FrodoKEM based on multicore implementation: enhancing performance and memory efficiency in IoT applications

Marcos V. C. Amorim[1], **Vinícius Lagrota**[2], **Mateus de L. Filomeno**[1], **Moisés V. Ribeiro**[1]

[1] Electric Engineering Department
Federal University of Juiz de Fora (UFJF) – Juiz de Fora, MG – Brazil

[2]Research and Development Center for Communication Security (CEPESC)
Brasília, DF – Brazil

`marcos.amorim@estudante.ufjf.br, vinicius.lagrota@gmail.com, delimafilomeno@gmail.com, moises.ribeiro@ufjf.br`

Abstract. *This paper presents a multicore implementation of FrodoKEM, a post-quantum cryptographic scheme, targeting near real-time Internet of Things (IoT) applications. By combining tiled matrix multiplication and data parallelism, we enable FrodoKEM to efficiently operate under the memory and performance constraints of multicore microcontrollers. Implemented on the ESP32-S3, the single-core version demonstrates feasibility through memory optimization, while the dual-core version achieves a* 44.5% *reduction in execution time, meeting the latency requirements of IoT systems.*

1. Introduction

Cryptography is essential for securing digital communications, playing a key role in protecting sensitive data across various applications, including financial transactions, e-commerce, and secure messaging. In IoT systems, where interconnected devices exchange critical data, cryptographic protocols ensure data confidentiality and integrity. For example, in smart grids, secure communication between utility providers and smart meters is vital [Costa et al. 2022].

The advent of cryptographically relevant quantum computers (CRQCs) threatens conventional cryptographic schemes such as Rivest-Shamir-Adleman (RSA) and elliptic curve cryptography (ECC), which are vulnerable to quantum algorithms like Shor's [Shor 1994]. In response, post-quantum cryptography (PQC) has emerged as a solution based on quantum-resistant mathematical problems. Since 2017, National Institute for Standards and Technology (NIST) has led efforts to standardize PQC, culminating in the selection of Module-Lattice-Based Key Encapsulation Mechanism (ML-KEM) as the primary Key Encapsulation Mechanism (KEM) for general-purpose use. Nevertheless, certain scenarios—such as critical infrastructure—demand conservative designs prioritizing long-term confidentiality.

FrodoKEM [Alkim et al. 2020], grounded in the well-established Learning With Errors (LWE) problem, was recommended by the Federal Office for Information Security (BSI) for such high-assurance applications [BSI 2020]. However, FrodoKEM's resource-intensive nature—especially in terms of memory and execution time—challenges its deployment in embedded systems typical of IoT ecosystems.

Several studies have proposed optimizations to mitigate these challenges. These include GPU-based Advanced Encryption Standard (AES) acceleration [Lee et al. 2022], matrix multiplication optimizations [Bos et al. 2021], hardware/software co-designs on RISC-V and SoC platforms [Karl et al. 2022, Costa et al. 2022], vectorized implementations using ARMv8 NEON [Kwon et al. 2021], and memory-efficient embedded versions [Bos et al. 2023]. Yet, multicore implementations of FrodoKEM remain underexplored.

This work proposes a multicore approach combining tiled matrix multiplication and data parallelism to enhance FrodoKEM's feasibility and performance in constrained microcontrollers. Implementations on the ESP32-S3 demonstrate that tiled multiplication enables FrodoKEM execution within memory limits, while data parallelism achieves a 44.5% reduction in execution time. The proposed method supports near real-time execution and complies with FrodoKEM specifications, providing a viable path for deploying quantum-resistant security in IoT environments.

The remainder of this paper is organized as follows: Section 2 outlines FrodoKEM operations. Section 3 presents the proposed multicore implementation. Section 4 details the practical setup. Section 5 discusses results and conclusions.

2. The FrodoKEM scheme

FrodoKEM [Alkim et al. 2020] is a lattice-based cryptographic scheme built upon the hardness of the LWE problem [Regev 2010]. We adopt FrodoKEM's original notation [Alkim et al. 2020]: matrices are denoted by uppercase bold letters, vectors by lowercase bold letters, $\mathbb{Z}_q$ represents integers modulo q, and $||$ indicates concatenation.

FrodoKEM emphasizes conservative security by relying on simple operations (e.g., addition, multiplication) and omitting structured lattices. It supports three parameter sets: FrodoKEM-640, -976, and -1344, targeting security levels equivalent to AES-128, -192, and -256, respectively. However, its large matrix sizes—summarized in Table 1—pose significant challenges for embedded implementations.

Table 1. Public key, private key, and ciphertext sizes for FrodoKEM (in bytes).

	Public key size	**Private key size**	**Ciphertext size**
FrodoKEM-640	9616	19888	9720
FrodoKEM-976	15632	31296	15744
FrodoKEM-1344	21520	43088	21632

The scheme consists of three main operations: key generation, encapsulation, and decapsulation. In key generation, the main computational task is forming matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times\overline{n}}$ as

$$\mathbf{B} \leftarrow \underbrace{\mathbf{AS}}_{\mathbf{D}} + \mathbf{E}, \tag{1}$$

where $\mathbf{A}$ is deterministically generated from a pseudorandom seed, and $\mathbf{S}$, $\mathbf{E}$ are sampled from a discrete Gaussian distribution χ.

In encapsulation, additional noise matrices $\mathbf{S}'$, $\mathbf{E}'$, and $\mathbf{E}''$ are sampled. The following core operations are performed:

$$\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}', \quad \mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''. \tag{2}$$

A message μ is encoded into matrix $\mathbf{C}$ using $\mathbf{V}$, forming part of the ciphertext alongside $\mathbf{B}'$.

During decapsulation, the ciphertext is unpacked and used to recover μ' via:

$$\mathbf{M} \leftarrow \mathbf{C} + \mathbf{B}'\mathbf{S}. \tag{3}$$

A re-encapsulation process computes:

$$\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}', \quad \mathbf{C}' \leftarrow \text{Encode}(\mu'). \tag{4}$$

If $\mathbf{B}'' = \mathbf{B}'$ and $\mathbf{C}' = \mathbf{C}$, the shared secret is derived deterministically; otherwise, a random value is returned.

FrodoKEM's performance bottlenecks arise primarily from matrix multiplications involving $\mathbf{A}$, which motivate the optimizations explored in this work.

3. Proposal of multicore implementation for FrodoKEM

This section presents a multicore implementation strategy for FrodoKEM, targeting its performance bottlenecks. According to [Costa et al. 2022], operations involving matrix $\mathbf{A}$ dominate runtime when using Secure Hash Algorithm and Keccak (SHAKE)-128, accounting for over 90% of total execution time. Specifically, $\mathbf{B} \leftarrow \mathbf{A}\mathbf{S} + \mathbf{E}$ consumes 8.22%, while $\mathbf{B}' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$ and $\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$ together take 30.94%. The standalone generation of $\mathbf{A}$ via SHAKE-128 accounts for 53.90%.

To address these costs, we propose a combined approach using data parallelism and tiled matrix multiplication. Data parallelism distributes matrix operations across U cores, enabling concurrent processing and improved performance. However, FrodoKEM also imposes high memory demands due to large matrix dimensions—especially for matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times n}$.

To overcome this, we apply tiled multiplication, which divides $\mathbf{A}$ into smaller row-wise blocks (tiles) that can fit into limited internal memory. Each tile is generated on-the-fly using SHAKE-128, multiplied by $\mathbf{S}$, and accumulated into the resulting matrix $\mathbf{B}$ alongside $\mathbf{E}$. The next tile then overwrites the previous one, avoiding the need to store the entire matrix $\mathbf{A}$ in memory. This technique significantly reduces the memory footprint without altering the FrodoKEM specification, making the scheme viable for resource-constrained microcontrollers.

Subsection 3.1 describes the multicore and tiled implementation of $\mathbf{A}\mathbf{S}$, while Subsection 3.2 details the strategy for computing $\mathbf{S}'\mathbf{A}$[1].

3.1. A by S matrix multiplication

To compute the matrix multiplication of $\mathbf{A}$ by $\mathbf{S}$, which results in the matrix $\mathbf{D} \in \mathbb{Z}_q^{n\times\overline{n}}$ in (1), the data parallelism of the matrix $\mathbf{A}$ in U cores is proposed. This procedure is

[1]The implementation source code is available in `https://github.com/mv-amorim/frodokem`.

illustrated in Figure 1. The main strategy is to divide the matrix $\mathbf{A}$ into U portions with n/U rows and n columns. The u-th portion of $\mathbf{A}$ is called $\mathbf{A}_u$, where $u \in \{1, 2, \ldots, U\}$. By multiplying the u-th slice of $\mathbf{A}$ by S, the corresponding u-th slice of $\mathbf{D}$, i.e., $\mathbf{D}_u$, of dimensions $n/U \times \overline{n}$ is obtained. By stacking all the slices $\mathbf{D}_u$, the matrix $\mathbf{D}$ (of size $n \times \overline{n}$) is formed.

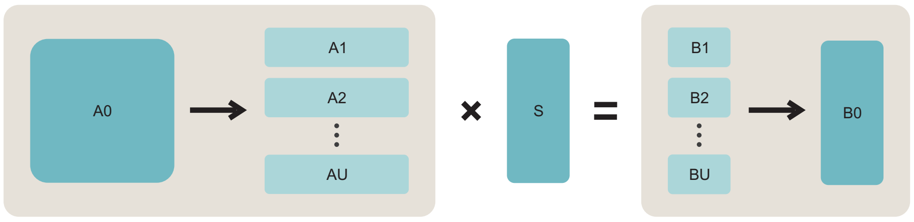

Figure 1. Illustration of the multiplication of A by S using U processing cores.

3.2. S' by $\mathbf{A}$ matrix multiplication

The multiplication of S' by $\mathbf{A}$ slightly differs from the multiplication of $\mathbf{A}$ by S. In the former operation, the matrix $\mathbf{A}$ is placed on the right-hand side of the multiplication. However, the matrix $\mathbf{A}$ presented in (1) must be the same as that presented in (2) and (4), which means that it must be generated in the same way, row by row. On the other hand, as $\mathbf{A}$ is on the right-hand side, a column should be fully generated to perform the multiplication, which is not achieved using the tiled multiplication technique. Consequently, the same procedure reported in Subsection 3.1 cannot be used, requiring another matrix multiplication technique, see [Costa et al. 2022].

Therefore, we use another procedure for calculating the multiplication of S' by $\mathbf{A}$ using U processing cores. This procedure is illustrated in Figure 2. Note that the matrix S' is partitioned into U distinct column portions, where each portion, S'_u, is assigned to the u-th processing core, with $u \in \{1, 2, \ldots, U\}$. Similarly, the matrix $\mathbf{A}$ is divided into U distinct row portions, with each portion, $\mathbf{A}_u$, assigned to the corresponding u-th processing core. Observe that the matrix $\mathbf{A}$ is generated row by row using SHAKE-128 and therefore the u-th processing core only needs to generate the rows associated with its corresponding portion, $\mathbf{A}_u$. Likewise, the u-th processing core only needs the u-th portion of matrix S' (i.e., S'_u). The u-th processing core then computes the matrix multiplication of S'_u by $\mathbf{A}_u$, resulting in the matrix $\mathbf{D}'_u$. Finally, the sum of all matrices $\mathbf{D}'_u$ produces the matrix $\mathbf{D}'$.

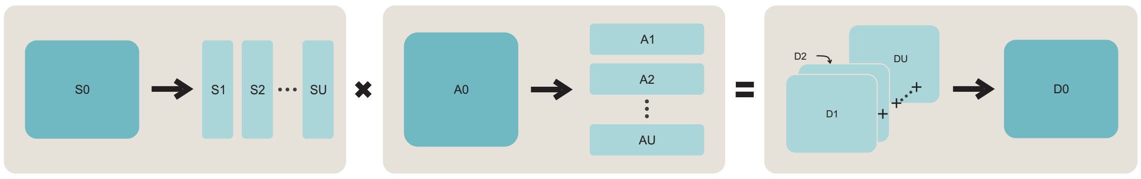

Figure 2. Illustration of the multiplication of S' by A using U processing cores.

3.3. Other matrices multiplication

Encapsulation and decapsulation of FrodoKEM involve additional matrix multiplications, such as the ones presented in (3). Since these operations do not use matrix $\mathbf{A}$, their matrix multiplications can be optimized using a multicore approach based on the strategy presented in Subsection 3.1. The key difference is that $\mathbf{A}$ does not need to be generated and the involved matrices are already pre-generated. Therefore, there is no evident need to use the tiled multiplication technique. Instead, multicore multiplication and loop unrolling can achieve better performance.

4. Practical implementation

To analyze the execution time performance of the proposed multicore implementation of FrodoKEM, a practical implementation was performed using a hardware-constrained microcontroller. In this context, Subsection 4.1 addresses the particularities of the chosen microcontroller, an ESP32-S3. In Subsection 4.2, the feasibility of embedding FrodoKEM in the ESP32-S3 and the techniques required to make it viable are investigated. Finally, Subsection 4.3 discusses details of a dual-core implementation of FrodoKEM.

4.1. Main components

The hardware used in the implementation was an Espressif ESP32-S3 system-on-a-chip (SoC) [Espressif 2023], which is a dual-core Xtensa 32-bits LX7 microcontroller with 512 kB of SRAM memory. The chip also has integrated Wi-Fi 2.4 GHz (802.11 b/g/n) and Bluetooth Low Energy. Software development for the ESP32-S3 uses C programming language and the ESP-IDF framework. Additionally, it includes a modified FreeRTOS[2] implementation for dual-core support, enabling tasks to be pinned to specific core. Furthermore, FreeRTOS resources, such as semaphores, were also used for dual-core implementation. In this work, ESP-IDF version 5.1.4 was used.

4.2. Feasibility analysis

As previously discussed, the size of matrix $\mathbf{A}$ in FrodoKEM is significantly large. This makes it impractical to embed FrodoKEM as it is, even the lower security level variant FrodoKEM-640, in the ESP32-S3 microcontroller due to its limited memory capacity. It occurs because the ESP32-S3 provides 512kB of SRAM, while FrodoKEM-640 requires 819kB for storing matrix $\mathbf{A}$ alone. Therefore, the tiled multiplication technique must be employed. This implementation applied the tiled multiplication technique alongside a four-row loop unrolling. Consequently, each block (tile) has four rows of matrix $\mathbf{A}$. Table 2 lists the memory required to store each block (tile) of matrix $\mathbf{A}$ using the tiled multiplication technique.

Table 2. Size of matrix $\mathbf{A}$ and memory resource usage by matrix $\mathbf{A}$ in reference implementation and implementations with tiled multiplication technique.

	Size of matrix A		**Memory resource usage**	
	Reference	Tiled multiplication	Reference	Tiled multiplication
FrodoKEM-640	819.0 kB	5.12 kB	160%	1%
FrodoKEM-976	1.8 MB	7.80 kB	372%	1.5%
FrodoKEM-1344	3.6 MB	10.75 kB	705%	2.1%

[2]FreeRTOS is a real-time operating system that enables multitasking on limited hardware through preemptive scheduling.

Although the tiled multiplication technique does not enhance execution time performance, it substantially reduces the memory footprint, see Table 2. By applying this technique, the memory resource usage of matrix $\mathbf{A}$ in ESP32-S3 was reduced to around 2.1% in the worst case, enabling operations where matrix $\mathbf{A}$ is required and leaving more memory resources for other tasks.

4.3. Dual-core implementation details

Aiming to prove the performance improvement of the proposed multicore implementation of FrodoKEM, a dual-core implementation is embedded in the ESP32-S3. The proposed implementation aims to take advantage of the dual-core available in this microcontroller (i.e., $U = 2$) based on a modified FreeRTOS by ESP-IDF. When matrix multiplications are required, two tasks are created and pinned to each core. Each function executes half of the operations needed for the matrices using the tiled multiplication technique alongside loop unrolling. A structure is sent to each task containing pointers of data to be processed.

All operations were performed using memory pointers given the RAM limitations of the hardware and the pursuit of better performance. Thus joining all blocks (tiles) of the output matrix was unnecessary since the access to the output variable is shared between the tasks. However, another concern with this strategy was simultaneous access to the same memory location. In cases where the matrix $\mathbf{A}$ is on the left-hand side of the matrix product, it is not necessary to manage access because each task will write only the block (tile) of memory corresponding to the u-th block (tile), i.e., $\mathbf{D}_u$, as indicated in Figure 1. Since each value of u is unique, each processing core will always access different memory locations. On the other hand, if $\mathbf{A}$ is on the right-hand side, then multiple processing cores may access the same memory location of the matrix $\mathbf{D}'$ (or $\mathbf{D}''$) to sum its corresponding portion, i.e., $\mathbf{D}'_u$ (or $\mathbf{D}''_u$). Consequently, it is necessary to implement a mutal exclusion (mutex) lock strategy, which consists of controlling access to the memory blocks corresponding to the rows of the matrix $\mathbf{D}'$ (or $\mathbf{D}''$).

To address the issue in multiplying $\mathbf{S}'$ by $\mathbf{A}$, the authors propose that each processing core starts computing $\mathbf{D}'_u$ from a different row. To implement this, memory access coordination can be effectively managed within the matrix $\mathbf{S}'$, with data acquisition for matrices $\mathbf{S}'_u$ starting at different rows based on the processing core index u. Specifically, the u-th processing core begins acquiring data from $\mathbf{S}'_u$ at row index $(u-1)n/U + 1$. In other words, the first processing core ($u = 1$) starts at the 1-st row of the matrix $\mathbf{S}'_1$, the second processing core ($u = 2$) starts at the $(n/U + 1)$-th row of the matrix $\mathbf{S}'_2$, the third processing core ($u = 3$) starts at the $(2n/U + 1)$-th row of the matrix $\mathbf{S}'_3$, and so on. Note that all processing cores must access all n rows of their respective matrices $\mathbf{S}'_u$, for all $u \in \{1, 2, \ldots, U\}$. Consequently, a cyclic behavior in row scanning is introduced: Upon reaching the last row n of $\mathbf{S}'_u$, each processing core returns to the first row and continues until the row before the one where it started. Figure 3 illustrates this process.

5. Numerical Results

This section evaluates the performance improvement obtained from a dual-core implementation of FrodoKEM on the ESP32-S3 microcontroller. We compare execution times for key generation, encapsulation, and decapsulation across the FrodoKEM-640, -976, and -1344 parameter sets. Each function was executed 101 times, and the median times are reported.

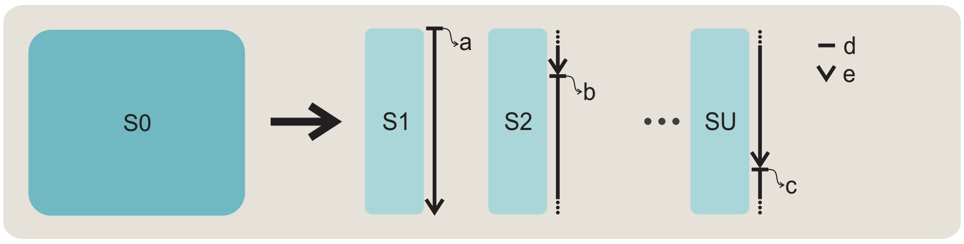

Figure 3. Illustration of the memory access coordination to the matrix S'.

Table 3. Execution time, in seconds, of FrodoKEM algorithms implemented in single- and dual-core of the ESP32-S3 microprocessor.

Description			Frodo-640	Frodo-976	Frodo-1344
Execution time (s)	**Single-core**	Key Pair Generation	2.20	5.10	9.35
		Encapsulation	2.45	5.63	10.46
		Decapsulation	2.45	5.65	10.47
		Total	**7.10**	**16.38**	**30.29**
	Dual-core	Key Pair Generation	1.12	2.59	4.75
		Encapsulation	1.40	2.91	5.75
		Decapsulation	1.41	2.91	5.77
		Total	**3.94**	**8.42**	**16.28**
Performance Improvement (%)		Key Pair Generation	48.8	49.1	49.2
		Encapsulation	42.8	48.4	45.0
		Decapsulation	42.3	48.4	44.9
		Total	**44.5**	**48.6**	**46.3**

As shown in Table 3, execution time increases with the matrix size, which is expected given that matrix $\mathbf{A}$ operations represent the main computational bottleneck. The dual-core implementation consistently outperformed the single-core version, achieving more than 44% reduction in execution time across all parameter sets and functions.

Ideally, data parallelism across two cores should yield a 50% speedup. In practice, the ESP32-S3 microcontroller incurs overhead from memory access and system-level tasks, preventing full parallel efficiency. Still, the results demonstrate that distributing matrix operations across cores is highly beneficial. For instance, FrodoKEM-1344 dropped from 30.29s to 16.28s, while FrodoKEM-640 was reduced from 7.10s to 3.94s.

These execution times show that FrodoKEM, even in its most computationally demanding version, can be deployed on resource-constrained devices when operating under relaxed latency constraints. This is particularly relevant for IoT applications such as smart metering and smart water systems, where data is typically exchanged in intervals of minutes. For example, smart meters report consumption every 15 minutes, and water management systems may transmit parameters every few minutes.

Moreover, reducing cryptographic processing time is critical in battery-powered IoT devices. Faster key establishment shortens the active processing period, allowing devices to return to low-power sleep modes more quickly, thereby improving energy efficiency and extending operational lifespan.

6. Conclusions

This work demonstrated the viability of deploying FrodoKEM on hardware-constrained microcontrollers by combining tiled multiplication and data parallelism. Tiled multiplication enabled execution within memory limits, while data parallelism significantly reduced execution time. A dual-core implementation on the ESP32-S3 achieved a 44.5% speedup over the single-core version, with execution times ranging from 3.94 to 16.28 seconds. These results confirm the feasibility of using FrodoKEM in near real-time IoT environments requiring post-quantum security.

References

[Alkim et al. 2020] Alkim, E. et al. (2020). FrodoKEM: Learning with errors key encapsulation. Technical report, NIST, Gaithersburg, MD.

[Bos et al. 2023] Bos, J. W. et al. (2023). Enabling frodokem on embedded devices. *Cryptology ePrint Archive, Paper 2023/158.*

[Bos et al. 2021] Bos, J. W., Ofner, M., Renes, J., Schneider, T., and van Vredendaal, C. (2021). The matrix reloaded: Multiplication strategies in frodokem. In Conti, M., Stevens, M., and Krenn, S., editors, *Cryptology and Network Security*, pages 72–91, Cham. Springer International Publishing.

[BSI 2020] BSI, C. M. (2020). Recommendations and key lengths. Technical report, BSI.

[Costa et al. 2022] Costa, V. L. R. d., López, J., and Ribeiro, M. V. (2022). A system-on-a-chip implementation of a post-quantum cryptography scheme for smart meter data communications. *Sensors*, 22(19).

[Espressif 2023] Espressif (2023). ESP32-S3 series datasheet v1.8. Technical report, Espressif Systems.

[Karl et al. 2022] Karl, P., Fritzmann, T., and Sigl, G. (2022). Hardware accelerated frodokem on risc-v. In *2022 25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 154–159.

[Kwon et al. 2021] Kwon, H., Jang, K., Kim, H., Kim, H., Sim, M., Eum, S., Lee, W.-K., and Seo, H. (2021). Armed frodo. In Kim, H., editor, *Information Security Applications*, pages 206–217, Cham. Springer International Publishing.

[Lee et al. 2022] Lee, W.-K., Seo, H. J., Seo, S. C., and Hwang, S. O. (2022). Efficient implementation of aes-ctr and aes-ecb on gpus with applications for high-speed frodokem and exhaustive key search. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6):2962–2966.

[Regev 2010] Regev, O. (2010). The learning with errors problem (invited survey). In *Proc. IEEE 25th Annual Conference on Computational Complexity*, pages 191–204.

[Shor 1994] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. In *Proc. 35th Annual Symposium on Foundations of Computer Science*, pages 124–134.