PUF-Kyber: Design of a PUF-Based Kyber Architecture Benchmarked on Diverse ARM Processors

Saeed Aghapour [1], Kasra Ahmadi [1], Mila Anastasova [1], Mehran Mozaffari Kermani [2], and Reza Azarderakhsh [1]

[1]Affiliation not available
[2]University of South Florida

September 21, 2023

Abstract

In this paper, through using physical unclonable functions (PUF) and true random number generators (TRNG), we improve the overall security of CRYSTALS-Kyber and provide physical security to it. Our implementation results on ARMv7 and ARMv8 architectures indicate significant speedup, compared to the reference work.

PUF-Kyber: Design of a PUF-Based Kyber Architecture Benchmarked on Diverse ARM Processors

Saeed Aghapour, Kasra Ahmadi, Mila Anastasova, Mehran Mozaffari Kermani, *Senior Member*, *IEEE,* and Reza Azarderakhsh, *Member, IEEE*

***Abstract*—It is well-studied that quantum computing completely breaks the security of the current worldwide implemented public key cryptosystems. This forces us toward post quantum cryptography (PQC) schemes whose security remains solid even against adversaries having access to quantum computers. For this matter, National Institute of Standards and Technology (NIST) started a PQC standardization competition in 2016. After three rounds of contests, four algorithms were announced as the winners in 2022. Among them, CRYSTALS-Kyber which is the only KEM/PKE algorithm, is the aim of this paper. In this paper, through using physical unclonable functions (PUF) and true random number generators (TRNG), we improve the overall security of CRYSTALS-Kyber and provide physical security to it. Our implementation results on ARMv7 and ARMv8 architectures, indicate significant speedup, compared to the reference work. For example, for the CCA.KEM-KeyGen() algorithm, we achieved roughly 26%, 13%, and 10% speedup at security levels of 512, 768, and 1024 on ARMv7 implementation, and 25%, 12%, and 10% for ARMv8 implementation, respectively. Comparing the implementation results of our design with the reference design indicates that both the security and the system performance are improved.**

***Index Terms*—CRYSTALS-Kyber, physical unclonable functions (PUF), post quantum cryptography (PQC), true random number generator (TRNG).**

I. Introduction

Although, as of today, the existence of a practical quantum computer is a matter of debate among researchers, their advent in near future is unquestionable. If eventually, a quantum computer emerges, discrete logarithm and integer factorization problems which are the bases of the current classic public key cryptography, break in polynomial time through the infamous Shor's algorithm [1]. As a result, most of the current communications whose security is based on classic cryptography will be decrypted easily by quantum computers. Therefore, the need for fully transitioning to new cryptosystems that are secure even against quantum computing, named post quantum cryptography (PQC), is eminent. PQC schemes are categorized into 6 classes of lattice-based, code-based, multivariate-based, hash-based, symmetric methods with long length keys, and elliptic curve isogeny-based cryptography. Among these schemes, due to their high efficiency, lattice-based cryptography gained more interest than others.

S. Aghapour, K. Ahmadi, and M. Mozaffari-Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620, USA. e-mails: {aghapour, ahmadi1, mehran2}@usf.edu.

M. Anastasova and R. Azarderakhsh are with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA. e-mails: {manastasova2017, razarderakhsh}@fau.edu.

In order to facilitate the process of the transition to PQC, NIST standardization was concluded in 2022 by announcing four winner algorithms named CRYSTALS-Kyber [2], CRYSTALS-Dilithium [3], Falcon [4], and SPHINCS+ [5]. Among these four algorithms, except CRYSTALS-Kyber which is a lattice-based key encapsulation mechanism (KEM)/public key encryption (PKE) scheme, the other three are signature schemes. Now, as the competition concluded, further analysis such as resistance against physical and side-channel attacks and performance evaluation on different platforms, need to be scrutinized for these algorithms.

A. Related Work

The research in this field is mainly divided into two divisions of side-channel attack analysis and optimized implementation. Side-channel attack analysis itself divides into two categories. The first is to perform various side-channel attacks on Kyber and evaluate its results while the second category is to implement Kyber in a side-channel secure manner and not have such security as aforethought.

In [6], a configurable and side-channel resistant implementation of Kyber is introduced. This research work reported an increase of around 5% to the overhead of the original implementation. In [7], the impact of electromagnetic chosen ciphertext side-channel attack on Kyber is investigated. In [8], a side-channel message recovery attack based on deep learning on the ARM Cortex-M4 implementation of Kyber is provided. They showed that a message bit can be obtained by probability of more than 99%.

From the optimized implementation aspect, the related work can be divided into two categories of software or hardware implementations. In [9] and [10], the authors implemented the pure hardware architecture of Kyber on the AMD/Xilinx Artix-7 FPGA family and provided their results in detail. Moreover, in [11], an instruction set architecture hardware implementation of Kyber is provided on the AMD/Xilinx Artix-7 FPGA family. By optimizing Kyber components such as number theoretic transform (NTT) through various techniques like parallelization, the authors achieved significant improvement over the state-of-the-art. By utilizing hiding and masking techniques, the work in [12] presented a hardware implementation of Kyber that is secure against simple and differential power analysis side-channel attacks. In [13], the authors presented a highly area-time efficient implementation of Kyber on AMD/Xilinx Artix-7 and Zynq-UltraScale+ FPGA families. By taking advantage of pipelining and resource-reusing approaches, this work reported roughly 40% improvement in speed and 50% improvement in area and time product.

On the software implementation side, the work of [14] implemented Kyber on ARM Cortex-M4. By improving the NTT computations, they were able to improve the overall speed of the system by around 18%. In [15], a configurable ASIC processor is introduced that can handle several lattice-based algorithms such as Kyber and Dilithium for a RISC-V architecture. Furthermore, by aiming at ARMv8 architecture, the authors of [16] provided an optimized implementation of Kyber, NTRU, and Saber by using NEON instructions. This work achieved considerable speed up compared to pure C implementation. The work in [17] presented a new extension to the instruction set for RISC-V finite field arithmetic. Besides reducing code and data size, they improved the polynomial arithmetic by up to 85% compared to the reference implementation. Moreover, the authors of [18] presented the first formally verified implementation of Kyber in EasyCrypt proof assistant written in Jasmin language.

B. Major Contributions

In this paper, we endeavor to pinpoint the advantages of using PUFs and TRNGs in CRYSTALS-Kyber schemes. To the best of our knowledge, the only work that utilizes PUF in PQC schemes is [19] which mainly focuses on the management of public key infrastructure (PKI). To cover a broad range of applications, we implemented our design on ARMv7 and ARMv8 architectures and compared them with the reference work. For ARMv7, we chose ARM Cortex-M4 processor which is a low-power processor suited for embedded systems. For ARMv8, which acts as a mediator between Cortex-M4 and power-hungry platforms such as AMD64, we implemented our design on both ARM Cortex-A72 and Apple M1 processors. Our result shows that not only did we enhance the overall security of the scheme, but also the total performance of the system improved significantly.

Our contributions of this paper are summarized as follows:

1) We provide physical security to the original Kyber scheme, making it suitable for different applications like IoT or smart grid networks where the involved devices are prone to be captured physically.
2) Because of using PUF there is no need to store the long-length keys or the seeds, hence the storage burden is reduced.
3) This work also enhances the entropy of the secret keys because of the true randomness feature of PUFs and TRNGs.
4) We implemented our designs on 2 architectures and provided a detailed comparison with the original designs. Our results indicate a performance improvement in both architectures especially at lower security levels.

The remainder of the paper is structured as follows: Section II provides the preliminaries, including the original Kyber algorithms and basics of PUF and TRNG. In Section III, we propose our design and discuss its security advantages. Details of our implementation, performance evaluation, and thorough comparison are provided in Section IV. Finally, the paper is concluded in Section V.

II. Preliminaries

In this section, we provide a brief description of Kyber algorithms and basics of PUFs and TRNGs.

A. CRYSTALS-Kyber

CRYSTALS-Kyber has been introduced in 2018 and been revised and improved three times since its introduction, on last version of which we focus [20]. Kyber has a PKE and a KEM scheme. Each PKE scheme consists of three algorithms, i.e., KeyGen(), Encryption(), and Decryption(). Algorithms 1 and 2 in this section depict KeyGen() and Enc() algorithms of the Kyber CPA.PKE scheme. In these algorithms, E, D, C, DC are Encode, Decode, Compress, and Decompress functions, respectively.

By taking advantage of the Fujisaki-Okamoto transform [21], Kyber CCA.KEM scheme results directly from Kyber CPA.PKE scheme. A typical KEM scheme consists of three algorithms of KeyGen(), Encapsulation(), and Decapsulation(). Furthermore, there are two variants of CRYSTALS-Kyber scheme named Kyber and Kyber 90s which are similar in algorithms and only differ in their functions instantiation. Please refer to [20] for further details, omitted here for the sake of brevity.

B. True Random Number Generators (TRNG)

While pseudo-random number generators (PRNG) use a deterministic algorithm to create sequences of random numbers, TRNGs use the unpredictable intrinsic features of their environment (a physical process) to do that. PRNGs use a secret seed and then by performing a mathematical algorithm on it, create a pseudo-random sequence with arbitrary length deterministically. On the other hand, in TRNG there is no seed, therefore, the resulting sequences cannot be repeated. In cryptography, TRNGs are usually used in seed creation because of their high entropy, and then the seed is used in a PRNG to obtain a sequence of arbitrary length.

There are various sources to implement TRNGs in practice, such as radioactive decay, thermal noise, clock drift, photon arrival times, and the like [22]. Nonetheless, the most practical and inexpensive methods for cryptography purposes are based on delay, noise, phase jitter, chaos, and memory. Moreover, TRNGs can be implemented through FPGA components.

C. Pysical Unclonable Functions (PUF)

A typical PUF is an object that takes advantage of the unwanted inherent random variations that are created in its manufacturing processes, to create unique values [23]. For

Algorithm 1 $Kyber.CPA.PKE.KeyGen()$

```
Output: Secret key $sk \in B^{12.k.n/8}$
Output: Public key $pk \in B^{12.k.n/8+32}$
1:  $d \leftarrow B^{32}$
2:  $(\rho, \sigma) := G(d)$
3:  $N := 0$
4:  for $i = 0$ to $k-1$ do
5:      for $j = 0$ to $k-1$ do
6:          $\hat{A}[i][j] := Parse(XOF(\rho, j, i))$
7:      end for
8:  end for
9:  for $i = 0$ to $k-1$ do
10:     $s[i] := CBD_{\eta_1}(PRF(\sigma, N))$
11:     $N := N + 1$
12: end for
13: for $i = 0$ to $k-1$ do
14:     $e[i] := CBD_{\eta_1}(PRF(\sigma, N))$
15:     $N := N + 1$
16: end for
17: $\hat{s} := NTT(s)$
18: $\hat{e} := NTT(e)$
19: $\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$
20: $pk := (E_{12}(\hat{t} \bmod^{+} q) \,||\, \rho)$
21: $sk := E_{12}(\hat{s} \bmod^{+} q)$
22: return $(pk, sk)$
```

example, silicon PUFs use the differences in transistors properties, such as threshold voltage or delay, to generate a unique response. These unique, random, and unpredictable variations in each PUF are the reasons to consider them as a digital device's fingerprint. In general, PUFs are modeled as deterministic one-way mathematical functions that take a challenge as input and output a random, unpredictable, and yet repeatable response. As PUFs use physical variations to produce randomness, their outputs are considered true randoms and with some techniques, they can act as TRNG [24].

Similar to TRNG, PUFs can also be instantiated by FPGA fabric components without additional hardware. PUFs can be implemented through various methods, however the most important families of PUFs in cryptography are delay and memory based silicon PUFs. Furthermore, for evaluating PUFs' performance, several features and metrics are considered including reliability, uniqueness, uniformity, unpredictability, tamper-evident, and the like [25].

III. The Proposed PUF-KYBER Architecture

As seen in Subsection II-A, Kyber has a CPA.PKE and a CCA.KEM scheme. In this section, we target both of these schemes. In [20], it is stated that the choice of a random generator is a local decision and could be platform dependent. In original paper, PRF is instantiated with SHAKE256 and AES256 in Kyber and Kyber 90s, respectively. For our design, we instantiate PRF with a PUF and a TRNG in KeyGen() and Enc() algorithms, respectively. We divide this section into three parts. In parts A and B, we propose our new CPA.PKE and CCA.KEM designs while in part C, we discuss our gains and advantages over the original design.

Algorithm 2 $Kyber.CPA.PKE.Enc(pk, m, r)$

```
Input Public key $pk \in B^{12.k.n/8+32}$
Input Message $m \in B^{32}$
Input Random coins $r \in B^{32}$
Output: Ciphertext $c \in B^{d_u.k.n/8+d_v.n/8}$
1:  $N := 0$
2:  $\hat{t} := D_{12}(pk)$
3:  $\rho := pk + 12.k.n/8$
4:  for $i = 0$ to $k-1$ do
5:      for $j = 0$ to $k-1$ do
6:          $\hat{A}^T[i][j] := Parse(XOF(\rho, i, j))$
7:      end for
8:  end for
9:  for $i = 0$ to $k-1$ do
10:     $r[i] := CBD_{\eta_1}(PRF(r, N))$
11:     $N := N + 1$
12: end for
13: for $i = 0$ to $k-1$ do
14:     $e_1[i] := CBD_{\eta_2}(PRF(r, N))$
15:     $N := N + 1$
16: end for
17: $e_2 := CBD_{\eta_2}(PRF(r, N))$
18: $\hat{r} := NTT(r)$
19: $u := NTT^{-1}(\hat{A}^T \circ \hat{r}) + e_1$
20: $v := NTT^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + DC_q(D_1(m), 1)$
21: $c_1 := E_{d_u}(C_q(u, d_u))$
22: $c_2 := E_{d_v}(C_q(v, d_v))$
23: return $(c_1 \,||\, c_2)$
```

A. New CPA.PKE Scheme

In Kyber CPA.PKE scheme, according to Algorithm 1, d is chosen randomly (line 1). Then, this d is hashed (line 2) and the result will be used as the seed of the PRF function alongside a counter (lines 10 and 11) to create the secret key. As a result, the security of the secret key is directly dependent on the security of the random value d. Similarly, in Algorithm 2, the value r which has a direct impact on the creation of noise polynomials being responsible for the security of the ciphertext, is chosen randomly.

With these in mind, although the original paper did not mention this specifically, to have high entropy and randomness for d and r, these values should be created through a true random generator source. Our idea is to extend the application of the existing true random source to additional functionalities, to prevent introducing excessive hardware components or complexity to the design.

In our CPA.PKE.KeyGen() algorithm, we instantiate PRF with a PUF to use the reproducibility feature of PUFs and create the secret keys whenever needed without storing them. The KeyGen() algorithm of our design is provided in Algorithm 3.

For CPA.PKE.Enc(), (Algorithm 2), PRF is used several times to create noise and error polynomials r, e_1, and e_2. Unlike the secret keys, noise polynomials have one-time usage and will be discarded after the encryption process is over. Therefore, the reproducibility feature of PUFs is not required here. For this reason, in this algorithm, we instantiate PRF

Algorithm 3 *Our* $Kyber.CPA.PKE.KeyGen()$

```
Output: Secret key $sk \in B^{12.k.n/8}$
Output: Public key $pk \in B^{12.k.n/8+32}$
1:  $\rho \leftarrow B^{32}$
2:  $(a_0 || ... || a_{k-1} || b_0 || ... || b_{k-1}) := PUF(\rho)$
3:  for $i = 0$ to $k - 1$ do
4:      for $j = 0$ to $k - 1$ do
5:          $\hat{A}[i][j] := Parse(XOF(\rho, j, i))$
6:      end for
7:      $s[i] := CBD_{\eta_1}(a_i)$
8:      $e[i] := CBD_{\eta_1}(b_i)$
9:  end for
10: $\hat{s} := NTT(s)$
11: $\hat{e} := NTT(e)$
12: $\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$
13: $pk := (E_{12}(\hat{t} \bmod^{+} q) || \rho)$
14: $sk := E_{12}(\hat{s} \bmod^{+} q)$
15: return $(pk, sk)$
```

with a TRNG whose role is to create one-time true random noise polynomials with higher entropy in comparison with pseudo-random number generators. The new Enc() algorithm is proposed in Algorithm 4. Note that in Algorithm 3, a_i and b_i coefficients are $64\eta_1$ bytes, while in Algorithm 4 a_i, b_i, and c are $64\eta_1$, $64\eta_2$, and $64\eta_2$ bytes, respectively.

CPA.PKE.Dec() algorithm of our design remains unchanged and as it works independently on how PRF is instantiated, it does not affect the soundness of our design.

B. New CCA.KEM Scheme

Similar to Kyber CPA.PKE, we assume that Kyber CCA.KEM also requires some sort of true randomness in its design. The random variables in this scheme are z, m, and d. For our new CCA.KEM.KeyGen() algorithm, as it performs CPA.PKE.KeyGen(), by modifying the latter as we did in Subsection III-A (Algorithm 3), we modify the CCA.KEM.KeyGen() algorithm. However, a similar strategy is not applicable for encapsulation algorithm. With more details, as Kyber KEM scheme is created by applying FO transform on its PKE version, there is one step in the decapsulation algorithm to actively check the validity of the received message. In that step, the receiver encrypts the message himself and compare it with the received ciphertext (line 6 of Algorithm 9 in [20]). This means that the receiver must be able to successfully perform the CPA.PKE.Enc() algorithm on the message. This process is straightforward in the original paper as the PRF is instantiated with either SHAKE256 or AES256 which, in both cases, it can be done by knowing the seed. However, as in our design, Enc() algorithm is not deterministic, the receiver cannot compute the same result as the sender did. In other words, if we instantiate the PRF with TRNG, due to the unrepeatability feature of TRNGs, when the receiver encrypts the message himself, it computes a new valid ciphertext that is different from the sender's ciphertext. Additionally, if we use PUF in PRF, the issue remains, as the receiver does not have access to the sender's PUF to compute the exact ciphertext the sender had computed previously. As a result, in our CCA.KEM

Algorithm 4 *Our* $Kyber.CPA.PKE.Enc(pk, m)$

```
Input Public key $pk \in B^{12.k.n/8+32}$
Input Message $m \in B^{32}$
Output: Ciphertext $c \in B^{d_u.k.n/8+d_v.n/8}$
1:  $\hat{t} := D_{12}(pk)$
2:  $\rho := pk + 12.k.n/8$
3:  $(a_0 || ... || a_{k-1} || b_0 || ... || b_{k-1} || c) \leftarrow TRNG(.)$
4:  for $i = 0$ to $k - 1$ do
5:      for $j = 0$ to $k - 1$ do
6:          $\hat{A}^T[i][j] := Parse(XOF(\rho, i, j))$
7:      end for
8:      $r[i] := CBD_{\eta_1}(a_i)$
9:      $e_1[i] := CBD_{\eta_2}(b_i)$
10: end for
11: $e_2 := CBD_{\eta_2}(c)$
12: $\hat{r} := NTT(r)$
13: $u := NTT^{-1}(\hat{A}^T \circ \hat{r}) + e_1$
14: $v := NTT^{-1}(\hat{t}^T \circ \hat{r}) + e_2 + DC_q(D_1(m), 1)$
15: $c_1 := E_{d_u}(C_q(u, d_u))$
16: $c_2 := E_{d_v}(C_q(v, d_v))$
17: return $(c_1 || c_2)$
```

scheme, only the KeyGen() algorithm is changed, and the Enc() and Dec() algorithms remain unchanged.

C. Security Analysis

It is well established that the entropy of random sequences that are created by a TRNG source is significantly higher in comparison to those created by a PRNG source. Hence, the secret keys of our design have higher entropy and security compared to the original design.

Besides that, in the original design, the value d is hashed to create a secret seed value σ, which is then used to create the secret key. This means that either the secret key or the value d must be stored in the memory of the device. In applications where storage burden is not an issue while the computational cost is, it is better to store the whole secret key to eliminate the extra computation of the secret key from the seed. On the other hand, in applications with limited storage space, only the seed value d is stored and the secret key will be computed from that every time it is needed.

In either case, storing a secret value in memory is essential which leads to a security breach in the event of physical attacks. Physical attacks are very common in applications like IoT or Smart grid networks where the adversary can capture the users physically and access their memories. Hence, in the original design by capturing the user, even momentarily, the adversary can obtain the secret value d and compute σ, and eventually the secret key s. On the other hand, in our design, the seed value ρ is not secret and is part of the public key. Meaning that even by having ρ, the adversary cannot compute the secret key without having the PUF. The reason is that the randomness of PUF responses comes from its intrinsic physical building structure not the secrecy of its challenges. This provides physical security for our design.

Moreover, in Algorithm 1, d is hashed to obtain the values ρ and σ (line 2). As the hash function is applied on

Table I
THE RANDOM BYTES NEEDED IN DIFFERENT SECURITY LEVELS OF KYBER

	k	η_1	η_2	B_{PUF}	B_{TRNG}
Kyber512	2	3	2	768	768
Kyber768	3	2	2	768	896
Kyber1024	4	2	2	1024	1152

a secret value that the secret key depends on it, it could become the target of side-channel attacks. For instance, in [26], it is showed that the input of Keccak functions can be obtained by performing single trace soft analytical side-channel attack (SASCA). However, by eliminating this step, our design improves the security of the scheme against side-channel attacks. Furthermore, based on Algorithm 2, the value r is responsible for the creation of noise polynomials and eventually the ciphertext. If r gets leaked, the corresponding message of that communication can be obtained. However, in our design, there is no need for a random coin as a seed, and the randomness for the noise polynomials comes from a true random generator source which has much higher entropy in comparison with the original design.

In summary, compared to the reference work, our design provides the security advantages of (i) higher entropy for secret keys, (ii) physical security, and (iii) more resistance against side-channel attacks.

IV. Implementation Benchmarks and Comparison

In this section, after choosing a suitable PUF and TRNG for our design, we present the thorough details of our implementation and compare it with the original design. One of the performance advantages of our work over the original paper is omitting one hash function computation in KeyGen() algorithm. As seen in Algorithm 1, the seed value σ is created by applying the hash function G on d, while because of the intrinsic randomness of PUF, our design does not need this step, leading to lower computational cost.

As mentioned earlier, in the original design, at least the seed value d must be stored in each user's memory as a secret value. On the other hand, in our design, by having the public value ρ, secret key can be computed but only by the user possessing the specific PUF. Now, since ρ is public, there is no need for users to store it in their memories and it can be published in a public key infrastructure server. Thus, our design provides more flexibility in applications that have limited memory storage capacity.

Overall, our performance gains over the original designs are summarized as follows: (i) improving computational cost and (ii) eliminating the need for memory storage.

A. Choosing PUF and TRNG

In the original designs, the output of each PRF function is given as an input of the CBD_{η_i} function which its role is to get $64\eta_i$ bytes as input and to output a polynomial deterministically. This yields that we require $64\eta_i$ random bytes for each CBD_{η_i} call. Table I shows the exact number of required bytes for each Kyber scheme. B_{PUF} and B_{TRNG} refer to the number of needed bytes to be generated from PUF and TRNG modules, respectively.

For our specific needs, high throughput TRNG and PUF with strong security and high reliability are well-suited. From a security standpoint, when PUFs are utilized for authentication purposes, their challenge-response pairs become exposed, making them vulnerable to potential adversaries who could gather a large number of these pairs and attempt to clone the PUF using machine-learning (ML) attacks. Therefore, in such applications, it is recommended to use robust PUFs with high resistance against ML attacks. However, in our case, we solely employ the PUF for key generation and encryption algorithms, which necessitates strict non-disclosure of their outputs. This precautionary measure makes it difficult for adversaries to gather the required pairs and successfully model the PUF. As a result, even weak PUFs can be utilized in our design. Consequently, we have opted for SRAM-based designs due to their widespread presence in various devices, offering a high level of applicability.

SRAM PUFs exploit the randomness in the start-up pattern of SRAM memories in digital chips. Although, SRAM based functions are easy to implement and utilize small area, they are highly sensitive to environmental variables such as temperature change. This could cause unstable, unreliable, and different responses from the function in different environmental conditions. With that being said, since TRNG is used to create one time random numbers, reliability is not a concern there, but it is vital to obtain the same response from PUF in different environmental conditions. Therefore, in order to be used in KeyGen, a PUF must provide high reliability and robustness to environmental changes.

For these reasons, we chose [27] and [28] to be used as our TRNG and PUF, respectively. The work in [27], proposed an SRAM based TRNG, offering 100 MBps throughput on Virtex-II Pro FPGA and utilizes 369 slices. Moreover, their design successfully passes all NIST statistical randomness tests with high scores. In [28], a new PUF is proposed that offers 800 MBps on Artix-7 FPGA while utilizing only 17 slices and 1 BRAM resources. Their design, provides 98.9%, 37.5%, 55.2%, and 46.7% in average reliability, uniqueness, uniformity, and bit aliasing, respectively. To obtain their results, they tested their PUF in 3 different temperatures. The combination of high reliability and high throughput makes this PUF suitable for our design.

B. Methodology and Implementation Results

As mentioned in Section III, in practice almost all of the cryptosystems utilize some sort of true random number generators which is used for seed creation. As a result, this module can be replaced with a more efficient one to extend its usage. Furthermore, as PUF and TRNG run parallel to the software entities, in theory the overall performance of the system will be determined by the slowest part. That being said, by choosing a high performance PUF and TRNG, we can obtain their results by the time they are required by the software entities of the algorithm without causing any delay. Hence, the overall performance of the system will be bound to the software entities.

Table II
ARM CORTEX-M4 IMPLEMENTATION RESULTS BASED ON NUMBER OF CLOCK CYCLES

Frequency	Scheme	Security Level	Speed Implementation			Stack Implementation		
			CPA.PKE		CCA.KEM	CPA.PKE		CCA.KEM
			KeyGen()	Enc()	KeyGen()	KeyGen()	Enc()	KeyGen()
24 Mhz		Kyber512	**241,759**	247,531	319,813	241,539	249,257	319,560
	This work	Kyber768	496,585	501,486	612,436	498,018	519,086	613,810
		Kyber1024	847,964	851,421	1,003,037	852,192	859,755	1,019,193
		Kyber512	**356,125**	287,307	433,708	355,938	339,770	433,890
	Kyber	Kyber768	588,632	594,256	704,423	602,938	611,853	706,866
		Kyber1024	967,366	970,696	1,122,664	971,140	979,023	1,126,112
		Kyber512	**(32.1%)**	(13.8%)	**(26.2%)**	(32.1%)	(26.6%)	(26.3%)
	Speedup[1]	Kyber768	(15.6%)	(15.6%)	(13.1%)	(17.4%)	(15.1%)	(13.1%)
		Kyber1024	(12.3%)	(12.2%)	**(10.6%)**	(12.2%)	(12.1%)	(9.4%)
		Kyber512	214,004	219,354	248,918	214,644	222,348	249,207
	This work (90s)	Kyber768	434,963	439,432	479,740	437,388	445,359	487,728
		Kyber1024	734,833	732,800	798,576	744,282	752,395	809,528
		Kyber512	334,695	280,492	365,220	335,775	339,227	370,112
	Kyber (90s)	Kyber768	566,047	582,045	607,037	568,659	587,970	619,049
		Kyber1024	902,444	916,159	976,099	917,466	935,746	982,636
		Kyber512	(36.1%)	(21.7%)	(31.8%)	(36.1%)	(34.4%)	(32.6%)
	Speedup (90s)[1]	Kyber768	(23.1%)	(24.5%)	(20.9%)	(23.1%)	(24.2%)	(21.2%)
		Kyber1024	(18.5%)	(20.1%)	(18.1%)	(18.8%)	(19.5%)	(17.6%)
168 Mhz		Kyber512	264,539	266,833	349,293	264,709	269,589	349,570
	This work	Kyber768	540,268	541,679	666,221	543,062	547,629	669,240
		Kyber1024	920,072	932,513	1,089,159	926,792	928,603	1,093,898
		Kyber512	388,422	310,179	473,562	389,104	368,273	473,810
	Kyber	Kyber768	641,447	642,896	767,758	643,760	648,820	769,083
		Kyber1024	1,064,821	1,062,554	1,218,593	1,055,863	1,058,650	1,223,259
		Kyber512	(31.8%)	(13.9%)	(26.2%)	(31.9%)	(26.7%)	(26.2%)
	Speedup[1]	Kyber768	(15.7%)	(15.7%)	(13.2%)	(15.6%)	(15.5%)	(12.9%)
		Kyber1024	(13.5%)	(12.2%)	(10.6%)	(12.2%)	(12.2%)	(10.5%)
		Kyber512	260,555	263,438	300,758	261,649	266,649	300,948
	This work (90s)	Kyber768	532,248	533,107	587,093	537,585	542,102	592,124
		Kyber1024	906,978	904,148	980,638	917,580	912,584	988,457
		Kyber512	406,746	337,906	446,144	407,966	408,332	447,014
	Kyber (90s)	Kyber768	693,217	706,953	748,234	698,627	715,921	748,067
		Kyber1024	1,117,672	1,127,613	1,191,304	1,121,077	1,136,063	1,199,222
		Kyber512	(35.9%)	(22.1%)	(32.5%)	(35.8%)	(34.6%)	(32.6%)
	Speedup (90s)[1]	Kyber768	(23.2%)	(24.5%)	(21.5%)	(23.1%)	(24.2%)	(20.8%)
		Kyber1024	(18.8%)	(19.8%)	(17.6%)	(18.1%)	(19.6%)	(17.5%)

[1]Speedup $= \frac{\text{Kyber} - \text{This work}}{\text{Kyber}} \times 100$ and Speedup (90s) $= \frac{\text{Kyber (90s)} - \text{This work (90s)}}{\text{Kyber (90s)}} \times 100$.

To benchmark the software entities of our design, we implemented it on two different architectures of ARMv7 and ARMv8. For ARMv7 implementation, we took advantage of STM32F407G discovery board featuring the widely deployed Cortex-M4 processor and the pqm4 library[1]. The pqm4 library provides a framework for performance evaluation of the emerging post quantum cryptographic primitives, targeting the SMT32F407VG - Discovery Board, recommended by NIST.

The constrained specifications of the Cortex-M4 based STM32 board, the target of this work, makes the implementation of post-quantum primitives on the platform a challenging task. The device offers 192KB of SRAM, and another 1MB of flash memory. The limited register set consists of 16 general purpose registers where 2 of them are reserved and not accessible to the programmer. Additionally, the 3-stage pipeline ensures a powerful, yet limited instruction set. Because of these limitations in memory resources and computational power, efficient implementation of the heavy PQC schemes requires additional effort.

Despite the effort of different cryptographic engineering and research teams in optimizing the design of PQC schemes, a trade-off between latency and stack usage is required. That is the reason for the two different designs of the CRYSTALS-Kyber contained in the pqm4 library named stack and speed designs. Speed design ensures minimal execution time while stack design relaxes the stack usage. The main difference between them is the creation of the matrix, forming part of the public key value, and the execution flow when operating on it. While the stack optimal design generates the matrix and then operates on it, the speed implementation generates and operates in row-by-row fashion on the matrix.

Table II represents our benchmark on the ARM Cortex-M4 platform and compares it to the reference work on three security levels in two different frequencies and two different implementation designs. For example, in the speed implementation and at 512-security level, the CPA.PKE-KeyGen() algorithm of the original design takes 356,125 cycles while in our design it takes 241,759 cycles yielding in 32.1% speedup.

[1]Available at https://github.com/mupq/pqm4.

Table III
ARMv8 IMPLEMENTATION RESULTS BASED ON NUMBER OF CLOCK CYCLES

Platform	Scheme	Security Level	Pure C			NEON Instructions		
			CPA.PKE		CCA.KEM	CPA.PKE		CCA.KEM
			KeyGen()	Enc()	KeyGen()	KeyGen()	Enc()	KeyGen()
Cortex-A72	This work	Kyber512	109,331	144,699	120,223	42,834	52,851	53,604
		Kyber768	207,350	248,890	222,238	84,592	98,475	**100,423**
		Kyber1024	322,459	367,675	343,044	145,566	157,792	166,250
	Kyber	Kyber512	126,354	157,348	137,286	60,680	67,245	71,410
		Kyber768	223,107	263,740	239,033	98,528	112,951	**114,415**
		Kyber1024	344,183	387,816	364,835	163,623	176,452	184,264
	Speedup	Kyber512	(13.4%)	(8.1%)	(12.4%)	(29.4%)	(21.4%)	(24.9%)
		Kyber768	(7.1%)	(5.6%)	(7.1%)	(14.1%)	(12.8%)	**(12.2%)**
		Kyber1024	(6.3%)	(5.2%)	(5.9%)	(11.1%)	(10.5%)	(9.7%)
	This work (90s)	Kyber512	181,494	222,820	195,097			
		Kyber768	376,332	424,132	395,461			
		Kyber1024	628,563	682,470	654,490			
	Kyber (90s)	Kyber512	230,763	265,434	244,423			
		Kyber768	433,840	480,697	454,377		**N/A**[1]	
		Kyber1024	706,423	757,977	732,872			
	Speedup (90s)	Kyber512	(21.3%)	(16.1%)	(20.1%)			
		Kyber768	(13.2%)	(11.7%)	(12.9%)			
		Kyber1024	(11.1%)	(9.9%)	(10.7%)			
Apple-M1	This work	Kyber512	70,246	96,195	77,775	12,122	14,126	17,281
		Kyber768	134,746	162,844	145,867	24,569	27,331	31,730
		Kyber1024	216,967	243,097	230,482	40,833	43,633	50,003
	Kyber	Kyber512	81,559	104,489	89,191	17,709	18,857	22,867
		Kyber768	145,993	172,730	157,122	29,083	32,083	36,254
		Kyber1024	231,471	256,230	245,031	46,510	49,577	55,682
	Speedup	Kyber512	(13.8%)	(7.9%)	(12.8%)	(31.5%)	(25.0%)	(24.4%)
		Kyber768	(7.7%)	(5.7%)	(7.1%)	(15.5%)	(14.8%)	(12.4%)
		Kyber1024	(6.2%)	(5.1%)	(5.9%)	(12.2%)	(11.9%)	(10.2%)
	This work (90s)	Kyber512	86,248	101,251	94,302			
		Kyber768	181,638	194,866	193,218			
		Kyber1024	307,482	316,423	322,747			
	Kyber (90s)	Kyber512	111,431	123,024	119,491			
		Kyber768	210,607	223,076	222,396		**N/A**[1]	
		Kyber1024	346,102	353,956	361,065			
	Speedup (90s)	Kyber512	(22.6%)	(17.6%)	(21.0%)			
		Kyber768	(13.7%)	(12.6%)	(13.1%)			
		Kyber1024	(11.1%)	(10.6%)	(10.6%)			

[1]There is no NEON instruction set optimized codes for 90s variant of Kyber in [16].

For performance evaluation on the high-end ARMv8 architecture devices, we used the widely deployed Performance Application Programming Interface (PAPI) [29], which measures the elapsed time for an event. The library is used in [16] and since it offers APIs on different target platforms, such as the ARM Cortex-A72 and Apple M1 processors, it is also adapted to our design. The Apple-M1 is equipped with four high-performance "Firestorm" cores and four energy-efficient "Icestorm" cores. The high-performance cores feature 192 KB of L1 instruction cache and 128 KB of L1 data cache, as well as a shared 12 MB L2 cache; the energy-efficient cores have 128 KB of L1 instruction cache, 64 KB of L1 data cache, and a shared 4 MB L2 cache.

Besides the implementation on Apple-M1, we also implemented our design on Raspberry Pi 4 which utilizes four 1.5 GHz ARM Cortex-A72 cores. To provide a further comparison, we implemented a pure C code and an optimized NEON instruction set C code from [16]. Table III, provides the results of our implementation in ARMv8 architecture. For example, in Cortex-A72 implementation with NEON instruction set optimization and at security level of 768, the original KeyGen() algorithm for CCA.KEM scheme uses 114,415 cycles to complete, while our design reduces it to 100,423 cycles resulting in 12.2% speedup. For ease of tracking, the mentioned data are bold in Tables II and III.

C. Further Discussion

According to Tables II and III, we achieved the best performance improvement at the 512 security level, regardless of the architecture and platform. The reason is that the ratio of our improvement to the total computational cost, is higher at lower security levels. At higher security levels, other modules, utilize the majority of resources, thus the impact of our improvement will be less noticeable. As a result, our design is most suitable in networks with computational and memory restrictions and lower security requirement that are also prone to physical attacks. Typical examples of such networks are smart grid, wireless sensor, and IoT networks.

Additionally, to satisfy the high speed demands of high-end applications with no area boundary, several PUFs and TRNGs can run in parallel in Algorithms 3 and 4, to create the random sequences faster. This can be critical in applications where PUF is slower than the software entities. On the contrary, in applications where the area is limited, PUF can be used as a TRNG without requiring additional hardware for TRNGs. Besides that, many of PUF methods like SRAM PUFs, can be implemented through the existing hardware of the devices without requiring extra circuitry, making them applicable in area limited devices.

Although our design offers several advantages compared to the reference work, it also comes with its own set of drawbacks. One notable disadvantage is the increase in area and overall complexity of the design. Given this complexity, it becomes crucial to implement the design with extreme care, as even a small mistake could lead to the complete exposure of the secret key, posing a significant security risk.

Another pressing concern is the sensitivity of SRAM PUFs to environmental variables, particularly changes in temperature. This sensitivity can lead to different outputs under slightly varying environmental conditions. To address this issue, different potential solutions can be deployed. One approach is to use other families of PUFs, such as delay-based PUFs, which are known to provide higher robustness to environmental variables. Alternatively, we can employ mitigation techniques, like error correction codes. However, that either of these solutions will increase the implementation complexity of the design while decreasing its overall performance. As a result, depending on the specific application at hand and its constraints, a careful trade-off between performance and reliability must be maintained.

Furthermore, despite the fast and theoretically parallel operation of PUFs and TRNGs alongside software components, inadequate synchronization could lead to potential delays. To address this concern, System-on-Chip (SoC) boards are suggested. These boards facilitate the hardware/software co-design in which the FPGA is used for PUF and TRNG functionalities, while the algorithm execution is handled by the microprocessor, ensuring precise coordination and mitigating potential delays.

In addition to the security claims and proofs of a proposed PUF, it is crucial to conduct more detailed analysis before employing them in PQC applications. Quantum computing has the potential to significantly reduce the search space of PUFs in machine learning-based attacks, rendering many existing PUFs unsuitable for PQC. To tackle this issue, dedicated research has been focused on proposing quantum-secure PUFs [30], [31]. However, at present, these designs still lack the performance efficiency required for high-demand applications. While progress is being made in the pursuit of quantum-resistant PUFs, their practical implementation in PQC scenarios demands further refinement and optimization to meet the performance requirements of real-world applications.

Overall, the primary goal of this paper has been to highlight the advantages of incorporating PUFs and TRNGs in PQC schemes. However, further analysis might be needed to address the remaining pressing issues detailed above.

V. Conclusion

As the NIST competition has been concluded, improving and implementing a standardized PQC scheme has gained more interest compared to proposing a new one. Hence, in this paper, we improved the security of CRYSTALS-Kyber, the only PKE/KEM standardized scheme among the NIST winners. We replaced the pseudo-randomness of the original scheme with true randomness through using PUFs and TRNGs. Our analysis indicates significant improvements in performance and security over the reference work. From security aspects, we provided physical security and more resistance against side-channel attack. While from performance aspect, not only did we eliminate the need for secure storage, but also we reduced the total computational cost of the scheme.

To have a broad comparison and cover a wide range of applications, we implemented our design in two architectures of ARMv7 and ARMv8 on three different processors of ARM Cortex-M4, ARM Cortex-A72, and Apple-M1. Our implementation results conclude that our best results were achieved at lower security levels making our design suitable especially in applications with resource-constrained devices, (computational and memory) with the possibility of physical attacks. However, despite the security and performance advantages of our design it still requires further analysis.

Acknowledgements

This work was supported by the US National Science Foundation (NSF) through the award SaTC-1801488.

References

[1] P. W. Shor, “Algorithms for quantum computation: Discrete logarithms and factoring,” in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994.

[2] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, “Crystals - Kyber: A CCA-secure module-lattice-based KEM,” in *Proc. 2018 IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, London, U. K., Apr. pp. 353-367, 2018.

[3] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals-Dilithium: A lattice-based digital signature scheme,” in *Proc. IACR Trans. Cryptograph. Hardw. Embedded Syst.*, pp. 238–268, 2018.

[4] P. A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, “Falcon: Fast-Fourier lattice-based compact signatures over NTRU,” *Submission to NIST, 36(5).* 2018.

[5] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, “The SPHINCS + signature framework,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, pp. 2129–2146, 2019.

[6] A. Jati, N. Gupta, A. Chattopadhyay, and S. K. Sanadhya, “A configurable CRYSTALS-Kyber hardware implementation with side-channel protection,” *ACM Trans. Embedded Comput. Syst.*, 2023.

[7] Z. Xu, O. M. Pemberton, S. Sinha Roy, D. Oswald, W. Yao, and Z. Zheng, “Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber,” *IEEE Trans. Comput.*, 2021.

[8] E. Dubrova, K. Ngo, and J. Gärtner “Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste,” *IACR Cryptol. ePrint Arch*, 2022.

[9] Y. Xing and S. Li, “A compact hardware implementation of CCA-Secure Key Exchange Mechanism CRYSTALS-Kyber on FPGA,” in *Proc. IACR Trans on Cryptograph. Hardw. Embedded Syst*, pp. 328–356, 2021.

[10] Y. Huang, M. Huang, Z. Lei, and J. Wu, “A pure hardware implementation of CRYSTALS-Kyber PQC algorithm through Resource Reuse,” *IEICE Electron. Exp.*, vol. 17, no. 17, Art. no. 20200234, 2020.

[11] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, “Instruction-set accelerated implementation of CRYSTALS-Kyber,” *IEEE Trans. Circuits Syst. I,* vol. 68, no. 11, pp. 4648–4659, 2021.

[12] T. Kamucheka, A. Nelson, D. Andrews, and M. Huang, “A masked pure-hardware implementation of Kyber cryptographic algorithm,” in *Proc 2022 Int. Conf. on Field-Program. Techn. (ICFPT)*, 2022.

[13] Z. Ni, A. Khalid, M. O’Neill, and W. Liu, “Efficient Pipelining Exploration for a High-performance CRYSTALS-Kyber Accelerator,” *IACR Cryptol. ePrint Arch*, 2022.

[14] L. Botros, M. J. Kannwischer, and P. Schwabe, “Memory-efficient high-speed implementation of Kyber on Cortex-M4,” in *Proc. 11th Int. Conf. Cryptol.,* Rabat, Morocco, pp. 209-228, 2019.

[15] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, “Sapphire: A configurable crypto-processor for Post-Quantum lattice-based protocols,” in *Proc. IACR,* p. 1140, 2019.

[16] D. T. Nguyen and K. Gaj, “Optimized software implementations of CRYSTALS-Kyber, NTRU, and Saber using NEON-based special instructions of ARMv8,” in *Proc. NIST 3rd PQC Conf.*, 2021.

[17] E. Alkim, H. Evkan, N. Lahr, R. Niederhagen, and R. Petri, “ISA extensions for finite field arithmetic accelerating Kyber and NewHope on RISC-V,” in *Proc. IACR,* vol. 3, pp. 219–242, 2020.

[18] J. B. Almeida, M. Barbosa, G. Barthe, B. Grégoire, V. Laporte, J. C. Léchenet, T. Oliveira, H. Pacheco, M. Quaresma, P. Schwabe, and A. Séré, “Formally verifying Kyber Part I: Implementation Correctness.” *IACR Cryptol. ePrint Arch*, 2023.

[19] B. Cambou, M. Gowanlock, B. Yildiz, D. Ghanaimiandoab, K. Lee, S. Nelson, C. Philabaum, A. Stenberg, and J. Wright, “Post Quantum cryptographic keys generated with physical unclonable functions,” *Applied Sciences,* vol. 11, no. 6, p. 2801, 2021.

[20] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, “CRYSTALS-kyber–algorithm specifications and supporting documentation,” v 3.02, 2021. Accessed: Jan. 20, 2023. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[21] E. Fujisaki and T. Okamoto, “Secure integration of asymmetric and symmetric encryption schemes,” in *Proc. Int. Cryptol. Conf.,* 1999.

[22] M. Stipčević and Ç. K. Koç, “True random number generators,” in *Open Problems in Mathematics and Computational Science,* C. K. Koc, Ed. Cham, Switzerland: Springer, pp. 275–315, 2014.

[23] Y. Gao, S. F. Al-Sarawi, and D. Abbott, “Physical unclonable functions,” *Nature Electron.,* vol. 3, pp. 81–91, 2020.

[24] S. Buchovecká, R. Lórencz, F. Kodýtek, and J. Bucek, “True random number generator based on ring oscillator PUF circuit,” *Microprocessors Microsyst.*, vol. 53, pp. 33–41, Aug. 2017.

[25] N. N. Anandakumar, M. S. Hashmi, and M. Tehranipoor, “FPGA-based physical unclonable functions: A comprehensive overview of theory and architectures,” *Integration,* vol. 81, pp. 175–194, 2021.

[26] M. J. Kannwischer, P. Pessl, and R. Primas, “Single-trace attacks on Keccak,” *IACR Trans. Cryptogr. Hardw. Embed. Syst.,* vol. 2020, no. 3, pp. 243–268, 2020.

[27] D. Li, Z. Lu, X. Zou, and L. Zhenglin, “PUFKEY: A high-security and high-throughput hardware true random number generator for sensor networks,” *Sensors*, vol. 15, pp. 26251–26266, Oct. 2015.

[28] I. Cicek and A. Al Khas, "A new read-write collision-based SRAM PUF implemented on Xilinx FPGAs," *J. of Cryptograph. Eng,* pp. 1-18, 2022.

[29] D. Terpstra, H. Jagode, H. You, and J. Dongarra, “Collecting performance data with PAPI-C,” in *Tools for High Performance Computing. Springer*, 2010.

[30] Y. Wang, X. Xi, and M. Orshansky, “Lattice PUF: A strong physical unclonable function provably secure against machine learning attacks,” in *Proc. IEEE Int. Symp. Hardware Oriented Secur. Trust*, 2020.

[31] X. Xi, G. Li, Y. Wang, and M. Orshansky. "A provably secure strong puf based on lwe: Construction and implementation," *IEEE Trans. Comput.*, vol 72, no. 2, pp. 346-359, 2023.

Saeed Aghapour received his B.Sc. degree in Electrical Engineering, from Babol Noshirvani University of Technology, Babol, Iran in 2014 and his M.Sc. in Communication Cryptology from the Electrical Engineering department of Sharif University of technology, Tehran, Iran in 2016. He is currently a Ph.D. student at the University of South Florida with the Department of Computer Science and Engineering, Tampa, FL. His current research interests include applied cryptography, post-quantum cryptography, hardware security, and fault detection.

Kasra Ahmadi received a B.Sc. degree in Computer Engineering from Isfahan University of Technology, Isfahan, Iran and an M.Sc. degree in Information Technology from AmirKabir University of Technology, Tehran, Iran. He is currently pursuing a Ph.D. program at the Computer Science and Engineering Department of University of South Florida. His current research interests include fault detection on elliptic curves, post-quantum cryptography, optimized implementation of cryptographic schemes, side-channel attacks and applied cryptography.

Mila Anastasova graduated in 2019 from the University Carlos III of Madrid, Spain, with a degree in Computer Science and Engineering. She earned her MS in Computer Engineering from Florida Atlantic University, United States, where she is currently pursuing her Ph.D. in Computer Engineering with the Institute for Sensing and Embedded Network Systems Engineering (I-SENSE). Her research interests include emerging security primitives for real-time IoT systems, classical and post-quantum public key cryptography schemes, as well as their optimum and SCA resistant implementation on low-end devices and incorporation into network protocols.

Mehran Mozaffari Kermani (S’00-M’11-SM’16) received the B.Sc. degree from the University of Tehran, Tehran, Iran, in 2005, and the M.E.Sc. and Ph.D. degrees from University of Western Ontario, London, Canada, in 2007 and 2011, respectively.

He joined the Advanced Micro Devices as a senior ASIC/layout designer, integrating sophisticated security/cryptographic capabilities into accelerated processing. In 2012, he joined the Electrical Engineering Department, Princeton University, New Jersey, as an NSERC post-doctoral research fellow. From 2013-2017 he was a faculty with Rochester Institute of Technology and starting 2017, he is an Associate Professor with the Computer Science and Engineering Department of University of South Florida.

Currently, he is serving as an Associate Editor for the *IEEE Transactions on VLSI Systems*, the *ACM Transactions on Embedded Computing Systems*, and the *IEEE Transactions on Circuits and Systems.* He has been the TPC member for HOST (Publications Chair), CCS (Publications Chair), DAC, DATE, RFIDSec, LightSec, WAIFI, FDTC, and DFT. He was a recipient of the prestigious Natural Sciences and Engineering Research Council of Canada Post-Doctoral Research Fellowship in 2011 and the Texas Instruments Faculty Award (Douglas Harvey) in 2014. He is also the awardee for USF 2021 Faculty Outstanding Research Achievement Award, and USF College of Engineering’s 2018 Outstanding Junior Research Achievement Award. He is a Senior Member of the IEEE.

Reza Azarderakhsh received the Ph.D. degree in electrical and computer engineering from Western University in 2011. He was a recipient of the NSERC Post-Doctoral Research Fellowship working in the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo. Currently, he is a Professor at Florida Atlantic University. He was the Guest Editor for the *IEEE Transactions on Dependable and Secure Computing* for the special issue of Emerging Embedded and Cyber Physical System Security Challenges and Innovations (2016 and 2017). He was also the Guest Editor for the *IEEE/ACM Transactions on Computational Biology and Bioinformatics* for special issue on security. He is serving as an Associate Editor of *IEEE Transactions on Circuits and Systems (TCAS-I).*