Rank-metric Encryption on Arm-Cortex M0

Porting code-based cryptography to lightweight devices

A. al Abdouli
Emanuele Bellini
Florian Caullery
Marcos Manzano
Victor Mateu
DarkMatter L.L.C., Dubai, United Arab Emirates
firstname.surname@darkmatter.ae

ABSTRACT

Since its invention by McEliece in 1978, cryptography based on Error Correcting Codes (ECC) has suffered from the reputation of not being suitable for constrained devices. Indeed, McEliece's scheme and its variants have large public keys and relatively long ciphertexts.

Recent works on these downsides explored the possible use of ECC based on rank metric instead of Hamming metric. These codes were introduced in the late 80's to eliminate errors with repeating patterns, regardless of their Hamming weight. Numerous proposals for the NIST Post-Quantum Cryptography (PQC) competition rely on these codes.

It has been proven that lattice-based cryptography and even hash-based signatures can run on lightweight devices, but the question remains for code-based cryptography.

In this work, we demonstrate that this is actually possible for rank metric: we have implemented the encryption operation of 5 schemes based on ECC in rank metric and made them run on an Arm Cortex-M0 processor, the smallest Arm processor available. We describe the technical difficulties of porting rank-based cryptography to a resource-constrained device while maintaining decent performance and a suitable level of security against side-channel attacks, especially timing attacks.

CCS CONCEPTS

• **Security and privacy → Cryptography**; **Public-key (asymmetric) techniques**; **Public-key encryption**; **Embedded systems security**;

KEYWORDS

Post-quantum cryptography, Code-based cryptography, Rank metric, Lightweight cryptography

ACM Reference format:
A. al Abdouli, Emanuele Bellini, Florian Caullery, Marcos Manzano, and Victor Mateu. 2019. Rank-metric Encryption on Arm-Cortex M0. In *Proceedings of The 6th ACM ASIA Public-Key Cryptography Workshop, Auckland, New Zealand, July 8, 2019 (APKC '19),* 9 pages.
https://doi.org/10.1145/3327958.3329544

APKC '19, July 8, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6784-4/19/07…$15.00
https://doi.org/10.1145/3327958.3329544

1 INTRODUCTION

The Internet of Things (IoT) is regarded to as a large network of physical devices with the ability to communicate with each other. The communication flow in IoT is asymmetric given that one part of the network is focused on capturing or measuring data that is latterly sent to a recipient hub or central server. Over the last decade, IoT has been gaining world-wide attention from a broad number of industries. Some IoT applications support critical infrastructures and strategic services, as well as generate enormous amounts of sensitive data about health or financial status. Therefore, it is of paramount importance for the IoT ecosystem to provide security and protect its end-users privacy [20]. The devices comprising the IoT are computationally constrained and consequently traditional cryptography has to be adapted in order to be run on such conditions.

Substantial advances in quantum computing in the past decade have re-assured researchers about the necessity to build quantum-resistant cryptosystems [8]. The announcement by the National Institute of Standards and Technology (NIST) to define new standards for Public-Key Encryption (PKE), digital signatures and Key-Encapsulation Mechanism (KEM) schemes [27] has augmented the attention of the scientific community towards Post-Quantum Cryptography (PQC) in general, and cryptography based on Error Correction Codes (ECC) in particular, due to the fact that ECC represents the most conservative approach for PKE and KEM.

Cryptography based on ECC traces back to McEliece's proposal in 1978 [23]. At the time, the RSA cryptosystem [33] was preferred over McEliece's for a simple reason: McEliece's public-key and ciphertext were too large to be practical and allow a widespread deployment. Nevertheless, while Shor's quantum algorithm [36] makes RSA's underlying mathematical problem solvable in polynomial time, the best quantum attacks against McEliece are still exponential in the length of the used ECC [19]. On top of it, McEliece benefits from an impressive 40 years long unsuccessful cryptanalysis effort, increasing strongly the confidence in the scheme.

Progress on the aforementioned McEliece's scheme drawbacks have mainly been obtained by replacing the Goppa codes used in the McEliece's original cryptosystem. Nonetheless, such attempts have often been broken by cryptanalysis efforts: see for example

the QC-MDPC scheme [26] and the reaction attack proposed in [16] breaking 80-bits of security instances in minutes.

Another direction of research considered the use of ECC based on *rank metric* instead of the classical Hamming distance. The notion of error correcting codes in rank metric was introduced by Gabidulin in [10] and used for the first time in cryptography by the Gabidulin, Paramonov, Tretjakov (GPT) cryptosystem [11]. Given that the complexity of decoding a random code in the rank metric is higher than decoding a random code using Hamming distance, it is possible to design cryptosystems with smaller keys and ciphertexts. However, the GPT scheme and its successors were broken by the cryptanalysis framework introduced by Overbeck [30, 31] (see also [9] and the structural attack proposed in [13]).

The lessons learned in the design of schemes based on rank metric, as well as the attacks targeting such schemes, made cryptographers confident enough to submit new code-based cryptography schemes to the NIST PQC standardization process. Although these schemes provide appealing performances and key and ciphertext sizes, are they small enough to allow large-scale deployments including resource-constrained devices for IoT?

In this work, we give a positive answer by porting the encryption operation of 5 rank metric cryptosystems to the Arm Cortex-M0 processor, the smallest Arm processor available, demonstrating the possibility to run them on resource-constrained devices. The schemes we have selected are dRANKula [1, 21] and the NIST PQC candidates LAKE, LOCKER, Rank-Ouroboros (which have recently been merged into ROLLO [24]) and RQC [25]. We have selected the first scheme because it is the scheme using rank metric which is the closest rank metric analogue of the original McEliece proposal. The latter four have been chosen because they offer different trade-offs between performance and key and ciphertext sizes.

In our implementation, we have considered only the encryption or encapsulation operations for PKE and KEM schemes, respectively. This is due to the fact that in IoT, lightweight devices are mostly going to initiate the communication to send the information to the recipient. We have taken into account the scheme variants for a classical level of security of 128 bits because it is the minimum security level that NIST is recommending in the Symmetric Lightweight Cryptography competition [28].

In addition, we have kept in mind that Side-Channel Attacks (SCA) are a concern in this context, as these thin devices might be out in the field and reachable by an attacker, and we developed a constant-time implementation for each cryptosystem in order to protect against timing attacks. It is important to note that the overall objective of this work is not to compare the cryptosystems between themselves as they are achieving different security notions ranging from an IND-CCA KEM to IND-CPA PKE, but to prove to the community that rank-based cryptography can be considered for resource-constrained devices.

The paper is structured as follows. We first recall mathematical notions necessary to understand the cryptosystems and describe them. We then discuss the key points and major difficulties we encountered during our implementation work. Next, we present the platform-specific optimization we carried out and the performance of the different schemes. We conclude by enumerating the strengths of the schemes as well as their bottlenecks, hoping to give insights to the community when it comes to parameters selection for rank metric schemes in IoT.

2 ERROR CORRECTING CODES IN RANK METRIC

The development of ECC is due to Richard W. Hamming in 1947. A description of Hamming's code appeared in Claude Shannon's A Mathematical Theory of Communication [35] and was quickly generalized by Marcel J. E. Golay [14].

The general principle of ECC theory is to add more information to the message, which we refer to as redundancy, in order to be able to detect and correct errors that occurred during the transmission. We call the fact of adding redundancy to the message the *encoding step* and the result of the encoding a *codeword*. The mechanism to recover the original message from the codeword which might contain errors is called the *decoding step*. These two operations and the full collection of codewords form a so-called *code*.

The most common type of codes are *linear codes*. For those codes, the encoding step is simply the multiplication of the message by a matrix, called *generator matrix* of the code. Note that as the code is defined as the image of a matrix multiplication it is a vectorial subspace. Precisely, a $[n, k]$-code C over a finite field $\mathbb{F}$ is a vector subspace of $\mathbb{F}^n$ of dimension k, where n is called the length, and k the dimension of the code. A *generator matrix* for an $[n, k]$ code C is thus any $k \times n$ matrix G whose rows form a basis for C. In general there are many generator matrices for a code. Because a linear code is a subspace of a vector space, it is the kernel of some linear transformation. In particular, there is an $(n-k) \times n$ matrix H, called a *parity check matrix* for the $[n, k]$ code C, defined by $C = \{x \in \mathbb{F}^n | Hx^T = 0\}$. In general, there are also several possible parity check matrices for C. The decoding algorithm depends on the structure of the code but will always output the "closest" codeword to the received message if it is within the range of the error capacity. For a comprehensive introduction to the topic, see [22]. To define how "close" two vectors are, we need to define a metric. The more natural choice is usually the Hamming metric. In this metric the distance of two vectors is given by the number of different coordinates between the two. There exists also other metrics. Another popular example is the *rank metric*. To define it, we first need to explain the notion of *rank weight*.

Definition 2.1. Let q be a prime power and let $\mathbb{F}_q$ be the only finite field (up to isomorphism) with q elements. Let $e \in \mathbb{F}_{q^m}^n$ be written as $(e_1, \ldots, e_n)$. Denote by $e_{i,j}$ the j-th component of e_i seen as a vector in $\mathbb{F}_q^m$. Then $rk(e)$ is defined as

$$rk(e) = rk \begin{pmatrix} e_{1,1} & \ldots & en,1 \\ \vdots & & \vdots \\ e_{1,m} & \ldots & e_{n,m} \end{pmatrix}$$

and is called the rank weight of e. The rank distance between two vectors $e, f \in \mathbb{F}_{q^m}^n$ is defined by $rk(e - f)$.

The rank metric was introduced by Gabidulin in [10] in order to introduce codes which can correct errors with repeating patterns, regardless of their Hamming weight.

2.1 Classes of codes needed

Here we present the definition of the codes on which the proposals of Section 3 are based on.

We begin with the definition of a circulant matrix.

Definition 2.2 (Circulant Matrix). A square matrix of size $n \times n$ is said to be circulant if it is of the form:

$$\begin{pmatrix} m_0 & m_1 & \dots & m_{n-1} \\ m_{n-1} & m_0 & \dots & m_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ m_1 & m_2 & \dots & m_0 \end{pmatrix}$$

The following classes of codes will be used to define the schemes we have implemented in this work:

Definition 2.3 (Double Circulant (2-Quasi-Cyclic) Codes). A $[2n, n]_{q^m}$ code C is a double circulant code if it has a generator matrix of the form $(A|B)$ where A and B are two circulant matrices of size n.

Definition 2.4 (Ideal Codes). Let $P(X) \in \mathbb{F}_q[X]$ be a polynomial of degree n and $g_1, g_2 \in \mathbb{F}_{q^m}^n$. Let $G_1(X) = \sum_{i=0}^{n-1} g_{1,i} X^i$ and $G_2(X) = \sum_{i=0}^{n-1} g_{2,i} X^i$ the polynomials associated to g_1 and g_2.

We define the $[2n, n]_{q^m}$ ideal code C of generator (g_1, g_2) as the code with generator matrix

$$\left(\begin{array}{cc|cc} G_1(X) & \bmod P & G_2(X) & \bmod P \\ XG_1(X) & \bmod P & XG_2(X) & \bmod P \\ \multicolumn{2}{c|}{\vdots} & \multicolumn{2}{c}{\vdots} \\ X^{n-1}G_1(X) & \bmod P & X^{n-1}G_2(X) & \bmod P \end{array}\right).$$

If g_1 is invertible, C can be written with generator $(x, x \cdot g_1^{-1} g_2 \bmod P)$.

Let $M_k(R)$ be the set of $k \times k$ matrices over the ring R.

Definition 2.5 (LRPC codes). Let $H \in M_{(n-k)\times n}(\mathbb{F}_{q^m})$ be a full rank matrix such that its coefficients generate an $\mathbb{F}_q$-subspace F of small dimension d.

$$F = < h_{i,j} >_{\mathbb{F}_q} .$$

The code C of parity check matrix H is called an LRPC code of weight d.

Definition 2.6 (Ideal LRPC codes). Let F be a $\mathbb{F}_q$-subspace of dimension d of $\mathbb{F}_{q^m}$, (h_1, h_2) two vectors of $\mathbb{F}_{q^m}^n$ with support in F and $P \in \mathbb{F}_q[X]$ a polynomial of degree n. Let H_1 and H_2 be two matrices defined by

$$H_1 = \begin{pmatrix} h_1 \\ Xh_1 \bmod P \\ \vdots \\ X^{n-1}h_1 \bmod P \end{pmatrix}, H_2 = \begin{pmatrix} h_2 \\ Xh_2 \bmod P \\ \vdots \\ X^{n-1}h_2 \bmod P \end{pmatrix}.$$

The code C with parity check matrix $(H_1|H_2)$ is called an ideal LRPC code of type $[2n, n]_{q^m}$.

Definition 2.7 (Gabidulin codes). Let $k < n \leq m$ be non-negative integers and let $\{g_1, \dots, g_n\} \in \mathbb{F}_{2^m}$, be linearly independent over $\mathbb{F}_2$. Let $[i] = 2^i$ such that $x \to x^{[i]}$ is the ith power of the Frobenius automorphism $x \to x^2$. The $[n, k]$ Gabidulin code $\mathrm{Gab}_{k,n}(\mathbf{g})$, is the $[n, k]$ linear code with generator matrix

$$G = \begin{bmatrix} g_1 & \dots & g_n \\ g_1^{[1]} & \dots & g_n^{[1]} \\ \vdots & \ddots & \vdots \\ g_1^{[k-1]} & \dots & g_n^{[k-1]} \end{bmatrix}$$

that is:

$$\mathrm{Gab}_{k,n}(\mathbf{g}) = \{\mathbf{xG} | \mathbf{x} \in \mathbb{F}_{2^m}^k\}.$$

3 THE SCHEMES

This section describes the code-based schemes considered in this work.

3.1 dRANKula

McEliece cryptosystem is based on the so called Goppa codes [15]. This codes are subfield subcodes of generalized Reed-Solomon codes [32]. Instead, dRANKula, a scheme first proposed in [21] and implemented in [1], is based on Gabidulin codes, which are considered as the analogs of Reed-Solomon codes in rank metric. In particular, dRANKula uses a special subspace for the entries of the scrambling matrix which transforms the private key into the public key. Another essential difference from the traditional McEliece instantiations is that, instead of XORing the encoded plaintext with an error of a given Hamming weight, XOR in dRANKula is done with an error of a specific *rank weight*. Note that the original scheme and its implementation only provide One-Way security (OW-CPA) meanwhile the other schemes are at least proposing IND-CPA security. Hence, we have implemented dRANKula with the SXY transform used to transform the McEliece cryptosystem into a DPKE [34, Appendix D] for a fair comparison. Its KEM algorithm is detailed below:

- Alice selects a Gabidulin code over $\mathbb{F}_{q^m}$ of length n, dimension k with generator matrix G. She then generates a random non-singular matrix $S \in M_k(\mathbb{F}_{q^m})$, a random vectorial subspace $V \subset \mathbb{F}_{q^m}$ of dimension λ and a random non-singular matrix $P \in M_n(V)$.
- Alice defines $\mathrm{pk} = G_{pub} = S \cdot G \cdot P^{-1}$ and $\mathrm{sk} = (G, S, P)$.
- Bob chooses a random vector $e \in \mathbb{F}_{q^m}^n$ of rank weight $t := \lfloor (n-k)/(2\lambda) \rfloor$ and computes $y = \mathrm{m} \cdot G_{pub} + e$, for a message $\mathrm{m} \in \mathbb{F}_{q^m}^k$. The shared secret is $H(e, \mathrm{m})$ where H is a hash function.
- Alice computes $y \cdot P = \mathrm{m}' \cdot S \cdot G + e \cdot P$ and recovers $\mathrm{m}' \cdot S$ by decoding. Finally, she gets m' from $\mathrm{m}' \cdot S$ multiplying it by S^{-1}. From m', she computes e' and derives the secret $H(e', \mathrm{m}')$.

Its parameters for 128 bits of classical security are detailed in Table 1.

3.2 ROLLO

ROLLO is the merge of three initial propositions to the NIST-PQC competition: LAKE, LOCKER and Rank-Ouroboros. The three schemes are detailed in the rest of this section.

q	n	m	k	λ	t
2	60	64	30	3	5
failure rate		pk size		ct size	Sec. lev.
0		14400B		480B	128b

Table 1: Parameters for dRANKula-128

3.2.1 LAKE. LAKE is an IND-CPA KEM running for standardization to NIST's competition. LAKE follows the approach inaugurated by the public key encryption protocol NTRU in 1998 [18]. The main idea behind the protocol is that the secret key consists in the knowledge of a small Euclidean weight vector, which is used to derive a double circulant matrix. This matrix is then seen as a dual matrix of an associated lattice and a specific decoding algorithm based on the knowledge of this small weight dual matrix is used for decryption.

This idea of having as a trapdoor a small weight dual matrix (with a specific associated decoding algorithm) can naturally be generalized to other metrics. It was done in 2013 with MDPC [26] for Hamming metric and also in 2013 for Rank metric with LRPC codes [12]. These three protocols derive from the same basic main idea, adapted for different metrics, which have different properties in terms of efficiency, size of parameters and security reduction.

LAKE is a small variation of the LRPC rank metric approach, by introducing Ideal-LRPC codes, and proposes an IND-CPA KEM for Key Exchange, efficient in terms of size of parameters and computational complexity which benefits from the nice properties of rank metric. The scheme has a failure probability, but this probability is well understood and made very low.

The LAKE KEM algorithm works as follow:

- Alice chooses an irreducible polynomial $P \in \mathbb{F}_q[X]$ of degree n.
- Alice chooses a random vectorial subspace F of $\mathbb{F}_{q^m}$ of dimension d and samples a couple of vectors $(x, y) \in F^n \times F^n$ such that x is invertible $\mod P$.
- Alice computes $h = x^{-1}y \mod P$.
- Alice defines $\mathsf{pk} = (h, P)$ and $\mathsf{sk} = (x, y)$.
- Bob chooses uniformly at random a subspace E of $\mathbb{F}_{q^m}$ of dimension r and samples a couple of vectors $(e_1, e_2) \in E^n \times E^n$.
- Bob computes $c = e_1 + e_2 h \mod P$ and $K = G(E)$ where $G(E)$ is a hash function and outputs the ciphertext c.
- Alice computes $xc = xe_1 + ye_2 \mod P$ and recovers E by decoding, to finally get $K = G(E)$.

LAKE exists in three variants. We have chosen to implement the variant labeled as LAKE-I whose parameters are recalled in Table 2.

q	n	m	d	r	P
2	47	67	6	5	$X^{47} + X^5 + 1$
failure rate		pk size		ct size	Sec. lev.
2^{-30}		394B		394B	128b

Table 2: LAKE-I parameters

3.2.2 LOCKER. The LOCKER PKE proposal is very similar to the LAKE KEM but adapted with parameters supporting very low decryption probability failures. It is proven to be IND-CPA. The scheme is efficient in terms of size of parameters and has a failure probability but this probability is well understood and made very low from 2^{-64} to 2^{-128}.

The LOCKER PKE algorithm works as follow:

- Alice chooses an irreducible polynomial $P \in \mathbb{F}_q[X]$ of degree n.
- Alice chooses a random vectorial subspace F of $\mathbb{F}_{q^m}$ of dimension d and samples a couple of vectors $(x, y) \in F^n \times F^n$ such that x is invertible $\mod P$.
- Alice computes $h = x^{-1}y \mod P$.
- Alice defines $\mathsf{pk} = (h, P)$ and $\mathsf{sk} = (x, y)$.
- Bob chooses uniformly at random a subspace E of $\mathbb{F}_{q^m}$ of dimension r and samples a couple of vectors $(e_1, e_2) \in E^n \times E^n$.
- Bob computes $c = e_1 + e_2 h \mod P$ and $cipher = \mathrm{m} \oplus G(E)$ where G is a hash function and outputs the ciphertext $(c, cipher)$.
- Alice computes $xc = xe_1 + ye_2 \mod P$ and recovers E to finally recover m.

LOCKER exists in nine variants. We have chosen to implement the variant LOCKER-I as it is the most suitable candidate for lightweight devices, see Table 3 for the actual parameters.

q	n	m	d	r	P
2	83	71	7	5	$X^{83} + X^7 + X^4 + X^2 + 1$
failure rate		pk size		ct size	Sec. lev.
2^{-64}		737B		(737 + |m|)B	128b

Table 3: LOCKER-I parameters

3.2.3 Rank-Ouroboros. Rank-Ouroboros is an adaptation for rank metric of the Hamming metric based key exchange Ouroboros [7]. Both Ouroboros, which is now part of the BIKE proposal, and Rank-Ouroboros uses the same approach than the two aforementioned schemes, but having at the same time a reduction to decoding random quasi-cyclic codes, rather than a more specific code. However, this comes at a cost: doubling the size of the ciphertext. The resulting scheme benefits from the features of NTRU-like schemes but has also a reduction to a generic problem, at the cost of doubling the size of the ciphertext. In addition, as all associated decoding algorithm for the NTRU-like family of schemes, there is a decryption failure, but in the case of rank metric, this decryption failure is low and perfectly estimated. The Rank-Ouroboros IND-CPA KEM is explained below:

- Alice samples a random seed and derives a vector $h \in \mathbb{F}_{q^m}^n$ from it. She then samples a pair (x, y) of random vector in $S_{1,w}^n(\mathbb{F}_{q^m})$ which stands for the space of vectors of length n of rank weight w such that their support contains 1. She then computes $s = x + y$. Her public key is (h, s) and her private key is F, the support of x and y.
- Bob chooses a random vectorial subspace F of $\mathbb{F}_{q^m}$ of dimension w and samples vectors $r_1, r_2, e_r \in F^n$ and keeps E,

the support of (r_1, r_2, e_r). He then sends $s_r = r_1 + hr_2$ and $s_e = sr_2 + e_r$.
- Upon receiving (s_r, s_e), Alice computes $e_c = s + e - s_r y$, from which she can easily recover the support E of (r_1, r_2, e_r) through an efficient decoding algorithm.
- The shared secret is $H(E)$.

We have chosen to implement the variant Rank Ouroboros-I whose parameters are described in Table 4.

q	n	m	w	w_r
2	53	89	5	6
failure rate		pk size	ct size	Sec. lev.
2^{-36}		1180B	1180B	128b

Table 4: Rank-Ouroboros-I parameters

3.3 Rank Quasi Cyclic (RQC)

Rank Quasi Cyclic (RQC) is running for standardization to NIST's post quantum competition and is currently under revision for publication in IEEE Transactions on Information Theory. RQC provides both a KEM and a PKE scheme and uses two types of codes: a $[n, k]$ Gabidulin code C, generated by $G \in M_{k\times n}(\mathbb{F}_{q^m})$ and which can correct $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors via an efficient algorithm and a random double circulant $[2n, n]$ code, of parity check matrix $(1, h)$. The polynomial-time algorithms constituting the KEM and the PKE are described below:

- Alice samples at random h and the generator matrix G of a Gabidulin code C. The secret key will be a pair of two random vectors (x, y) of weight w and the public key will be $(h, s = x + h \cdot y)$
- Bob generates a triplet of random vectors (e, r_1, r_2) of given weight. Then he computes $u = r_1 + H \cdot r_2$ and $v = \mathrm{m}G + s \cdot r_2 + e$ and sends $c = (u, v)$.
- To decrypt, Alice simply decodes $v - u \cdot y$.

In this study we have implemented the RQC-I KEM variant. To go from the PKE variant to the KEM, the message m is simply sampled at random and the shared secret is the hash of m.

Notice that the generator matrix G of the code C is publicly known, so the security of the scheme and the ability to decrypt do not rely on the knowledge of the error correcting code C being used. The parameters of the variant we implemented are recalled in Table 5.

q	n	m	k	w	w_r
2	67	89	7	6	5
failure rate		pk size		ct size	Sec. lev.
0		1491B		1555B	128b

Table 5: RQC-I parameters

4 BINARY FIELD ARITHMETIC ON THE ARM-CORTEX-M0

For this work, we have targeted the smallest Arm microprocessor available. The Arm-Cortex M0 is a 32-bit microprocessor with an ultra-low gate count based on the Armv6-M architecture [3] with THUMB instructions set as well as a subset of THUMB-2 instructions set available [2]. More precisely, we have deployed our code on a development board NXP LPC11u24 as showed on Figure 1. The board runs at 48 MHz and has 8 KB of RAM and 32 KB of FLASH memory. The full specification can be found on NXP's website [29]. NXP's Software Development Kit (SDK) and dedicated operating system called "mbed" allows for very efficient and fast prototyping on such devices. The code was compiled on top of mbed OS using arm-none-eabi-g++ version 6.3.1 and the -Os flag (surface optimization).

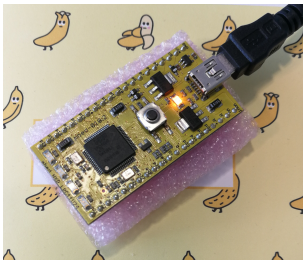

Figure 1: NXP LPC11u24 board

The main, and common, component of the schemes under scrutiny is the binary field arithmetic. Indeed, all of our targeted cryptosystems heavily rely on addition and multiplication in extensions of $\mathbb{F}_2$ (also called binary fields). Hence, it is worth it to optimize these operations to speed up the running time of the algorithms.

The Arm-Cortex M0 is a 32-bit processor. Since the bit size of a field element is given by the parameter m, it is impossible to fit a field element in one register for any of the schemes, forcing us to use either two registers for dRANKula ($m = 64$) or three for LAKE ($m = 67$), LOCKER ($m = 71$), Rank-Ouroboros ($m = 89$) and RQC ($m = 89$). The addition in a binary fields is a simple XOR, hence the addition algorithm is simply XORing each corresponding register of the arrays of 32-bit registers. However, the multiplication is not as straightforward.

The multiplication of two elements of a binary field is executed in two steps: first perform a carry-less multiplication and, second, perform a reduction. Let $a, b \in \mathbb{F}_{2^m}$ be two field elements and consider their polynomial representation $a(X) = \sum_{i=0}^{m-1} a_i X^i, b(X) = \sum_{i=0}^{m-1} b_i X^i, a_i, b_i \in \mathbb{F}_2$ in $\mathbb{F}_2[X]/P(X)$ where P is an irreducible polynomial of degree m, usually a trinomial or a pentanomial.

For the second step, the reduction, we use standard techniques, e.g. as in [17, Section 2.3.5].

As far as it concern the first step, the carry-less multiplication $a \otimes b$ is a standard polynomial multiplication

$$a \otimes b \cong a(X)b(X) = \sum_{i=0}^{m-1} a_i \sum_{j=0}^{m-1} b_j X^{i+j},$$

where the sum is performed in $\mathbb{F}_2[X]$ and which results in a polynomial of degree at most $2m-2$. Now remark that $X^m = P(X) - X^m$ in $\mathbb{F}_2[X]/P(X)$ and use this relation in order to obtain a polynomial of degree $m-1$ (the reduction step). When ported on software, this translate to the following algorithm:

Algorithm 1: Carry-less multiplication of $a(X)$ and $b(X)$

```
c = 0
for i = 0 to m − 1 do
    if a_i = 1 then
        c = c XOR bX^i
return c
```

The only tricky point of the algorithm is the multiplication of $b(X)$ by X^i but it is actually a simple shift of b (left or right, depending on the internal representation choice).

This algorithm is not asynchronous as the number of addition in $\mathbb{F}_2[X]$ depends on the Hamming weight of a. Leaking this information through timing measurement is clearly not suitable, especially since in most of the cases, a will be the message that we are encrypting. A simple method to fix this issue is to use a mask which depends on a_i, more specifically we set $mask = 0 - a_i$ (the subtraction operation being the usual integer subtraction) at each step of the algorithm. The mask will be equal to 0 if a_i is 0, -1 otherwise. The value -1 being represented as `0xFFFFFFFF` in the Cortex-M0, we can just apply a logical bit-wise AND to $b \cdot X^i$ before XORing it to c. In definitive, we obtain the following algorithm:

Algorithm 2: Asynchronous carry-less multiplication of $a(X)$ and $b(X)$

```
c = 0
for i = 0 to m − 1 do
    mask = 0 − a_i
    c = c XOR (bX^i AND mask)
return c
```

This multiplication being the bottleneck of the cryptosystems under consideration, we needed to optimize it. Indeed, the only non trivial operations in the schemes are the hash functions and the multiplication in the binary fields. There exist different methods giving asymptotic improvement on the carry-less multiplication, see [6] for a recent survey, but the complexity of these methods make them not suitable for such a resource constrained target.

Thus, we focused on optimizing the only non-trivial part of Algorithm 2: the shift. Indeed, we recall that none of the elements we need to multiply fit in a single 32-bit register, hence the shift has to take care of transferring the bits shifted out from the lower register to the higher one. The usual method consists in using the following algorithm:

The algorithm above takes a total of $2 \times$ `array_length` $- 1$ shifts and `array_length` $- 1$ ORs. For the needs of our carry-less multiplication, one can note that we only need to shift the array representing b by 1 at each step of the algorithm which allows us a nice optimization using the carry flag at assembly level. In fact, the

Algorithm 3: Shift right an array of 32-bit registers by $r < 32$

```
for i = array_length −1 to 1 do
    array[i] = array[i] » r OR array[i − 1] « (32 − r)
array[0] = array[0] » r
return array
```

instruction `LSLS` (logical shift left) or `LSRS` (logical shift right) only set the carry flag to 1 if the last bit shifted out was a 1. We can then use the instruction `ADCS` which adds two registers plus the carry flag to add the last bit shifted out from the first register to the next register. This results in `array_length` shifts and `array_length` -1 additions. For the binary fields in consideration, we need to shift arrays from two to four cells, hence we can save up to three shifts per iteration of the `for` loop of Algorithm 2. The assembly code of the left shift by one of an array of four cells is presented in Algorithm 4 as an example and we report the speeds of the implementation with and without assembly optimization in Table 6. This function takes as an argument R0 as the address of the first element of the array representing our binary field element.

Algorithm 4: Shift left an array of 32-bit registers by 1 using ADD with Carry

```
MOVS R3, #0
LDR R1, [R0]
LDR R2, [R0, #4]
LSLS R1, #1
LSLS R2, #1
ADCS R1, R1, R3
STR R1, [R0]
LDR R1, [R0, #8]
LSLS R1, #1
ADCS R2, R2, R3
STR R2, [R0, #4]
LDR R2, [R0, #12]
LSLS R2, #1
ADCS R1, R1, R3
STR R1, [R0, #8]
STR R2, [R0, #12]
BX LR
```

Field	Non-opt. mul	Opt. mul.
$\mathbb{F}_{2^{89}}$	177.04μs/op	173.38μs/op
$\mathbb{F}_{2^{71}}$	137.68μs/op	134.77μs/op
$\mathbb{F}_{2^{67}}$	126.13μs/op	123.38μs/op
$\mathbb{F}_{2^{64}}$	110.27μs/op	108.87μs/op

Table 6: Comparison of the speed of optimized and non-optimized carry-less multiplications

Even though the gains are marginal (around 2%), it is still interesting to obtain such gains as they reduce both the execution time and the power consumption of the algorithms.

The last optimization that we perform is about the reduction step. In each scheme, the finite field multiplication is only performed within a matrix multiplication when performing the encryption / encapsulation operation. One can opt for a strategy limiting the number of reductions by performing it at the end of each scalar product. Indeed, the result of a XOR operation has the same bit size than the two addends meaning that we can perform the reduction after having XORed the results of all the carry-less multiplications. That is, for $x, y \in \mathbb{F}2^{mn}$, the scalar product $x \cdot y$ is computed as

$$x \cdot y = ((x_1 \otimes y_1) \text{ XOR } \ldots \text{ XOR } (x_n \otimes y_n)) \mod P$$

rather than

$$x \cdot y = ((x_1 \otimes y_1) \mod P) \text{ XOR } \ldots \text{ XOR } ((x_n \otimes y_n) \mod P)$$

5 PERFORMANCE AND PRACTICAL CONSIDERATIONS

In this section, we report the running time of each scheme on the Arm-Cortex M0 with and without the constant-time carry-less multiplication. We bring to the attention of the reader the fact that the microprocessor does not have enough memory to store the full public key and keep the necessary state to perform the full encryption operation of any of the cryptosystems we have implemented except LAKE and LOCKER. We only detail the execution time without counting the transfers of the necessary part of the public key from external memory as the speed of those transfers highly depends on the device in which the microprocessor will be embedded in. For information, a maximum data transfer speed of up to 1MB/s can be achieved. However, we describe the strategy we used to deal with the issue of the public key and ciphertexts not fitting in memory.

All the schemes are using a hash function. We have chosen to use BLAKE2s and the XOF BLAKE2X [4, 5] over SHA2 for performance and over SHA3 for the fact that it is more suitable for a 32-bit platform.

dRANKula is the scheme which has the largest public key of all five. The obvious strategy to proceed to the encryption is to transfer the columns of the public key one by one and perform the scalar product between the column and the message and then add an error to obtain one component of the ciphertext. The ciphertext can be kept in memory until the end of the computation, there is no need to send it component by component.

LAKE and LOCKER are the only two schemes whose public key and ciphertext can fit entirely in the microprocessor memory during the encryption. Hence, there is no need to adopt any strategy to deal with any memory transfer.

For Rank-Ouroboros, we need to split the encryption operation into two steps. The first consists into transferring h to the microprocessor and compute s_r. We then transfer s and computed s_e. The computation of the shared secret can be performed afterwards.

For RQC, the encryption operation needs to be split into three steps. The first consists into transferring h to the microprocessor and compute u. We then transfer s and compute $s \cdot r_2 + e$. Finally, we transfer G to compute $v = \mathrm{m} \cdot G + s \cdot r_2 + e$.

The execution times are reported in Tables 7 and 8. As it can be observed, a constant-time implementation can represent an overhead of around 50% in the worst-case.

Scheme	Time in μs per encryption operation	
	Constant-time	Non constant-time
LOCKER	940,096	697,032

Table 7: Execution time of the PKE scheme on Arm-Cortex M0

Scheme	Time in μs per encapsulation operation	
	Constant-time	Non constant-time
dRANKula	119,559	85,466
LAKE	277,430	206,382
Rank-Ouroboros	994,048	650,811
RQC	1,666,554	1,197,113

Table 8: Execution times of the four KEM schemes on Arm-Cortex M0

Finally, in Table 9 we recall the performance of the reference implementation of the schemes under scrutiny on Intel Core i7. The exact reference of the benchmark platforms can be found in [1, 24, 25]. By comparing Tables 7, 8 and 9 there are differences worth noting. We highlight that the choice of the field on which every scheme is based has an important impact when porting to micro-controllers. Indeed, dRANKula is obviously not the fastest option on PC but, because it only deals with field elements fitting on 2 machine words, it outperforms LAKE (the fastest on PC) on Arm-Cortex M0. The significance of the base field is also striking when one compares the ratio between LAKE and LOCKER performance on Arm-Cortex M0 and on PC. With just 4 extra bits for each element, LAKE becomes three times faster than LOCKER on a resource-constrained device whereas it is only twice as fast on PC.

Scheme	Time in μs per encoding operation
dRANKula	334.28
LAKE	85.71
LOCKER	157.14
Rank-Ouroboros	280.00
RQC	562.85

Table 9: Performance of the five schemes on Intel Core i7

6 CONCLUSIONS

At the light of our experiments, we show that porting the encryption operation of rank metric schemes on resource-constrained devices while maintaining correct performance and minimal protection against side-channel attacks is possible. When compared to lattice-based cryptography, rank-metric schemes are slower by a factor of two at minimum (see [37]) but are still a viable option. Also, one can note that implementing a constant-time carry-less multiplication brings an overhead varying from 35% to 50% depending on the scheme.

Between the five schemes under consideration, dRANKula is clearly the fastest option. The second fastest option is LAKE but

it has the additional advantage of avoiding extra memory transfer due to the size of the public key at the expense of a non-null failure rate.

We leave as a future work the study of feasibility of key generation and decoding algorithms on thin devices as well as a more in-depth analysis of possible side-channels of the schemes.

REFERENCES

[1] A Al Abdouli, Mohamed Al Ali, Emanuele Bellini, Florian Caullery, Alexandros Hasikos, Marc Manzano, and Victor Mateu. 2018. DRANKULA, a McEliece-like rank metric based cryptosystem implementation. In *Proceedings of SECRYPT*.

[2] ARM. 2009. Cortex-M0 - Technical Reference Manual. (2009). https://static.docs.arm.com/ddi0432/c/DDI0432C_cortex_m0_r0p0_trm.pdf?_ga=2.65362413.2132650085.1547543449-853925946.1539837347

[3] ARM. 2017. ARMv6-M Architecture, Reference Manual. (2017). https://static.docs.arm.com/ddi0419/d/DDI0419D_armv6m_arm.pdf

[4] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2013. BLAKE2. (2013). https://blake2.net/blake2.pdf

[5] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. 2016. BLAKE2X. (2016). https://blake2.net/blake2x.pdf

[6] Alessandro De Piccoli, Andrea Visconti, and Ottavio Rizzo. 2018. Polynomial multiplication over binary finite fields: new upper bounds. (03 2018). https://eprint.iacr.org/2018/091.pdf

[7] Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. 2017. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In *International Workshop on Post-Quantum Cryptography*. Springer, 18–34.

[8] M. H. Devoret and R. J. Schoelkopf. 2013. Superconducting Circuits for Quantum Information: An Outlook. *Science* 339, 6124 (2013), 1169–1174.

[9] Cédric Faure and Pierre Loidreau. 2006. *A New Public-Key Cryptosystem Based on the Problem of Reconstructing p-Polynomials*. Vol. 3969. 304–315. https://doi.org/10.1007/11779360_24

[10] Ernest Mukhamedovich Gabidulin. 1985. Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii* 21, 1 (1985), 3–16.

[11] E. M. Gabidulin, A. V. Paramonov, and O. V. Tretjakov. 1991. *Ideals over a Non-Commutative Ring and their Application in Cryptology*. 482–489.

[12] Philippe Gaborit, Gaétan Murat, Olivier Ruatta, and Gilles Zémor. 2013. Low Rank Parity Check codes and their application to cryptography. In *Proceedings of the Workshop on Coding and Cryptography WCC-2013, Bergen, Norway*.

[13] Philippe Gaborit, Ayoub Otmani, and Hervé Talé Kalachi. 2018. Polynomial-time key recovery attack on the Faure–Loidreau scheme based on Gabidulin codes. *Designs, Codes and Cryptography* 86, 7 (01 Jul 2018), 1391–1403.

[14] Marcel Golay. 1949. Notes on Digital Coding. *Proc.I.R.E., IEEE* (1949).

[15] Valerii Denisovich Goppa. 1970. A new class of linear correcting codes. *Problemy Peredachi Informatsii* 6, 3 (1970), 24–30.

[16] Qian Guo, Thomas Johansson, and Paul Stankovski Wagner. 2018. A Key Recovery Reaction Attack on QC-MDPC. *IEEE Transactions on Information Theory* (22 10 2018).

[17] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. 2006. *Guide to elliptic curve cryptography*. Springer Science & Business Media.

[18] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. 1998. NTRU: A Ring-Based Public Key Cryptosystem. In *Lecture Notes in Computer Science*. Springer-Verlag, 267–288.

[19] Ghazal Kachigar and Jean-Pierre Tillich. 2017. Quantum Information Set Decoding Algorithms. In *Post-Quantum Cryptography*, Tanja Lange and Tsuyoshi Takagi (Eds.). Springer International Publishing, Cham, 69–89.

[20] In Lee and Kyoochun Lee. 2015. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons* 58, 4 (2015), 431–440.

[21] Pierre Loidreau. 2017. *A New Rank Metric Codes Based Encryption Scheme*. 3–17.

[22] F.J. MacWilliams and N.J.A. Sloane. 1978. *The Theory of Error-Correcting Codes*.

[23] R. J. McEliece. 1978. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report* 44 (1978), 114–116.

[24] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. 2018. ROLLO - Rank-Ouroboros, LAKE & LOCKER. (2018). http://pqc-rollo.org/doc/rollo-specification_2018-11-30.pdf

[25] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. 2017. Rank Quasi-Cyclic (RQC). (2017). https://pqc-rqc.org/doc/rqc-specification_2017-11-30.pdf

[26] R. Misoczki, J. P. Tillich, N. Sendrier, and P. S. L. M. Barreto. 2013. MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. In *2013 IEEE International Symposium on Information Theory*. 2069–2073.

[27] NIST. 2018. PQC Call for Proposals. (2018). Available at https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals.

[28] NIST. 2019. Lightweight Cryptography Call for Proposals. (2019). Available at https://csrc.nist.gov/projects/lightweight-cryptography.

[29] NXP. 2019. OM13032: Arm mbed LPC11U24 Board. (2019). https://www.nxp.com/support/developer-resources/software-development-tools/lpc-developer-resources-/lpc-partner-evaluation-and-development-boards/arm-mbed-lpc11u24-board:OM13032

[30] Raphael Overbeck. 2005. *A New Structural Attack for GPT and Variants*. 50–63.

[31] R Overbeck. 2008. Structural attacks for public-key cryptosystems based on gabidulin codes. *Journal of Cryptology* 21, 2 (2008), 280–301.

[32] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.

[33] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM* 21, 2 (Feb. 1978), 120–126.

[34] Tsunekazu Saito, Keita Xagawa, and Takashi Yamakawa. 2018. Tightly-Secure Key-Encapsulation Mechanism in the Quantum Random Oracle Model. In *Advances in Cryptology – EUROCRYPT 2018*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer International Publishing, Cham, 520–551.

[35] C. E. Shannon. 1948. A Mathematical Theory of Communication. *The Bell System Technical Journal* (1948).

[36] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 26, 5 (1997), 1484–1509.

[37] Rui Xu, Chi Cheng, Yue Qin, and Tao Jiang. 2018. Lighting the Way to a Smart World: Lattice-Based Cryptography for Internet of Things. *CoRR* abs/1805.04880 (2018).