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

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

***Abstract*—It is well-studied that quantum computing breaks the security of the current worldwide implemented public key cryptosystems. This forces us toward post quantum cryptography (PQC) whose security remains solid even against adversaries having access to quantum computers. For this matter, national institute of standards and technology (NIST) announced four winners in 2022. Among them, CRYSTALS-Kyber which is the only key encapsulation mechanism (KEM)/PKE algorithm, is the aim of this article. In this article, through using physical unclonable functions (PUFs) and true random number generators (TRNGs), we improve the overall security of 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. Comparing the implementation results of our design with the reference work indicates that both the security and the system performance are improved.**

***Index Terms*—CRYSTALS-Kyber, physical unclonable functions (PUFs), post quantum cryptography (PQC).**

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, current classic public key cryptography, will be broken in polynomial time by the Shor's algorithm [1]. Therefore, the need for fully transitioning to new cryptosystems that are secure even against quantum computing is eminent. In order to facilitate the process of the transition to post quantum cryptography (PQC), national institute of standards and technology (NIST) concluded a standardization competition 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 key encapsulation mechanism (KEM), 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, needs to be scrutinized for these algorithms.

Manuscript received 23 July 2023; revised 4 November 2023, 16 January 2024, and 19 April 2024; accepted 7 May 2024. Date of publication 10 May 2024; date of current version 22 November 2024. This work was supported in part by the U.S. National Science Foundation (NSF) under Grant SaTC-1801488. This article was recommended by Associate Editor R. S. Chakraborty. *(Corresponding author: Mehran Mozaffari Kermani.)*

Saeed Aghapour, Kasra Ahmadi, and Mehran Mozaffari Kermani are with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL 33620 USA (e-mail: aghapour@usf.edu; ahmadi1@usf.edu; mehran2@usf.edu).

Mila Anastasova and Reza Azarderakhsh are with the Department of Computer and Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431 USA (e-mail: manastasova2017@fau.edu; razarderakhsh@fau.edu).

Digital Object Identifier 10.1109/TCAD.2024.3399669

A. Related Work

The research is mainly divided into two divisions of side-channel analysis and optimized implementation. Side-channel 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. In [6], a configurable and side-channel resistant implementation of Kyber is introduced which reported an increase of around 5% to the overhead of the original design. 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 Cortex-M4 implementation of Kyber is provided.

The Kyber resources are primarily dominated by the number theoretic transform (NTT) and Keccak modules. Keccak operations are employed for hashing and sampling, whereas NTT handles polynomial operations. In software implementation, Keccak operations consume more than half of the total clock cycles [9]. Additionally, as demonstrated in our prior work [10], around 32% and 25% of the area is related to NTT and Keccak modules over ASIC platform. Moreover, in the FPGA implementation in [11], SHAKE-256 utilizes 15,704 LUTs and 7,592 FFs, while NTT component uses 1,107 LUTs, 1,407 FFs, 28 DSPs, and 3.5 BRAMs. Additionally, in another FPGA implementation [12], hash and Keccak modules consume 62% of the total resources. In summary, for software implementations, Keccak accounts for more than half of the total clock cycles. Nevertheless, in hardware implementations, although Keccak operations can be accelerated, they still occupy 25% of the total area [13].

Xing and Li [14] and Huang et al. [15] provided the results of their pure hardware implementation of Kyber on the AMD/Xilinx Artix-7 FPGA in detail. By utilizing hiding and masking techniques, the work in [16] presented a hardware implementation of Kyber that is secure against simple and differential power analysis side-channel attacks. Ni et al. [17] presented a highly area-time efficient implementation of Kyber on AMD/Xilinx Artix-7 and Zynq-UltraScale+ FPGA families.

On the software implementation side, the work of [18] implemented Kyber on ARM Cortex-M4. By improving the NTT computations, they improved the overall speed of the system by around 18%. In [19], 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, Nguyen and Gaj [20] provided an optimized implementation of Kyber, NTRU, and Saber by using NEON instructions. The work in [21] presented a new extension to the instruction set for RISC-V finite field arithmetic which efficiently reduced code and data size and improved the polynomial arithmetic by up to 85%.

B. Major Contribution

While various physical unclonable function (PUF)-true random number generator (TRNG)-based designs have been introduced for different cryptographic objectives, their application to the new standardized NIST schemes remains unexplored. With Kyber being chosen as the sole KEM scheme to replace the classical cryptography, a comprehensive investigation of its various aspects becomes crucial prior to practical implementation. One of the paramount considerations for a cryptosystem in network environments, such as IoT, WSN, and smart grids is its resilience against physical attacks. Hence, our goal is to leverage PUF technology to enhance the physical security of Kyber, which, as the only standardized KEM scheme to date.

To the best of our knowledge, the only work that utilizes PUF in PQC schemes is [22] 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 article 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 seed or keys, hence the storage burden is reduced.
3) This work also enhances the entropy of the secret keys because of the true randomness of PUFs and TRNGs.

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:     $s[i] := \mathrm{CBD}_{\eta_1}(PRF(\sigma, N))$
11:     $N := N + 1$
12: end for
13: for $i = 0$ to $k-1$ do
14:     $e[i] := \mathrm{CBD}_{\eta_1}(PRF(\sigma, N))$
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: $pk := (E_{12}(\hat{t} \bmod^{+} q) \parallel \rho)$
21: $sk := E_{12}(\hat{s} \bmod^{+} q)$
22: return $(pk, sk)$
```

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.

II. Preliminaries

In this section, we provide a brief description of the 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 final version of which we focus [23]. Kyber has a PKE and a KEM scheme. Algorithms 1 and two depict KeyGen() and Enc() algorithms of the Kyber CPA.PKE scheme.

By taking advantage of the FO transform [24], Kyber CCA.KEM scheme results directly from the Kyber CPA.PKE scheme. A typical KEM scheme consists of three algorithms: 1) KeyGen(); 2) encapsulation(); and 3) decapsulation(). Furthermore, there are two variants of the Kyber scheme named Kyber and Kyber 90s which are similar in the algorithms and only differ in their functions instantiation. In the original scheme, PRF is instantiated with SHAKE-256 or AES-256, while in our case, it is instantiated by PUF and TRNG. Please refer to [23] for further details, omitted here for the sake of brevity.

B. True Random Number Generators

While pseudo-random number generators (PRNGs) 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. 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 thermal noise, clock drift, photon arrival times, and the like [25]. Nonetheless, the most practical and inexpensive methods for cryptography purposes are based on delay, noise, phase jitter, and memory. Moreover, TRNGs can be implemented through the FPGA components.

C. Physical Unclonable Functions

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 [26]. 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. 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 metrics, including reliability, uniqueness, uniformity, unpredictability, and tamper-evident are considered [27].

Generally, physical attacks encompass a wide range of threats, including memory attacks and the complete physical capture of a device. PUFs are primarily effective at mitigating memory-related physical attacks, as they do not rely on memory, making it impossible for adversaries to probe for sensitive information. Furthermore, most PUFs are tamper-evident, meaning that any attempt to probe or modify the device can disrupt the PUF's original functionality, rendering its responses unreliable. Consequently, adversaries cannot extract the PUF from the device for separate use.

III. Proposed PUF-KYBER Architecture

In this section, we target both of the Kyber schemes. In [23], 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 SHAKE-256 and AES-256 for Kyber and Kyber 90s, respectively. For our design, we instantiate PRF with a PUF and a TRNG in the KeyGen() and Enc() algorithms. We divide this section into three parts. In parts A and B, we propose our new designs while in part C, we discuss our gains and advantages over the original design.

A. New CPA.PKE Scheme

In Kyber CPA.PKE scheme, according to Algorithm 1, d is chosen randomly (Step 1). Then, this d is hashed and the result will be used as the seed of the PRF function alongside a counter (Steps 10 and 11) to create the secret key. As a result, the security of the secret key is directly dependent on d. Similarly, in Algorithm 2, the value r is chosen randomly. With these in mind, although the original paper did not mention this specifically, to have high entropy and randomness

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:    $r[i] := \mathrm{CBD}_{\eta_1}(PRF(r, N))$
11:    $N := N + 1$
12: end for
13: for $i = 0$ to $k-1$ do
14:    $e_1[i] := \mathrm{CBD}_{\eta_2}(PRF(r, N))$
15:    $N := N + 1$
16: end for
17: $e_2 := \mathrm{CBD}_{\eta_2}(PRF(r, N))$
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 $(c_1 \| c_2)$
```

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 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(), (see Algorithm 2), PRF is used to create noise and error polynomials r, e_1, and e_2. Unlike the secret keys, noise polynomials have one-time usage. Therefore, the reproducibility feature of PUFs is not required here. For this reason, in this algorithm, we instantiate PRF with a TRNG whose role is to create one-time true random noise polynomials with higher entropy in comparison with PRNG. The new Enc() algorithm is proposed in Algorithm 4. CPA.PKE.Dec() algorithm of our design remains unchanged.

B. New CCA.KEM Scheme

Similar to Kyber CPA.PKE, we assume that the 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 Section III-A (Algorithm 3), we modify the CCA.KEM.KeyGen() algorithm. However, a similar strategy

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:  for $i = 0$ to $k-1$ do
3:      for $j = 0$ to $k-1$ do
4:          $\hat{A}[i][j] := Parse(XOF(\rho, j, i))$
5:      end for
6:      $a_i := PUF(\rho)$
7:      $\rho = \rho << 1$
8:      $b_i := PUF(\rho)$
9:      $\rho = \rho << 1$
10:     $s[i] := \mathrm{CBD}_{\eta_1}(a_i)$
11:     $e[i] := \mathrm{CBD}_{\eta_1}(b_i)$
12: end for
13: $\hat{s} := NTT(s)$
14: $\hat{e} := NTT(e)$
15: $\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$
16: $pk := (E_{12}(\hat{t} \bmod^+ q) \parallel \rho)$
17: $sk := E_{12}(\hat{s} \bmod^+ q)$
18: return $(pk, sk)$
```

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:  $(a_0 \parallel \ldots \parallel a_{k-1} \parallel b_0 \parallel \ldots \parallel b_{k-1} \parallel c) \leftarrow TRNG(.)$
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:      $r[i] := \mathrm{CBD}_{\eta_1}(a_i)$
9:      $e_1[i] := \mathrm{CBD}_{\eta_2}(b_i)$
10: end for
11: $e_2 := \mathrm{CBD}_{\eta_2}(c)$
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 $(c_1 \parallel c_2)$
```

is not applicable for encapsulation algorithm. With more details, as the 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 [23, Algorithm 9, Step 6]. 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 SHAKE-256 or AES-256 which 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. Thus, in our CCA.KEM scheme, only the KeyGen() algorithm is changed.

C. Security Analysis

It is well established that the entropy of random sequences that are created by a TRNG source is significantly higher than 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, if an adversary captures the users physically and access their memories, they 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. This provides physical security for our design.

Similarly, 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, 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: 1) higher entropy for secret keys; 2) physical security; and 3) more resistance against the 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 the 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. Conversely, 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. 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.

1) Improving computational cost.
2) Eliminating the need for secure storage.

TABLE I
RANDOM BYTES NEEDED IN DIFFERENT SECURITY LEVELS OF KYBER

	k	η_1	η_2	B_{PUF}	B_{TRNG}
Kyber-512	2	3	2	768	768
Kyber-768	3	2	2	768	896
Kyber-1024	4	2	2	1024	1152

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 output a polynomial deterministically from $64\eta_i$ bytes of input. 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 the PUF and TRNG modules, respectively.

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 [28] as our TRNG. This work, proposes an SRAM-based TRNG, offering 100 MBps throughput on Virtex-II Pro and utilizes 369 slices, while passing all NIST statistical randomness tests with high scores.

It is worth noting to mention that, the choice of PUF is not universal and could be based on the designated application. However, several criteria must be met before selecting a PUF. The most crucial one is that the PUF must offer 100% reliability (error probability of less than 10^{-9}). That being said, while SRAM PUFs are relatively fast and easy to implement, they require error correction codes (ECCs) to achieve 100% reliability. Error correction methods involve helper data, increasing not only storage overhead but also introducing potential security issues. Additionally, the length of the helper data is proportional to the number of reliable bits required from the PUF. Consequently, ECC is suitable for applications, where the PUF is employed for creating a small seed. However, based on Table I, we require up to 1024 reliable bytes, demonstrating the impracticality of ECC in our work. Therefore, our best choice is self-error correction PUFs that do not necessitate error correction methods. To that end, we selected [29], which introduces a PUF providing 100% reliability without requiring ECCs.

This PUF is an arbiter PUF that removes any unstable bits in predicted environmental conditions that would probably cause unreliability issues later. As a result, the responses will be 100% reliable in the predicted environment. Furthermore, this PUF exhibits almost 100% reliability (error probability of less than 10^{-9}), 52.43% uniformity, and 48.82% uniqueness in tests conducted across a temperature range of 0–80 °C. The implementation of this PUF on Spartan 6 FPGA utilizes only 104 LUTs and 38 FFs. From a performance perspective, to generate 128 bits of a reliable key, it requires 8200 clock cycles on Spartan 6 FPGA with a clock frequency of 100 MHz.

From security standpoint, when dealing with a PUF, its security against the machine learning and side-channel attacks becomes a concern. In machine learning-based attacks, adversaries gather numerous challenge-response pairs (CRPs) and attempt to simulate or clone the PUF. The objective of this attack is to create a function that replicates the same physical functionality as those of the PUF without having the physical access to it. However, as previously mentioned, this attack necessitates access to a large number of CRPs. In applications where PUF is utilized for authentication, this attack could be applicable, as PUF responses are not kept secret. On the contrary, in applications where PUF responses are confidential and directly used as keys, collecting a high number of CRPs is not feasible. Consequently, this attack is not practical in key generation applications of PUFs [27].

Moreover, when addressing side-channel attacks on PUFs, it is crucial to recognize that numerous attacks aim to exploit sensitive information about the PUF response derived from the helper data employed for error correction [30]. In our case, the deployed PUF stands out as it eliminates the necessity for both the helper data and ECC, rendering these specific attacks and their corresponding countermeasures, inapplicable [31]. However, it is imperative to acknowledge that even though the chosen PUF configuration does not rely on helper data and ECC, there remains a potential for the deployed PUF to inadvertently leak sensitive information if its implementation is not executed with due diligence. Therefore, comprehensive and ongoing studies are warranted, focusing on the inherent security aspects of the deployed PUF itself.

B. Methodology and Implementation Result

To gain insight on the area overhead of our design, we need to delve into the hardware specifics of the original research. In our prior study [13], conducted on an Artix 7 FPGA, we executed the Kyber-1024 algorithm using 16k LUTs, 6k FFs, 5k slices, 12 DSPs, and 17 BRAMs. Consequently, the additional area utilization amounts to 0.34% when integrating PUF and 7.38% when incorporating TRNG. Moreover, in practical scenarios, the original design already necessitates a TRNG for seed generation, which has not been considered in most prior studies. Thus, the new PUF/TRNG module will replace the existing one further reducing the area overhead.

Furthermore, as PUF and TRNG run parallel to the software entities, in theory the overall performance of the system will be bound by the slowest part. Hence, 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, meaning the overall performance will be limited by the software entities.

To benchmark the software entities of our design, we implemented it on two different architectures of ARMv7 and ARMv8. For ARMv7, we used STM32F407G discovery board featuring the widely deployed Cortex-M4 processor and the pqm4 library.[1] The pqm4 library provides a framework for

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

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	This work	Kyber-512	241,759	247,531	319,813	241,539	249,257	319,560
		Kyber-768	496,585	501,486	612,436	498,018	519,086	613,810
		Kyber-1024	847,964	851,421	1,003,037	852,192	859,755	1,019,193
	Kyber	Kyber-512	356,125	287,307	433,708	355,938	339,770	433,890
		Kyber-768	588,632	594,256	704,423	602,938	611,853	706,866
		Kyber-1024	967,366	970,696	1,122,664	971,140	979,023	1,126,112
	Speedup[1]	Kyber-512	(32.1%)	(13.8%)	(26.2%)	(32.1%)	(26.6%)	(26.3%)
		Kyber-768	(15.6%)	(15.6%)	(13.1%)	(17.4%)	(15.1%)	(13.1%)
		Kyber-1024	(12.3%)	(12.2%)	(10.6%)	(12.2%)	(12.1%)	(9.4%)
	This work (90s)	Kyber-512	214,004	219,354	248,918	214,644	222,348	249,207
		Kyber-768	434,963	439,432	479,740	437,388	445,359	487,728
		Kyber-1024	734,833	732,800	798,576	744,282	752,395	809,528
	Kyber (90s)	Kyber-512	334,695	280,492	365,220	335,775	339,227	370,112
		Kyber-768	566,047	582,045	607,037	568,659	587,970	619,049
		Kyber-1024	902,444	916,159	976,099	917,466	935,746	982,636
	Speedup (90s)[1]	Kyber-512	(36.1%)	(21.7%)	(31.8%)	(36.1%)	(34.4%)	(32.6%)
		Kyber-768	(23.1%)	(24.5%)	(20.9%)	(23.1%)	(24.2%)	(21.2%)
		Kyber-1024	(18.5%)	(20.1%)	(18.1%)	(18.8%)	(19.5%)	(17.6%)
168 Mhz	This work	Kyber-512	264,539	266,833	349,293	264,709	269,589	349,570
		Kyber-768	540,268	541,679	666,221	543,062	547,629	669,240
		Kyber-1024	920,072	932,513	1,089,159	926,792	928,603	1,093,898
	Kyber	Kyber-512	388,422	310,179	473,562	389,104	368,273	473,810
		Kyber-768	641,447	642,896	767,758	643,760	648,820	769,083
		Kyber-1024	1,064,821	1,062,554	1,218,593	1,055,863	1,058,650	1,223,259
	Speedup[1]	Kyber-512	(31.8%)	(13.9%)	(26.2%)	(31.9%)	(26.7%)	(26.2%)
		Kyber-768	(15.7%)	(15.7%)	(13.2%)	(15.6%)	(15.5%)	(12.9%)
		Kyber-1024	(13.5%)	(12.2%)	(10.6%)	(12.2%)	(12.2%)	(10.5%)
	This work (90s)	Kyber-512	260,555	263,438	300,758	261,649	266,649	300,948
		Kyber-768	532,248	533,107	587,093	537,585	542,102	592,124
		Kyber-1024	906,978	904,148	980,638	917,580	912,584	988,457
	Kyber (90s)	Kyber-512	406,746	337,906	446,144	407,966	408,332	447,014
		Kyber-768	693,217	706,953	748,234	698,627	715,921	748,067
		Kyber-1024	1,117,672	1,127,613	1,191,304	1,121,077	1,136,063	1,199,222
	Speedup (90s)[1]	Kyber-512	(35.9%)	(22.1%)	(32.5%)	(35.8%)	(34.6%)	(32.6%)
		Kyber-768	(23.2%)	(24.5%)	(21.5%)	(23.1%)	(24.2%)	(20.8%)
		Kyber-1024	(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) $= \frac{\text{Kyber (90s)} - \text{This work (90s)}}{\text{Kyber (90s)}} \times 100$.

performance evaluation of the emerging post quantum cryptographic primitives, targeting the SMT32F407VG - discovery board. Despite the effort of different cryptographic engineering in optimizing the design of PQC schemes, a tradeoff between the latency and stack usage is required. That is the reason for the two different designs of the Kyber contained in the pqm4 library named stack and speed designs. Speed design ensures minimal execution time while the 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.

Table II represents our benchmark on the ARM Cortex-M4 platform and compares it to the reference work in two different frequencies and two different implementation designs. Cortex-M series is well-suited for resource-constrained usage models like IoT devices. Thanks to their low power consumption and high efficiency, the Cortex-M4, for example, can effectively manage even demanding tasks, such as PQC within its limited and constrained resource environment. Moreover, regardless of the application or available computational power, the looming threat of quantum computing on classical cryptography necessitates a transition to PQC for every device in the future. Kyber, as the only standardized KEM scheme up to this date, and notably the most efficient one in terms of computational cost among the remaining NIST round four candidates, is well-positioned to likely replace classical cryptography, particularly in resource-constrained devices. Therefore, the results presented in Table II provide valuable insights into how these devices perform in the real-world IoT applications.

For performance evaluation on the high-end ARMv8 architecture devices, we used the widely deployed performance application programming interface (PAPI) [32], which measures the elapsed time for an event. The library is used in [20] 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. 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

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	Kyber-512	109,331	144,699	120,223	42,834	52,851	53,604
		Kyber-768	207,350	248,890	222,238	84,592	98,475	100,423
		Kyber-1024	322,459	367,675	343,044	145,566	157,792	166,250
	Kyber	Kyber-512	126,354	157,348	137,286	60,680	67,245	71,410
		Kyber-768	223,107	263,740	239,033	98,528	112,951	114,415
		Kyber-1024	344,183	387,816	364,835	163,623	176,452	184,264
	Speedup	Kyber-512	(13.4%)	(8.1%)	(12.4%)	(29.4%)	(21.4%)	(24.9%)
		Kyber-768	(7.1%)	(5.6%)	(7.1%)	(14.1%)	(12.8%)	(12.2%)
		Kyber-1024	(6.3%)	(5.2%)	(5.9%)	(11.1%)	(10.5%)	(9.7%)
	This work (90s)	Kyber-512	181,494	222,820	195,097			
		Kyber-768	376,332	424,132	395,461			
		Kyber-1024	628,563	682,470	654,490			
	Kyber (90s)	Kyber-512	230,763	265,434	244,423			
		Kyber-768	433,840	480,697	454,377		**N/A**[1]	
		Kyber-1024	706,423	757,977	732,872			
	Speedup (90s)	Kyber-512	(21.3%)	(16.1%)	(20.1%)			
		Kyber-768	(13.2%)	(11.7%)	(12.9%)			
		Kyber-1024	(11.1%)	(9.9%)	(10.7%)			
Apple-M1	This work	Kyber-512	70,246	96,195	77,775	12,122	14,126	17,281
		Kyber-768	134,746	162,844	145,867	24,569	27,331	31,730
		Kyber-1024	216,967	243,097	230,482	40,833	43,633	50,003
	Kyber	Kyber-512	81,559	104,489	89,191	17,709	18,857	22,867
		Kyber-768	145,993	172,730	157,122	29,083	32,083	36,254
		Kyber-1024	231,471	256,230	245,031	46,510	49,577	55,682
	Speedup	Kyber-512	(13.8%)	(7.9%)	(12.8%)	(31.5%)	(25.0%)	(24.4%)
		Kyber-768	(7.7%)	(5.7%)	(7.1%)	(15.5%)	(14.8%)	(12.4%)
		Kyber-1024	(6.2%)	(5.1%)	(5.9%)	(12.2%)	(11.9%)	(10.2%)
	This work (90s)	Kyber-512	86,248	101,251	94,302			
		Kyber-768	181,638	194,866	193,218			
		Kyber-1024	307,482	316,423	322,747			
	Kyber (90s)	Kyber-512	111,431	123,024	119,491			
		Kyber-768	210,607	223,076	222,396		**N/A**[1]	
		Kyber-1024	346,102	353,956	361,065			
	Speedup (90s)	Kyber-512	(22.6%)	(17.6%)	(21.0%)			
		Kyber-768	(13.7%)	(12.6%)	(13.1%)			
		Kyber-1024	(11.1%)	(10.6%)	(10.6%)			

[1]There is no NEON instruction set optimized codes for 90s variant of Kyber in [20].

and an optimized NEON instruction set C code from [20]. Table III provides the results of our implementation in ARMv8 architecture.

C. Comparison

In this section, we compare our work with related efforts, considering hardware and software overhead. Table IV provides a concise overview, showcasing a fair comparison with the current state-of-the-art. Specifically, the table highlights the KeyGen() algorithm's overhead in CCA-Kyber-1024. Our design, as depicted in Table IV, brings a slight increase in area overhead while simultaneously removing the need for secure storage and enhancing security against memory attacks.

D. Further Discussion

According to Tables II and III, we achieved the best improvement at the 512 security level, regardless of the architecture and platform, since the ratio of our improvement to the total computational cost, is higher at lower security levels. Thus, our design is most suitable in networks with computational and memory restrictions that are prone to physical attacks. Although our design offers several advantages compared to the reference work, it also has its drawbacks.

One notable disadvantage is the increase in 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. 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.

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

TABLE IV
COMPREHENSIVE COMPARISON WITH RELATED WORKS FOR KYBER-1024

	Method	Features	Overhead						
			HW					SW	
			Platform	Freq (MHz)	Area	CC[1]	Secure Storage	Platform	CC[1]
[12]	HW	——	Artix-7	159	7.9K LUTs- 3.9K FFs	7.8	32 MB	——	——
[6]	HW	SCA Resistance	Artix-7 Artix-7	250 258	5.2K LUTs- 2.4K FFs 7.1K LUTs- 3.7K FFs	1148 43.8	32 MB	——	——
[14]	HW	——	Artix-7	161	7.4K LUTs- 4.6K FFs	9.4	32 MB	——	——
[13]	HW	——	Artix-7	112	16K LUTs- 6K FFs	10	32 MB	——	——
[21]	HW	——	Artix-7	59	1.8K LUTs- 1.6K FFs	2203	32 MB	——	——
[20]	SW	——	——	——	——	——	32 MB	Cortex-A72 (NEON) Apple-M1 (NEON)	184.2 55.6
[35]	SW	——	——	——	——	——	32 MB	Cortex-A75	228
[36]	SW	——	——	——	——	——	32 MB	Cortex-M4	1138
[23]	SW	——	——	——	——	——	32 MB	Cortex-M4 Cortex-A72 Apple-M1	1122.6 364.8 245
Ours	HW/SW	Memory Attacks Resistance	Spartan 6	100	+ 104 LUTs- 38 FFs	+ 520	0	Cortex-M4 Cortex-A72 Apple-M1 Cortex-A72 (NEON) Apple-M1 (NEON)	1003 343 230.4 166.2 50

[1] Number of clock cycles for KeyGen algorithm of CCA-Kyber-1024 based on kilo cycles.

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 [33], [34], but these designs still lack the performance efficiency required for high-demand applications.

Furthermore, the proposed methodology could be adopted to other schemes, including CRYSTALS-Dilithium. Dilithium, being a lattice-based standard signature scheme from the same team as Kyber, shares many characteristics with it. Particularly, the KeyGen algorithms exhibit significant similarities between the two schemes. Initial results from implementing our method on Dilithium show promising improvements in terms of performance obtaining up to 30% improvement in number of clock cycles on the Cortex-M4 platform.

Overall, the primary goal of this article has been to highlight the advantages of incorporating PUFs and TRNGs in the 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 article, we improved the security of CRYSTALS-Kyber, the only PKE/KEM standardized scheme among the NIST winners. We replaced the pseudorandomness of the original scheme with true randomness through using the PUFs and TRNGs. Our analysis indicates significant improvements in performance and security over the reference work. From security aspects, we provided physical security to the original work. 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 in software while mildly increasing the area overhead.

To have a broad comparison, 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 the 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 with more focus on side-channel analysis.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proc. 35th Annu. Symp. Found. Comput. Sci.*, 1994, pp. 124–134.
[2] J. Bos et al., "Crystals-Kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, London, U.K., 2018, pp. 353–367.
[3] L. Ducas et al., "Crystals-Dilithium: A lattice-based digital signature scheme," in *Proc. IACR Trans. Hardw. Embed. Syst.*, 2018, pp. 238–268.
[4] P. A. Fouque et al., "Falcon: Fast-Fourier lattice-based compact signatures over NTRU," *Submiss. NIST, Post-Quantum Cryptogr.*, vol. 36, no. 5, pp. 1–75, 2018.
[5] D. J. Bernstein, A. Hlsing, S. Klbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS +signature framework," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 2129–2146.
[6] A. Jati, N. Gupta, A. Chattopadhyay, and S. K. Sanadhya, "A configurable CRYSTALS-Kyber hardware implementation with side-channel protection," *ACM Trans. Embed. Comput. Syst.*, vol. 23, no. 2, pp. 1–25, 2023.

[7] Z. Xu, O. Pemberton, S. 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. Comp.*, vol. 71, no. 9, pp. 2163–2176, Sep. 2021.

[8] E. Dubrova, K. Ngo, and J. Grtner, "Breaking a fifth-order masked implementation of CRYSTALS-Kyber by copy-paste," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2022/1713, 2022.

[9] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen, "pqm4: Testing and benchmarking NIST PQC on ARM Cortex-M4," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2019/844, 2019.

[10] M. Bisheh-Niasar, R. Azarderakhsh, and M. Mozaffari-Kermani, "A monolithic hardware implementation of Kyber: Comparing apples to apples in PQC candidates," in *Proc. Int. Conf. Cryptol. Inf. Secur.*, 2021, pp. 108–126.

[11] S. Ricci, P. Jedlicka, P. Cbik, P. Dzurenda, L. Malina, and J. Hajny, "Towards CRYSTALS-Kyber VHDL implementation," in *Proc. 18th Int. Conf. Secur. Cryptogr. (SECRYPT)*, 2021, pp. 760–765.

[12] W. Guo, S. Li, and L. Kong, "An efficient implementation of KYBER," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 3, pp. 1562–1566, Mar. 2022.

[13] 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, Nov. 2021.

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

[15] 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, 2020, Art. no. 20200234.

[16] T. Kamucheka, A. Nelson, D. Andrews, and M. Huang, "A masked pure-hardware implementation of Kyber cryptographic algorithm," in *Proc. Int. Conf. Field-Program. Techn. (ICFPT)*, 2022, pp. 1–1.

[17] Z. Ni, A. Khalid, M. O' Neill, and W. Liu, "Efficient pipelining exploration for a high-performance CRYSTALS-Kyber accelerator," Cryptol. ePrint Arch., IACR, Bellevue, WA, USA, Rep. 2022/1093, 2022.

[18] 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, 2019, pp. 209–228.

[19] U. Banerjee, T. S. Ukyab, and A. P. Chandrakasan, "Sapphire: A configurable crypto-processor for post-quantum lattice-based protocols," in *Proc. IACR*, 2019, p. 1140.

[20] 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, pp. 1–24.

[21] 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*, 2020, pp. 219–242.

[22] B. Cambou et al., "Post Quantum cryptographic keys generated with physical unclonable functions," *Appl. Sci.*, vol. 11, no. 6, p. 2801, 2021.

[23] J. Bos et al. (CRYSTALS-Kyber, Gaithersburg, MD, USA). *CRYSTALS-Kyber Algorithm Specifications and Supporting Documentation, Version 3.02*. 2021. Accessed: Apr. 10, 2024. [Online]. Available: https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf

[24] E. Fujisaki and T. Okamoto, "Secure integration of asymmetric and symmetric encryption schemes," in*Proc. Int. Cryptol. Conf.*, 1999, pp. 537–554.

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

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

[27] 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, Nov. 2021.

[28] 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.

[29] M. Kaveh, M. R. Mosavi, D. Martin, and S. Aghapour, "An efficient authentication protocol for smart grid communication based on on-chip-error-correcting physical unclonable function," *Sustain. Energy Grids Netw.*, vol. 36, Dec. 2023, Art. no. 101228.

[30] A. Alipour et al., "Helper data masking for physically unclonable function-based key generation algorithms," *IEEE Access*, vol. 10, pp. 40150–40164, 2022.

[31] M. Hiller and A. Gurur, "Hiding secrecy leakage in leaky helper data," in *Proc. Int. Conf. Cryptogr. Hardw. Embed. Syst*, 2017, pp. 601–619.

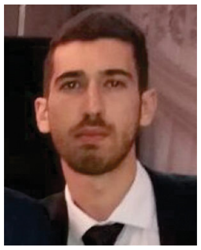
[32] D. Terpstra, H. Jagode, H. You, and J. Dongarra, "Collecting performance data with PAPI-C," in *Proc. Tools High Perform. Comput.*, 2010, pp. 157–173.

[33] 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. Hardw. Orient. Secur. Trust*, 2020, pp. 273–283.

[34] 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, Feb. 2023.

[35] P. Sanal, E. Karagoz, H. Seo, R. Azarderakhsh, and M. Mozaffari-Kermani, "Kyber on ARM64: Compact implementations of Kyber on 64-Bit ARM cortex-A processors," in *Proc. Int. Conf. Secur. Privacy Commun. Syst.*, 2021, pp. 424–440.

[36] A. Abdulrahman, V. Hwang, M. J. Kannwischer, and A. Sprenkels, "Faster Kyber and Dilithium on the cortex-m4," in *Proc. 20th Int. Conf. Appl. Cryptogr. Netw. Secur.*, 2022, pp. 853–871.



Saeed Aghapour received the B.Sc. degree in electrical engineering from Babol Noshirvani University of Technology, Babol, Iran, in 2014, and the M.Sc. degree in communication cryptology from the Department of Electrical Engineering, Sharif University of Technology, Tehran, Iran, in 2016. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA.

His current research interests include applied cryptography, post-quantum cryptography, hardware security, and fault detection.

Kasra Ahmadi received the B.Sc. degree in computer engineering from Isfahan University of Technology, Isfahan, Iran, 2017, and the M.Sc. degree in information technology from the AmirKabir University of Technology, Tehran, Iran, 2020. He is currently pursuing the Ph.D. degree with the Computer Science and Engineering Department, University of South Florida, Tampa, FL, USA.

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 (Graduate Student Member, IEEE) received the bachelor's degree in computer science and engineering from the University Carlos III of Madrid, Getafe, Spain, and the M.S. degree in computer engineering from Florida Atlantic University, Boca Raton, FL, USA, where she is currently pursuing the Ph.D. degree in computer engineering with the Institute for Sensing and Embedded Network Systems Engineering.

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 (Senior Member, IEEE) 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, ON, Canada, in 2007 and 2011, respectively.

He joined with the Advanced Micro Devices as a Senior ASIC/layout Designer, integrating sophisticated security/cryptographic capabilities into accelerated processing. In 2012, he joined with the Department of Electrical Engineering, Princeton University, Princeton, NJ, USA, as an NSERC Postdoctoral Research Fellow. From 2013 to 2017, he was a Faculty with Rochester Institute of Technology, Rochester, NY, USA. In 2017, he was an Associate Professor with the Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA.

Dr. Kermani is the awardee for the USF 2021 Faculty Outstanding Research Achievement Award and the USF College of Engineering's 2018 Outstanding Junior Research Achievement Award. 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 currently 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.

Reza Azarderakhsh (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Western University, London, ON, Canada. in 2011.

He is currently a Professor with Florida Atlantic University, Boca Raton, FL, USA. He was a recipient of the NSERC Postdoctoral Research Fellowship working with the Center for Applied Cryptographic Research and the Department of Combinatorics and Optimization, University of Waterloo, Waterloo, ON.

Prof. Azarderakhsh was the Guest Editor for IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING for the Special Issue of Emerging Embedded and Cyber Physical System Security Challenges and Innovations in 2016 and 2017. He was also the Guest Editor of *IEEE/ACM Transactions on Computational Biology and Bioinformatics* for the Special Issue on Security. He is serving as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS (TCAS-I).