Post-quantum cryptographic key distribution for autonomous systems operating in contested areas

Conference Paper · June 2023

DOI: 10.1117/12.2663235

4 authors:

Michael Partridge
Northern Arizona University
5 PUBLICATIONS **57** CITATIONS
SEE PROFILE

Saloni Jain
Northern Arizona University
18 PUBLICATIONS **110** CITATIONS
SEE PROFILE

Michael Garrett
Northern Arizona University
3 PUBLICATIONS **30** CITATIONS
SEE PROFILE

Bertrand Cambou
Northern Arizona University
100 PUBLICATIONS **791** CITATIONS
SEE PROFILE

Post-quantum cryptographic key distribution for autonomous systems operating in contested areas

Michael Partridge, Saloni Jain, Michael Garrett, Bertrand Cambou

Michael Partridge, Saloni Jain, Michael Garrett, Bertrand Cambou, "Post-quantum cryptographic key distribution for autonomous systems operating in contested areas," Proc. SPIE 12540, Autonomous Systems: Sensors, Processing, and Security for Ground, Air, Sea, and Space Vehicles and Infrastructure 2023, 125400D (13 June 2023); doi: 10.1117/12.2663235

Event: SPIE Defense + Commercial Sensing, 2023, Orlando, Florida, United States

Post-quantum cryptographic key distribution for autonomous systems operating in contested areas

Michael Partridge, Saloni Jain, Michael Garrett, and Bertrand Cambou

School of Informatics Computing and Cyber Systems, 1295 S. Knoles Dr., Flagstaff AZ 86011, USA

ABSTRACT

The deployment of post-quantum cryptographic algorithms currently under standardization by NIST have the potential to mitigate quantum computer based attacks, which present a whole new range of cybersecurity challenges. We recognize post-quantum cryptography (PQC) key distribution, across autonomous systems operating in contested areas, as a particularly difficult problem to solve. Verifying the origin and integrity of mission instructions is of extreme importance, especially in the context of autonomous vehicles, missiles, and drones; the lack of which can enable opponents to leverage attacks effecting catastrophic damage to life, property, and strategy.

This work proposes a protocol for controlled key distribution through the use of PQC algorithms, strengthened by fingerprints of embedded devices. With this framework, we can assure the origin and correctness of digital information sent to autonomous systems, and enforce that only authorized members can send instructions to these devices. This protocol replaces the deterministic, pseudo-random seed in these PQC protocols with a fingerprint derived from the physical disorder of the embedded device to enhance security and maintain the integrity of the system. We demonstrate how this system can be used to send instructions, verify the origin of the instructions upon receipt, and certify the integrity of the instruction packet received. We also discuss situations in which an attacker attempts to falsify information and how to detect that malicious action. We measure the quality and performance of this prototype system by measuring the latency and bit error rates. We demonstrate this prototype system using the CRYSTALS-DILITHIUM digital signature algorithm (DSA).

Keywords: post-quantum cryptography (PQC), key distribution, fingerprint, data integrity, dilithium, autonomous systems, digital signature algorithm (DSA), cryptography, physical unclonable function (PUF)

1. INTRODUCTION

We may not know when cryptographically-relevant quantum computers will come, but we do know it took about two decades to deploy our current public key infrastructure.[1] With that timeline in mind, billions being invested worldwide, and significant investments being made by our adversaries,[2] winning the race to deploying (not just establishing) post quantum cryptography (PQC) algorithms before the arrival of quantum computers is a matter of national security.[3]

Stepping up to the task on the establishment front, NIST is hopeful that their standardization process, which started in 2016,[1] will complete by 2024.[4] It is important to progress research of such systems, that work on conventional computers, yet can withstand quantum computer attacks. With this crucial process almost behind us, we must turn attention toward secure deployment of such systems for real-world use.

In doing so, it should be considered that quantum computers not only threaten cryptographic algorithms, but also, the pseudo-random number generators (PRNGs) these algorithms use and depend on.[5] The algorithms being evaluated by NIST fall into two categories: key encapsulation methods (KEMs), and digital signature algorithms (DSAs). The contending algorithms all share a common notion of a seed, which is created, thus far, by PRNGs.[6] Thereby, we identify the PRNG as the weakest link of these hardened algorithms.

Further author information: (Send correspondence to M.P.)
M.P.: E-mail: mcp292@nau.edu, Telephone: +1 928 523 0101

Autonomous Systems: Sensors, Processing and Security for Ground, Air, Sea, and Space Vehicles and Infrastructure 2023,
edited by Michael C. Dudzik, Stephen M. Jameson, Theresa J. Axenson, Proc. of SPIE
Vol. 12540, 125400D · © 2023 SPIE · 0277-786X · doi: 10.1117/12.2663235

To address this weakness, we propose replacing the PRNG in PQC algorithms with a physical unclonable function (PUF), whereby fingerprints are generated and then used to create the cryptographic seed.

By replacing the PRNG with a PUF, we also afford the opportunity to address the deployment issue, and propose a modifiable, modular framework for real-time key distribution across autonomous assets in contested areas (see Section 2.1).

1.1 Contributions

By replacing the PRNG of PQC algorithms with a PUF, we aim to address an identified weakness of current PQC algorithms, while simultaneously providing a mechanism for real-time key distribution in contested areas. We propose a distribution framework, and test the quality, reliability, and speed of this system using a proof-of-concept system. We also prescribe alternative module substitutions and potential improvements for this system.

2. DESIGN

2.1 System Overview

This work proposes a system that aims to strengthen the security of PQC algorithms by replacing the PRNG used for seed creation, and thereby, simultaneously afford a mechanism for trusted registration and real-time key distribution in contested areas. We will restrict the conversation to the context of digital signature algorithms (DSAs) for the remainder of this paper. That said, this system is completely modular, allowing for the PUF, challenge-response mechanism, error correction scheme, and PQC algorithm to be switched out as the administrator finds necessary. This overview is given in general terms and takes for granted additional security measures and fine-details in order to prioritize understanding of the framework itself. The concerns generated should be addressed by the following sections.

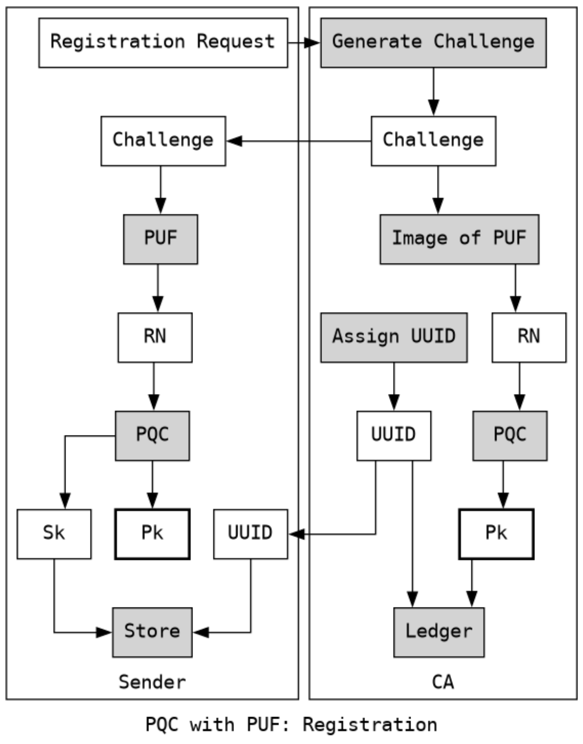


Figure 1. A diagram depicting the registration sequence between the registrant (sender) and CA. Using the challenge provided by the CA, the sender should arrive at the same public key (Pk).

In order to address deployment, we first present a simplified scenario: Consider real-time deployment of public/private key pairs, in contested areas using a server-client model. In this server-client model, the server acts as a certificate authority (CA) and keeps a ledger of all registered users and their corresponding public keys. Using a PRNG, the server will likely issue slots on the ledger on a first-come, first-serve basis (i.e., if I claim I am client A first, I am granted that spot on the ledger). If that spot is later contested, what information does the server have to settle the dispute?

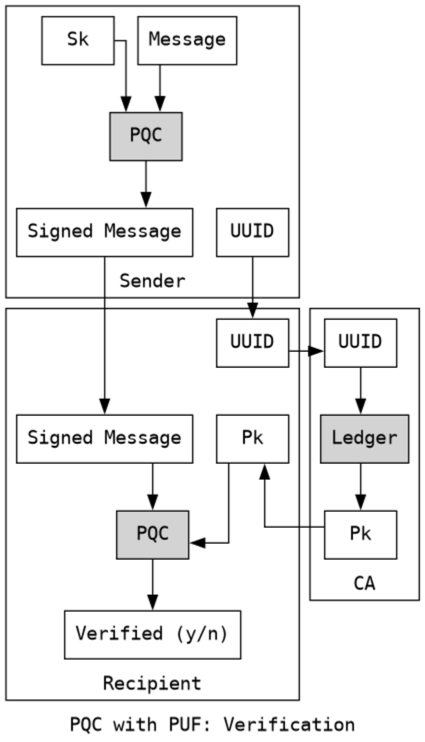


Figure 2. A diagram depicting the verification sequence between the sender, recipient, and CA. The sender signs the message with their secret/private key (Sk) and sends to recipient along with their UUID. Recipient requests the sender's public key (Pk) from the CA using the UUID, and uses the gathered information to verify the message.

By using a PUF, the server will send a challenge at the time of registration, and only issue slots on the ledger based on the PUF response. If the response deviates considerably from the expectation for that challenge, registration is denied. This protects valid users from "identity-theft", in this context, and ensures only qualified users can participate in the system.

To describe this system, we will use three parties: a sender, a certificate authority (CA), and a recipient. In the context of DSAs, there are two main actions to consider: registration and message verification.

For registration (see Fig. 1), the potential sender makes a request to register with the CA. The CA sends a challenge to the PUF, and using an enrollment of the sender's PUF, computes the expected response. The sender applies the received challenge to its PUF, generates a response, and sends it to the CA for comparison. The CA corrects for potential error in the received key, and registers the user if all requirements are satisfied.

For message verification (see Fig. 2), the client receives a signed message from a user who is claiming to be user A. The client sends a request to the CA for user A's public key. Using that public key, the signed message, and the PQC DSA algorithm, the recipient verifies that the message was signed by the claimed sender.

Most security protocols assume that the client has the capability and know-how to securely store secret information. Using a PUF in combination with more aggressive error prevention can eliminate the need to put that responsibility on the user.[6] Instead, the user would store the less sensitive challenge, and generate the secret key on-demand.

Equipping assets with an enrolled PUF before sending them out into the field, enables live deployment and modification of the system described herein, and also affords the opportunity for later deployment of PUF-secured systems developed in the future.

2.2 TAPKI

TAPKI (ternary addressable public key infrastructure) is a challenge-response protocol our lab has developed.[7,8] It is modular in the sense that it can and has been adapted for use with many different PUF technologies.[9–13] More information on TAPKI can be found in the aforementioned citations. Focus is given here only to the components that contribute directly to understanding the system and usage in the scope of this work. We'd like to emphasize that this is one of many challenge-response protocols that can be used in the framework we present.

This proof-of-concept prototype was developed using our pre-formed, ReRAM PUF. We have previously demonstrated this wafer-level technology to be suitable for use as true random number generators (TRNGs) in Cambou et al.[14] We have since developed a working dev-kit, that uses the same technology, but in a more commercial package, which we used for this study. Details on the hardware can be found in Riggs et al.[15]

To generate keys from our ReRAM PUFs we employ the TAPKI cell-pairing protocol. In short, the ReRAM cell-pairing protocol uses two ReRAM chips, randomly addresses each chip producing two resistances, and compares those resistances to get a bit (`r1 > r2`). The addresses used to generate a single bit are called an address pair. If we want a 256-bit key, we would need 256 address pairs.

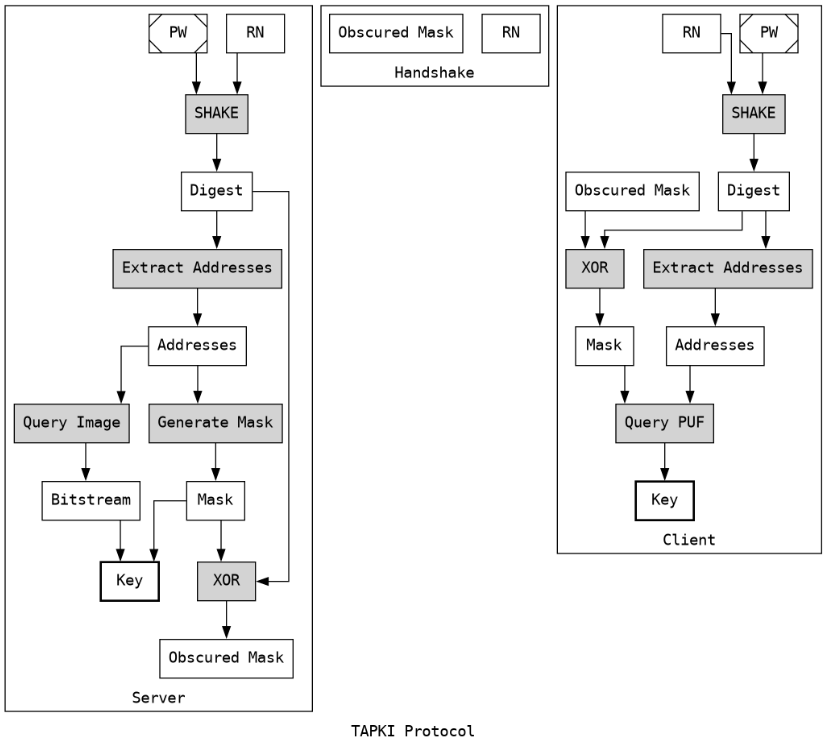


Figure 3. A diagram depicting the fundamentals of the TAPKI protocol. After generating the key and obscuring the mask, the server sends the handshake to the client, at which point the client begins its sequence.

In a server-client model, the server would take an enrollment (also called "image") of the PUF, storing the resistances of each address, of each chip, at different temperatures and currents. When the server wants to authenticate a client (see Figure 3), the server generates a random number, combines it with a password shared between server and client, feeds the combination to a extended output function (XOF), and extracts address pairs from the digest. Using the extracted address pairs and the enrollment, the server compares resistances and produces a key. The server then sends that random number over to the client. The client uses the shared password and this random number, following the same steps as the server, with the only difference being that the address pairs are used to query a PUF, not an enrollment database. Authentication is complete if the keys match.

There is an additional step in the process above that is used to reduce error. Error is defined as how many bits of the client key deviate from the server key. When using the XOF to generate addresses, the server will generate a number of addresses more than it needs. For example, if we want a 256-bit key, we only need 256 address pairs, but we might choose to generate 384 address pairs. This would give us a "buffer" of 128 address pairs that we can selectively "mask" out. Because we have excess, we can cherry pick pairs according to various criterion of our choosing. In the context of key generation, we want stability, and so, will cherry pick pairs that have the furthest differences, making it less likely for slight deviations to flip the relationship and change the output bit. We also use the mask to ensure each key contains 128 zeros and 128 ones. For security, the client device stores no information that would help make a decision as to which pairs to eliminate in the case of excess. Even during comparison, measures are taken so that resistance values are never stored or read into memory. Instead, resistances are fed to a hardware comparator that simply returns a single bit to the controller. Thus, the client needs to get the mask from the only party that can produce it: the server. Before sending the mask to the client, the server obscures it by XORing it with a piece of information the client will have if all goes well: the XOF digest. The transfer of this obscured mask and random number is often referred to as the "handshake" (see Figure 3). The bigger the buffer, the more cherry picking we can do and the lower our error. (Error correction is described in Section 2.3.)

It is important to note that the client computation is constant time, while the server's time is proportional to buffer size. With increasing buffer, the server has to compare, sort, and filter more pairs. The client on the other hand, references the mask and only reads unmasked pairs. If key size is constant and fixed at 256 bits, the client will only ever read 256 pairs from its PUF, regardless of buffer. (This feature also aids in tamper detection.[16]).

2.3 RBC

RBC (response-based cryptography) is our error correcting search engine, which off-loads the work of error correction onto the higher powered server. It does not require any client-side correction methods, nor the use or transmission of error correction code. RBC enables the client to use their un-corrected key, as-is, and has the sever perform authentication by searching a limited subspace of keys using a known starting point.

The RBC search can be sped up, and thus handle more error, through the use of fragmentation and parallelization. Fragmentation is the process of breaking keys into smaller chunks or sub-keys, thereby decreasing the search-space for each sub-key and allowing for the search on each sub-key to run in parallel. An in-depth discussions of RBC can be found in the work of Cambou et al.[17–19] and Philabaum et al.[20]

For the purpose of understanding this work, the concept is as follows: The registrant sends their key to the CA, hashing it first to protect this information. The CA has idea of what the key should be before hashing based on the enrollment (see Section 2.2), and so the RBC engine injects error into that know starting point, hashes the outcome, and performs a comparison. The search is continued until a match is found, injecting error in various locations, and in various amounts, within a preset threshold or timeout.

RBC makes the system resilient to client error, and does not impose any correction mechanisms or requirements on the client. (See Section 6 for discussion on replacing the described RBC engine with a faster, more secure, and more capable RBC implementation in the next phase of this work.)

2.4 PQC Algorithm

The PQC algorithm chosen for this study is the DSA, CRYSTALS-DILITHIUM (Dilithium). We chose Dilithium for the following reasons: it became a NIST selected algorithm in 2022, provides good documentation and readable code, and has been demonstrated as a viable option for server-client models.[21,22] In addition, it is a convenient selection for this study because it places no restraints or conditions on the random number used to generate the cryptographic seed.

2.5 Malicious Action Detection

Consider the following two attempts to falsify information: stealing a slot on the ledger and modifying instruction packets during transmission.

If attempting to steal a slot on the ledger, the opponent would have to falsify the PUF hash sent over in order to be authenticated. If they do that successfully, they would still have to determine the input that produces the matching hash, in order to generate the correct secret key for future signing. That is two 2^{256} problems to solve, at a minimum (longer keys *and* hashes can be used), which is unlikely to be solved before the real party registers themselves.

If attempting to edit instruction packets while they are being transmitted, this will immediately be detected by the receiving party in that DSA verification of the modified instruction packet will fail. In order to succeed, the intercepting party would need to uncover the secret key, modify the contents, and finally generate a new signature using the modified contents; all before being detected or thwarted (i.e., transmission timeout).

Should a malicious actor attempt to modify instruction packets during transmission, the integrity is inherently protected through the use of DSAs. Should the actor try to pose as a participating party, they would have a hard time falsifying the PUF response if trying to steal a slot during registration, or secret key for signing, depending on the opportunity of attack.

3. TESTING AND METHODOLOGY

This paper analyzes a proof-of-concept prototype of the PQC key distribution system described in Section 2.1. Using this prototype we measure quality (NIST random number test), reliability (error rate and resilience), and speed (computation time).

3.1 Hardware

The PUF used is shown in Figure 4. It contains two ReRAM chips whose output is fed to a comparator returning either a `0` or a `1`. The microcontroller board (red) is flashed with code that allows interaction with the chips mounted on the custom printed circuit board (PCB) (green). The chips contain pre-formed ReRAM arrays, whose resistances are fed to the comparator by injecting current. The chips were manufactured for us by CrossBar. More information about this setup can be found in Riggs et al.[15]

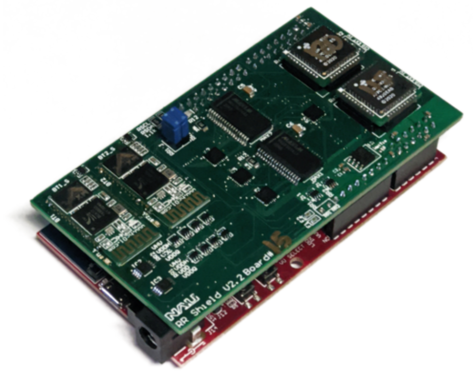


Figure 4. The in-house, ReRAM PUF dev-kit used for this test.

Each die contains four-thousand ninety-six cells. Using the cell pairing protocol (see Section 2.2), the number of possible pairs is:

$$\binom{4096}{1}\binom{4096}{1} = 16{,}777{,}216 \tag{1}$$

The number of pair configurations for a 256-bit key, with replacement and where order matters is:

$$\left(\binom{4096}{1}\binom{4096}{1}\right)^{256} = 16{,}777{,}216^{256} = 3.38 \times 10^{1849} \quad (2)$$

The computer used has an x86-64 architecture Intel Core i7-8850H CPU with AVX2 instruction set support and is running Fedora 37.

3.2 Software

The TAPKI system described in Section 2.2 is programmed in `python` and communicates to the ReRAM PUF over serial. Keys produced by the TAPKI system are called TAP-KEYs in this paper. Error is defined by how many bits of the client TAP-KEY deviate from the server TAP-KEY. In this context, the CA is the server, and the registrant is the client. During the RBC process (see Section 2.3), hashing is done with `python`'s `hashlib.sha3_256()`. The CA uses an enrollment of 20 reads, meaning 20 reads per cell. This is a relatively low enrollment. Paired with a relatively low buffer of 256, we should be approaching worst case scenario error rates. We can be more aggressive with error prevention by using a more robust enrollment (100-200 reads), and a bigger buffer (512-1024).

The deterministic, `avx2` Dilithium implementation was slightly modified to take in commandline arguments, which we pass in from the calling `python` code which is running TAPKI and the rest of the test. The modifications add support for on-demand and isolated key generation, message signing, and message verification. The key generation sequence was modified to take in the TAP-KEY (see Section 2.2) and use it as the random number in the seed generation process. The modifications were made on top of version 3.1 at commit `61b51a7`. The modified, compiled `C` binary is called using the `python subprocess` module.

The following variables are fixed across trials:

1. The message is "this is my message to you", padded with ASCII zeros to meet Dilithium's length requirement.
2. The buffer is 256 address pairs.
3. The injected current is 800 nA.

We tested two-hundred and fifty thousand trials. Each iteration (or trial) of the test performs the following registration and verification sequences:

Registration

1. A fresh password is generated to simulate a new user registration.
2. The CA (server) generates a TAP-KEY and sends the challenge (handshake) to the registrant (client).
3. The registrant (client) applies the challenge to its PUF and sends the CA the hashed key
4. The CA runs RBC to "correct" error.
5. Using the TAP-KEY found by RBC, the CA generates a public/private key pair using Dilithium, discarding the private key.
6. Using the TAP-KEY sent, the registrant generates a public/private key pair using Dilithium.

Verification

1. The sender (registrant) signs the message with their private key, using Dilithium.
2. The recipient attempts to verify the message with Dilithium, using the CA's record of the sender's (registrant's) public key.

3.3 Quality

Quality of this system is measured in terms of randomness. To gauge the randomness of the keys produced, we collect every client TAP-KEY across trials, concatenate them into a bitstream ($256 \times 250{,}000 = 64$ millon bits), and feed that bitstream to NIST's random number generator test suite (SP 800-22 Rev. 1a).[23]

As a continuation of work, and to facilitate comparison, we use the same subset of 12 tests used in the wafer-level TRNG qualification of Cambou et al.: frequency, block frequency, cumulative sums, runs, longest run of ones, rank, discrete fourier transform, approximate entropy, serial, linear complexity.[14]

We instruct the test suite to break up the sixty-four million bits into one-hundred bitstreams, each six-hundred and forty thousand bits in length. Defaults were used otherwise.

3.4 Reliability

We measure reliability in terms of error and verification rates; error rates coming from the PUF, and message verification rates using Dilithium. As a test of reliability, we want to ensure that slight mismatches between server and sender PUF responses (errors), can be overcome such that the user is successfully registered and messages sent by that user can be successfully verified.

To consider the system reliable, it should produce a high message verification success rate, and a low error rate. It should also exhibit resilience in the face of error.

3.5 Speed

We measure the average times of the registration and verification cycle to observe whether or not this is a viable solution for a multi-user network. We also measure each sequence of the protocol individually to identify bottlenecks and understand where to focus optimization efforts.

4. RESULTS AND DISCUSSION

4.1 Quality

```
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  0   0   0   2  15  44  19  16   1   3  0.000000 *  100/100     Frequency
  0   0   0   0   0   0   0   0   0 100  0.000000 *  100/100     BlockFrequency
  0   0   0   0   0   0   2   3  12  83  0.000000 *  100/100     CumulativeSums
  0   0   0   0   0   0   2   2  13  83  0.000000 *  100/100     CumulativeSums
 91   2   3   2   2   0   0   0   0   0  0.000000 *   37/100  *  Runs
 95   4   1   0   0   0   0   0   0   0  0.000000 *   20/100  *  LongestRun
  8  15  16   7   3  12   9  10  13   7  0.102526     99/100     Rank
 37  11   5   6   8  14   7   7   2   3  0.000000 *   92/100  *  FFT
 77   9   9   1   2   1   1   0   0   0  0.000000 *   59/100  *  ApproximateEntropy
 27  17   8   7   8  11   5   4   6   7  0.000001 *   94/100  *  Serial
  8   9  15   9  13   5  11  14   8   8  0.437274     98/100     Serial
 15   9   9   7   7   5  16  12   6  14  0.115387     98/100     LinearComplexity
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 5. NIST random number test results. With 100 streams, minimum pass rate is 96.

Zeros	Ones	Difference
320214	319786	427

Figure 6. Average split and difference of ones and zeros per tested stream.

The results in Figure 5 show that the frequency and cumulative sum tests pass in proportion, but fail in p-value. According to NIST, the frequency test supplies the most basic evidence for non-randomness, and if this test fails, the likelihood of other tests failing is high.[23] Inspecting `freq.txt` (captured by Figure 6), we can see there is a clear, but minimal, bias towards zero, but not by much. We interpret the results as indicating that the frequency split of each stream is as expected for each random sequence (passing proportion), but the fact that every stream biases towards zero means the overall test fails (failing p-value).

We have determined that this is due to one chip having slightly higher resistances than the other. We ensure a perfect 128 zero-one split per 256-bit key by using the mask (see Section 2.2), which means the bias is coming from error, which tends towards zero in this case due to the resistance difference of the chips.

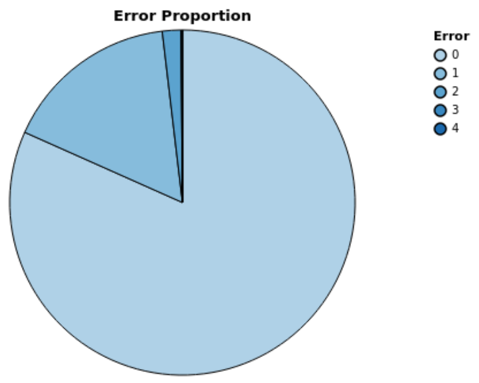
There are a few remedies for this. First, we could virtually eliminate error by using more aggressive error prevention as discussed in Section 3.2. With zero error, the mask will ensure a perfect 50-50 zero-one split. Second, bias in the chips (revealed by error) can be corrected by developing more scrutinous chip pairing methods and mechanisms. An alternative would be to move to a single chip system, where the chip is logically split in two; however, this would require a new chip design in order to maintain being able to feed the two signals to a comparator for security (see Section 2.2). Third, if sticking with a two-chip system, using analog shifting to bring the chips closer to the same median value might be a quick-fix. Research on this topic is currently underway for our MRAM PUFs, led by Manuel Aguilar Rios.

The streams tested were generated by a system configured for stability. We need stability in this use-case for reliable registration. To configure the system strictly for random number generation, we would generate streams using the unstable cells or pairs of the device.[14]

We find it promising that some of the other tests are passing, despite the system being configured for stability and despite failing the frequency test. We expect that fixing the frequency problem alone should result in passing more of the tests.

It should be considered that the randomness in this case is derived from the physical randomness of the hardware. As such, we take failing some of the NIST tests more lightly than if we were proposing a mathematical formula. In the system proposed, there is no requirement for true randomness, just that the streams used are hard to guess and unique per registration. Therefore, we deem it acceptable to have a PUF with periodic features, as indicated by the failing of the Fourier transform test (FFT), so long as the periodic features of one PUF do not help determine the periodic features of another PUF, and the periodic features across PUFs are unique.

4.2 Reliability



Error	Count	Percent
0	204176	81.67%
1	41148	16.46%
2	4312	1.72%
3	343	0.14%
4	21	0.01%

Figure 7. Proportion of errors observed with accompanying error counts and percentages.

Out of a quarter of a million trials, 98.13% had zero to one errors. Less than 2% of the observations had more than one error. The maximum error observed is four. This demonstrates low error, despite a lax error-prevention configuration (see Section 3.2). More importantly, this demonstrates the ability to overcome 100% of the error encountered, as we observed no failed verification. All of the error was able to be corrected by the proposed system, and all properly signed messages were verified.

4.3 Speed

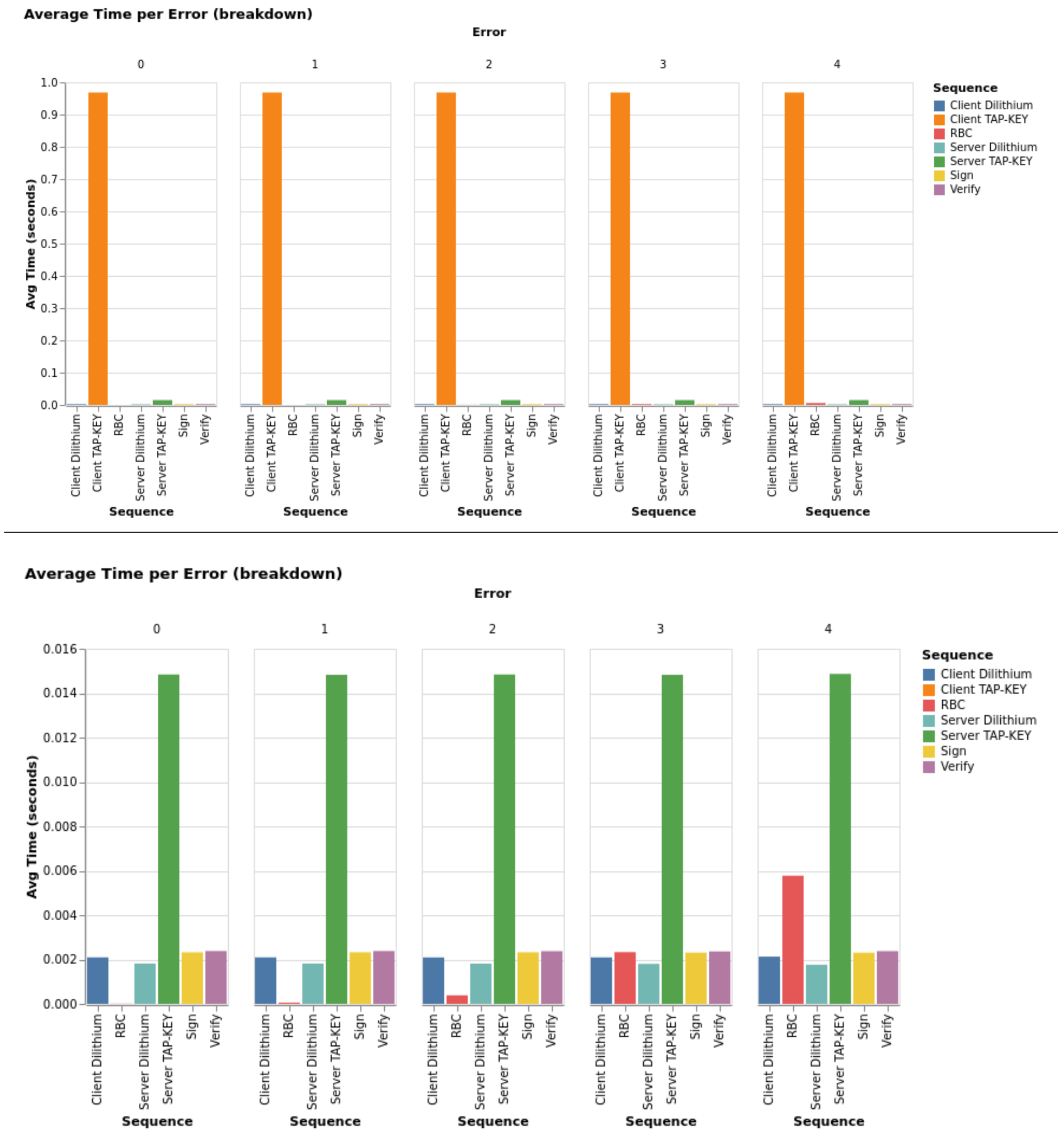


Figure 8. Average time of each sequence per error, full and zoomed in (without client TAP-KEY time).

According to Figure 8, the only time that varies with error is RBC computation time. No other parts of the system are slowed down by increasing error. RBC time only affects the registration process. This demonstrates that the effects of error are isolated and allow for swapping out the current RBC engine with a more optimized, more secure RBC engine such as the one proposed in Wright et al.,[24] without affecting the rest of the system.

Studying the total average times and RBC times per error (Figure 9), we see that even at the highest error observed, the entire test completes in under a second, on average. The lowest RBC time is 57 microseconds (1 error) and the highest is 5.7 milliseconds (4 errors).

Clearly (see Figures 8 and 10), the limiting factor is client TAP-KEY time. The server TAP-KEY generation,

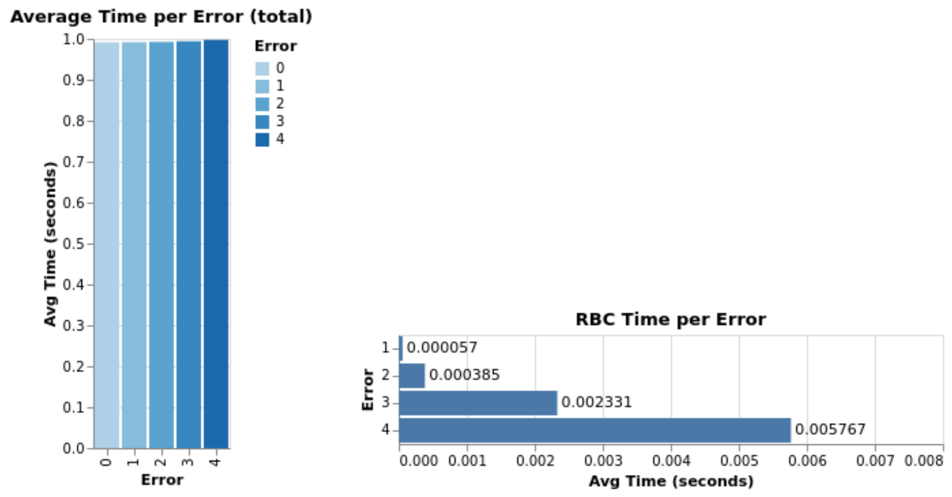
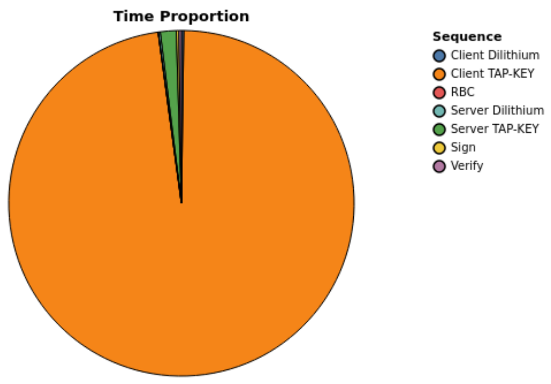


Figure 9. Average time of the entire sequence per error and average RBC time per error.



Sequence	Avg Time (seconds)	Percent
Server TAP-KEY	0.014834	1.50%
Client TAP-KEY	0.968518	97.63%
Server Dilithium	0.001817	0.18%
Client Dilithium	0.002096	0.21%
RBC	0.000026	0.00%
Sign	0.002323	0.23%
Verify	0.002381	0.24%

Figure 10. Time proportion for each sequence using average times with accompanying table showing numerical values and percentage of total time.

Sequence	Avg Time (seconds)	Proportion
Registration	0.987290	209
Verification	0.004704	1

Figure 11. Average time values per action with relative proportion (registration:verification).

despite it being much is faster, is far more computational intensive. We have identified the problem on the client side, to be a serial bottleneck in the lower powered device. Correcting this problem should bring the client TAP-KEY time well below the server TAP-KEY time. Especially considering the fact that the client computation is constant time and independent of buffer size (see Section 2.2).

Looking at the data from the perspective of registration and verification (Figure 11), verification is over two-hundred times faster than registration. Registration is a one-time expense per user, while verification should happen many times per user. The action that happens most is orders of magnitude faster. This gap will be decreased by addressing the client-side serial bottleneck as discussed above, bringing registration time closer to verification time. Message verification does not require a PUF, although on-demand key generation can be enforced to prevent storing the secret key, as discussed in Section 2.1, which would increase security at the expense of verification time.

5. CONCLUSIONS

This work proposed a PQC key distribution framework using PUFs for real-time authentication and registration of autonomous assets, and tested a proof-of-concept prototype thereof. The prototype demonstrates low error rates despite non-aggressive prevention methods, and excellent resilience to the error encountered, with no failed registrations or verifications. Computation times are in the tens of milliseconds range except client key generation time, for which we have identified the correctable bottleneck which should bring it into that range. NIST random number tests show promise, but indicate a bit frequency problem, for which we have proposed a potential solution. Overall, the proof-of-concept system demonstrated viability and the modular framework proposed readily accepts modifications, adaptations, and improvements to make it suitable for real-world use.

6. FUTURE WORK

As mentioned throughout, the framework we have presented is designed to be modular, and intended to be modified to fit multiple use-cases and evolving requirements.

We plan to design a new ReRAM PUF that solves the serial bottleneck at the hardware level, and employs one of the two frequency corrections proposed in Section 4.1. Testing shall be done to bring this new PUF design in compliance with NIST's qualifications of randomness. An additional layer of security can be added to this PUF through advancement of our tamper sensitivity methods, highlighted in the work of Wilson et al.[16]

We prescribe building a real-world implementation using the work of Wright et al.[24] Their work presents an RBC error-correcting engine (see Section 2.3), which instead of sending the hash of the key to the CA, would only require that the public key is sent to the CA in order for the CA to correct for error. This is an excellent enhancement since it eliminates the transfer of sensitive information over the network, even if that sensitive information is hashed. The work also describes optimizations, parallelization, and the use of GPUs, which make the RBC engine and attributed computation times better suited for scalable server-client models, especially in the face of higher error.

Finally, a similar system can be discussed and tested in detail for KEM PQC algorithms, and later tested with the RBC engine developed by Lee et al.[25]

ACKNOWLEDGMENTS

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-23-2-0014. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Chen, L., Chen, L., Jordan, S., Liu, Y.-K., Moody, D., Peralta, R., Perlner, R. A., and Smith-Tone, D., [*Report on post-quantum cryptography*], vol. 12, US Department of Commerce, National Institute of Standards and Technology ... (2016).

[2] Ford, P., "The quantum cybersecurity threat may arrive sooner than you think," *Computer* **56**(2), 134–136 (2023).

[3] Campbell Sr, R., "Evaluation of post-quantum distributed ledger cryptography," *The Journal of The British Blockchain Association* **2**(1) (2019).

[4] Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al., "Status report on the third round of the nist post-quantum cryptography standardization process," *US Department of Commerce, NIST* (2022).

[5] Pandit, A. A., Kumar, A., and Mishra, A., "Lwr-based quantum-safe pseudo-random number generator," *Journal of Information Security and Applications* **73**, 103431 (2023).

[6] Cambou, B., Gowanlock, M., Yildiz, B., Ghanaimiandoab, D., Lee, K., Nelson, S., Philabaum, C., Stenberg, A., and Wright, J., "Post quantum cryptographic keys generated with physical unclonable functions," *Applied Sciences* **11**(6), 2801 (2021).

[7] Habib, B., Cambou, B., Booher, D., and Philabaum, C., "Public key exchange scheme that is addressable (pka)," in [*2017 IEEE Conference on Communications and Network Security (CNS)*], 392–393, IEEE (2017).

[8] Cambou, B. and Telesca, D., "Ternary computing to strengthen cybersecurity," in [*Intelligent Computing*], Arai, K., Kapoor, S., and Bhatia, R., eds., 898–919, Springer International Publishing, Cham (2019).

[9] Cambou, B., "Physically unclonable function generating systems and related methods," (May 29 2018). US Patent 9,985,791.

[10] Cambou, B. and Gibbs, J., "Nanomaterial physically unclonable function systems and related methods," (Sept. 29 2020). US Patent 10,790,994.

[11] Cambou, B. F., Quispe, R. C., and Babib, B., "Puf with dissolvable conductive paths," (May 28 2020). US Patent App. 16/493,263.

[12] Korenda, A. R., Afghah, F., Cambou, B., and Philabaum, C., "A proof of concept sram-based physically unclonable function (puf) key generation mechanism for iot devices," in [*2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*], 1–8 (2019).

[13] Cambou, B. and Orlowski, M., "Puf designed with resistive ram and ternary states," in [*Proceedings of the 11th Annual Cyber and Information Security Research Conference*], *CISRC '16*, Association for Computing Machinery, New York, NY, USA (2016).

[14] Cambou, B., Telesca, D., Assiri, S., Garrett, M., Jain, S., and Partridge, M., "Trngs from pre-formed reram arrays," *Cryptography* **5**(1), 8 (2021).

[15] Riggs, B., Partridge, M., Cambou, B., Burke, I., Rios, M. A., Heynssens, J., and Ghanaimiandoab, D., "Multi-wavelength quantum key distribution emulation with physical unclonable function," *Cryptography* **6**(3), 36 (2022).

[16] Wilson, T. and Cambou, B., "Tamper-sensitive pre-formed reram-based pufs: Methods and experimental validation," *Frontiers in Nanotechnology* , 89 (2022).

[17] Cambou, B., Philabaum, C., Booher, D., and Telesca, D. A., "Response-based cryptographic methods with ternary physical unclonable functions," in [*Future of Information and Communication Conference*], 781–800, Springer (2019).

[18] Cambou, B., Gowanlock, M., Heynssens, J., Jain, S., Philabaum, C., Booher, D., Burke, I., Garrard, J., Telesca, D., and Njilla, L., "Securing additive manufacturing with blockchains and distributed physically unclonable functions," *Cryptography* **4**(2), 17 (2020).

[19] Cambou, B., "Unequally powered cryptography with physical unclonable functions for networks of internet of things terminals," in [*2019 Spring Simulation Conference (SpringSim)*], 1–13, IEEE (2019).

[20] Philabaum, C., Coffey, C., Cambou, B., and Gowanlock, M., "A response-based cryptography engine in distributed-memory," in [*Intelligent Computing*], 904–922, Springer (2021).

[21] Kampanakis, P. and Sikeridis, D., "Two post-quantum signature use-cases: Non-issues, challenges and potential solutions," in [*Proceedings of the 7th ETSI/IQC Quantum Safe Cryptography Workshop, Seattle, WA, USA*], **3** (2019).

[22] Sikeridis, D., Kampanakis, P., and Devetsikiotis, M., "Post-quantum authentication in tls 1.3: a performance study," *Cryptology ePrint Archive* (2020).

[23] Bassham III, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., Leigh, S. D., Levenson, M., Vangel, M., Banks, D. L., et al., [*Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*], National Institute of Standards & Technology (2010).

[24] Wright, J., Gowanlock, M., Philabaum, C., and Cambou, B., "A crystals-dilithium response-based cryptography engine using gpgpu," in [*Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*], 32–45, Springer (2021).

[25] Lee, K., Gowanlock, M., and Cambou, B., "Saber-gpu: A response-based cryptography algorithm for saber on the gpu," in [*2021 IEEE 26th Pacific Rim International Symposium on Dependable Computing (PRDC)*], 123–132, IEEE (2021).