A Performant Quantum-Resistant KEM for Constrained Hardware: Optimized HQC

Ridwane Aissaoui, Jean-Christophe Deneuville, Christophe Guerber, Alain Pirovano

A Performant Quantum-Resistant KEM for Constrained Hardware: Optimized HQC

Ridwane Aissaoui[1][a], Jean-Christophe Deneuville[1][b], Christophe Guerber[2][c] and Alain Pirovano[1][d]

[1]*Fédération ENAC ISAE-SUPAERO ONERA, Université de Toulouse, France*

[2]*Direction de la Technique de l'Innovation, Toulouse, France*

fi

Keywords: HQC: Hamming Quasi-Cyclic, Optimization, PQC: Post-Quantum Cryptography, KEM: Key Encapsulation Mechanism.

Abstract: Secure Key Encapsulation Mechanisms (KEMs) are necessary for providing authentication and confidentiality through symmetrical encryption. The emergence of quantum computers is a threat to current KEM standards, therefore new quantum-resistant algorithms have been developed in recent years. One of these propositions is the code-based Hamming Quasi-Cyclic (HQC) algorithm. However, a lightweight version of this algorithm is required to run on low-performance systems such as Internet of Things (IoT) devices or small Unmanned Aerial Vehicles (UAVs). This article presents an algorithmic optimization of the HQC algorithm applied on constrained hardware. The goal is to improve the performance for real-life applications, and thus the test bed uses a Real-Time Operating System (RTOS) to emulate a system able to complete complex tasks. This optimization reduces the completion time of key generation, encapsulation, and decapsulation by a factor of 10, and reduces significantly the Random Access Memory (RAM) usage for the algorithm. These improvements make HQC viable for real-life applications on constrained hardware, and the performance could be further improved by using hardware-specific optimizations.

1 INTRODUCTION

1.1 Security of Constrained Devices

In recent years, the number of connected objects, including IoT devices and UAVs, has surged exponentially, leading to increased wireless communication and security challenges. UAVs require secure video feeds and authenticated commands to prevent unauthorized access (Aissaoui et al., 2023). Similarly, IoT devices can leak sensitive information without secure communications(Naru et al., 2017).

While larger systems rely on standard cryptographic protocols, implementing them on small, constrained devices consumes significant computational and memory resources, potentially disrupting system operations (Thakor et al., 2021). Thus, optimizing cryptographic algorithms for resource-constrained systems is crucial to minimize resource consumption and ensure compatibility.

This study focuses on the ARM Cortex-M4, a 32-bit Microcontroller Unit (MCU) suitable for UAV flight controllers, embedded automotive systems like Engine Control Units (ECUs), and infotainment systems. Popular devices such as Elle0, LizaMX, PixHawk PX4, and Crazyflie feature an ARM Cortex-M4 MCU with 1 Mbyte of Flash memory and 192 kB of RAM, which also meets the requirements for many IoT devices (Yiu and Frame, 2013).

1.2 Post-Quantum Cryptography

The emergence of quantum computing poses a significant threat to current cryptography standards. Shor's quantum algorithm (Shor, 1994) provides a polynomial time solution to the factorization problem, compromising the security of Rivest Shamir Adleman (RSA) keys and Diffie-Hellman (DH)-based primitives. As a result, there is a pressing need for post-quantum solutions, resistant to attacks performed with quantum computing abilities.

[a] https://orcid.org/0000-0003-1567-8458
[b] https://orcid.org/0000-0002-5128-6729
[c] https://orcid.org/0000-0003-4127-7615
[d] https://orcid.org/0000-0002-5254-0178

Aissaoui, R., Deneuville, J., Guerber, C. and Pirovano, A.
A Performant Quantum-Resistant KEM for Constrained Hardware: Optimized HQC.
DOI: 10.5220/0012757800003767

In *Proceedings of the 21st International Conference on Security and Cryptography (SECRYPT 2024)*, pages 668-673
ISBN: 978-989-758-709-2; ISSN: 2184-7711

The quantum threat extends to key exchange and signature schemes, where RSA, Elliptic Curve Cryptography (ECC), and DH schemes are no longer secure. Post-quantum cryptography offers five main public-key primitives: (Euclidean) Lattices, (Error Correcting) Codes, Hash functions, Multivariate polynomials, and (Elliptic curves) Isogeny.

In 2017, the National Institute of Standards and Technology (NIST) initiated a standardization process for post-quantum cryptography, with the first standards announced in July 2022. This process notably evaluated KEM candidates. A Lattice-based candidate - CRYSTALS-Kyber (Avanzi et al., 2019) - was selected, alongside three competing code-based propositions: HQC (Aguilar et al., 2023), BIKE (Aragon et al., 2022), and Classic McEliece (Albrecht et al., 2022). Despite the selection, widespread adoption of these standards will require time as public trust in their security develops.

1.3 Post-Quantum Cryptography on Constrained Devices

While post-quantum cryptographic algorithms address the threat from quantum computing, integrating them into constrained devices poses challenges (Kumari et al., 2022). These devices operate with limited computational power and memory, making traditional cryptographic protocols designed for more powerful systems less applicable. KEMs provide authentication, integrity, and confidentiality, but are vulnerable to quantum attacks and are being standardized by NIST.

One KEM candidate is HQC but its lack of optimized implementations hinders its integration in constrained hardware like the ARM Cortex-M4. Motivated by this gap in the existing literature, we sought to identify potential optimizations for HQC on constrained devices and assess their practical viability for real-world deployment. An algorithmic approach is not hardware or software-specific, which allows the optimization to be portable on different systems. This paper presents algorithmic optimizations of HQC in the context of the feasibility of implementing post-quantum cryptographic algorithms on constrained systems. Through our investigations, we aim to provide an optimized version of HQC that can be recommended as a secure and efficient encryption scheme tailored specifically for resource-constrained systems like UAVs and IoT nodes.

2 HQC

HQC is a code-based public-key encryption scheme. The HQC KEM ensures a secure key exchange between parties. The sender encrypts a shared secret with the public key of the receiver, which is used to produce a key for symmetrical encryption. The message can only be deciphered with the receiver's secret key. The sender is guaranteed that subsequent communication using the shared key is authenticated, as only the receiver can retrieve it. The following paragraphs detail the workings of HQC based on the NIST fourth round submission (Aguilar et al., 2023):

2.1 Notations

Vectors are depicted in lowercase bold and matrices in uppercase bold. For instance, $\mathbf{x}$ is a vector, $\mathbf{X}$ is a matrix. $\mathbf{I}_n$ represents the identity matrix of size $n \times n$. $\mathcal{H}()$ represents a hash function. We use $\mathbb{F}_2$ to represent the binary finite field and $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$ to denote the quotient ring where vectors and operations of HQC are defined. An element of $\mathbb{F}_2^n$ can be depicted as a n-dimension vector $\mathbf{x} = (x_0, x_1, ..., x_{n-1})$ in $\mathbb{F}_2$ or as a polynomial $\mathbf{x} = \sum_{i=0}^{n-1} x_i X^i$ in $\mathcal{R}$. The Hamming weight of $\mathbf{x}$, denoted $wt(\mathbf{x})$, is defined as the number of non-zero coefficients. Formally, $wt(\mathbf{x}) = \#\{x_i \neq 0\}$. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_2^n$, the addition $\mathbf{a} = \mathbf{x} + \mathbf{y} \in \mathbb{F}_2^n$ is defined as $a_i = x_i + y_i \mod 2$. Vector multiplication is defined as its analog over $\mathcal{R}$, that is if $\mathbf{a} = \mathbf{x} \times \mathbf{y} \in \mathbb{F}_2^n$, then $a_k = \sum_{i+j \equiv k \mod n} x_i \cdot y_j \mod 2$ for $k = 0, ..., n-1$. The parameters n, w, w_r depend on the security level and can be found in Table 1.

2.2 Key Generation

A vector $\mathbf{h}$ is randomly sampled, representing the foundation for generating a circulant matrix and, consequently, a systematic quasi-cyclic code with an index of 2. Specifically, let $\mathbf{h} = (h_0, \ldots, h_{n-1})$. The matrix

$$\mathrm{rot}(\mathbf{h}) = \begin{bmatrix} h_0 & h_{n-1} & \cdots & h_1 \\ h_1 & h_0 & \cdots & h_2 \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-1} & h_{n-2} & \cdots & h_0 \end{bmatrix}$$

is a circulant matrix, and $\mathbf{H} = [\mathbf{I}_n | \mathrm{rot}(\mathbf{h})]$ forms the parity-check matrix of a systematic quasi-cyclic code with an index of 2. The secret key consists of two vectors, $\mathbf{x}$ and $\mathbf{y}$, sampled with a specified low weight w. $(\mathbf{x}, \mathbf{y})$ can be viewed as a secret error with a low Hamming weight relative to n (roughly, $w = O(\sqrt{(n)})$). The syndrome, denoted as $\mathbf{s}$, is given by $\mathbf{s} = (\mathbf{x}, \mathbf{y})\mathbf{H}^T = \mathbf{x} + \mathbf{y} \times \mathrm{rot}(\mathbf{h})^T = \mathbf{x} + \mathbf{h} \times \mathbf{y}$. The public key comprises the vector $\mathbf{h}$ and the syndrome $\mathbf{s}$.

2.3 Encryption and Encapsulation

Three vectors $\mathbf{r_1}$, $\mathbf{r_2}$, and $\mathbf{e}$ are randomly sampled with a specified low weight w_r (of order w). The syndrome $\mathbf{u}$ of $(\mathbf{r_1}, \mathbf{r_2})$ is then computed. Formally, $\mathbf{u} = \mathbf{r_1} + \mathbf{h} \times \mathbf{r_2}$. The message is encoded using a concatenated Reed–Muller and Reed–Solomon code (of generator matrix $\mathbf{G}$), and it is further modified by $\mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$ to obtain $\mathbf{v}$. Formally, $\mathbf{v} = \mathbf{m} \times \mathbf{G} + \mathbf{s} \times \mathbf{r_2} + \mathbf{e}$. The final ciphertext is composed of $\mathbf{u}$ and $\mathbf{v}$, i.e., $\mathbf{c} = (\mathbf{u}, \mathbf{v})$. For IND-CCA2 encapsulation, a seed m is encrypted to produce c. The shared secret is derived from m and c, and the ciphertext is formed as $(c, \mathcal{H}(m))$. More details for the encapsulation process are available in Section 2.3.2 of (Aguilar et al., 2023).

2.4 Decryption and Decapsulation

The process involves decoding $\mathbf{v} - \mathbf{u} \times \mathbf{y}$ using the concatenated Reed–Muller and Reed–Solomon code. The message can be correctly decoded when the Hamming weight of the given element is less than the minimum distance of the code. The probability that the message cannot be decoded from $\mathbf{v} - \mathbf{u} \times \mathbf{y}$ is negligible in the security parameter. For decapsulation, $(c, \mathcal{H}(m))$ is received. m is retrieved from c, and verified with $\mathcal{H}(m)$. Then the shared secret is derived from m. More details for the decapsulation process are available in Section 2.3.2 of (Aguilar et al., 2023).

Table 1: Parameter sets for HQC. n represents the length of the vector (polynomial). w indicates the weight of vectors $\mathbf{x}$ and $\mathbf{y}$, whereas w_r denotes the weight of vectors $\mathbf{r_1}$, $\mathbf{r_2}$, and $\mathbf{e}$. The security level for each parameter set is specified.

Instance	n	w	w_r	Security
hqc128	17,669	66	75	128
hqc192	35,851	100	114	192
hqc256	57,637	131	149	256

3 RELATED WORK

The PQClean project (Kannwischer et al., 2022) provided a well-organized implementation of HQC for the ARM Cortex-M4 platform. Their analysis revealed deficiencies in reference implementations included in NIST submission packages. Addressing those issues, many of which were related to non-adherence to software engineering practices required significant effort. While PQClean offers a deployable implementation, it lacks optimization.

The pqm4 project (Kannwischer et al., 2019) optimized various post-quantum candidates on ARM Cortex-M4, yet HQC lacks similar attention. Only PQClean's HQC runs on ARM Cortex-M4, underscoring the need for further optimization efforts for equitable evaluations.

(Deshpande et al., 2023) implements similar algorithmic improvements, but mostly outlines hardware optimizations for HQC on Xilinx Artix 7 FPGA. This restricts its applicability to other Central Processing Unit (CPU) architectures, challenging broader utilization and comparison across platforms.

4 METHODS

4.1 Hardware and Software

The chosen platform is the STM32F4 Discovery development board. It is equipped with a Cortex-M4 microcontroller. It features a clock frequency of 128 MHz, 192 kB of RAM, and 1 MB of flash memory, closely resembling the capabilities of a low-performance system, such as UAVs.

For the software foundation, we opted for the ChibiOS RTOS. This choice is consistent with Paparazzi, an open-source UAV autopilot software, which also uses it. This operating system provides the essential tools to implement algorithms on the development board and monitor their execution time.

The performance evaluation of KEMs is based on two metrics: computing cycles and memory usage. We assess these metrics for the key generation, encapsulation, and decapsulation processes. Our test bed provides the maximum number of cycles and memory usage for each process.

4.2 Algorithmic Optimization

The algorithms presented in this section are not directly applicable. For the sake of clarity, they are presented with vector representations of individual bits, whereas they are arrays of 64-bit unsigned integers in the code, and the smallest directly accessible value is a byte. Modifying individual bits requires data manipulation that is not detailed here.

During the HQC processes of key generation, encapsulation, and decapsulation, there exist variables with a very low Hamming weight. They only possess a few non-zero bits therefore using arrays of indexes for storing them reduces the size of those variables. The new sparse vector generator is based on the reference implementation. No changes were made to the generation of entropy. Only the format of the result is modified, going from a n-size vector of bits to a w-size list of integers. The complexity is $O(w) = O(\sqrt{N})$ instead of $O(N+w) = O(N)$ for the original algorithm.

In terms of memory, the low Hamming weight vector is now stored in $w*16$ bits instead of N bits.

Data: w: Hamming weight, SIZE: size of the vector
Result: x: list of indexes of non-zero values
initialization;
for $(i = 0, i < w, i++)$ **do**
  $x[i]$ = random(SIZE);
  if $x[i]$ *in* $x[0 : i-1]$ **then**
    $i--$;
  end
end

Algorithm 1: Sparse vector generator.

Polynomial addition uses the list of indexes instead of the entire sparse vector. It iterates on each index, then XORes 1 to the corresponding bit in u. It modifies u in place as every instance of addition does not require saving the initial vector. The new addition algorithm is detailed in Algorithm 2. The complexity is reduced from $O(N)$ to $O(w) = O(\sqrt{N})$, and the memory usage is reduced by N bits (excluding the savings made by the new storage method, which is $N-(16*w)$).

Data: u: Vector represented by an array of bits, x: indexes of non-zero values of a low Hamming weight vector
Result: u: result of the addition
for $(index\ in\ x)$ **do**
  $u[i] = u[i]$ XOR 1;
end

Algorithm 2: Polynomial addition.

Polynomial multiplication is the most expensive part of HQC, both in terms of complexity and memory usage. The PQClean implementation, as well as the reference implementation, uses the Karatsuba algorithm (Karatsuba and Ofman, 1962) for polynomial multiplication. However, this can be further improved by exploiting the properties of low Hamming weight vectors. As seen in Algorithm 3, the polynomial multiplication can be performed by rotating the vector u to the left by a value corresponding to the indexes given by x and then XORing those rotations together. In practice, we build temporary 64-bit buffers of a fraction of the resulting rotated vector. These buffers are then successively XORed to the corresponding 64-bit block in the result. This method allows the rotation to remain constant-time, as each iteration on a given index requires exactly $n/64+1$ times the construction of the buffer. Thus a timing analysis of the operation only leaks the size of x, which is already public. This new algorithm has a complexity of $O(N*w) = O(N\sqrt{N})$ instead of $O(N^{\log_2(3)})$ for Karatsuba. For memory consumption, Karatsuba uses $16*N$ bits for temporary variables, whereas Algorithm 3 only uses N bits.

Data: u: Vector represented by an array of bits, x: indexes of non-zero values of a low Hamming weight vector
Result: v: result of the multiplication
$v = [0]\times$ sizeof(u);
for $(index\ in\ x)$ **do**
  $v = v$ XOR $rotl(u, index)$;
end

Algorithm 3: Polynomial multiplication.

5 RESULTS

5.1 Performance Analysis

Table 2 provides a summary of the optimized HQC performance compared to its PQC competitors, as well as the RSA and Elliptic Curve Diffie-Hellman (ECDH) standards. RSA and ECDH are not quantum-resistant and have been extensively researched and optimized over many years. They serve as the current standards for asymmetric encryption and key exchange and are considered secure against non-quantum computing attacks. For ECDH, the computation of the shared secret between parties is equivalent to the encapsulation and decapsulation processes of KEM. We compared this single result to both encapsulation and decapsulation for KEM. RSA key generation, which requires the generation of large prime numbers, is considered infinite in the comparison due to its extensive computational requirements. Therefore, using RSA would necessitate key generation by entities with stronger hardware.

All three algorithms improve memory performance. The multiplication optimization enables running HQC-192, previously impossible due to RAM constraints. Results for HQC-192 will be presented. In addition to gaining 4 kB in flash memory, a significant 35 kB of RAM is saved compared to the PQClean implementation.

For computational complexity, the maximum number of cycles encountered during 10,000 iterations of each scenario is presented. The new algorithms are expected to run in constant time, so slight variations in cycles may result from other parts of the implementation or delays induced by the RTOS. To ensure reliability, the maximums encountered over the iterations are presented, reflecting the computational cost for evaluating the real-time needs of the algorithms.

Table 2: Global performance comparison with current standards and Post-Quantum Cryptography (PQC) competitors. The Keygen, Encapsulation, and Decapsulation columns include the maximum (over 10000 iterations) number of cycles to complete the process, and the resulting execution time in ms. RSA Keygen is considered infinite, as explained in Section 5.1. The RAM usage numbers are round due to the static allocation required by the Operating System (OS).

Algorithm	Max Cycles Keygen	Max Cycles Encapsulation	Max Cycles Decapsulation	RAM usage (bytes)	Flash memory usage (bytes)
HQC-128 (PQClean)	48,030,414 (285.9 ms)	96,874,954 (576.7 ms)	145,737,544 (867.5 ms)	85,000	33,692
HQC-128 (opt)	1,837,507 (11 ms)	4,878,515 (29.1 ms)	7,502,580 (44.7 ms)	50,000	29,484
BIKE-1	36,895,891 (219.6 ms)	4,309,361 (25.7 ms)	73,127,121 (435.8 ms)	90,000	121,668
Kyber-2	747,259 (4.5 ms)	898,939 (5.4 ms)	798,862 (4.8 ms)	10,000	18,332
ECDH x25519	3,587,785 (21.3 ms)	3,564,808 (21.3 ms)	3,564,808 (21.3 ms)	2,048	19,932
RSA 2048	∞	11,216,240 (66.8ms)	118,744,894 (706.8 ms)	2,048	15,784

Key generation now takes less than 4% of the time required for the non-optimized version, while encapsulation and decapsulation times are reduced to 5% of their original duration. This algorithmic optimization results in a substantial improvement, even exceeding expectations based on algorithm complexity.

Several factors contribute to this improvement. Our multiplication algorithm works with 64-bit blocks, resulting in fewer operations compared to the recursive Karatsuba algorithm, which incurs a higher complexity. Karatsuba also uses more basic operations for low-level polynomial multiplication, contributing to exponential increases in completion time. Furthermore, the addition optimization further accelerates processes by 15 to 20%, allowing the removal of the original sparse vector generator from the code. Finally, algorithm 1 provides a net gain, as the original generation algorithm already generated the list of indexes before filling an entire vector.

The comparison yields several insights. First, non-PQC standards require less memory space than post-quantum KEMs, particularly in RAM, which may be preferable for highly constrained systems. In flash memory, the overhead from using any algorithm is similar, except for BIKE, which requires significantly more space. CRYSTALS-Kyber is the most computationally efficient algorithm, followed by ECDH. HQC takes a similar amount of time as ECDH, while BIKE is faster for encapsulation but slower for key generation and decapsulation. RSA is the least efficient computationally, and only the PQClean version of HQC is slower. With optimization, HQC appears more performant than BIKE, being slightly slower for encapsulation but faster for key generation and decapsulation, while requiring less space in RAM and flash memory. CRYSTALS-Kyber-3 shows the best performance for higher security levels. HQC-192 has comparable computational complexity to ECDH x448.

5.2 Discussion

Our optimized HQC offers secure key exchange on ARM Cortex-M4 systems with sufficient RAM and flash memory. CRYSTALS-Kyber remains more efficient, even surpassing current standards. However, HQC lacks hardware acceleration optimization. Implementing it with NEON or AVX2 instructions could enhance its competitiveness. This version of HQC meets the performance needs for UAV unicast communications, IoT devices, data-gathering sensors, and medical devices like cardiac monitors. Systems with Cortex-M4 or similar processing units, including smart devices and industrial systems, can now consider optimized HQC as a viable alternative to CRYSTALS-Kyber for ensuring confidentiality and authentication in their communications.

It also offers flexibility across diverse hardware platforms. Implemented at the high-level code, it ensures portability to systems with ChibiOS support. The algorithmic enhancements are independent of specific system calls, facilitating integration with any OS that supports PQClean. Our optimized HQC has a smaller footprint than the original PQClean version, making it suitable for resource-limited systems. Performance may vary across platforms, but its relative efficiency compared to other algorithms remains consistent on 32-bit CPUs. This broad compatibility makes it relevant for widespread usage.

The test bed, using ChibiOS, introduces variability in outcomes compared to other CPU-RTOS combinations, as shown by PQM4 benchmarks. Real hardware implementation limits monitoring capabilities, making actual RAM assessment unfeasible. Our

code includes tasks for monitoring and sharing values, marginally impacting results. While algorithmic optimization ensures portability, a hardware-specific approach, like the Xilinx Artix 7 CPU (Deshpande et al., 2023), would yield better results with hardware acceleration. The optimized HQC implementation runs in constant time but must be evaluated against other side-channel attacks for potential vulnerabilities.

6 CONCLUSIONS AND FUTURE WORKS

This article presents algorithmic optimization for the HQC post-quantum KEM on the ARM Cortex-M4 using the ChibiOS RTOS. The optimization is portable and compatible with various operating systems, making it applicable to a wide range of hardware platforms. Performance improvements include a 96% reduction in key generation time and 95% reductions in encapsulation and decapsulation times. These enhancements make HQC a viable solution for resource-constrained systems, surpassing BIKE as the most performant code-based KEM. While CRYSTALS-Kyber remains the top-performing algorithm, further optimization could narrow the performance gap.

Future work will explore hardware-specific optimizations using NEON for ARM or AVX2 for Intel CPUs. Additionally, BIKE will be investigated for a similar optimization strategy. Evaluation across different hardware and OSs will validate the results.

Overall, this optimized version of HQC offers enhanced cryptographic capabilities for applications such as IoT and UAVs without compromising system availability.

REFERENCES

Aguilar, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Bos, J., Deneuville, J.-C., Dion, A., Gaborit, P., Lacan, J., Persichetti, E., Robert, J.-M., Véron, P., and Zémor, G. (2023). Hamming quasi-cyclic (HQC) fourth round version. *Submission to the NIST's post-quantum cryptography standardization process*.

Aissaoui, R., Deneuville, J.-C., Guerber, C., and Pirovano, A. (2023). A survey on cryptographic methods to secure communications for uav traffic management. *Vehicular Communications*.

Albrecht, M. R., Bernstein, D. J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., Von Maurich, I., Misoczki, R., Niederhagen, R., Paterson, K. G., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Tjhai, C. J., Tomlinson, M., and Wang, W. (2022). Classic mceliece: conservative code-based cryptography. *Submission to the NIST's post-quantum cryptography standardization process*.

Aragon, N., Aguilar Melchor, C., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.-C., Gaborit, P., Ghosh, S., Gueron, S., Güneysu, T., Misoczki, R., Persichetti, E., Richter-Brockmann, J., Sendrier, N., Tillich, J.-P., Vasseur, V., and Zémor, G. (2022). Bike: bit flipping key encapsulation. *Submission to the NIST's post-quantum cryptography standardization process*.

Avanzi, R., Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J. M., Schwabe, P., Seiler, G., and Stehlé, D. (2019). Crystals-kyber algorithm specifications and supporting documentation. *Submission to the NIST's post-quantum cryptography standardization process*.

Deshpande, S., Xu, C., Nawan, M., Nawaz, K., and Szefer, J. (2023). Fast and efficient hardware implementation of HQC. In *Proceedings of the Selected Areas in Cryptography*.

Kannwischer, M. J., Rijneveld, J., Schwabe, P., and Stoffelen, K. (2019). pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4. Cryptology ePrint Archive, Paper 2019/844.

Kannwischer, M. J., Schwabe, P., Stebila, D., and Wiggers, T. (2022). Improving software quality in cryptography standardization projects. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE.

Karatsuba, A. and Ofman, Y. (1962). Multiplication of many-digital numbers by automatic computers. *Dokl. Akad. Nauk SSSR*.

Kumari, S., Singh, M., Singh, R., and Tewari, H. (2022). Post-quantum cryptography techniques for secure communication in resource-constrained internet of things devices: A comprehensive survey. *Software: Practice and Experience*.

Naru, E. R., Saini, H., and Sharma, M. (2017). A recent review on lightweight cryptography in iot. In *2017 international conference on I-SMAC (IoT in social, mobile, analytics and cloud)(I-SMAC)*.

Shor, P. (1994). Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*.

Thakor, V. A., Razzaque, M. A., and Khandaker, M. R. (2021). Lightweight cryptography algorithms for resource-constrained iot devices: A review, comparison and research opportunities. *IEEE Access*.

Yiu, J. and Frame, A. (2013). Cortex-M Processors and the Internet of Things (IoT). *ARM whitepaper*.