2021 IEEE 12th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON) | 978-1-6654-0690-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/UEMCON53757.2021.9666622

Implementing CRYSTAL-Dilithium on FRDM-K64

1st K. Denisse Ortega L.
DESI
ITESO, Universidad Jesuita de Guadalajara
Tlaquepaque, Mexico
katya.ortega@iteso.mx

2nd Luis J. Dominguez Perez
DESI
ITESO, Universidad Jesuita de Guadalajara
Tlaquepaque, Mexico
0000-0002-3275-470X

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.**

Index Terms—**Dilithium, Cortex-M4, FRDM-K64, quantum-safe, signature scheme**

I. Introduction

Cryptography, is the systematic study of the mathematical techniques to offer the security services required for the information storage, transmission, and processing, such services may vary depending on the specific purpose of every piece of information. The three basic services are *confidentiality*, which protects the information from unauthorized eyes; *integrity*, which provides assurance that the provided information is still as intended; and availability, which is more related to the information, and communication technologies to provide continuous access to the information, regardless of the situation. These three basic security services are the pillars of the information security, some of them require cryptography, and proper software engineering or electronic designs to be provided in a quality manner according to the user or client.

Additionally, there are other services such as *authentication*, that let us verify the user or computer system identity; and *non-repudiation*, which guarantee to the recipient from a message that it was created by the original content creator (and that she cannot deny creating it). The latter is a crucial security service since it provides a scenario where people can produce contracts, and other legal documents.

In computing, a digital archive is formed by a bit-string of zeroes, and ones, that can represent alphanumeric values, either text, images, sound, music, etc. In a computer device, any digital file could easily be created, copied, inserted onto other files, modified, and deleted, a mechanism to detect such changes is required. A digital signature is a cryptographic mechanism that let us verify the authenticity, integrity, and no-repudiation of a digital file.

In the foreseen future, it could be possible to construct a large enough quantum computer that could -hypothetically- be able to use the Shorr algorithm [16] in order to break contemporary cryptography in polynomial time, leaving every single piece of encrypted information exposed to unintended eyes, or to undetectable modification by malicious or unintentional parties. As a result of these possibility, the NIST (National Institute of Standards and Technology) of the United States, has announced the need to create a post-quantum cryptographic standard [1] in advance, in order to have plenty of time for widely use of the newly standardized cryptographic primitives, and to deploy it world-wide to any device, including, of course, embedded devices or more restricted artifacts.

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 presents the mathematical background on digital signatures, as well as an introduction to the digital signature scheme Dilithium and a brief introduction to the Cortex-M4 system. Section 3 shows the improvements in the speed of the scheme on Cortex-M4. Section 4 presents the speed results.

II. Preliminaries

In this section we present the mathematics behind the Dilithium scheme.

The authors acknowledge support from CONACyT, Mexico.

Authorized licensed use limited to: National Taiwan Univ of Science and Technology. Downloaded on October 07,2024 at 07:42:09 UTC from IEEE Xplore. Restrictions apply.

A. Digital signatures

The two most important security goals for cryptography are to provide privacy to the peers in a communication protocol, and to provide authentication from one to the other party, and vice versa. These security goals can be achieved with traditional symmetric cryptography, one that uses the same encryption key for both parties; however, symmetric cryptography does not provides a way to share the key in a secure manner between the parties, creating a new problem. To "solve" the problem, one can take the turn around, and provide the credentials in advance, in-person in the device, these however, could be inconvenient when the parties cannot meet before, such is the case of electronic commerce, different geographic locations, and many other situations. [11]

Around the mid 1970s Merkle [10], Diffie and Hellman [9] published the idea of public key cryptography in which the user could create a pair of keys: a public key that can be freely shared without any concern, and a private key that the user has to take special care. These key pairs are mathematically linked in such a way, that information that was processed with public key, can only be processed with the private key, and also, any information that was processed with the private key can only be processed with the public key. This way, any entity having a piece of information that was accessed using the public key of a user, has certainty that was created only by the creator. In addition, a user that treated a piece of information with the public key of the receiver, also has the certainty that only she would have access to the information, or the intended results.

In the case of a digital signature, a signatory can use her private key to provide a digital identifier of a message or document that is linked to her public key, this way, any interested party could use the public key, and verify a document. In addition to this, a signatory can request her ID, and public key to be linked by a certificate authority to provide a standardized digital certificate.

The RSA algorithm scheme is as follows: [12] Let p y q be two different, large, random, prime numbers. Let $n = p \cdot q$, and $\Phi(n) = (p-1) \cdot (q-1)$, choose the public key as a low Hamming Weight number in the range: $1 < e < \Phi(n)$, such that the $\mathrm{GCD}(e, \Phi(n)) = 1$; for example, one would typically choose $e = 2^{16}+1$, the main reason for choosing this particular value is for efficiency purposes, as one would like to use the Square-and-Multiply algorithm, which should provide fast response with low Hamming Values as the algorithm implies more operations for each bit equal to 1. Then, the Private Key is computed as follows: $d \equiv e^{-1} \bmod \Phi(n)$, which is an expensive operation, but it's only computed once during the Key Generation. The public key is (e, n), whereas the private key is (d, p, q), perhaps p, and q are no longer required, and could be removed from the device. An up-to-date website with the key length recommendations from various vendors, and standards organizations can be consulted in [15]

To encrypt a message M, first, we need to encode it, for instance, using a hash function [14] to fit the range $1 \leq M < n$, then compute, and transmit $c = M^e \bmod n$. To decrypt the ciphertext, compute $p' = c^d \bmod n$. Please note, that for the encryption, the exponent is e, the low Hamming value, which means it's easy for a party to encrypt something, whereas, the party interested in recovering the message would need to be raised to a random-looking power with (probably) not a low Hamming weight value, making it a much more expensive operation.

For the signature, there are a few standards, for instance, the RFC 8446 [8] mentions the RSA Probabilistic Signature Scheme [13] as a way to digitally sign a message using the RSA signature scheme in a secure way, the RSA signature can be used in the inverse direction as the RSA encryption since the goal is that anyone having access to the public key can verify the signature, hence, the signature should be computed as $s = H(M)^d \bmod n$, and the verification is done as follows: $(s^e \bmod n) \overset{?}{\equiv} H(M) \bmod n$, this is, to sign the "decrypt" procedure with the private key is applied to the encoded (hashed) message M, and it's appended to the message; the received signed message is "encrypted" with the public key e, and compared to the encoded (hashed) received message, if the encode of the received message equals to the opened signature sent, then, the message preserves it's integrity, and only the owner of the private key could have signed the message.

B. Lattices

A lattice is a subset of the vector space $\mathbb{R}^m$, writing all vectors as rows, it is possible to denote by $||\underline{v}||$ the norm of the vector $\underline{v} \in \mathbb{R}^m$.

Let $\{\underline{b}_1, \ldots, \underline{b}_n\}$ be a linearly independent set of (row) vectors in $\mathbb{R}^m$ with $m \geq n$, the lattice generated by $\{\underline{b}_1, \ldots, \underline{b}_n\}$ is the set $L = \{\sum_{i=1}^{n} l_i \underline{b}_i \in \mathbb{Z}\}$ of integer linear combinations of the $\underline{b}_i$. The vectors $\underline{b}_1, \ldots, \underline{b}_n$ are called the lattice basis. The lattice rank in n, and the lattice dimension is m, if $n = m$, then, the lattice is a full rank lattice. [11]

1) Computational importance.: The problems arising from lattices can be resumed as: [11]

- Lattice membership. Given a matrix, and a vector, determine if the vector is part of the lattice.
- Lattice basis. Given a set of vectors, find a lattice basis for them.
- Kernel lattice. Given a matrix, compute a lattice basis with kernel equal to 0.
- Kernel lattice modulo M. Extend the previous problem for the discrete instance.

Given the above problems, the following are computational problems that are believed to be hard:

- The shortest vector problem (SVP): given a basis matrix B for L, compute a non-zero vector $\underline{v} \in L$ such that $||\underline{v}||$ is minimal ($||\underline{v}|| + \lambda_1$ the minimal length).
- The closest vector problem (CVP): given a basis matrix B for L, and a vector $\underline{w} \in \mathbb{Q}m$ (or $\mathbb{R}^m$), compute $\underline{v} \in L$, such that $||\underline{w} - \underline{v}||$ is minimal.
- The decision closest vector problem (DCVP): given a basis matrix B for a lattice L, a vector $\underline{w} \in \mathbb{Q}^m$, and a real number $r > 0$ decide whether or not there is a vector $\underline{v} \in L$, such that $||\underline{w} - \underline{v}|| \leq r$.

Authorized licensed use limited to: National Taiwan Univ of Science and Technology. Downloaded on October 07,2024 at 07:42:09 UTC from IEEE Xplore. Restrictions apply.

- The decision shortest vector problem (DSVP): given a basis matrix B for a lattice L and a real number $r > 0$ decide whether or not there is a non-zero $\underline{v} \in L$, such that $||v|| \leq r$.

These computational problems (and other omitted lattice-based problems) are known to be hard when the rank is sufficiently large, hence, for cryptographically large rank values, these problems can be used to construct cryptographic schemes. Further more, the Shorr algorithm [16] does not cover the lattice case, hence, can be used to construct quantum-safe cryptographic schemes.

C. 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.

a) 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	$\lvert pk\rvert$	$\lvert sig\rvert$	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	4594	3.85

b) Key generation: The Dilithium operations are over the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$, 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. The KeyGen operations are as follows 1:

The public key is $pk = (A, t)$, and the secret key is $sk = (s_1, s_2)$

Algorithm 1 Key Generation Procedure

```
Input: $(\kappa, \ell, \eta, q)$
Output: pk, sk
  A matrix $A \leftarrow \mathbb{R}_q^{\kappa \times \ell}$
  Random vectors $(s_1, s_2) \leftarrow S_\eta^\ell \times S_\eta^\kappa$
  $t \leftarrow As_1 + s_2$
```

c) 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. The Sign operations are as follows 2:

Algorithm 2 Signing Procedure

```
Input: $(sk, M)$
Output: $\sigma$ signature of $M$
  $z := \perp$
  while $z = \perp$ do
    $y \leftarrow S_{\gamma_1}^\ell - 1$
    $w_1 := \text{HighBits}(Ay, 2\gamma_2)$
    $c \in B\gamma := H(M||w_1)$
    $z := y + cs_1$
    if $||z||_\infty \geq \gamma_1 - \beta$ or $||\text{LowBits}(Ay - cs_2, 2\gamma_2)||_\infty \geq \gamma_2 - \beta$ then
      $z := \perp$
    end if
  end while
  $\sigma = (z, c)$
```

d) 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. The Verify operations are as follows 3:

Algorithm 3 Verification Procedure

```
Input: $pk, M, \sigma = (z, c)$
Output: True or False
  $w_1' := \text{HighBits}(Az - ct, 2\gamma_2)$
  If return$[\![||z||_\infty < \gamma_1 - \beta]\!]$ and $[\![= H(M||w_1')]\!]$
```

D. 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.

The Cortex-M4 microcontroller has the following characteristics: [17]

- Low power. Since the microcontroller is in a small package, the power consumption is below $200\mu A$/MHz, some

Authorized licensed use limited to: National Taiwan Univ of Science and Technology. Downloaded on October 07,2024 at 07:42:09 UTC from IEEE Xplore. Restrictions apply.

variants can be programmed to reach up to $100\mu A$/MHz of power consumption.
- Performance. Can deliver over 1.25 DMIPS/MHz.
- Supports the C programming language, low latency interrupts, supports on board debug, and can be scaled up to 200 MHz for the CPU clock.

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 II 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	(k,l)		η		β	
	[4]	[7]	[4]	[7]	[4]	[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 implementation (continuation)

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

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 IV 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.

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.

TABLE IV
Performance previous results on the Cortex-M4 at 24 MHz

NIST Level	KeyGen (kcc)		Sign (kcc)		Verify (kcc)	
	[4]	[7]	[4]	[7]	[4]	[7]
2	2013	1600	6053	3911	1917	1578
3	2837	2834	6001	7081	2720	2699

TABLE V
Performance results on the FRDM-K64 Cortex-M4 at 120 MHz.

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

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 in [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:

a) 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).

b) 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.

c) 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.

Authorized licensed use limited to: National Taiwan Univ of Science and Technology. Downloaded on October 07,2024 at 07:42:09 UTC from IEEE Xplore. Restrictions apply.

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

NIST Level	Verify (kcc)		
	Ref [4]	Ref [7]	Opt
2	2642	2212	2189
3	3649	3597	3598

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 (CPU CYCLES).

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

TABLE VIII
COMPARATIVE PERFORMANCE OF THE SIGNATURE (CPU CYCLES).

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

TABLE IX
COMPARATIVE PERFORMANCE OF THE VERIFY (CPU CYCLES).

NIST Level	Verify (kcc)					
	Ref [4]	Opt	%	Ref [7]	Opt	%
2	2642	2189	17.1	2212	2189	0.1
3	3649	3598	1.4	3598	598	0

TABLE X
COMPARATIVE PERFORMANCE OF THE KEY GENERATION (TIME).

NIST Level	KeyGen (ms)					
	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

TABLE XI
COMPARATIVE PERFORMANCE OF THE SIGNATURE (TIME).

NIST Level	Sign (ms)					
	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

TABLE XII
COMPARATIVE PERFORMANCE OF THE VERIFY (TIME).

NIST Level	Verify (ms)					
	Ref [4]	Opt	%	Ref [7]	Opt	%
2	110	18.2	83.4	92.1	18.2	80.2
3	152	29.9	80.4	149.8	29.9	79.9

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 other embedded systems where resources such as time and memory are limited, implementing it will be difficult and not recommended.

REFERENCES

[1] ”Post-Quantum Cryptography — CSRC”, Computer Security Resource Center, 2017. [Online]. Available: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography. [Accessed: 28- Sep- 2021].
[2] ”CRYSTALS - Cryptographic Suite for Algebraic Lattices”, Pq-crystals.org, 2017. [Online]. Available: https://pq-crystals.org/. [Accessed: 22- Sep- 2021].
[3] P. Ravi, S. S. Gupta, A. Chattopadhyay, and S. Bhasin, “Improving Speed of Dilithium’s Signing Procedure,” in Smart Card Research and Advanced Applications - 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11-13, 2019, Revised Selected Papers, 2019, vol. 11833, pp. 57–73.
[4] D. Greconici, M. Kannwische, and D. Sprenkels, “Compact Dilithium Implementations on Cortex-M3 and Cortex-M4”, IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2021, no. 1, pp. 1-24, 2020. Available: 10.46586/tches.v2021.i1.1-24.
[5] L. Ducas amd E. Kiltz and T. Lepoint and V. Lyubashevsky and P. Schwabe and G. Seiler and D. Stehlé, “CRYSTALS-Dilithium Algorithm Specifications and Supporting Documentation,” 2020.
[6] NXP Semiconductors, “Kinetis K64 Sub-Family Data Sheet With 1 MB Flash”, NXP Semiconductors, Tech. Available: https://www.nxp.com/docs/en/data-sheet/K64P142M120SF5.pdf. [Accessed: 28- Sep- 2021]
[7] M. Kannwischer, J. Rijnevel, P. Schwabe and K. Stoffelen, ”PQM4: Post-quantum crypto library for the ARM Cortex-M4”, GitHub, 2020. [Online]. Available: https://github.com/mupq/pqm4. [Accessed: 28- Sep-2021].
[8] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC Editor, RFC 8446, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446.txt
[9] W. Diffie and M. Hellman, ”New directions in cryptography”, IEEE Transactions on Information Theory, vol. 22, no. 6, pp. 644-654, 1976.
[10] R. Merkle, ”Secure communications over insecure channels”, Communications of the ACM, vol. 21, no. 4, pp. 294-299, 1978.
[11] S. Galbraith, Mathematics of public key cryptography. Cambridge: Cambridge University Press, 2012.
[12] J. Katz and Y. Lindell, Introduction to modern cryptography, 2nd ed. Boca Raton, FL: CRC Press, 2015.

Authorized licensed use limited to: National Taiwan Univ of Science and Technology. Downloaded on October 07,2024 at 07:42:09 UTC from IEEE Xplore. Restrictions apply.

[13] M. Bellare and P. Rogaway, ”PSS: Provably Secure Encoding Method for Digital Signatures”, IEEE, 1998 [Online]. Available: https://web.archive.org/web/20170810025803/http://grouper.ieee.org/groups/1363/P1363a/contributions/pss-submission.pdf. [Accessed: 24-Nov- 2021]

[14] National Institute of Standards and Technology, ”Secure Hash Standard (SHS)”, NIST, 2015 [Online]. Available: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf. [Accessed: 25- Nov- 2021]

[15] D. Giry, ”Keylength - Cryptographic Key Length Recommendation”, Keylength.com, 2020. [Online]. Available: https://www.keylength.com/. [Accessed: 24- Nov- 2021]

[16] P. W. Shor, ”Algorithms for quantum computation: discrete logarithms and factoring”. Proceedings 35th Annual Symposium on Foundations of Computer Science. IEEE Comput. Soc. Press: 124–134. doi:10.1109/sfcs.1994.365700. ISBN 0818665807.

[17] J. Yiu, The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors, 3rd ed. Cambridge: Elsevier Inc, 2014, pp. 8-10.

Authorized licensed use limited to: National Taiwan Univ of Science and Technology. Downloaded on October 07,2024 at 07:42:09 UTC from IEEE Xplore. Restrictions apply.