $r_4 = 2 * (r_0 * 4 + r_2)$;
10:    $r_5 = r_1 * 4 + r_3$;
11:    $r_6 = r_4 + r_5$;     $r_7 = r_4 - r_5$;
12:    $aws_5[j] = r_6$;     $aws_6[j] = r_7$;
13:    $r_4 = 8 * r_3 + 4 * r_2 + 2 * r_1 + r_0$;
14:    $aws_2[j] = r_4$;
15:    $aws_7[j] = r_0$;     $aws_1[j] = r_3$;
16: Repeat the above steps to generate the weighted polynomials $bws_1$ to $bws_7$
17: for $i = 1$ to 7 do
18:     $w_i = aws_i * bws_i$;
19: return $w_1$ to $w_7$
```

Toom-Cook k-way uses a divide-and-conquer approach to break down a single multiplication of polynomials with n coefficients each into $2k - 1$ multiplications where the new operands have n/k coefficients each instead. The procedure to generate the intermediate operands is called evaluation. The final result can be retrieved applying the inverse transformation to evaluation, namely interpolation. Mathematically, the inputs to the multiplication are lifted from $R[x]$ to the isomorphic ring $R[x][y]/(x^k - y)$. Then, the operands can be expressed as $a(x) = (a_0 + a_1 x + \cdots + a_{k-1}x^{k-1}) + \cdots + (a_{n-k} + \cdots + a_{n-1}x^{k-1})y^{r-1}$ where $r = n/k$. Particularizing $n = 256$ and $k = 4$, we can derive Algorithm 7 with violet text to perform Toom-Cook evaluation on points $x = \{0, 1, -1, 1/2, -1/2, 2, \infty\}$. This algorithm incorporates the vertical scanning method proposed in [25] to reduce the overhead introduced by memory operations. On hardware this method allows the parallelization of the evaluation to build all seven intermediate polynomials simultaneously.

However, there are two factors that penalize the performance and memory of Toom-Cook when implemented this way. First, the memory access pattern does not follow any spatial locality. The sequence of the coefficients indexes accessed in classical Toom-Cook has offsets of 64. When following a HW/SW co-design approach this can be solved by transferring the polynomials with the appropriate layout [44], but this is not efficient when the whole scheme is accelerated in hardware. Second, this Toom-Cook algorithm requires size doubling in the intermediate polynomials.

We address the two inefficiencies of Toom-Cook by using a less known variant of Toom-Cook [54] referred to as striding Toom-Cook. In this variant, a different ring isomorphism is used. The inputs are lifted from $R[x]$ to $R[y][x]/(x^k - y)$ instead of $R[x][y]/(x^k - y)$. Since the ring of the original multiplication is $R[x]/(x^n - y)$ and $y = x^k$, the base ring for the intermediate multiplications becomes $R[y]/(x^r + 1)$ which is also a negacyclic polynomial ring. Therefore, the modular reduction corresponding to the ring multiplication can be deferred to the point multiplication of Toom-Cook. Mathematically, the operands of this Toom-Cook variant can be written as $a(x) = (a_0 + a_k y + \cdots + a_{(r-1)k} y^{r-1}) + \cdots + (a_{k-1} + a_{2k-1} y + \cdots + a_{(r-1)k+k-1} y^{r-1}) x^{k-1}$ where $y = x^k$ and $n = k \cdot r$. Again, we take into account the parameters of our multiplication, $n = 256$, $k = 4$, and we evaluate the polynomials in the same points as for classical Toom-Cook, $x = \{0, 1, -1, 1/2, -1/2, 2, \infty\}$. Thus, Algorithm 7 with blue text can be derived as evaluation of striding Toom-Cook. If we compare both versions, i.e., Algorithm 7 with violet or blue text, the only difference between classical Toom-Cook and striding Toom-Cook evaluation lies in the load operations that happen at the beginning of every iteration. We can observe that the striding version reads coefficients with consecutive indexes which allows a more efficient hardware implementation.

Algorithm 8 Interpolation of classical and striding Toom-Cook 4-way

```
Input: Seven arrays $w_1$ to $w_7$ with 127 / 64 coefficients of the intermediate products each
Output: An array with the coefficients of $C = A(x) * B(x)$
 1: $C \leftarrow 0$
 2: for $i = 0$ to 126 / 63 do
 3:     $r_7 = r_0;\quad r_8 = r_1;\quad r_9 = r_2;$
 4:     $r_1 = w_2[i];\quad r_4 = w_5[i];\quad r_5 = w_6[i];\quad r_0 = w_1[i];$
 5:     $r_2 = w_3[i];\quad r_3 = w_4[i];\quad r_6 = w_7[i];$
 6:     $r_1 = r_1 + r_4;\quad r_5 = r_5 - r_4;\quad r_3 = (r_3 - r_2)/2;$
 7:     $r_4 = r_4 - r_0;\quad r_8 = 64 \cdot r_6;\quad r_4 = 2 \cdot r_4 + r_5;$
 8:     $r_4 = r_4 - r_8;\quad r_2 = r_2 + r_3;\quad r_1 = r_1 - 65 \cdot r_2;$
 9:     $r_2 = r_2 - r_6;\quad r_2 = r_2 - r_0;\quad r_1 = r_1 + 45 \cdot r_2;$
10:     $r_4 = (r_4 - 8 \cdot r_2)/_{24};\quad r_5 = r_5 + r_1;$
11:     $r_1 = (r_1 + 16 \cdot r_3)/_{18};\quad r_3 = -(r_3 + r_1);$
12:     $r_5 = (30 \cdot r_1 - r_5)/_{60};\quad r_2 = r_2 - r_4;$
13:     $r_1 = r_1 - r_5;$
14:     $C[i] += r_6;\quad C[i+64] += r_5;$
15:     $C[i+128] += r_4;\quad C[i+192] += r_3;$
16:     $C[i+256] += r_2;\quad C[i+320] += r_1;$
17:     $C[i+384] += r_0;$
18:     $C[4i+3] = r_3;$
19:     if i == 0 then
20:         $C[4i] = r_6;\quad C[4i+1] = r_5;\quad C[4i+2] = r_4;$
21:     else
22:         $C[4i] = (r_6 + r_9);\quad C[4i+1] = (r_5 + r_8);$
23:         $C[4i+2] = (r_4 + r_7);$
24: $C[0] -= r_2;\quad C[1] -= r_1;\quad C[2] -= r_0;$
25: $C \leftarrow C(x) \mod (x^n + 1)$
26: return $C$
```

The interpolation stage within the Toom-Cook multiplication is the inverse operation of the evaluation. Toom-Cook k-way works by evaluating the two operands in $2k-1$ different points to reduce the complexity of the polynomial multiplication. Once the result is obtained in the point-value domain, we need to interpolate these points to recover the coefficients of the polynomial. The evaluation can be defined as a matrix-vector multiplication where each row of the matrix is formed by the powers of the chosen point $((x_0)^0, (x_0)^1, \cdots, (x_0)^{2k-2})$ and the vector is formed by the coefficients of the polynomial. Therefore, the interpolation matrix is the inverse of the evaluation matrix. Moreover, it has been shown [55] that the sequence of operations determined by the Gaussian elimination method to invert the matrix yields the optimal sequence of operations performed on the intermediate polynomials to interpolate the result. Thus, Algorithm 8 with violet text is derived as the classical Toom-Cook interpolation.

Since the evaluation points are the same for classical and striding Toom-Cook, the evaluation and interpolation matrices are also the same. Consequently, the sequence of operations to compute the interpolation, i.e., lines 4-29 in Algorithm 8, is also the same for both versions. However, the access pattern to the coefficients is different. As explained when discussing evaluation, the striding version performs the ring reduction implicitly, prevents size doubling during multiplication and allows sequential access to the coefficients. Algorithm 8 with blue text shows the striding interpolation. If we compare classical and striding versions of interpolation we can observe four differences. First, in classical Toom-Cook the array where the result is

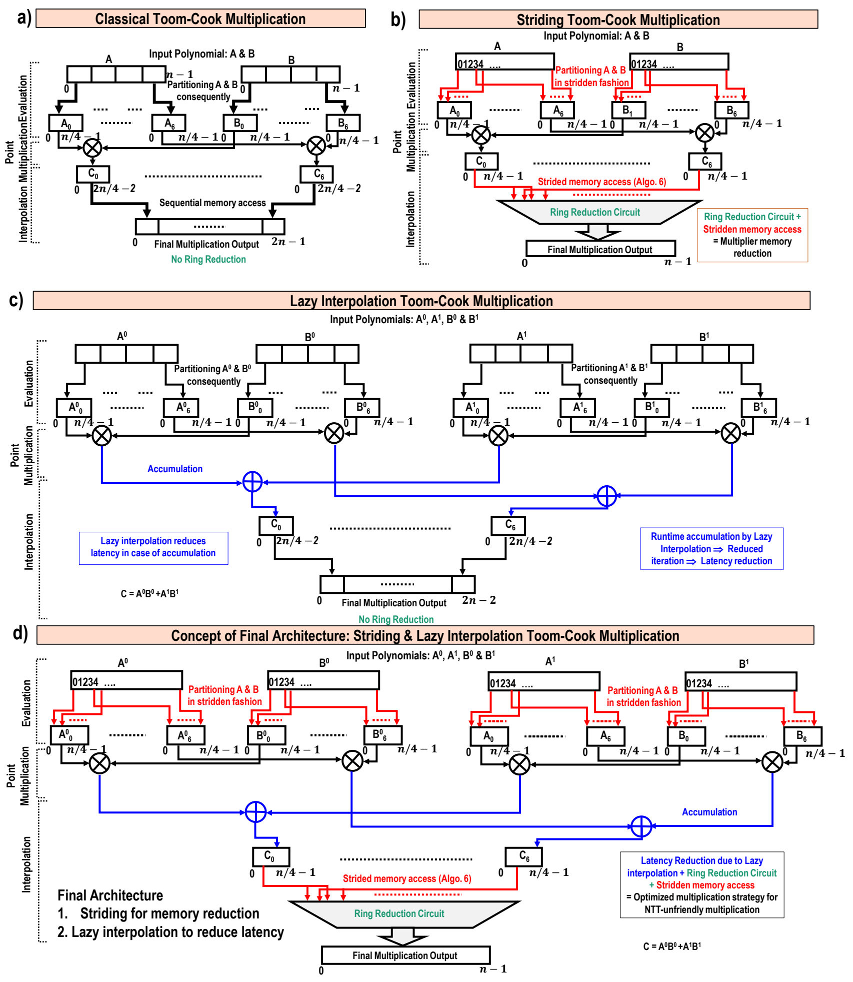

a) Classical Toom-Cook Multiplication
b) Striding Toom-Cook Multiplication
c) Lazy Interpolation Toom-Cook Multiplication
d) Concept of Final Architecture: Striding & Lazy Interpolation Toom-Cook Multiplication
Partitioning A & B in stridden fashion
Strided memory access (Algo. 6)
Ring Reduction Circuit
Final Multiplication Output
No Ring Reduction
Accumulation
Lazy interpolation reduces latency in case of accumulation
Runtime accumulation by Lazy Interpolation ⇒ Reduced iteration ⇒ Latency reduction
Final Architecture
1. Striding for memory reduction
2. Lazy interpolation to reduce latency
Latency Reduction due to Lazy interpolation + Ring Reduction Circuit + Stridden memory access = Optimized multiplication strategy for NTT-unfriendly multiplication

Fig. 4: a) Classical Toom-Cook architecture. b) Striding Toom Cook architecture that reduces memory footprint using a strided memory access. c) Lazy interpolation in classical Toom-Cook that reduces latency in interpolation with on-the-fly accumulation. d) Striding Toom-Cook with lazy interpolation, final implemented architecture to minimize both memory footprint and latency.

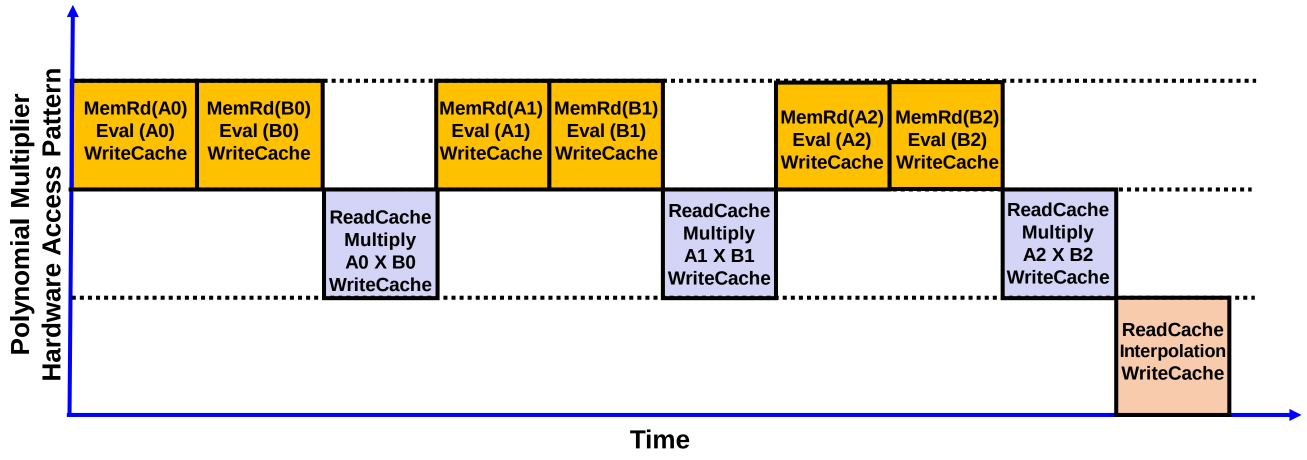

Polynomial Multiplier Hardware Access Pattern
MemRd(A0) Eval (A0) WriteCache
MemRd(B0) Eval (B0) WriteCache
ReadCache Multiply A0 X B0 WriteCache
MemRd(A1) Eval (A1) WriteCache
MemRd(B1) Eval (B1) WriteCache
ReadCache Multiply A1 X B1 WriteCache
MemRd(A2) Eval (A2) WriteCache
MemRd(B2) Eval (B2) WriteCache
ReadCache Multiply A2 X B2 WriteCache
ReadCache Interpolation WriteCache
Time

Fig. 5: Scheduling of polynomial multiplication.

stored is initialized to 0, see line 1 of Algorithm 8. This is because the ring reduction must be performed explicitly due to the size-doubling property of classical Toom-Cook. Second, thanks to the lack of size doubling the interpolation of striding Toom-Cook iterates over 64 coefficients instead of 127, see line 2 of Algorithm 8. This has an impact on memory compactness as well as on performance, which is particularly relevant in hardware where the latency of interpolation is comparable to the intermediate multiplications [44]. Third, the coefficients of the result are accumulated with 64 coefficient offsets in the classical version while they are consecutive coefficients in the striding version, see lines 30-33 in Algorithm 8 in contrast to lines 34-39. Accessing consecutive coefficients is more beneficial since it enables a higher throughput in memory operations. Lastly, in the classical version the ring reduction must be performed explicitly after the multiplication due to the size doubling. In the striding version, the output polynomial already belongs to the polynomial ring since the ring reduction happens inherently to the algorithm. Figure 4 compares the full multiplication using classical Toom-Cook (a) and striding Took-Cook (b). Both methods are described visually and the main differences are highlighted in the figure.

B. *Toom-Cook and Lazy Interpolation*

Lazy interpolation and its application to software implementations of module lattice-based cryptography has been formalized in [24]. Lazy interpolation takes advantage of the fact that Toom-Cook evaluation and interpolation are linear transformations. Therefore, operations performed in the Toom-Cook domain are equivalent to operations performed after interpolation. In the setting of Saber, we exploit the matrix-vector structure to compute the entire row-column product in the Toom-Cook domain, thus deferring the interpolation to the end. This way we trade off all but the last interpolation operations in each row-column product at the expense of extra storage in the Toom-Cook domain. A visual representation of this technique is shown in Figure 4(c). On the one hand, saving up interpolation operations becomes particularly relevant in hardware implementations where the latency of interpolation can be comparable to the latency of the intermediate multiplication of Toom-Cook. On the other hand, since the intermediate multiplications within Toom-Cook are parallelized anyway we do not incur in any memory penalization for utilizing lazy interpolation in our design. Additionally, we use lazy interpolation in combination with striding Toom-Cook to achieve an efficient hardware implementation of polynomial multiplication based on Toom-Cook.

A simplified schematic view of our proposed polynomial multiplication combining striding Toom-Cook with lazy interpolation is shown in Figure 4(d). For simplicity, the impact of each optimization is shown on a row-column multiplication of dimension only 2. We highlight with violet the changes due to the striding version of Toom-Cook and with teal the changes due to the use of lazy interpolation.

A simplified scheduling algorithm is presented in Fig. 5. Once the polynomial multiplier is enabled, it accesses system memory to access the polynomials and evaluate it. The evaluated polynomial is stored in Cache memory. After that, point multiplication is activated to access the evaluated coefficients and multiplication outputs are again stored in the Cache. Finally, at interpolation stage, cache memory is accessed and interpolated values are stored back in system memory. It should be noted that due to run-time lazy interpolation, this stage happens once after 3 evaluations and point multiplications as Saber needs 3×3 A matrix [24].

III. System Architecture & Optimizations

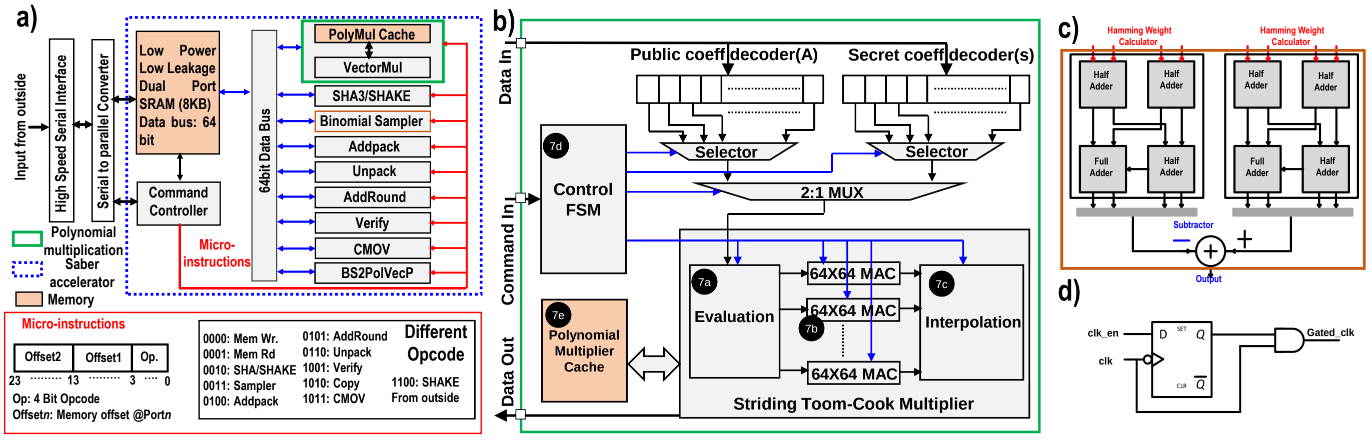


Fig. 6: a) The full system architecture of Saber core including opcode structure. b) Striding Toom-Cook multiplier architecture. c) Hamming weight-based binomial sampler. d) Clock-gating circuit used in each block to minimize power consumption.

Figure 6(a) shows the fully implemented architecture for Saber. The high-speed serial interface is used as the chip interface from the outside. The serial-to-parallel converter changes the serial input to 64-bit parallel data as memory is implemented with the 64-bit data bus. Different blocks are controlled by the command controller, which takes the input from outside and breaks it down to the command format. Based on the command, different blocks activate themselves independently. The most area and power-hungry block is the polynomial multiplier block, highlighted with green boundaries in Figure 6(a). Multiplier optimization is discussed in the next subsection. The full multiplier architecture with striding Toom-Cook and lazy interpolation is presented in Figure 6(b) SHA3/SHAKE is implemented using the standard Keccak core provided by Keccak team [56]. Binomial sampler is another important block towards any PQC core security. This is implemented in simple combinatorial xor gates to reduce power overhead as shown in Figure 6(c). The total memory requirement at the system level is 8KB. The system memory block is implemented using TSMC 65nm low power low leakage SRAM cell. It has 1K addresses with 64-bit words which is implemented continuously. Standard clock gating as shown in Figure 6(d) has been introduced to reduce leakage power further in each block, including system memory and polynomial multiplication caches.

A 24-bit micro-instruction format is used as a command to control different blocks. 4-bit opcode is used for enabling different blocks. Address offset1 and address offset2 are used as the input and output of a block respectively. However, polynomial multiplication needs two operands as inputs. Both input offset addresses are taken from both the offsets and the output offset starts from offset2 in this case.

A. *Multiplier*

The polynomial multiplier is the most power and area-hungry block even after multiple optimizations as shown in the literature [45] [53]. We have observed the same tendency from baseline implementations which has motivated us to optimize the polynomial multiplication at algorithmic, architecture and circuit level. Multiple works earlier [52] [47] have shown low latency implementations. However, key exchange mechanism is used only at the beginning of the secret communication. Hence, when co-processor cores run at hundreds of MHz, i.e., latency in the order of milliseconds, higher latency is less important as much as achieving low area or lower energy consumption.

Multiplier architecture is depicted in Figure 6(b). Two decoders are there for public matrix A and secret s. Decoders are required as data comes in packed format from the data memory. The decoder selects 13-bit or 4-bit of data as one coefficient of the public or secret polynomial, respectively. The data is fed to evaluation datapath for preprocessing. Processed data is used by 64$\times$64 MAC units to perform the intermediate products of Toom-Cook algorithm. The coefficients of such intermediate products are cached until interpolation happens. Different datapaths are discussed in subsequent subsections. Evaluation, MAC, and interpolation are implemented according to the striding ToomCook multiplication architecture. A small memory cache is used for internal data storage during polynomial multiplication operations. A control FSM controls all the sub-operations within multiplication in a timely fashion.

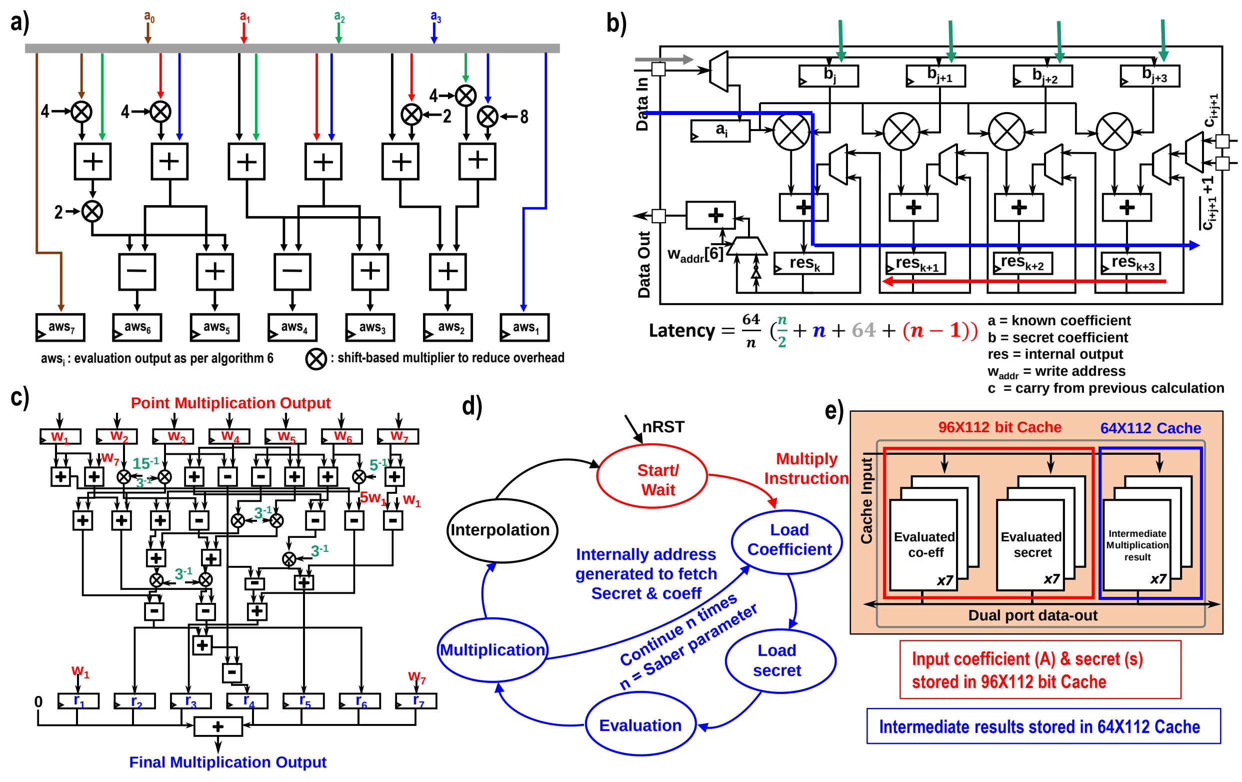

a)
a_0
a_1
a_2
a_3
aws_7
aws_6
aws_5
aws_4
aws_3
aws_2
aws_1
aws_i : evaluation output as per algorithm 6
⊗ : shift-based multiplier to reduce overhead
b)
Data In
Data Out
b_j
b_{j+1}
b_{j+2}
b_{j+3}
a_i
$W_{addr}[6]$
res_k
res_{k+1}
res_{k+2}
res_{k+3}
c_{i+j+1}
$\overline{c_{i+j+1}+1}$
$\text{Latency} = \frac{64}{n}\left(\frac{n}{2} + n + 64 + (n-1)\right)$
a = known coefficient
b = secret coefficient
res = internal output
w_{addr} = write address
c = carry from previous calculation
c)
Point Multiplication Output
w_1 w_2 w_3 w_4 w_5 w_6 w_7
15^{-1}
3^{-1}
5^{-1}
$5w_1$
r_1 r_2 r_3 r_4 r_5 r_6 r_7
Final Multiplication Output
d)
nRST
Start/ Wait
Multiply Instruction
Load Coefficient
Load secret
Evaluation
Multiplication
Interpolation
Internally address generated to fetch Secret & coeff
Continue n times n = Saber parameter
e)
96X112 bit Cache
64X112 Cache
Cache Input
Evaluated co-eff x7
Evaluated secret x7
Intermediate Multiplication result x7
Dual port data-out
Input coefficient (A) & secret (s) stored in 96X112 bit Cache
Intermediate results stored in 64X112 Cache

Fig. 7: a) Evaluation datapath of striding Toom-Cook multiplier. b) Unit multiplication of striding Toom-Cook multiplier. Latency is optimized to be comparable with [57]. c) Interpolation datapath of striding Toom-Cook. d) Finite state machine to control full vector multiplication. e) Cache structure of the striding Toom-Cook multiplier.

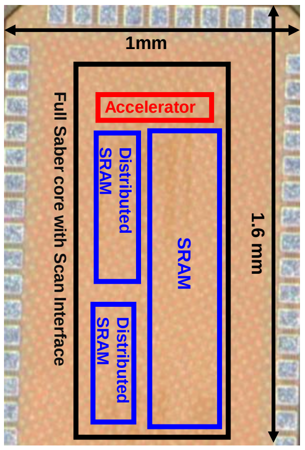
1) Evaluation Datapath: Evaluation is the first step of striding Toom-Cook multiplication. Public matrix and secret key are fed to evaluation datapath, which is implemented using Algorithm 7 as shown in Figure 7(a). Since the four coefficients used in every iteration of evaluation are consecutive, we can take advantage of 64-bit reads to fetch all four in a single clock cycle. The arithmetic operations performed during evaluation are additions of depth 2 in the worst case and multiplications by powers of 2 that can be implemented as simple bit shifts. Therefore, no additional pipelining is required to achieve perfect throughput as well as to reduce power and area. The latency of a single polynomial evaluation is only 64 clock cycles for generating the 7 intermediate polynomials in parallel.

2) Multiply and Accumulation Units: Saber performs 256×256 polynomial multiplications. Striding Toom-Cook splits each 256×256 multiplication into 7 64×64 polynomial multiplications. Each of these 7 multiplications is performed in parallel by a Multiply and Accumulation Unit as presented in Figure 7(b). Next we explain how to choose the number of parallel multipliers in every MAC Unit.

Our design is optimized to equate latency with respect to state-of-the-art NTT implementations [53] despite being lower energy and area implementation. If we assume n number of parallel multipliers in the MAC architecture, hence, all the operations will be carried out $\frac{64}{n}$ times. Now, data is fetched from dual port memory. Hence, coefficients from b, as shown in Figure 7(b), are fetched in $\frac{n}{2}$ clock cycles. n clock cycles are required to fill up the full datapath. After filling up the datapath, 64 cycles are required to perform the calculation as all 64 coefficients are multiplied. $n-1$ clock cycles are required to flush out the datapath. Total latency of the point multiplication is $\frac{64}{n} \times (\frac{n}{2} + n + 64 + (n-1))$. As n = 4, 1168 clock cycles will be required which is comparable with respect to PQC core published in [53] despite not being an NTT-based architecture.

3) Interpolation Datapath: The interpolation datapath can be derived in an equivalent manner as the evaluation datapath by directly mapping the operations in Algorithm 8 to hardware. However, this would lead to a long critical path that would result in higher area and power consumption due to additional pipelining. Instead, we reorder the operations in lines 11-29 of Algorithm 8 to reduce the depth of the circuit. In Section II it is explained that this sequence of operations is obtained from applying Gauss elimination to invert the evaluation matrix. We find a different sequence of operations that produces the same result where the depth is optimized instead of the number of operations. This new sequence of operations is equivalent to the previous one and is matched directly to the hardware shown in Figure 7(c). This datapath is pipelined in three stages to match the frequency requirements of the rest of the circuit.

4) Finite State Machine for Multiplication: Figure 7(d) sketches the finite state machine that controls multiplication. Active low reset is used to enable the FSM. After reset the FSM waits in start/wait state. As soon as a multiply instruction comes, it goes to next state. Next states are used to load the coefficients of the public polynomial and the secret. Then, evaluation followed by unit multiplication is performed. MAC units get enabled when the FSM enters the Multiplication state. This step continues in the loop for n times. This n is a cryptographic parameter, e.g., $n = 3$ for Saber. The final state is interpolation, which is performed only once for each row-vector multiplication according to the lazy interpolation optimization. After that, the FSM waits for the next row-column multiply instruction.



Chip Specification	
Technology	65nm LP
Supply Voltage	0.7-1.1 V
Package	Wire-Bond (Glob-Top Encapsulation)
Die Size	2mm X 1mm
Cryptographic Core	
Total Area	0.158mm^2
Memory Size	10.1875KB
Memory Type	Dual Port Low Power Low Leakage On-chip TSMC SRAM
Max. Frequency	160MHz
Hash Function	SHA-3
PRNG	SHAKE
Avg. Power	333.9uW @10MHz, 1.1V
Supply Decap.	60pF MOS
Scan Chain Interface	207b

Fig. 8: IC micrograph and specification.

5) Memory Components for Multiplication: Memory components are implemented using separate memory as cache. Key idea is to avoid stall cycles in the general operation. The cache is implemented using low power low leakage SRAM cells. The coefficients of the evaluated polynomial and secret are stored in a 96 × 112 cache as shown in the red border of Figure 7(e). Intermediate results are stored in a 64×112 cache implemented with the same type of SRAM cells. This multiplication memory is clock gated when data memory is enabled and multiplication FSM is at wait state. This helps us to reduce energy when multiplication is not operational. Clock gating circuit is shown in Figure 6(d).

B. Sampler

Secret polynomials in Saber follow a discrete binomial distribution. The sampler module creates this distribution from uniformly distributed data. First, data is sampled from the PRNG. This data is taken nibble(4-bit) wise and hamming weight is calculated from that. Difference between hamming weight is used as sampler output. This sampling is combinatorial. Hamming distance is calculated using a half adder and full adder as shown in Figure 6(c).

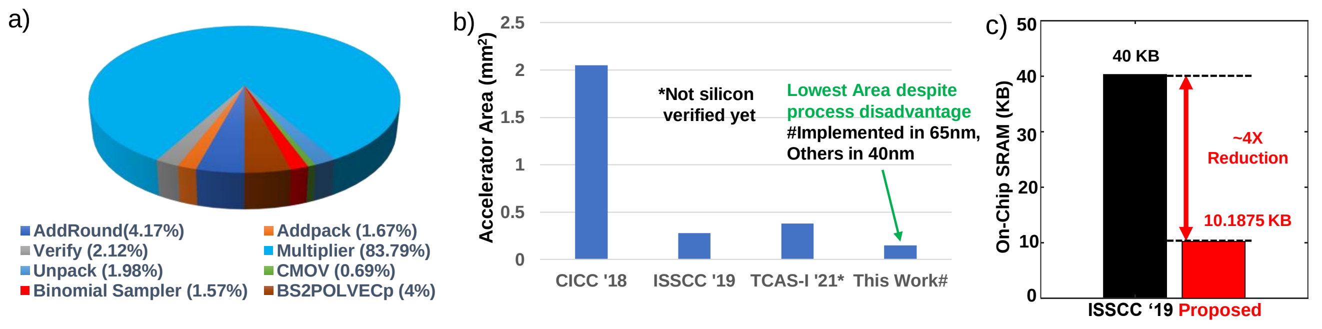


Fig. 9: (a) Area requirement of different blocks. (b) Area comparison with state-of-the-art. Our design achieves the lowest area despite process disadvantages. (c) Memory footprint improvement with respect to the state-of-the-art.

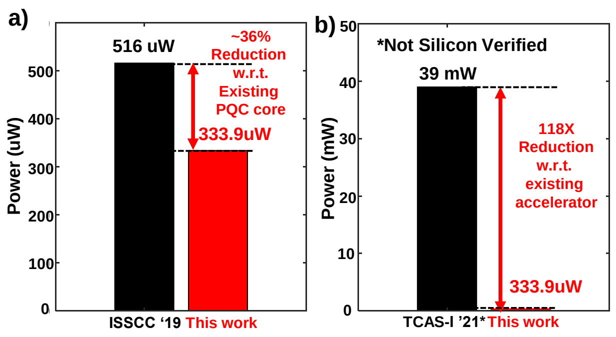


Fig. 10: Power comparison with respect to state-of-the-art.

C. Memory Components

Memory components are designed by low-power low leakage TSMC SRAM cells. A total of 8KB of memory is required for the full system as data memory. 2.1875KB distributed memory is required for the multiplication operation. All the blocks have clock gating circuit to gate themselves when they are not individually active. For example, when the multiplication block is not active, it is clock-gated. This technique helps us to reduce leakage power by a significant amount and helps us to improve energy efficiency.

D. Other Components

Throughout this section special attention was given to the design of the multiplier, sampler and system memory. The implementation of SHA3/SHAKE and the micro-instructions used to control the processor were also discussed. In this subsection the remaining necessary blocks to perform all Saber operations shown in Figure 6(a) are briefly described.

The modules AddPack, Unpack and AddRound perform very similar coefficient-wise operations on the polynomials. Particularly, AddRound performs the addition of a constant followed by the rounding operation which is used during key generation and encryption (see line 5 in Algorithms 1 and 2). AddPack performs a very similar operation during encryption except that it also adds the message encoded (see line 7 of Algorithm 2). Unpack performs a subtraction also followed by the addition of a constant followed by rounding but the constants used are different (see line 2 of Algorithm 3). These three operations are straightforward to implement with a single adder and a buffer for the rounding operation. Since these operations are performed coefficient-wise, we parallelize them for four coefficients at a time according to the data width used in the rest of the processor.

The block Verify compares two arrays, i.e., two regions of memory. It is implemented to run in constant time for a given data length. The data is compared using a xor operation and the result of the comparison is accumulated. The block CMOV implements a secure conditional move which is used during the decapsulation (see lines 4-7 of Algorithm 6). The data move takes place according to a given flag, but the block runs in any case to avoid leakage on decryption failures.

The multiplier accepts polynomials that are packed and stored in memory as arrays of 4 bits if it is a secret polynomial, as arrays of 13 bits if it is a polynomial in Saber ring, or as arrays of 16 bits for the rest of polynomials. Internally in the multiplier, all coefficients are fetched as 16 bit coefficients using the decoders shown in Figure 6(b) and explained in Section III-A. However, Saber also contemplates polynomials modulo p. The block BS2PolVecP is necessary to transform the packing of polynomials. This operation is not complicated and is implemented using a buffer. Note that all the individual blocks are clock gated when idle to reduce leakage power.

E. Scalability of Saber ASIC components

Saber defines three sets of parameters called LightSaber, Saber, and FireSaber, which match NIST security levels 1, 3, and 5 respectively. All three levels use polynomial degree N= 256, and moduli q= 2^{13} and p= 2^{10}. The three variants mostly differ in the module dimension, the binomial distribution parameter, and the message space. It should be noted that our Toom-Cook 4-way multiplier supports the multiplication with maximum width, this polynomial multiplication architecture can be reused in any of the Saber variants. However, the implemented ASIC is dedicated to Saber. Hence, the binomial sampler is 4 bits. Binomial sampler architecture slightly changes based on the Saber variant. Due to the lack of reconfigurability of the binomial sampler, this ASIC is dedicated to Saber. However, except this, architecture is fully scalable and support for LightSaber and FireSaber can be added in future version of the IC with minimal change.

IV. Silicon Results

The integrated circuit is fabricated with 65nm TSMC LP process. The wirebond type is chip-on-board and a glob-top encapsulation layer is put on top of the die. However, this area includes test circuits, free space, and memory. It should be noted that we should only care about the accelerator area as data memory can be used as general system memory when the secured connection is already established. Saber core is varied with V_{DD} from 0.7-1.1V and maximum frequency and energy is noted.

A. Area efficiency and memory footprint comparison

The total accelerator area is $0.158mm^2$. It should be noted that the accelerator area does not include area for Keccak as there is no innovation in that and standard Keccak module provided by Keccak team [56] has been used. Keccak is used as Pseudo Random Number Generator (PRNG) which can be generated using other crypto-engine as well [53]. It should be noted that Keccak core takes $0.09mm^2$ active area. However, optimized Keccak or other PRNG might take less area to provide a compact and complete solution. Optimizing Keccak module is beyond the scope of this work. It should be noted that latency and power number includes Keccak module.

Area requirements for the different blocks have been shown in Figure 9(a). The multiplier is still consuming around 83.79% of the area even after multiplier optimization. Other blocks namely binomial sampler, unpack, etc. take around 2%. The next biggest block is addround block which consumes 4.17% of area. The total accelerator area is compared with respect to state-of-the-art PQC cores. A detailed percentage of area is mentioned in figure 9 (a). Comparison with different Saber cores(not silicon-verified) and silicon-verified PQC cores is plotted in Figure 9(b). This work consumes the lowest area to date with respect to PQC cores and accelerators. It should be noted that the lowest area PQC core has been demonstrated by Banerjee et al. [57] though that is done in 40nm hence our accelerator has process disadvantages. The lowest area for Saber accelerator is reported by Zhu et al. [47] as $0.38mm^2$ which is much higher than we have implemented in our IC as shown in Figure 9(b). This work uses the lowest memory footprint till date. The accelerator needs 10.1875KB to operate which is 4$\times$ lower than existing state-of-the-art as shown in Figure 9(c). State-of-the-art implementation needs 40KB of memory.

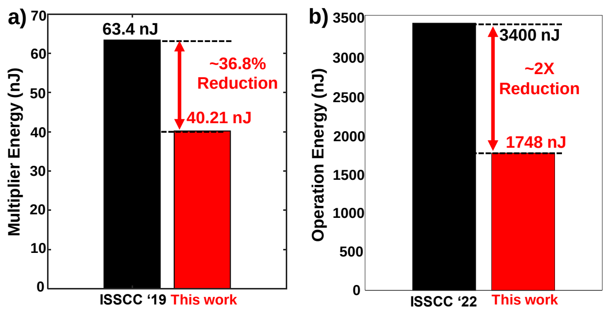


Fig. 11: Energy comparison with respect to the state-of-the-art for a) multiplier energy and b) energy per operation.

B. Power & Energy efficiency

Figure 10 shows the power difference between this work and state-of-the-art implementations. The total power consumed by the design is 333.9uW which is 38% less that the state-of-the-art PQC core (Figure 10(a)). Moreover, this accelerator is compared with respect to LWRPro [47] which is not silicon verified (Figure 10(b)). LWRPro consumes 39mW which is 118$\times$ higher than this accelerator. Average power in [48] varies from 0.855mW to 153.6mW which is higher than our implementation. Also, we refrain from mentioning it as it has incorrect functionality. Another recent solution [58] provides a reconfigurable architecture for PQC cores. The solution is mostly focused on Kyber, however, can be reconfigured to operate for Saber. This

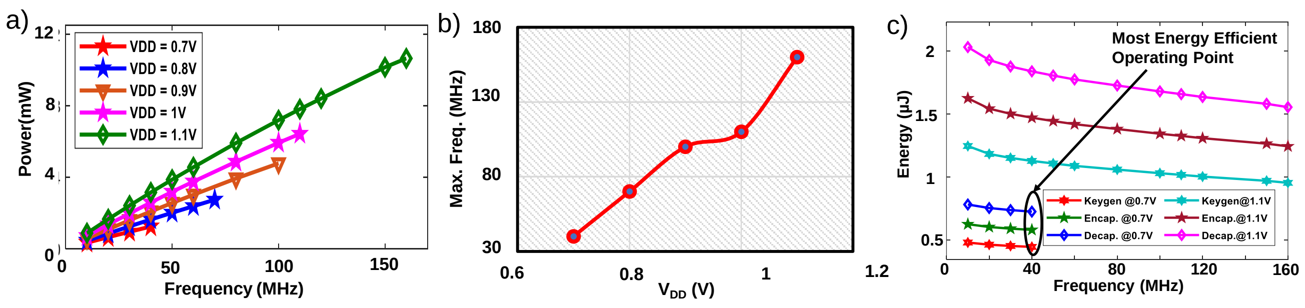


Fig. 12: (a) Load characteristics of Saber core (b) Maximum frequency at different V_{DD}. (c) The energy at the different operating frequencies. The maximum energy efficiency point is 0.7V, 40MHz.

solution mostly focuses on optimizing the latency and consumes 39-368mW (at 0.9V VDD) which is $118\times - 1000\times$ higher than our solution. Maximum frequency is observed in different V_{DD} in Figure 12(a). The IC is operational at 40MHz frequency at 0.7V V_{DD}. The accelerator works at the maximum frequency of 160MHz at 1.1V V_{DD}. Figure 12(c) shows the energy of Key generation, encapsulation, and decapsulation by the co-processor. Key generation, encapsulation, and decapsulation take $\leq$ 2uJ energy at 1.1V at all frequencies. Energy is reduced as frequency is increased. The increasing frequency will reduce latency, hence the effect of leakage power will be reduced in the final energy calculation. This concept leads to the lowest energy at 0.7V V_{DD} and 40MHz frequency. Key generation, encapsulation, and decapsulation consume 444.1, 579.4 & 724.5 nJ energy for all the operations combined in the above-mentioned operating point.

To have an apple-to-apple comparison we compare the energy of point multiplication with Banerjee et al [57]. It should be noted that the state-of-the-art PQC core uses NTT structure for point multiplication however we use optimized striding Toom-Cook-based polynomial multiplication with lazy interpolation which is performance-wise very similar to NTT structure. As discussed in section IIIA earlier, the total latency of the point multiplication is $\frac{64}{n} \times (\frac{n}{2} + n + 64 + (n-1))$. As $n = 4$, 1168 clock cycles will be required which is comparable with respect to PQC core published in [53] despite not being an NTT-based architecture. Additional 60 clock cycles are required at the evaluation and 70 clock cycles at interpolation. However, it should be noted that interpolation happens once in 3 multiplication. However, to consider the worst case scenario, we need a total of 1298 clock cycles which is similar to NTT architecture presented in [57]. As our implementation is a low power implementation (333.9uW at 0.7V VDD and 40MHz), this leads to 40.21nJ energy consumption per multiplication which is 36.8% less than state-of-the-art [57] as shown in Fig. 11(a). We define a single operation by a combination of key generation, encapsulation, and decapsulation and compare this with respect to state-of-the-art core [58] in FIg. 11(b). It is observed that the implemented core consumes 1748nJ energy per operation which is $2\times$ lower than [58].

	Cortex-M4[19]	CICC'18[43]	ISSCC'19[47]	TCAS-I[a][40]	ISSCC'22[49]	This work
Technology	-	40nm	40nm	40nm	28nm	65nm
Supply Voltage	3-5	0.9	0.68-1.1	1.1	0.9	0.7-1.1
Frequency (MHz)	100	300	12-72	400	500	40-160
Total Processor Area (mm^2)	-	2.05	0.28	0.38	3.6	0.158
Supported Lattice Crypto Primitives	All	Ring-LWE	Ring-LWE & Module-LWE	Module-LWR	Ring-LWE & Module-LWE	Module-LWR
Supported Lattice Parameters	All	N: 64-2048 q: 32-bit conf.	N: 64-2048 q: 24-bit conf.	N: 256 q: 13-bit	<24bit; power of 2;	N: 256 q: 13-bit
Average Power	-	140mW	519uW	35-41mW	39-368mW	333.9uW
KEM Scheme (keygen + encapsulation + decapsulation)						
Energy Efficiency (uJ/Op)	-	-	26.6	12.8	3.4	1.748[c]
Latency (cycles)	-	-	479644	4230	10433	57014
Latency (us)	-	-	4830	10.6	21	352
Individual Multiplier Performance						
Multiplier Cycle	65459	160	1288	81+pipeline	32	1298*
Energy (nJ)	40.1×10^3	31	63.4	-	2.5-23.6[d]	40.21

[a] Not silicon verified. Results reported in simulation. *Interpolation needs 70 clock cycle, which happens once in 3 multiplication, Evaluation clock cycle added. [c]Lowest energy consumption despite process disadvantage. [d]Estimated from minimum power consumption and number of clock cycle.

TABLE I: Comparison with state-of-the-art. This accelerator uses the lowest memory footprint and lowest area till date. It also consumes 333.9uW power which is the lowest till now.

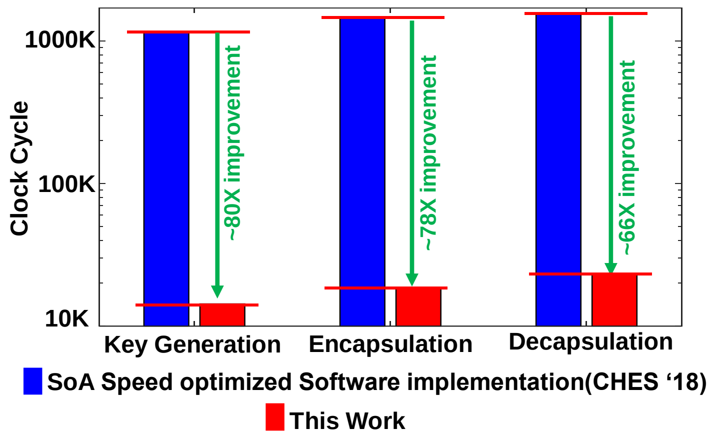


Fig. 13: Clock cycle comparison with respect to state-of-the-art speed-optimized software implementation.

C. Latency comparison

Though latency is not our utmost priority as stated earlier, we wanted to match latency with respect to state-of-the post-quantum cores and improve with respect to speed-optimized software implementation. All KEM operations (keygen, encapsulation, and decapsulation) are observed to be finished within 89, 117 & 146 μs respectively at 160MHz frequency (80X improvement over fastest software implementation as shown in Figure 13). Precisely, the design takes 14642, 18984 & 23388 clock cycles to finish Saber Keygen, Encapsulation & Decapsulation respectively. Note that this design is not optimized based on latency as KEM will be used once in a while to establish secure communication links, unlike symmetric key. However, we still ensure the design is as fast as possible within the scope of a fully compact area and power-optimized design as shown in Figure 13.

V. Conclusion

This work presents the first silicon-verified IC for the Saber accelerator. Moreover, a combined striding Toom-Cook and lazy interpolation Toom-Cook 4-way approach is taken to reduce computation complexity which helps in reducing area and energy overhead which was one of the selection criteria for NIST PQC standardization procedure. Moreover, clock gating in all the blocks including in 3 different memories while other operations are ongoing, multiplication using shift multiplier at evaluation step, less number of loop operations due to algorithmic innovation namely lazy interpolation and optimized memory operation by striding Toom-Cook polynomial multiplication reduces total energy consumption with respect to state-of-the-art PQC core as well requires less memory footprint and area. This IC consumes $0.158mm^2$ area which is the lowest amongst all the PQC cores and accelerators. It also consumes 333.9μW of average power at the optimum point which is the lowest reported to date amongst the PQC cores. Polynomial multiplication takes 40.21nJ energy with comparable latency with state-of-the-art hardware as shown in Table I.

References

[1] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.
[2] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, apr 1988.
[3] Christof Paar and Jan Pelzl. *Hash Functions*, pages 293–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.
[4] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.
[5] Neal Koblitz. *Elliptic curve cryptosystems*, pages 203–209. Mathematics of Computation. 48, 1987.
[6] 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.
[7] Peter W. Shor. Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Sci. Statist. Comput.*, 26:1484, 1997.
[8] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *eprint arXiv:quant-ph/0301141*, January 2003.
[9] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.
[10] NIST. Post-quantum cryptography standardization. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization, 2017. [Online; accessed 30-june-2022].
[11] 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. https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf, 2022. [Online; accessed 10-August-2022].
[12] Oded Regev. New lattice-based cryptographic constructions. *Journal of the ACM*, 51(6):pp. 899–942, 2004.
[13] 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.

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

[15] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, Jose Maria Bermudo Mera, Michiel Van Beirendonck, and Andrea Basso. SABER. Technical report, National Institute of Standards and Technology, 2020. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[16] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 333–342, 2009.

[17] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012.

[18] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2016.

[19] Jacob Alperin-Sheriff and Daniel Apon. Dimension-preserving reductions from LWE to LWR. *IACR Cryptol. ePrint Arch.*, page 589, 2016.

[20] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties and applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 57–74. Springer, 2013.

[21] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancréde Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber. algorithm specifications and supporting documentation. (round 3 submission). https://pq-crystals.org/kyber/data/kyber-specification-round3-20210131.pdf, 2021. [Online; accessed 30-June-2022].

[22] Cong Chen, Oussama Danba, Jeffrey Hoffstein, Andreas Hülsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, and Zhenfei Zhang. NTRU algorithm specifications and supporting documentation,. Second PQC Standardization Conference, 2019, University of California, Santa Barbara, USA, 2019.

[23] J. M. Pollard. The fast fourier transform in a finite field. *Mathematics of Computation*, 25(114):365–374, 1971.

[24] Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede. Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):pp. 222–244, 2020.

[25] 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 Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):pp. 243–266, 2018.

[26] Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang. NTT multiplication for ntt-unfriendly rings new speed records for saber and NTRU on cortex-m4 and AVX2. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):pp. 159–188, 2021.

[27] 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.

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

[29] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962.

[30] Angshuman Karmakar, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. Pushing the speed limit of constant-time discrete gaussian sampling. A case study on falcon. *IACR Cryptol. ePrint Arch.*, page 267, 2019.

[31] Angshuman Karmakar, Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Constant-time discrete gaussian sampling. *IEEE Trans. Computers*, 67(11):1561–1571, 2018.

[32] Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede. A side-channel-resistant implementation of SABER. *ACM J. Emerg. Technol. Comput. Syst.*, 17(2):10:1–10:26, 2021.

[33] Suparna Kundu, Jan-Pieter D'Anvers, Michiel Van Beirendonck, Angshuman Karmakar, and Ingrid Verbauwhede. Higher-order masked saber. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*, volume 13409 of *Lecture Notes in Computer Science*, pages 93–116. Springer, 2022.

[34] Arvind Singh, Monodeep Kar, Sanu Mathew, Anand Rajan, Vivek De, and Saibal Mukhopadhyay. 25.3 A 128b AES Engine with Higher Resistance to Power and Electromagnetic Side-Channel Attacks Enabled by a Security-Aware Integrated All-Digital Low-Dropout Regulator. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 404–406, February 2019. ISSN: 2376-8606, 0193-6530.

[35] Archisman Ghosh, Debayan Das, Josef Danial, Vivek De, Santosh Ghosh, and Shreyas Sen. 36.2 an em/power sca-resilient aes-256 with synthesizable signature attenuation using digital-friendly current source and ro-bleed-based integrated local feedback and global switched-mode control. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 499–501. IEEE, 2021.

[36] Archisman Ghosh, Debayan Das, Josef Danial, Vivek De, Santosh Ghosh, and Shreyas Sen. Syn-stellar: An em/power sca-resilient aes-256 with synthesis-friendly signature attenuation. *IEEE Journal of Solid-State Circuits*, 57(1):167–181, 2021.

[37] Archisman Ghosh, Dong-Hyun Seo, Debayan Das, Santosh Ghosh, and Shreyas Sen. A digital cascoded signature attenuation countermeasure with intelligent malicious voltage drop attack detector for em/power sca resilient parallel aes-256. In *2022 IEEE Custom Integrated Circuits Conference (CICC)*, pages 01–02. IEEE, 2022.

[38] Archisman Ghosh, Debayan Das, Santosh Ghosh, and Shreyas Sen. Em sca & fi self-awareness and resilience with single on-chip loop & ml classifiers. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 592–595. IEEE, 2022.

[39] Archisman Ghosh, Mayukh Nath, Debayan Das, Santosh Ghosh, and Shreyas Sen. Electromagnetic analysis of integrated on-chip sensing loop for side-channel and fault-injection attack detection. *IEEE Microwave and Wireless Components Letters*, 32(6):784–787, 2022.

[40] James Howe, Marco Martinoli, Elisabeth Oswald, and Francesco Regazzoni. Exploring parallelism to improve the performance of frodokem in hardware. *Journal of Cryptographic Engineering*, 11(4):317–327, 2021.

[41] Ferhat Yaman, Ahmet Can Mert, Erdinç Öztürk, and Erkay Savas. A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 1020–1025. IEEE, 2021.

[42] Viet Ba Dang, Kamyar Mohajerani, and Kris Gaj. High-speed hardware architectures and FPGA benchmarking of crystals-kyber, ntru, and saber. *IACR Cryptology ePrint Archive*, page 1508, 2021.

[43] Bo-Yuan Peng, Adrian Marotzke, Ming-Han Tsai, Bo-Yin Yang, and Ho-Lin Chen. Streamlined NTRU prime on FPGA. *IACR Cryptology ePrint Archive*, page 1444, 2021.

[44] 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.

[45] 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):pp. 443–466, 2020.

[46] Yihong Zhu, Min Zhu, Bohan Yang, Wenping Zhu, Chenchen Deng, Chen Chen, Shaojun Wei, and Leibo Liu. A high-performance hardware implementation of saber based on karatsuba algorithm. *IACR Cryptology ePrint Archive*, page 1037, 2020.

[47] Yihong Zhu, Min Zhu, Bohan Yang, Wenping Zhu, Chenchen Deng, Chen Chen, Shaojun Wei, and Leibo Liu. Lwrpro: An energy-efficient configurable crypto-processor for module-lwr. *IEEE Transactions on Circuits and Systems I: Regular Paper*, 68(3):1146–1159, 2021.

[48] Malik Imran, Felipe Almeida, Andrea Basso, Sujoy Sinha Roy, and Samuel Pagliarini. High-speed SABER key encapsulation mechanism in 65nm CMOS. *IACR Cryptology ePrint Archive*, page 530, 2022.

[49] Hugo Krawczyk, Kenneth G Paterson, and Hoeteck Wee. On the security of the tls protocol: A systematic analysis. In *Annual Cryptology Conference*, pages 429–448. Springer, 2013.

[50] Archisman Ghosh, Jose Maria Bermudo Mera, Angshuman Karmakar, Debayan Das, Santosh Ghosh, Ingrid Verbauwhede, and Shreyas Sen. A 334uw 0.158 mm2 asic for post-quantum key-encapsulation mechanism saber with low-latency striding toom–cook multiplication. *IEEE Journal of Solid-State Circuits*, 2023.

[51] Archisman Ghosh, Angshuman Karmakar, Debayan Das, Santosh Ghosh, Ingrid Verbauwhede, and Shreyas Sen. A 334uw 0.158 mm2 saber learning with rounding based post-quantum crypto accelerator. In *2022 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–2. IEEE, 2022.

[52] Shiming Song, Wei Tang, Thomas Chen, and Zhengya Zhang. LEIA: A 2.05mm^2 140mw lattice encryption instruction accelerator in 40nm CMOS. In *2018 IEEE Custom Integrated Circuits Conference, CICC 2018, San Diego, CA, USA, April 8-11, 2018*, pages 1–4. IEEE, 2018.

[53] Utsav Banerjee, Tenzin S. Ukyab, and Anantha P. Chandrakasan. Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(4):17–61, 2019.

[54] Daniel J. Bernstein. Multidigit multiplication for mathematicians, 2001. http://cr.yp.to/papers/m3.pdf.

[55] Marco Bodrato and Alberto Zanoni. Integer and polynomial multiplication: towards optimal toom-cook matrices. In *Symbolic and Algebraic Computation, International Symposium, ISSAC 2007, Waterloo, Ontario, Canada, July 28 - August 1, 2007, Proceedings*, pages 17–24, 2007.

[56] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Keccak in vhdl. https://keccak.team/hardware.html.

[57] Utsav Banerjee, Abhishek Pathak, and Anantha P Chandrakasan. 2.3 an energy-efficient configurable lattice cryptography processor for the quantum-secure internet of things. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 46–48. IEEE, 2019.

[58] Yihong Zhu, Wenping Zhu, Min Zhu, Chongyang Li, Chenchen Deng, Chen Chen, Shuying Yin, Shouyi Yin, Shaojun Wei, and Leibo Liu. A 28nm 48kops 3.4µj/op agile crypto-processor for post-quantum cryptography on multi-mathematical problems. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, volume 65, pages 514–516, 2022.