Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Especialidad en Sistemas Embebidos

Implementing CRYSTAL-Dilithium on FRDM-K64

TRABAJO RECEPCIONAL para obtener el **GRADO** de
ESPECIALISTA EN SISTEMAS EMBEBIDOS

Presenta: **KATYA DENISSE ORTEGA LUNA**

Asesor **LUIS JULIAN DOMINGUEZ PEREZ**

Tlaquepaque, Jalisco. septiembre de 2021.

Implementing CRYSTALS-Dilithium on FRDM-K64

K. Denisse Ortega L.[1], Luis J. Dominguez Perez[1],

[1]*Department of Electronics, Systems and Informatics,*
Western Institute of Technology and Higher Education (ITESO), Tlaquepaque, México
katya.ortega@iteso.mx, luisjdominguezp@iteso.mx

***Abstract*—This document presents the performance of the lattice-based signature scheme Dilithium for FRDM-K64 ARM Cortex-M4. Dilithium is a recent post-quantum cryptographic scheme, such as a candidate in the third round of the NIST. The Cortex-M4 is considered a suitable microcontroller for post-quantum cryptography so that FRDM-K64 is used to execute the Dilithium scheme. The PQM4 library and Greconici, Kannwischer, and Sprenkels implementation are compared and improved to achieve a low speed of 17% in NIST Security Level 2.**

Keywords—Dilithium, Cortex-M4, FRDM-K64, post-quantum, signature scheme

I. Introduction

Before long, quantum computers could breach the security of today's cryptographic systems. For this reason, NIST (National Institute of Standards and Technology) announced the need to create a post-quantum cryptographic standard [1]. The Dilithium lattice-based signature scheme is one of the third-round candidates for digital schemes. This scheme is based on a hard lattice framework called the "Fiat-Shamir with Aborts", so that a compact scheme with security is enabled based on the worst-case hardness. Dilithium can also be implemented in embedded systems due to its small public key and signature size [2].

However, in real-time embedded systems, speed is a limited resource, so Dilithium implementation must be fast. Ravi, Gupta, Chattopadhyay, and Bhasin [3] improved the optimal number of computations and implemented these improvements for Dilithium on Cortex-M4. Similarly, Greconici, Kannwischer, and Sprenkels [4] presented optimized implementations for Dilithium on Cortex-M3 and Cortex-M4, which are optimizations of speed and stack usage.

The contribution of this paper is to improve the speed presented by Greconici, Kannwischer, and Sprenkels by 10% using the Dilithium3 parameter set (third-round parameters) and the existing Dilithium implementation for Cortex-M4.

The sections in this paper are organized as follows: Section 2 is an introduction to the digital signature scheme Dilithium and Cortex-M4. Section 3 shows the improvements in the speed of the scheme on Cortex-M4. Section 4 presents the speed results.

II. Preliminaries

A. Dilithium

Dilithium is a lattice-based signature scheme that resolves two lattice problems in the QROM (Quantum Random Oracle model). The first is the Module-Learning With Errors (M-LWE) problem, which consists of finding a short vector where they are usually nonexistent. On the other hand, the second is the Module-Short Integer Solutions (M-SIS) problem that entails finding a vector with small coefficients in a random lattice [5].

The Dilithium signature scheme is a finalist candidate in the NIST Post-Quantum Competition third round. This scheme is based on the "Fiat-Shamir with Aborts" framework and provides the option to perform a deterministic or random signature scheme. The description below is based on the specification submitted for CRYSTALS-Dilithium in the third round.

Parameters. Dilithium can be configured to obtain a NIST Security Level [5]. This is possible when the parameters are chosen. In the third round, the security levels proposed by Dilithium for NIST are 1--, 1-, 2, 3, 5, 5+, and 5++. However, the 1--, 1-, 5+, and 5++ levels are omitted because they are the lowest or highest Dilithium security levels. Table 1 shows the parameters for levels 2, 3, and 5. Furthermore $|pk|$ and $|sig|$ represent the bytes that the public key and signature used in memory, respectively.

TABLE **I.** DILITHIUM PARAMETERS.

NIST Level	(k,l)	η	β	ω	$\|pk\|$	$\|sig\|$	*Exp. reps*
2	(4,4)	2	78	80	1312	2420	4.25
3	(6,5)	4	196	55	1952	3293	5.1
5	(8,7)	2	120	75	2592	4595	3.85

Key generation. The Dilithium operations are over the ring $Rq = \mathbb{Z}q[X]/(Xn+1)$, with $q = 2^{23}-2^{13}+1$ and $n = 256$. In the key generation, the secret and public keys are generated. A random seed is used to generate ρ, ρ', and K, then the Fast Fourier Transform (also called Number Theory Transform (NTT)) is used to expand the matrix. This operation is important because the operations over NTT allow efficient implementation and reduce operations.

Signing. The signature is the most laborious part of the Dilithium implementation, furthermore, it is the functionality that uses the most time. The SHAKE256 is used to achieve randomness. In this section, the deterministic or random model can be chosen. It depends on the implementation, if the message does not need to be known, the better

option is to use the random model. In the signature, the M-SIS problem is resolved into the while loop, when all checks comply.

Verification. The verification only reviews the signature with the message. For this part, it is only necessary to review the high bits to assure the correct signature.

B. Cortex-M4

The proposed implementations of the Dilithium in microcontrollers are based on Cortex-M4. The Cortex-M4 makes it easy to implement the scheme because the implementation is in C language. The ARM Cortex-M4 family has the following characteristics: powerful, small, and cheap.

NIST declared that the Cortex-M4 can be used for post-quantum schemes [4] as Dilithium. The Cortex-M4 used is the FRDM-K64. This microcontroller provides a 1 MB flash with 256 KB of embedded SRAM and runs a frequency of 120 Mhz. It has communications interfaces like Ethernet and CAN, which enables its use in the automotive industry [6].

III. Optimization on Cortex-M4

The optimizations on Cortex-M4 are based on the Dilithium reference implementation submitted in the third round of the NIST standardization process. Furthermore, this implementation is based on the reference [4] and reference [7] codes.

The reference [4] used the parameters of the two-round while the reference [7] used the parameters and code of the third round.

Table 2 shows the difference in the parameters used in Dilithium. In the third round, Dilithium improved its implementation and the parameters changed without affecting the NIST Security Level.

TABLE II. PARAMETERS USED IN THE REFERENCE IMPLEMENTATIONS.

NIST Level[a]	(k,l)		η		β	
	Ref [4]	*Ref [7]*	*Ref [4]*	*Ref [7]*	*Ref [4]*	*Ref [7]*
2	(5,4)	(4,4)	5	2	325	78
3	(6,5)	(6,5)	3	4	275	196

TABLE III. PARAMETERS USED IN THE REFERENCE IMPLEMENTATIONS (CONTINUATION).

NIST Level[a]	ω		d	
	Ref [4]	*Ref [7]*	*Ref [4]*	*Ref [7]*
2	96	80	14	13
3	120	55	14	13

[a]Only the NIST Security Levels 2 and 3 are not considered because the 5 NIST Level is not implemented in these references.

The advantage of Dilithium is that it can change its parameters but the implementation is not affected. For choosing of NIST Security Level, the correct parameters must be configured.

Table 4 shows speeds obtained in the references. The speed values in reference 7 are higher than in reference 4 because the changes in the third round achieve better security.

TABLE IV. PERFORMANCE PREVIOUS RESULTS ON THE CORTEX-M4 AT 24 MHZ.

NIST Level	KeyGen[a]		Sign[a]		Verify[a]	
	Ref [4]	*Ref [7]*	*Ref [4]*	*Ref [7]*	*Ref [4]*	*Ref [7]*
2	2013	1600	6053	3911	1917	1578
3	2837	2834	6001	7081	2720	2699

[a]The speeds are reported in units of thousand (10^3) clock cycles.

Reference [7] is considered as the base code in this document. Some modules are in assembly language, which allows better performance in the microcontroller and it is based on the Dilithium scheme. The code includes the third round parameters and the implementation of the Dilithium. Furthermore, the operations NTT, NTT^{-1}, and SHAKE256 are optimized with the assembler code.

Reference [7] includes the improvements of reference [4]. Changing the polynomial coefficients to signed representation is the main improvement on Cortex-M4 from the article published by [4]. These improvements were made in the Cooley-Tukey algorithm in the NTT operations and the Gentleman-Sande algorithm in the NTT^{-1} operations. One improvement proposal in this article is the change the "for loop" in the packing, poly, and sign files, for "for loop unrolling" and it achieves reduced overhead.

The implementations were executed on Cortex-M4 but there is no previous information on the implementation of the Dilithium in an FRDM-K64.

IV. Results

The experimentation was conducted on the K64 board, in this case, a novel Ditilium scheme shows the following results:

Cortex-M4. The "hardware random-number generator" is used to obtain the random seeds. In the executions, the microcontroller is configurated at 120 MHz. The MCUXpresso IDE v11.2 is used to compile the code. The algorithm latency was measured using the internal cycle counter (CYCCNT).

Configuration. Dilithium is configurated as deterministic when the reference [4] and reference [7] are executed. It is changed to randomized as an improvement because the microcontroller has a hardware random-number generator that can be used, and its speed is increased.

Code. The implementation of Dilithium, with the proposed improvements and adapted to FRDM-K64, can be downloaded at https://github.com/kattdenisse/DilithiumFRDM-K64.

Tables 5 and 6 present the measurements obtained from the Dilithium implementation. First is the value of the Greconici, Kannwischer, and Sprenkels [4] implementation is shown with the difference that is executed on a different microcontroller at 120Mhz. The second column is the value of the library PQM4 with a different microcontroller at 120Mhz, too. Both implementations were executed on Cortex-M4 (STM32F4) at 24 MHz.

The code implementations are similar, reference [4] is not updated with the last version of Dilithium.

TABLE V. PERFORMANCE RESULTS ON THE FRDM-K64 CORTEX-M4 AT 120 MHZ.

NIST Level	KeyGen[a]			Sign[a]		
	Ref [4]	*Ref [7]*	*Opt*	*Ref [4]*	*Ref [7]*	*Opt*
2	2604	2214	2117	6147	7525	5057
3	3600	3673	3661	7036	8585	7012

TABLE VI. PERFORMANCE RESULTS ON THE FRDM-K64 CORTEX-M4 AT 120 MHZ (CONTINUATION).

NIST Level	Verify[a]		
	Ref [4]	*Ref [7]*	*Opt*
2	2642	2212	2189
3	3649	3597	3598

[a]The speeds are reported in units of thousand (10^3) clock cycles.

Performance values shown in the Tables demonstrate the percentage of improvement to the previous implementations. Each table shows the speed of one algorithm that is part of Dilithium implementation. These comparisons are in clock cycles and the last 3 are in milliseconds.

TABLE VII. COMPARATIVE PERFORMANCE OF THE KEY GENERATION.

NIST Level	KeyGen (x103cycles)					
	Ref [4]	*Opt*	*%*	*Ref [7]*	*Opt*	*%*
2	2604	2117	18.7	2214	2117	4.4
3	3600	3661	-1.69	3672	3661	0.3

[a]The speeds are reported in units of thousand (10^3) clock cycles.

TABLE VIII. COMPARATIVE PERFORMANCE OF THE SIGNATURE.

NIST Level	Sign (x103cycles)					
	Ref [4]	*Opt*	*%*	*Ref [7]*	*Opt*	*%*
2	6147	5057	17.7	7525	5057	32.8
3	7036	7012	0.44	8585	7012	18

[a]The speeds are reported in units of thousand (10^3) clock cycles.

TABLE IX. COMPARATIVE PERFORMANCE OF THE VERIFY.

NIST Level	Verify (x103cycles)					
	Ref [4]	*Opt*	*%*	*Ref [7]*	*Opt*	*%*
2	2642	2189	17.1	2212	2189	0.1
3	3649	3598	1.4	3598	3598	0

[a]The speeds are reported in units of thousand (10^3) clock cycles.

TABLE X. COMPARATIVE PERFORMANCE OF THE KEY GENERATION.

NIST Level	KeyGen (x103ms)					
	Ref [4]	*Opt*	*%*	*Ref [7]*	*Opt*	*%*
2	108.5	17.6	83.7	92.2	17.6	80.9
3	150	30.5	79.7	153	30.5	80.1

[a]The speeds are reported in units of milliseconds (10^3).

TABLE XI. COMPARATIVE PERFORMANCE OF THE SIGNATURE.

NIST Level	Sign (x103ms)					
	Ref [4]	*Opt*	*%*	*Ref [7]*	*Opt*	*%*
2	256.1	42.1	83.5	313.5	42.1	86.6
3	293.1	58.4	80.1	357.7	58.4	83.7

[a]The speeds are reported in units of milliseconds (10^3).

TABLE XII. COMPARATIVE PERFORMANCE OF THE VERIFY.

NIST Level	Verify (x103ms)					
	Ref [4]	*Opt*	*%*	*Ref [7]*	*Opt*	*%*
2	110	18.2	83.4	92.1	18.2	80.2
3	152	29.9	80.3	149.8	29.9	79.9

[a]The speeds are reported in units of milliseconds (10^3).

V. CONCLUSIONS

Dilithium is a recent scheme and it is updated frequently. The scheme improves a lot on Cortex-M4 with the contributions in the PQM4 library.

Dilithium is a scheme easy to implement and configure. Additionally, this work showed that it can be implemented in embedded systems because the time execution is fast. In systems where security is important but is not dangerous the NIST Security Level 2 is the suitable level to be implemented. However, if message security is an important factor, the NIST Security Level 3 is the most suitable because it achieves good security and optimal performance.

On Cortex-M4, Dilithium implementation is possible. However, in embedded systems where resources such as time and memory are limited, implementing it will be difficult and not recommended.

ACKNOWLEDGMENT

This work was supported by the Mexican federal government through the CONACyT agency.

REFERENCES

[1] (2017, Jan 3). Post-Quantum Cryptography. (2nd ed.) [Online] https://csrc.nist.gov/Projects/Post-Quantum-Cryptography

[2] P. Schwabe. (2017). Cryptographic Suite for Algebraic Lattices. [Online] pq-crystals.org

[3] P. Ravi, S.S. Gupta, A. Chattopadhyay, S. Bhasin "Improving Speed of Dilithium's Signing Procedure". *CARDIS* 2019, vol 11833. pp 57-73, Mar. 2020

[4] D. O. C. Greconici, M. J. Kannwischer, and D. Sprenkels, "Compact Dilithium Implementations on Cortex-M3 and Cortex-M4", *TCHES*, vol. 2021, no. 1, pp. 1-24, Dec. 2020.

[5] CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation (Version 3.1)

[6] "Kinetis K64 Sub-Family Data Sheet With 1 MB Flash", NXP Semiconductors, Tech. Data, K64P142M120SF5, 2019

[7] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen (2020, Jul 23). PQM4. (2nd ed.) [Online] https://github.com/mupq/pqm4