A 334 μW 0.158 mm^2 ASIC for Post-Quantum Key-Encapsulation Mechanism Saber With Low-Latency Striding Toom–Cook Multiplication

Archisman Ghosh, *Graduate Student Member, IEEE*, Jose Maria Bermudo Mera, Angshuman Karmakar, Debayan Das, Santosh Ghosh, Ingrid Verbauwhede, *Fellow, IEEE*, and Shreyas Sen, *Senior Member, IEEE*

***Abstract*— Lattice-based cryptography is a novel approach to public key cryptography (PKC), 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, the National Institute of Standards and Technology (NIST) follows this approach. The multiplication of polynomials is the central bottleneck in the computation of lattice-based cryptography. Because PKC is mostly used to establish common secret keys, the 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 article, 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 post-quantum cryptography (PQC) core, 4× less memory, 36.8% reduction in multiplier energy, and 118× reduction in active power with respect to state-of-the-art Saber accelerator (not silicon verified). This accelerator consumes 0.158-mm^2 active area, which is the lowest reported to date despite the process disadvantages of the state-of-the-art designs.**

***Index Terms*— Compact design, energy-efficient architecture, first accelerator, lazy interpolation, memory-efficient, post-quantum cryptography (PQC), striding Toom–Cook.**

Manuscript received 16 September 2022; revised 23 December 2022; accepted 15 February 2023. This article was approved by Associate Editor Kathryn Wilcox. This work was supported in part by the National Science Foundation (NSF) under Grant CNS 17-19235 and Grant CNS 19-35573 and in part by Intel Corporation. *(Corresponding author: Archisman Ghosh.)*

Archisman Ghosh and Shreyas Sen are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: ghosh69@purdue.edu).

Jose Maria Bermudo Mera is with COSIC, Katholieke Universiteit Leuven (KU Leuven), 3000 Leuven, Belgium, and also with PQShield Ltd., OX2 7HT Oxford, U.K.

Angshuman Karmakar is with IIT Kanpur, Kanpur 208016, India.

Debayan Das and Santosh Ghosh are with the Intel Labs, Intel Corporation, Hillsboro, OR 97124 USA.

Ingrid Verbauwhede is with COSIC, Katholieke Universiteit Leuven (KU Leuven), 3000 Leuven, Belgium.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSSC.2023.3253425.

Digital Object Identifier 10.1109/JSSC.2023.3253425

I. Introduction

The bulk of our current public-key infrastructure is based on two schemes, Rivest–Shamir–Adleman (RSA) [1], and elliptic-curve cryptography (ECC) [2], [3]. The security of these public key cryptography (PKC) schemes arises from the computational intractability of the underlying hard problems, which are large integer factorization and elliptic-curve discrete logarithm problems, respectively. However, quantum algorithms, such as Shor's [4] and Proos-Zalka's [5] algorithms, can solve these two problems *easily*, i.e., in polynomial time, using a *large* quantum computer and, therefore, completely compromising their security. Although it might take many years to develop quantum computers large enough to pose a threat to our current PKC schemes, it also takes decades to develop cryptosystems from theoretically hard mathematical problems to make them suitable for widespread public deployment. Considering that our current public-key infrastructure is based mostly on RSA and ECC, the replacement of our current PKC with quantum-resistant PKC is needed to maintain the security of our digital world in the future, and action must be taken now.

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) [6] in 2017 was a prudent step toward developing PQC. Initially, 59 and 23 schemes were submitted in key-encapsulation mechanism (KEM) and digital signature categories, respectively. Three lattice-based (Saber, Kyber, and NTRU) and one code-based algorithms (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 Fig. 1. NIST has recently mentioned Kyber as the new standard. However, the NIST report on finalist candidates [7] mentioned that all the different experiments suggest that 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)

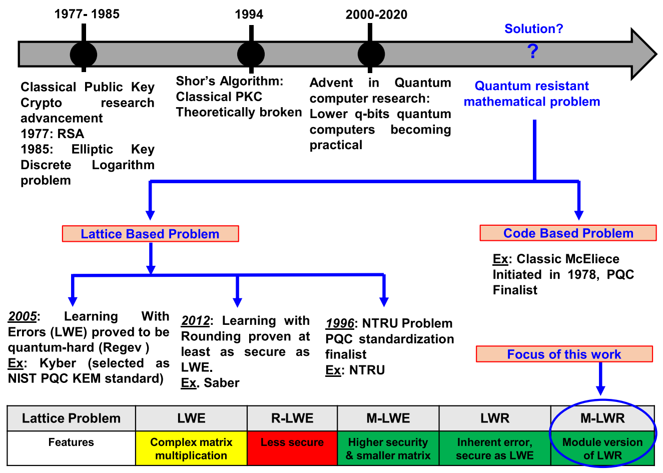

1977- 1985
1994
2000-2020
Solution?
?
Classical Public Key Crypto research advancement
1977: RSA
1985: Elliptic Key Discrete Logarithm problem
Shor's Algorithm: Classical PKC Theoretically broken
Advent in Quantum computer research: Lower q-bits quantum computers becoming practical
Quantum resistant 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.

problem [8]. 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-LWE (R-LWE) [9] 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 LWE (M-LWE) [10] 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 KEM [11] that 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 out that $A \cdot s$ is a very good *entropy-diffuser*. In fact, $A \cdot s$ itself can be used as a pseudorandom 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 [8] 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 [8]. 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 (SVPs) in random lattices in the worst case [12]. As no known quantum algorithms can solve the 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. [13] first introduced the LWR problem that 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 the LWE problem. One of their main results was to show that, if q/p is sufficiently big, then $\lfloor A \cdot s \rceil \overset{\text{stat}}{\approx} \lfloor A \cdot s + e \rceil$. Thus, combining this with the Regev's [8], [9] 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. [14], Alperin-Sheriff and Apon [15], and Alwen et al. [16] 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 [11] of Saber submission.

B. Saber

Saber [11] is a lattice-based post-quantum KEM. It is one of the four 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 LWR (M-LWR) problem. **Key generation**, as 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**, as 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'. In addition, 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**, as 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()` [11]

```
1: $\text{seed}_{\boldsymbol{A}} \leftarrow \mathcal{U}(\{0,1\}^{256})$
2: $\boldsymbol{A} = \texttt{gen}(\text{seed}_{\boldsymbol{A}}) \in R_q^{l\times l}$
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 $(pk := (\text{seed}_{\boldsymbol{A}}, \boldsymbol{b}), \boldsymbol{s})$
```

Algorithm 2 `Saber.PKE.Enc`$(pk = (\boldsymbol{b}, \text{seed}_{\boldsymbol{A}}), m \in R_2; r)$ [11]

```
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`$[\boldsymbol{s}, c = (c_m, \boldsymbol{b}')]$ [11]

```
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()`

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

The chosen ciphertext attack (CCA)-secure 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. In addition, 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 and 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.

Algorithm 5 `Saber.KEM.Encaps`$[pk := (\text{seed}_{\boldsymbol{A}}, \boldsymbol{b})]$

```
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 $K = \mathcal{H}(\mathcal{H}(c), z)$
```

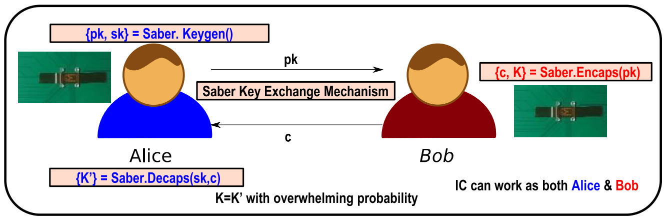

{pk, sk} = Saber. Keygen()
pk
Saber Key Exchange Mechanism
{c, K} = Saber.Encaps(pk)
c
Alice
Bob
{K'} = Saber.Decaps(sk,c)
K=K' with overwhelming probability
IC can work as both Alice & Bob

Fig. 2. Sample KEM mechanism between two communicating parties. Note that our IC can act as both parties. Keygen(), Encaps(), and Decaps() are presented in Algorithms 4–6, respectively.

A sample KEM is shown in Fig. 2 between two parties, namely, Alice and Bob. Alice creates the public key and secret key using Algorithm 4. She sends the 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 and Bob here.

C. Features of Saber

Three salient features of Saber as the PQC-KEM scheme are given as follows.

1) *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)-based polynomial multiplication [17]. However, Saber uses an unorthodox choice of power-of-two modulus. It has been shown that Saber performance is at par with similar schemes that use NTT-based polynomial multiplication on different platforms [18], [19], [20] by combining sub-quadratic polynomial multiplication algorithms Toom–Cook [21], [22] and Karatsuba [23].

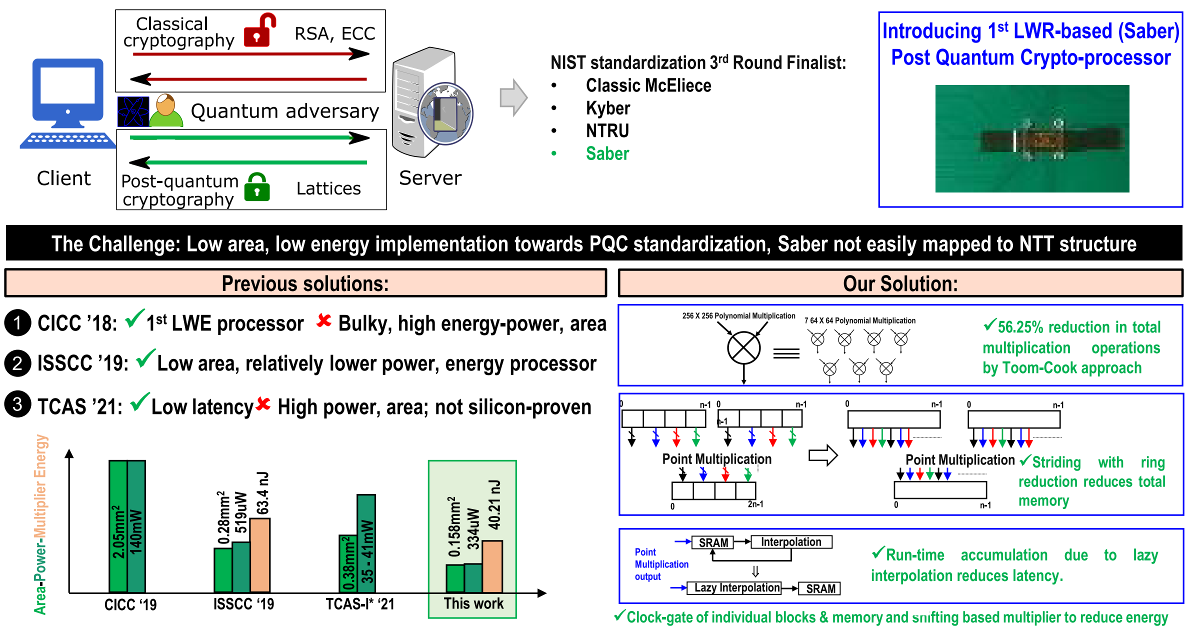

Classical cryptography
RSA, ECC
Quantum adversary
Post-quantum cryptography
Lattices
Client
Server
NIST standardization 3rd Round Finalist:
Classic McEliece
Kyber
NTRU
Saber
Introducing 1st LWR-based (Saber) Post Quantum Crypto-processor
The Challenge: Low area, low energy implementation towards PQC 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
✓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.

2) *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.

3) *Resistance to Side-Channel Attacks:* Saber uses constant-time algorithms to prevent side-channel attacks, such as timing or simple-power analysis attacks. Generating noise inherently by rounding and using centered binomial distributions instead of more complex discrete Gaussian distributions [24], [25] 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 [26], [27]. 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 [11]. 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 [28], [29], [30], [31]. Integrating these countermeasures (masking or physical countermeasures) with this Saber architecture will help in more sophisticated DPA/CPA security. These countermeasures can be integrated into 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 [7, Sec. 4.3.4] mentions that the security, performance, bandwidth usage, and so on of Saber are 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 [32] is already present in many popular products, such as OpenSSH (version 9.0 onward), GPL, and WolfSSL.

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 the 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 datapaths to suit the parameters of Kyber, deciding how matrix A is generated, and so on. 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 [33], Kyber [34], NTRU [35], NTRUPrime [36], and Saber [37] [38].

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) [37]. *Second*, high performance can be achieved with a fully parallel multiplier where the full polynomials are loaded and shifted after every multiplication [38]. *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 [39]. Both the second and third approaches have been used to implement Saber on full custom hardware [40] [41]. Though Imran et al. [41] propose another hardware for Saber, as mentioned in the paper [41], due to a logic bug, the address offsets of three of the four distributed memories are incorrectly decoded, and data are overwritten. This logical error results in a few flipped bits on the output of the chip compared with the expected results. Due to this issue, we refrain from comparing performance directly with this work [41] in the comparison table as the table includes only solid-state circuit literature with functional ICs.

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 the transport layer security (TLS) protocol, which is a newer version of the secure socket layer (SSL) protocol [42]. 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, such as 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 that 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 the 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 and reduce operational costs by achieving low power and low energy.

In this article, we also aim at filling this gap in the State-of-the-Art by bringing the first approach to silicon. Moreover, fundamental constraints, such as low area and low energy, when implementing key exchange operations make the *first approach*, i.e., Toom–Cook, more attractive for a dedicated IC. In Section IV, we show different metrics to compare our design to other Saber ASIC designs and ASIC designs [43] that accelerate other lattice-based schemes, such as NewHope [43] and Kyber [44].

F. Contributions

Fig. 3 shows the key contributions of this work that we categorically describe in the following.

1) *First Silicon-Verified Implementation of Saber:* We have presented the first Saber core and, un until now, one of the few *fully functional* PQC ASICs. Therefore, this work is a cornerstone in the development of PQC and in the transition from classical PKC to PQC.
2) *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 [20]. The 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 the total SRAM used in the IC. A 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 [18] in Toom–Cook

multiplication context. Mera et al. [18] focus on a software Cortex-M4-based latency versus memory optimization problem for Toom–Cook multiplication. In addition, 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.

3) *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.
4) *Lowest Area and Power Implementation Among the PQC Cores and Accelerators:* This work provides 36% power improvement from silicon-verified PQC core and 118× improvement with respect to state-of-the-art Saber accelerator (not silicon verified) [40]. Saber core takes 0.158 mm^2 of the area and 10.1875-kB memory, which is the lowest among PQC cores to date.

G. System Overview and Article Organization

This IC has dedicated blocks for each micro-operations, namely, polynomial multiplication, binomial sampler, addpack, addround, cmov, verify, and so on. Polynomial multiplication is the most computationally complex and, hence, the most area and power-hungry block. This is chosen for algorithmic and 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 an 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 of 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, in 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** [38]. 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 [35].

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 the platform level to take advantage of the available resources. This approach is the most popular for software implementations [20], [37], but it can also be followed with successful results on hardware [37].

The polynomial multiplication module is implemented following the second approach. Moreover, we use Toom–Cook four-way to reduce the complexity of the polynomial multiplication and parallelize it as in [37]. Thus, for Saber parameters, a single 256×256 polynomial multiplication is broken down into seven 64×64 polynomial multiplications, which are parallelized at the 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.

1) 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.
2) Striding Toom–Cook reduces the memory requirement and, hence, helps in reducing energy consumption and total area.
3) Lazy interpolation helps in reducing the number of iterations and helps in latency improvement.

A. Classical Toom–Cook Versus Striding Toom–Cook

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 by 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 [19] 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 an HW/SW co-design approach, this can be solved by transferring the

Algorithm 7 Evaluation of Classical and Striding Toom–Cook Four-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 $w_7$
```

Algorithm 8 Interpolation of Classical and Striding Toom–Cook Four-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:     $r_7 = r_0$;   $r_8 = r_1$;   $r_9 = r_2$;
4:     $r_1 = w_2[i]$;   $r_4 = w_5[i]$;   $r_5 = w_6[i]$;   $r_0 = w_1[i]$;
5:     $r_2 = w_3[i]$;   $r_3 = w_4[i]$;   $r_6 = w_7[i]$;
6:     $r_1 = r_1 + r_4$;   $r_5 = r_5 - r_4$;   $r_3 = (r_3 - r_2)/2$;
7:     $r_4 = r_4 - r_0$;   $r_8 = 64 \cdot r_6$;   $r_4 = 2 \cdot r_4 + r_5$;
8:     $r_4 = r_4 - r_8$;   $r_2 = r_2 + r_3$;   $r_1 = r_1 - 65 \cdot r_2$;
9:     $r_2 = r_2 - r_6$;   $r_2 = r_2 - r_0$;   $r_1 = r_1 + 45 \cdot r_2$;
10:    $r_4 = (r_4 - 8 \cdot r_2)/_{24}$;   $r_5 = r_5 + r_1$;
11:    $r_1 = (r_1 + 16 \cdot r_3)/_{18}$;   $r_3 = -(r_3 + r_1)$;
12:    $r_5 = (30 \cdot r_1 - r_5)/_{60}$;   $r_2 = r_2 - r_4$;
13:    $r_1 = r_1 - r_5$;
14:    $C[i] += r_6$;   $C[i + 64] += r_5$;
15:    $C[i + 128] += r_4$;   $C[i + 192] += r_3$;
16:    $C[i + 256] += r_2$;   $C[i + 320] += r_1$;
17:    $C[i + 384] += r_0$;
18:    $C[4i + 3] = r_3$;
19:    if i == 0 then
20:        $C[4i] = r_6$;   $C[4i + 1] = r_5$;   $C[4i + 2] = r_4$;
21:    else
22:        $C[4i] = (r_6 + r_9)$;   $C[4i + 1] = (r_5 + r_8)$;
23:        $C[4i + 2] = (r_4 + r_7)$;
24: $C[0] -= r_2$;   $C[1] -= r_1$;   $C[2] -= r_0$;
25: $C \leftarrow C(x) \mod (x^n + 1)$
26: return $C$
```

polynomials with the appropriate layout [37], 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 [45] 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 an 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.

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 $((x_0)^0, (x_0)^1, \ldots, (x_0)^{2k-2})$, 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 [46] 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 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 and performance, which is particularly relevant in hardware, where the latency of interpolation is comparable to the intermediate multiplications [37]. 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. Finally, 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. Fig. 4 compares the full multiplication using classical Toom–Cook [see Fig. 4(a)] and striding Took-Cook [see Fig. 4(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 have been formalized in [18]. 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 Fig. 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 any memory penalization for utilizing lazy interpolation in our design. In addition, 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 Fig. 4(d). For simplicity, the impact of each optimization is shown on a row–column multiplication of dimension only 2. We highlight the changes from classical Toom–Cook multiplication to striding Toom–Cook multiplication with violet in Algorithms 7 and 8. We also highlight the changes required to use lazy interpolation in teal in those algorithms.

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 them. 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 the 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 three evaluations and point multiplications as Saber needs 3×3 A matrix [18].

III. System Architecture and Optimizations

Fig. 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 Fig. 6(a). Multiplier optimization is discussed in Section III-A. The full multiplier architecture with striding Toom–Cook and lazy interpolation is presented in Fig. 6(b). SHA3/SHAKE is implemented using the standard Keccak core provided by the Keccak team [48]. The binomial sampler is another important block toward any PQC core security. This is implemented in simple combinatorial XOR gates to reduce power overhead, as shown in Fig. 6(c). The total memory requirement at the system level is 8 kB. The system memory block is implemented using a TSMC 65-nm low-power low-leakage SRAM cell. It has 1k addresses with 64-bit words, which is implemented continuously. Standard clock gating, as shown in Fig. 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. The 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 [38], [44]. We have observed the same tendency from baseline implementations, which has motivated us to optimize the polynomial multiplication at algorithmic, architecture, and circuit levels. Multiple works earlier [40], [43]

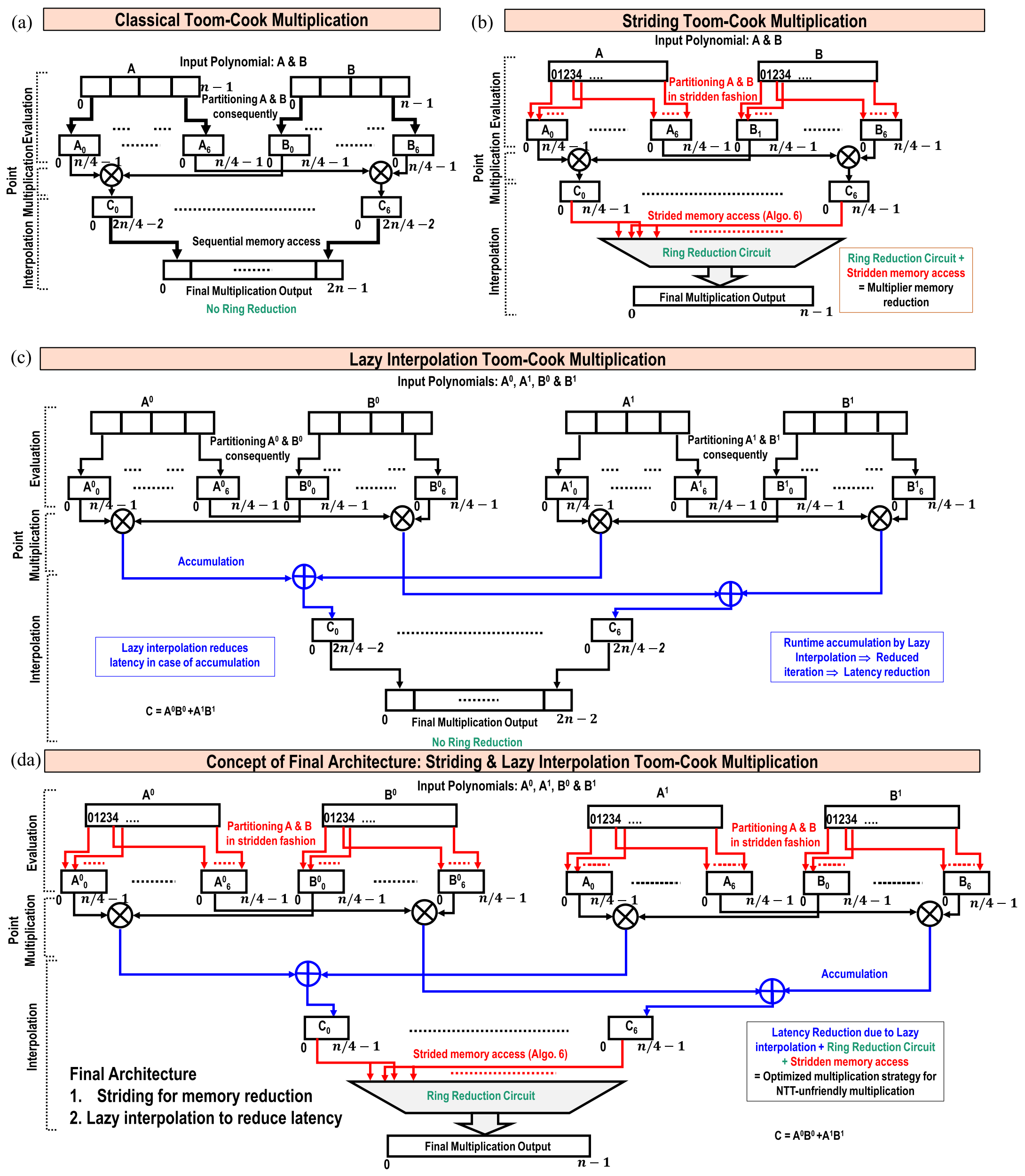

(a) Classical Toom-Cook Multiplication
Input Polynomial: A & B
Partitioning A & B consequently
Sequential memory access
Final Multiplication Output
No Ring Reduction
(b) Striding Toom-Cook Multiplication
Partitioning A & B in stridden fashion
Strided memory access (Algo. 6)
Ring Reduction Circuit
Ring Reduction Circuit + Stridden memory access = Multiplier memory reduction
(c) Lazy Interpolation Toom-Cook Multiplication
Input Polynomials: A0, A1, B0 & B1
Accumulation
Lazy interpolation reduces latency in case of accumulation
Runtime accumulation by Lazy Interpolation ⇒ Reduced iteration ⇒ Latency reduction
C = A0B0+A1B1
(da) Concept of Final Architecture: Striding & Lazy Interpolation Toom-Cook Multiplication
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
Evaluation
Point Multiplication
Interpolation

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.

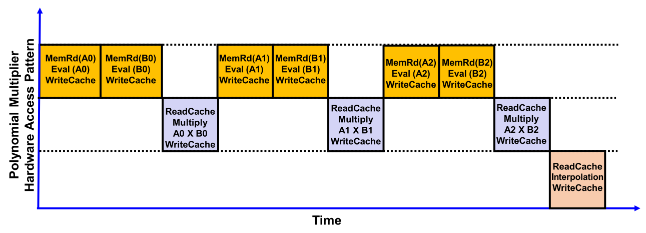


Fig. 5. Scheduling of the polynomial multiplication.

have shown low latency implementations. However, the 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 Fig. 6(b). Two decoders are there for public matrix A and secret s. Decoders are required as data come in packed format from the data memory. The decoder selects 13 or 4 bits of data as one coefficient of the public or secret polynomial, respectively. The data are fed to the evaluation datapath for preprocessing. Processed data are used by 64 $\times$ 64 multiply and accumulation (MAC) units to perform the intermediate products of the Toom–Cook algorithm. The coefficients of such intermediate products are cached until interpolation happens. Different datapaths are discussed in Sections III-A1–III-A3. Evaluation, MAC, and interpolation are implemented according to the striding Toom–Cook 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.

1) Evaluation Datapath: Evaluation is the first step of striding Toom–Cook multiplication. Public matrix and secret

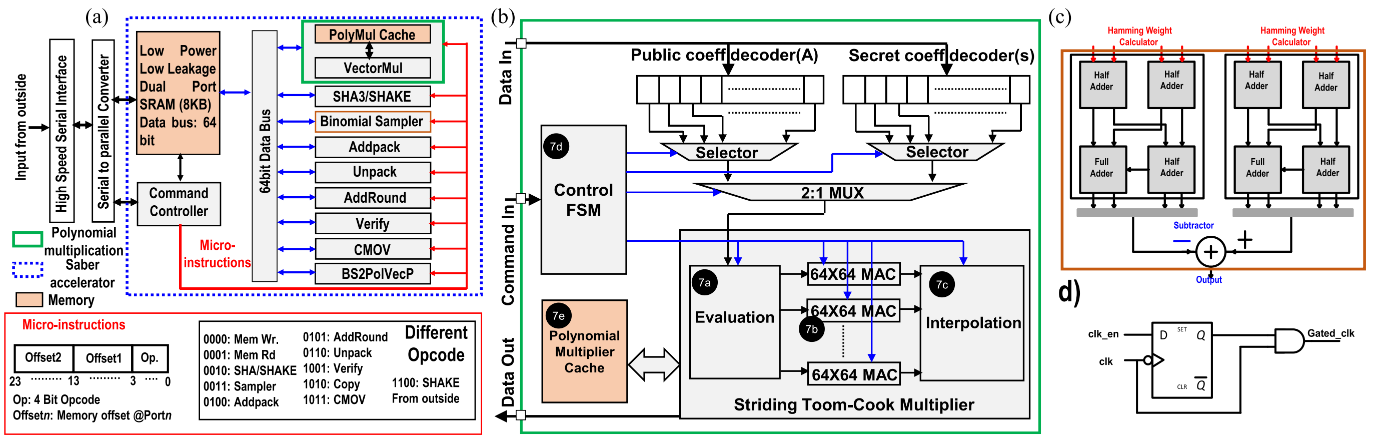
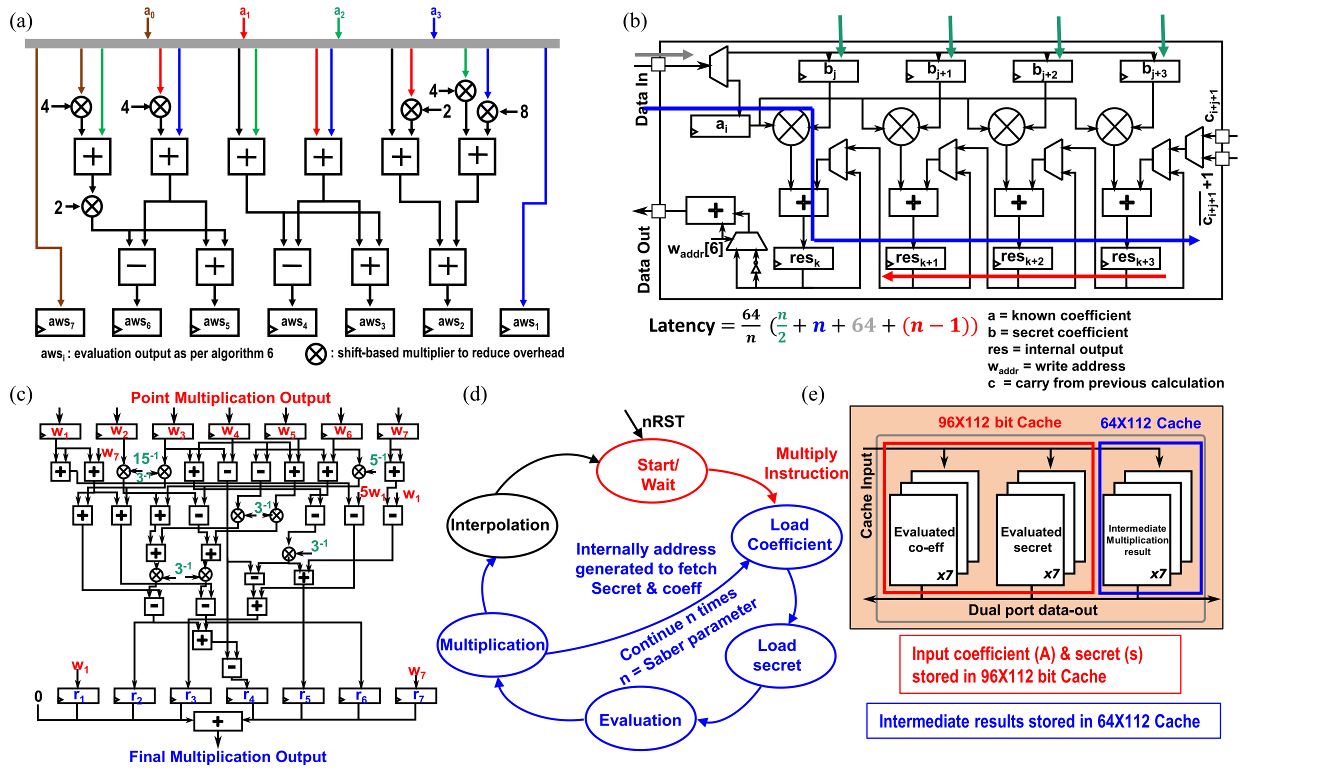


Fig. 6. (a) Full system architecture of the Saber core, including the opcode structure. (b) Striding Toom–Cook multiplier architecture. (c) Hamming weight-based binomial sampler. (d) Clock-gating circuit used in each block to minimize power consumption.



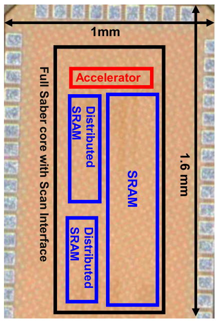
(a)
a_0 a_1 a_2 a_3
aws_7 aws_6 aws_5 aws_4 aws_3 aws_2 aws_1
aws_i : evaluation output as per algorithm 6
: shift-based multiplier to reduce overhead
(b)
Data In
Data Out
b_j b_{j+1} b_{j+2} b_{j+3}
a_i
c_{i+j+1}
$w_{addr}[6]$
res_k res_{k+1} res_{k+2} res_{k+3}
$\text{Latency} = \frac{64}{n}\left(\frac{n}{2} + n + 64 + (n-1)\right)$
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 the striding Toom–Cook multiplier. (b) Unit multiplication of the striding Toom–Cook multiplier. Latency is optimized to be comparable with [47]. (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.

key are fed to evaluation datapath, which is implemented using Algorithm 7, as shown in Fig. 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 and reduce power and area. The latency of a single polynomial evaluation is only 64 clock cycles for generating the seven intermediate polynomials in parallel.

2) Multiply and Accumulation Units: Saber performs 256×256 polynomial multiplications. Striding Toom–Cook splits each 256×256 multiplication into seven 64×64 polynomial multiplications. Each of these seven multiplications is performed in parallel by an MAC unit, as presented in Fig. 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 [44] 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 $(64/n)$ times. Now, data are fetched from dual port memory. Hence, coefficients from b, as shown in Fig. 7(b), are fetched in $(n/2)$ 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



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.

cycles are required to flush out the datapath. Total latency of the point multiplication is $(64/n) \times ((n/2) + n + 64 + (n-1))$. As $n = 4$, 1168 clock cycles will be required which is comparable with respect to PQC core published in [44] 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 Fig. 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: Fig. 7(d) sketches the finite state machine that controls multiplication. Active low reset is used to enable the FSM. After resetting, the FSM waits in the start/wait state. As soon as a multiply instruction comes, it goes to the 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 an 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.

5) Memory Components for Multiplication: Memory components are implemented using separate memory as a 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 Fig. 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 a wait state. This helps us to reduce energy when multiplication is not operational. Clock gating circuit is shown in Fig. 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 are sampled from the PRNG. These data are taken nibble (4-bit) wise, and the Hamming weight is calculated from that. The difference between the Hamming weights is used as sampler output. This sampling is combinatorial. The Hamming distance is calculated using a half adder and a full adder, as shown in Fig. 6(c).

C. Memory Components

Memory components are designed by low-power low-leakage TSMC SRAM cells. A total of 8 kB of memory is required for the full system as data memory. The 2.1875-kB distributed memory is required for the multiplication operation. All the blocks have clock gating circuits 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 section, the remaining necessary blocks to perform all Saber operations shown in Fig. 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 are compared using an 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

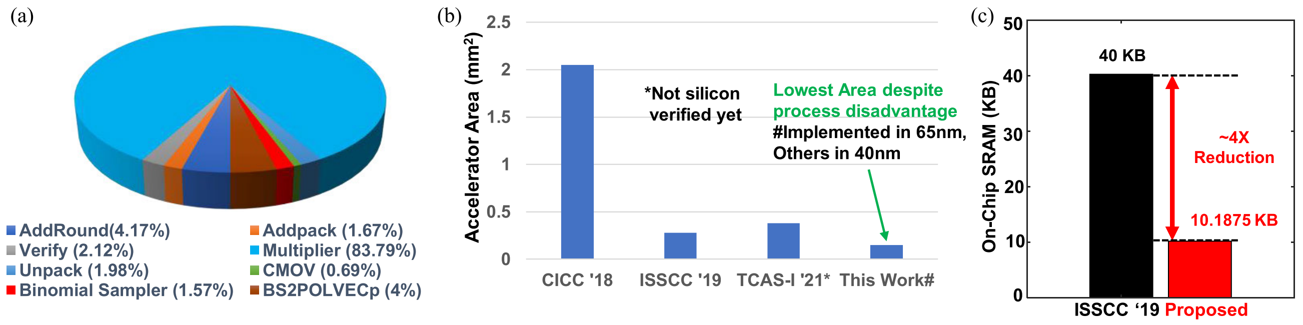


Fig. 9. (a) Area requirement of different blocks. (b) Area comparison with state of the art. Our design achieves the lowest area despite process disadvantages. (c) Memory footprint improvement with respect to the state-of-the-art.

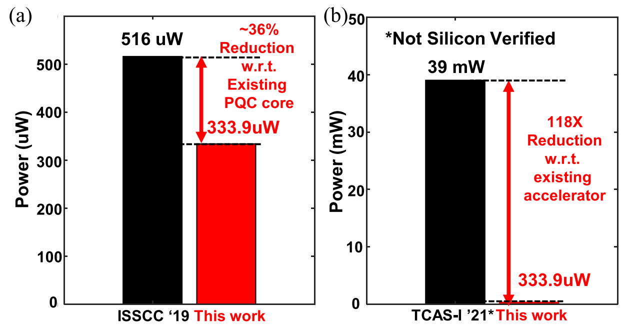


Fig. 10. Power comparison with respect to (a) state-of-the-art for any PQC core and (b) state-of-the-art Saber accelerator.

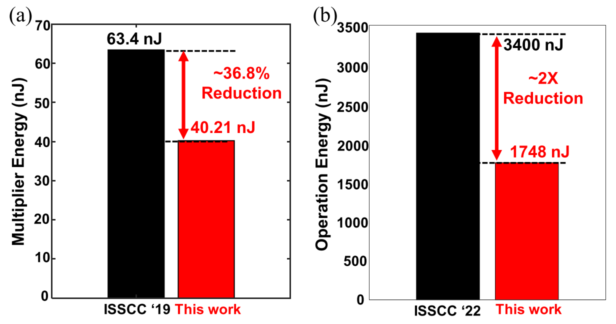


Fig. 11. Energy comparison with respect to the state of the art for (a) multiplier energy and (b) energy per operation.

using the decoders shown in Fig. 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 four-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, the architecture is fully scalable, and support for LightSaber and FireSaber can be added in future versions of the IC with minimal change.

IV. Silicon Results

The integrated circuit shown in Fig. 8 is fabricated with the 65-nm 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 to 1.1 V, and maximum frequency and energy are noted.

A. Area Efficiency and Memory Footprint Comparison

The total accelerator area is 0.158 mm^2. It should be noted that the accelerator area does not include an area for Keccak as there is no innovation in that, and the standard Keccak module provided by the Keccak team [48] has been used. Keccak is used as the pseudorandom number generator (PRNG), which can be generated using other crypto-engine as well [44]. It should be noted that Keccak core takes 0.09-mm^2 active area. However, optimized Keccak or other PRNG might take less area to provide a compact and complete solution. Optimizing the Keccak module is beyond the scope of this work. It should be noted that latency and power numbers include the Keccak module.

Area requirements for the different blocks have been shown in Fig. 9(a). The multiplier is still consuming around 83.79% of the area even after multiplier optimization. Other blocks, namely, binomial sampler, unpack, and so on, take around 2%. The next biggest block is the addround block, which consumes 4.17% of the area. The total accelerator area is compared with respect to state-of-the-art PQC cores. A detailed percentage of area is mentioned in Fig. 9(a). Comparison with different Saber cores (not silicon-verified) and silicon-verified PQC cores is plotted in Fig. 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. [47] though that is done in 40 nm; hence, our accelerator has process disadvantages. The lowest area for the Saber accelerator is reported by

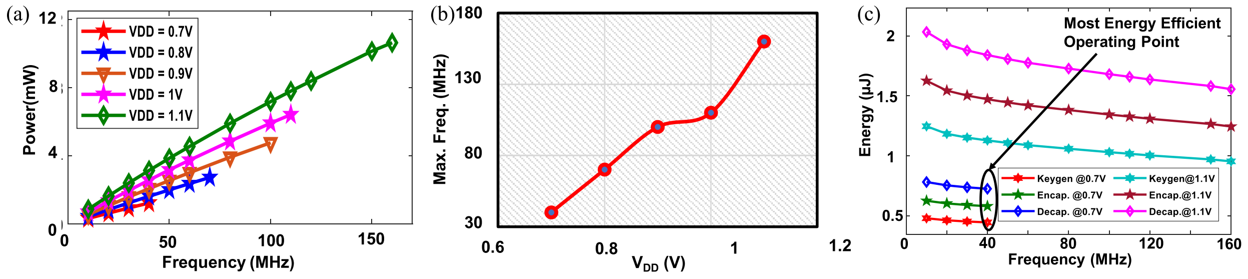


Fig. 12. (a) Load characteristics of Saber core. (b) Maximum frequency at different V_{DD}'s. (c) Energy at the different operating frequencies. The maximum energy efficiency point is 0.7 V, 40 MHz.

Zhu et al. [40] as 0.38 mm^2, which is much higher than we have implemented in our IC, as shown in Fig. 9(b). This work uses the lowest memory footprint to date. The accelerator needs 10.1875 kB to operate, which is 4× lower than the existing state-of-the-art, as shown in Fig. 9(c). state-of-the-art implementation needs 40 kB of memory.

B. Power and Energy Efficiency

Fig. 10 shows the power difference between this work and state-of-the-art implementations. The total power consumed by the design is 333.9 μW, which is 38% less that the state-of-the-art PQC core [see Fig. 10(a)]. Moreover, this accelerator is compared with respect to LWRPro [40], which is not silicon verified [see Fig. 10(b)]. LWRPro consumes 39 mW, which is 118× higher than this accelerator. Average power in [41] varies from 0.855 to 153.6 mW, which is higher than our implementation. Also, we refrain from mentioning it as it has incorrect functionality. Another recent solution [49] provides a reconfigurable architecture for PQC cores. The solution is mostly focused on Kyber and, however, can be reconfigured to operate for Saber. This solution mostly focuses on optimizing the latency and consumes 39–368 mW (at 0.9-V VDD), which is 118×–1000× higher than our solution. As our implementation is a low-power implementation (333.9 μW at 0.7-V VDD and 40 MHz), this leads to 40.21-nJ energy consumption per multiplication, which is 36.8% less than state of the art [47], 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 [49] in FIg. 11(b). It is observed that the implemented core consumes 1748-nJ energy per operation, which is 2× lower than [49]. Maximum frequency is observed in different V_{DD}'s in Fig. 12(a). The IC is operational at a 40-MHz frequency at 0.7-V V_{DD}. The accelerator works at the maximum frequency of 160 MHz at 1.1-V V_{DD}. Fig. 12(c) shows the energy of key generation, encapsulation, and decapsulation by the coprocessor. Key generation, encapsulation, and decapsulation take $\leq$2-μJ energy at 1.1 V 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 0.7-V V_{DD} and 40-MHz frequency. Key generation, encapsulation, and decapsulation consume 444.1-, 579.4-, and 724.5-nJ energy for all the operations combined in the abovementioned operating point.

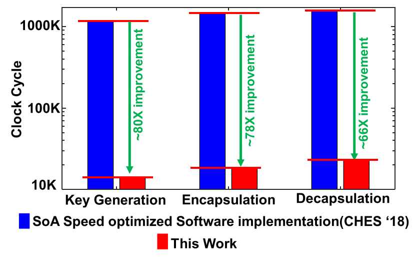


Fig. 13. Clock cycle comparison with respect to state-of-the-art speed-optimized software implementation.

To have an apple-to-apple comparison, we compare the energy of point multiplication with Banerjee et al. [47]. 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 III-A, the total latency of the point multiplication is $(64/n) \times ((n/2) + n + 64 + (n-1))$. As $n = 4$, 1168 clock cycles will be required, which is comparable with respect to PQC core published in [44] 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 three multiplications. However, to consider the worst case scenario, we need a total of 1298 clock cycles, which is similar to NTT architecture presented in [47].

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, and 146 μs, respectively, at 160-MHz frequency (80× improvement over fastest software implementation, as shown in Fig. 13). Precisely, the design takes 14 642, 18 984, and 23 388 clock cycles to finish Saber Keygen, Encapsulation, and 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 keys. However, we still ensure that the design is as fast as possible within the scope of a fully compact area and power-optimized design, as shown in Fig. 13.

TABLE I

COMPARISON WITH STATE OF THE ART. THIS ACCELERATOR USES THE LOWEST MEMORY FOOTPRINT AND LOWEST AREA TO DATE. IT ALSO CONSUMES 333.9-uW POWER, WHICH IS THE LOWEST UP UNTIL NOW

	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 x 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.

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 four-way approach is taken to reduce computation complexity, which helps in reducing area and energy overhead, which was one of the selection criteria for the NIST PQC standardization procedure. Moreover, 1) clock gating for the blocks at rest, 2) shift-based multiplier at evaluation, 3) reduced loop operation by lazy interpolation, and 4) optimized memory operation by striding approach reduces total energy consumption, memory footprint and area of our PQC core. This IC consumes 0.158-mm^2 area, which is the lowest among all the PQC cores and accelerators. It also consumes 333.9 μW of average power at the optimum point, which is the lowest reported to date amongst the PQC cores. Polynomial multiplication takes 40.21-nJ energy with comparable latency with state-of-the-art hardware, as shown in Table I.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Math. Comput.*, vol. 48, no. 177, pp. 203–209, 1987.

[3] S. V. Miller, "Use of elliptic curves in cryptography," in *Advances in Cryptology—CRYPTO*, C. H. Williams, Ed. Berlin, Germany: Springer, 1986, pp. 417–426.

[4] W. P. Shor, "Polynomial time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Sci. Statist. Comput.*, vol. 26, p. 1484, Jan. 1997.

[5] J. Proos and C. Zalka, "Shor's discrete logarithm quantum algorithm for elliptic curves," 2003, *arXiv:quant-ph/0301141*.

[6] NIST. (2017). *Post-Quantum Cryptography Standardization*. Accessed: Jun. 30, 2022. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization

[7] G. Alagic. (2022). *Angela Robinson, and Daniel Smith-Tone, Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process*. Accessed: Aug. 10, 2022. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/ir/2022/NIST.IR.8413.pdf

[8] O. Regev, "New lattice-based cryptographic constructions," *J. ACM*, vol. 51, no. 6, pp. 899–942, 2004.

[9] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Advances in Cryptology* (Lecture Notes in Computer Science), vol. 6110, H. Gilbert, Ed. Berlin, Germany: Springer-Verlag, 2010, pp. 1–23.

[10] A. Langlois and D. Stehlé, "Worst-case to average-case reductions for module lattices," *Des., Codes Cryptogr.*, vol. 75, no. 3, pp. 565–599, 2015.

[11] J.-P. D'Anvers. (2020). *SABER, Technical Report, National Institute of Standards and Technology*. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

[12] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, May 2009, pp. 333–342.

[13] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* (Lecture Notes in Computer Science), vol. 7237. Cambridge, U.K.: Springer 2012, pp. 719–737.

[14] A. Bogdanov, S. Guo, D. Masny, S. Richelson, and A. Rosen, "On the hardness of learning with rounding over small modulus," in *Theory of Cryptography* (Lecture Notes in Computer Science), vol. 9562. Tel Aviv, Israel: Springer, 2016, pp. 209–224.

[15] J. Alperin-Sheriff and D. Apon, "Dimension-preserving reductions from LWE to LWR," *IACR Cryptol. ePrint Arch.*, p. 589, Jun. 2016.

[16] J. Alwen, S. Krenn, K. Pietrzak, and D. Wichs, "Learning with rounding, revisited—New reduction, properties and applications," in *Proc. 33rd Annu. Cryptol. Conf.* (Lecture Notes in Computer Science), vol. 8042, R. Canetti, and J. A. Garay, Eds. Santa Barbara, CA, USA: Springer, 2013, pp. 57–74.

[17] J. M. Pollard, "The fast Fourier transform in a finite field," *Math. Comput.*, vol. 25, no. 114, pp. 365–374, Apr. 1971.

[18] J. M. B. Mera, A. Karmakar, and I. Verbauwhede, "Time-memory trade-off in Toom-Cook multiplication: An application to module-lattice based cryptography," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2, pp. 222–244, Mar. 2020.

[19] A. Karmakar, J. M. Mera, S. S. Roy, and I. Verbauwhede, "Saber on ARM CCA-secure module lattice-based key encapsulation on ARM," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 3, pp. 243–266, 2018.

[20] C.-M.-M. Chung, V. Hwang, M. J. Kannwischer, G. Seiler, C.-J. Shih, and B.-Y. Yang, "NTT multiplication for NTT-unfriendly rings: New speed records for saber and NTRU on cortex-M4 and AVX2," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2, pp. 159–188, Feb. 2021.

[21] A. L. Toom, "The complexity of a scheme of functional elements realizing the multiplication of integers," *Soviet Math.-Doklady*, vol. 7, pp. 714–716, Jul. 1963. [Online]. Available: http://toomandre.com/my-articles/engmat/MULT-E.PDF

[22] S. A. Cook, "On the minimum computation time of functions," Ph.D. thesis, Harvard Univ., Cambridge, MA, USA, 1966, ch. 3, pp. 51–77.

[23] A. Karatsuba and Y. Ofman, "Multiplication of many-digital numbers by automatic computers," *Doklady Akademii Nauk SSSR*, vol. 145, no. 2, pp. 293–294, 1962.

[24] A. Karmakar, S. S. Roy, F. Vercauteren, and I. Verbauwhede, "Pushing the speed limit of constant-time discrete Gaussian sampling—A case study on the Falcon signature scheme," in *Proc. 56th ACM/IEEE Design Automat. Conf. (DAC)*, Las Vegas, NV, USA, 2019, pp. 1–6.

[25] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede, "Constant-time discrete Gaussian sampling," *IEEE Trans. Comput.*, vol. 67, no. 11, pp. 1561–1571, Nov. 2018.

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

[27] S. Kundu, J. D'Anvers, M. V. Beirendonck, A. Karmakar, and I. Verbauwhede, "Higher-order masked saber," in *Security and Cryptography for Networks* (Lecture Notes in Computer Science), vol. 13409, C. Galdi and S. Jarecki, Eds. Amalfi, Italy: Springer, 2022, pp. 93–116.

[28] A. Singh, M. Kar, S. Mathew, A. Rajan, V. De, and S. Mukhopadhyay, "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 *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 404–406.

[29] A. Ghosh, D. Das, J. Danial, V. De, S. Ghosh, and S. Sen, "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 *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 499–501.

[30] A. Ghosh, D. Das, J. Danial, V. De, S. Ghosh, and S. Sen, "Syn-STELLAR: An EM/power SCA-resilient AES-256 with synthesis-friendly signature attenuation," *IEEE J. Solid-State Circuits*, vol. 57, no. 1, pp. 167–181, Jan. 2022.

[31] A. Ghosh, D.-H. Seo, D. Das, S. Ghosh, and S. Sen, "A digital cascoded signature attenuation countermeasure with intelligent malicious voltage drop attack detector for EM/power SCA resilient parallel AES-256," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2022, pp. 1–2.

[32] C. Chen et al., "NTRU algorithm specifications and supporting documentation," in *Proc. 2nd PQC Standardization Conf.*, vol. 2019. Santa Barbara, CA, USA: Univ. California, 2019, pp. 1–41.

[33] J. Howe, M. Martinoli, E. Oswald, and F. Regazzoni, "Exploring parallelism to improve the performance of FrodoKEM in hardware," *J. Cryptograph. Eng.*, vol. 11, no. 4, pp. 317–327, Nov. 2021.

[34] F. Yaman, A. C. Mert, E. Ozturk, and E. Savas, "A hardware accelerator for polynomial multiplication operation of CRYSTALS-KYBER PQC scheme," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Grenoble, France, Feb. 2021, pp. 1020–1025.

[35] V. B. Dang, K. Mohajerani, and K. Gaj, "High-speed hardware architectures and FPGA benchmarking of CRYSTALS-Kyber, NTRU, and Saber," *IEEE Trans. Comput.*, vol. 72, no. 2, pp. 306–320, Feb. 2023, doi: 10.1109/TC.2022.3222954.

[36] B.-Y. Peng, A. Marotzke, M.-H. Tsai, B.-Y. Yang, and H.-L. Chen, "Streamlined NTRU prime on FPGA," *J. Cryptograph. Eng.*, p. 1444, Nov. 2022, doi: 10.1007/s13389-022-00303-z.

[37] J. M. B. Mera, F. Turan, A. Karmakar, S. S. Roy, and I. Verbauwhede, "Compact domain-specific co-processor for accelerating module lattice-based KEM," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, San Francisco, CA, USA, Jul. 2020, pp. 1–6.

[38] S. S. Roy and A. Basso, "High-speed instruction-set coprocessor for lattice-based key encapsulation mechanism: Saber in hardware," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2020, no. 4, pp. 443–466, 2020.

[39] Y. Zhu et al., "LWRpro: An energy-efficient configurable crypto-processor for module-LWR," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 3, pp. 1146–1159, Mar. 2021, doi: 10.1109/TCSI.2020.3048395.

[40] Y. Zhu et al., "LWRpro: An energy-efficient configurable crypto-processor for module-LWR," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 3, pp. 1146–1159, Mar. 2021.

[41] M. Imran, F. Almeida, A. Basso, S. S. Roy, and S. Pagliarini, "High-speed SABER key encapsulation mechanism in 65 nm CMOS," *IACR Cryptology ePrint Archive*, p. 530, May 2022.

[42] H. Krawczyk, K. G. Paterson, and H. Wee, "On the security of the TLS protocol: A systematic analysis," in *Proc. Annu. Cryptol. Conf.* Cham, Switzerland: Springer, 2013, pp. 429–448.

[43] S. Song, W. Tang, T. Chen, and Z. Zhang, "LEIA: A 2.05 mm^2 140 mW lattice encryption instruction accelerator in 40 nm CMOS," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, San Diego, CA, USA, Apr. 2018, pp. 1–4.

[44] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2019, pp. 17–61, Aug. 2019.

[45] J. D. Bernstein. (2001). *Multidigit Multiplication for Mathematicians*. [Online]. Available: http://cr.yp.to/papers/m3.pdf

[46] M. Bodrato and A. Zanoni, "Integer and polynomial multiplication: Towards optimal Toom-Cook matrices," in *Proc. Int. Symp. Symbolic Algebr. Comput.*, Waterloo, ONT, Canada, Jul. 2007, pp. 17–24.

[47] U. Banerjee, A. Pathak, and A. P. Chandrakasan, "An energy-efficient configurable lattice cryptography processor for the quantum-secure Internet of Things," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2019, pp. 46–48.

[48] G. Bertoni, J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. *Keccak in Vhdl*. Accessed: Mar. 14, 2022. [Online]. Available: https://keccak.team/hardware.html

[49] Y. Zhu et al., "A 28 nm 48 KOPS 3.4 μJ/Op agile crypto-processor for post-quantum cryptography on multi-mathematical problems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2022, pp. 514–516.

Archisman Ghosh (Graduate Student Member, IEEE) received the Bachelor of Electronics and Telecommunication Engineering degree from Jadavpur University, Kolkata, India, in 2017. He is currently pursuing the Ph.D. degree in electrical and computer engineering with Purdue University, West Lafayette, IN, USA, working with Prof. Shreyas Sen.

Prior to starting his Ph.D. degree, he worked as a Digital Design & Verification Engineer at Samsung Semiconductor India Research and Development (R&D), Bengaluru, India. He interned with the Power Delivery Circuits & Systems Group, Intel Labs, Hillsboro, OR, USA, over the summer of 2020. His research interests include mixed-signal IC design and hardware security.

Mr. Ghosh was a recipient of the prestigious SSCS Predoctoral Achievement Award for the year 2022. He was a recipient of the prestigious ECE Fellowship and the Bilsland Dissertation Fellowship from Purdue University. He has been serving as a Primary Reviewer for multiple reputed journals and conferences, including IEEE International Conference on VLSI Design (VLSID), IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II: EXPRESS BRIEFS (TCAS-II), and *Wireless Personal Communications* (WPC) (Springer).

Jose Maria Bermudo Mera received the bachelor's and M.Sc. degrees in telecommunication engineering from the Technical University of Madrid, Madrid, Spain, in 2014 and 2016, respectively, and the Ph.D. degree in electrical engineering from Katholieke Universiteit Leuven, Leuven, Belgium, in 2022, working with Prof.Dr.Ir. Ingrid Verbauwhede and Dr. Angshuman Karmakar. The topic of his thesis was the implementation aspects of lattice-based cryptography.

He worked as a Research Assistant at the Technical University of Munich, Munich, Germany. He joined COSIC, Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2017. Since November 2022, he has been a Cryptography Hardware Design Engineer with PQShield Ltd., Oxford, U.K.

Angshuman Karmakar received the B.E. degree in computer science and engineering from Jadavpur University, Kolkata, India, in 2010, the M.Tech. degree in computer science and engineering from IIT Kharagpur, Kharagpur, India, in 2012, and the Ph.D. degree from the Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in 2020, for his dissertation titled "Design and implementation aspects of post-quantum cryptography."

He is one of the primary designers of the post-quantum Saber KEM scheme that is one of the finalists in the NIST's post-quantum standardization procedure. He received the FWO Post-Doctoral Fellowship from the COSIC Research Group, KU Leuven. He is currently an Assistant Professor with the Department of Computer Science and Engineering, IIT Kanpur, Kanpur, India. His research interests span different aspects of lattice-based post-quantum cryptography and computation on encrypted data.

Debayan Das received the Bachelor of Electronics and Telecommunication Engineering degree from Jadavpur University, Kolkata, India, in 2015, and the M.S. and Ph.D. degrees in electrical and computer engineering from Purdue University, West Lafayette, IN, USA, in 2021.

Prior to starting his Ph.D. degree, he worked as an analog design engineer at a startup based in India. He has interned with the Security Research Lab, Intel Labs, Intel Corporation, Hillsboro, OR, USA, over the summers of 2018 and 2020. He is currently a Research Scientist with Intel Corporation. He has authored/coauthored more than 50 peer-reviewed conferences and journals, including two book chapters and two U.S. patents. His research interests include mixed-signal IC design and hardware security.

Dr. Das was a recipient of the IEEE HOST Best Student Paper Awards in 2017 and 2019, the Third Best Poster Award in IEEE HOST 2018, and the 2nd Best Demo Award in HOST 2020. In 2019, one of his papers was recognized as a Top Pick in Hardware and Embedded Security published over the span of the last six years. He was recognized as the Winner (Third Place) of the ACM ICCAD 2020 Student Research Competition (SRC). During his Ph.D. degree, he has been awarded the ECE Fellowship from 2016 to 2018, the Bilsland Dissertation Fellowship from 2020 to 2021, the SSCS Predoctoral Achievement Award in 2021, and the Outstanding Graduate Student Research Award by the College of Engineering, Purdue University, in 2021, for his outstanding overall achievements. He has been serving as a Primary Reviewer for multiple reputed journals and conferences, including JSSC, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I: REGULAR PAPERS (TCAS-I), TVLSI, TCAD, *IEEE Design & Test*, TODAES, JETCAS, TBME, IEEE ACCESS, IoTJ, DAC, *VLSI Design*, and HOST.

Santosh Ghosh received the Ph.D. degree from IIT Kharagpur, Kharagpur, India, in 2011.

He completed his post-doctoral studies at COSIC, Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium, in the area of cryptographic hardware and side-channel attacks. He is currently a Research Scientist with the Intel Labs, Hillsboro, OR, USA. He has coauthored about 65 research publications in refereed international conferences and journals with a citation H-index of 21 and 20 issued with other 51 patents filed (pending). The primary focus of his research includes: 1) design and implement cryptographic hardware microarchitecture and RTL with aggressive area, latency, and throughput constraints; multiple of them are already being deployed in high-volume Intel products; 2) investigate and develop timing, power, and EM side-channel countermeasures; and 3) collaborate with academic partners and provide cryptography and security guidance to Intel business units.

Ingrid Verbauwhede (Fellow, IEEE) is currently a Professor with the Research Group COSIC, Department of Electrical Engineering, Katholieke Universiteit Leuven (KU Leuven), Leuven, Belgium. At COSIC, she leads the Secure Embedded Systems and Hardware Group. She is a pioneer in the field of efficient and secure implementations of cryptographic algorithms on many different platforms: ASIC, FPGA, embedded, and cloud. With her research, she bridges the gaps between electronics, the mathematics of cryptography, and the security of trusted computing. Her group owns and operates an advanced electronic security evaluation laboratory.

Dr. Verbauwhede is a fellow of IACR. She was elected as a member of the Royal Flemish Academy of Belgium for Science and the Arts in 2011. She received the IEEE 2017 Computer Society Technical Achievement Award. She was a recipient of two ERC Advanced Grants: one in 2016 and a second one in 2021. She received the 2023 IEEE Donald O. Pederson Award.

Shreyas Sen (Senior Member, IEEE) received the Ph.D. degree from the School of Electrical and Computer Engineering (ECE), Georgia Institute of Technology (Georgia Tech), Atlanta, GA, USA, in 2011.

He has over five years of industry research experience at Intel Labs, Hillsboro, OR, USA, Qualcomm, San Diego, CA, USA, and Rambus, Sunnyvale, CA, USA. He is currently an Elmore Associate Professor of ECE and biomedical engineering (BME) with Purdue University, West Lafayette, IN, USA, where he is also the Director of the Center for Internet of Bodies (C-IoB). He is the inventor of the Electro-Quasistatic Human Body Communication (EQS-HBC), or Body as a Wire Technology, for which he was a recipient of the MIT Technology Review Top-10 Indian Inventor Worldwide under 35 (MIT TR35 India) Award in 2018 and the Georgia Tech 40 Under 40 Award in 2022. To commercialize this invention, he founded Ixana, and serves as the Chairperson and the CTO. His work has been covered by more than 250 news releases worldwide and invited appearance on TEDx Indianapolis, the Indian National Television CNBC TV18 Young Turks Program, the NPR subsidiary Lakeshore Public Radio, and the CyberWire Podcast. He has authored/coauthored three book chapters, and over 175 journal articles and conference papers. He has over 20 patents granted/pending. His current research interests span mixed-signal circuits/systems and electromagnetics for the Internet of Things (IoT), biomedical, and security.

Dr. Sen serves/has served as an Executive Committee Member of the IEEE Central Indiana Section and a Technical Program Committee Member of DAC, Computer and Communications Security (CCS), Custom Integrated Circuits Conference (CICC), International Microwave Symposium (IMS), Design, Automation, and Test in Europe (DATE), International Symposium on Low Power Electronics and Design (ISLPED), International Conference on Computer-Aided Design (ICCAD), International Test Conference (ITC), and *VLSI Design*, among others. He was a recipient of the NSF CAREER Award in 2020, the AFOSR Young Investigator Award in 2016, the NSF CISE CRII Award in 2017, the Intel Outstanding Researcher Award in 2020, the Google Faculty Research Award in 2017, the Purdue CoE Early Career Research Award in 2021, the Intel Labs Quality Award in 2012 for industry-wide impact on USB-C type, the Intel Ph.D. Fellowship in 2010, the IEEE Microwave Fellowship in 2008, the GSRC Margarida Jacome Best Research Award in 2007, and nine best paper awards, including IEEE CICC 2019 and 2021 and in IEEE HOST from 2017 to 2020. His work was chosen as one of the top-ten papers in the Hardware Security Field (TopPicks 2019). He serves/has served as an Associate Editor for IEEE SOLID-STATE CIRCUITS LETTERS (SSC-L), *Scientific Reports* (Nature), *Frontiers in Electronics*, and *IEEE Design & Test*.