# XCRYPT: Accelerating Lattice Based Cryptography with Memristor Crossbar Arrays

**Sarabjeet Singh**
University of Utah

**Xiong Fan**
Rutgers University

**Ananth Krishna Prasad**
University of Utah

**Lin Jia**
University of Utah

**Anirban Nag**
Uppsala University

**Rajeev Balasubramonian**
University of Utah

**Mahdi Nazm Bojnordi**
University of Utah

**Elaine Shi**
Carnegie Mellon University

***Abstract*—This paper makes a case for accelerating lattice-based post quantum cryptography with memristor-based crossbars. We map the polynomial multiplications in a representative algorithm, SABER, and show that analog dot-products can yield $1.7-32.5\times$ performance and energy efficiency improvement, compared to recent hardware proposals. We introduce several additional techniques to address the bottlenecks in this initial design. First, we show that software techniques used in SABER, that are effective on CPU platforms, are unhelpful in crossbars. Relying on simpler algorithms further improves our efficiency by $1.3-3.6\times$. Second, modular arithmetic in SABER offers an opportunity to drop most significant bits, enabling techniques that exploit a few variable precision ADCs, and yielding up to $1.8\times$ higher efficiency. Third, to further reduce ADC pressure, we propose a simple analog Shift-and-Add technique, demonstrating a $1.3-6.3\times$ improvement. Overall, XCRYPT achieve $3-15\times$ higher efficiency over the initial design and highlight the importance of algorithm-accelerator co-design.**

**THE** recent emergence of several quantum computing systems has increased the likelihood that integer factorization and discrete logarithm will be tractable in the near future, thus rendering several modern-day cryptographic primitives obsolete. This has spurred interest in alternative cryptographic primitives, termed post-quantum cryptography (PQC).

Asymmetric key encryption schemes like RSA/ECC, used in the handshake protocols of modern cloud-based infrastructures, are vulnerable to quantum attacks; these may be replaced by their PQC counterparts that have been proposed recently. Popular PQC schemes rely on large matrix and vector operations that place a significant burden on the hardware. Recent implementations on GPUs/ASICs report latencies of tens to hundreds of micro-seconds. Classic cryptographic functions can be typically executed in *micro-seconds* on modern hardware; for instance Intel QuickAssist technology executes RSA decrypt operation in 5us. The metrics for PQC therefore fall well short of the demands of modern deployments, affecting applications like cloud services and secure transactions.

In this work[1], we demonstrate a compute-in-memory based accelerator for one such scheme, SABER [1]. We explore a promising technology – analog computing in resistive memories (memristors) – as the foundation for new architectures that efficiently execute PQC algorithms. While memristors have been used before to implement computations in deep neural networks, we show that SABER offers new opportunities to further improve the efficiency of this approach, e.g., the use of modulo operations that (unlike DNNs) allows us to drop most significant bits in computations. We first design a basic accelerator targeting client and server applications of SABER in the handshake protocol, leveraging prior best practices [2]. We demonstrate software techniques like decomposing polynomial degree and smart scheduling to increase sharing of analog to digital converters (ADCs), that improves energy efficiency by 2.4-2.7$\times$ and computational efficiency by 4.7$\times$, relative to our baseline. We propose a novel technique to perform write-free in-analog shift-and-add operations using crossbars, allowing us to trade-off cell programming with ADC complexity. Overall, XCRYPT (Xbar based accelerator for post-quantum CRYPTography) achieves server deployment with decryption latency of 0.08 us and client deployment with encryption latency of 4 us, with an overall chip area of just 0.04 and 0.3 $mm^2$ respectively.

[1]An extended version of the paper is available at https://arxiv.org/abs/2302.00095.

## BACKGROUND

**SABER.** SABER [1] is a round 3 lattice-based finalist among the NIST solicited PQC algorithms. Certain computational problems on lattices, an infinite set of periodic points in an n-dimensional Euclidean space, are conjectured to have no probabilistic polynomial time algorithms, which form the basis for lattice-based cryptography. We focus on the public key encryption (PKE) aspect of SABER since it is employed in the handshake protocol. PKE consists of 3 algorithms - key generation, encryption, and decryption. KeyGen determines a matrix $\mathbf{A}$ of polynomials using a pseudo-random number generator. A secret vector $\vec{s}$ of polynomials is generated by sampling from a distribution. Public key consists of the matrix seed and the rounded product $\vec{b} = \mathbf{A}^T\vec{s}$. Encryption generates a new secret $\vec{s}'$, and adds the message $m$ (a polynomial with coefficients $\in \{0,1\}$) to the inner product of public key $\vec{b}$ and $\vec{s}'$, forming the first part of ciphertext. The second part hides the encrypting secret by rounding the product $\mathbf{A}\vec{s}'$. Decryption uses the secret key $\vec{s}$ to extract the message encoded in the ciphertext.

Overall, SABER performs 24 PolyMults, 14 PolyModulo, and 8 PolyRounding operations – each PolyMult does $256^2$ integer products and Modulo/Rounding are applied per coefficient. SABER adds a few constants in its calculations so that rounding can be replaced with simple bit shifts.

In lattice-based cryptography, PolyMult plays a key role in the overall performance. In our experiments, we observed that PolyMult kernels consume $> 90\%$ of the execution time. When implementing PolyMult, the Number Theoretic Transform (NTT) algorithm has the asymptotically fastest time complexity of $O(n \log n)$, but it requires prime modulo, which in turn leads to non-trivial complexity for modulo operations. SABER instead chooses a power-of-two modulo, which speeds up the modulo (by simply dropping the most significant bits). Since NTT is not an option, SABER uses the Toom-Cook-4 algorithm to reduce each degree-256 to 7 degree-64 PolyMults, and then further reduces them to degree-32 using the Karatsuba algorithm once. This choice (along with AVX2 support) brings the contribution of PolyMult to 57%, and SABER outperforms implementations (like Kyber) that employ a faster NTT-based multiplier. SABER demonstrates computations in tens of microseconds using AVX2 support [1]. However, modern web-services and secure transactions demand an order of magnitude lower latencies. A direct replacement with SABER would introduce delays in connection establishments and hence, user service. Given its favorable properties in terms of speed and lower hardware/ciphertext complexity, we choose SABER as the target for our hardware acceleration. Many of the techniques introduced in this work are also applicable to other lattice-based schemes that are used heavily for PQC, homomorphic encryption, etc.

■ **Memristor Crossbar Arrays.** In the past decade, industry and academia have made significant investments in resistive memory technologies, that use cell resistance value to store information. Resistive memory arrays have several advantages – very high density, non-volatility, competitive read latency/energy, and ability to perform analog computations in-memory. Such *processing-in-memory* technologies have the potential for high parallelism and low data movement.

A memristor array is implemented as a *crossbar* – a grid of cells. Sandwiched between the X- and Y-dimension wires (wordlines and bitlines) are the resistive cells. When voltages are applied across two wordlines, current is injected into each bitline, proportional to the conductance of each cell in those rows. Thus, the current in each bitline is an analog representation of the dot-product of two vectors – the voltage vector applied to the wordlines and the conductance vector preprogrammed into a column of cells. Further, the wordline voltage is broadcast to all the columns; each column performs an independent parallel dot-product on the same voltage vector, but using a different conductance vector. The basic Kirchoff's Law equation is being exploited to design an analog vector-matrix multiplication circuit that yields a vector of output bitline currents in a single step.

Recent works like ISAAC [2] have employed these analog resistive circuits as the central component in accelerators for DNNs, which can tolerate small noise in computations. Typically, a major power/area contributor to such a design is the analog-to-digital signal converter (ADC). ADCs consume significant power that grows exponentially with resolution. Managing ADC resolution and power is a key challenge in exploiting the capabilities of this emerging technology. This paper builds on the insight from these prior architectures and explores how the specific properties of cryptographic applications can better exploit this emerging technology.

## SABER ON MEMRISTOR ARRAYS

*We first adapt the ISAAC architecture [2] to create a strong memristor crossbar baseline that can execute PolyMult. Next, we make the case that SABER's PolyMult algorithm (Toom Cook-4 + Karatsuba) is not always well suited for crossbars, and identify alternative multiplication algorithms. Then, we show techniques, enabled by SABER's power-of-2 modulo, that reduce the overheads of ADCs. In the next section, we extend this design to allow shift-and-adds in analog, further lowering ADC requirements. Overall, we make the case that amenability to acceleration with emerging technology should be a strong consideration as cryptographic primitives are standardized.*

At its core, almost all lattice-based candidate schemes in NIST PQC perform modular polynomial multiplication. *Modular* here means that $p_{out} = p_1 * p_2$, where $p_{out}, p_1, p_2 \in \mathbb{Z}_q[x]/(x^n+1)$. Modular PolyMult is often represented as vector-matrix multiplication. We use a methodology very similar to that of ISAAC to map the required computations to crossbars. Since secret key $\vec{s}$ remains constant for much of the server's runtime, we designate $\vec{s}$ as the operand to be encoded in crossbar cells. We need 72 128×128 1-bit cross-

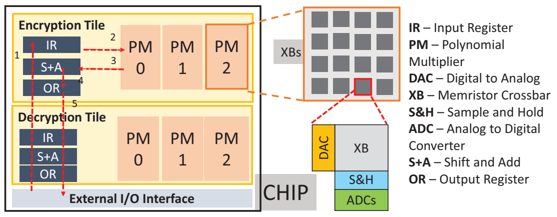


Figure 1: XCRYPT-SchoolBook or X-SB architecture, with encryption/decryption tiles.

Table 1: XCRYPT-Schoolbook (X-SB) parameters, at 32nm.

| Component | Count | Power (uW) | Area (um$^2$) |
|---|---|---|---|
| XBar (128x128) | 1 | 300 | 25 |
| DAC 1bit | 128 | 3.9 | 0.16 |
| S+H 6bit | 128 | 0.007 | 0.029 |
| ADC 6bit | 16 | 945 | 435 |
| ADC 7bit | 1 | 1365 | 628.33 |
| **Total (1 array)** | | **123.13** | **7737.557** |
| **X-SB = 1 Encryption + 1 Decryption Tile = 2 x (48 arrays)+(IR+OR+SA)** | | **11.92 mW** | **0.743 mm$^2$** |

bars to store key $\vec{s}$. A tile in our design (shown in Figure 1) has multiple crossbars and can perform 3 PolyMults, as required by a vector-vector multiplication. Using ISAAC's flip-encoding scheme, each column produces a 6-bit current. We use an area-efficient ADC that can convert samples at a frequency of 1 GSps. To further reduce area overhead, we share 1 ADC across 8 columns of a crossbar, which results in crossbar read cycle time of 8 ns. We designate this architecture as *XCRYPT-SchoolBook* or *X-SB* because it uses the basic algorithm for PolyMult.

As illustrated in Figure 1, we design 2 different tiles - encryption and decryption. This is because we target two different deployments - a cloud server and a client device. In a typical handshake protocol, the server and client establish a private key using asymmetric key encryption (like SABER in the post-quantum world). During the protocol, client encrypts using the server's public key while the server decrypts using its secret key. Hence, we design XCRYPT encryption/decryption tiles for client/server respectively.

Programming the cells is expensive, in terms of energy, performance, and endurance. Typically, memristors-based cells have a budget of $10^{12}$ writes before the cells malfunction. Since the server only periodically changes its private key (for freshness), the number of writes are low enough to sustain a reasonable lifetime. For instance, considering a conservative assumption of a private key update every second, the accelerator lifetime exceeds 30K years. On the client side, every new connection demands a new write. However, the typical number of connections established by a client is small. Even with $10^5$ connection setups per day, the client accelerator lifetime will exceed 27K years.

**■ METHODOLOGY.** To makes it convenient to discuss XCRYPT design choices with supporting results, we state our methodology here. We leverage many of the primitives introduced in the ISAAC architecture [2] and adopt an evaluation methodology very similar to that work, with parameters updated as below. The energy and area model for crossbar arrays, including Shift-and-Add Crossbars, is based on Hu et al. [3]. The memristor cell model is derived from [5], with 25 ns write latency and 0.1 pJ/cell/bit write energy, and NVSim is used to extract array level numbers. Read latency is determined by the ADC readout as RC delay of the crossbar is typically sub-ns [3]. The read energy of an memristor cell is four orders of magnitude lower than write energy. Area and energy for shift-and-add, sample-and-hold, and 1-bit DAC circuits are adapted from ISAAC [2]. We have considered an energy and area efficient adaptive ADC that can handle one giga-samples per second. ADC components have been scaled appropriately to arrive at different precision models. Detailed parameters of our initial design (X-SB, described in next subsection) are listed in Table 1. Note that we propose various versions of XCRYPT with varying XBar/ADC sizes, leading to various parameters. We also model CASCADE components based on parameters mentioned by Chou et al. [4]. We modify the SABER code to execute various XCRYPT design features. We consider the SABER variant that has post-quantum security level similar to AES-192. We consider 2 key metrics to evaluate XCRYPT efficiency: (i) Computational Efficiency (CE), the number of 1-bit plaintext/ciphertext operations per second per mm$^2$ of area $(Gbits/s \times mm^2)$, and (ii) Energy Efficiency (EE), the number of 1-bit plaintext/ciphertext operations per 1J of energy $(Gbits/J)$.

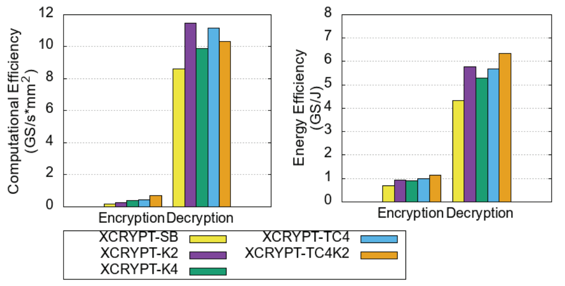


Figure 2: CE and EE for SABER on memristor crossbars, evaluating various polynomial multiplication algorithms.

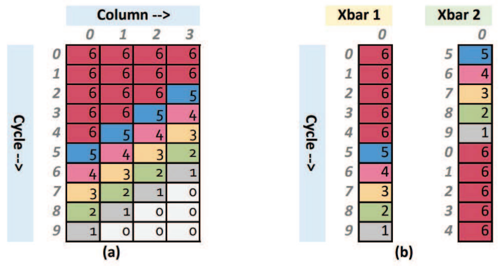


Figure 3: (a) Number of bits that contribute to the final coefficient. (b) Reordering computations in Xbars to enable sharing of high precision ADCs.

■ **Existing Works.** Efforts are already underway to implement PQC algorithms in hardware. Most of the efforts have attributed hardware overheads to multiplication units. The CPU implementations have reported latencies in 10-20 us, running on an Intel Haswell machine at 3.1 GHz, with AVX2 support [1]. Zhu et al. [6] use an 8-level hierarchical Karatsuba framework for PolyMult and a task scheduler that reduces resource utilization by up to 90%. Their post-layout chip is approximately the same size as our proposed designs, which gives a fair comparison. *While our basic X-SB design outperforms most existing works, state-of-the-art ASIC [6] achieves 2.4× better energy efficiency. In this paper, we then introduce additional techniques that enable XCRYPT to outperform [6] by 2 orders of magnitude.*

■ **Impact of PolyMult Algorithm.** In this sub-section, we explore different multiplication algorithms to identify the implementation that is most amenable to crossbar acceleration. Similar to matrix multiplication, polynomial multiplication also benefits from various lower complexity algorithms. Figure 2 quantifies CE and EE for these algorithms. We start with the standard $O(n^2)$-complexity Schoolbook algorithm, designated *X-SB*, for multiplying 256-degree polynomials. Karatsuba's algorithm ($O(n^{1.58})$-complexity) breaks down a 256-degree multiplication to 3 128-degree PolyMults, reducing the number of required 128×128 crossbars from 32 to 24, labeled *X-K2*. This proportionally decreases ADC and crossbar write energy. Encryption also has a lower CE than decryption as it requires more crossbars, has more input cycles, and requires a long initial crossbar programming step (90% of end-to-end encryption latency). We experiment with further reducing the polynomial degree to 128 using Karatsuba, labeled *X-K4*. We see that X-K4 results in lower improvements

ToomCook-4 ($O(n^{1.4})$-complexity) reduces a 256-degree PolyMult to 7 64-degree PolyMults. While ToomCook-4 is asymptotically faster than K2, it creates more polynomials that require increased ADC samples and crossbar requirements, which explains the worse behavior for *X-TC4*, relative to *X-K2*, for decryption. However, TC4 performs well during encryption as the benefits from a smaller crossbar (lower write latency and energy) outweigh the higher crossbar count requirements. Software implementations of SABER use ToomCook-4 + Karatsuba-2 (labeled as *X-TC4K2* in the figures) to do 21 32-degree PolyMults, which performs best for encryption due to its small crossbar write latency/energy. For encryption, X-TC4K2 improves CE and EE over X-SB by 3.6× and 1.6×, respectively. Lowering degree beyond 128 doesn't result in better efficiency for decryption where no writes happen.

*Thus, the ideal SABER algorithm on a crossbar accelerator varies, and we choose X-K2 for decryption and X-TC4K2 for encryption, for the rest of the paper. This analysis highlights the importance of algorithm-accelerator co-design.*

■ **Using Modulo to Reduce ADC Overheads** The *X-K2* implementation of SABER on memristor crossbars is primarily constrained by ADC overhead. ADC consumes 90% of the area, and 78% of the energy during decryption. Similar

ADC overheads are also reported by previous studies for DNN applications. In the 128×128 crossbar, since the computations are spread across cells in a row and across cycles, the 6-bit ADC results have to be aggregated after appropriate shifts. SABER's parameters define that the coefficients of the output polynomial must go through modulo $2^{10}$ or $2^{13}$, which keeps the coefficients at 10 or 13 bits. Thus, given the modulo operation, not all bits from all 6-bit ADC samples contribute to the final output.

*Unlike DNN computations, where most significant bits carry the most information, most significant bits in our computations are often ineffectual.*

For instance, all 6 bits from cycle 0's Least Significant Bit (LSB) column are needed as they add to the LSB 6 bits of the output coefficient. However, the output of cycle 9's LSB column is added to the same coefficient after shifting left 9 times, i.e., only the LSB of the sampled ADC value will contribute to the final 10-bit result. In Figure 3(a), we illustrate the number of relevant bits during decryption. Each coefficient of secret $\vec{s}$ is 4 bits, which is stored across 4 cells in a row, with LSB stored in column 0. Therefore, the outputs of 4 columns have to be added after appropriate shifts. Furthermore, values across cycles are also shifted-and-added. This is shown in Algorithm 1. As seen from the figure, *the number of bits and hence, the ADC precision varies depending upon the value of (cycle+column).* For vector-vector multiplications, we require full precision (6 bits) only for (cycle+column) $\leq 4$.

**Algorithm 1:** Pseudo-code for polynomial multiplication, per coefficient, using crossbar

```
1 coeff = 0;
2 for(cycle=0; cycle<10; ++cycle)
3     for(column=0; column<4; ++column)
4         coeff +=
            bitline_value[cycle][column]≪(cycle+column);
5 coeff = (coeff)  mod $2^{10}$
```

*We take advantage of this flexibility by reordering computations in such a manner that at any given cycle, at most 1 crossbar is producing an output with maximum precision of 6.*

This is illustrated in Figure 3(b), which depicts the column 0 output precision requirements for 2 crossbars. Computations of the second crossbar are reordered such that the $5^{th}$ gets executed in the $0^{th}$ cycle. This staggering ensures that only one crossbar produces a 6-bit output in a given cycle. By sharing 2 ADCs, one 6-bit and the other 5-bit among the crossbars, we lower the ADC overheads relative to the baseline with two 6-bit ADCs. Each ADC handles half the workload. This concept can be further applied to lower energy while slightly increasing area. The 5-bit ADC workload can be split across a 5-bit ADC and a 4-bit ADC. The 4-bit ADC handles nearly 90% of this split workload, thus saving energy. The area overhead of an extra ADC is reduced by sharing the 5-bit ADC across 10 crossbars (since the 5-bit ADC is assigned a small fraction of the conversions). Since ADC overheads grow exponentially with its precision, this technique of leveraging shared, lower precision ADCs allows us to improve EE by 1.8× and CE by 1.5× over X-K2 in decryption. Since this technique doesn't reduce the number of crossbar writes, benefits in encryption are lower - 1.7× in EE and 1.08× in CE.

## Shift-and-Add Crossbars (SACs)

PolyMult with crossbars generates many intermediate ADC readout values, that are later shifted-and-added to obtain the final output polynomial. For instance, each PolyMult in decryption generates 4 values per cycle, for 10 cycles, which are appropriately shifted-and-added to obtain one output coefficient value (Algorithm 1). Intermediate analog value readouts are expensive since they are done using ADCs.

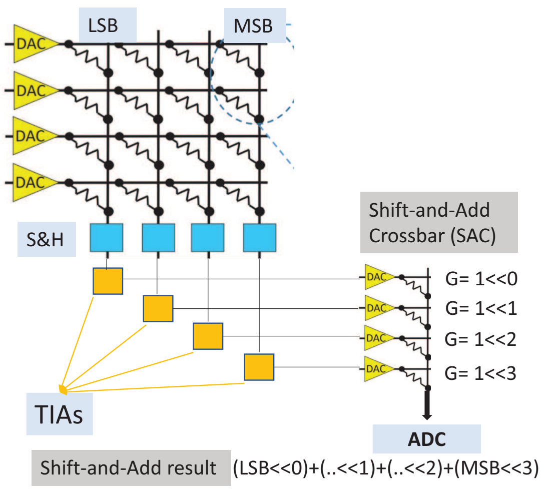
*In this section, we explore the possibility of performing the shift-and-add operation in analog to delay the ADC readout.*

We achieve this functionality by forwarding the output currents of bitlines in a crossbar to a specialized crossbar whose cells are pre-programmed to hold powers of 2, resulting in a shifted addition of inputs, and fewer ADC readouts.

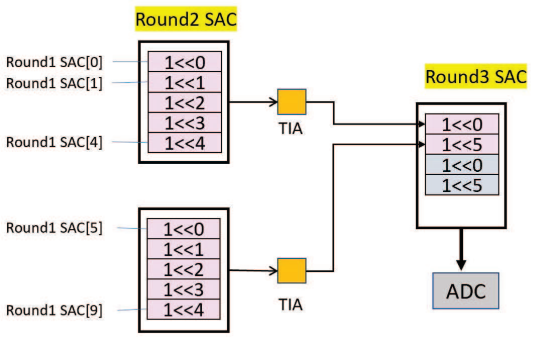
**■ Existing In-analog Shift-and-Add Implementation.** CASCADE [4] proposed in-analog shift-and-add of intermediate values by writing the output of a crossbar's columns to another Buffer crossbar. In a given cycle, the column outputs are written in adjacent cells of a row, while values across cycles are written to different rows with appropriate shifts. A simple readout of the Buffer crossbar performs shift-and-add of all intermediate values, and delays the ADC readout to a single final value. However, CASCADE assumes a lower memristor cell write energy than that reported in recent literature. While multiple write drivers allow parallel cell programming, the write latency is expected to be about 25 ns. Memristor cells also have 4 orders of magnitude higher write energy (0.1 pJ/cell/bit) than read energy [5]. We next discuss an alternative, more efficient analog shift-and-add technique.

**■ Write-Free In-Analog Shift-and-Add using SACs.** We propose a novel technique to perform shift-and-add in analog. We introduce Shift-and-Add Crossbars (SACs) - tiny single column crossbars whose cells are pre-programmed to hold successive powers of 2. The intuition behind SAC is that given an input vector, when passed through SAC, individual values are multiplied by cell values and aggregated. In practice, up to 6-bit precision memristor cells have been demonstrated. Therefore, SACs, with highest multiplier factor of $1 \ll 5$, can only add inputs with a maximum of 5 shifts. However, this can be overcome with hierarchical deployment of SACs.

We first start by using SACs within a cycle. Since the secret $\vec{s}$ is written to 4 1-bit cells in a row, the output of 4 columns must be added with appropriate shifts, every cycle. In the baseline implementation, this addition happens in digital, after ADC readout. We propose using a single SAC to perform these shift-and-adds, as demonstrated in Figure 4a. In order to feed a crossbar's output current to SAC's DAC, it must be first converted to a proportional voltage signal, which is usually done using TransImpedance Amplifiers (TIAs). We use a fast (11 ns sense+transfer time) TIA circuit proposed by CASCADE [4]. Note that SAC's output is the shift-and-add result of 4 6-bit values, resulting in a 10-bit value. Therefore, a more expensive 10-bit precision ADC is required.



(a) Shift-and-Add Crossbars (SACs)



(b) Hierarchical-SAC

Figure 4: In-analog Shift-and-Add across outputs of (a) a single crossbar, (b) different crossbars.

However, by delaying ADC readout, it converts 4× fewer samples and allows more sharing within the crossbar. Moreover, as the cycles proceed, fewer bits from the accumulated value contribute to the final output coefficient, as described in Algorithm 1. This enables flexibility to increase ADC sharing, as only a single 10-bit sample is generated across the whole dot product. On the other hand, in a non-SAC implementation, 4× 6-bit samples are generated during many cycles, as seen from Figure 3. SAC significantly lowers the number of samples, increasing the effectiveness of the ADC sharing and smart scheduling described in the previous section. We refer to this design as *SAC-Basic*.

SAC-Basic is a synergistic technique that exploits known analog circuits (like TIA), the flexibility offered by SABER's modulo operation, and resource sharing through smart scheduling on crossbars.

**IN-ANALOG ACCUMULATION ACROSS CYCLES.** In the previous subsection, we demonstrated accumulating column outputs within a cycle in analog, delaying the ADC readout. Since input values are also streamed 1-bit per cycle, outputs across different cycles must be shifted-and-added once in digital. Since our design does not write the cycle output to a crossbar (like CASCADE), it cannot readily accumulate across cycles. However, we can overcome this roadblock by doing multiple cycles in parallel.

**Accumulate across cycles:** SAC requires inputs to be fed simultaneously across different rows in order to shift-and-add them. Therefore, to add outputs from 2 cycles, their computations need to be done in parallel. This requires 2× compute/storage resources but reduces ADC samples by 2× – a reasonable trade-off as ADC accounts for a majority of the energy/area.

We label the per-crossbar single-cycle SAC as *Round1-SAC*. Unlike SAC-Basic, Round1-SAC's outputs are redirected to another SAC using TIA. This SAC is shared across multiple crossbars that contribute to the same output coefficient, and is termed as *Round2-SAC*. This 2-cycle accumulated column value is finally read out with an ADC at the end of Round2-SAC, labeled *SAC-2x*. While SAC-2x reduces ADC overheads, it increases the number of crossbars that run in parallel, in turn increasing the overall crossbar write costs. We perform a design space exploration evaluating this trade-off while increasing the number of cycles that are accumulated in-analog. At the extreme end, SAC-All accumulates all columns over all cycles per output polynomial coefficient (similar to CASCADE) and hence generates only 1 ADC sample per output coefficient.

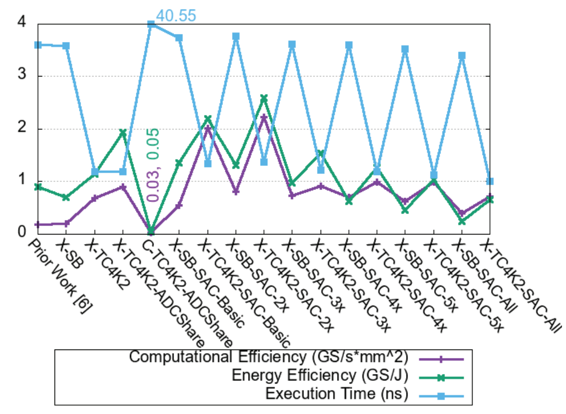
**Hierarchical SACs:** Since the memristor cell has a max resolution of 6 bits, larger computations are performed hierarchically, thus extending to Round3, 4 SACs, as shown in Figure 4b.

**RESULTS. SAC-Basic Results:** We compare our basic XCRYPT design with ADC Sharing techniques, CASCADE with similar techniques, and our novel design with SAC adding column values (*SAC-Basic*) within a cycle in Figure 5. Cycle time is determined by the TIA's sense+transfer time (=11ns). In a cycle, inputs are streamed to the first crossbar, sensed by TIA, and then sent as inputs to SAC. In the second cycle, outputs are streamed through SAC, producing the final value, which is sampled using ADC. Due to increase in the number of cycles and cycle time, end to end latency increases, relative to basic XCRYPT. However, by delaying ADC readout, SAC-Basic achieves 2.1× (2.3× for encryption) higher CE and 1.1× (1.1×) higher EE, over X-K2-ADCShare (X-TC4K2-ADCShare). On the other hand, CASCADE (*C-K2-ADCShare*) performs worse than the basic design, highlighting the overheads of performing crossbar writes every cycle.

**Parallel SAC results:** We also compare various SAC-* designs in Figure 5. As we increase the parallelism to reduce ADC samples, the bottleneck shifts from ADCs to crossbar writes. We demonstrate this using energy breakdown results in Figure 6. From SAC-Basic to SAC-All, the bottleneck shifts from ADC (77% of total energy) to crossbar writes (83% of total energy). The benefits from fewer ADC samples cannot outweigh the energy overheads of more crossbar writes, which is why SAC-All's EE decreased by 66% from X-TC4K2-ADCShare, for encryption. Meanwhile, since decryption doesn't need to write to the crossbar (after being written once on boot), increasing the level of parallelism increases CE and EE. We also observe that *the use of these circuit techniques changes the software algorithm that is most amenable to acceleration*, e.g., Schoolbook out-performs Karatsuba in some cases.

Overall, the best decryption design (*X-SB-SAC-All*) yields 4.5× and 6.3× increase in CE and EE, respectively, over X-K2-ADCShare. For encryption, these improvements for *X-TC4K2-SAC-2x* are 2.5× and 1.3× over X-TC4K2-ADCShare. Compared to state-of-the-art ASIC [6], XCRYPT shows 3-51× higher efficiencies with 2.6-16× speedup. A single tile can perform $0.7M/22.7M$ encryptions/decryptions per second with a $0.09/0.07$ $mm^2$ area budget.

Due to non-ideal device behaviors and circuit issues, analog computations are vulnerable to errors. We simulated with cell's non-ideality variance ranging up to 10%, and observed no


(a) Encryption

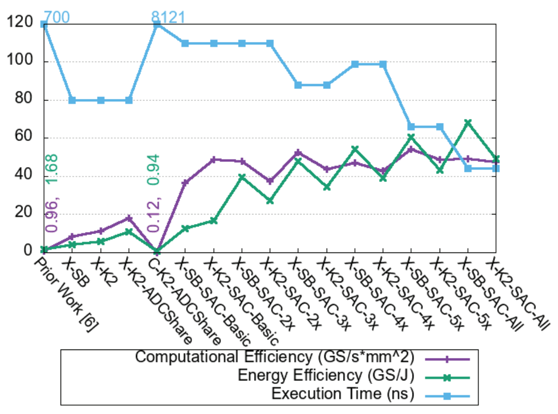
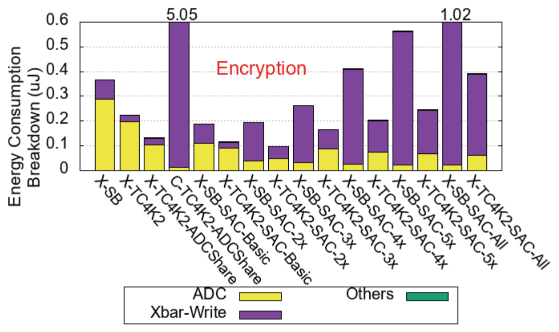

(b) Decryption

Figure 5: Comparison of XCRYPT designs. *X-*, *C-* represents XCRYPT, CASCADE respectively.


(a) Encryption

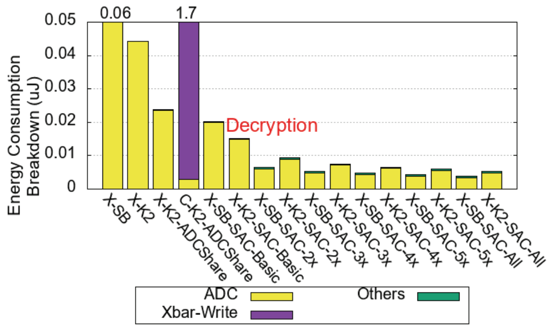

(b) Decryption

Figure 6: Energy Breakdown of various XCRYPT designs. *X-*, *C-* represents XCRYPT, CASCADE respectively.

change in failure probability till 6% variance. Beyond that, failures in decryption are detected with ECC/CRC and fixed with re-transmissions. For commercial reality, such circuits will have to show high yield, reliability and low cost in the manufacturing processes.

## CONCLUSION

This work evaluates the use of memristor crossbars for accelerating lattice-based PQC. We show that even a simple implementation of SABER, a PQC candidate for NIST Round-3, performs up to 3.4× faster than existing hardware proposals for SABER. By exploiting SABER's algorithmic properties, e.g., its power-of-2 modulo operations, we can further boost the accelerator's efficiency. We identify polynomial multiplication as the key operation in lattice-based schemes, and show that crossbar-based designs might not benefit from some of the existing software techniques for efficient multiplication. We propose SABER-specific variable precision ADCs, which, along with computation reordering, allow high levels of ADC sharing. We observe that ADC sharing improves computational and energy efficiency by $1.3-1.8\times$. To further reduce ADC overheads, we propose simple analog Shift-and-Add techniques, which offer an additional $1.3-6.3\times$ increase in efficiency.

## REFERENCES

1. D'Anvers, Jan-Pieter, et al. "Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM." International Conference on Cryptology in Africa. Springer, Cham, 2018.
2. Shafiee, Ali, et al. "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in cross-

bars." ACM SIGARCH Computer Architecture News (2016).

3. Hu, Miao, et al. "Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication." 2016 ACM/EDAC/IEEE design automation conference (DAC).
4. Chou, Teyuh, et al. "Cascade: Connecting RRAMs to extend analog dataflow in an end-to-end in-memory processing paradigm." Proceedings of IEEE/ACM International Symposium on Microarchitecture. 2019.
5. Zahoor, Furqan, Tun Zainal Azni Zulkifli, and Farooq Ahmad Khanday. "Resistive random access memory (RRAM): an overview of materials, switching mechanism, performance, multilevel cell (MLC) storage, modeling, and applications." Nanoscale research letters (2020).
6. Zhu, Yihong, et al. "Lwrpro: An energy-efficient configurable crypto-processor for module-lwr." IEEE Transactions on Circuits and Systems I: Regular Papers (2021).