 doi:10.20944/preprints202102.0336.v1

Article

Key Distribution for Post Quantum Cryptography using Physical Unclonable Functions

Bertrand Cambou*, Michael Gowanlock, Bahattin Yildiz, Dina Ghanaimiandoab, Kaitlyn Lee, Stefan Nelson, Christopher Philabaum, Alyssa Stenberg, and Jordan Wright

Northern Arizona University, Flagstaff, AZ86011, USA; bertrand.cambou; Michael.Gowanlock; Bahattin.yildiz; dg856; kdl222; swn34; cp723; ajs937; jaw566@nau.edu

* Correspondence: bertrand.cambou@nau.edu; +1-928-523-7824

Featured Application: Using PUFs, in support of networks secured with public key infrastructure, to generate on demand the key pairs needed for lattice and code PQC algorithms.

Abstract: Lattice and code cryptography can replace existing schemes such as Elliptic Curve Cryptography because of their resistance to quantum computers. In support of public key infrastructures, the distribution, validation and storage of the cryptographic keys is then more complex to handle longer keys. This paper describes practical ways to generate keys from physical unclonable functions, for both lattice and code based cryptography. Handshakes between client devices containing the PUFs and a server are used to select sets of addressable positions in the PUFs, from which streams of bits called seeds are generated on demand. The public and private cryptographic key pairs are computed from these seeds together with additional streams of random numbers. The method allows the server to independently validate the public key generated by the PUF, and act as a certificate authority in the network. Technologies such as High performance computing, and graphic processing units can further enhance security by preventing attackers to make this independent validation when only equipped with less powerful computers.

Keywords: Lattice cryptography; code cryptography; post quantum cryptography; physical unclonable function; public key infrastructure; high performance computing.

1. Introduction

In most public key infrastructure (PKI) schemes for applications such as cryptographic currencies, financial transactions, secure mails and wireless communications, the public keys are generated by private keys with RSA and elliptic curve cryptography (ECC). These private keys are natural numbers, typically 3000-bit long for RSA and 256-bits long for ECC. For example, in the case of ECC, the primitive element of the elliptic curve cyclic group is multiplied by the private key to find the public key. It is now anticipated that Quantum Computers (QC) will be able to break both RSA and ECC when the technology to manufacture enough quantum nodes becomes available. The paper entitled "A Riddle Wrapped in an Enigma" by N. Koblitz and A. J. Menezes suggested that the ban of RSA and ECC by National Security Agency is unavoidable, and that the risk of QC being is only one element of the problem [1]. Plans to develop post quantum cryptographic (PQC) schemes have been proposed to secure blockchains by Kiktenko et al. [2], and for cryptocurrency security by Semmouni et al. [3], even if the timeline for availability of powerful QC is highly speculative. Recently Campbell et al. [4], and Kampanakisy et al. [5], are proposing distributed ledger cryptography, and digital signatures with PQC.

In 2015, the National Institute of Standards and Technology (NIST) initiated a large scale program to standardize PQC algorithms. In 2019 the number of candidates was narrowed to 26, as part of phase two of the program [6]. In July 2020, NIST announced the selection of seven likely finalists for phase

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

three of the program [7]: CRYSTAL-KYLER, CRYSTAL-DILITHIUM, SABER, NTRU, and FALCON with lattice cryptography [8-12]; RAINBOW with multivariate cryptography [13], and Classic McEliece with code-based cryptography [14-15]. The software developed is mainly targeting DSA applications, as well as KEM. Lattice cryptography is relatively mature, well documented, and is most likely to become mainstream for cybersecurity.

One possible configuration using the PQC algorithms for PKI is to have each client device, or designate, generating the public-private key pairs, and to send the public key to a certificate authority (CA). This assumes that a separate authentication process is in place, and that each client device can securely store its key pair securely. The research question that is the subject of this paper is the feasibility of using physical unclonable functions (PUFs), together with a handshake process with the CA that generate new key pairs from the PUF at each transaction, thereby eliminating the need to store the key pair. Attempts to retrieve the secret keys are not relevant anymore as they are only used once. Such a configuration is raising several structural and technical questions. A secure enrollment process of each PUF needs to be established, and the CA has to store the challenges and reference values of each PUF. Such an infrastructure is already known when the security is based on secure hardware elements and tokens and require special protections against opponents. From a technical standpoint, it is questionable that the long key pairs necessary for PQC algorithms can be generated from physical elements. While a single bit mismatch is not acceptable for PQC algorithms, the natural drifts of PUFs over environmental conditions, and aging are real concerns that need to be addressed.

This paper is structured in the following way:

[Section 2]: The lattice and code-based cryptographic algorithms under consideration for standardization by NIST are presented. These algorithms are well documented, and the software stack written in C can be downloaded for IoT implementation. The schemes are based on the generation of random numbers, and the computation of public-private key pairs. The digital signature algorithms (DSA), and key encapsulation mechanisms (KEM) are not more complex to implement with PQC than with existing asymmetrical cryptographic schemes.

[Section 3]: We present some of the challenges and remedies associated with the use the PUF technology to secure PKI architectures driving networks of devices. The proposed architectures are based on existing asymmetrical cryptographic schemes, and commercially available PUFs. We present how the response based cryptographic (RBC) scheme can overcome the bit error rates (BER) that naturally occurs when cryptographic primitives are generated from physical elements. Finally, we present some hardware considerations in the implementation of PQC for PKI architectures, and the value of parallel computing.

[Section 4]: In this section, we propose schemes that use PUFs to generate the public-private key pairs for lattice and code-based cryptography. We show how the combination of random number generators, combined with the streams generated by the PUF can generate key pairs with relatively low error rates. We show how the error in these streams can be corrected using a search engine on the authenticating server.

[Section 5]: Finally, in the implementation and experimental section, we compare cryptographic schemes and algorithms. We analyze experimental results comparing the efficiency of RBC operating with various PQC schemes, ECC, and AES. As expected, asymmetrical schemes are slower than AES; however, the performance of the selected PQC algorithms is encouraging for the implementation of PUF-based architecture, using the RBC to handle the expected BER.

2. Lattice and code-based post quantum cryptography

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

Lattice-based algorithms exploit hardness to resolve problems such as the Closest Vector Problem (CVP), learning with error (LWE), and learning with rounding (LWR) algorithms and share some similarities with the knapsack cryptographic problem.

2.1. *Learning with Error cryptography*

The LWE of the CVP problem was first introduced by Regev [16]: the knowledge of integer-based vector t, and matrix A with $t = A.s_1$ cannot hide the vector s_1; however, the addition of a "small" vector of error s_2 with $t = A.s_1+s_2$, makes it hard to distinguish the vectors s_1 and s_2 from t. The vector s_2 needs to be small enough for the encryption/decryption cycles, but large enough to prevent a third party from uncovering the private key (s_1; s_2) from the public information (t; A). The public-private cryptographic key pair generation for client device i can be based on polynomial computations in a lattice ring, and is described in Fig1:

1- The generation of a first data stream called seed $a_{(i)}$ that is used for the key generation; in the case of LWE, the seed $a_{(i)}$ is shared openly in the network.
2- The generation of a second data stream called seed $b_{(i)}$ that is used to compute a second data stream for the private key $Sk_{(i)}$; the seed $b_{(i)}$ is kept secret.
3- The public key $Pk_{(i)}$ is computed from both data streams and is openly shared.
4- The matrix $A_{(i)}$ is generated from seed $a_{(i)}$.
5- The two vectors $s_{1(i)}$ and $s_{2(i)}$ are generated from seed $b_{(i)}$.
6- The vector $t_{(i)}$ is computed: $t_{(i)} \leftarrow A_{(i)}\ s_{1(i)} + s_{2(i)}$.
7- Both seed $a_{(i)}$ and $t_{(i)}$ become the public key $Pk_{(i)}$.
8- Both $s_{1(i)}$ and $s_{2(i)}$ become the private key $Sk_{(i)}$.

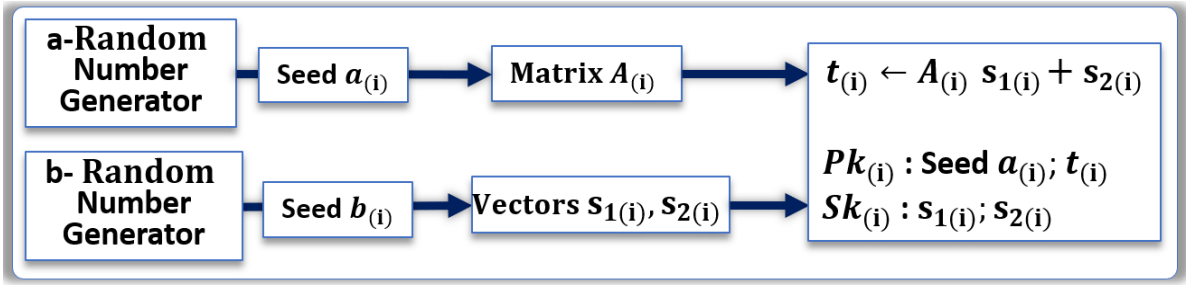

Figure 1: Example of public-private key generation for LWE based cryptography.

A digital signature algorithm (DSA) can be realized from the LWE instance by first generating a public-private key pair as in Figure 1. The secret key is then used to sign a message, and the public key is used to verify this signed message. In CRYSTALS-Dilithium [8], the authors use a Fiat-Shamir with Aborts approach [17] for their signing and verification procedure. The outline of the signing procedure is as follows:

1- Generate a masking vector of polynomials y.
2- Compute vector $A.y$ and set w_1 to be the high-order bits of the coefficients in this vector.
3- Create the challenge c, as the hash of the message and w_1.
4- Compute intermediate signature $z = y + c.s_1$.
5- Set parameter β to be the maximum coefficient of $c.s_1$.
6- If any coefficient of z is larger than $\gamma_1 - \beta$, then reject and restart at step 1.
7- If any coefficient of the low-order bits of $A.z - c.t$ is greater than $\gamma_2 - \beta$, then reject and restart at step 1.

Note: γ_1, γ_2, and β are set such that the expected number of repetitions is between 4 and 7.
The general outline of the verification procedure is given by the following:

Compute w_1' to be the high-order bits of $A.z - c.t$ and accept if all coefficients of z are less than $\gamma_1 - \beta$ and if c is the hash of the message and w_1'.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

Encapsulation allows for two parties to securely share a symmetric key by encapsulating the key in ciphertext. When both parties have the symmetric key, they are then able to use a symmetric-key encryption algorithm to communicate (e.g., AES). These algorithms are known as key encapsulation mechanisms (KEM) and a few examples from NIST are SABER [18], Classic McEliece [14-15], CRYSTALS KYBER [19], and NTRU [20]. The process of using encapsulation with LWE/LWR is described below:

- The public and private keys of both parties are constructed as described in Figure 1.
- Person A sends Person B their public key.
- Person B randomly generates a symmetric key and encapsulates it in a ciphertext with the public key of person A.
- Person B sends the ciphertext to person A.
- Person A decapsulates the ciphertext with their private key.

Both parties now have the symmetric key in their possession.

2.2. *Learning with Rounding cryptography*

The learning with rounding problem was first introduced by Banerjee [21]. It is the derandomized version of learning with error, which deterministically generates the noise in the LWE by rounding coefficients. This will eliminate the noise sampling, and significantly reduces the bandwidth [22]. The LWR is proved to be as hard as LWE to solve. Hence, it remains secure to be used in cryptographic applications. In schemes such as "Saber", a constant h is added as a constant vector to simulate the rounding operation by bit shifting, therefore playing a similar protecting role than the error vectors of LWE [18]. Saber, which is one of the NIST's finalists on key encapsulation category, uses LWR for key generation in public key encryption and key encapsulation. Below all three steps of PKE and KEM are described:

Saber PKE Key Generation

1- Similar to LWE, seed $a_{(i)}$ is used to generate matrix $A_{(i)}$.
2- Seed $b_{(i)}$ is used to generate vector $s_{(i)}$.
3- The vector $t_{(i)}$ is computed: $t_{(i)} \leftarrow A_{(i)} \,.\, s_{(i)} + h_{(i)}$.
4- Both seed $a_{(i)}$ and $t_{(i)}$ become the public key $Pk_{(i)}$.
5- $s_{(i)}$ becomes the private key $Sk_{(i)}$.

Saber PKE Encryption

1- The seed $a_{(i)}$ and $t_{(i)}$ is extracted from public key to encrypt the message m.
2- Matrix $A_{(i)}$ and vector $s'_{(i)}$ are generated.
3- The vector $t'_{(i)}$ is computed by rounding the product of $A_{(i)} \,.\, s'_{(i)}$: $t'_{(i)} \leftarrow A'_{(i)} \,.\, s'_{(i)} + h_{(i)}$.
4- Polynomial $v'_{(i)}$ is calculated as: $v'_{(i)} = t_{(i)} \,.\, s'_{(i)}$.
5- $v'_{(i)}$ is used to encrypt the message m which denoted as c_m.
6- ciphertext consists of c_m and $t'_{(i)}$.

Saber PKE Decryption

1- $v_{(i)}$ is calculated as: $v_{(i)} = t'_{(i)} \,.\, s_{(i)}$.
2- The message m' is decrypted by reversing computations with $v_{(i)}$ and c_m.

Saber key encapsulation mechanism has three steps of Saber KEM Key Generation, Saber KEM Encapsulation, Saber KEM Decapsulation:

Saber KEM Key Generation

1- Saber PKE key generation is used to return seed $a_{(i)}$, $t_{(i)}$ and $s_{(i)}$.
2- Both seed $a_{(i)}$ and $t_{(i)}$ become the Saber KEM public key $Pk_{(i)}$.
3- Hashed public key $Pkh_{(i)}$ is generated using SHA3-256.
4- Parameter z is randomly sampled.
5- z, $Pkh_{(i)}$ and $s_{(i)}$ become the Saber KEM secret key.

Saber KEM Encapsulation

1- Message m and public key $Pk_{(i)}$ is hashed using SHA3-256.
2- Saber PKE encryption is used to generate ciphertext.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

3- Hash of the $Pk_{(i)}$ and ciphertext are concatenated, then hashed to encapsulate the key.

Saber KEM Decapsulation

1- Message m' is decrypted by using Saber PKE Decryption.
2- Decrypted message m' and hashed public key $Pkh_{(i)}$ are hashed to generate K'.
3- ciphertext c'_m is generated from saber PKE Encryption for message m'.
4- If $c_m = c'_m$ then the $K = Hash(K',c)$, if not, $K = Hash(z,c)$.

2.3. *NTRU cryptography*

Cryptographic algorithms such as FALCON, which uses NTRU (***Nth*** degree of TRUncated polynomial ring) arithmetic is also based Lattice cryptography. The parameters of the scheme include a large prime number N, a large number **q** and a small number **p** that are both used for modulo arithmetic. Two numbers **df** and **dg** are used to truncate the polynomials $f_{(i)}$ and $g_{(i)}$. The key generation cycle for client device (i), as shown in Fig.2, is the following:

1- Generation of the two truncated polynomials $f_{(i)}$ and $g_{(i)}$ from seed $a_{(i)}$, and seed $b_{(i)}$.
2- Computation of $Fq_{(i)}$ which is the inverse of polynomial $f_{(i)}$ modulo **q**.
3- Computation of $Fp_{(i)}$, which is the inverse of polynomial $f_{(i)}$ modulo **p**.
4- Computation of polynomial $h_{(i)}$: $h_{(i)} \leftarrow p \,.\, Fq_{(i)} \,.\, g_{(i)}$.
5- The private key $Sk_{(i)}$ is $\{f_{(i)}; Fp_{(i)}\}$.
6- The public key $Pk_{(i)}$ is $h_{(i)}$.

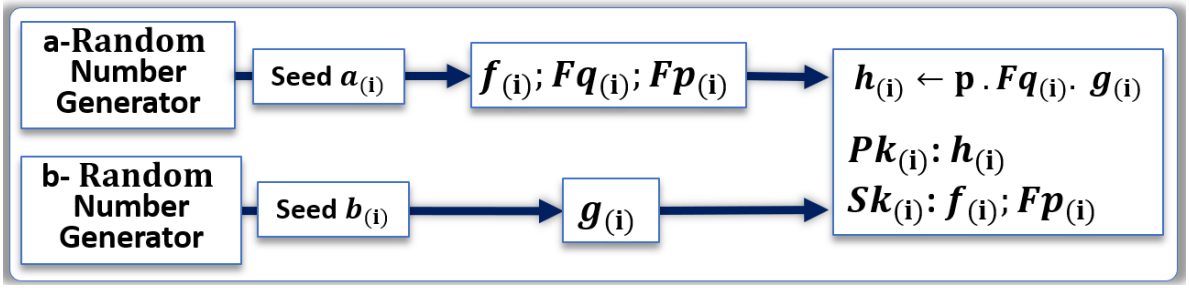

Figure 2: Example of public-private key generation for NTRU cryptography.

The polynomials $f_{(i)}$ and $g_{(i)}$ are not always usable, they are subject to some pre-conditions such as invertible modulo **p** and **q**. The client device needs to try several possible random numbers, and select the ones giving acceptable private keys. Once sufficient public and private keys are available, the encryption of the plaintext message m, $m \in \{-1, 0, 1\}^N$, is done by finding the random polynomial r, $r \in \{-1, 0, 1\}^N$ which uses a corresponding parameter d_r and calculating the ciphertext with the equation $e \equiv r.h + m \pmod q$. To retrieve m from e, we first calculate $a \equiv f.e \pmod q$, lift the coefficients of a to be between $\mp q/2$. Then, $a \pmod p$ is equal to m. [23].

NTRU lattices can also be applied to DSA. This was originally introduced in NTRUSign, but NIST submissions such as Falcon expand on these algorithms [9]. Falcon utilizes the GPV framework applied to NTRU lattices; that is, the public key is a long basis for an NTRU lattice while the private key is a short basis. From here, the message m is sent a non-lattice point C utilizing a random value salt and hash function H. Using the short basis, a user signs by finding the closest vector v to c. The signature is (salt, $s = c - v$), verified by checking if s is short and H (msg ‖ salt)- s is a point on the lattice (verified using the long basis [24]).

2.4. *Code-based cryptography*

Code based algorithms such as Classic McEliece are implemented with binary Goppa codes, that is Goppa codes with underlying computations in finite Galois fields **GF(2^m)**. The parameters are an irreducible polynomial of degree **t**, the field exponent **m**, and code length **n**. The resulting code has error-correction capability of **t** errors, the information-containing part of the code word has a size of **k** = **n** - **m** x **t** and has generator matrix **G** with a size of **k** x **n** [14-15].

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

The block diagram of Fig.3 is showing an example of public-private key generation for Code-based cryptography, and client device i.

1- Seed $a_{(i)}$ is used to create a random invertible binary **k** x **k** scrambling matrix $S_{(i)}$.
2- Seed $b_{(i)}$ is used to create a random **n** x **n** permutation matrix $P_{(i)}$.
3- The public key $Pk_{(i)} = \hat{G}_{(i)}$ is computed with the generator matrix G: $\hat{G}_{(i)} \leftarrow S_{(i)} \, . \, G \, . \, P_{(i)}$
4- The private key $Sk_{(i)}$ is $\{G; S_{(i)}^{-1}, P_{(i)}^{-1}\}$.

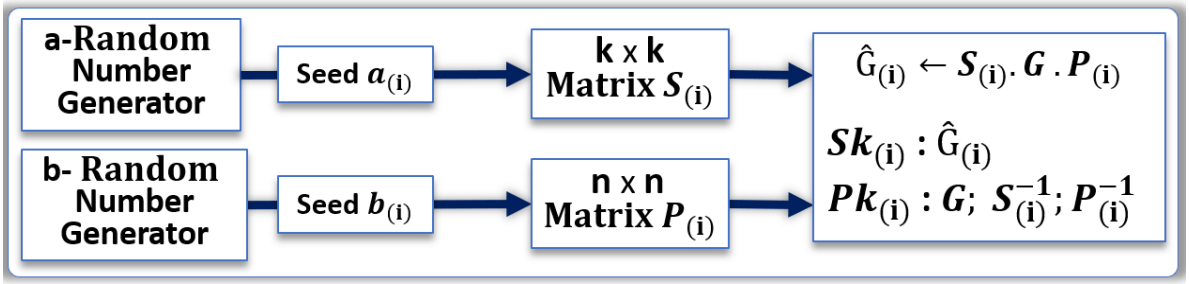

Figure 3: Example of public-private key generation for code-based cryptography.

Given a generator matrix of a binary Goppa code G, an irreducible polynomial of degree t, the field exponent m, and the code length n, the encryption process involves the following steps:

1- Create the public key, $\hat{G}_{(i)}$ as described above.
2- Multiply the message m by $\hat{G}_{(i)}$, creating the ciphertext message $\hat{m}$.
3- Add a random error vector e of Hamming weight t to $\hat{m}$ to obtain the ciphertext **c.**

Given a ciphertext c, a decoding algorithm, and the private key $\{G; S_{(i)}^{-1}, P_{(i)}^{-1}\}$, decryption involves the following steps:

1- Compute $\hat{c} = c \, P_{(i)}^{-1}$.
2- Use the decoding algorithm to correct the errors to obtain $\hat{m}$.
3- Obtain the original message by computing $m = \hat{m}.S_{(i)}^{-1}$.

One example of a decoding algorithm is Patterson's algorithm. This algorithm calculates the error-locator polynomial which has roots corresponding with the locations of the error bits added to the encrypted message. This algorithm can be implemented as follows [25]:

Input: Syndrome polynomial s, Goppa polynomial g of degree t

Patterson (s, g)

1- $t = s^{-1} \bmod g$.
2- $t = \sqrt{t + x}$.
3- Find polynomials a, b such that $b.t \equiv \bmod g$ with deg(a) ≤ ⌊t/2⌋ and deg(b) ≤ ⌊(t-1)/2⌋ using the extended Euclidean algorithm.
4- Calculate and return the error locator polynomial, $e = a^2 + x.b^2$.

Once the error locator polynomial is found, the Berlekamp Trace Algorithm can be used to find the roots of the polynomial via factorization. These correspond to the locations of the error bits added to the message. The Berlekamp Trace Algorithm can be implemented as follows [26]:

Input: Polynomial to factor p, trace polynomial t, basis index i

Berlekamp Trace (p, t, i):

1- if deg(p) ≤ 1
2- return the root of p.
3- $p_0 = gcd(p, t(B_i \cdot x))$.
4- $p_1 = gcd(p, 1 + t(B_i \cdot x))$.
5- return berlekampTrace(p_0, i+1), berlekampTrace(p_1, i+1).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

3. Public Key Infrastructure

3.1. Public-private key pairs

As part of a PKI, the public-private key pairs can be used to securely transmit shared secret keys though KEM, and to digitally sign messages with DSA, see Fig. 4. The public key $Pk_{(2)}$ of client 2 encapsulates the shared secret key of client 1, that can only be viewed by client (2), thanks to its private key $Sk_{(2)}$ that reverses the encapsulation. Client 1 uses its private key $Sk_{(1)}$ to digitally sign a message that is verified with the public key $Pk_{(1)}$, providing non-alteration and non-repudiation in the transaction. The trust and integrity of such architecture relies on the following:

i. The secure generation and distribution of the public-private key pairs to the client devices that are participating to the PKI.
ii. The identification of the client devices, and trust in their public keys.
iii. The sharing of the public keys among participants.

Most PKI's are relying on certificate authorities (CA) and registration authorities (RA) to offer such an environment of trust and integrity. The architecture is vulnerable to several threats, including loss of identity, man-in-the-middle attacks, and side channel attacks in which the private keys are exposed during KEM, and DSA.

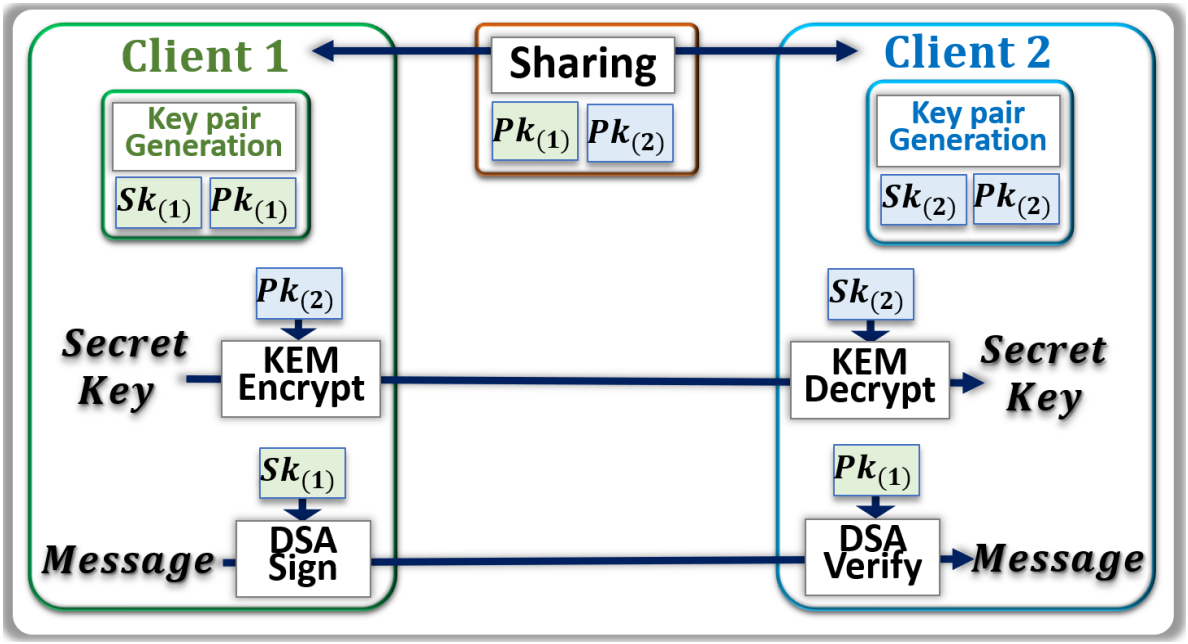

Figure 4: communication protocol between two client devices with shared public keys.

3.2. PKI with network of PUFs

The use of networks of PUFs can mitigate the vulnerabilities of PKI's. PUF technology exploits the variations created during fabrication to differentiate each device from all other devices, acting as a hardware "fingerprint" [27-29]. Solutions based on PUFs embedded in the hardware of each node can mitigate the risk of an opponent reading the keys stored in non-volatile memories. The keys for the PKI can be generated on demand, with one-time use, stealing a key become useless as new keys are needed at each transaction. During enrollment cycles, the images of the PUF's are stored in look-up tables in the CA, see Fig 5; enrollment has to be done only once in a secure environment. Handshake protocols [30] can select a portion of the PUFs, and their image stored in the CA, to extract a data stream that generates the key pairs. The PUFs can be erratic, therefore the generation cryptographic keys, the focus of this work, is challenging. A single-bit mismatch in a cryptographic key is not acceptable for most encryption protocols. Therefore, the use of error correcting (ECC) methods, helper data, and fuzzy extractors can minimize the levels of errors [31-33]. The alternate

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

method is the ones where the CA has search engines, such as response-based-cryptography (RBC), that can handle the validation of erratic keys [34-38].

The RBC engine that validates public keys shown in Figure 5 generates public/private key pairs until the public key matches the client's provided key. The server searches over a seed (e.g., a 256 bit seed), and uses that seed for key generation. If the generated public key matches the client's public key, then the client is authenticated. If the public keys do not match, then the server flips one bit of the seed at a time (increasing the Hamming distance) until the public keys match. Thus, the search is carried out by generating the public/private key pairs by iterating over the seed and increasing the Hamming distance until the seed is found that matches the client's public key. The search space for a 256 bit key is 2^{256} and would be nearly impossible to authenticate a user in a fixed time without the use of parallel computing. HPC and GPU technologies are valuable to enhance the ability of the CA to validate of the public key generated by the client devices. For instance, Graphics Processing Units (GPUs) can be employed to parallelize and accelerate the authentication process. By using a GPU, the server can search over multiple keys in parallel.

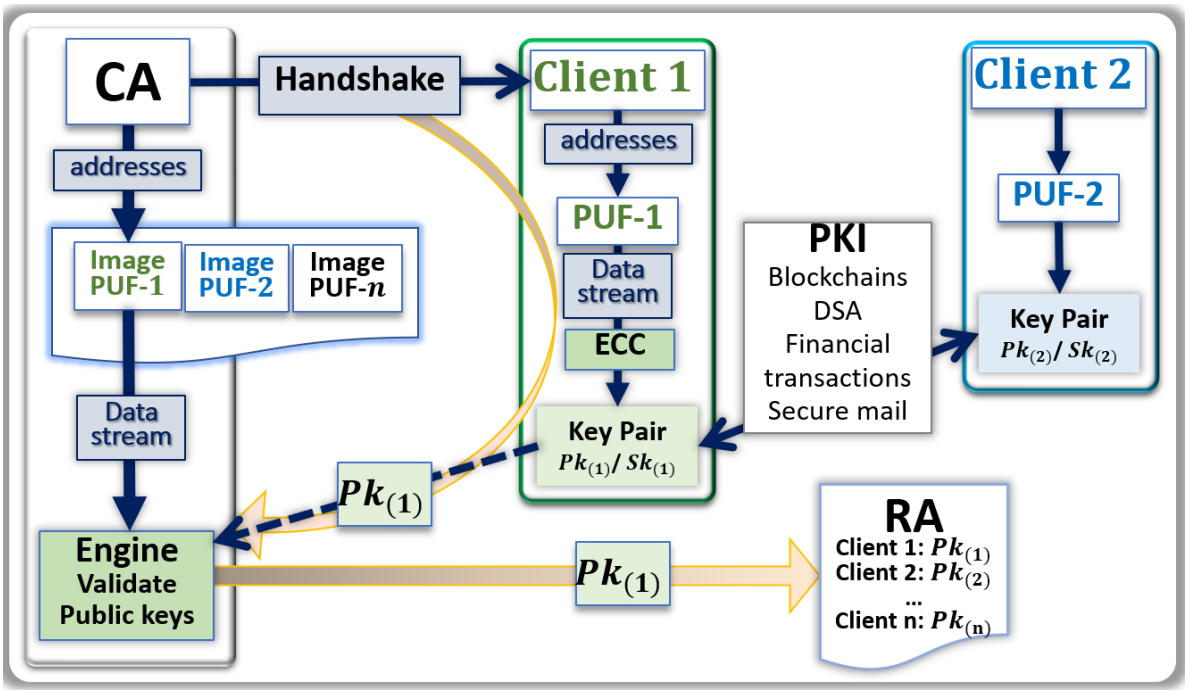

Figure 5: PUF-based public key infrastructure.

3.3. *Implementation of PQC algorithms for PKI.*

The CRYSTALS-Dilithium digital signature algorithm consists of the following procedures: key generation, signing, and verification. These procedures are computationally bounded by two operations: multiplication in the polynomial ring, $\mathbb{Z}_q[X]/(X_n+1)$, and matrix/vector expansion via an eXtendable Output Function (XOF). Therefore, any attempt to optimize Dilithium should target these operations. We describe below literature that focuses on such optimizations.

The operation of polynomial multiplication has a quasi-linear time complexity bounded by the Number Theoretic Transform (NTT) implementation, and the operation of expansion via XOF is bounded by the SHAKE-128 implementation. Using the AVX2 instruction set, matrix and vector expansion is optimized by using a vectorized SHAKE-128 implementation that operates on 4 sponges that can absorb and squeeze blocks in parallel. Additionally, Ducas et al. [8] use the AVX2 instruction set to optimize the NTT thus speeding up the polynomial ring multiplication by about a factor of 2. This optimization is achieved by interleaving the vector multiplications and Montgomery reductions so that parts of the multiplication latencies are hidden.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

Nejatollahi et al. [39] outline two different works that optimize the NTT using an Nvidia GPU. The first reports higher throughput polynomial multiplication [40] and the second is a performance evaluation between several versions of the NTT, including iterative NTT, parallel NTT, and CUDA-based FFT (cuFFT) for different polynomial sizes [41]. Strictly algorithmic optimizations of the NTT are presented in other works [42-43]. Longa et al. [42] show that limiting the coefficient length in polynomials to 32 bits yields an efficient modular reduction technique. By employing this new technique in NTT, reduction is only required after multiplication, and significant performance gains are achieved when compared to a baseline implementation. Additionally, the authors use signed integer arithmetic which decreases the number of add operations necessary in both sampling and polynomial multiplication. Greconici et al. [43] use signed integer arithmetic to decrease the number of add operations which leads to performance gains in several functions including NTT and SHAKE-128. The authors also employ a merging layers technique in NTT that reduces the number of loads and stores by about a factor of 2.

The SABER KEM algorithm is similarly computationally bounded by polynomial multiplication and hashing functions. As mentioned by D'Anvers et al. [18], since SABER uses power-of-2 moduli, this eliminates the need for rejection sampling and makes modular reduction fast by using bit shift operations. However, one drawback of using power-of-2 moduli is the inability to take advantage of faster NTT multiplication since the moduli are not prime. As described above, Akleylek et al. [41] examines the performance of different multiplication techniques. By implementing a verison of cuFFT in a similar fashion for SABER, we may observe a speedup in polynomial multiplication. In addition, SABER is computationally bounded by hashing and extendible functions. SABER uses SHA3_256 and SHA3_512 functions for hashing and SHAKE128 as an XOF. Roy et al. [44] demonstrate parallelizing SHAKE128 using AVX2 and batching four operations, thus achieving a 38% increase in throughput for SABER's key generation. Additionally, optimizing the hashing functions and SHAKE128 in a different way, the SABER technical documentation describes replacing the SHA3 functions with SHA2 and replacing SHAKE128 with AES in counter mode [18].

Focusing on three PQC algorithms, SABER, CRYSTALS-DILITHIUM, and NTRU, a breakdown of the fraction of time spent (as a percentage) in the hashing/XOR and polynomial multiplication components of the algorithms is reported in Table 1. NTRU spends the majority of its time doing polynomial multiplication first, then hashing second [45], but no benchmarks have been calculated thus far. The times spent for the hashing and polynomial multiplication components of CRYSTALS-Dilithium and SABER are reported as percentages of the total execution time for the key pair generation procedure where the percentages are an average of 10 time trials.

Table 1: Breakdown of the fraction of the time hashing/XOF and polynomial multiplication.

	Hashing/XOF	Polynomial Mult.	Reference
SABER	~30%	~60%	Our benchmarks
CRYSTALS-DILITHIUM	~42%	~33%	Our benchmarks
NTRU	2nd bottleneck	1st bottleneck	[40], [41]

4. PUF-based key distribution for PQC.

4.1. PUF-based key distribution for LWE lattice cryptography.

The proposed generic protocol to generate public-private key pairs with PUFs for LWE lattice cryptography is shown in Fig. 6. The random number generator (a) is used for the generation of seed $a_{(i)}$, which is public information. However, Seed k that is needed for the generation of the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

private key $Sk_{(i)}$ is generated from the PUF. The outline of a protocol generating a key pair for LWE cryptography is the following:

1- The CA uses random numbers generator and hash function to be able to point at a set of addresses in the image of the PUF-i.
2- From these addresses a stream of bits called Seed K' is generated by the CA.
3- The CA communicate to Client (i) through a handshake the instructions needed to find the same set of addresses in the PUF.
4- Client (i) uses the PUF to generate the stream of bits called Seed K. The two data streams Seed K and Seed K' are similar, however slightly differ from each other's due to natural physical variations and drifts occurring in PUFs.
[If needed Client (i) applies error correcting codes (ECC) to reduce the difference between Seed K and Seed K'; the corrected, or partially corrected, data stream is used to generate the vectors $s_{1(i)}$ and $s_{2(i)}$]
5- Client (i) independently uses a random numbers generator (a) to generate a second data stream Seed $a_{(i)}$, which is used for the computation of the matrix $A_{(i)}$.
6- The vector $t_{(i)}$ is computed: $t_{(i)} \leftarrow A_{(i)}\, s_{1(i)} + s_{2(i)}$.
7- The private key $Sk_{(i)}$ is $\{s_{1(i)}; s_{2(i)}\}$.
8- The public key $Pk_{(i)}$ is $\{a_{(i)}; t_{(i)}\}$.
9- Client (i) communicate to the CA through the network the public key $Pk_{(i)}$;
10- The CA uses a search engine to verify that $Pk_{(i)}$ is correct. The search engine initiates the validation by generating a public key from Seed $a_{(i)}$ and Seed K' with lattice cryptography codes. If the resulting public key is not $Pk_{(i)}$, an iteration process gradually injects errors into Seed K' and computes the corresponding public keys. The search converges when a match in the resulting public key is found, or when the CA concludes that the public key should be bad.
11- If the validation is positive the public key $Pk_{(i)}$ is posted online by the RA.

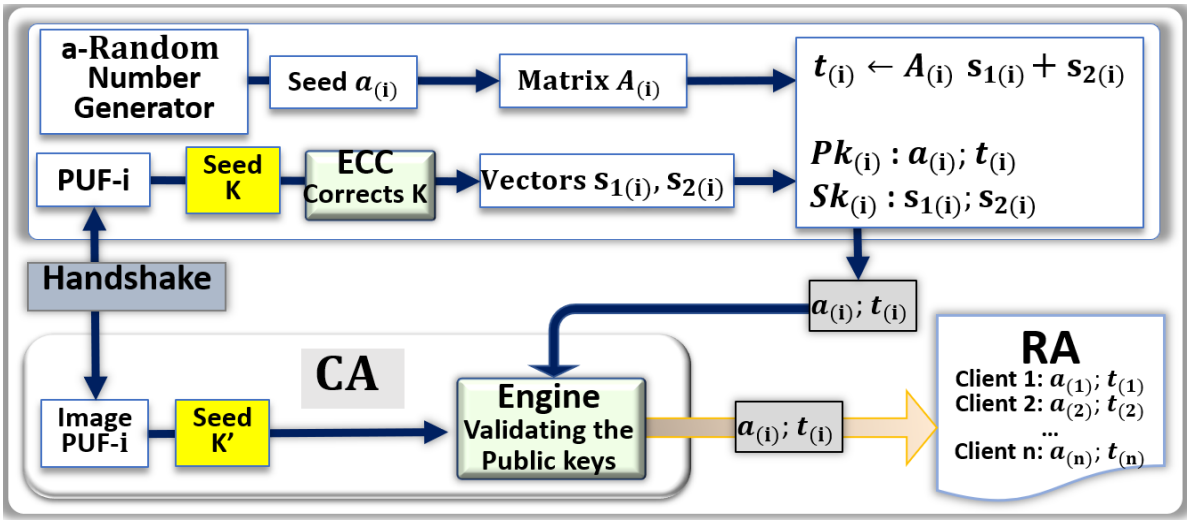

Figure 6: PUF-based key pair generation for LWE.

This protocol is applicable for single use key pairs that are generated for each transaction. The random number generators of the first step of the protocol can generate new data streams, which point at different portions of the PUFs, thereby can trigger the generation of new key pairs. The search engine described above can benefit from noise injection and high performance computing. The injection of noise in Seed K will make the search too difficult for CA, unless equipped with HPC's, or GPUs. This can preclude hostile CA's from participating.

4.2. PKI architecture with PUF-based key distribution and LW.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

The PUF-based key pair generation scheme with LWE cryptography, as presented in the previous section, can be integrated in a PKI securing a network of i clients. The example of Fig . 7, is showing two client devices communicating directly, either by exchanging secret keys through KEM or using DSA. The client devices independently generate the seed $a_{(i)}$, while the PUFs and their images are used for the independent generation of the vectors $s_{1(i)}$ and $s_{2(i)}$. The role of the CA is to check the validity of the vectors $t_{(i)}$, and to transmit both the seeds $a_{(i)}$ the vectors $t_{(i)}$ to the RA, which maintain a ledger with valid public keys. Such an architecture is secured assuming the following conditions:

i. The enrollment process in which the PUFs are characterized to generate their image is accurate and was not compromised by the opponent.
ii. The database stored in the CA that contains the image of the PUFs for the i client devices is protected from external, and internal attacks.
iii. The PUFs embedded in each client device are reliable, unclonable, and tamper resistant.
iv. The key generation process, KEM, and DSA are protected from side channel analysis.

As we experimentally verified that the latencies of the key generation process from the PUFs are low enough, such a protocol can be used to change the key pairs after each encryption cycles. Therefore, the potential loss of the secret keys during an encryption/decryption cycles has minimum impact as different keys will be used during the subsequent cycles.

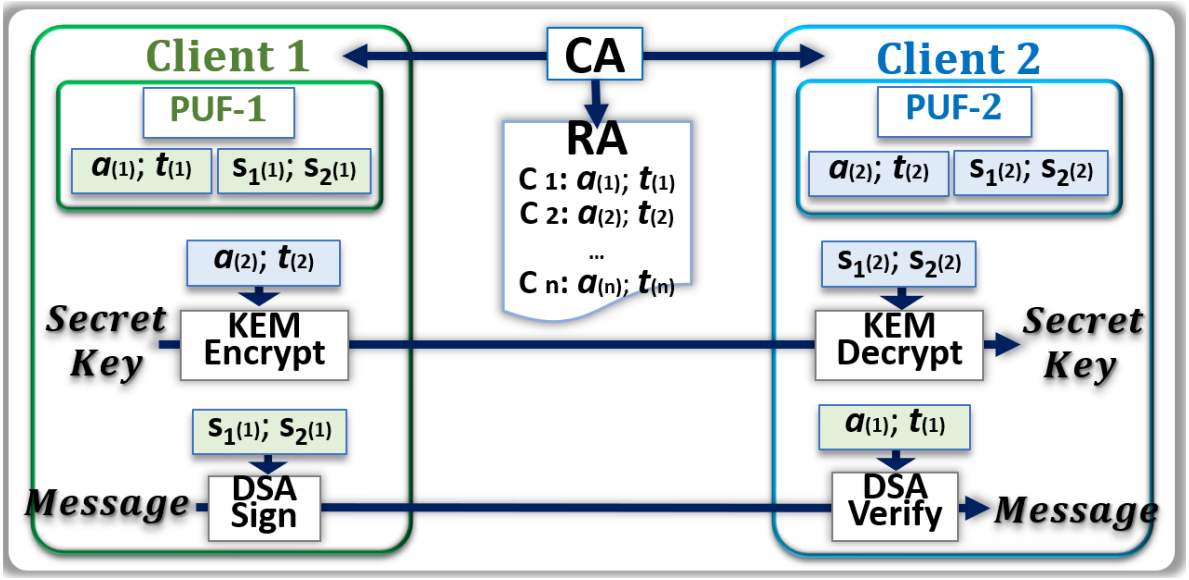

Figure 7: PUF-based PKI with LWE cryptography.

4.3. PUF-based key distribution for LWR lattice cryptography.

There are some similarities between LWE and LWR implementations. The seed k of the PUF is only used to generate one vectors $s_{1(i)}$, while a constant vector $h_{(i)}$ can be generated independently. The public vector $t_{(i)}$ is computed in a similar way: $t_{(i)} \leftarrow A_{(i)} \cdot s_{1(i)} + h_{(i)}$.

4.4. PUF-based key distribution for NTRU lattice cryptography.

The protocol to generate key pairs from PUFs for NTRU cryptography is similar than the one presented above in section 4.1 for LWE, see Fig. 8. We are suggesting a method where the only source of randomness is the PUF, seed **K**, to compute both the public key $Pk_{(i)}$, and the private key $Sk_{(i)}$. In our implementation, seed **K** feeds the hash function SHA-3, and SHAKE, to generate a long stream of bits, then compute the two polynomials $f_{(i)}$ and $g_{(i)}$.

As previously discussed in section 2.3, the polynomials $f_{(i)}$ and $g_{(i)}$ are not always usable due to pre-conditions, therefore a scheme to try several possible addressing of the PUF has to be developed. One way is to implement a deterministic method that is known by both the client device, and the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

CA, which can have a negative impact on the latencies. We preferred the solution driven by the client device that ask the CA to initiate new handshakes. The summary of the method used to generate the key pairs for NTRU cryptography is the following:

1- The CA uses random numbers to point at a set of addresses in the image of the **PUF-i**.
2- From these addresses a stream of bits called seed **K'** is generated by the CA.
3- The CA sent the handshake to the client (i) to find the same addresses.
4- Client (i) uses the PUF to generate seed **K**.
5- Client (i) applies ECC) on seed **K** and generates the truncated polynomials $f_{(i)}$ and $g_{(i)}$.
6- Computation of $Fp_{(i)}$ and $Fq_{(i)}$ and verify that the pre-conditions are fulfilled.
7- If needed ask for a new handshake and iterate.
8- The polynomial $h_{(i)}$ is computed: $h_{(i)} \leftarrow \mathbf{p} \,.\, Fq_{(i)} \,.\, g_{(i)}$.
9- The private key $Sk_{(i)}$ is $\{f_{(i)}; Fp_{(i)}\}$.
10- The public key $Pk_{(i)}$ is $h_{(i)}$.
11- Client (i) communicate to the CA through the network the public key $h_{(i)}$.
12- The CA uses a search engine to verify that $h_{(i)}$ is correct.
13- If the validation is positive the public key $h_{(i)}$ is posted online by the RA.

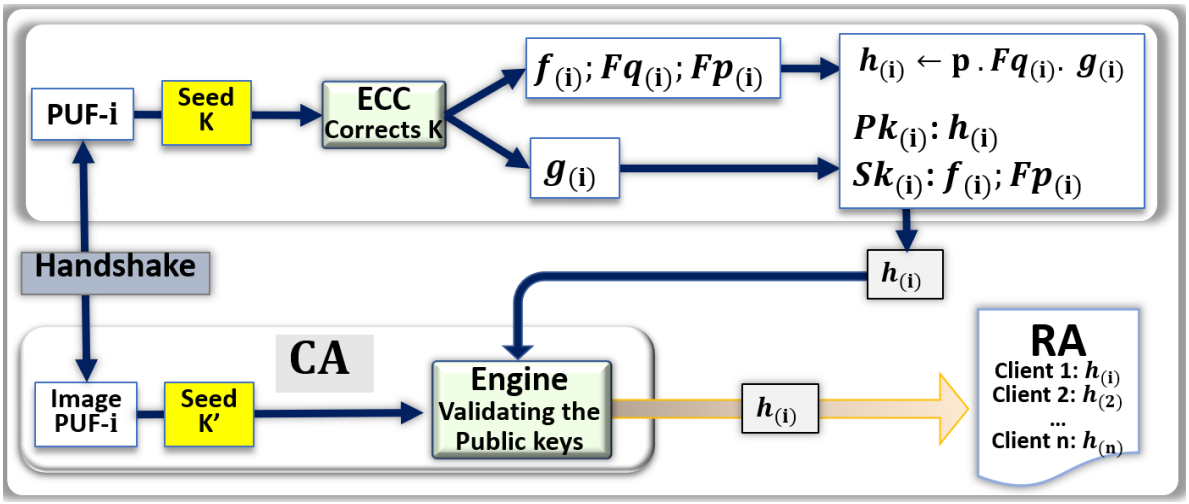

Figure 8: PUF-based key pair generation for NTRU.

4.5. PUF-based key distribution for code-based cryptography.

An example of a protocol to generate the key pairs with PUFs for Code-based cryptography is shown in Fig. 9. The overall protocol is similar to the one presented above for lattice cryptography. Like it is done with NTRU the only source of randomness is seed **K** that is generated from the PUF to compute the two matrices $S_{(i)}$ and $P_{(i)}$.

The brief outline of a protocol generating key pairs for code-based cryptography is the following:

1- The CA uses random numbers to point at a set of addresses in the image of the **PUF-i**.
2- From these addresses a stream of bits called seed **K'** is generated by the CA.
3- The CA sent the handshake to the client (i) to find the same addresses.
4- Client (i) uses the PUF to generate seed **K**.
5- Client (i) applies ECC) on seed **K** and generates the matrices $S_{(i)}$ and $P_{(i)}$.
6- Computation of $S_{(i)}^{-1}$ and $P_{(i)}^{-1}$.
7- The public key $Pk_{(i)} = \hat{G}_{(i)}$ is computed with the generator matrix G: $\hat{G}_{(i)} \leftarrow S_{(i)} \,.\, G \,.\, P_{(i)}$.
8- The private key $Sk_{(i)}$ is $\{G; S_{(i)}^{-1}, P_{(i)}^{-1}\}$.
9- Client (i) communicate to the CA through the network the public key $\hat{G}_{(i)}$.
10- The CA uses a search engine to verify that $\hat{G}_{(i)}$ is correct.
11- If the validation is positive the public key $\hat{G}_{(i)}$ is posted online by the RA.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

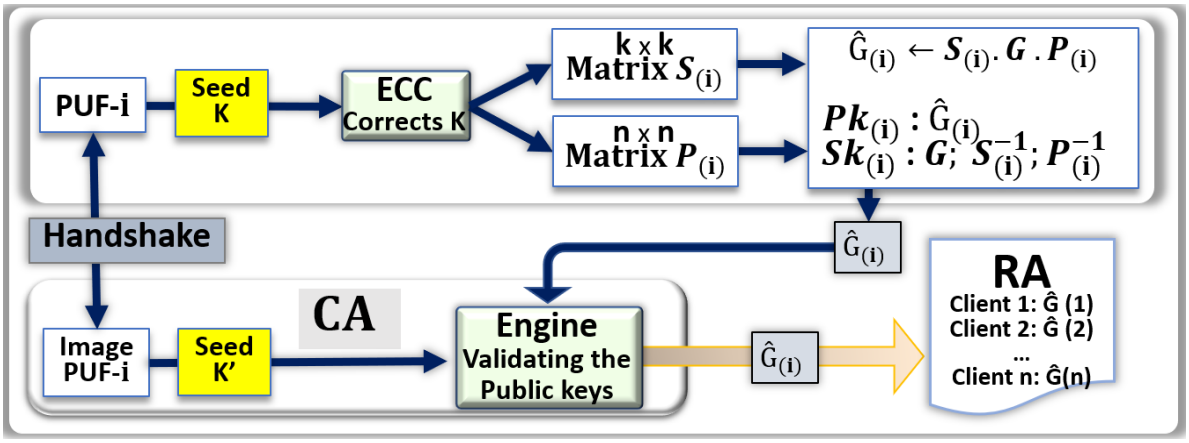

Figure 9: PUF-based key pair generation for code-based cryptography.

5. Experimental section

To demonstrate the principles and protocols discussed in sections 3 and 4, we designed a simple, small scale experiment. Five cryptosystems were selected as candidates for consideration: AES, ECC, qTESLA, CRYSTALS-Dilithium and SABER. However, each of these cryptosystems have a list of parameter sets as a part of their specifications. To narrow down the scope, we chose parameter sets that were inherently compatible with a 256-bit output from a hypothetical PUF as well as ones that were best optimized between speed, size, and security for IoT devices. PQC PKI cryptosystems that remained through NIST's round 3 were heavily favored, and a strong enough NIST security level was also an important consideration. For these reasons, the parameter sets AES256, ECC Secp256r1, qTESLA-p-I, CRYSTALS-Dilithium 2, and LightSABER were used as a performance comparison.

5.1. Experimental Methodology.

As of the time of writing, there are few implementations of RBC engines proposed. A previously established algorithmic approach targeted for HPC was chosen and scaled down to work using OpenMP instead of MPI for a single machine. This engine does not have to rely on the scalability of distributed memory nor the complications of MPI communication, so a flag in shared memory protected by critical sections was used instead. All implementations utilized the same overall structure and key iteration mechanism proposed in the distributed memory variant. The use of compilation with AVX was thusly carried over and maintained as a fair comparison between all variants; however, further optimizations can be made by taking advantage of AVX2 or other wide vector technologies. The AES256 implementation takes advantage of the AES-NI instruction set, whereas all other cryptosystems tested do not use any extra instruction sets over the vector-based ones such as AVX and SSE.

RBC engines targeting purely CPU platforms were only considered for demonstrative purposes. The purpose of this experiment is to compare the relativistic performance between all five chosen cryptosystems. The ease of porting one cryptosystem to another all on the CPU influenced the scope of experiments. Future experimental evaluations exploring GPU focus will require more dedicated, specialized programming for each cryptosystem. The CPU used for the experiments was an AMD Ryzen 9 3900X 12-Core CPU, with SMT (hyperthreading) and PBO (an opportunistic auto-overclocking) enabled.

5.2. Evaluation of the effective key throughput

For our performance metric used to compare the RBC cryptosystem implementations, we chose what we coin as the "effective key throughput." Given a small enough number of mismatches between the keys, overhead of the setup and clean up phases takes over. Thus, when computing

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

the key throughput (the number of keys searched per second), an insufficient number of mismatches will reflect inaccurately on maximum effective throughput possible. To combat this issue, we took each cryptosystem and iteratively found the minimum Hamming distance before the key throughput levels off. This is what we refer to as the "effective key throughput." For AES256, the minimum Hamming distance is 4, while for ECC Secp256r1, qTESLA-p-I, CRYSTALS-Dilithium 2, and LightSABER the minimum Hamming distance is 3. Unfortunately, due to the intractable nature of the problem, the single bit error jump from a Hamming distance of 3 to 4 makes it infeasible to run a statistically sufficient number of runs for ECC and qTESLA-p-I. For this reason, the AES256 benchmarks ran at a Hamming distance of 4, and the remaining cryptosystems ran at a Hamming distance of 3.

Similar to the distributed memory experiments, the experiments were running by randomly generating a key, then the combination of N bit difference that was sufficiently the middle point of a thread's workload was chosen to. This was done to reduce the need for a high number of iterations to reach a statistical central point. Thus, 10 iterations were performed for each cryptosystem, and the median was taken rom set of 10 iterations as the statistic point of interest to reduce the influence of background processes.

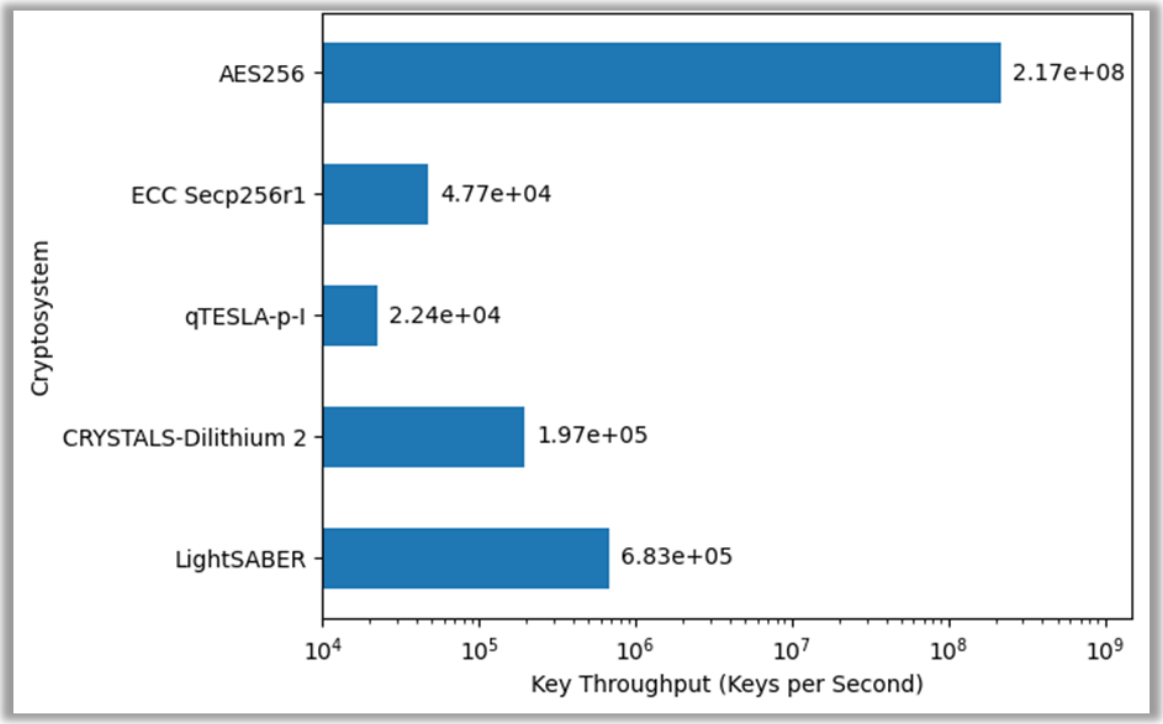

Figure 10: Maximum effective throughput (in keys per second) achieved for each RBC cryptosystem implementation with AMD Ryzen 9 3900X.

In Figure 10, the median of each RBC cryptosystem's effective key throughput is plotted against each other. The AES256 implementation, aided by AES-NI, runs several orders of magnitude more efficiently than the public key cryptography variants at 2.17 10^8 keys per second. ECC Secp256r1 performed the second slowest at 4.77 10^4 keys per second. The post-quantum cryptographies largely performed better than ECC with 1.97 10^5 and 6.83 10^5 keys per second for CRYSTALS-Dilithium 2 and LightSABER respectively. qTESLA-p-I was the worst performing PQC and overall cryptosystem out of all five at 2.24 10^4 keys per second. To be better get a sense of the relativistic scaling, we set ECC Secp256r1's effective key throughput results as the reference point since we are interested in how the PQC algorithms perform when replacing it in future PKI cryptosystems. This is plotted in Figure 11, where now the response variable is displayed in percentage of throughout relative to ECC Secp256r1's. Shown here, AES256 is roughly 4550 times more performant than ECC Secp256r1. CRYSTALS-Dilithium 2 is over 4.14 times more efficient

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 16 February 2021 doi:10.20944/preprints202102.0336.v1

than ECC Secp256r1. The most efficient PQC was LightSABER at 14.3 times faster, and the worst overall cryptosystem was qTESLA-p-I at 0.469 times slower.

From these results, we strongly advise against qTESLA when used in an RBC environment, only made worse by NIST's round 3 ruling to push this cryptography to the side. Out of what was tested, this leaves CRYSTALS-Dilithium as the strongest candidate for DSA in a PQC environment. For key encapsulation, our results show that SABER is a strong candidate for its relatively fast key generation. Future testing might consider comparing FALCON against CRYSTALS-Dilithium for DSA, and CRYSTALS-KYBER, NTRU, and Classic McEliece against SABER for KEM.

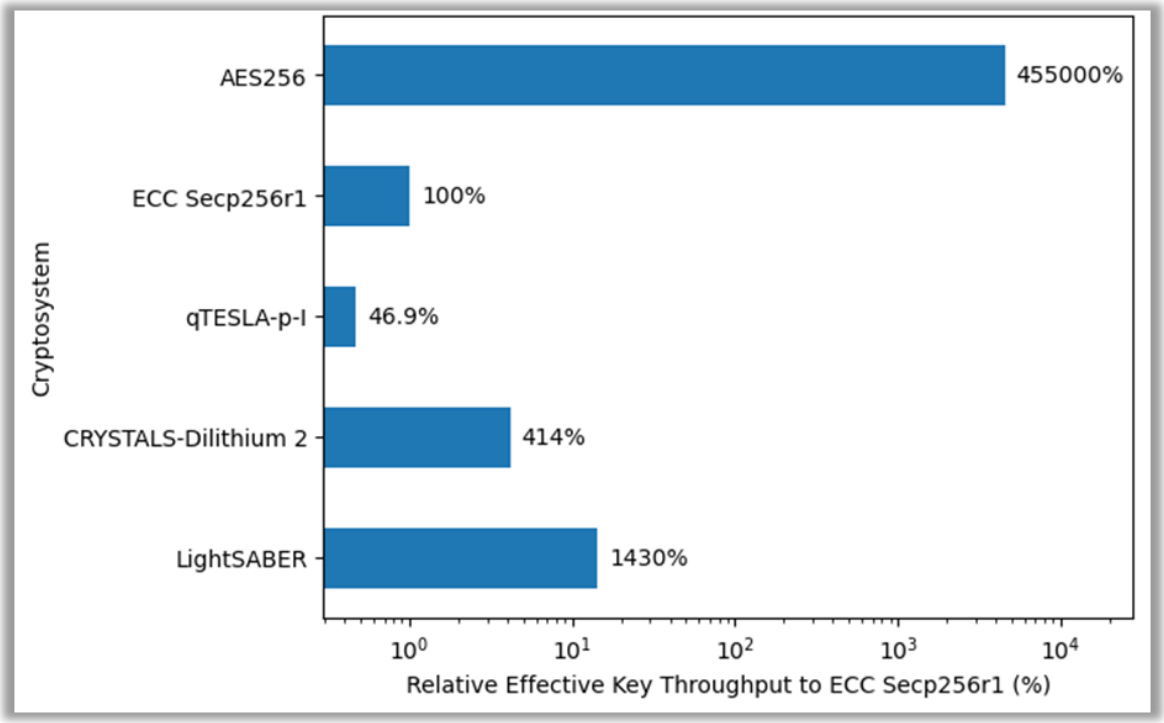

Figure 11: Maximum effective throughput relative to the performance of ECC Secp256r1 (as a percentage) achieved for each RBC cryptosystem implementation with AMD Ryzen 9 3900X.

6. Conclusion and future work

The PQC algorithms under standardization are encouraging, the latencies are reasonable, making the protocols suitable for PKIs securing networks of client devices and IoTs. The generation, distribution, and storage of the public-private key pairs for PQC can be complex because the keys are usually very long. This paper proposes to generate the public-private key pairs by replacing the random number generators by data streams generated from addressable PUFs to get the seeds needed in the PQC algorithms. Unlike the key pairs computed by PQC algorithms, the seeds are relatively short, typically 256-bit long. The use of PUFs as source of randomness is applicable to all five lattice-based codes under consideration in the phase III investigation of NIST, and to the code-based Classic McEliece scheme. In order to simultaneously generate key pairs from a server acting as the certificate authority, and the client device having access to its PUF, it is critical to handle the bit error rates (BERs) that are frequent with physical elements. We verified in the experimental section that the RBC can find the erratic seeds by testing an excess of 10^5 seeds per second with CRYSTAL DILITHIUM II and light SABER, which is faster than what we measured with mature algorithms such as the ones with elliptic curves.

In this work we have not yet studied the multivariate-based RAINBOW code, which is also an important scheme under consideration for standardization; we are currently studying ways to use

PUFs for key generation. The task needed to deploy PUF-based PQC solutions is not underestimated by the authors of this paper. This will include the use of highly reliable PUFs, and the optimization of the cryptographic protocol pointing simultaneously at the same set of addresses in the PUF, and in the look up table capturing the challenge-response pairs stored in the server. Further optimizing the PUF's protocols and the RBC for PQC algorithms is seen as an opportunity. The use of noises, nonces, errors, and rounding vectors can exploit the stochasticity of PUFs, and the ability to handle erratic streams of the RBC. The PQC algorithms analyzed in this paper can also benefit from the use of distributed memories, high performance computing, and parallel computing, which has the potential to further reduce the latencies of the RBC.

Author Contributions: Conceptualization, B. Cambou (BC).; methodology, BC, Michael Gowanlock (MG), Bahattin Yildiz (BY); software, MG, Dina Ghanaimiandoab (DG), Kaitlyn Lee (KL), Stefan Nelson (SN), Christopher Philabaum (CP), Alyssa Stenberg (AS), Jordan Wright (JW); validation, CP; formal analysis, MG, BY, DG, KL, SN, CP, AS, BC; investigation, MG, BY, DG, KL, SN, CP, AS, BC; resources, BC, MG, BY; data curation, MG, BY, DG, KL, SN, CP, AS, BC; writing—original draft preparation, BC, DG, KL, SN, CP, JW; writing—review and editing, BC, CP, MG; visualization, MG, BY, DG, KL, SN, CP, AS, BC; supervision, BC, MG, BY; project administration, BC.; funding acquisition, BC, MG. All authors have read and agreed to the published version of the manuscript.

Acknowledgments of support and disclaimer: (a) Contractor acknowledges Government's support in the publication of this paper. This material is partially based upon the work funded by the Information Directorate, under the Air Force Research Laboratory (AFRL); (b) Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFRL.

Acknowledgments: The authors are thanking the staff, students, and faculty from Northern Arizona University (NAU), in particular Brandon Salter who is software engineer in NAU's cybersecurity lab. We are also thanking the professionals of the Information Directorate of the Air Force Research laboratory (AFRL) of Rome, New York (US), who supported this effort.

Funding: This material is based upon the work funded by the Information Directorate under AFRL award number FA8750-19-2-0503.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Koblitz, N.; Menezes, A. A Riddle Wrapped in an Enigma"; http://eprint.iacr.org/2015/1018, 2015.
2. Kiktenko, E.; Pozhar, N.; Anufriev, M.; Trushechkin, A.; Yunusov, R.; Kurochkin, Y.; Lvovsky, A.; Fedorov, A. Quantum Secured Blockchains. Open Source, arXiv:1705.09258v3 [quant-ph], 2018.
3. Semmouni, M.; Nitaj, A.; Belkasmi, M. Bitcoin Security with Post Quantum Cryptography. https://hal-normandie-univ.archives-ouvertes.fr/hal-02320898, 2019.
4. Campbell, R. Evaluation of Post-Quantum Distributed Ledger Cryptography. Open Access, JBBA, Vol 2, https://doi.org/10.31585/jbba-2-1-(4)2019, 2019.
5. Kampanakisy, P.; Sikeridisz, D. Two Post-Quantum Signature Use-cases: Non-issues, Challenges and Potential Solutions. 7th ETSI/IQC Quantum Safe Cryptography Workshop, 2019.
6. Ding, J.; Chen, M-S; Petzoldt, A.; Schmidt, D.; Yang; B-Y. Rainbow. NIST PQC project round 2, documentation, 2019.
7. NIST status report of phase 3 of PQC program, NISTIR.8309, July 2020.
8. Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehlé, D. CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation. Part of the round 3 submission package to NIST. https://pq-crystals.org/dilithium, 2019.
9. Fouque, P-A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Prest, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. NIST PQC project round 2, documentation, 2019.
10. Peikert, C.; Pepin Z. Algebraically Structured LWE Revisited. *Theory of Cryptography. Conf.* (Springer), pp. 1-23. https://doi.org/10.1007/978-3-030-36030-6_1. 2019.

11. IEEE computing society. IEEE Standard 1363.1-2008 - Specification for Public Key Cryptographic Techniques Based on Hard Problems over Lattices (IEEE, Piscataway, New Jersey, United States). https://doi.org/ 10.1109/IEEESTD.2009.4800404 , Mar. 10, 2009.
12. Regev, O. New lattice-based cryptographic constructions, Journal of the ACM, 51(6): 899-942. https://doi.org/10.1145/1039488.1039490, 2004.
13. Casanova, A.; Faugere, J.-C.; Macario-Rat, G.; Patarin, J.; Perret, L.; Ryckeghem, J. GeMSS: A Great Multivariate Short Signature. NIST PQC project round 2, documentation, 2019.
14. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory. In: DSN Prog. Rep., Jet Prop. Lab., California Inst. Technol., Pasadena, CA, pp. 114-116, January 1978.
15. Biswas B., Sendrier N. McEliece Cryptosystem Implementation: Theory and Practice. In: Buchmann J., Ding J. (eds) Post-Quantum Cryptography. PQCrypto. Lecture Notes in Computer Science, vol 5299. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-88403-3_4, 2008.
16. Regev, O.; On lattices, learning with errors, random linear codes, and cryptography. In: STOC '05. pp. 84–93. ACM, http://doi.acm.org/10.1145/1060590.1060603, 2005.
17. Lyubashevsky, V. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In ASIACRYPT, pages 598–616, 2009. 1, 2, 5, 13, 20.
18. D´Anvers, J-P.; Karmakar, A.; Roy, S.; Vercauteren, F. Saber: Module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. Cryptology ePrint Archive, Report 2018/230, 2018, https://eprint.iacr.org/2018/230, Africacrypt 2018.
19. Bos, J.; Ducas, L.; Kiltz, E.; Lepoint, T.; Lyubashevsky, V.; Schanck, J.; Schwabe, P.; Seiler, G.; Stehle, D. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. 353-367. 10.1109/EuroSP.2018.00032.
20. Hülsing, A.; Rijneveld, J.; Schanck, J.; Schwabe, P. High-speed key encapsulation from NTRU. IACR Cryptol. ePrint Arch., 2017, 667.
21. Banerjee, A.; Peikert, C; Rosen, A. Pseudorandom functions and lattices. Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 719-737. Springer, Berlin, Heidelberg. April 2012.
22. Alwen, J.; Stephan K.; Krzysztof P.; Daniel W. Learning with rounding, revisited. Annual Cryptology Conference, pp. 57-74. Springer, Berlin, Heidelberg, 2013.
23. Nurshamimi, S.; Kamarulhaili. H. NTRU Public-Key Cryptosystem and Its Variants: An Overview. International Journal of Cryptology Research, vol. 10(1), p. 21, 2020.
24. Gentry, C.; Peikert, C.; Vaikuntanathan, V. How to Use a Short Basis: Trapdoors for Hard Lattices and New Cryptographic Constructions. https://eprint.iacr.org/2007/432.pdf. Aug 25, 2008.
25. Heyse, S. Post-quantum cryptography: Implementing alternative public key schemes on embedded devices, For the degree of Doktor-Ingenieur of the Faculty of Electrical Engineering and Information Technology at the Ruhr-University Bochum, Germany, October 8, 2013.
26. Menezes A.; van Oorschot P; Vanstone S. Some computational aspects of root finding in GF(q^m). https://doi.org/10.1007/3-540-51084-2_24, Lecture Notes in Computer Science, vol 358. Springer, Berlin, Heidelberg. 1989.
27. Papakonstantinou, I.; Sklavos, N. Physical Unclonable Function Design Technologies: Advantages & Tradeoffs. Computer and Network Security, editor Kevin Daimi, Spingler, ISBN: 978-3-319-58423-2, 2018.
28. Herder, C.; Yu, M.; Koushanfar, F. 2014. Physical Unclonable Functions and Applications: A Tutorial. Proceedings of the IEEE, vol. 102, no. 8, pp. 1126-1141. 2014.
29. Cambou, B., and M. Orlowski. 2016. "Design of Physical Unclonable Functions with ReRAM and ternary states". Cyber and Information Security Research Conference, CISR-2016, Oak Ridge, TN, USA.
30. Cambou, B., and D. Telesca. 2018. "Ternary Computing to Strengthen Information Assurance, Development of Ternary State based public key exchange'. IEEE, SAI-2018, Computing Conference, London, UK.
31. Taniguchi, M., M. Shiozaki, H. Kubo and T. Fujino. 2013. "A stable key generation from PUF responses with a Fuzzy Extractor for cryptographic authentications". In IEEE 2nd Global Conference on Consumer Electronics (GCCE), Tokyo, Japan.
32. Kang, H., Y. Hori, T. Katashita, M. Hagiwara and K. Iwamura. 2014. "Cryptographie key generation from PUF data using efficient fuzzy extractors". In 16th International Conference on Advanced Communication Technology, Pyeongchang, Korea.

33. Delvaux, J., D. Gu, D. Schellekens and I. Verbauwhede. 2015. "Helper Data Algorithms for PUF-Based Key Generation: Overview and Analysis". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 6, pp. 889-902.
34. Cambou, C. Philabaum, D. Booher, D. Telesca; "Response-Based Cryptographic Methods with Ternary Physical Unclonable Functions"; 2019 SAI FICC, IEEE; March 2019.
35. B. Cambou, C. Philabaum, D. Duane Booher; Response-based Cryptography with PUFs, NAU case D2018-049, Jun 2018.
36. B. Cambou; " Unequally powered Cryptograpgy with PUFs for networks of IoTs"; IEEE Spring Simulation Conference, May 2019.
37. B. Cambou, C. Philabaum, and D. Booher; "Replacing error correction by key fragmentation and search engines to generate error-free cryptographic keys from PUFs", CryptArchi2019, June 2019.
38. B. Cambou; M. Mohammadi; C. Philabaum and D. Booher; Statistical Analysis to Optimize the Generation of Cryptographic Keys from Physical Unclonable Functions; SAI computing conference, IEEE, 2020.
39. Nejatollahi, H.; Dutt, N.; Ray, S.; Regazzoni, F.; Banerjee, I.; Cammarota. R. Post-Quantum Lattice-Based Cryptography Implementations: A Survey. ACM Comput. Surv. 51, 6, Article 129, February 2019. DOI: https://doi.org/10.1145/3292548.
40. Emeliyanenko, P.Efficient multiplication of polynomials on graphics hardware. In APPT 09, proceeding 8th International Symposium on Advanced Parallel Processing Technologies, August 2009 Pages 134–149 https://doi.org/10.1007/978-3-642-03644-6_11.
41. Akleylek, S.; Dağdelen, Ö.; Tok Y. On the efficiency of polynomial multiplication for lattice-based cryptography on gpus using cuda. In: Cryptography and Information Security in the Balkans, Springer International Publishing, 2016.
42. Longa, P.; Naehrig, M. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. DOI: 124-139. 10.1007/978-3-319-48965-0_8. Comp. Science, Mathematics. IACR 2016.
43. Greconici, D.; Kannwischer, M.; Sprenkels, D. Compact Dilithium Implementations on Cortex-M3 and Cortex-M4. IACR Cryptol. ePrint Arch. 2020: 1278. 2020.
44. Roy, S. SaberX4: High-Throughput Software Implementation of Saber Key Encapsulation Mechanism. In 37th IEEE International Conference on Computer Design, ICCD 2019, Abu Dhabi, United Arab Emirates, pages 321–324, November 17-20, 2019.
45. Farahmand, F.; Sharif, M.; Briggs, K.; Gaj, K. A High-Speed Constant-Time Hardware Implementation of NTRUEncrypt SVES. International Conference on Field-Programmable Technology (FPT), Naha, Okinawa, Japan, pp. 190-197, DOI: 10.1109/FPT.2018.00036. 2018.