Lattice-based Multisignature Optimization for RAM Constrained Devices

Sara Ricci
ricci@vut.cz
Brno University of Technology
Brno, Czech Republic

Vladyslav Shapoval
236349@vutbr.cz
Brno University of Technology
Brno, Czech Republic

Petr Dzurenda
dzurenda@vut.cz
Brno University of Technology
Brno, Czech Republic

Peter Roenne
peter.roenne@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Jan Oupicky
jan.oupicky@uni.lu
University of Luxembourg
Esch-sur-Alzette, Luxembourg

Lukas Malina
malina@vut.cz
Brno University of Technology
Brno, Czech Republic

ABSTRACT

In the era of growing threats posed by the development of quantum computers, ensuring the security of electronic services has become fundamental. The ongoing standardization process led by the National Institute of Standards and Technology (NIST) emphasizes the necessity for quantum-resistant security measures. However, the implementation of Post-Quantum Cryptographic (PQC) schemes, including advanced schemes such as threshold signatures, faces challenges due to their large key sizes and high computational complexity, particularly on constrained devices. This paper introduces two microcontroller-tailored optimization approaches, focusing on enhancing the DS2 threshold signature scheme. These optimizations aim to reduce memory consumption while maintaining security strength, specifically enabling the implementation of DS2 on microcontrollers with only 192 KB of RAM. Experimental results and security analysis demonstrate the efficacy and practicality of our solution, facilitating the deployment of DS2 threshold signatures on resource-constrained microcontrollers.

CCS CONCEPTS

• **Theory of computation → Cryptographic protocols**; **Cryptographic primitives**; • **Security and privacy → Digital signatures**; **Privacy-preserving protocols**.

KEYWORDS

Lattice-based cryptography, Dilithium, threshold signature, microcontroller, memory optimization, random access memory, RAM

ACM Reference Format:
Sara Ricci, Vladyslav Shapoval, Petr Dzurenda, Peter Roenne, Jan Oupicky, and Lukas Malina. 2024. Lattice-based Multisignature Optimization for RAM Constrained Devices. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30-August 2, 2024, Vienna, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3664476.3670461

1 INTRODUCTION

In a (n, t)-threshold signature scheme, n parties can jointly generate a single public key from their n private shares of the secret key. This secret key can be used to securely sign messages if and only if t parties collaborate in the signing process. Moreover, no group of $t - 1$ colluding parties should be able to recover the secret key. Though they have been studied for a long time [16, 23], these signatures received renewed interest in recent years due to their practical applicability to Blockchain technology and electronic transactions, including cryptocurrencies such as Bitcoin [7]. For instance, in blockchain networks, threshold signatures are used for multi-signature wallets. Specifically, multiple parties have to collaborate to sign a transaction, enhancing security and reducing the risk of single points of failure. Notably, in 2023, National Institute of Standards and Technology (NIST) has published a draft for their First Call for Multi-Party Threshold Schemes [10], which reflects the growing interest in these schemes.

The emergence of quantum computing poses a growing threat to cryptographic security, necessitating a shift towards Quantum-Resistant (QR) signatures. This is also a call for long-term security, as the development of a quantum computer could enable attackers to retroactively compromise previous digital signatures. Therefore, passing to QR schemes is needed not only to preserve the integrity of future communications but also to protect the authenticity of past digital signatures from potential attacks. Addressing this concern, Damgaard, Orlandi, Takahashi, and Tibouchi [13] proposed a two-round n-out-of-n signature scheme, namely DS2, with low round complexity derived from the Fiat–Shamir with Aborts (FSwA) paradigm [27]. DS2 signature is a distributed variant of the Dilithium-G signature [19], with its security proof based on the hardness of Module Short Integer Solution (MSIS) and Module Learning with Errors (MLWE) problems. Notably, Dilithium is one of the signatures selected for standardization by NIST in 2022[1], underscoring its suitability for post-quantum security challenges.

However, QR schemes come at a price, i.e., keys/signatures are significantly bigger and they can be computationally demanding. This makes QR deployment on constrained devices prohibitive without suitable cryptographic co-processors, secure outsourcing computations or code optimization methods. Currently, several techniques have already been suggested to overcome memory and

[1]https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

computation issues, such as key-reduction techniques in [19] (for Dilithium) and hybrid implementation involving the constrained device only for the secure computation [38] (for Classic McEliece).

The first practical implementation of the DS2 algorithm was developed by Dobias *et al.* [18], where the algorithm was slightly modified to rum with Dilithium scheme instead of Dkiluthium-G signature. The scheme was executed on a Linux environment hosted on a Raspberry Pi. For the benchmarking, the Dilithium-G weak parameters [20] were used, that is setting the matrix dimension to $k = \ell = 2$. Note that this setting is not in line with the current security recommendation according to the Federal Information Processing Standards (FIPS) 204 [32]. Finally, the number of signers were fixed to 10. Notably, this implementation necessitates a constant allocation of at least 176,256 bytes of Static Random Access Memory (SRAM) to store data related to parties and resultant signatures. Furthermore, it requires a minimum of 445,744 bytes of both heap and stack allocated memory for temporary parameters and data buffers. Accordingly, a microcontroller with a minimum of 640 KB of SRAM is essential to execute this DS2 implementation effectively. Moreover, this implementation relies on the transmission of substantial data volume over the network, amounting to hundreds of kilobytes. Such extensive data transfer could potentially introduce bottlenecks, particularly when utilized in conjunction with low-powered Wireless Personal Area Network (WPAN) standards such as 802.15.4. It is worth noting that only a limited selection of microcontrollers available on the market currently meet these memory and networking requirements.

1.1 Contribution

Seeking to bridge the knowledge gaps, we focus on the development, implementation, and practical deployment of a DS2 signature scheme with minimum RAM memory requirements. It will allow us to run DS2 even on memory-limited microcontrollers widely used in industrial and Internet of Things (IoT) ecosystems. Specifically, we seek to answer the following Research Questions (RQ):

(RQ1) How can threshold signature schemes be optimized for practical deployment on resource-constrained microcontrollers while maintaining security and efficiency?

(RQ2) What are the most effective optimization methods for reducing memory complexity and computational overhead in threshold signature schemes?

The first implementation of DS2 scheme was introduced in [18]. However, a decrease of the key size, and therefore, of the security strength was needed to let the signature run on ARM Cortex-M3 and ESP32. This paper seeks to build on the conclusions of previous work by:

(1) Introducing an optimization method enabling the implementation of a DS2 scheme with (2,2)-threshold on the chip STM32WB55RGV6 with 192 KB bytes of RAM.
(2) Introducing an optimization method based on secure outsourcing of computation outside the microcontroller to reduce memory complexity by 90%. This optimization allows to extend the signature from 2 to an unbounded number of signers.
(3) Providing several experimental results showing the efficiency of our solution on off-the-shelf microcontrollers. Note that both optimizations are presented for parameters achieving NIST signature security level 2, however, they do not depend on the signature security level and can be easily generalized.

While Points (1) and (2) answer to (RQ1), Point (3) evaluates the answer to (RQ2). It is important to mention that our implementation is open-source and available on the public GitLab repository[2].

The rest of this article is organized as follows. Section 2 extensively reviews the state-of-the-art for QR schemes optimizations on constrained devices. Section 3 presents the basic structure of both proposed DS2 optimizations, and lists the selected parameters and the implementation practical aspects. Section 4 provides the security analysis of the scheme. Both Sections 3 and 4 reflect (RQ1). Section 5 reports the experimental results and reflects (RQ2). Section 6 sums up some open problems and potential extensions of the DS2 signature. The final section contains the conclusions.

2 RELATED WORK

Memory-constrained implementations of post-quantum threshold signatures. To the best of our knowledge, Dobias *et al.* [18] provide the only memory-optimized implementation of post-quantum threshold signatures. They implemented the DS2 [13] scheme with some additional tweaks that further improve its performance. However, in order to run DS2 on memory-constrained devices, such as the ARM Cortex-M3, they had to use alternative parameters providing inadequate security ($<$ 128 bits).

Furthermore, there is only few implementations of other post-quantum threshold signature schemes. For example, Alkadri *et al.* [2] proposed a lattice-based (n, n)-threshold signature scheme designed to be more efficient than DS2 [13] in the case where the number of participants is small. They provide a proof-of-concept implementation in C++ and evaluate their scheme on a MacBook Air M1. Del Pino *et al.* [15] provided a lattice-based (n, t)-threshold signature scheme called Raccoon that "supports a threshold size as large as 1024 signers". The scheme relies on the same hardness assumptions as Dilithium [19]. They developed a high-performance implementation of the scheme and benchmarked it on a desktop system with an Intel i7 CPU. Thus, their implementation does not focus on memory-constrained devices. Laud *et al.* [26] proposed a lattice-based $(2, 2)$-threshold signature scheme called DiLizium 2.0, which can be seen as a version of DS2 [13] using Dilithium [19] (as opposed to Dilithium-G [20] used in DS2). They implemented the scheme in Java and provide some brief benchmark results on a desktop system with an AMD Ryzen 5 CPU and 16 GB of RAM. Their implementation can be seen as a proof-of-concept and does not focus on memory optimizations. Note that the aforementioned implementations [2, 15, 26] do not appear to be public as the papers either do not provide links to repositories or, if they do, they are expired.

Constrained implementations of post-quantum algorithms. From the beginning of the NIST Post-Quantum Cryptography (PQC) standardization process, NIST encouraged submitters to consider the performance of their algorithms on resource-constrained devices [30]. Consequently, there has been a lot of research evaluating and implementing post-quantum KEMs and post-quantum signatures on such devices. In particular, the pqm4 [24] library provides

[2]https://gitlab.com/brno-axe/pqc/ds2ram

implementations of post-quantum algorithms that are optimized for the ARM Cortex-M4. Specifically, their optimizations involve custom assembly code. Note that pqm4 does not contain implementations of all post-quantum algorithms in the NIST PQC standardization process [30]. From KEMs, it supports BIKE [3] and Kyber [8] and from signatures, it supports Dilithium [19] and Falcon [33].[3] More importantly, it does not focus on advanced primitives such as threshold signatures. Tasopoulos *et al.* [37] integrated the pqm4 library into WolfSSL, a TLS library, in order to evaluate the performance of post-quantum TLS embedded systems.

In [9] Botros *et al.* provided a memory-efficient implementation of a lattice-based KEM Kyber [8] for ARM Cortex-M4. Their improvements come from optimizing the number-theoretic transform (NTT), which provides in-place polynomial multiplication resulting in less stack space usage, and specific tweaks to Kyber's subroutines. Similarly, Sanal *et al.* [35] produced an speed-optimized implementation of Kyber for 64-bit ARM Cortex-A. Their improvements come from using the SIMD (vector) instructions available on a 64-bit ARM Cortex-A. Saber [14], an another lattice-based KEM, has also been the scope of optimized implementations for resource-constrained systems: an ARM implementation by Karmakar *et al.* [25] and an implementation for ESP32 by Wang *et al.* [39].

Classic McEliece [28] is a code-based KEM with significantly larger public keys compared to the rest post-quantum algorithms; its smallest public key is 261,120 bytes whereas Kyber's smallest public key is 800 bytes. As a result, there have been a memory-optimized implementations of Classic McEliece which employed novel techniques to handle the large public keys. Urian *et al.* [38] propose to offload the non-critial key generation steps on an untrusted device with larger memory. Alternatively, Roth *et al.* [34] devise a method how to stream the public keys into RAM from the larger flash memory which was further improved by Chen *et al.* [12].

Gonzalez *et al.* [21] focused on implementations of post-quantum signatures on resource-constrained devices. Specifically, they implemented signature verification for Dilithium [19], Falcon [33], SPHINCS+ [4], GeMSS [11] and Rainbow [17][4] on ARM Cortex-M3 with 8 KB of memory. Niederhagen *et al.* [29] extend their work to provide also key generation and signing for SPHINCS+.

In [22] Greconici *et al.* present implementations of Dilitihum optimized for ARM Cortex-M3 and ARM Cortex-M4 in which they propose several strategies how to balance between memory consumption and speed during signing.

3 PROPOSED ARCHITECTURE

Building upon the implementation by Dobias *et al.* [18], the DS2 signature scheme, taking into account the chosen parameters $k = \ell = 2$ and the number of parties $n = 10$, requires a constant allocation of at least 176,256 bytes of SRAM to store data related to parties and resultant signatures. Additionally, it necessitates a minimum of 445,744 bytes of both heap and stack allocated memory for temporary parameters and data buffers. Consequently, a microcontroller would require a minimum of 640 KB of SRAM to effectively execute this DS2 implementation. Moreover, this implementation relies on the transmission of substantial data volumes over the network, amounting to hundreds of kilobytes. Such extensive data transfer could potentially introduce bottlenecks, particularly when utilized in conjunction with low-powered WPAN standards such as 802.15.4. It is worth noting that only a limited selection of microcontrollers available on the market currently meet these memory and networking requirements.

[3]pqm4 also contains implementations of a few PQ signatures, such as MAYO [6] or PERK [1], from the additional PQ signature standardization process by NIST [31].
[4]GeMSS and Rainbow have since been broken [5, 36].

Table 1: Choice of parameters for DS2 scheme.

n	α	σ	B	M
2	22	298694	27034695	1.7272
5	55	746735	67586737	1.2440
10	110	1493470	135173474	1.1153
20	220	2986940	270346948	1.0561
50	550	7467350	675867370	1.0221

In this article, we present two levels of optimizations. At first, we explore memory and communication optimizations aimed at enabling the implementation of the DS2 scheme to run on a microcontroller. Note that without these optimizations, the signature would not be executable. However, this implementation restricts the number of signers to 2. We refer to Section 3.2 for more details. Hence, we propose securely outsource calculations outside the microcontroller to reduce memory complexity. This optimization allows us to extend the signature from 2 to an unlimited number of signers. We refer to Section 3.3 for more details. The security requirement and the chosen parameters are specified in Section 3.1. Our implementation is open-source and available on the public GitLab repository[5].

3.1 Chosen parameters

The recommended parameters for the level 2 security strength from the NIST specification[1] were used since Damgaard *et al.* [13] does not provide the instantiation of the parameters. Specifically, $N = 256, q = 8380417, (k, \ell) = (4, 4), \eta = 5, \kappa = 60, d = 12$. Note that the Dobias *et al.* implementation [18] selected parameters recommended for the security strength level 1. Additional parameters specific to the DS2 scheme were chosen to satisfy the required bounds specified in [13] and to reduce the complexity of communication and the size of the resulting signature. For different number of parties n and targeting the expected total number of rejections during the signing phase to be $M = 3$, the selected parameters

[5]https://gitlab.com/brno-axe/pqc/ds2ram
[1]https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022

Algorithm 1 Mult$(n, m, h, (Gen_A, s_A), (Gen_B, s_B))$

```
1: For $k \in [0, n-1]$ do
2:     For $j \in [0, h-1]$ do
3:         $B[k][j] := Gen_B(s_B, j, k)$
4:         For $i \in [0, m-1]$ do
5:             $A[i][k] := Gen_A(s_A, i, k)$
6:             $Res[i][j] += A[i][k] \cdot B[k][j]$
7: Return $Res$
```

Algorithm 2 $\mathtt{KeyGen}(1^\kappa)$

```
1: $\rho_n, \rho_{s_n} \leftarrow \{0,1\}^{256} := \mathtt{SHAKE\text{-}256}(1^\kappa)$
2: Send out $\rho_n$
3: Receive $\rho_i; i \in [n-1]$
4: $\rho := \mathtt{SHAKE\text{-}256}(\rho_1, \rho_2, ..., \rho_n)$
5: $\mathbf{t}_n := \mathtt{Mult}(l, k, 1, (\mathtt{SHAKE\text{-}256}, \rho), (\mathtt{SHAKE\text{-}256}, \rho_{s_n})) + \bar{\mathbf{s}}_n; \bar{\mathbf{s}}_n :=$ last $k$ bits of $\mathtt{SHAKE\text{-}256}(\rho_{s_n})$
6: $(\mathbf{t}_{n_1}, \mathbf{t}_{n_0}) := \mathtt{Power2Round}(\mathbf{t}_n, d)$
7: $g_n := \mathtt{SHAKE\text{-}256}(\mathbf{t}_{n_1})$
8: Send out $g_n$
9: Receive $g_i; i \in [n-1]$
10: Send out $\mathbf{t}_{n_1}$
11: Receive $\mathbf{t}_{i_1}$ and check $g_i = \mathtt{SHAKE\text{-}256}(\mathbf{t}_{i_1}); i \in [n-1]$
12: $\mathbf{t}_1 := \sum_{i \in [n]} \mathbf{t}_{i_1}$
13: $tr \in \{0,1\}^{512} := \mathtt{SHAKE\text{-}256}(\rho, \mathbf{t}_1)$
14: return $pk = (\rho, \mathbf{t}_1, tr)$, $sk_n = (\rho_{s_n}, \mathbf{t}_{n_0})$
```

are presented in Table 1. In particular, M states for "the expected number of restarts until a single party can proceed", B for "the maximum L^2-norm of signature share $\mathbf{z}_j$ for $j = 1, \ldots, n$", σ for the "standard deviation of the Gaussian distribution", and α for a "parameter defining σ and M". Note that a new commitment needs to be sent any time a rejection appears, therefore, for each rejection the transmitted data increases by $d_{com} = 5888B$ for the standard DS2 scheme. Moreover, the total transmitted data d_{sig} increases linearly with the number of rejections. Finally, the parameters for the lattice-based commitment scheme (i.e., Algorithm 3) were chosen to satisfy the specified bounds. Specifically, the following parameters were chosen: $\bar{s} = 2059, s = 815, B = 48414, l = 23, w = 23$. We refer to [13], Section 5.2 for more details.

3.2 Compression Optimization

This section introduces memory and communication optimizations aimed at enabling the implementation of the DS2 scheme to run on a microcontroller. It is noteworthy that without these optimizations, the signature would not be executable. Specifically, we consider the following optimizations:

O1 **Dynamic Value Computation**. If a value can be reconstructed from the ones we already have access to, we compute it only when needed and then discard it. This approach helps save memory, as we do not store or transmit values that can be reconstructed from known ones.

O2 **Matrix generation and multiplication.** Matrices and vectors are generated one block at a time, where each block is represented by a single polynomial. Note that polynomials are the entries of matrices and vectors, and any calculations can be split to manage "only" two polynomials at once. Therefore, we limit the system to generate the blocks needed at

Algorithm 3 $\mathtt{Commit}(w_n, \rho_{r_n}, \rho_{ck})$

```
1: $com_n := \mathtt{Mult}(68, k, k, (\mathtt{SHAKE\text{-}256}, \rho_{r_n}), (\mathtt{SHAKE\text{-}256}, \rho_{ck}))$
2: $com_n += w_n$
3: return $com_n$
```

Algorithm 4 $\mathtt{Sign}(pk, sk_n, \mu \in M)$

```
1: $\rho_{ck} \in \{0,1\}^{512} := \mathtt{SHAKE\text{-}256}(\mu, tr)$    ▷ SE/CE
2: $\rho_{r_n}, \rho_{y_n} \leftarrow \{0,1\}^{512} := \mathtt{SHAKE\text{-}256}(1^\kappa)$    ▷ SE
3: $\mathbf{w}_n := \mathtt{Mult}(l, k, 1, (\mathtt{SHAKE\text{-}256}, \rho), (\mathtt{SHAKE\text{-}256}, \rho_{y_n})) + \bar{\mathbf{y}}_n;$ $\bar{\mathbf{y}}_n :=$ last $k$ bits of $\mathtt{SHAKE\text{-}256}\rho_{y_n})$    ▷ SE
4: $com_n := \mathtt{Commit}(\mathbf{w}_n, \rho_{r_n}, \rho_{ck});$    ▷ SE
5: Send out $com_n$    ▷ SE
6: Receive $com_i; i \in [n-1]$    ▷ CE
7: $com := \sum_{i \in [n]} com_i$    ▷ CE
8: $\rho_c \in \{0,1\}^{512} := \mathtt{SHAKE\text{-}256}(com, \mu, tr);$    ▷ CE
9: Send out $\rho_c$    ▷ CE
10: Receive $\rho_c$    ▷ SE
11: $c := \mathtt{SHAKE\text{-}256}(\rho_c)$    ▷ SE/CE
12: $\mathbf{z}_n := \begin{pmatrix} \mathbf{z}_{n_1} \\ \mathbf{z}_{n_2} \end{pmatrix} := c\mathbf{s}_n + \mathbf{y}_n;$ $\mathbf{s}_n := \mathtt{SHAKE\text{-}256}(\rho_{s_n})$    ▷ SE
13: Rejection sampling on $(c\mathbf{s}_n, \mathbf{z}_n)$, with probability $min(1, D_s^{l+k}(\mathbf{z}_n)/(M \cdot D_s^{l+k}(\mathbf{z}_{c\mathbf{s}_n,n})))$ continue, otherwise go to Line 3    ▷ SE
14: $\mathbf{z}_{n_2} = \mathbf{z}_{n_2} - c\mathbf{t}_{n_0}$    ▷ SE
15: Send out $(\mathbf{z}_n, \rho_{r_n})$    ▷ SE
16: Receive $(\mathbf{z}_i, \rho_{r_i}); i \in [n-1]$    ▷ CE
17: For $i \in [n-1]$ do
18:     $\mathbf{w}_i := \bar{\mathbf{A}}\mathbf{z}_i - c\mathbf{t}_{i_1} \cdot 2^d$    ▷ CE
19:     If $||z_i||_2 > B$ or $\mathtt{Open}(\rho_{ck}, com_i, \rho_{r_i}, \mathbf{w}_i) \neq 1$ then abort    ▷ CE
20:     $\mathbf{z} := \sum_{i \in [n]} \mathbf{z}_i$    ▷ CE
21: return $\Sigma = (\rho_c, \mathbf{z}, \rho_{r_1}, \rho_{r_2}, ..., \rho_{r_n})$
```

Algorithm 5 $\mathtt{Open}(\rho_{ck}, com_i, \rho_{r_n}, \mathbf{w}_n)$

```
1: $com_n := \mathtt{Mult}(68, k, k, (\mathtt{SHAKE\text{-}256}, \rho_{r_n}), (\mathtt{SHAKE\text{-}256}, \rho_{ck}))$
2: $com_n += w_n$
3: If $com_i == com_n$ return 1
4: Else return 0
```

any given moment. Accordingly, the multiplication of a matrix with a vector becomes a block-by-block product of two polynomials at a time. However, this make the computation more time consuming since the same block need to be generated multiple times. Let $\mathbf{A}$ and $\mathbf{B}$ be $m \times n$ and $n \times h$ matrices, respectively. Algorithm 1 shows the computation of the product $\mathbf{Res} = \mathbf{A} \cdot \mathbf{B}$, where Gen_A and Gen_B are the algorithms for generating each matrix with respective seeds s_a and s_b. Note that in case of a matrix-vector multiplication h will be equal to 1 in Algorithm 1.

O3 **Transfer through seed.** Randomly generated matrices and vectors can be sent through the seeds used to generate them.

The aforementioned optimizations are applied as follows:

- O2-**Algorithm 2, Line 5 and Algorithm 4, Line 3**: Both $\mathbf{t_n}$ and $\mathbf{w_n}$ are computed multiplying matrix $\bar{\mathbf{A}} \leftarrow R_q^{k \times \ell + k}$ with a randomly generated secret vector, respectively $\mathbf{s_n} := [\hat{\mathbf{s}}_n | \bar{\mathbf{s}}_n] \leftarrow S_\eta^{\ell+k}$ and $\mathbf{y}_n := [\hat{\mathbf{y}}_n | \bar{\mathbf{y}}_n] \leftarrow D_s^{\ell+k}$. Note that $\bar{\mathbf{A}} := [\mathbf{A}|\mathbf{I}]$, where $\mathbf{A} \leftarrow R_q^{k \times \ell}$ and $\underline{\mathbf{I}}$ is the $k \times k$ identity matrix. For instance for $\mathbf{t_n}$, Algorithm 1 generates both $\mathbf{A}$ and the first ℓ elements of $\mathbf{s_n}$ block-by-block and multiply

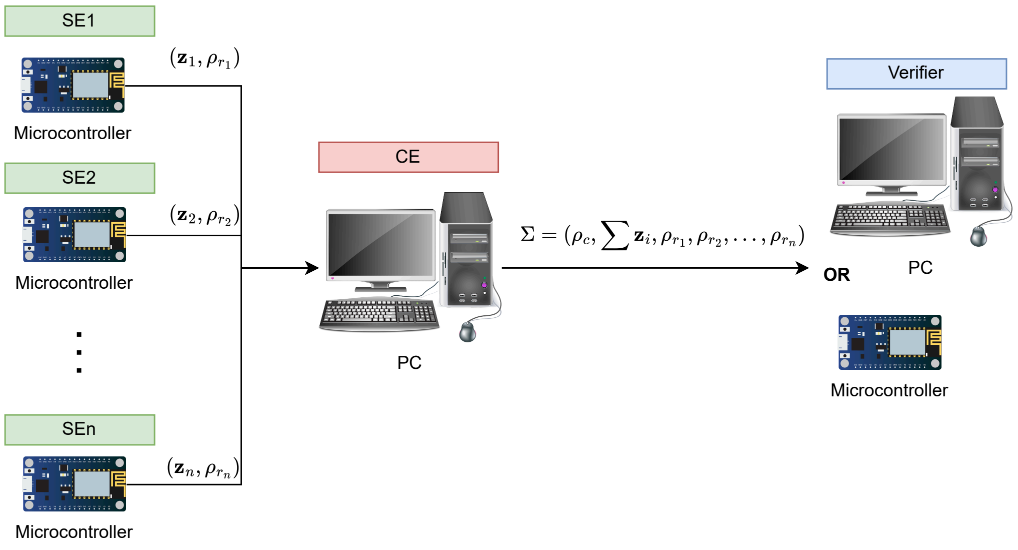

SE1
$(\mathbf{z}_1, \rho_{r_1})$
Microcontroller
SE2
$(\mathbf{z}_2, \rho_{r_2})$
Microcontroller
SEn
$(\mathbf{z}_n, \rho_{r_n})$
Microcontroller
CE
PC
$\Sigma = (\rho_c, \sum \mathbf{z}_i, \rho_{r_1}, \rho_{r_2}, \dots, \rho_{r_n})$
Verifier
PC
OR
Microcontroller

Figure 1: Outsource optimization: Architecture.

them. This approach allows to represent $\mathbf{A}$, $\mathbf{s}_n$ and $\mathbf{y}_n$ in constant memory size of 3 KB (1 KB each), independent of k and ℓ.

- O2-**Algorithm 4, Lines 1 and 2**: The commitment key ck and the random parameter r_n in the original DS2 scheme occupy at least 138 KB each. Therefore, we represent these parameters by their respective seed values ρ_{r_n} and ρ_{ck}, which are 64 bytes each. The ρ_{ck} is the short hash derived from the message and the parameter tr.
- O2-**Algorithm 3 and Algorithm 4, Line 4**: The Commitment key ck and the random parameter $\mathbf{r}_n$ are generated from the respective seeds one block at a time. The computation of the commitment is also done block by block. This approach reduces the buffer sizes for both ck and $\mathbf{r}_n$ from 138 KB to 1 KB each. Note that ck is generated uniformly and $\mathbf{r}_n$ is generated normally and, therefore, their generation takes different amount of time.
- O2-**Algorithm 4, Line 19 and Algorithm 5**: The Open algorithm uses the same optimization function as the Commit and follows the same logic of block-by-block computation from the given seeds.

Algorithm 6 `Verify`$(pk, \Sigma, \mu \in M)$

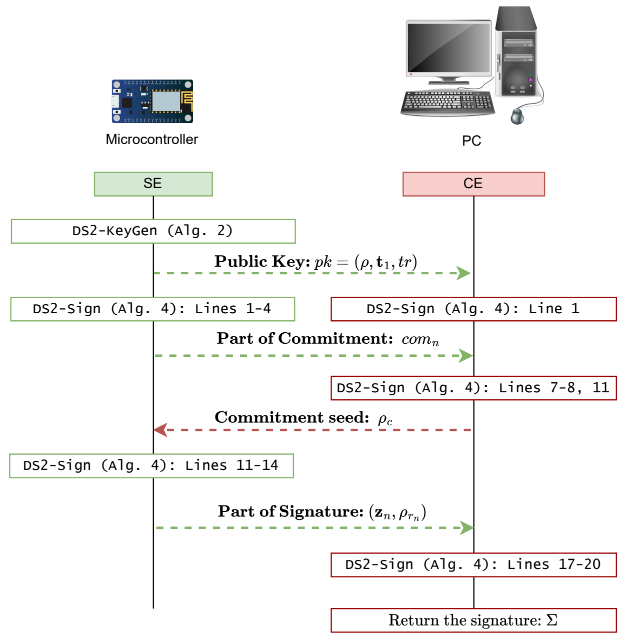
```
1: $c$ := SHAKE-256$(\rho_c)$
2: For $i \in [n-1]$ do
3:     $\mathbf{r}'_i$ := SHAKE-256$(\rho_{r_i}) \longleftarrow D(S_r, \rho_{r_i})$
4:     $\mathbf{r}' := \sum_{i \in [n]} \mathbf{r}'_i$
5: $ck$ := SHAKE-256$(\mu, pk)$
6: $\mathbf{w}' := \bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d$
7: $c'$ := SHAKE-256(Commit$_{ck'}(\mathbf{w}', \mathbf{r}'), \mu,$ SHAKE-256$(\rho, \mathbf{t}_1)$)
8: If $||\mathbf{z}||_2 \le \sqrt{n}B$ and $c = c'$ return 1
9: Else return 0
```

- O2-**Algorithm 4, Lines 8-11**: The challenge polynomial is generated from the seed ρ_c, which is 64 bytes long. The seed is generated from the calculated commitment *com* in the form of a matrix of polynomials. Instead of sending the whole matrix to the SE node, the seed is sent in a single packet. Accordingly, the SE node can regenerate the challenge polynomial c from the seed.
- O3-**Algorithm 4 - Lines 15-16**: the algorithm sends the seeds ρ_{r_i} instead of the whole matrix r_i, which allows eliminating a bottle neck caused by the slow communication speed (on average it takes 5 s to send 138 KB matrix at 250 Kbps).
- O1-**Algorithm 4, Lines 20 and 21**: The signature contains all received seeds ρ_{r_i} and ρ_c, instead of the resulting parameter r. This allows for a reduction in the signature size from 143 KB to 5 KB. Therefore, the following computation is omitted $\mathbf{r} := \sum_{i \in [n]} \mathbf{r}_i$.

3.3 Outsource Optimization

In this section, we introduce another optimization technique aimed to further reduce memory footprint and removing the limit of maximal number of participating parties. Our system consists of two parts: a security-critical part and a non-security-critical part. The security-critical part, namely the Secure Element (SE), performs operations requiring secret knowledge and must be executed on the microcontroller itself. Conversely, the non-critical part, referred to as the Central Element (CE), contains operations that do not require secret knowledge and can be performed either on another microcontroller or on a host computer connected to the CE.

The original DS2 scheme, proposed by Damgaard *et al.* [13] and implemented by Dobias *et al.* [18], employs a full-mesh topology for communication among parties. While robust, this architecture mandates each party to store additional information about all other parties and the random values generated by them. To mitigate this



Microcontroller
PC
SE
CE
DS2-KeyGen (Alg. 2)
Public Key: $pk = (\rho, \mathbf{t}_1, tr)$
DS2-Sign (Alg. 4): Lines 1-4
DS2-Sign (Alg. 4): Line 1
Part of Commitment: com_n
DS2-Sign (Alg. 4): Lines 7-8, 11
Commitment seed: ρ_c
DS2-Sign (Alg. 4): Lines 11-14
Part of Signature: $(\mathbf{z}_n, \rho_{r_n})$
DS2-Sign (Alg. 4): Lines 17-20
Return the signature: Σ

Figure 2: Outsource optimization: communication flow between SE and CE.

issue, the CE aggregates data from all parties and partially offloads the nodes, relieving them of the burden to perform non-security-critical complex calculations. Consequently, we effectively change the network topology from "full-mesh" to "star" configuration.

Figure 1 depicts the system architecture with the addition of a CE. Each secure element SE$_i$ computes part of the signature with the help of the CE. Then CE generates the final signature Σ from the partial information. Note that the Verifier is not part of the signing process and can be represented by either a PC or microcontroller. Accordingly, the compression optimizations (see Section 3.2) are only necessary in the latter case.

Figure 2 sketches the communication flow between a SE and the CE. It is worth noting that Algorithm 2 for key generation remains based on "full-mesh" communication. In fact, the introduction of the CE would not release the SE of the memory-demanding computations. Specifically, in Algorithm 2, Line 5 requires knowledge of the secret value s_{ρ_n}, thereby requiring execution within the SE. Moreover, the introduction of a CE and, consequently, a "star" network, may introduce security concerns, as discussed in Section 4. On the contrary, in Algorithm 4, the SE is relieved of matrix-matrix multiplications, specifically Lines 3 and 19, that are outsourced to the CE. Finally, Algorithm 6 is solely involved in the verification of the signature.

4 SECURITY ANALYSIS

In this section, we prove that our optimized DS2 scheme is complete and secure. To do so, we prove that the `Verify` algorithm is sound and correct, and therefore, only valid signatures generated by authorized parties will be always verified correctly, and invalid signatures will always fail verification. Then, we prove that `KeyGen` and `Sign` algorithms are secure and do not leak any information about the individual parties' secret keys.

Theorem 4.1 (verification Soundness and Correctness). *The verification process in Algorithm 6 (*`Verify`*) is correct and sound.*

Proof. For the signature to be accepted, we need to show that the newly computed c' is equal to c, that is $\texttt{SHAKE-256}(\rho_c) = \texttt{SHAKE-256}(com, \mu, tr)$ is equal to $\texttt{SHAKE-256}(\texttt{Commit}_{ck'}(\mathbf{w}', \mathbf{r}'), \mu, \texttt{SHAKE-256}(\rho, \mathbf{t}_1))$. If this equality holds, it means that the signatures $\mathbf{z_n}$ were generated by the authorized parties knowing the shares s_n of the common secret key. Since tr is equal to $\texttt{SHAKE-256}(\rho, \mathbf{t}_1))$, this can be done by proving that com is equal to $\texttt{Commit}_{ck'}(\mathbf{w}', \mathbf{r}')$. Note that

$$\mathbf{r}' = \sum_{i\in[n]} \mathbf{r}'_i = \sum_{i\in[n]} \texttt{SHAKE-256}(\rho_{r_i}) = \sum_{i\in[n]} \mathbf{r_i} = \mathbf{r}$$

Therefore, the commitments equality can be proven as follows:

$$\begin{aligned}\texttt{Commit}_{ck'}(\mathbf{w}', \mathbf{r}) &= \texttt{Commit}_{ck'}(\bar{\mathbf{A}}\mathbf{z} - c\mathbf{t}_1 \cdot 2^d, \mathbf{r}) \\ &= \texttt{Commit}_{ck'}\Big(\sum_{i\in[n]} \bar{\mathbf{A}}\mathbf{z_i} - c\mathbf{t}_{i1} \cdot 2^d, \sum_{i\in[n]} \mathbf{r}_i\Big) \\ &= \sum_{i\in[n]} \texttt{Commit}_{ck'}(\bar{\mathbf{A}}z_i - c\mathbf{t}_{i1} \cdot 2^d, \mathbf{r_i}) \\ &= \sum_{i\in[n]} \texttt{Commit}(\mathbf{w_i}, \mathbf{r_i}) = \sum_{i\in[n]} com_i = com\end{aligned}$$

Note that it is straightforward that $||\mathbf{z}||_2 \leq \sqrt{n}B$ since the signers only output a signature shares that respect $||\mathbf{z}_i||_2 \leq B$. We refer to [13], Lemma 2 for more details. □

Theorem 4.2 (Key Generation Security). *The key generation process in Algorithm 2 (*`KeyGen`*) is secure.*

Proof. The Algorithm 2 does not leak any information about the individual parties' secret keys as we do not modify the communication model of the original DS2 scheme. In fact, we kept a "full mesh" communication to avoid any possible misbehaving of malicious CE. In the algorithm, ρ represents a committed random value agreed between involved parties. Therefore, outsourcing its computation to a CE could cause a security risk. For example, a malicious CE would be able to pilot the computation of matrix $\bar{\mathbf{A}}$ by sending its own ρ' instead of a fairly computed value $\rho = \texttt{SHAKE-256}(\rho_1, \rho_2, \cdots, \rho_n)$ to each SE. □

Theorem 4.3 (Signing Security). *The signing process in Algorithm 4 (*`Sign`*) is secure.*

Proof. It is worth noting that a malicious CE can break the correctness of the verification process in such a way that the signature generated by authorized parties will fail verification. It can be done easily by spoofing the transmitted data between signers. However, the Algorithm 4 does not leak any information about the individual parties' secret keys to any misbehaving CE. In the algorithm, the signer uses the secret key s_n to compute Proof of Knowledge (PK): $z_n = \begin{pmatrix} z_{n_1} \\ z_{n_2} \end{pmatrix} = cs_n + \mathbf{y}_n$, where $c = \texttt{SHAKE-256}(\rho_c)$, and ρ_c is the challenge computed by CE. This PK does not reveal any information about $\mathbf{s}_n$ even if the CE would have complete control over the challenge c, which it does not have since the output of the hash function SHAKE-256(ρ_c) is unpredictable by CE. □

5 IMPLEMENTATION RESULTS

To evaluate our implementation, we use the STM32WB55 microcontroller with an embedded low-power radio which is compliant with IEEE 802.15.4-2011 standard. An important limitation point is that the STM32WB55 has only 256 KB SRAM1 available. Furthermore, the SRAM1 consists of 192 KB of actual RAM memory and 64 KB of hardware memory parity check. It is also worth mentioning that the STM32WB55 provides access to the NIST-compliant true random number generator[6] that brings more security to the implementation. Besides constrained microcontrollers (MCU) deployed on the signers' devices in form of SEs, we use also a computationally more powerful device in the form of a Personal Computer (PC) representing the CE. Table 2 shows more details about the hardware specification of deployed devices.

Table 2: Hardware specification of deployed devices.

Device	MCU / SE	PC / CE
Chip	Cortex-M4	Core i5-11400H
Number of cores	2 (M4 & M0+)	12
Frequency	64 MHz	2.7 - 4.5 GHz
RAM memory	192 KB	16 GB
Wireless standard	802.15.4	803.11 Wi-Fi

Below, we provide more details about our DS2 implementation. Namely, Section 5.1 presents the implementation of our transportation layer protocol to the 802.15.4 MAC frame payload to ensure a smooth data exchange between SE nodes at the application level. Section 5.2 presents the implementation of our memory-optimisation techniques and their evaluation. Finally, Section 5.3 presents benchmark results of our implementation.

5.1 Communication protocol

For communication between nodes, a Medium Access Control (MAC) layer of the 802.15.4 standard is used. Therefore, in order to correctly identify data and its source in the network, we created a simplified transportation layer protocol. The maximum 802.15.4 MAC frame payload length varies between 72 and 116 bytes, depending on the length of the MAC frame header. We reserve 8 bytes for the packet header and, from empirical experience, we limited data length to 100 bytes. Each SE node is given its own ID from 0 to 253. Node IDs 254 and 255 are reserved for CE and broadcast, respectively. The structure of the packet is shown in Figure 3.

1B	1B	1B	1B	4B (optional)	max 100B (optional)
packet_len	dst_node_id	src_node_id	msg_code	data_offset	data

Figure 3: Packet structure.

5.2 Memory consumption analysis

The memory footprint of the original DS2 implementation [18] consists of several parts: the static memory region (M_{static}), which stores data for the public key (M_{pk}) of the party, secret key (M_{sk}), and the resulting signature (M_{sign}), as well as the stack-allocated memory (M_{stack}) and heap-allocated memory (M_{heap}), which are used for the storage of temporary values during computation. Let k, ℓ, T_c be the dimensions of the matrices, N the number of coefficients in each polynomial, and n the number of parties involved. Therefore, the memory complexity for static parameters without signature can be described by Equation 1, for the signature by Equation 2, for the stack by Equation 3, and for the heap by Equation 4.

$$M_{static} = 4N(k(\ell + (n+2)) + \ell + (n+2)k^2) \quad (1)$$

$$M_{sign} = 4N(k + \ell + kT_c) + s \quad (2)$$

$$M_{stack} = 4N((k+1)T_c + 5\ell + 7k + 1) + 3N(2k + \ell + k^2) \quad (3)$$

$$M_{heap} = (k+1)(3N \cdot T_c) \quad (4)$$

If we take into account that the parameters $N = 256$ and $T_c = 69$ are constants, the total memory footprint of the DS2 implementation developed by Dobias *et al.* [18] depends on the chosen security level and the number of parties. Additionally, the largest portion of the memory will be occupied by the commitment key ck, the pseudo random matrix r (which is part of the resulting signature), and additional memory buffers to store the shares of the matrix r, generated by each party.

Application of our first optimization technique allows us to represent A, s_1, s_2, ck, and r by a single polynomial each, along with random seeds ρ ρ_{s_i}, ρ_{ck} and ρ_{r_i}, where $i \in [1 \cdots n]$. Each seed has a length of s bytes, thus eliminating the dependency of Equations 2, 3, and 4 on the parameter T_c. This, in turn, permits us to completely remove M_{heap} part and reduce the size of resulting signature M_{sign}.

Anyway, each node must store $n-1$ received commitments to be able to check them at the final stage of signature generation and calculate their sum. As stated before, we address this issue by changing the topology of the local network through the introduction of the central node, namely CE, which is represented by the PC. We split our system into a security-critical part and a non-security-critical part. The security-critical part must always be calculated by the SE nodes, but SE no longer have to store commitments from other nodes. The final note is that all coefficients in each polynomial are taken modulus 8380417, which is a 23-bit number. To further reduce memory consumption and network traffic, vectors and matrices, which are meant to be sent wirelessly, namely t_1, z_1, z_2, and the com, can be "packed" to take only 3 bytes instead of 4 for each coefficient. After applying both optimizations, we obtain Equations 5 and 6.

$$M_{static} = 4N(\ell + 2k) + 3N(2k + \ell + k^2) + 4s \quad (5)$$

$$M_{stack} = 4N(k^2 + 4k + 2\ell + 6) \quad (6)$$

[6]STMICROELECTRONICS, 2022. Datasheet - STM32WB55xx STM32WB35xx. STMicroelectronics. Available at: https://www.st.com/en/microcontrollers-microprocessors/stm32wb55rg.html

Table 3: Memory consumption of different DS2 variants.

Parameter set	Original DS2	This Work	Mem. Reduction
$k = 4, \ell = 4$	1039 KB	80 KB	92.4%
$k = 6, \ell = 5$	1542 KB	134 KB	91.4%
$k = 8, \ell = 7$	2093 KB	206 KB	90.2%

Based on the equations above, we estimated the amount of memory required to run DS2: a) without our improvements, b) only with vector multiplication optimization (i.e., Compression Optimization, see Section 3.2), c) with multiplication optimization and outsourced computations on the CE node (i.e., Outsource Optimization, see Section 3.3). These estimations were calculated for the values $N = 256$, $T_c = 69$, $s = 64$ and for (k, ℓ) dimensions, which correspond to those stated in the ML-DSA standard, namely (4,4), (6,5), (8,7). The introduction of compression optimization reduces memory consumption by an average of 80%. Outsourcing optimization allows for an additional 10% reduction from the original. The total memory reduction after the application of all optimizations is, on average, 90%. The results of our calculations (considering applying both optimisation methods) are shown in Table 3.

Additionally, we have to reserve on average 20 KB of static memory space for the main application. Taking into account all of the above, we can confidently say that with all our changes, we are able to run DS2(4,4) and DS2(6,5) variants on the STM32WB55RGV6.

5.3 Time complexity analysis

First of all, we have to address two main limitations we encountered during our experiments. First, due to the memory limitations of the microcontroller, we could not conduct empirical experiments with DS2(8,7) variant. Second, we chose the number of CPU cycles as the metric for performance measurements in order to obtain frequency independent values. However, the register-counter for the number of CPU cycles in Cortex-M4 is limited to 32 bits. If k is chosen to be larger than 4, this counter overflows multiple times during large matrix multiplication. Thus, we should rely only on estimations.

The key difference between the original signature algorithm and our modified version lies in the commitment function, see Algorithm 4. The commitment is represented by the multiplication of two large matrices, shown in Algorithm 1. We decided to analyze its impact on the overall performance. For each measurement, we do 10 experiments and find an average of the results. The average total execution time of DS2(4,4) for all operations conducted by the SE node is 4.4E+9 CPU cycles. At the same time, the average execution time of the matrix multiplication in DS2(4,4) is 4.2E+9 CPU cycles, which is 95% of the total execution time. It follows from this that the analysis of Algorithm 1 will provide us with good approximation about the dependence of the execution time of the entire DS2(k, ℓ) implementation on the selected parameters (k, ℓ).

To compare original version (the "Standard") and our optimized version (the "memory-optimized") of the matrix multiplication, we approximated their execution time as shown by Equations 7 and 8.

$$C_{fast}(k) = O(T_c k(k t_{mul} + t_{ck} + t_r)) \tag{7}$$

$$C_{slow}(k) = O(T_c(k(k(t_{ck} + t_{mul}) + t_r))) \tag{8}$$

Table 4: Comparison between theoretical execution time of the slow and fast algorithms of matrix multiplication.

k	C_{fast}	C_{fast} [s]	C_{slow}	C_{slow} [s]	C_{slow}/C_{fast}
2	7.54E+8	11.79	1.12E+9	17.52	1.49
3	1.14E+9	17.81	2.24E+9	35.00	1.96
4	1.53E+9	23.93	3.73E+9	58.29	2.44
5	1.93E+9	30.13	5.59E+9	87.40	2.90
6	2.33E+9	36.41	7.83E+9	122.32	3.36

Table 5: Comparison between theoretical and real execution time of the slow algorithm of matrix multiplication.

k	C_{real}	C_{real} [s]	C_{teor}	C_{teor} [s]	C_{real}/C_{teor}
2	1.22E+9	19.13	1.12E+9	17.52	1.09
3	2.48E+9	38.74	2.24E+9	35.00	1.11
4	4.18E+9	65.28	3.73E+9	58.29	1.12
5	6.27E+9	98.01	5.59E+9	87.40	1.12
6	8.54E+9	133.41	7.83E+9	122.32	1.09

Where:

- C_{fast}, C_{slow} are functions returning the time needed to multiply two matrices with dimensions $k \times T_c$ in CPU cycles.
- $T_c = 69$ is a matrix dimension constant.
- k is a matrix dimension variable.
- t_{ck} is time needed to generate a single polynomial of the commitment key ck from the seed in CPU cycles.
- t_r is time needed to generate a single polynomial of the random matrix r from the seed in CPU cycles.
- t_{mul} is time in CPU cycles needed to multiply two polynomials with N coefficients.

Parameters t_{ck}, t_r and t_{mul} are linearly dependent on the number of coefficients in the polynomial N. Through all test $N = 256$ remained constant. Therefore, t_{ck}, t_r and t_{mul} should also be constants and after our measurements, we obtained on average $t_{ck} =$ 2.6E+6, $t_r =$ 2.7E+6, $t_{mul} =$ 4E+4 CPU cycles. If we substitute those values into Equations 7 and 8, we can plot those functions on the k axis. As can be seen from the values in Table 4, the modified matrix multiplication algorithm for the DS2(4,4) variant is 2.44 times slower than the original and is 3.36 times slower for the DS2(6,5) variant.

Finally, we compared our estimations from Table 4 with real measurements, obtained from the actual SE device for the memory-optimized version of multiplication. However, as was stated before, we were able to run only the DS2(6,5) version at maximum. Thus, we have no measurements for k greater than 6. The differences between our estimations and real-time values are shown in Table 5 and Figure 4. It can be seen that the actual implementation is 1.1 times slower than its theoretical model, but this was expected beforehand.

6 DISCUSSION

Several key points emerge regarding the performance of our optimized matrix multiplication on the Cortex-M4 microcontroller

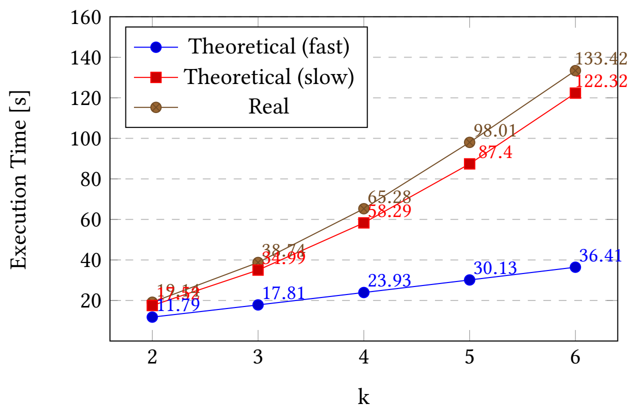


Figure 4: Comparison between theoretical and real execution time of the algorithm of matrix multiplication.

compared to the original implementation. Firstly, we identify unoptimized hash and floating-point calculations as the primary factors contributing to the observed slowdown, resulting in our optimized multiplication being four times slower. To address this performance bottleneck, hardware support for hash functions or Advanced Encryption Standard (AES) encryption presents a promising solution. For instance, utilizing Davies-Meyer hash with hardware AES acceleration could potentially achieve a significant speedup, estimated to be at least five times faster than the current software implementation of SHAKE. However, modifications to the Davies-Meyer scheme would be necessary to accommodate hashes of variable length.

Moreover, optimizing floating-point calculations on microcontrollers emerges as another critical consideration. Notably, Dobias' DS2 [18] implementation heavily relies on floating-point calculations to obtain normally and uniformly sampled values. Unfortunately, floating-point operations exhibit significant slowness on the Cortex-M4, even when supported by a hardware Floating Point Unit (FPU) unit. As a result, the generation of a single polynomial incurs a substantial performance overhead, being ten times slower than the Number Theoretic Tranform (NTT) transformation and point-wise multiplication of two polynomials combined.

7 CONCLUSIONS

In this article, we addressed the challenge of deploying the DS2 threshold signature scheme on memory-constrained microcontrollers. Our aim was to optimize the scheme for practical implementation while maintaining security and efficiency standards. We introduced two optimization methods tailored for microcontrollers with minimal RAM requirements. Firstly, we applied techniques enabling the implementation of a DS2 scheme with a (2,2)-threshold on the STM32WB55RGV6 chip, requiring only 192 KB bytes of RAM. Additionally, we proposed a method based on secure outsourcing of computations outside the microcontroller, effectively reducing memory complexity by 90% and facilitating the extension of the signature to an unlimited number of signers. Our optimization methods highlight the importance of efficient memory utilization for the practical deployment of threshold signature schemes on resource-constrained microcontrollers. Notably, both optimizations were presented for a signature security strength of 2 but can be readily generalized to accommodate other security levels.

In our future work, we intend to investigate the integration of a hardware cryptographic accelerator to enhance hashing and floating-point calculation performance. Additionally, we plan to explore the application of secret sharing techniques to extend DS2 scheme from n-out-of-n to t-out-of-n threshold signature.

ACKNOWLEDGMENTS

The following funding source is gratefully acknowledged: Ministry of the Interior of the Czech Republic under Grant VJ03030014. Jan Oupický was supported by the industrial partnership project between the interdisciplinary research center SnT and LuxTrust. Peter Roenne received support from the Luxembourg National Research Fund (FNR) under the CORE project (C21/IS/16221219/ ImPAKT).

REFERENCES

[1] Najwa Aaraj, Slim Bettaieb, Loïc Bidoux, Alessandro Budroni, Victor Dyseryn, Andre Esser, Phillipe Gaborit, Mukul Kulkarni, Victor Mateu, Marco Palumbi, Lucas Perin, and Jean-Pierre Tillich. 2024. PERK - Submission to Round 1 of the Additional Signatures NIST Post-Quantum Project. https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures
[2] Nabil Alkeilani Alkadri, Nico Döttling, and Sihang Pu. 2024. Practical Lattice-Based Distributed Signatures for a Small Number of Signers. In *Applied Cryptography and Network Security*, Christina Pöpper and Lejla Batina (Eds.). Springer Nature Switzerland, Cham, 376–402. https://doi.org/10.1007/978-3-031-54770-6_15
[3] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zemor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. 2022. BIKE – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
[4] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. 2019. The SPHINCS+ Signature Framework. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, London United Kingdom, 2129–2146. https://doi.org/10.1145/3319535.3363229
[5] Ward Beullens. 2022. Breaking Rainbow Takes a Weekend on a Laptop. In *Advances in Cryptology – CRYPTO 2022 (Lecture Notes in Computer Science)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer Nature Switzerland, Cham, 464–479. https://doi.org/10.1007/978-3-031-15979-4_16
[6] Ward Beullens. 2022. MAYO: Practical Post-quantum Signatures from Oil-and-Vinegar Maps. In *Selected Areas in Cryptography*, Riham AlTawy and Andreas Hülsing (Eds.). Springer International Publishing, Cham, 355–376. https://doi.org/10.1007/978-3-030-99277-4_17
[7] BitcoinCore. 2017. Technology roadmap - Schnorr signatures and signature aggregation. https://bitcoincore.org/en/2017/03/23/schnorr-signature-aggregation/
[8] Joppe Bos, Leo Ducas, Eike Kiltz, T Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehle. 2018. CRYSTALS - Kyber: A CCA-Secure Module-Lattice-Based KEM. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, London, 353–367. https://doi.org/10.1109/EuroSP.2018.00032
[9] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. 2019. Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4. In *Progress in Cryptology – AFRICACRYPT 2019*, Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi (Eds.). Springer International Publishing, Cham, 209–228. https://doi.org/10.1007/978-3-030-23696-0_11
[10] Luıs Brandao and Rene Peralta. 2023. NIST First Call for Multi-Party Threshold Schemes. https://doi.org/10.6028/NIST.IR.8214C.ipd.
[11] A. Casanova, J.-C. Faugère, G. Macario-Rat, J. Patarin, L. Perret, and J. Ryckeghem. 2020. GeMSS – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
[12] Ming-Shing Chen and Tung Chou. 2021. Classic McEliece on the ARM Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (July 2021), 125–148. https://doi.org/10.46586/tches.v2021.i3.125-148

[13] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. 2021. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In *IACR International Conference on Public-Key Cryptography*. Springer, 99–130.

[14] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren. 2018. Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM. In *Progress in Cryptology – AFRICACRYPT 2018*, Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi (Eds.). Springer International Publishing, Cham, 282–305. https://link.springer.com/chapter/10.1007/978-3-319-89339-6_16

[15] Rafael del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani Saarinen. 2024. Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions. https://eprint.iacr.org/2024/184

[16] Yvo Desmedt and Yair Frankel. 1989. Threshold cryptosystems. In *Conference on the Theory and Application of Cryptology*. Springer, 307–315.

[17] Jintai Ding and Dieter Schmidt. 2005. Rainbow, a New Multivariable Polynomial Signature Scheme. In *Applied Cryptography and Network Security (Lecture Notes in Computer Science)*, John Ioannidis, Angelos Keromytis, and Moti Yung (Eds.). Springer, Berlin, Heidelberg, 164–175. https://doi.org/10.1007/11496137_12

[18] Patrik Dobias, Sara Ricci, Petr Dzurenda, Lukas Malina, and Nikita Snetkov. 2023. Lattice-Based Threshold Signature Implementation for Constrained Devices. (2023).

[19] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.

[20] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. CRYSTALS – Dilithium: Digital Signatures from Module Lattices. https://repository.ubn.ru.nl/bitstream/handle/2066/191703/191703.pdf

[21] Ruben Gonzalez, Andreas Hülsing, Matthias J. Kannwischer, Juliane Krämer, Tanja Lange, Marc Stöttinger, Elisabeth Waitz, Thom Wiggers, and Bo-Yin Yang. 2021. Verifying Post-Quantum Signatures in 8 kB of RAM. In *Post-Quantum Cryptography*, Jung Hee Cheon and Jean-Pierre Tillich (Eds.). Springer International Publishing, Cham, 215–233. https://doi.org/10.1007/978-3-030-81293-5_12

[22] Denisa O. C. Greconici, Matthias J. Kannwischer, and Amber Sprenkels. 2021. Compact Dilithium Implementations on Cortex-M3 and Cortex-M4. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2021), 1–24. https://doi.org/10.46586/tches.v2021.i1.1-24

[23] Kazuharu Itakura. 1983. A public-key cryptosystem suitable for digital multisignatures. *NEC J. Res. Dev.* 71 (1983).

[24] Matthias J. Kannwischer, Richard Petri, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2024. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[25] Angshuman Karmakar, Jose Maria Bermudo Mera, Sujoy Sinha Roy, and Ingrid Verbauwhede. 2018. Saber on ARM: CCA-secure Module Lattice-Based Key Encapsulation on ARM. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Aug. 2018), 243–266. https://doi.org/10.13154/tches.v2018.i3.243-266

[26] Peeter Laud, Nikita Snetkov, and Jelizaveta Vakarjuk. 2022. DiLizium 2.0: Revisiting Two-Party Crystals-Dilithium. https://eprint.iacr.org/2022/644

[27] Vadim Lyubashevsky. 2009. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 598–616.

[28] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Info von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. 2022. Classic McEliece – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[29] Ruben Niederhagen, Johannes Roth, and Julian Wälde. 2022. Streaming SPHINCS+ for Embedded Devices Using the Example of TPMs. In *Progress in Cryptology - AFRICACRYPT 2022*, Lejla Batina and Joan Daemen (Eds.). Springer Nature Switzerland, Cham, 269–291. https://doi.org/10.1007/978-3-031-17433-9_12

[30] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.rip/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf

[31] NIST. 2022. Call for Additional Digital Signature Schemes for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf

[32] NIST. 2023. FIPS 204 (Initial Public Draft), Module-Lattice-Based Digital Signature Standard. https://doi.org/10.6028/NIST.FIPS.204.ipd.

[33] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2022. Falcon – Submission to Round 3 of the NIST Post-Quantum Project. https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions

[34] Johannes Roth, Evangelos Karatsiolis, and Juliane Krämer. 2021. Classic McEliece Implementation with Low Memory Footprint. In *Smart Card Research and Advanced Applications*, Pierre-Yvan Liardet and Nele Mentens (Eds.). Springer International Publishing, Cham, 34–49. https://doi.org/10.1007/978-3-030-68487-7_3

[35] Pakize Sanal, Emrah Karagoz, Hwajeong Seo, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. 2021. Kyber on ARM64: Compact Implementations of Kyber on 64-Bit ARM Cortex-A Processors. In *Security and Privacy in Communication Networks*, Joaquin Garcia-Alfaro, Shujun Li, Radha Poovendran, Hervé Debar, and Moti Yung (Eds.). Springer International Publishing, Cham, 424–440. https://doi.org/10.1007/978-3-030-90022-9_23

[36] Chengdong Tao, Albrecht Petzoldt, and Jintai Ding. 2021. Efficient Key Recovery for All HFE Signature Variants. In *Advances in Cryptology – CRYPTO 2021 (Lecture Notes in Computer Science)*, Tal Malkin and Chris Peikert (Eds.). Springer International Publishing, Cham, 70–93. https://doi.org/10.1007/978-3-030-84242-0_4

[37] George Tasopoulos, Jinhui Li, Apostolos P. Fournaris, Raymond K. Zhao, Amin Sakzad, and Ron Steinfeld. 2022. Performance Evaluation of Post-Quantum TLS 1.3 on Resource-Constrained Embedded Systems. In *Information Security Practice and Experience (Lecture Notes in Computer Science)*, Chunhua Su, Dimitris Gritzalis, and Vincenzo Piuri (Eds.). Springer International Publishing, Cham, 432–451. https://doi.org/10.1007/978-3-031-21280-2_24

[38] Rainer Urian and Raphael Schermann. 2022. Classic McEliece Key Generation on RAM constrained devices. Cryptology ePrint Archive, Paper 2022/1613. https://eprint.iacr.org/2022/1613

[39] Bin Wang, Xiaozhuo Gu, and Yingshan Yang. 2020. Saber on ESP32. In *Applied Cryptography and Network Security: 18th International Conference, ACNS 2020, Rome, Italy, October 19–22, 2020, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 421–440. https://doi.org/10.1007/978-3-030-57808-4_21