


# Preventing a Crypto-Apocalypse – From Mathematics to Circuits for Post-Quantum Cryptography

Sujoy Sinha Roy<sup>1</sup> and Utsav Banerjee<sup>2</sup>

<sup>1</sup>Graz University of Technology, Graz, Austria

<sup>2</sup>Indian Institute of Science, Bengaluru, India

Communication over the internet is an integral part of the connected world. As the internet is a public network, the data packets exchanged by the communicating parties pass through various insecure channels and untrusted servers before they reach their destinations. Still, we feel it is safe to send emails, visit social media websites, watch movies of our own preferences online, shop online using credit cards. Cryptography or encryption makes it possible to protect our private information in the presence of third parties. When we browse a secured website, we see a lock symbol  on the browser. It means our communication with the website is encrypted and hence no third party will be able to read the data packets exchanged between the browser and the website. 'Hypertext Transfer Protocol Secure' or HTTPS is used to establish an encrypted hence secured communication channel between a web browser and a website. The browser-website is just one example; cryptography is happening in your mobile phones, smart cards, IoT devices, and in almost all connected devices.

'Public Key Cryptography' is used in the authentication and key agreement steps during the establishment of a secured communication [1]. Here we give the basic idea of how a browser and a server, who are not familiar to each other, agree on a common key remotely over an insecure channel. Let  $p$  be a prime and  $g$  be a generator of the group modulo  $p$ . Figure 1-a shows the Diffie–Hellman key agreement where simple modular arithmetic is performed by the two communicating parties to obtain the common integer  $k = g^{ss'}$  mod  $p$ . From now on the two parties could use  $k$  as their encryption key. Diffie and Hellman were awarded the 2015 Turing Award for pioneering public key cryptography.

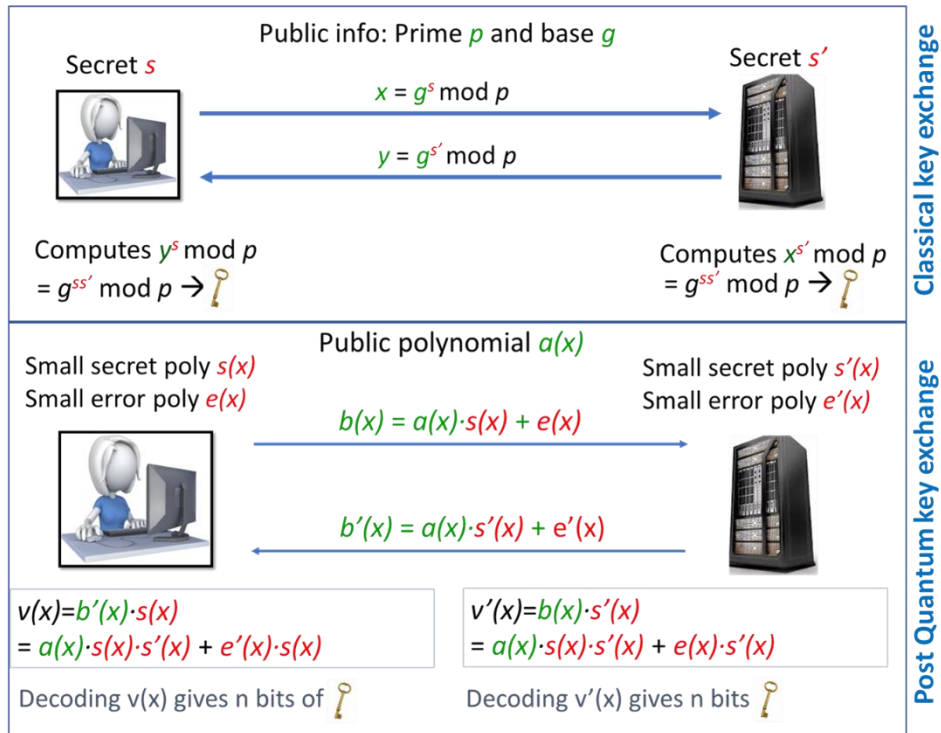


Figure 1 a and b: Key exchange protocols

What makes the Diffie–Hellman key agreement secure? The security of the Diffie–Hellman key agreement is based on the presumption that when the prime  $p$  is at least 600 digits long, given only  $g$ ,  $p$  and  $x = g^s \text{ mod } p$ , finding the secret exponent  $s$  is computationally infeasible. This mathematical problem of obtaining the secret exponent is called the discrete logarithm problem as it requires computing the logarithm  $\log_g(x)$  in the discrete group modulo  $p$ . Present-day key agreement protocols use some variants of the discrete logarithm problem as their mathematical foundation. Similar to the discrete logarithm problem, another widely used mathematical problem for public key cryptography is the integer factorization where you are asked to factorize a large integer  $N$  which is a product of two secret primes. The integer factorization problem is used as the foundation of the famous RSA cryptosystem. The short form 'RSA' originates from the names of its inventors Rivest, Shamir, and Adleman, the recipients of the 2002 Turing Award.

Here comes a twist to the story. A significantly powerful quantum computer could easily solve these mathematical problems using Shor's algorithm and therefore break the backbone of public key cryptography [2]. A quantum computer can perform many computations simultaneously unlike classical computers. Quantum computers will bring revolutions in artificial intelligence, financial modeling, drug developments, weather and climate forecasting, traffic optimization, etc. At the same time, they will completely break the present-day public key cryptographic algorithms. Now you may ask whether such a magical quantum computer exists or not? In the last several years small-scale quantum computers have been constructed independently by Google, IBM, Intel, as well as public-funded research labs. The latest quantum computing records belong to Zuchongzi and Jiuzhang 2.0. Many quantum computing experts anticipate that sufficiently powerful quantum computers to break present-day public key cryptography will become available within the next 10 to 15 years. If a powerful quantum computer becomes available, will public key cryptography be obsolete?

To prevent a quantum apocalypse, researchers around the globe have been researching new ways of constructing public key algorithms known as Post-Quantum Cryptography (PQC). They use different mathematical problems unsolvable by quantum as well as classical computers. PQC algorithms are classified into five major categories: lattice-based, multivariate polynomial-based, hash-based, code-based, and supersingular isogeny-based. In 2017, the American standardization body NIST initiated a project to standardize PQC algorithms and in July 2022 they announced the first set of algorithms for standardization. In this tutorial we will focus on lattice-based PQC due to their conceptual simplicity and wide range of applications. We remark that each mathematical class has its own beauty and advantages (and some limitations). Lattice-based cryptographic schemes use computationally infeasible problems from the lattice theory. One such problem is the Learning with Errors (LWE) problem which has seen applications in constructing many post-quantum signatures and key agreement schemes as well as advanced protocols such as fully homomorphic encryption, functional encryption, zero knowledge proofs, etc [3]. So, what is this LWE problem?

In the following, we present a simple introduction to the LWE problem and then discuss its application to the post-quantum key agreement. Let's consider a system of linear equations using commonly used matrix-vector notation e.g.,  $[2X + 3Y; 3X + 5Y] = [8; 13]$ . The solution to this system of linear equations will be  $X=1$  and  $Y=2$ . The system has a solution as the number of unknowns (i.e.,  $X$  and  $Y$ ) is two and the number of independent equations is also two. As long as the number of unknown variables is equal or smaller than the number of equations, we can compute the solution by hand calculations for small dimensions or using a Gaussian elimination program for larger dimensions. So, solving such well-defined system of linear equations is easy. Now consider the system of equations  $[2X + 3Y + e_1; 3X + 5Y + e_2] = [9; 12] \text{ mod } 15$  where an unknown small error is added to each linear equation. You are still able to find a solution e.g.,  $X=1$  and  $Y=2$  with  $e_1 = 1$  and  $e_2 = -1$  but it requires more effort. Computing a solution becomes infeasibly hard when the number of unknown variables  $X, Y, Z, \dots$  excluding the error terms is around 512 or larger. Even a very large quantum computer cannot find a solution! This mathematical problem of finding the secret vector  $[s]$  from a given matrix  $[A]$  and vector  $[b]$  where  $[b] = [A] \cdot [s] + [e]$  is known as the learning with errors (LWE) as we have to learn or solve the unknown variables  $X, Y, Z, \dots$  in the presence of unknown error vector  $[e]$ . Solving an LWE instance with around 512 unknowns will require a time more than the age of the universe. There are several variants of the LWE problem for achieving performance, communication bandwidth, and security trade-offs. One such variant is the 'Ring-LWE' problem where all matrices and vectors are replaced by polynomials i.e.,  $b(x) = a(x) * s(x) + e(x)$ . Another variant is called the 'module LWE' which is a hybrid of original LWE and ring-LWE. In module-LWE there are matrices and vectors of polynomials.

From the last paragraph you got an idea of the LWE problem. How could you use the an LWE problem (say ring-LWE) to replace the discrete logarithm-based Diffie–Hellman key agreement of Figure 1-a? For a key agreement, what you need is some kind of commutative mathematics that the two parties can compute for deriving a common number which is the source of the session key. Figure 1-b shows how the browser and client use the LWE problem as the foundation and perform simple polynomial arithmetic for agreeing on the session key. At the completion of the protocol of Figure 1-b, the two polynomials  $v(x)$  and  $v'(x)$  differ by  $|e'(x)s(x) - e(x)s'(x)|$  which can be considers as a small 'noise' because  $e(x)$ ,  $e'(x)$ ,  $s(x)$ , and  $s'(x)$  are all small-coefficient polynomials, where as  $a(x)$  is a large-coefficient polynomial. To get rid of the noise, the two communicating parties use a simple comparison-based decoding and obtain the same key  $K$ .

Modern lattice-based key agreement algorithms e.g., NewHope, Frodo, Kyber, Saber, etc., essentially use the same concept of Figure 1-b.

In the above we got a brief introduction to the post-quantum cryptography, mathematics of LWE problem and lattice-based post-quantum key agreement. Now on, we discuss ‘cryptographic engineering’ of the LWE-based public key cryptography targeting hardware platforms. So, how to implement the key agreement scheme of Figure 1-b efficiently in hardware? The fundamental building blocks will be polynomial arithmetic (addition, subtraction and multiplication) and a sampler that generates the coefficients of the error polynomial/vector  $\mathbf{e}$ . In the following we study how to implement these building blocks in hardware.

In present-day lattice-based key agreement schemes, the error terms are sampled from a discrete sub-Gaussian distribution, typically a binomial distribution. For a targeted security parameter  $\mu$ , a binomial error sampler takes two random bit-strings  $r_1$  and  $r_2$  of length  $\mu$  bits (generally 2 to 5 bits) as inputs and then subtracts their Hamming weights i.e.,  $\text{HW}(r_1) - \text{HW}(r_2)$  to obtain the sample value as the output error-coefficient. On hardware platforms, we can compute the Hamming weight of a small number simply by adding its bits. Therefore, the circuit for a binomial sampler is easy to implement in hardware.

Polynomial arithmetic is relatively more complex than the sampling of errors values. It involves not just computational considerations but also memory bandwidth limitations. Due to security reasons, polynomials for post-quantum key agreement or signature are of generally of 512 to 1024 coefficients where each coefficient is close to 16-bits long. An efficient hardware implementation requires memory elements and processing cores to work in tandem – each will have an influence on the design decisions for the other. So, how to store several polynomials inside a cryptoprocessor? On-chip memory can be implemented as a register file or as an addressable SRAM. A register is made up of a chain of flip-flops where each flip-flop stores one bit. You can read/write these flip-flops in parallel. For example, you can store an entire polynomial of 512 coefficients in a register consisting of  $16 \times 512$  flip-flops, so that you can read or write all coefficients in parallel. That is great from a data-bandwidth perspective especially when you want to implement a high-performance cryptoprocessor. However, registers are quite expensive area-wise and furthermore updating an entire register file in parallel will cause a major increase in power consumption. On the other hand, an SRAM is an addressable memory, somewhat similar to an array in the C language – so, you can access only one or two words every cycle depending on the number of read/write ports. For storing large polynomials, SRAMs are much cheaper than registers and also low power. Hence, many hardware implementations of lattice-based post-quantum cryptography use SRAMs (or BRAMs in FPGAs) to store polynomials. Indeed, the design choice whether to use a register file or an SRAM, greatly depends on the application constraints namely, speed, area and power. It is also influenced by the choice of polynomial multiplication algorithm.

Now the big question is: how to multiply two polynomials efficiently? The conceptually simplest method of multiplying two polynomials is the school-book approach that we learnt in high school. It has a quadratic time complexity as you need to perform  $n^2$  coefficient-multiplications for polynomials having  $n$  coefficients. For a 512-coefficient polynomial, the school-book approach will require around 262 thousand elementary multiplications -- quite a slow performance! A computationally more efficient approach was proposed by Karatsuba. In 1960, during a seminar at Moscow State University, Prof. Andrey Kolmogorov conjectured that multiplying two integers has  $O(n^2)$  complexity. Anatoly Karatsuba, then a 23-year-old student, attended the seminar and after a week came up with a divide-and-conquer algorithm for

multiplying two integers in  $O(n^{\log_2 3})$  complexity. As an integer is essentially a binary polynomial, Karatsuba's approach can be used to multiply polynomials in  $O(n^{\log_2 3})$  complexity. The input polynomial  $a(x)$  of size  $n$  coefficients is split into two half-sized polynomials  $a_H$  and  $a_L$  as  $a(x) = a_H x^{n/2} + a_L$ . Similar splitting happens to the other input  $b(x)$ . The multiplication is performed as  $a(x)*b(x) = a_H*b_H x^n + [(a_H + a_L)*(b_H + b_L) - a_H*b_H - a_L*b_L] x^{n/2} + a_L*b_L$  by computing only three small multiplications plus several extra additions and subtractions. On the contrary, the school-book method would require four multiplications. You can apply Karatsuba's trick recursively on the smaller operands until the operand polynomials become sufficiently small to perform direct polynomial multiplications. When to stop this recursion is a design choice and it depends a lot on the type of digital platform. Generally, implementing recursion in software is simply recursive function calls. However, managing deep recursion in hardware would require mimicking a software-styled execution of subroutines and therefore manual memory management. A recommended paper on this is [4].

In the last paragraph, you got an idea of algorithmic optimization and implementation challenges for polynomial multiplication. Now we describe the FFT method of multiplying polynomials and an example hardware architecture. Fast Fourier Transform or FFT is one of the most widely used algorithms in signal processing that does not need an introduction. It is used to transform data from the time domain to the frequency domain. The beautiful thing is that, in the frequency domain polynomial multiplication is a coefficient-wise hence  $O(n)$  complexity operation. The FFT method of polynomial multiplication applies the forward FFT transformation to the two input polynomials, then multiplies them coefficient-wise in the frequency domain, and finally applies the inverse FFT transform to bring the result of the polynomial multiplication to the time domain. As the time-frequency domain transformations have  $O(n \log n)$  complexity, the FFT method of polynomial multiplication has  $O(n \log n)$  complexity too – which is better than the Karatsuba's  $O(n^{\log_2 3})$  complexity. Does that mean we should use the FFT method for implementing the best polynomial multiplication? The answer depends on the size of polynomials, digital platform, and finally the implementation strategy. The asymptotic superiority of the FFT method does not take the overhead of memory access into account. The problem of memory access during FFT is explained as follows.

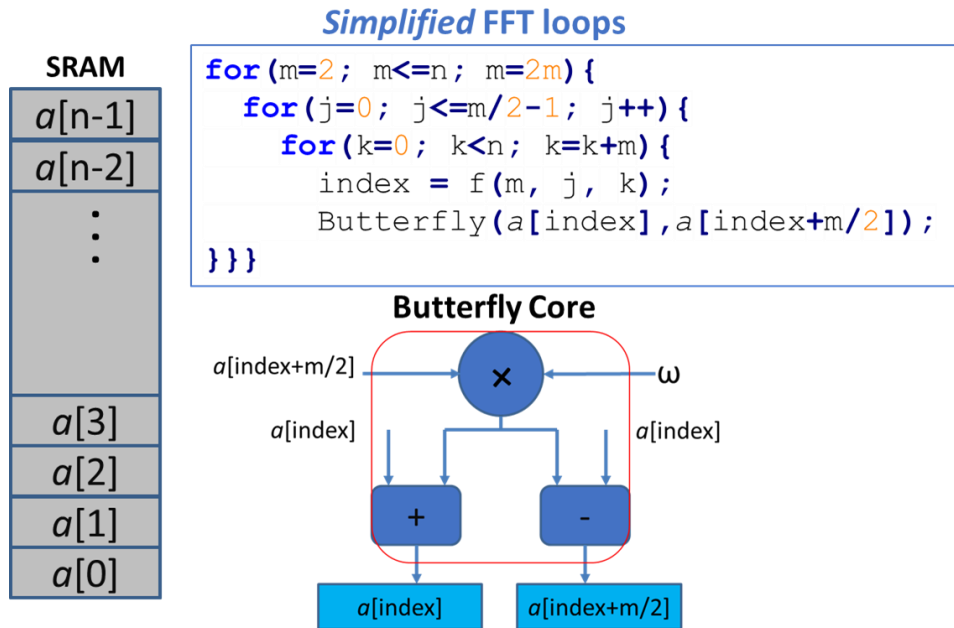


Figure 2: Understanding memory access in FFT

Figure 2 introduces a simple architecture for FFT. A polynomial  $a(x)$  is stored in the SRAM. An SRAM memory IP comes in single-port or dual-port configurations. Let's assume that the SRAM in Figure 2 is dual-port i.e., we can perform two memory accesses per cycle. The FFT has three nested loops as shown in the pseudocode in Figure 2. The Butterfly() function consists of a multiplier, adder and a subtracter. Inside the inner-most loop of FFT, it consumes two coefficients from the memory and writes two new coefficients into the memory. In hardware you can use bit-parallel circuits for implementing the butterfly core such that it computes one Butterfly() function call per cycle. This is a fair design decision as the internal arithmetic circuits are small-area because of the small coefficient size (around 16 bits). Now the memory has to meet the speed of the compute core – which means the memory must produce two coefficients per cycle to feed the butterfly core, and also consume the two coefficients per cycle. The memory access problem is that the SRAM has only two ports to support two accesses per cycle – not four per cycle. As the memory is not fast enough, the butterfly core is underutilized. How could we solve the memory access bottleneck? One simple solution will be to use two SRAM elements so that the data rate is doubled. However, in real-life applications, a crypto coprocessor is instantiated in a shared memory configuration with a host CPU such that they share the same memory e.g., SRAM. Therefore, duplicating memory IP arbitrarily may not be a resource friendly design strategy. Considering this constraint, can we find a way such that the memory access bottleneck is overcome?

We note that general-purpose processors have 32-bit or 64-bit memory word size whereas the polynomial coefficients are around 16 bits long. Wishful thinking: how about keeping two 16-bit coefficients in the same memory word such that one memory read produces two coefficients and similarly one memory write stores two coefficients produced by the butterfly into a word. If the wish comes true, the butterfly core and memory will have the same data rate and none of them will be underutilized. Additionally, the number of memory accesses will be reduced by 2x since each access deals with two coefficients. However, simply keeping two coefficients in a single word will not work out as a coefficient pair for butterfly (Figure 2) is formed of coefficients from varying addresses depending on the loop counters. To solve the problem,

a memory access-friendly FFT-based polynomial multiplication with a hardware implementation is presented in [5]. It exploits a pattern in the memory address and thereafter performs on-the-fly coefficient rearrangements to make the wish come true. The paper has a complete hardware implementation of lattice-based public key encryption.

In the above we introduced some hardware challenges and concepts for implementing the building blocks used in lattice-based PQC. Hardware platforms offer the freedom of making various design choices for meeting different area-performance-energy tradeoffs. In the context of NIST’s PQC standardization project, many algorithmic, architectural and design optimization techniques have been proposed. Secure and efficient implementation of PQC is an active research area.

Now we go a level up and see how a full post-quantum crypto coprocessor is constructed following a modular and hierarchical approach. The polynomial arithmetic components, error sampler etc., are integrated with random number generator, hash function, and dozens of additional compute blocks, finally the on-chip memory. Figure 3 shows one example of an instruction-set crypto coprocessor called ‘KaLi’ [6] for the lattice-based signature algorithm Dilithium and key agreement algorithm Kyber. Both algorithms have been selected by the NIST for standardization. The two crypto algorithms share a common polynomial arithmetic unit, a Keccak-based hash and random number generator core, and error samplers – all shown in purple in Figure 3. Algorithm specific exclusive blocks are shown in green for Dilithium and in blue for Kyber. The memory elements BRAMs are used to store polynomials as there are many of them. A host processor, in this case Microblaze, communicates with the crypto coprocessor using the AXI bus. In the TSMC 28nm node, the crypto coprocessor consumes 0.263 mm<sup>2</sup> of area, computes 13.5K signatures and 40K key agreements per second.

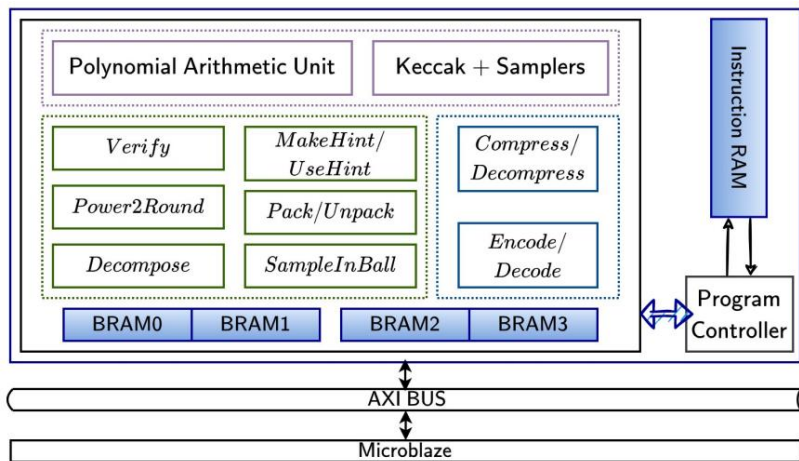


Figure 3: Organization of ‘KaLi’ crypto coprocessor for post-quantum signature and key agreement [6]

The design and implementation strategy will be different when the application scenario demands extremely low power crypto. Figure 4 shows one low-power configurable post-quantum crypto chip [7] made for constrained IoT devices. Its average power consumption is only 8mW at 1.1 V and 72 MHz in TSMC 40nm node.

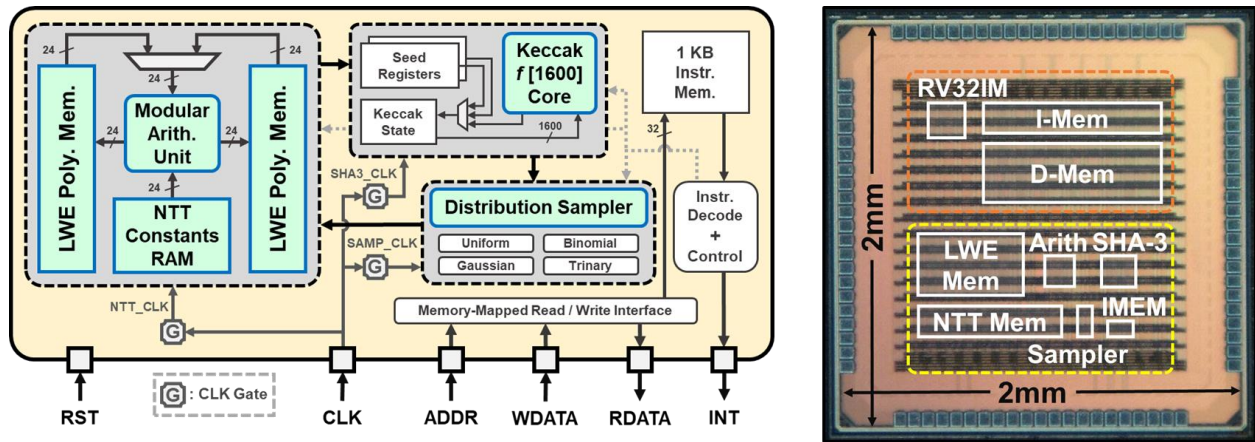


Figure 4: Low-power configurable chip ‘Sapphire’ for lattice-based post-quantum cryptography [7]

In this article, we mainly focused on efficient hardware implementations. Securing implementations of cryptography from side-channel attacks is an important task of a crypto engineer. By observing variations in the execution time, or power consumption or heat or electromagnetic (EM) emanations, an attacker can know about the internal secret key-dependent computation steps and thereafter use this ‘leakage’ to guess the secret key. Such attackers are called side-channel attacks. Most state-of-the-art hardware implementations of lattice-based crypto are not vulnerable to timing-based attacks but vulnerable to power and EM-based attacks. To make implementations resistant against such attacks, there are algorithmic as well as circuit-level solutions. Algorithmic solutions aim at randomizing the computation steps every time such that the attacker cannot relate the steps with the static secret key. Circuit-level countermeasures include designing the fundamental compute elements (transistors or gates) in such a way that the power consumption remains the same irrespective of the data being processed. Commercial crypto processor chips generally mix algorithmic and circuit level countermeasures for offering security and at the same time bring down the computational overhead of the side-channel protection.

In the conclusions part of this article, we highlight a couple of interesting research directions. Researching design methodologies for realizing PQC in extremely constrained applications is an unexplored research area with lots of challenges. Furthermore, attacking PQC implementations (SW or HW) and designing low-overhead countermeasures will be particularly interesting for new PhD students to pursue.

## References

- [1] A. J. Menezes, P. C. Van Oorschot, S. A. Vanstone: Handbook of Applied Cryptography. CRC Press 2018.
- [2] D. J. Bernstein, T. Lange: Post-Quantum Cryptography. Nature 2017.
- [3] C. Peikert: A Decade of Lattice Cryptography. Now Publishers 2016.
- [4] J. M. Bermudo Mera, F. Turan, A. Karmakar, S. Sinha Roy, I. Verbauwhede: Compact Domain-Specific Co-Processor for Accelerating Module Lattice-Based KEM. DAC 2020.
- [5] S. Sinha Roy, F. Vercauteren, N. Mentens, D. Donglong Chen, I. Verbauwhede: Compact Ring-LWE Cryptoprocessor. CHES 2014.
- [6] A. Aikata, A. C. Mert, M. Imran, S. Pagliarini, S. Sinha Roy: KaLi: A Crystal for Post-Quantum Security using Kyber and Dilithium. IEEE Transactions on Circuits and Systems-1 (2023). IACR Cryptol. ePrint Arch. 2022/1086 (2022).



[7] U. Banerjee, T. S. Ukyab, A. P. Chandrakasan: Sapphire: A Configurable Crypto-Processor for Post-Quantum Lattice-based Protocols. TCHES 2019.

## **About the authors:**

Sujoy Sinha Roy:



Sujoy Sinha Roy is an assistant professor at IAIK, Graz University of Technology. He works on secure and efficient implementation of cryptographic algorithms on hardware and software platforms. His doctoral thesis was awarded the 'IBM Innovation Award 2018' that recognizes an outstanding doctoral thesis in informatics. He is a co-designer of 'Saber' which was a finalist KEM candidate in NIST's Post-Quantum Cryptography Standardization Project.

Utsav Banerjee:



Utsav Banerjee ([utsav@iisc.ac.in](mailto:utsav@iisc.ac.in)) received his B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology (IIT) Kharagpur in 2013, and his S.M. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology (MIT) in 2017 and 2021 respectively. Since October 2021, he has been with the Indian Institute of Science (IISc), where he is currently an Assistant Professor in the Department of Electronic Systems Engineering. His research interests include cryptography, hardware security, digital circuits and embedded systems. He received the President of India Gold Medal from IIT Kharagpur in 2013, the Irwin and Joan Jacobs Presidential Fellowship from MIT in 2015, the Qualcomm Innovation Fellowship in 2016, the Pratiksha Trust Young Investigator Award from IISc in 2022 and the ABB Research Award in 2022.