UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Running Longer To Slim Down: Post-Quantum Cryptography on Memory-Constrained Devices

Fournaris, A.P.; Tasopoulos, Georgios; Brohet, M.; Regazzoni, F.

DOI
10.1109/COINS57856.2023.10189268

Publication date
2023

Document Version
Final published version

Published in
2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS)

License
Article 25fa Dutch Copyright Act (https://www.openaccess.nl/en/in-the-netherlands/you-share-we-take-care)

Link to publication

Citation for published version (APA):
Fournaris, A. P., Tasopoulos, G., Brohet, M., & Regazzoni, F. (2023). Running Longer To Slim Down: Post-Quantum Cryptography on Memory-Constrained Devices. In *2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS): July 23-July 25, 2023, Berlin, Germany* (pp. 151-156). IEEE. https://doi.org/10.1109/COINS57856.2023.10189268

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:16 Oct 2024

Running Longer To Slim Down: Post-Quantum Cryptography on Memory-Constrained Devices

Apostolos P. Fournaris*, George Tasopoulos*, Marco Brohet† and Francesco Regazzoni†‡
* Industrial Systems Institute, ATHENA Research Center, Greece, email: {fournaris, g.tasop}@isi.gr
† University of Amsterdam, The Netherlands, email: {m.j.a.brohet, f.regazzoni}@uva.nl
‡ Università della Svizzera italiana, Switzerland, email: regazzoni@alari.ch

2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS) | 979-8-3503-4647-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/COINS57856.2023.10189268

***Abstract*—Since we are getting closer to the realisation of a quantum computer capable of breaking the currently deployed public key cryptosystems, we need to be ready with the next generation of quantum-safe cryptosystems. In the case of small, low-memory embedded/cyber physical systems frequently used in the IoT world, the adoption of these post-quantum cryptography algorithms (PQC) is far from being straightforward. Currently available implementations are mostly characterized by high memory requirements that make the adoption on constrained devices a difficult challenge. In this work, we explore the feasibility of implementing quantum resistant cryptography on memory-restricted devices and we present the strategies that should be adopted while tackling the problem. We summarize and discuss the most common techniques currently available in literature for trading speed with reduced memory footprint. Discussed techniques range from strategies to minimise the static memory required by an implementation to techniques to deal with large artifact sizes. We show that, by using the appropriated optimization technique, even PQC schemes that require an amount of memory that largely exceeds the memory available on certain devices, can be successfully implemented, while also leaving enough memory for other applications that might reside on the same device.**

***Keywords*—post-quantum cryptography, constrained embedded systems, memory optimisations, cryptographic protocols**

I. Introduction

As research groups and major technology stakeholders invest significant effort and resources into the development of functional quantum computers, the security community is getting ready to update existing public key cryptography and deploy new schema capable of withstandimg the computational power of quantum computers. It is estimated that quantum computers with more than 2000 error free qubits processing capability will be able to break public key cryptography in realistic time using the Schor's algorithm [1]. Addressing this risk, new quantum resistant cryptography schemes have been proposed by various research groups and are under standardization (for instance, by the US National Institute of Standards and Technology (NIST) [2]). Those post-quantum cryptography (PQC) schemes are indeed capable of resisting quantum computer cryptanalytic attacks, but they introduce overheads in terms of performance, resource utilization, communication, and power consumption [3], [4]. While this overhead does not seem to be problematic for traditional computers or servers, in the embedded system domains the situation is quite different.

The embedded system ecosystem is very diverse. It includes devices that have a very broad range of performance and memory capabilities. For instance, it includes edge devices with high speed processors (like ARM Cortex A class), FPGA fabric, dedicated GPU (or multiple GPU) cards as well as a few Gigabytes (GBs) of RAM. On the other hand, the embedded world also includes small scale, low-end, resource constrained environments that are based on low power 8-bit or 16-bit (or even 32-bit) processors (acting as micro-controllers) with a limited speed and a RAM of just a few Kilobytes (KB). This constrained devices constitute the bulk of devices that are deployed in the Internet-of-Things (IoT), Industrial IoT environment, or in Wireless Sensor Networks. In many use-cases of these settings, the main goal is not performance but rather feasible security using just the limited resources (memory, power, energy) of an embedded system device. To fit the appropriated cryptographic primitive in such a device, designers must trade mostly with computation speed.

NIST has held an open competition to select the PQC schemes to be standardized and the Lattice-Based Cryptography (LBC) schemes of Kyber for Key Encapsulation Mechanism (KEM) operation was chosen while the Dilithium and Falcon LBC schemes where chosen for Digital Signatures along with the SHPINCS+ as non-LBC scheme (hash-based scheme). In parallel, Germany's Federal Office for Information Technology (BSI) suggests FrodoKEM (LBC KEM scheme) and Classic McEliece (code-based scheme) for PQC-based security on Key Exchange [5]. The above selections indicates that the relevance of LBC schemes and code-based ones in achieving quantum resistance[1]. LBC, hash-based and code-based schemes suffer from large key sizes (especially the hash and code-based schemes) and from an elaborate processing that require high memory storage for keys and precomputed values. All currently collected results indicate that the above schemes are not lightweight and cannot be implemented "as is" on a low memory device. Considering that the typical use-cases where such devices are used are the ones where speed is not the main requirement, it is possible (and necessary) to leverage design techniques that allow to minimize the memory

979-8-3503-4647-3/23/$31.00 © 2023 IEEE

[1]NIST has issued a new round (round 4) of the PQC competition that is focused on code-based schemes BIKE, Classic McEliece and HQC

Authorized licensed use limited to: Universiteit van Amsterdam. Downloaded on October 16,2023 at 09:57:33 UTC from IEEE Xplore. Restrictions apply.

TABLE I
PQC ALGORITHMS PRIMITIVE OPERATIONS MEMORY REQUIREMENTS AND SIZES (IN BYTES)

		PQClean			*pqm4*			*Sizes*		
Algorithm	***NIST Security Level***	***Key gen***	***Enc. / Sign***	***Dec. / Ver.***	***Key gen***	***Enc. / Sign***	***Dec. / Ver.***	***Public Key***	***Secret Key***	***Ciphertext/ Signature***
Kyber512	*1*	6116	8780	9556	2248	2336	2352	800	1632	768
Kyber768	*3*	10 212	13 380	14 476	2784	2856	2872	1184	2400	1088
Kyber1024	*5*	15 100	18 772	20 348	3296	3368	3392	1568	3168	1568
Dilithium2	*2*	38 308	51 932	36 220	38 408	49 380	36 212	1312	2355	2420
Dilithium3	*3*	60 844	79 588	57 732	60 836	68 836	57 724	1952	3708	3293
Dilithium5	*5*	–	–	–	97 692	115 932	92 788	2592	4539	4595
Falcon512	*1*	18 416	42 508	4724	41 248	42 492	40 348	897	1281	666
Falcon1024	*5*	36 296	82 532	8820	81 272	82 516	80 284	1793	2305	1280
SPHINCS+128s	*1*	2340	2408	1980	–	–	–	32	64	7856
Bike1	*1*	36 025	25 933	78 613	44 157	32 205	91 457	1541	281	1573
Hqc1	*1*	48 852	64 436	71 172	–	–	–	2249	56	4481
Frodo640-SHAKE	*1*	36 480	58 136	78 760	26 408	51 784	72 408	9616	19 888	9752

footprint while sacrificing the performance.

In this work we explore the possibility to run PQC algorithms on highly memory-constrained devices. To achieve this, firstly describe the most popular PQC software libraries for embedded systems and we report the memory footprint for the NIST selected and round 4 PQC schemes when these libraries are deployed in a typical embedded system. We then highlight the memory bloating that happens when such algorithms are adopted in common security protocols, such as the TLS 1.3 protocol. The provided results show that in highly memory constrained embedded systems that have at most 10 KB of available RAM memory, it is practically impossible to utilize most of the NIST selected PQC schemes. We thus argue that a different trade-off between speed and memory usage needs to be made for such systems. In such trade-off, speed is sacrificed for memory usage reduction. We argue that this is a reasonable compromise since many use-cases where resource constrained embedded systems are used do not have high speed requirements. Eventually, given the related research work on the topic, in the paper we provide a series of memory reduction strategies that can effectively be applied in most of the NIST selected PQC schemes.

The rest of the paper is organized as follows. In Section II summarizes existing PQC libraries optimized for embedded system optimized and reports the memory footprint that such libraries have when used in security protocols such as TLS 1.3. In Section III we present current strategies to reduce the memory footprint and we propose further possible optimizations.Section IV reports our concluding remarks.

II. PQC SCHEME IMPLEMENTATIONS

To reduce dependencies with other software libraries, the OQS project [6] provides a portable and standalone PQC implementation, called PQClean [7], that can be used easily by third-party frameworks. This project has collected most of the PQC algorithms throughout the NIST standardization process and at the moment of this paper's writing includes all the PQC algorithms proposed for standardisation by NIST and some other (i.e. Classic McEliece that has not been selected by NIST but is the recommended algorithm from the German BSI [5]).

The PQClean library however mostly targets mid to high-end machines like laptops, PCs and servers. When memory resources are low and processing power is orders of magnitude less than the above types of machines, deploying this library directly is practically impossible. To tackle this problem, the *mupq* project, started to collect and develop optimised implementations of PQC algorithms for a number of Microcontroller Units (MCUs). The *pqm4* [8] library, for instance, targets the NIST PQC process reference platform for embedded systems, ARM Cortex-M4. It offers optimised implementations specifically for this device, dealing with its low processing power and its limited memory resources. The library uses optimised assembly instructions, i.e. vector instructions, to speed up the operations and techniques to reduce the required memory. At its current state *pqm4* only offers the PQC algorithms selected for standardisation. The *mupq* repository includes more libraries, such as the *pqm3* [9], that targets the Cortex-M3 microcontroller and *pqriscv* [10] that targets the RISC-V family of microcontrollers. Unfortunately these projects are currently not very actively developed and, at the moment, offer only a small number of implementations, often deprecated.

A. Memory Requirements of PQC Primitives

Table I presents the reported memory requirements for a number of PQC algorithms, regarding static implementation memory of *PQClean* and *pqm4* libraries as well as the size of the public keys, the secret keys and the ciphertexts or the signatures of each PQC algorithm. The reported measurements are extracted from the benchmarks of [8], from Table I of [3] and from the individual PQC algorithm specifications found in the respective algorithms official websites [11]–[17]. As mentioned, the PQClean implementations target generic PCs, while the pqm4 implementations are targeting embedded system boards such as Nucleo-F439ZI [2]. This board is equipped with an ARM Cortex-M4 microcontroller and offer 192 KB

[2]the Table I and II results from [3] have been collected using this board

Authorized licensed use limited to: Universiteit van Amsterdam. Downloaded on October 16,2023 at 09:57:33 UTC from IEEE Xplore. Restrictions apply.

of RAM [3] and 2 MB of Flash memory. The table includes all 4 PQC algorithms selected for standardisation (Kyber, Dilithium, Falcon and SPHINCS+), two out of three of the NIST PQC Round 4 (Bike and HQC) and FrodoKEM, the BSI recommendation for KEMs. Even if these microcontrollers cannot be categorized as low memory devices, the measurements provide an indication of the memory that each PQC algorithm needs when the implementation optimize the used static memory.

In this paper, we aim at discussing the feasibility of porting PQC algorithms on a constrained device that has a RAM of 8KB to 10 KB at most. Looking into the optimised pqm4 implementation of the Kyber KEM family, it can be noted that the memory requirements have been reduced significantly compared to the PQClean version, making this implementation suitable for low memory devices. On the contrary, the optimised implementations of Bike, HQC and FrodoKEM, the reduction of memory usage is still not sufficiently to fit in 8-10 KB. Regarding digital signatures, optimised implementations of Dilithium and Falcon require a substantial amount of memory and cannot fit constrained devices. Sphincs+ on the contrary, has very low memory requirements thus appears to be suitable for our target device profile, even without the need of aggressively optimizing the original PQClean version.

In addition to the memory requirements, a significant restriction is introduced by the size of the PQC algorithm artifacts (the public keys, secret keys and ciphertexts or signatures). Table I provides measurements on the size of the public key, secret key and ciphertext/signature in bytes for each of the PQ KEM/Digital signature algorithm as reported in the specification document of each algorithm [11]–[17]. These artifacts are important since, during the computation, they should be saved in memory. For example, SPHINCS+, requires limited static memory to operate, but requires a substantial amount of memory to store a single digital signature. Also FrodoKEM requires a large amount of memory for each of its artifacts, with the secret key being almost 20 KB. Also for the other schemes the size of the artifacts can not be considered negligible for memory constrained devices.

B. Memory Requirements of PQC TLS 1.3

To provide an evaluation closer to a realistic application, it is important to assess the PQC impact on memory usage when such schemes are instantiated in commonly used security algorithms and protocols. To have such an evaluation we should consider the memory usage of the PQC scheme itself, the memory usage of the security protocol where the PQC scheme is instantiated, and also the impact of the keys transferred across a network which have to be buffered in memory during communication. A typical example of the above, is the transferred bytes of a TLS or a SSH session that are highly affected by the underlying digital signature and key exchange cryptographic schemes. To this end, Table II reports the memory usage of the TLS 1.3 [18] protocol (from [3]) focusing on the stack memory usage of the whole program and the .bss memory region (these two are combined in the column named "Total Static Memory") along with the transferred bytes during TLS 1.3 handshake communication. Note that TLS 1.3 uses a combination of key agreement and digital signature schemes which in PQC is translated in combinations of Digital Signature and KEMs, as those that can be seen in the first column of Table II.

TABLE II
PQ TLS 1.3 HANDSHAKE MEMORY REQUIREMENTS.

Notation	*Total Static Memory (bytes)*	*Communication Sizes (bytes)*
Selected Algorithms for Standardization		
Dil2-Kyb1	49 648	14 748
Falc1-Kyb1	43 616	6833
Dil3-Kyb3	69 072	20 224
Falc5-Kyb3	84 072	11 789
Dil3-Kyb5	69 104	21 088
Falc5-Kyb5	84 584	12 647
Sph1s-Kyb1	800	33 892
Fourth Round Algorithms		
Dil2-Bike1	81 577	16 292
Dil2-Hqc1	71 672	19 910
Traditional Algorithms		
RSA-ECDHE	2368	3742
ECDSA-ECDHE	2368	2353

In Table II it can be observed that compared to their traditional counterparts (*Traditional Algorithms* section of the Table), PQC schemes consume significantly more memory. A full security solution consists of the PQC schemes, the network stack (lwIP in our case), and often, an embedded OS like RTOS that include the storage requirements of PQC public keys, signatures and certificates. Thus, the memory impact, as this is depicted in Table II, that the PQC TLS 1.3 combinations introduce becomes not affordable for devices having only few KB or RAM. All PQC schemes combinations in Table II under the TLS 1.3 setup need significantly more memory than the 10 KB limit we envision for our target device. There may be an exception when using TLS 1.3 with SPHINCS+ for authentication (for example the *Sph1s-Kyb1* combination) since the static memory needed is significantly small as long as there will be a memory reduction mechanism for the exchanged keys during communication (i.e. currently the communication size of *Sph1s-Kyb1*, without any memory reduction strategy, is too high).

Based on these numbers, running PQC schemes in memory constrained devices is challenging. Designers should thus apply a series of design strategies that allow to trade the execution speed with memory usage.

III. STRATEGIES TO REDUCE THE MEMORY FOOTPRINT

A. Existing attempts

At the first stages of the NIST PQC competition, memory occupation was analyzed only for few algorithms. Among them, there was Classic McEllice, that requres from 260 KB to 1.3 MB to store its public key alone. To reduce the usage of the RAM, in [19] and [20] the authors have moved the storage of the McEliece public key in the Flash memory. Unfortunately

Authorized licensed use limited to: Universiteit van Amsterdam. Downloaded on October 16,2023 at 09:57:33 UTC from IEEE Xplore. Restrictions apply.

on low-memory systems, even the flash could be of a few KB, thus making it impossible to use this approach. Strenzke [21] proposed to stream the public key by introducing a technique for dealing with large public keys in code-based cryptosystems and in [22] this concept was applied on the Classic McEliece thus offering a memory-optimised implementation.

Regarding digital signatures, the NIST chosen PQC schemes require large public keys and signatures. There is a significant effort to make such schemes usable in memory-restricted environments. In [23] the authors have been able to run Dilithium, Falcon SPHINCS+, Rainbow, and GeMSS on devices that offer less than 8 KB of RAM but unfortunately they were able only to *verify* post-quantum signatures and not to *sign* messages. In [24] improving previous literature [23], the authors successfully managed to run all the operations of Dilithium (*verify*, *sign* and *key generation*) on a device with less than 8 KB of RAM. In [25], similarly,, the authors successfully run all the operations of Sphincs+ on a TPM with overall memory usage less than 8 KB. The same philosophy is followed by a pull request of the pqm4 [8] project, that offers a low memory Dilithium [26] achieved by computatimg on-the-fly the public key, instead of having it pre-computed and stored in RAM.

B. Generic Memory Reduction Strategies

Given the above research attempts, taking into account the specificity of the PQC algorithms and also the use-case requirements of very resource constrained embedded systems, a more aggressive trade-off between speed and memory needs to be devised. In practice, when the target applications allows, the designers should significantly reduce the performance of the implementation in favor of reduced memory footprint. This approach has shown to be feasible in the part to implementing other cryptography algorithm, such as AES or ECC, in extremely constrained devices such as RFIDs. This approach typically requires to remove any pre-processing (offline) optimizations that may reside in the memory and change in the way in which arithmetic constructs and operations are realized in software. Apart from the above mentioned memory requirements, PQC schemes also requres storage for keys, certificates, and signatures as indicated in Table II. These should also be target of memory optimization. In the following subsections we provide some general strategies on how to deal with these memory reduction aspects focusing on the PQC schemes selected by NIST.

1) PQC Scheme Implementations with reduced memory footprint: A computation profile of the NIST selected PQC schemes (that are mostly LBC based) reveals that there are several internal components that consume memory and/or dominate performance. The NIST LBC based PQC schemes use Module Lattices and rely on the Ring Learning With Error (RLWE) problem (i.e. Kyber and Dilithium) or use NTRU lattices and rely on the short integer solution problem (SIS) over NTRU lattices (i.e .Falcon). Hash based schemes as expected rely heavily on Hash functions while code based schemes follow error correction code logic. Conceptually, in the above schemes, the identified components are:

- **XOF/Hash Function cores:** The LBC based PQC schemes include some eXtendable-Output Function (XOF) and/or hash function mechanism that reduce the size of input messages or used for compression/transformation of generated random lattices within the PQC schemes. SHAKE-128 or SHAKE-256 relying on the Keccak structure is used in most LBC based PQC schemes. Along with that SHA-3 hash function is also adopted and AES based XOFs. These cryptocores (mostly Keccak based) need to be implemented in a memory conserving manner so that they do not overburden the overall memory usage. Fully unrolled versions of XOFs/hash functions, (e.g Keccak based) for example that are typically found on research literature [27], designed to reduce time delay, should not be adopted. Special attention should be paid in order to make cryptocore implementation lightweight in terms of memory footprint. This, in practice, could mean that the XOF/Hash function operations should be streamlined by iterating the code of a hash function single (or a few) rounds offering an "on the fly" XOF/hash function outcome. Generalizing the discussion beyond LBC based PQC schemes, when dealing with hash-based schemes, like SPHINCS+, the memory requirements of the implementation is not an major issue. Nevertheless, memory-optimised versions of the underlying hash function, that is the building block of hash-based schemes, can still be used and trade speed for less required memory.
- **Gaussian Sampler:** Typically, LBC based PQC schemes use discrete samplers to generate randomness. In most cases these are Gaussian samplers. While implementations of Gaussian samplers do not consume excessive amounts of memory (especially RAM memory), an implementer should utilize the microcontroller's capabilities to generate randomness if possible in order to reduce the need of additional code for the sampler itself.
- **Algebraic operations**: In LBC based PQC schemes based on both Module and NTRU Lattices, there are series of matrix to vector multiplications using polynomial with integer or complex number coefficients. These operations include polynomial multiplications over a Ring (thus performing also polynomial modulo operations) that are computationally slow and typically constitute the first goal of speed optimization on a PQC implementation. Given the size of the involved polynomials, the LBC based PQC schemes employ Number Theoretic Transform (NTT) operation or Fast Fourier Transformations (FFT) in order to perform polynomial multiplication on the NTT/FFT domain where such operation is easier (point-wise multiplication) and faster than other multiplication methods. However, the NTT/FFT multiplication is still in several cases not fast enough to compete traditional public key cryptography schemes. So, in the *pqm4* and

Authorized licensed use limited to: Universiteit van Amsterdam. Downloaded on October 16,2023 at 09:57:33 UTC from IEEE Xplore. Restrictions apply.

PQClean libraries, offline precomputations are made of the NTT/FFT twiddle factors. Those precomputed values are stored in memory thus significantly increasing a PQC scheme memory footprint. An obvious memory reduction strategy is to remove the precomputations from memory and rely on a online, "on the fly" calculation of all needed values for NTT/FFT multiplication. Similar "on the fly" computations can be tailored to each PQC scheme algorithm. Dilithium, for example, uses a two dimensional matrix A of 256-degree polynomials that is generated and then stored in memory thus consuming $k \times 1KB$ of memory (where k is one of the A matrix dimensions). This memory overhead could be avoided by generate rows or columns of A on the fly when needed to avoid the fully matrix memory storage need.
In addition to the above, the NTT version of polynomials, for example secret keys, may be significantly larger than their non-NTT counterpart. For example, in Dilithium, the secret keys polynomial vectors s_1 and s_2 include 256-degree polynomials with small coefficients (in the range of a few bits) while their NTT versions have coefficients of 23 bits each. As expected this also increases the memory footprint of LBC based PQC schemes. To reduce such memory overheads, simpler than NTT/FFT polynomial multiplication algorithms can be used that operate iteratively without the need of precomputations. To this end, polynomial multiplication can be realized apart from using NTT, using Karatsuba multiplication [28] in combination with Barrett [29] or Montgomery modular reduction. Alternatively, traditional schoolbook multiplication can be used. It should be noted however, that while storing polynomials in simpler forms might reduce memory sizes, it can raise potential security concerns and may lead to deviation from the standard because the NTT multiplication usage for polynomials in schemes like Kyber or Dilithium, is part of their specification [11], [13]. For the reduction of the polynomial sizes, compression techniques can be used. Such a technique, for example, is been used in Dilithium Digital Signature where the signature component c is been compressed (as a binary vector) to reduce the scheme's [24] memory footprint especially when c interacts with the Dilithium secret keys (a polynomial multiplication is performed between c and s_1, s_2 i.e $c \times s_1$ and $c \times s_2$.

Finally, regarding code-based schemes, which are not selected for standardisation by NIST but are further evaluated in NIST PQC Round 4, there are not many reported strategies in reducing the required memory, especially on embedded systems. In [20] and in [30] techniques are used to reduce the memory overhead of the McEliece (not Classic McEliece) and the Niederreiter Encryption Scheme respectively, the two schemes that are the predecessor of the NIST PQC submission Classic McEliece. Regarding Classic McEliece, in [22] a number of techniques, often drawn from [20], [30] are reported and are presented below:

- **Retrieving public key**: Introduction of an extended private key and an ad-hoc algorithm to *retrieve* the public key directly from the private key can be used. This means that the public key (which can grow up to 1 MB and is the main memory problem in Classic McEliece) does not need to be stored and it can be retrieved in "chunks", i.e single columns. This also allow the streaming of the key to a peer in the same time that the columns are retrieved.
- **Optimised Encapsulation**: An Encapsulation algorithm that does not need to process the whole public key at once but instead can process it in "chunks" can be applied in the PQC scheme implementation. Thus the encapsulation function can be processed while the key is streamed from a remote peer. The streaming process is so unrelated to the different parts of the matrix (the public key) that in fact, the authors of [22] state that with their technique "*it is not necessary to keep more that a single byte of the public key at a time*".
- **Memory-Efficient Matrix Inversion**: Assuming that the public key is retrieved in small chunks, the dominant factor of the memory utilisation is the size of the matrix S which must be retrieved from S^{-1}. We refer the reader to the Classic McEliece specification [31] and [22] for the notation we are using. An algorithm is described in [22] that can retrieve S from S^{-1} in 4 steps: First, an LU Decomposition is implemented, then an inversion on U and L, then a multiplication of $U^{-1}L^{-1}$ is performed and finally an undo on the permutation is made. This offers the ability to do this inversion almost in-place, without the need of extra memory for the inversion.

2) Artifact sizes: Another restriction on the use of PQC algorithms on memory-restrained devices is the large size of artifacts, i.e public keys, signatures or even private keys.

Regarding private keys, schemes specifications offer the option to make a memory-speed trade-off in these artifacts. Firstly the NIST API explicitly asks for the secret key to be concatenated with the public key. A simple trade-off would be to not store the public key as part of the secret key. A step further could be made, for example on Dilithium and Kyber, where private keys can be stored as a seed of 32 and 64 bytes respectively [11], [13] and re-run the key generation operation to fully derive them in case they are needed. Although a memory-speed trade-off may be possible in the Falcon specification, it is not something that is explicitly presented in the supporting documentation [12]. Sphincs+, Bike and HQC already have small secret keys that have small impact on memory usage. On the contrary, FrodoKEM has a huge private key and no memory-speed trade-off is described in its specification [17].

In public key and ciphertexts or signature, where optimizations are not straightforward, it is necessary to adopt different strategies to enable the use of these schemes on memory-constrained systems. These strategies are presented below:

- **Off-loading**: A common technique to reduce the required memory is to try to "off-load" the big artifacts on a

Authorized licensed use limited to: Universiteit van Amsterdam. Downloaded on October 16,2023 at 09:57:33 UTC from IEEE Xplore. Restrictions apply.

larger but slower memory, i.e. a flash memory. This would result in a slower access to the artifact but it would very likely help to make the given PQC algorithm suitable for memory-constrained system. In many embedded systems though, the flash memory is restricted and is not sufficiently large to store the whole artifact. In this case, another strategy must be adopted.

- **Streaming**: Another technique to avoid the memory storage of large artifacts is to stream them and do the required operations "on-the-fly". This technique introduces a new API that, breaks down the artifact, the public key or the signature, in "chunks" and sends or receives them in a streaming fashion. This is possible on hash-based signature schemes that may have small public keys (of a few bytes) but have signatures of many KB. The verification or signing process could then be in "chunks" as the signature is streamed in or out of the memory-restricted system. Regarding LBC based PQC schemes, this technique could be used when dealing with the large public keys of Dilithium or Falcon. The public keys or signatures can be re-arranged to enable the signing or verification in a streaming fashion. Finally, regarding Code-based schemes, like Classic McEliece, the largest obstacle of using such schemes, even on devices that offer RAM of a few hundreds of KB, is the huge public keys. The streaming technique can be used in this case too, thus enabling the use of Code-based schemes in these devices.

ACKNOWLEDGEMENT

The work of A.P. Fournaris & G. Tasopoulos has been funded from the EU Horizon Europe research and innovation programme SecOPERA under grant agreement No 101070599

IV. CONCLUSIONS

Quantum resistance transition should not be envisioned only for high end systems but also on the full spectrum for computing systems even on resource constrained devices. In this paper, we discussed the challenges and the limitation of porting PQC schemes in devices with approximately 10KB of memory. Initially we analyzed the memory requirements of PQC scheme implementation currently reported in literature and we assessed their suitability for such tiny devices. We then estimated the memory requirements of complete protocols (e.g. TLS 1.3) when they are made quantum resistant. We concluded our work by presenting strategies to reduce the memory usage of PQC implementations for embedded systems.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *FOCS*, 1994.

[2] G. Alagic, D. Cooper, Q. Dang, T. Dang, J. M. Kelsey, J. Lichtinger, Y.-K. Liu, C. A. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, D. Smith-Tone, and D. Apon, "Status report on the third round of the NIST post-quantum cryptography standardization process," 2022.

[3] G. Tasopoulos, J. Li, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, "Performance evaluation of post-quantum TLS 1.3 on resource-constrained embedded systems," in *Information Security Practice and Experience*, 2022.

[4] G. Tasopoulos, C. Dimopoulos, A. P. Fournaris, R. K. Zhao, A. Sakzad, and R. Steinfeld, "Energy consumption evaluation of post-quantum TLS 1.3 for resource-constrained embedded devices," Cryptology ePrint Archive, Paper 2023/506, 2023.

[5] BSI, "BSI technical reference TR-02102-1," 2023, https://www.bsi.bund.de/dok/TR-02102-en.

[6] D. Stebila and M. Mosca, "Post-quantum key exchange for the internet and the Open Quantum Safe project," in *Lecture Notes in Computer Science*, 2017.

[7] M. J. Kannwischer, P. Schwabe, D. Stebila, and T. Wiggers, "Improving software quality in cryptography standardization projects," in *EuroS&P Workshops*, 2022.

[8] M. J. Kannwischer, R. Petri, J. Rijneveld, P. Schwabe, and K. Stoffelen, "PQM4: Post-quantum crypto library for the ARM Cortex-M4," https://github.com/mupq/pqm4.

[9] mupq, "pqm3, sister project of pqm4 and pqclean," 2021, https://github.com/mupq/pqm3.

[10] riscv, "the pqriscv counterpart of pqm4," 2021, https://github.com/mupq/pqriscv.

[11] Dilithium team, "Diltihium specifications v3.1," 2021, https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[12] Faclon team, "Falcon specifications v1.2," 2020, https://falcon-sign.info/falcon.pdf.

[13] Kyber team, "Kyber specifications v3.02," 2021, https://pq-crystals.org/kyber/data/kyber-specification-round3-20210804.pdf.

[14] Sphincs+ team, "Sphincs+ specifications v3.1," 2022, https://sphincs.org/data/sphincs+-r3.1-specification.pdf.

[15] HQC team, "HQC specifications version 30/04/2023," 2023, https://pqc-hqc.org/documentation.html.

[16] Bike team, "Bike specifications v5.1," 2022, https://bikesuite.org/files/v5.0/BIKE_Spec.2022.10.10.1.pdf.

[17] FrodoKEM team, "FrodoKEM specification v04/06/2021," 2021, https://frodokem.org/files/FrodoKEM-specification-20210604.pdf.

[18] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8446

[19] M.-S. Chen and T. Chou, "Classic McEliece on the ARM Cortex-M4," Cryptology ePrint Archive, Paper 2021/492, 2021.

[20] T. Eisenbarth, T. Güneysu, S. Heyse, and C. Paar, "MicroEliece: McEliece for embedded devices," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, 2009.

[21] F. Strenzke, "How to implement the public key operations in code-based cryptography on memory-constrained devices," Cryptology ePrint Archive, Paper 2010/465, 2010.

[22] J. Roth, E. Karatsiolis, and J. Krämer, "Classic McEliece implementation with low memory footprint," Cryptology ePrint Archive, Paper 2021/138, 2021.

[23] R. Gonzalez, A. Hülsing, M. J. Kannwischer, J. Krämer, T. Lange, M. Stöttinger, E. Waitz, T. Wiggers, and B.-Y. Yang, "Verifying post-quantum signatures in 8 kB of RAM," Cryptology ePrint Archive, Paper 2021/662, 2021.

[24] J. W. Bos, J. Renes, and A. Sprenkels, "Dilithium for memory constrained devices," Cryptology ePrint Archive, Paper 2022/323, 2022.

[25] R. Niederhagen, J. Roth, and J. Wälde, "Streaming SPHINCS+ for embedded devices using the example of TPMs," Cryptology ePrint Archive, Paper 2021/1072, 2021.

[26] Dilithium on-the-fly, "Pull request on pqm4: Low memory Dilithium impl." 2022, https://github.com/mupq/pqm4/pull/222.

[27] T. Vandervelden, R. De Smet, K. Steenhaut, and A. Braeken, "SHA3 and Keccak variants computation speeds on constrained devices," *Future Generation Computer Systems*, 2022.

[28] A. Karatsuba, "Multiplication of multidigit numbers on automata," in *Soviet physics doklady*, vol. 7, 1963, pp. 595–596.

[29] P. Barrett, "Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[30] S. Heyse, "Low-Reiter: Niederreiter encryption scheme for embedded microcontrollers," in *Post-Quantum Cryptography*, 2010.

[31] Classic McEliece team, "Classic McEliece specification 23/10/2022," 2022, https://classic.mceliece.org/mceliece-spec-20221023.pdf.

Authorized licensed use limited to: Universiteit van Amsterdam. Downloaded on October 16,2023 at 09:57:33 UTC from IEEE Xplore. Restrictions apply.