Analysis of Layered ROLLO-I: A BII-LRPC code-based KEM

Seongtaek Chee[1], Kyung Chul Jeong[1], Tanja Lange[2], Nari Lee[1], Alex Pellegrini[2], and Hansol Ryu[1]

[1] The Affiliated Institute of ETRI, Daejeon 34044, Republic of Korea
{chee, jeongkc, narilee, hansolryu}@nsr.re.kr
[2] Eindhoven University of Technology, 5612 AZ Eindhoven, Netherlands
tanja@hyperelliptic.org, alex.pellegrini@live.com

Abstract. We analyze Layered ROLLO-I, a code-based cryptosystem published in IEEE Communications Letters and submitted to the Korean post-quantum cryptography competition. Four versions of Layered ROLLO-I have been proposed in the competition. We show that the first two versions do not provide the claimed security against rank decoding attacks and give reductions to small instances of the original ROLLO-I scheme, which was a candidate in the NIST competition and eliminated there due to rank decoding attacks. As a second contribution, we provide two efficient message recovery attacks, affecting every security level of the first three versions of Layered ROLLO-I and security levels 128 and 192 of the fourth version.

Keywords: Post-quantum cryptography · code-based cryptography · rank-metric code · BII-LRPC code · ROLLO-I.

1 Introduction

The advancement of research in quantum technologies poses an increasing threat to, e.g., the daily security of communications. For example, Shor's algorithm [18] enables a quantum computer to solve the classically hard problems underlying current widely adopted asymmetric cryptography. A new area of cryptographic research, called *post-quantum cryptography* [5], is active in developing cryptosystems that can resist such a threat. One branch of post-quantum cryptography, called *code-based cryptography*, studies cryptosystems which base their security on hard problems coming from the theory of error-correcting codes. For an up-to-date survey of code-based cryptography, we refer the reader to [19]. The very first proposal of such a cryptosystem has been described in 1978 by Robert J. McEliece [15] which uses Goppa codes [7]. Despite more than 45 years of analysis the cryptosystem essentially still maintains the same security, however, the system imposes huge public keys which are prohibitive for applications that frequently send public keys. Many attempts have been made to decrease the key size, mainly by replacing Goppa codes with families of structured linear codes, but promptly being proved insecure shortly after.

Another strategy to tackle the key size issue is that of considering codes endowed with the rank metric, which allows a more compact representation of codes. Rank metric codes were introduced to cryptography by Gabidulin, Paramonov, and Tretjakov at Eurocrypt'91 [6] but attacked and broken 10 years later by Overbeck in several papers, covered in [16]. In 2017, the NIST competition on post-quantum cryptography saw a revival of rank-metric codes in rounds 1 and 2 until some new algebraic attacks [2,4] were found near the end of round 2. These attacks did not completely break the systems and larger parameters were proposed that would resist the new attacks, but the attacks showed that rank-metric codes were not mature enough to be used. Furthermore, the larger parameters would have hurt the performance of the systems. Consequently, NIST deselected all rank-metric-based designs from advancing to round 3.

In this paper, we analyze a blockwise interleaved ideal low-rank parity-check (BII-LRPC) code-based KEM, which was proposed by Kim, Kim, and No in [12] and submitted to the Korean post-quantum Cryptography (KpqC) competition under the name Layered ROLLO-I [8]. Layered ROLLO-I is a modified version of the NIST candidate ROLLO [1], and particularly of ROLLO-I. Layered ROLLO-I adds additional structure to increase the length of the codewords that an attacker is faced with while at the same time permitting the legitimate receiver, using the secret key, to peel off this layer of structure and then to perform rank decoding with parameters which are even smaller than in ROLLO-I, thus increasing performance.

In this paper, we describe attacks on four versions of Layered ROLLO-I that have been released subsequently to the communication of our analyses on the KpqC bulletin. We show how to reduce every instance of the first two versions to an instance of the original ROLLO-I at the smaller parameter size that Layered ROLLO-I uses internally. This shows that the additional structure does not add any security. As a consequence, the parameter sets proposed for Layered ROLLO-I offer less security than the parameter sets for the corresponding levels in the original ROLLO-I. After these two reduction attacks, we show a message recovery attack that works very efficiently for all parameter sets in the first three versions and for two out of three levels for the fourth version.

We emphasize that the message recovery attack applies to all versions and thus is the more powerful attack. We chose to present our results in this order to build up to the more devastating attack and included both approaches because possible fixes would need to target different aspects of the system and in part have mitigated the attacks.

This paper is organized as follows. The next section is dedicated to the needed notation and background along with the description of ROLLO-I and the most relevant approaches to Rank Syndrome Decoding (RSD). For the sake of simplicity, we will refrain from introducing the entire framework of (ideal and blockwise interleaved ideal) low-rank parity check (LRPC) codes, since these notions will not be directly used in the attacks and analysis. In Section 3.1, we give the specification of Layered ROLLO-I [8,12] and propose an attack reducing Layered ROLLO-I to ROLLO-I and recalculate costs of RSD attacks following the improvements in [3,4,2] (covered in Section 2). In Section 3.2, we describe the first Modified Layered ROLLO-I (MLR1) system [9]. We then show that we can adapt the attack in Section 3.1 to reduce this system to ROLLO-I, again with smaller parameters, and further improve on RSD attacks complexities. Section 4 introduces two message recovery attacks. In Section 4.1, we describe the second modified Layered ROLLO-I (MLR2) [10] that resists the reduction attacks presented in Section 3. We propose an efficient message recovery attack that can be applied to all the versions of Section 3 and MLR2. Finally, in Section 4.3 we introduce the third modified Layered ROLLO-I (MLR3) [11] and a message recovery attack. This attack recovers messages efficiently for security levels 128 and 192 of MLR3.

2 Notation and background

This section gives the necessary background on rank-metric codes for the rest of the paper. Let $\{\alpha_1, \dots, \alpha_m\}$ be a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Write $v \in \mathbb{F}_{q^m}$ uniquely as $v = \sum_{i=1}^{m} V_i \alpha_i$, $V_i \in \mathbb{F}_q$ for all i. So v can be represented as $(V_1, \dots, V_m) \in \mathbb{F}_q^m$. We will call this the *vector representation* of v. Extend this process to $\mathbf{v} = (v_1, \dots, v_n) \in \mathbb{F}_{q^m}^n$ defining a map $\mathsf{Mat} : \mathbb{F}_{q^m}^n \to \mathbb{F}_q^{m\times n}$ by:

$$\mathbf{v} \mapsto \begin{bmatrix} V_{11} & V_{21} & \dots & V_{n1} \\ V_{12} & V_{22} & \dots & V_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ V_{1m} & V_{2m} & \dots & V_{nm} \end{bmatrix}.$$

Definition 1. *The rank weight of* $\mathbf{v} \in \mathbb{F}_{q^m}^n$ *is defined as* $\mathsf{wt}_R(\mathbf{v}) := \mathsf{rk}_q(\mathsf{Mat}(\mathbf{v}))$ *and the rank distance between* $\mathbf{v}, \mathbf{w} \in \mathbb{F}_{q^m}^n$ *is* $d_R(\mathbf{v}, \mathbf{w}) := \mathsf{wt}_R(\mathbf{v} - \mathbf{w})$.

Remark 2. It can be shown that the rank distance does not depend on the choice of the basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. In particular, the choice of the basis is irrelevant for the results in this document.

When talking about the space spanned by $\mathbf{v} \in \mathbb{F}_{q^m}^n$, denoted as $\langle \mathbf{v} \rangle$, we mean the $\mathbb{F}_q$-subspace of $\mathbb{F}_q^m$ spanned by the columns of $\mathsf{Mat}(\mathbf{v})$.

For completeness, we introduce the Hamming weight and the Hamming distance. These notions will be used in our message recovery attacks.

The *Hamming weight* of a vector $\mathbf{v} \in \mathbb{F}_{q^m}^n$ is defined as $\mathsf{wt}_H(\mathbf{v}) := \#\{i \in \{1, \dots, n\} \mid v_i \neq 0\}$ and the Hamming distance between vectors $\mathbf{v}, \mathbf{w} \in \mathbb{F}_{q^m}^n$ is defined as $d_H(\mathbf{v}, \mathbf{w}) := \mathsf{wt}_H(\mathbf{v} - \mathbf{w})$.

Let $D = d_R$ or $D = d_H$. Then an $[n, k, d]$-code C with respect to D over $\mathbb{F}_{q^m}$ is a k-dimensional $\mathbb{F}_{q^m}$-linear subspace of $\mathbb{F}_{q^m}^n$ with *minimum distance*

$$d := \min_{\mathbf{a}, \mathbf{b} \in C,\, \mathbf{a} \neq \mathbf{b}} D(\mathbf{a}, \mathbf{b})$$

and *correction capability* $\lfloor (d-1)/2 \rfloor$. If $D = d_R$ (resp. $D = d_H$) then the code C is also called a *rank-metric* (resp. *Hamming-metric*) code. All codes in this document are linear over the field extension $\mathbb{F}_{q^m}$.

We say that $G \in \mathbb{F}_{q^m}^{k\times n}$ is a *generator matrix* of C if its rows span C. We say that $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$ is a *parity check matrix* of C if C is the right-kernel of H.

In the specifications of this paper, we will follow the notation of [12] with minor changes. Denote by $S_w^n(\mathbb{F}_{q^m})$ the set of vectors of length n and rank weight w over $\mathbb{F}_{q^m}$:

$$S_w^n(\mathbb{F}_{q^m}) = \{\mathbf{v} \in \mathbb{F}_{q^m}^n \mid \mathsf{wt}_R(\mathbf{v}) = w\}.$$

Let $\mathbf{w} = \mathbf{c} + \mathbf{e}$ for some codeword $\mathbf{c} \in C$ and error vector $\mathbf{e} \in \mathbb{F}_{q^m}^n$ with $\mathsf{wt}_R(\mathbf{e}) \leq r$ and let $\mathbf{s} = \mathbf{w}H^T \in \mathbb{F}_{q^m}^{n-k}$ be the *syndrome* of $\mathbf{w}$ using a parity-check matrix H of the code.

The Rank Support Recovery ($\mathsf{RSR}(F, \mathbf{s}, r)$) algorithm is used as a decoder in the decapsulation procedures of ROLLO-I and the follow-up designs. It recovers the support E of (the $\mathbb{F}_q$-linear subspace of $\mathbb{F}_{q^m}$ generated by) the error vector $\mathbf{e}$ given the support F of the dual code[3], the syndrome $\mathbf{s}$, and the rank r of the error. This corresponds to actually finding the error coordinates, by solving a linear system of equations (see p. 13 of the ROLLO specification [1]).

Let $P(x) \in \mathbb{F}_{q^m}[x]$ be a polynomial of degree n. We can add an algebraic structure to the vector space $\mathbb{F}_{q^m}^n$ by interpreting vectors as coefficient vectors of the ring $\mathbb{F}_{q^m}[x]/(P(x))$, where $(P(x))$ is the ideal of $\mathbb{F}_{q^m}[x]$ generated by $P(x)$. Given $\mathbf{u} = (u_0, \ldots, u_{n-1}) \in \mathbb{F}_{q^m}^n$, denote by $\mathbf{u}(x) \in \mathbb{F}_{q^m}[x]$ the polynomial $\mathbf{u}(x) = \sum_{i=0}^{n-1} u_i x^i$. Given $\mathbf{u}, \mathbf{v} \in \mathbb{F}_{q^m}^n$, we define their product $\mathbf{uv}$ as the unique vector $\mathbf{w} \in \mathbb{F}_{q^m}^n$ such that $\mathbf{w}(x) = \mathbf{u}(x)\mathbf{v}(x) \bmod P(x)$. Similarly, we define $Q\mathbf{u} = Q(x)\mathbf{u}(x) \bmod P(x)$ for $Q(x) \in \mathbb{F}_{q^m}[x]$ and define $\mathbf{u}^{-1}$ for $\mathbf{u}(x)$ invertible modulo $P(x)$.

Given two polynomials $R(x), P(x) \in \mathbb{F}_{q^m}[x]$ the matrix representing multiplication by $R(x) \bmod P(x)$ is denoted by

$$M_{R,P} = \begin{pmatrix} R(x) \bmod P(x) \\ R(x)x \bmod P(x) \\ \vdots \\ R(x)x^{\deg(P(x))-1} \bmod P(x) \end{pmatrix}, \tag{1}$$

where each row consists of the coefficient vector of $R(x)x^i \bmod P(x)$ for $i = 0, \ldots, \deg(P(x)) - 1$.

Let A be any $n \times m$ matrix, with $n, m \in \mathbb{N}$. We denote by $A[a : b, c : d]$, with $a < b \in [1, n]$ and $c < d \in [1, m]$, the submatrix of A consisting of the rows in the range $[a, b]$ and columns in the range $[c, d]$. We omit a and b, i.e. use $A[:, c : d]$, to denote the submatrix consisting of all the rows and columns in $[c, d]$. Similarly, for all the columns. With this notation $A = A[:, :]$. If $S_1 \subset [1, n]$ and $S_2 \subset [1, m]$ we denote by $A[S_1, S_2]$ the submatrix of A consisting of rows indexed by S_1 and columns indexed by S_2.

[3] Thus F defines the parity check for the code.

2.1 ROLLO-I

We give a simple description of ROLLO-I [1], which is the core of Layered ROLLO-I.

The values (q, n, m, r, d, P) are the system parameters, where q, n, m, r, d are integers and $P(x) \in \mathbb{F}_q[x]$ is a primitive polynomial of degree n.

- KeyGen:
  - Pick random $\mathbf{x}, \mathbf{y} \in S_d^n(\mathbb{F}_{q^m})$.
  - Set $\mathbf{h}(x) = \mathbf{x}(x)^{-1}\mathbf{y}(x) \bmod P(x)$.
  - Return $\mathsf{pk} = \mathbf{h}$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$.
- Encap(pk):
  - Pick random $(\mathbf{e}_1, \mathbf{e}_2) \in S_r^{2n}(\mathbb{F}_{q^m})$.
  - Set $E = \langle \mathbf{e}_1, \mathbf{e}_2 \rangle$, where $\langle \mathbf{e}_1, \mathbf{e}_2 \rangle$ denotes the $\mathbb{F}_q$-vector space spanned by the columns of $\mathbf{e}_1$ and $\mathbf{e}_2$ (interpreted as vectors in $\mathbb{F}_q^m$).
  - Return $K = \mathsf{hash}(E)$ and $\mathbf{c}(x) = \mathbf{e}_1(x) + \mathbf{e}_2(x)\mathbf{h}(x) \bmod P(x)$.
- Decap(sk):
  - Set $\mathbf{s}(x) = \mathbf{x}(x)\mathbf{c}(x) \bmod P(x)$, $F = \langle \mathbf{x}, \mathbf{y} \rangle$ and $E = \mathsf{RSR}(F, \mathbf{s}, r)$.
  - Return $K = \mathsf{hash}(E)$.

ROLLO-I, and consequently Layered ROLLO-I, base their security on a special configuration of the *Rank Syndrome Decoding* problem.

Definition 3 (RSD problem). *Given a matrix $H \in \mathbb{F}_{q^m}^{(n-k)\times n}$, an element $\mathbf{s} \in \mathbb{F}_{q^m}^{n-k}$ and a positive integer $w \in \mathbb{Z}$, the $RSD(n, k, \mathbf{s}, w)$ problem asks to compute $\mathbf{v} \in \mathbb{F}_{q^m}^n$ such that $H\mathbf{v}^\top = \mathbf{s}$ and $\mathsf{wt}_R(\mathbf{v}) = w$.*

NIST deselected ROLLO-I in the second round of its post-quantum standardization effort, after the development of a new algebraic attack [2] on the RSD problem. The attack substantially associates a system of multivariate equations over $\mathbb{F}_{q^m}$ to the RSD instance and employs Gröbner basis techniques to find the solution. Among the proposed variants of the attack, the most relevant for this paper has complexity

$$\mathcal{O}\left(\left(\frac{((m+n)w)^{w+1}}{(w+1)!}\right)^{\omega}\right), \tag{2}$$

where $\omega = 2.3727$ is the exponent for matrix multiplication.

Further improvements to algebraic attacks have been introduced shortly after in [4], describing solving approaches specialized to the characteristic of the system of equations associated with the RSD instance. One approach that turned out to be quite effective to the RSD instances treated in this paper targets the case that the system of equation is over-determined. Without going too much in the detail, the complexity of such approach in solving $RSD(n, k, \mathbf{s}, w)$ is given by

$$\mathcal{O}\left(m\binom{n+k-1}{w}\binom{n}{w}^{\omega-1}\right). \tag{3}$$

It is worth noting that one of the approaches has been proven too optimistic in a later paper [3], hence has not been taken in consideration in our work. Nevertheless, the latter paper proposes a further approach that conjecturally fixes the bugged strategy, backed by experimental results. The idea is that the system associated with the RSD instance considers both polynomials over the base field $\mathbb{F}_q$ as well as over the extension field $\mathbb{F}_{q^m}$. Polynomials are then homogenized multiplying by degree-b monomials and then the system is solved by linearization, where b is minimal such that $\mathcal{N}_b^{\mathbb{F}_q} \geq \mathcal{M}_b^{\mathbb{F}_q} - 1$, following the notation

$$\begin{aligned}
\mathcal{N}_b^{\mathbb{F}_q} &= \mathcal{N}_b^{\mathbb{F}_{q^m}} - \mathcal{N}_{b,syz}^{\mathbb{F}_q},\\
\mathcal{N}_b^{\mathbb{F}_{q^m}} &= \sum_{s=1}^{k} \binom{n-s}{t}\binom{k+b-1-s}{b-1} - \binom{n-k-1}{t}\binom{k-b-1}{b},\\
\mathcal{N}_{b,syz}^{\mathbb{F}_q} &= (m-1)\sum_{s=1}^{b}(-1)^{(s+1)}\binom{k+b-s-1}{b-s}\binom{n-k-1}{t+s}, \text{ and}\\
\mathcal{M}_b^{\mathbb{F}_q} &= \binom{k+b-1}{b}\left(\binom{n}{t} - m\binom{n-k-1}{t}\right).
\end{aligned}$$

The complexity of solving the system is

$$T(m,n,k,t) = \mathcal{O}\left(m^2 \mathcal{N}_b^{\mathbb{F}_q}\left(\mathcal{M}_b^{\mathbb{F}_q}\right)^{\omega-1}\right).$$

A relevant attack for our work, which is a hybrid strategy described in [3] that combines brute force with the mentioned linearization method, has complexity

$$\min_{a\geq 0}\left(q^{ta}\cdot T(m,n-a,k-a,t)\right). \tag{4}$$

3 Reduction Attacks

Layered ROLLO-I introduced a structure of layers. Due to the special structure, the designers highlighted a performance improvement by 30-70% compared to ROLLO-I at what they considered the same security level. In this section, we give a simple description of Layered ROLLO-I and its variants and then show that the layer can be removed by exploiting public information. As a result, the security of each algorithm is reduced to that of ROLLO-I for the small parameters inside the layer, which gives far lower complexity than was suggested in [8,12].

3.1 Layered ROLLO-I

The values (q,n,m,r,d,b,P) are the system parameters, where q,n,m,r,d,b are integers, with n a multiple of b, and $P(x) \in \mathbb{F}_q[x]$ is a primitive polynomial of degree n/b.

Remark 4. Note that the choice of $P(x)$ defined over $\mathbb{F}_q[x]$ instead of $\mathbb{F}_{q^m}[x]$ is essential to maintain the ideal LRPC structure of the inner ROLLO-I code, see [8, Section 2.3]. This is not made explicit in the Layered ROLLO-I submission, but was used in their implementation, which took the ROLLO-I implementation and added the additional structure without touching the definition of P.

For all parameter sets defined in [12] we have $b = 2$ and in any case $b < n/b$. Layered ROLLO-I mainly works in two different polynomial rings, the *small* one $\mathbb{F}_{q^m}[x]/(P(x))$ where every operation regarding the core ROLLO-I scheme is carried out, and the *large* one $\mathbb{F}_{q^m}[x]/(P(x)^b)$ where operation concerning the outer layer is performed. The intended communication parties face encoding and decoding in the small ring. An attacker is instead meant to deal with elements and operations in the larger ring which require a bigger computational effort.

Elements of the two polynomial rings are transformed using the two following maps. The map

$$\Psi : \mathbb{F}_{q^m}[x]/(P(x)) \to \mathbb{F}_{q^m}[x]/(P(x)^b)$$

lifts polynomials of the first quotient to the second quotient by mapping the input to the unique polynomial of degree $< n/b$ that is congruent to it modulo $P(x)^b$. Similarly, the map

$$\Omega : \mathbb{F}_{q^m}[x]/(P(x)^b) \to \mathbb{F}_{q^m}[x]/(P(x))$$

reduces the input modulo $P(x)$. Since $P(x)^b$ is a multiple of $P(x)$ these maps are well-defined.

We are now all set to describe the Layered-ROLLO-I KEM.

- KeyGen:
  - Pick random $\mathbf{x}, \mathbf{y} \in S_d^{n/b}(\mathbb{F}_{q^m})$.
  - Pick random invertible $P_I(x) \in \mathbb{F}_{q^m}[x]/(P(x))$ of degree $(b-1)$.
  - Pick random $P_O(x), P_N(x) \in \mathbb{F}_{q^m}[x]/(P(x)^b)$ of degree n, with $P_O(x)$ invertible (this last restriction is not stated but is required for functionality).
  - Set $\mathbf{z}(x) = P_I(x)\mathbf{x}(x)^{-1}\mathbf{y}(x) \bmod P(x)$.
  - Set $P_P(x) = P_O(x)\Psi(P_I(x)) \bmod P(x)^b$ and $P_H(x) = P_O(x)\Psi(\mathbf{z}(x)) + P_N(x)P(x) \bmod P(x)^b$.
  - Return $\mathsf{pk} = (P_P, P_H)$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y}, P_O, P_I)$.
- Encap(pk):
  - Pick random $E = \langle \mathbf{e}_1, \mathbf{e}_2 \rangle$ with $(\mathbf{e}_1, \mathbf{e}_2) \in S_r^{2n/b}(\mathbb{F}_{q^m})$, each corresponding to a polynomial of degree $< n/b - b$.
  - Set $P_{E_1}(x) = \Psi(\mathbf{e}_1(x))$ and $P_{E_2}(x) = \Psi(\mathbf{e}_2(x))$.
  - Compute $\mathbf{c}(x) = P_P(x)P_{E_1}(x) + P_H(x)P_{E_2}(x) \bmod P(x)^b$.
  - Return $K = \mathsf{hash}(E)$ and $\mathbf{c}$.
- Decap(sk):
  - Compute $P_C(x) = P_O(x)^{-1}\mathbf{c}(x) \bmod P(x)^b$.
  - Compute $\mathbf{c}'(x) = P_I(x)^{-1}\Omega(P_C(x)) \bmod P(x)$.

- Decode $E = \mathsf{RSR}(\langle \mathbf{x}, \mathbf{y} \rangle, \mathbf{x}\mathbf{c}', r)$.
- Return $K = \mathsf{hash}(E)$.

We want to stress that the attacker thus faces polynomials modulo $P(x)^b$ as public key and as ciphertexts. These correspond to vectors of length n. The decapsulation process removes the outer layer so that the RSR step works modulo P and thus on vectors of length n/b.

Reduction of Layered ROLLO-I to ROLLO-I. We propose a new reduction of Layered ROLLO-I to ROLLO-I by using exclusively the public key of the former. The goal is to remove the outer layer and expose the public key and ciphertext of the inner ROLLO-I scheme.

Remark 5. Notice that P_O must have an inverse modulo P^b. This has not been declared in the specification but the decapsulation process requires P_O^{-1}. If not, decapsulation fails. Also, P_I is irreducible of degree $(b-1) < n/b = \deg(P(x))$, so it has an inverse modulo P and thus $\Psi(P_I)$ is invertible modulo P^b.

```
Algorithm ReduceLayeredROLLO-I
Input : A public key $\mathsf{pk} = (P_P, P_H)$ and a ciphertext $\mathbf{c}$ of Layered ROLLO-I.
Output : A public key $\mathsf{pk}'$ and a ciphertext $\mathbf{c}'$ of the inner ROLLO-I.
  1. Compute $P_{P\mathsf{inv}}(x) = P_P(x)^{-1} \bmod P(x)^b$;
  2. Compute $\mathsf{pk}' = \Omega(P_{P\mathsf{inv}} P_H(x))$;
  3. Compute $\mathbf{c}' = \Omega(P_{P\mathsf{inv}} \mathbf{c}(x))$;
  4. Return $\mathsf{pk}'$ and $\mathbf{c}'$.
```

Proposition 6. *Given a public key and a ciphertext of Layered ROLLO-I, algorithm* ReduceLayeredROLLO-I *computes the public key and the corresponding ciphertext of the inner ROLLO-I scheme in time* $\mathcal{O}((n^2 m^2 \log_2(q))^2)$.

Proof. From Remark 5, we can invert $P_P(x)$ modulo $P(x)^b$. Hence Step 2 first computes

$$P_P(x)^{-1} P_H(x) = (\Psi(P_I(x)))^{-1} \Psi(\mathbf{z}(x)) + P_P(x)^{-1} P_N(x) P(x) + k(x) P(x)^b \quad (5)$$

for some $k(x) \in \mathbb{F}_{q^m}[x]$. Since P divides P^b, we can reduce the equation modulo P, obtaining

$$\begin{aligned} \Omega(P_P(x)^{-1} P_H(x)) &= \Omega((\Psi(P_I(x)))^{-1} \Psi(\mathbf{z}(x))) \\ &\equiv (P_I(x))^{-1} P_I(x) \mathbf{x}(x)^{-1} \mathbf{y}(x) \\ &\equiv \mathbf{x}(x)^{-1} \mathbf{y}(x) \bmod P(x), \end{aligned}$$

where the second equivalence follows from that $\Omega(\Psi(\mathbf{z}(x))) \equiv P_I(x)\mathbf{x}(x)^{-1}\mathbf{y}(x) \bmod P(x)$ and $\Omega((\Psi(P_I(x)))^{-1}) \equiv P_I(x)^{-1} \bmod P(x)$. This shows that the public key of (q, n, m, r, d, b)-Layered ROLLO-I can be mapped efficiently to the public key

of $(q, n/b, m, r, d)$-ROLLO-I, validating step 2.
The same can be done for ciphertexts in step 3, by computing

$$\begin{aligned} P_P(x)^{-1}\mathbf{c}(x) &\equiv P_P(x)^{-1}(P_P(x)P_{E_1}(x) + P_H(x)P_{E_2}(x)) \\ &\equiv \mathbf{e}_1(x) + \mathbf{y}(x)\mathbf{x}(x)^{-1}\mathbf{e}_2(x) \bmod P(x), \end{aligned}$$

which is exactly a ROLLO-I ciphertext.

The complexity of the algorithm is dominated by that of inverting and multiplying polynomials in $\mathbb{F}_{q^m}[x]$ which can be performed in $\mathcal{O}((n^2m^2\log_2(q))^2)$ taking schoolbook multiplication in $\mathbb{F}_{q^m}$ and $\mathbb{F}_{q^m}[x]$.

$\square$

Therefore, it is possible to reduce an entire instance of (q, n, m, r, d, b)-Layered ROLLO-I to an instance of $(q, n/b, m, r, d)$-ROLLO-I.

Estimates for the security of Layered ROLLO-I. Layered ROLLO-I [8,12] considers an attack that removes the layer of a BII-LRPC code using exhaustive search and applies a structural attack to an instance of $(q, n/b, m, r, d)$-ROLLO-I. Their estimated cost of the attack is shown in the third column of Table 1. However, the calculation is wrong and furthermore, the formula given in [12] has a typo. The correct formula for the attack is given below.

$$S'_S = \left(\frac{n}{b}\right)^3 m^3 q^{(b-1)m+d\lceil\frac{m}{2}\rceil-m-\frac{n}{b}}. \tag{6}$$

While the correct formula in (6) increases attack complexity when compared to the suggested one in [12], the accurate computation yields significantly lower complexity, which are 65, 112, and 131 bits, respectively.

As we mentioned in Section 2.1, we consider the attacks in [3,4,2], where we discard the options in [4] that have been proved too optimistic in [3]. For the parameters ranges selected for Layered ROLLO-I, the most efficient attacks have complexities as in (3) and (4). For a concise overview of the latest developments in rank decoding attacks, we refer the reader to [14, Section 6]. Since Layered ROLLO-I did not consider applying these three attacks on the original parameters directly, we recompute the costs of rank decoding attacks, finding out that the proposed parameters did not meet the requirements for the claimed security levels. The complexities of the attacks are computed using the script provided in [13], the Sage script performs puncturing of the public code to find the optimal complexity. The costs of the most efficient among these attacks are reported in the fourth column of Table 1. The last column reports the cost of these attacks on the system after our reduction.

Security	(q, n, m, r, d, b)	Cost [12]	Cost	Cost red.
128	$(2, 148, 67, 3, 2, 2)$	130.83	48.76 (3)	40.65 (3)
192	$(2, 172, 79, 4, 3, 2)$	199.19	66.21 (3)	55.16 (3)
256	$(2, 212, 97, 5, 3, 2)$	274.98	85.68 (3)	72.05 (3)

Table 1. Suggested parameters and values of the $\log_2$ of attack costs for Layered ROLLO-I's suggested parameters. Cost [12] refers to the cost stated in the paper introducing the system; references after the costs refer to the complexity formula which gives the best value on these parameters.

3.2 First Modified Layered ROLLO-I (MLR1)

This subsection extracts the description of the modified system MLR1 from [9]. The designers modified the system to overcome the reduction in Section 3.1 by replacing the two moduli P and P^b by two primitive polynomials P_1 and P_2 of degree n_1 and n_2, respectively. Because they are primitive they are in particular irreducible and thus coprime. In this setting, one cannot simply reduce equation (5) modulo P_1 as the term $k(x)P_2(x)$ would not vanish which seems to stop the attack.

Remark 7. For the same reasons pointed out in Remark 4 one needs P_1 to be defined over the ring $\mathbb{F}_q[x]$.

In this setting, Ω first lifts to $\mathbb{F}_{q^m}[x]$ choosing the unique polynomial of degree less than n_2 and then reduces modulo P_1, Ψ similarly lifts to $\mathbb{F}_{q^m}[x]$ choosing the unique polynomial of degree less than n_1 and then considers this polynomial modulo P_2. Given that $n_2 > n_1$ no reduction is needed.

The values $(q, n_1, n_2, d_I, m, r, d)$, where $d_I < n_1 < n_2$ are the system parameters. The two polynomials P_1 and P_2 are primitive of degrees n_1 and n_2 respectively. These are not stated among the system parameters but are needed for the functioning of the system. In the following, we assume that P_1 and P_2 are part of the system parameters.

- KeyGen:
  - Pick random $\mathbf{x}, \mathbf{y} \in S_d^{n_1}(\mathbb{F}_{q^m})$.
  - Pick random invertible $P_I(x) \in \mathbb{F}_{q^m}[x]/(P_1(x))$ of degree d_I.
  - Pick random $P_O(x) \in \mathbb{F}_{q^m}[x]/(P_2(x))$.
  - Set $\mathbf{z}(x) = P_I(x)\mathbf{x}(x)^{-1}\mathbf{y}(x) \bmod P_1(x)$.
  - Set $P_P(x) = P_O(x)\Psi(P_I(x)) \bmod P_2(x)$ and $P_H(x) = P_O(x)\Psi(\mathbf{z}(x)) \bmod P_2(x)$.
  - Return $\mathsf{pk} = (P_P, P_H)$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y}, P_O, P_I)$.

- Encap(pk):
  - Pick random $E = \langle \mathbf{e}_1, \mathbf{e}_2 \rangle$ with $(\mathbf{e}_1, \mathbf{e}_2) \in S_r^{2n_2}(\mathbb{F}_{q^m})$ each corresponding to a polynomial of degree $< n_2 - n_1 - d_I$.
  - Set $P_{E_1} = \mathbf{e}_1(x)$ and $P_{E_2} = \mathbf{e}_2(x)$.
  - Compute $\mathbf{c}(x) = P_P(x)P_{E_1}(x) + P_H(x)P_{E_2}(x) \bmod P_2(x)$.

  - Return $K = \mathsf{hash}(E)$ and $\mathbf{c}$.
- Decap(sk):
  - Compute $\mathbf{c}''(x) = P_O(x)^{-1}\mathbf{c}(x) \bmod P_2(x)$.
  - Compute $\mathbf{c}'(x) = P_I(x)^{-1}\Omega(\mathbf{c}''(x)) \bmod P_1(x)$.
  - Decode $E = \mathsf{RSR}(\langle \mathbf{x}, \mathbf{y}\rangle, \mathbf{x}\mathbf{c}', r)$.
  - Return $K = \mathsf{hash}(E)$.

Note that all polynomials are invertible modulo $P_1(x)$ and modulo $P_2(x)$ because those are irreducible.

Remark 8. Note also that the coprime moduli and choosing $\mathbf{e}_1, \mathbf{e}_2 \in S_r^{n_2}(\mathbb{F}_{q^m})$ (rather than in $S_r^{n_1}(\mathbb{F}_{q^m})$ as required in the RSR computation) might make it seem like decapsulation cannot recover E. The KEM works around this problem by reducing the degrees of $\mathbf{e}_1(x)$ and $\mathbf{e}_2(x)$ to $< n_2 - n_1 - d_I$ which by Table 2 is $< n_1$ for all chosen parameters, so that $\Omega(P_{E_i}(x)) = P_{E_i}(x)$. This property is essential for decapsulation to work.

Decapsulation works because

$$\begin{aligned}
\mathbf{c}''(x) &= P_O(x)^{-1}(P_P(x)P_{E_1}(x) + P_H(x)P_{E_2}(x)) \\
&= P_O(x)^{-1}(P_O(x)\Psi(P_I(x))P_{E_1}(x) + P_O(x)\Psi(\mathbf{z}(x))P_{E_2}(x)) \\
&= \Psi(P_I(x))P_{E_1}(x) + \Psi(\mathbf{z}(x))P_{E2}(x) \bmod P_2(x)
\end{aligned}$$

and the degree of $\Psi(P_I(x))P_{E_1}(x) + \Psi(\mathbf{z}(x))P_{E_2}(x)$ is $< n_2$ by the choice of the error vectors. Hence, $\mathbf{c}''(x) = \Psi(P_I(x))P_{E_1}(x) + \Psi(\mathbf{z}(x)P_{E2}(x)$ in $\mathbb{F}_{q^m}[x]$ i.e., without reduction, and thus the reduction modulo $P_1(x)$ preserves the factors $P_I(x)$ which can then be divided out. Finally, by Remark 8, $\Omega(P_{E_i}(x)) = P_{E_i}(x)$.

Reduction of $\mathsf{MLR1}$ to ROLLO-I. We will describe a reduction of $\mathsf{MLR1}$. Along the way we compute P_I and P_O, meaning that the system leaks parts of the private key.

The idea of the reduction remains the same, observing that $P_H(x)/P_P(x)$ cancels the term P_O. However, because of the coprimality of the moduli, we cannot proceed directly from there to reducing modulo P_1. Nevertheless, we know that the polynomials involved have very low degrees which will provide a set of linear equations which will allow to compute $\Psi(P_I)$ and $\Psi(\mathbf{z}(x))$.

```
Algorithm ReduceMLR1
Input : A public key $\mathsf{pk} = (P_P, P_H)$ and a ciphertext $\mathbf{c}$ of MLR1.
Output : A public key $\mathsf{pk}'$ and a ciphertext $\mathbf{c}'$ of the inner ROLLO-I.
    1. Compute $R(x) = P_H(x)/P_P(x) \bmod P_2(x)$;
    2. Compute matrix $M_{R,P_2}$;
    3. Put $M_R = M_{R,P_2}[1 : d_I + 1, n_1 + 1 : n_2]$;
    4. Solve $\mathbf{v}M_R = \mathbf{0}$ for nonzero $\mathbf{v}$;
    5. Compute $P_O'(x) = P_P(x)\mathbf{v}(x)^{-1} \bmod P_2(x)$;
    6. Compute $\mathbf{z}'(x) = P_H(x)P_O'(x)^{-1} \bmod P_2(x)$;
```

7. Compute $\mathsf{pk}' = \Omega(\mathbf{z}'(x)\mathbf{v}(x)^{-1} \bmod P_2(x))$;
8. Compute $\mathbf{c}' = \Omega(\mathbf{c}(x)P'_O(x)^{-1} \bmod P_2(x))\mathbf{v}(x)^{-1} \bmod P_1(x)$;
9. Return pk' and $\mathbf{c}'$.

Step 4 is computed by taking a $(d_I + 1) \times d_I$ submatrix $M_R[:, J]$ of M_R, with $\#J = d_I$, because the left kernel of that is typically no larger than that of M_R. If the kernel has dimension larger than 1, more columns of M_R are included.

Our experiments have not encountered a case where $\mathsf{rk}(M_R) < d_I$.

Proposition 9. *Given a public key and a ciphertext of* MLR1*, algorithm* ReduceMLR1 *takes time* $\mathcal{O}(d_I n_2^2 m^2 (\log(q))^2)$ *and outputs polynomials* pk' *and* $\mathbf{c}'$ *of degree* $< n_1$ *satisfying* $\mathbf{c}' = \mathbf{e}_1 + \mathbf{e}_2\mathsf{pk}'$ *for some* $\mathbf{e}_1, \mathbf{e}_2$*.*

If $\mathsf{rk}(M_R) = d_I$ *these are the public key and the corresponding ciphertext of the inner ROLLO-I scheme.*

Proof. Since $R(x)$ is computed modulo $P_2(x)$ we have $\deg(R(x)) < n_2$ and $R(x) = \Psi(\mathbf{z}(x))/\Psi(P_I(x)) \bmod P_2(x)$ where $\deg(\mathbf{z}(x)) < n_1$ and $\deg(P_I(x)) = d_I$ are small. Note that the division might cancel common factors of P_I and $\mathbf{z}$, however, given the degrees this is unlikely.

The polynomial P_2 is irreducible and thus $\mathbb{F}_{q^m}[x]/(P_2(x))$ defines a field. Note that multiplication by $R \neq 0$ defines a bijective map, hence the associated $n_2 \times n_2$ matrix M_{R,P_2} has full rank n_2. Also, note that any submatrix of M_{R,P_2} consisting of a subset of the rows of M_{R,P_2} must have full rank. In particular, $M_{d_I+1} = M_{R,P_2}[1 : d_I + 1, :]$, it has rank $d_I + 1$. Let

$$\pi : \mathbb{F}_{q^m}^{n_2} \to \mathbb{F}_{q^m}^{d_I+1}$$
$$(v_1, \ldots, v_n) \mapsto (v_1, \ldots, v_{d_I+1})$$

be the projection of an element of $\mathbb{F}_{q^m}^{n_2}$ onto its first $d_I + 1$ coordinates. Consider

$$\pi(\Psi(P_I(x)))M_{d_I+1} = \Psi(\mathbf{z}(x)) \tag{7}$$

as a linear system of equations in the coefficients of $\Psi(P_I)$ and $\Psi(\mathbf{z})$, where in this case we view $\Psi(\mathbf{z})$ as an element of $\mathbb{F}_{q^m}^{n_2}$ consisting of the unknown coefficients of $\Psi(\mathbf{z})$ and $n_2 - n_1$ trailing zeroes as a result of its degree. Note that π does not induce any loss of information due to the degree of $\Psi(P_I)$. Since $\deg(\Psi(P_I(x))) + n_1 = d_I + n_1 < n_2$, the system has a solution corresponding to the representatives of P_I and $\mathbf{z}$ modulo P_1 (here we remove the Ψ notation as the solutions will have degree lower than n_1).

We can actually compute P_I from a subset of the equations defined by (7). Thanks to the presence of $n_2 - n_1$ trailing zeroes in $\Psi(\mathbf{z})$, we know that $\pi(\Psi(P_I))$ lies in the left kernel of $M_R = M_{R,P_2}[1 : d_i + 1, n_1 + 1 : n_2]$. Hence, that kernel is non-trivial and the submatrix has rank at most d_I.

The algorithm proceeds to compute $\mathbf{v}$ satisfying $\mathbf{v}M_R = \mathbf{0}$ where from (7) we know that P_I by satisfies

$$\pi(\Psi(P_I(x)))M_R = \mathbf{0}. \tag{8}$$

If $\mathsf{rk}(M_R) = d_I$ the kernel is one-dimensional and $\mathbf{v}(x) = \lambda\Psi(P_I(x))$ for some constant $\lambda \in \mathbb{F}_{q^m}$. We will now show that it is not a problem that we can recover P_I only up to such a constant factor. Step 6 computes $P'_O(x) = P_P(x)/\mathbf{v}(x) = P_O(x)/\lambda$ and step 7 then computes

$$\mathbf{z}'(x) = P_H(x)/P'_O(x) = \lambda\Psi(P_I(x)\mathbf{x}(x)^{-1}\mathbf{y}(x)),$$

and finally $\Omega(\mathbf{z}'(x)/\mathbf{v}(x)) = \mathbf{x}(x)^{-1}\mathbf{y}(x)$ which corresponds to a ROLLO-I public key.

Similarly, for the ciphertext, we compute

$$\mathbf{c}(x)/P'_O(x) = \lambda\Psi(P_I(x))P_{E_1}(x) + \lambda\Psi(\mathbf{z}(x))P_{E_2}(x) \bmod P_2(x).$$

Since λ is constant, the degree of the right-hand side is below n_2, so we can reduce modulo P_1. Given the degrees, $\mathbf{v}(x) = \lambda P_I(x)$ (without Ψ) and Step 8 divides by $\lambda P_I(x)$ to get $P_{E_1}(x) + \mathbf{x}(x)^{-1}\mathbf{y}(x)P_{E_2}(x)$, matching the ROLLO-I ciphertexts, where we use that $\Omega(P_{E_i}(x)) = P_{E_i}(x)$, see Remark 8.

If $\mathsf{rk}(M_R) < d_I$ the polynomials still satisfy the same equations but there is no guarantee that the choice of $\mathbf{v}$ leads to a $\mathbf{z}$ that is the ratio of low-rank vectors, so it need not be a valid public key.

The complexity of the algorithm is dominated by that of inverting and multiplying polynomials of degree bounded by n_2 in $\mathbb{F}_{q^m}[x]$, which can be performed in $\mathcal{O}((n^2m^2\log_2(q))^2)$ taking schoolbook multiplication in $\mathbb{F}_{q^m}$ and $\mathbb{F}_{q^m}[x]$, and by computing the left kernel of the $(d_I+1)\times n_2$ matrix M_R over $\mathbb{F}_{q^m}$ which can be done in $\mathcal{O}(d_I^2 n_2 m^2(\log(q))^2)$ using schoolbook multiplication. From $d_I < n_2$ the complexity is in $\mathcal{O}(d_I n_2^2 m^2(\log(q))^2)$. □

Our experiments show that the time to compute $\mathbf{v}$ is split roughly equally between the costs of polynomial division modulo P_2 to obtain R on the one side and the costs of computing the matrix M_R and computing the left kernel of $M_R[:, J]$ on the other side, where the choice of J as the last d_I columns of M_R typically succeeds.

Estimates for the security of MLR1. The proposed parameters for MLR1 along with the claimed attack costs are displayed in Table 2. The complexities of the attacks are again computed using the script provided in [13]. For each security level, the costs of the attacks on the proposed parameters are shown in the third column of Table 2, and those of reduced Layered ROLLO-I along are in the fourth column of Table 2. The time in seconds to compute the public key transformation is described in Algorithm ReduceMLR1 using SageMath on a Linux Mint virtual machine, is stated in the fifth column of Table 2. It is worth highlighting that the third column of Table 2 shows that even without our reduction, the security is still lower for these parameters than the targeted security levels, even though the designers were now aware of the attacks in [4].

Note that here we use P_I with $\deg(P_I(x)) = d_I$ as stated in [9]. The parameters file in the implementation package instead uses $\deg(P_I(x)) = 4$ for all security levels.

Security	$(q, n_1, n_2, d_I, m, r, d)$	Cost	Cost red.	Time (s)
128	$(2, 37, 61, 11, 67, 6, 2)$	103.83 (4)	96.95 (4)	1.85
192	$(2, 43, 71, 15, 79, 7, 3)$	185.52 (2)	156.16 (4)	2.42
256	$(2, 53, 103, 20, 97, 7, 3)$	187.91 (4)	151.11 (4)	4.21

Table 2. Values of the $\log_2$ of attack costs for MLR1's suggested parameters, before and after our reduction, and time consumed by the reduction. References after the costs refer to the complexity formula which gives the best value on these parameters.

4 Message Recovery Attacks

We describe the two message recovery attacks that we mounted against Layered ROLLO-I. The first one breaks all the versions described so far and MLR2 [10] (see Section 4.1). The second one applies to security levels 128 and 192 of MLR3 [10] (see Section 4.3). The idea is to reduce the modular equation in the encapsulation to a system of linear equations and exploit the knowledge of zero positions of the error vectors to solve the system.

Both of the attacks take advantage of the low degree of the polynomials P_{E_i} for $i = 1, 2$ to build an overdetermined system of linear equations, which can be easily solved for the unknowns $\mathbf{e}_i$ and thus recover the message.

In the following, we first describe another modification MLR2 the designers made which changes the structure of the public key to counter the attacks described in the previous section. This and both previous versions use the same equation for encapsulation, albeit with different constraints on the degrees of the P_{E_i}. The message recovery attack works solely with this equation and thus applies to all these versions, hence we put the attack after the description of MLR2 to demonstrate the range of applicability, but want to stress that it applies already to the journal version [12] and is not a byproduct of the designers' patches. Finally, we describe and then attack a third modified version MLR3 which the designers posted.

4.1 Second Modified Layered ROLLO-I (MLR2)

In this subsection, we describe the system from [10]. The new version of Layered ROLLO-I, which we denote by MLR2, uses polynomial masking techniques in order to avoid the reduction to ROLLO-I described in Section 3.2. To this end, the new system patch introduces an auxiliary polynomial P_N of small degree and modifies the P_P-part of the public key.

The values $(q, n_1, n_2, n_I, m, r, d)$, where $n_I < n_1 < n_2$ are the system parameters. There is also a primitive polynomial P_2 of degree n_2 which is a system parameter. We will report here only the key generation procedure, as the rest is the same as for $\mathsf{MLR1}$ except for the degree of the error polynomials. The key generation procedure of the new system works as follows.

– KeyGen:
  - Pick random $\mathbf{x}, \mathbf{y} \in S_d^{n_1}(\mathbb{F}_{q^m})$.
  - Pick random primitive $P_1(x) \in \mathbb{F}_q[x]$ of degree n_1.
  - Pick random $P_I(x) \in \mathbb{F}_{q^m}[x]/(P_1(x))$ of degree n_I.
  - Pick random $P_O(x), P_N(x) \in \mathbb{F}_{q^m}[x]/(P_2(x))$, with $\deg(P_N(x)) = n_N$.
  - Set $\mathbf{z}(x) = P_I(x)\mathbf{x}(x)^{-1}\mathbf{y}(x) \bmod P_1(x)$.
  - Set $P_P(x) = P_O(x)(\Psi(P_I(x)) + P_N(x)P_1(x)) \bmod P_2(x)$ and $P_H(x) = P_O(x)\Psi(\mathbf{z}(x)) \bmod P_2(x)$.
  - Return $\mathsf{pk} = (P_P, P_H)$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y}, P_O, P_I, P_1)$.

The encapsulation mechanism with updated error weights is equivalent to that of $\mathsf{MLR1}$ except that the random vectors $\mathbf{e}_1, \mathbf{e}_2$ should each correspond to a polynomial of degree $n_E < n_2 - n_1 - n_I - n_N$.

4.2 Message recovery attack on all versions described so far

We first give the algorithm of our attack and then show its correctness.

Recall that the public key is $\mathsf{pk} = (P_P, P_H)$ and the degree of the error polynomials P_{E_i} is limited to n_E to permit decapsulation. For all versions of Layered ROLLO-I, encapsulation computes the ciphertext as

$$\mathbf{c}(x) = P_{E_1}(x)P_P(x) + P_{E_2}(x)P_H(x) \bmod P_2(x),$$

where we put $P_2 = P^b$ for the first version to unify notation. It is the very small limit n_E on the degree of the error polynomials that makes this attack work.

```
Algorithm MsgRecovery
Input : A public key $\mathsf{pk} = (P_P, P_H)$ and a ciphertext $\mathbf{c}$ of Layered-ROLLO-I,
    MLR1, or MLR2.
Output : The shared secret $K$ corresponding to $\mathbf{c}$.
    1. Compute $\bar{\mathbf{c}} = \mathbf{c}(x)P_H(x)^{-1} \bmod P_2(x)$;
    2. Compute $R(x) = P_P(x)P_H(x)^{-1} \bmod P_2(x)$;
    3. Compute matrix $M_R = M_{R,P_2}[1 : n_E + 1, :]$;
    4. Pick random subset $J \subset [n_E + 2, n_2]$ with $\#J = n_E + 1$;
    5. Put $M_{R\mathsf{inv}} = M_R[:, J]$;
    6. If $\mathsf{rk}(M_{R\mathsf{inv}}) < n_E + 1$, go to step 4;
    7. Solve $\bar{\mathbf{e}}_1 = \bar{\mathbf{c}}[J]M_{R\mathsf{inv}}^{-1}$ for $\bar{\mathbf{e}}_1$;
    8. Compute $\bar{\mathbf{e}}_2 = \bar{\mathbf{c}}[1 : n_E + 1] - \bar{\mathbf{e}}_1 M_R[:, 1 : n_E + 1]$;
    9. Put $\mathbf{e}_1 = \bar{\mathbf{e}}_1||(0, 0, \ldots, 0)$ and $\mathbf{e}_2 = \bar{\mathbf{e}}_2||(0, 0, \ldots, 0)$;
    10. Compute $K = \mathsf{hash}(\langle \mathbf{e}_1, \mathbf{e}_2 \rangle)$;
    11. Return $K$.
```

Proposition 10. *Given a public key* pk *and a ciphertext* $\mathbf{c}$ *of Layered-ROLLO-I,* MLR1*, or* MLR2*, algorithm* MsgRecovery *outputs the shared secret* K *encapsulated in* $\mathbf{c}$ *to* pk*.*

Proof. If P_2 is irreducible then P_H is invertible modulo P_2. In Layered-ROLLO-I $\mathbf{z}(x)$ is invertible modulo the irreducible P_1 and thus also modulo $P_2 = P_1^b$, and, as stated earlier, P_O is required to be invertible modulo P_2 to permit decapsulation. Hence, P_H is invertible in all variants.

Step 1 computes

$$\bar{\mathbf{c}}(x) = \mathbf{c}(x)P_H(x)^{-1} = P_{E_1}(x)R(x) + P_{E_2}(x) \bmod P_2(x), \tag{9}$$

where $R(x) = P_P(x)P_H(x)^{-1} \bmod P_2(x)$ by step 2.

View equation (9) in terms of $\mathbb{F}_{q^m}$ vectors corresponding to the coefficient vectors of the polynomials involved. The $(n_E + 1) \times n_2$ full rank matrix M_R computed in step 3, represents the multiplication of a polynomial of degree up to n_E by R modulo P_2, defined as in (1). In other words, M_R generates a linear $[n_2, n_E + 1]$-code over $\mathbb{F}_{q^m}$.

With this in mind we can rewrite (9) as

$$\bar{\mathbf{c}} = \bar{\mathbf{e}}_1 M_R + \mathbf{e}_2, \tag{10}$$

which corresponds to a McEliece-like encryption of the "message" $\bar{\mathbf{e}}_1 \in \mathbb{F}_{q^m}^{n_E+1}$ using $\mathbf{e}_2$ as error vector. The last $n_2 - n_E - 1$ positions of $\mathbf{e}_2$ are 0 so that

$$\bar{\mathbf{c}}[n_E + 2, n_2] = \bar{\mathbf{e}}_1 M_R[:, n_E + 2, n_2] \tag{11}$$

holds exactly.

Assume that the choice of J in step 4 is such that $\mathsf{rk}(M_{R\mathsf{inv}}) = n_E + 1$, i.e., that $M_{r\mathsf{inv}}$ is invertible. Then step 7 computes a length-$n_E + 1$ vector $\bar{\mathbf{e}}_1$ satisfying (11).

Finally, step 8 computes $\bar{\mathbf{e}}_2 = \bar{\mathbf{c}}[1 : n_E + 1] - \bar{\mathbf{e}}_1 M_R[:, 1 : n_E + 1]$ from the first $n_E + 1$ positions of (10), and step 9 extends both vectors to $\mathbf{e}_1$ and $\mathbf{e}_2$ by appending 0s.

To finish the proof we consider the assumption that $\mathsf{rk}(M_{r\mathsf{inv}}) = n_E + 1$. The matrix $M_{R\mathsf{inv}} = M_R[:, J]$ for J a random subset of $n_E + 1$ of the last $n_2 - n_E - 1$ columns of M_R is a square $(n_E + 1) \times (n_E + 1)$ submatrix of M_R which is a submatrix of M_{R,P_2}. Polynomial R computed in step 2 is a ratio of invertible polynomials modulo P_2 which have no special structure and thus the matrix M_{R,P_2} can be considered random, apart from having rank n_2 as R is invertible modulo P_2. Similarly, the matrix M_R consists of the first $n_E + 1$ rows of M_{R,P_2} and has thus rank $n_E + 1$, but apart from that can be considered random. Hence, the chance that $M_{R\mathsf{inv}}$ is invertible is no lower than that of a random $(n_E + 1) \times (n_E + 1)$ matrix over $\mathbb{F}_{q^m}$ which is

$$q^{-m(n_E+1)^2} \prod_{i=0}^{n_E} (q^{m(n_E+1)} - q^{im}). \tag{12}$$

For all parameter sets of all variants of Layered-ROLLO-I, this probability rounds to 1 because the field size q^m is large. In the unlikely case that the first choice of J does not succeed, step 6 catches the exception. □

Due to the high probability of finding an invertible matrix our implementation just takes J as the last $n_E + 1$ columns which always succeeded.

Remark 11. At first sight one could want to employ Prange's algorithm [17] as a black box to simultaneously recover $\mathbf{e}_1$ and $\mathbf{e}_2$ from (10). This also would succeed, because the bound on the degree of the polynomial P_{E_2} implies that the Hamming weight of $\mathbf{e}_2$ is bounded by $n_E + 1$ which is much smaller than n_2.

However, the core of Prange's algorithm lies in finding an invertible submatrix of M_R that consists of a subset of columns corresponding to error-free positions in the ciphertext, taking time proportional to

$$\binom{n_2}{n_E+1} \Big/ \binom{n_2 - n_E - 1}{n_E + 1},$$

while for all versions of Layered-ROLLO-I we know that the last $n_2 - n_E - 1$ positions are error-free, thus the attack avoids the most costly part of Prange's algorithm.

Complexity of MsgRecovery. The most time-consuming steps of our attack are the polynomial divisions in steps 1 and 2 and computing the matrix M_R and $M_{R\mathsf{inv}}^{-1}$. Using only schoolbook multiplication the complexity is $\mathcal{O}(n_E n_2^2 m^2 \log_2(q)^2)$.

Steps 1 − 6 and computing $M_{R\mathsf{inv}}^{-1}$ are independent of the ciphertext. If an attacker breaks many ciphertexts for the same public key, this computation is done only once requiring only $\mathcal{O}(n_E^3 m^2 \log_2(q)^2)$ per ciphertext.

We implemented this attack in SageMath. An average of the time required, on a Linux Mint virtual machine, to recover the plaintext for the proposed parameters of MLR2 is given in Table 3. It is worth noting that in our experiments we always used error vectors of the maximum allowed Hamming weight $n_E + 1$ in order to simulate the worst-case scenario for our attack.

Security	$(q, m, n_I, n_1, n_2, n_N)$	n_E	Time (s)
128	$(2, 67, 4, 37, 61, 1)$	18	2.21
192	$(2, 79, 4, 43, 71, 2)$	21	3.18
256	$(2, 97, 4, 53, 103, 4)$	41	6.65

Table 3. Parameter sets for MLR2 and average time in seconds (on 50 samples for each security level) needed to recover a plaintext.

Remark 12. We would like to remark that this message recovery attack works for all three versions of Layered ROLLO-I presented to so far since the degrees of $\mathbf{e}_1$ and $\mathbf{e}_2$ are smaller than half of n_2, which is relevant for the positions in $M_{R\mathsf{inv}}$ not to overlap with the positions in $\mathbf{e}_2$.

4.3 Third Modified Layered ROLLO-I (MLR3)

In this subsection, we describe the system from [11]. MLR3 uses polynomial masking in the ciphertext to overcome the message recovery attack that we described in the previous subsection. We will only display the parts in the specification of KeyGen and Encap that differ from that of MLR2. The values $(q, n_1, n_2, n_I, n_A, m, r, d)$, where $n_I = n_1 < n_2$ and $n_A = 4$ are the system parameters. The updates to the key generation procedure of the new system are as follows.

- KeyGen:
  - Pick random $P_{N,A}(x), P_{N,B}(x) \in \mathbb{F}_{q^m}[x]/(P_2(x))$ of degree n_A
  - Set
    $P_P(x) = P_O(x)(\Psi(P_I(x)) + P_{N,A}(x)P_1(x)) \bmod P_2(x)$,
    $P_H(x) = P_O(x)\Psi(\mathbf{z}(x)) \bmod P_2(x)$, and
    $P_B(x) = P_O(x)P_{N,B}(x)P_1(x) \bmod P_2(x)$.
  - Return $\mathsf{pk} = (P_P, P_H, P_B)$ and $\mathsf{sk} = (\mathbf{x}, \mathbf{y}, P_O, P_I, P_1)$.

The updates to the encapsulation mechanism with updated error weights are as follows.

- Encap(pk):
  - Compute
    $\mathbf{c}(x) = P_P(x)P_{E_1}(x) + P_H(x)P_{E_2}(x) + P_B(x)P_{N,C}(x) \bmod P_2(x)$,

where P_{E_1}, P_{E_2} and $P_{N,C}$ have degree $n_E < n_2 - n_1 - n_A$. The decapsulation procedure has not been updated. It still works because

$$\begin{aligned}\mathbf{c}''(x) &= P_O(x)^{-1}(P_P(x)P_{E_1}(x) + P_H(x)P_{E_2}(x) + P_B(x)P_{N,C}(x)) \bmod P_2(x)\\ &= (\Psi(P_I(x)) + P_{N,A}(x)P_1(x))P_{E_1}(x) + \Psi(\mathbf{z}(x))P_{E_2}(x) + P_{N,B}P_1P_{N,C}(x)\end{aligned}$$

and the degree of $\Psi(P_I(x))P_{E_1}(x)+\Psi(\mathbf{z}(x))P_{E_2}(x)+P_{N,B}P_1P_{N,C}(x)$ is $< n_2$ by the choice of the error vectors. Hence, the reduction modulo $P_1(x)$ removes the last term and preserves the factors $P_I(x)$ which can then be divided out. Finally, for all parameter sets $n_E + 1 < n_1$ and thus $\Omega(P_{E_i}(x)) = P_{E_i}(x)$.

Message recovery attack on MLR3. We describe a fast message recovery attack on the security levels 128 and 192 of MLR3, that uses only linear algebra. Let $\ell = n_2 - n_1 - n_A \geq n_E + 1$ denote the maximum length of the non-zero coefficients in the error vectors.

Remark 13. The following attack will set up several systems of equations. For the parameters of any security level, Table 4 shows that we always have $3(n-\ell) > n_2$ where there exist at most n_2 linearly independent equations. For levels 128 and 192, we have $3\ell < n_2$ ensuring a unique solution of the system, which is not the case for security level 256.

Compute the polynomials

$$\begin{aligned}A_1(x) &= P_P(x)P_B^{-1}(x),\ B_1(x) = P_H(x)P_B^{-1}(x),\\ A_2(x) &= P_P(x)P_H^{-1}(x),\ C_2(x) = P_B(x)P_H^{-1}(x),\ \text{and}\\ B_3(x) &= P_H(x)P_P^{-1}(x),\ C_3(x) = P_B(x)P_P^{-1}(x),\end{aligned} \tag{13}$$

and let $M_{A_1}, M_{B_1}, M_{A_2}, M_{C_2}, M_{B_3}$ and M_{C_3} be the corresponding matrices representing multiplications modulo $P_2(x)$, as in (1). Set

$$\begin{aligned}\mathbf{c}_1(x) &= \mathbf{c}(x)P_B^{-1}(x) \bmod P_2(x),\\ \mathbf{c}_2(x) &= \mathbf{c}(x)P_H^{-1}(x) \bmod P_2(x),\ \text{and}\\ \mathbf{c}_3(x) &= \mathbf{c}(x)P_P^{-1}(x) \bmod P_2(x).\end{aligned} \tag{14}$$

From these values, we derive the following equations

$$\begin{aligned}\mathbf{c}_1 &= \mathbf{e}_1 M_{A_1} + \mathbf{e}_2 M_{B_1} + \mathbf{p},\\ \mathbf{c}_2 &= \mathbf{e}_1 M_{A_2} + \mathbf{e}_2 + \mathbf{p}M_{C_2},\ \text{and}\\ \mathbf{c}_3 &= \mathbf{e}_1 + \mathbf{e}_2 M_{B_3} + \mathbf{p}M_{C_3},\end{aligned} \tag{15}$$

where we denote the coefficient vector of $P_{N,C}$ by $\mathbf{p}$.

A first key observation is that, if we restrict to the last $n_2 - \ell$ columns of each matrix, corresponding to the terms of degree $\geq \ell$ we obtain

$$\begin{aligned}\mathbf{c}_1[:, \ell : n_2] &= \mathbf{e}_1 M_{A_1}[:, \ell : n_2] + \mathbf{e}_2 M_{B_1}[:, \ell : n_2]\\ \mathbf{c}_2[:, \ell : n_2] &= \mathbf{e}_1 M_{A_2}[:, \ell : n_2] + \mathbf{p}M_{C_2}[:, \ell : n_2]\\ \mathbf{c}_3[:, \ell : n_2] &= \mathbf{e}_2 M_{B_3}[:, \ell : n_2] + \mathbf{p}M_{C_3}[:, \ell : n_2],\end{aligned} \tag{16}$$

getting rid of one of the terms in each equation. A second key observation is that, thanks to the size of the field $\mathbb{F}_{q^m}$ we can find three sets $S_1, S_2, S_3 \subset [\ell + 1, n_2]$ of cardinality ℓ such that $\overline{M}_{A_1} = M_{A_1}[:, S_1], \overline{M}_{B_1} = M_{B_1}[:, S_1], \overline{M}_{A_2} = M_{A_2}[:, S_2], \overline{M}_{C_2} = M_{C_2}[:, S_2], \overline{M}_{B_3} = M_{B_3}[:, S_3], \overline{M}_{C_3} = M_{C_3}[:, S_3]$ are all invertible $\ell \times \ell$ matrices. This happens probability ~ 1 for security levels 128 and 192, see (12). Denote by $\overline{\mathbf{c}}_1, \overline{\mathbf{c}}_2$ and $\overline{\mathbf{c}}_3$ the subvectors of $\mathbf{c}_1, \mathbf{c}_2$ and $\mathbf{c}_3$ consisting of entries indexed by S_1, S_2 and S_3, respectively. Replacing into (16) we have that the equalities still hold, and write

$$\overline{\mathbf{c}}_1 = \mathbf{e}_1 \overline{M}_{A_1} + \mathbf{e}_2 \overline{M}_{B_1}, \tag{17}$$

$$\overline{\mathbf{c}}_2 = \mathbf{e}_1 \overline{M}_{A_2} + \mathbf{p}\overline{M}_{C_2},\ \text{and} \tag{18}$$

$$\overline{\mathbf{c}}_3 = \mathbf{e}_2 \overline{M}_{B_3} + \mathbf{p}\overline{M}_{C_3}. \tag{19}$$

Moreover, from (17) we get

$$\mathbf{e}_1 = (\overline{\mathbf{c}}_1 - \mathbf{e}_2 \overline{M}_{B_1})\overline{M}_{A_1}^{-1}, \tag{20}$$

while from (18)

$$\mathbf{e}_1 = (\overline{\mathbf{c}}_2 - \mathbf{p}\overline{M}_{C_2})\overline{M}_{A_2}^{-1}. \tag{21}$$

Combining (20) and (21) we can express $\mathbf{e}_2$ in terms of $\mathbf{p}$ as

$$\mathbf{e}_2 = \mathbf{p}\overline{M}_{C_2}\overline{M}_{A_2}^{-1}\overline{M}_{A_1}\overline{M}_{B_1}^{-1} - \bar{\mathbf{c}}_2\overline{M}_{A_2}^{-1}\overline{M}_{A_1}\overline{M}_{B_1}^{-1} + \bar{\mathbf{c}}_1\overline{M}_{B_1}^{-1} \tag{22}$$

From (19) we compute

$$\mathbf{e}_2 = (\bar{\mathbf{c}}_3 - \mathbf{p}\overline{M}_{C_3})\overline{M}_{B_3}^{-1}. \tag{23}$$

Combining (22) and (23) we end up with a system of ℓ linear equations that allows us to compute $\mathbf{p}$ from public data only. Formally

$$\mathbf{p}(\overline{M}_{C_2}\overline{M}_{A_2}^{-1}\overline{M}_{A_1}\overline{M}_{B_1}^{-1}+\overline{M}_{C_3}\overline{M}_{B_3}^{-1}) = \bar{\mathbf{c}}_2\overline{M}_{A_2}^{-1}\overline{M}_{A_1}\overline{M}_{B_1}^{-1}-\bar{\mathbf{c}}_1\overline{M}_{B_1}^{-1}+\bar{\mathbf{c}}_3\overline{M}_{B_3}^{-1}. \tag{24}$$

Once $\mathbf{p}$ has been recovered we can simply plug it into (21) and (22) to obtain $\mathbf{e}_1$ and $\mathbf{e}_2$ recovering the entire plaintext.

Complexity of the attack. The complexity of the attack is dominated by the complexity of constructing and inverting the matrices $M_{A_1}, M_{B_1}, M_{A_2}, M_{C_2}, M_{B_3}$ and M_{C_3}. As for algorithms ReduceMLR1 and MsgRecovery, a safe upper bound is thus $\mathcal{O}((n_2 m \log_2(q))^\omega)$. These computations are performed only once in case an attacker decrypts many ciphertexts. We implemented this attack in SageMath. An average of the time required, on a Linux Mint virtual machine, to recover the plaintext for the proposed parameters is given in Table 4.

Security	$(q, m, n_I, n_1, n_2, n_A)$	n_E	Time (s)
128	$(2, 67, 37, 37, 61, 4)$	19	11.66
192	$(2, 79, 43, 43, 71, 4)$	23	16.32
256	$(2, 97, 53, 53, 103, 4)$	45	–

Table 4. Parameter sets for MLR3 and average time in seconds (on 50 samples for each security level) needed to recover a plaintext.

As stated above in Remark 13 the size of ℓ relative to n_2 prevents our attack in the case of security level 256. We do not see how to get enough linearly-independent equations.

Acknowledgments

The authors would like to thank Daniel J. Bernstein for discussions regarding recovering P_I in the attack presented in Section 3.2.

References

1. Aragon, N., Blazy, O., Deneuville, J.C., Gaborit, P., Hauteville, A., Ruatta, O., Tillich, J.P., Zémor, G., Aguilar Melchor, C., Bettaieb, S., Bidoux, L., Bardet, M., Otmani, A.: ROLLO. Tech. rep., NIST (2019), available at Round 2 page

2. Bardet, M., Briaud, P., Bros, M., Gaborit, P., Neiger, V., Ruatta, O., Tillich, J.P.: An algebraic attack on rank metric code-based cryptosystems. In: Eurocrypt 2020. LNCS, vol. 12107, pp. 64–93 (2020). https://doi.org/10.1007/978-3-030-45727-3_3
3. Bardet, M., Briaud, P., Bros, M., Gaborit, P., Tillich, J.P.: Revisiting algebraic attacks on MinRank and on the rank decoding problem. Designs, Codes and Cryptography pp. 1–37 (2023). https://doi.org/10.1007/s10623-023-01265-x
4. Bardet, M., Bros, M., Cabarcas, D., Gaborit, P., Perlner, R.A., Smith-Tone, D., Tillich, J.P., Verbel, J.A.: Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In: Asiacrypt 2020. LNCS, vol. 12491, pp. 507–536 (2020). https://doi.org/10.1007/978-3-030-64837-4_17
5. Bernstein, D., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer Berlin Heidelberg (2009), https://books.google.ch/books?id=VB598lO47NAC
6. Gabidulin, E.M., Paramonov, A.V., Tretjakov, O.V.: Ideals over a non-commutative ring and thier applications in cryptology. In: Eurocrypt 1991. LNCS, vol. 547, pp. 482–489. Springer (1991)
7. Goppa, V.D.: A new class of linear error correcting codes. Problemy Peredachi Informatsii **60**, 24–30 (1970)
8. Kim, C., Kim, Y.S., No, J.S.: Layered ROLLO-I. Submission to KpqC Competition Round 1 (2022)
9. Kim, C., Kim, Y., No, J.: Comments and modification on Layered ROLLO on kPQC-forum. Slides attached to reply on KpqC bulletin, 19 May (2023)
10. Kim, C., Kim, Y., No, J.: Comments and modification on Layered ROLLO on kPQC-forum. Slides attached on KpqC Bulletin, 22 Sep (2023)
11. Kim, C., Kim, Y., No, J.: Comments and modification on Layered ROLLO on kPQC-forum. Slides attached to reply on KpqC Bulletin, 20 Oct (2023)
12. Kim, C., Kim, Y., No, J.: New design of blockwise interleaved ideal low-rank parity-check codes for fast post-quantum cryptography. IEEE Commun. Lett. **27**(5), 1277–1281 (2023)
13. Lange, T., Pellegrini, A., Ravagnani, A.: On the security of REDOG. Cryptology ePrint Archive, Paper 2023/1205 (2023)
14. Lange, T., Pellegrini, A., Ravagnani, A.: On the security of REDOG. In: Seo, H., Kim, S. (eds.) Information Security and Cryptology – ICISC 2023. pp. 282–305. Springer Nature Singapore, Singapore (2024)
15. McEliece, R.J.: A public-key cryptosystem based on algebraic coding theory (1978), jet Propulsion Laboratory DSN Progress Report 42–44. URL: http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF
16. Overbeck, R.: Structural attacks for public key cryptosystems based on Gabidulin codes. Journal of Cryptology **21**(2), 280–301 (2008)
17. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962). https://doi.org/10.1109/TIT.1962.1057777
18. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS. pp. 124–134. IEEE Computer Society (1994)
19. Weger, V., Gassner, N., Rosenthal, J.: A survey on code-based cryptography. CoRR **abs/2201.07119** (2022)

A Code Listings

A.1 Reduction attack

This section list the code snippets of our proof-of-concept implementations. Some optimizations, such as computing only parts of the multiplication matrices, are not taken into account because the computations are fast enough anyways.

A.2 Reduction attack

The first reduction consists just of one polynomial division and a reduction. For the one attacking MLR1 from Section 3.2 we include here the SageMath code to compute the ROLLO-I public key pk'.

```
reset()
from sage.doctest.util import Timer
# parameters for modified Layered-ROLLO-I
# 128 bits
q, m, degPI, n1, n2 = 2,67,11,37,61
# 192 bits
#q, m, degPI, n1, n2 = 2,79,15,43,71
# 256 bits security
#q, m, degPI, n1, n2 = 2,97,20,53,103

# the following parameters describe max polynomial degrees
# this script uses vectorspaces to construct these polynomials
# so considering the constant term we need 1 entry more to get
# the correct bound on max degree.
degPI += 1
n1 += 1
n2 += 1

# Choices of P1 according to modified Layered-ROLLO-I implementation
# 128 bits
P1=x^37 + x^22 + x^14 + x^2 + 1
# 192 bits
#P1=x^43 + x^27 + x^22 + x^5 + 1
# 256 bits
#P1 = x^53 + x^50 + x^41 +x^20 + 1

Fqm = GF(q^m)
P.<x> = Fqm[]
F2x.<x> = GF(2)[]

def shift(v,W):
  newv = [0] + [v[i-1] for i in range(1,len(v))]
  scalar = v[-1]
  return W(newv), scalar

def vecpol(pol, l):
  deg = pol.degree()
  assert deg < l
  W = VectorSpace(Fqm, l)
  return W(list(pol) + [0 for i in range(l-deg-1)])

def xnmodpol(n,pol):
  assert pol.is_monic()
  d = pol.degree()
  W = VectorSpace(Fqm, d)
  if n < d:
    return vecpol(x^n, d)
  else:
    Q = W(list(pol)[:-1])
    prev = Q
    for i in range(1,n-d+1):
      shifted, scalar = shift(prev, W)
      toadd = scalar*Q
      prev = shifted + toadd
    return prev

def getmultmatrix(B,pol):
  Bcoeff = list(B)
  nrows = degPI
  ncols = 2*n2 - 1
  vec = vecpol(B, ncols)
  N = [vec]
  for i in range(1,nrows):
    vec, _ = shift(vec, VectorSpace(Fqm, ncols))
    N.append(vec)

  N = matrix(N)
  leftN = N[:,:n2]
  for i in range(1,nrows):
    for j in range(-i,0):
      toadd = Bcoeff[j]*xnmodpol(n2+(i+j), pol)#vecpol((x^(n2+i+j)).mod(pol),n2)
      leftN[i] += toadd

  return leftN

V = VectorSpace(Fqm,n1)
u,v = V.random_element(), V.random_element()
Pu, Pv = P(list(u)), P(list(v))

# Pick a random irreducible P2 over F2. Might as well be taken over F_{2^m}.
P2 = F2x.irreducible_element(n2)
P1 = P(P1)
P2 = P(P2)

P2coeff = list(P2)
#PI = P(list(VectorSpace(Fqm,degPI).random_element()))
# PI is a polynomial of degree exctly degPI (-1 because I increased it at the beginning of the script)
PI = P.random_element(degPI - 1)
PO = P(list(VectorSpace(Fqm,n2).random_element()))
xy = (Pu.inverse_mod(P1)*Pv).mod(P1)
z = (PI*xy).mod(P1)

PP, PH = (PO*PI).mod(P2), (PO*z).mod(P2)

timer = Timer()
timer.start()
R = (PH*(PP.inverse_mod(P2))).mod(P2)

Rmat = getmultmatrix(R,P2)
M = Rmat[:,n2-degPI+1:]

ker = M.left_kernel()
i=1
while ker.dimension() > 1:
  print('taking an extra column..')
  M = Rmat[:,n2-degPI+1-i:]
  ker = M.left_kernel()
  i += 1

lamPI = P(list(ker.basis()[0]))
lamPO = (PP * (lamPI.inverse_mod(P2))).mod(P2)
lamz = (PH * (lamPO.inverse_mod(P2))).mod(P2)
key = (lamz * (lamPI.inverse_mod(P1))).mod(P1)
timer.stop()
print(f'found key matches real y/x : {key == xy} in {timer.walltime}')
```

A.3 First message recovery attack

```
reset()
from sage.doctest.util import Timer

# parameters for modified Layered-ROLLO-I
# 128 bits
q, m, degPI, n1, n2, nn = 2,67,4,37,61,1
# 192 bits
#q, m, degPI, n1, n2, nn = 2,79,4,43,71,2
# 256 bits security
#q, m, degPI, n1, n2, nn = 2,97,4,53,103,4

# Choices of P1 according to modified Layered-ROLLO-I implementation
# WE DO NOT CARE WHAT P1 IS. MIGHT AS WELL BE RANDOM.
# 128 bits
P1=x^37 + x^22 + x^14 + x^2 + 1
# 192 bits
#P1=x^43 + x^27 + x^22 + x^5 + 1
# 256 bits
#P1 = x^53 + x^50 + x^41 +x^20 + 1


lenE = n2 - n1- degPI - nn - 2
degE = lenE-1
print(degE)

Fqm = GF(q^m)
P.<x> = Fqm[]
F2x.<x> = GF(2)[]

def pis(G,y):
  M = copy(G)
  k,n = M.dimensions()
  p = list(Permutations(n).random_element())
  p = list(range(lenE+1,n2))
  shuffle(p)
  indexes = p[:k]
  indexes.sort()
  colsG = [M.columns()[i-1] for i in indexes]
  colsy = [y.columns()[i-1] for i in indexes]
  pisG = matrix(Fqm, colsG)
  pisy = matrix(Fqm, colsy)
  return pisG.transpose(), pisy.transpose()


def Prange(M, y, t):
  k,n = M.dimensions()
  while True:
    M1,y1 = pis(M,y)
    while not M1.is_invertible():
      M1,y1 = pis(M,y)
    U = M1.inverse()
    msg = y1*U
    x = msg*M
    wt = len([i for i in range(n) if x[0][i] != y[0][i]])
    if wt <= t:
      e = y - x
      return msg, e






def getmultmatrix(R,P2,nrows):
  M = []
  for i in range(n2):
    el = list((R*x^i).mod(P2))
    M.append(el + [0 for j in range(n2-len(el))])
  M = matrix(Fqm,M)
  return M[:nrows,:]


V = VectorSpace(Fqm,n1)
u,v = V.random_element(), V.random_element()
Pu, Pv = P(list(u)), P(list(v))

P2 = F2x.irreducible_element(n2)
P1 = P(P1)
P2 = P(P2)
PI = P.random_element(degPI)
PO = P(list(VectorSpace(Fqm,n2+1).random_element()))
PN = P.random_element(1)
xy = (Pu.inverse_mod(P1)*Pv).mod(P1)
z = (PI*xy).mod(P1)
lz = list(z) + [0 for j in range(0,n2-len(list(z)))]
#PP, PH = (PO*PI).mod(P2), (PO*z).mod(P2)

time = 0
timer = Timer()
for i in range(50):
  E1 = P.random_element(degE)
  E2 = P.random_element(degE)
  #### PUBLIC DATA
  PP, PH = (PO*(PI + PN*P1)).mod(P2), (PO*z).mod(P2)
  PC = (PP*E1 + PH*E2).mod(P2)

  timer.start()
  #### BEGIN ATTACK
  cipher = matrix(vector((PC*(PH.inverse_mod(P2))).mod(P2)))
  R = (PP*(PH.inverse_mod(P2))).mod(P2)
  Rmat = getmultmatrix(R,P2,lenE)
  e1_candidate, e2_candidate = Prange(Rmat,cipher,lenE)
  print(f'Message recovery  E_1: {E1 == P(list(e1_candidate[0]))}      E_2: {E2 == P(list(e2_candidate[0]))}')
  timer.stop()
  time += timer.walltime

print(f'Average recovery time : {time/50}')

#sol = Rmat.solve_left(vector(lz))
#print(P(list(sol)) == PI.mod(P2))
```

A.4 Second message recovery attack

```
reset()
from sage.doctest.util import Timer

# parameters for modified Layered-ROLLO-I
# 128 bits
#q, m, degPI, n1, n2, na = 2,67,37,37,61,4
# 192 bits
#q, m, degPI, n1, n2, na = 2,79,43,43,71,4
# 256 bits security
q, m, degPI, n1, n2, na = 2,97,53,53,103,4

# Choices of P1 according to modified Layered-ROLLO-I implementation
# WE DO NOT CARE WHAT P1 IS. MIGHT AS WELL BE RANDOM.
# 128 bits
#P1=x^37 + x^22 + x^14 + x^2 + 1
# 192 bits
#P1=x^43 + x^27 + x^22 + x^5 + 1
# 256 bits
P1 = x^53 + x^50 + x^41 +x^20 + 1
```

```
lenE = n2 - n1 - na - 1
degE = lenE-1
print(degE)

Fqm = GF(q^m)
P.<x> = Fqm[]
F2x.<x> = GF(2)[]

def getmultmatrix(R,P2,nrows):
  M = []
  for i in range(n2):
    el = list((R*x^i).mod(P2))
    M.append(el + [0 for j in range(n2-len(el))])
  M = matrix(Fqm,M)
  return M[:nrows,:]

def findcommoninv(M,N,cipher):
  k,n = M.dimensions()
  p = list(range(lenE+1,n))
  while True:
    shuffle(p)
    indexes = p[:k]
    indexes.sort()
    colsM = [M.columns()[i] for i in indexes]
    colsN = [N.columns()[i] for i in indexes]
    M1 = matrix(Fqm, colsM)
    N1 = matrix(Fqm, colsN)
    if M1.is_invertible() and N1.is_invertible():
      return M1.transpose(),N1.transpose(),vector([cipher[i] for i in indexes])

V = VectorSpace(Fqm,n1)
u,v = V.random_element(), V.random_element()
Pu, Pv = P(list(u)), P(list(v))

P2 = F2x.irreducible_element(n2)
P1 = P(P1)
P2 = P(P2)
PI = P.random_element(degPI)
PO = P(list(VectorSpace(Fqm,n2+1).random_element()))
PNA = P.random_element(na)
xy = (Pu.inverse_mod(P1)*Pv).mod(P1)
z = (PI*xy).mod(P1)
lz = list(z) + [0 for j in range(0,n2-len(list(z)))]
#PP, PH = (PO*PI).mod(P2), (PO*z).mod(P2)

time = 0
timer = Timer()
for i in range(50):
  E1 = P.random_element(degE)
  E2 = P.random_element(degE)
  PNC = P.random_element(degE)
  PNB = P.random_element(na)
  #### PUBLIC DATA
  PP, PH = (PO*(PI + PNA*P1)).mod(P2), (PO*z).mod(P2)
  PB = (PO*PNB*P1).mod(P2)
  PC = (PP*E1 + PH*E2 + PB*PNC).mod(P2)

  print('Begin ATTACK')
  timer.start()
  #### BEGIN ATTACK
  s1 = vector((PC*(PB.inverse_mod(P2))).mod(P2))
  s2 = vector((PC*(PH.inverse_mod(P2))).mod(P2))
  s3 = vector((PC*(PP.inverse_mod(P2))).mod(P2))

  A1 = (PP*(PB.inverse_mod(P2))).mod(P2)
  B1 = (PH*(PB.inverse_mod(P2))).mod(P2)
  A1mat = getmultmatrix(A1,P2,lenE)
  B1mat = getmultmatrix(B1,P2,lenE)
  A1bar,B1bar,s1bar = findcommoninv(A1mat,B1mat,s1)

  A2 = (PP*(PH.inverse_mod(P2))).mod(P2)
  C2 = (PB*(PH.inverse_mod(P2))).mod(P2)
  A2mat = getmultmatrix(A2,P2,lenE)
  C2mat = getmultmatrix(C2,P2,lenE)
  A2bar,C2bar,s2bar = findcommoninv(A2mat,C2mat,s2)

  B3 = (PH*(PP.inverse_mod(P2))).mod(P2)
  C3 = (PB*(PP.inverse_mod(P2))).mod(P2)
  B3mat = getmultmatrix(B3,P2,lenE)
  C3mat = getmultmatrix(C3,P2,lenE)
  B3bar,C3bar,s3bar = findcommoninv(B3mat,C3mat,s3)

  Sys = C3bar*B3bar.inverse() + C2bar*A2bar.inverse()*A1bar*B1bar.inverse()
  val = s2bar*A2bar.inverse()*A1bar*B1bar.inverse() - s1bar*B1bar.inverse() + s3bar*B3bar.inverse()
  z1 = Sys.solve_left(val)
  x1 = (s2bar - z1*C2bar)*A2bar.inverse()
  y1 = (s3bar - z1*C3bar)*B3bar.inverse()
  print(f'Message recovery  E_1: {E1 == P(list(x1))}      E_2: {E2 == P(list(y1))}    PNC: {PNC == P(list(z1))}')
  timer.stop()
  time += timer.walltime

print(f'Average recovery time : {time/50}')
```