On the security of REDOG

Tanja Lange, Alex Pellegrini, and Alberto Ravagnani

Eindhoven University of Technology, the Netherlands
tanja@hyperelliptic.org, alex.pellegrini@live.com, a.ravagnani@tue.nl

Abstract. We analyze REDOG, a public-key encryption system submitted to the Korean competition on post-quantum cryptography. REDOG is based on rank-metric codes. We prove its incorrectness and attack its implementation providing an efficient message recovery attack. Furthermore, we show that the security of REDOG is much lower than claimed. We then proceed to mitigate these issues and provide two approaches to fix the decryption issue, one of which also leads to better security.

1 Introduction

This paper analyzes the security of the REinforced modified Dual-Ouroboros based on Gabidulin codes, REDOG [KHL+22a] in short, a public-key encryption system submitted to KpqC, the Korean competition on post-quantum cryptography. REDOG is a code-based cryptosystem using rank-metric codes, aiming at providing a rank-metric alternative to Hamming-metric code-based cryptosystems.

Rank-metric codes were introduced by Delsarte [Del78] and independently rediscovered by Gabidulin [Gab85] in 1985, who focused on those that are linear over a field extension. Gabidulin, Paramonov, and Tretjakov [GPT91] proposed their use for cryptography in 1991. The GPT system was attacked by Overbeck [Ove05,Ove08] who showed *structural* attacks, permitting recovery of the private key from the public key.

During the mid 2010s new cryptosystems using rank-metric codes were developed such as Ouroboros [DGZ17] and the first round of the NIST competition on post-quantum cryptography saw 5 systems based on rank-metric codes: LAKE [ABD+17a], LOCKER [ABD+17b], McNie [GKK+17], Ouroboros-R [AAB+17a], and RQC [AAB+17b]. For all these systems see NIST's Round-1

Author list in alphabetical order; see https://www.ams.org/profession/leaders/culture/CultureStatement04.pdf. This work was funded in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2092 CASA—390781972 "Cyber Security in the Age of Large-Scale Adversaries" and by the Netherlands Organisation for Scientific Research (NWO) under grants OCENW.KLEIN.539 and VI.Vidi.203.045. Date: 2023.08.08.

Submissions page. Gaborit announced an attack weakening McNie and the McNie authors adjusted their parameters. A further attack was published in [LT18] and NIST did not advance McNie into the second round of the competition.

ROLLO, a merger of LAKE, LOCKER and Ouroboros-R, and RQC made it into the the second round but got broken near the end of it by significant advances in the cryptanalysis of rank-metric codes and the MinRank problem in general, see [BBB+20] and [BBC+20]. In their report at the end of round 2 [AASA+20], NIST wrote an encouraging note on rank-metric codes: "Despite the development of algebraic attacks, NIST believes rank-based cryptography should continue to be researched. The rank metric cryptosystems offer a nice alternative to traditional hamming metric codes with comparable bandwidth." (capitalization as in the original).

Kim, Kim, Galvez, and Kim [KKGK21] proposed a new rank-metric system in 2021 which was then analyzed by Lau, Tan, and Prabowo in [LTP21] who also proposed some modifications to the issues they found. REDOG closely resembles the system in [LTP21] and uses the same parameters.

1.1 Our contribution

In this paper we expose weaknesses of REDOG and show that the system, as described in the documentation, is incorrect. To start with, we prove that REDOG does not decrypt correctly. The documentation and [LTP21] contain an incorrect estimate of the rank of an element which causes the input to the decoding step to have too large rank. The system uses Gabidulin codes [Gab85] which are MRD (Maximum Rank Distance) codes, meaning that vectors with errors of rank larger than half the minimum distance will decode to a different codeword, thus causing incorrect decryption in the REDOG system.

As a second contribution we attack ciphertexts produced by REDOG's reference implementation. We show that we can use techniques from the Hamming metric to obtain a message-recovery attack. This stems from a choice in the implementation which avoids the above-mentioned decryption problem. However, the errors introduced in the ciphertext have a specific shape which allows us to apply basic techniques of Information Set Decoding (ISD) over the Hamming metric to recover the message in seconds.

As a third contribution, we show that, independently of the special choice of error vectors in the implementation, the security of the cryptosystem is lower than the claimed security level. The main effect comes from a group of attacks published in [BBC+20] which the REDOG designers had not taken into account. An smaller effect comes from a systematic scan through all attack parameters.

Finally, we provide two ways to make REDOG's decryption correct. The first is a minimal change to fix the system by changing the space from which some matrix P^{-1} is chosen in a way that differs from the choice in REDOG and avoids the issue mentioned above. However, this still requires choosing much larger parameters to deal with our third contribution. The second way makes a different change to REDOG which improves the resistance to attacks while also fixing the decryption issue. We show that the REDOG-128 parameters are

sufficient when using this second idea to reach 128 bits of security; however the REDOG-192 and REDOG-256 parameters still fall short and would need to be adjusted further. Note, however, that these estimates are obtained from big-$\mathcal{O}$ complexity estimates, putting all constants to 1 and lower-order terms to 0, and thus underestimate the security.

2 Preliminaries and background notions

This section gives the necessary background on rank-metric codes for the rest of the paper.

Let $\{\alpha_1, \ldots, \alpha_m\}$ be a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Write $x \in \mathbb{F}_{q^m}$ uniquely as $x = \sum_{i=1}^{m} X_i \alpha_i$, $X_i \in \mathbb{F}_q$ for all i. So x can be represented as $(X_1, \ldots, X_m) \in \mathbb{F}_q^m$. We will call this the *vector representation* of x. Extend this process to $\mathbf{v} = (v_1, \ldots, v_n) \in \mathbb{F}_{q^m}^n$ defining a map $\mathsf{Mat} : \mathbb{F}_{q^m}^n \to \mathbb{F}_q^{m \times n}$ by:

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

Definition 2.1. *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 2.2. 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 attack against REDOG's implementation.

The *Hamming weight* of a vector $\mathbf{v} \in \mathbb{F}_{q^m}^n$ is defined as $\mathsf{wt}_H(\mathbf{v}) := \#\{i \in \{1, \ldots, n\} \mid v_i \neq 0\}$ 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 is a *generator matrix* of C if its rows span C. We say that H is a *parity check matrix* of C if C is the right-kernel of H.

A very well-known family of rank metric codes are *Gabidulin codes* [Gab85], which have $d = n - k + 1$.

In this paper we can mostly use these codes as a black box, knowing that there is an efficient decoding algorithm using the parity-check matrix of the code and decoding vectors with errors of rank up to $\lfloor (d-1)/2 \rfloor$.

We will, however, use that the parity-check matrix of a Gabidulin code over $\mathbb{F}_q$, is a Moore matrix.

Definition 2.3. *Let $\mathbb{F}_{q^m}$ be a finite field. A matrix $M \in \mathbb{F}_{q^m}^{k \times n}$ is a* Moore matrix *if each row is the q-th power of the previous one.*

This structural property was used in the structural attacks by Overbeck [Ove08] to find the secret Gabidulin code hidden in the GPT system [GPT91]. This structure is also used in the analysis of [KKGK21] in [LTP21]. For a more general definition and further details on Moore matrices in this cryptographic context, see [HTMR15].

A final definition necessary to understand REDOG is that of isometries.

Definition 2.4. *Consider vectors in $\mathbb{F}_{q^m}^n$. An* isometry with respect to the rank metric *is a matrix $P \in \mathsf{GL}_n(\mathbb{F}_{q^m})$ satisfying that $\mathsf{wt}_R(\mathbf{v}P) = \mathsf{wt}_R(\mathbf{v})$ for any $\mathbf{v} \in \mathbb{F}_{q^m}^n$.*

Obviously matrices $P \in \mathsf{GL}_n(\mathbb{F}_q)$ are isometries as $\mathbb{F}_q$-linear combinations of the coordinates of $\mathbf{v}$ do not increase the rank and the rank does not decrease as P is invertible. The rank does also not change under scalar multiplication by some $\alpha \in \mathbb{F}_{q^m}^*$: $\mathsf{wt}_R(\alpha\mathbf{v}) = \mathsf{wt}_R(\mathbf{v})$. Note that the latter corresponds to multiplication by $P = \alpha I_n$.

Berger [Ber03] showed that any isometry is obtained by composing these two options.

Theorem 2.5. *[Ber03, Theorem 1] The isometry group of $\mathbb{F}_{q^m}^n$ for the rank metric is generated by scalar multiplications by elements in $\mathbb{F}_{q^m}^*$ and elements of $\mathsf{GL}_n(\mathbb{F}_q)$. This group is isomorphic to the product group $\left(\mathbb{F}_{q^m}^*/\mathbb{F}_q^*\right) \times \mathsf{GL}_n(\mathbb{F}_q)$.*

3 System specification

This section introduces the specification of REDOG. We follow the notation of [LTP21], with minor changes.

The system parameters are positive integers $(n, k, \ell, q, m, r, \lambda, t)$, with $\ell < n$ and $\lambda t \leq r \leq \lfloor (n-k)/2 \rfloor$, as well as a hash function $\mathsf{hash} : \mathbb{F}_{q^m}^{2n-k} \to \mathbb{F}_{q^m}^{\ell}$.

KeyGen:

1. Select $H = (H_1 \mid H_2)$, $H_2 \in \mathsf{GL}_{n-k}(\mathbb{F}_{q^m})$, a parity check matrix of a $[2n-k, n]$ Gabidulin code, with syndrome decoder Φ correcting r errors.
2. Select a full rank matrix $M \in \mathbb{F}_{q^m}^{\ell \times n}$ and isometry $P \in \mathbb{F}_{q^m}^{n \times n}$ (w.r.t. the rank metric).

3. Select a λ-dimensional subspace $\Lambda \subset \mathbb{F}_{q^m}$, seen as $\mathbb{F}_q$-linear space, containing 1 and select $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$; see Section 4 for the definition.
4. Compute $F = MP^{-1}H_1^T \left(H_2^T\right)^{-1} S$ and publish the public key $\mathsf{pk} = (M, F)$.
5. Store the secret key $\mathsf{sk} = (P, H, S, \Phi)$.

Encrypt $(\mathbf{m} \in \mathbb{F}_{q^m}^{\ell}, \mathsf{pk})$

1. Generate uniformly random $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_{q^m}^{2n-k}$ with $\mathsf{wt}_R(\mathbf{e}) = t$, $\mathbf{e}_1 \in \mathbb{F}_{q^m}^{n}$ and $\mathbf{e}_2 \in \mathbb{F}_{q^m}^{n-k}$.
2. Compute $\mathbf{m}' = \mathbf{m} + \mathsf{hash}(\mathbf{e})$.
3. Compute $\mathbf{c}_1 = \mathbf{m}'M + \mathbf{e}_1$ and $\mathbf{c}_2 = \mathbf{m}'F + \mathbf{e}_2$ and send $(\mathbf{c}_1, \mathbf{c}_2)$.

Decrypt $((\mathbf{c}_1, \mathbf{c}_2), \mathsf{sk})$

1. Compute $\mathbf{c}' = \mathbf{c}_1 P^{-1} H_1^T - \mathbf{c}_2 S^{-1} H_2^T = \mathbf{e}' H^T$ where the vector $\mathbf{e}' := (\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1})$.
2. Decode $\mathbf{c}'$ using Φ to obtain $\mathbf{e}'$ and thus recover $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ using P and S.
3. Solve $\mathbf{m}'M = \mathbf{c}_1 - \mathbf{e}_1$.
4. Output $\mathbf{m} = \mathbf{m}' - \mathsf{hash}(\mathbf{e})$.

3.1 Suggested parameters

We list the suggested parameters of REDOG for 128,192 and 256 bits of security, following the document [KHL+22a] submitted to KpqC.

Security parameter	$(n, k, \ell, q, m, r, \lambda, t)$
128	$(44, 8, 37, 2, 83, 18, 3, 6)$
192	$(58, 10, 49, 2, 109, 24, 3, 8)$
256	$(72, 12, 61, 2, 135, 30, 3, 10)$

Table 1. Suggested parameters; see [KHL+22a].

4 Incorrectness of decryption

This section shows that decryption typically fails for the version of REDOG specified in [KHL+22a,LTP21]. The novelty of this specification, compared to that introduced in [KKGK21], lies in the selection of the invertible matrix S^{-1} in Step 3, which is selected with the property that $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$, where Λ is a λ-dimensional $\mathbb{F}_q$-subspace of $\mathbb{F}_{q^m}$. This method has been first proposed by Loidreau in [Loi17], but it appears to be incorrectly applied in REDOG. Before providing more details about this claim and proving the incorrectness of REDOG's decryption process, we will shed some light on the object $\mathsf{GL}_{n-k}(\Lambda)$. Unlike the notation suggests, this is not a group, but a potentially unstructured subset of $\mathsf{GL}_{n-k}(\mathbb{F}_{q^m})$ defined as follows:

Let $\{1, \alpha_2, \ldots, \alpha_\lambda\} \subset \mathbb{F}_{q^m}$ be a set of elements that are $\mathbb{F}_q$-linearly independent. Let $\Lambda \subset \mathbb{F}_{q^m}$ be the set of $\mathbb{F}_q$-linear combinations of these α_i's. This set forms an $\mathbb{F}_q$-linear vectorspace. Now, $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ is defined to mean that S is an invertible $(n-k)\times(n-k)$ matrix with the property that the entries of S^{-1} are elements of Λ. Note that such an S exists because $\lambda \geq 1$ by assumption. The REDOG documentation [KHL+22a] points out that this does not imply that $S \in \mathsf{GL}_{n-k}(\Lambda)$, hence, despite what the notation may suggest, $\mathsf{GL}_{n-k}(\Lambda)$ is not a group in general.

We continue by giving a proof, and an easy generalization for any q, of [Loi17, Proposition 1].

Proposition 4.1. *Let λ, t, n be positive integers such that $\lambda t \leq n$, $A \in \mathsf{GL}_n(\Lambda)$ where $\Lambda \subset \mathbb{F}_{q^m}$ is a λ-dimensional subspace of $\mathbb{F}_{q^m}$, and $\mathbf{x} \in \mathbb{F}_{q^m}^n$ with $\mathsf{wt}_R(\mathbf{x}) = t$. Then*

$$\mathsf{wt}_R(\mathbf{x}A) \leq \lambda t.$$

Proof. Let Γ be the subspace of $\mathbb{F}_{q^m}$ generated by the entries of $\mathbf{x} = (x_1, \ldots, x_n)$. Since Γ has dimension t, we can write $\Gamma = \langle y_1, \ldots, y_t \rangle$ with $y_i \in \mathbb{F}_{q^m}$. Similarly for Λ, we can write $\Lambda = \langle \alpha_1, \ldots, \alpha_\lambda \rangle$ with $\alpha_i \in \mathbb{F}_{q^m}$. Express $\mathbf{x}A$ as

$$\mathbf{x}A = \left(\sum_{i=1}^{n} x_i A_{i,1}, \ldots, \sum_{i=1}^{n} x_i A_{i,n} \right).$$

Fix $j \in \{1, \ldots, n\}$. Then

$$(\mathbf{x}A)_j = \sum_{i=1}^{n} x_i A_{i,j} = \sum_{i=1}^{n} \left(\left(\sum_{h=1}^{t} x_{i,h} y_h \right) \left(\sum_{k=1}^{\lambda} A_{i,j,k} \alpha_k \right) \right),$$

with $x_{i,h}, A_{i,j,k} \in \mathbb{F}_q$. By rearranging the terms we obtain

$$(\mathbf{x}A)_j = \sum_{h=1}^{t} \sum_{k=1}^{\lambda} \left(\sum_{i=1}^{n} x_{i,h} A_{i,j,k} \right) y_h \alpha_k. \tag{1}$$

Therefore each entry of $\mathbf{x}A$ can be expressed as an $\mathbb{F}_q$-linear combination of the λt elements of the form $y_h \alpha_k$. □

We will now show that REDOG typically does not decrypt correctly. In order to do so, we need some preliminary results and tools. The proof of the next lemma uses some tools from combinatorics.

Proposition 4.2. *Let V be a t-dimensional subspace $V \subseteq \mathbb{F}_q^m$ and let $S \in V^s$ be a uniformly random s-tuple of elements of V. The probability that $\langle S_i \mid i \in \{1, \ldots, s\} \rangle = V$ is*

$$p(q,s,t) = \begin{cases} 0 & if \quad 0 \leq s < t; \\ \sum_{i=0}^{t} \begin{bmatrix} t \\ i \end{bmatrix}_q (-1)^{t-i} q^{s(i-t)+\binom{t-i}{2}} & otherwise, \end{cases} \tag{2}$$

where $\begin{bmatrix} t \\ i \end{bmatrix}_q$ is the q-binomial coefficient, counting the number of subspaces of dimension i of $\mathbb{F}_q^t$, and $\binom{a}{b} = 0$ for $a < b$. In particular, this probability does not depend on the choice of V but only on its dimension.

Proof. Let $(\mathcal{P}, \subseteq)$ be the poset (partially ordered set) of subspaces of $\mathbb{F}_q^m$ ordered by inclusion. Recall that the Möbius function of $\mathcal{P}$, and of any finite poset, is defined, for $A, B \in \mathcal{P}$, as

$$\mu(B, A) = \begin{cases} 1 & \text{if } B = A, \\ -\sum_{C \mid B \subseteq C \subset A} \mu(B, C) & \text{if } B \subset A, \\ 0 & \text{otherwise.} \end{cases}$$

This is computed e.g. in [Sta11, Example 3.10.2] as

$$\mu(B, A) = \begin{cases} (-1)^k q^{\binom{k}{2}} & \text{if } B \subseteq A \text{ and } \dim(A) - \dim(B) = k, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

We want to compute the function $f : \mathcal{P} \to \mathbb{N}$ defined as

$$f(A) = \#\left\{S \in \left(\mathbb{F}_q^m\right)^s \mid \langle S \rangle = A\right\}.$$

Clearly, if $s < \dim A$, there does not exist any s-tuple S spanning A, hence $f(A) = 0$, which gives the first case of (2). We can therefore restrict ourselves to the case $s \geq \dim A$. Define the auxiliary function $g : \mathcal{P} \to \mathbb{N}$ as

$$\begin{aligned} g(A) &= \sum_{B \subseteq A} f(B) \\ &= \#\left\{S \in \left(\mathbb{F}_q^m\right)^s \mid \langle S \rangle \subseteq A\right\} \\ &= |A|^s = q^{s \dim A}. \end{aligned}$$

Then by Möbius inversion we can compute:

$$f(A) = \sum_{B \subseteq A} g(B)\mu(B, A). \tag{4}$$

Splitting the sum over the dimensions, and substituting the values in Equation 3, we can obtain

$$\begin{aligned} f(V) &= \sum_{i=0}^{t} \sum_{U \subseteq V,\, \dim U = i} g(U)\mu(U, V) \\ &= \sum_{i=0}^{t} q^{si} (-1)^{t-i} q^{\binom{t-i}{2}} \sum_{U \subseteq V,\, \dim U = i} 1 \\ &= \sum_{i=0}^{t} \begin{bmatrix} t \\ i \end{bmatrix}_q (-1)^{t-i} q^{si + \binom{t-i}{2}} \end{aligned}$$

The probability can be computed by dividing $f(V)$ by the number of s-tuples of elements of V, that is, q^{st}. □

Remark 4.3. The probability given in Proposition 4.2 can be interpreted as the ratio of the number of surjective linear maps from $\mathbb{F}_q^s$ onto $\mathbb{F}_q^t$ over the total number of linear maps.

Corollary 4.4. *If* $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_{q^m}^{2n-k}$, *with* $\mathbf{e}_1 \in \mathbb{F}_{q^m}^{n}$ *and* $\mathbf{e}_2 \in \mathbb{F}_{q^m}^{n-k}$, *is a uniformly random error with* $\mathsf{wt}_R(\mathbf{e}) = t$, *then* $\mathsf{wt}_R(\mathbf{e}_1) = t$ *and* $\mathsf{wt}_R(\mathbf{e}_2) = t$ *with probability* $p(q, n, t)$ *and* $p(q, n-k, t)$ *respectively.*

Example 4.5. Consider the suggested parameters of REDOG for 128 bits of security from Table 1. Using SageMath [S+21] we computed the probability that $\mathsf{wt}_R(\mathbf{e}_1) = t$, that is

$$p(2, 44, 6) = 0.999999999996419,$$

and the probability that $\mathsf{wt}_R(\mathbf{e}_2) = t$, that is

$$p(2, 36, 6) = 0.999999999083229.$$

We are ready to state the following theorem, which directly implies that REDOG's decryption process fails with extremely high probability.

Theorem 4.6. *Let* (n, k, q, m, λ, t) *be integers with* $k < n < m$ *and* $\lambda t \leq m$. *Let* $\Lambda \subset \mathbb{F}_{q^m}$ *be a* λ*-dimensional subspace of* $\mathbb{F}_{q^m}$ *and* $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ *as in Corollary 4.4. Let* $P \in \mathbb{F}_{q^m}^{n \times n}$ *be a random isometry matrix (w.r.t. the rank metric) and* $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$. *Then* $\mathbf{e}' := \left(\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1}\right)$ *has rank weight* $\mathsf{wt}_R(\mathbf{e}') \geq \lambda t + 1$ *with probability bounded from below by*

$$p_{\mathsf{fail}}(n, k, q, m, \lambda, t) := p(q, n, t)\, p(q, n-k, \lambda t)\, p(q, n-k, t) \left(1 - \frac{\begin{bmatrix} \lambda t \\ t \end{bmatrix}_q}{\begin{bmatrix} m \\ t \end{bmatrix}_q}\right).$$

Proof. By Theorem 2.5, the isometry P is of the form $\alpha \bar{P}$ for $\alpha \in \mathbb{F}_{q^m}^*$ and $\bar{P} \in \mathsf{GL}_n(\mathbb{F}_q)$, where $q^m \gg q$ and thus typically $\alpha \notin \mathbb{F}_q$. Because of the multiplication by α^{-1}, we can assume that the linear transformation induced by P^{-1} takes a t-dimensional subvectorspace of $\mathbb{F}_q^m$ to a random t-dimensional subspace. Similarly we assume that S^{-1} sends a t-dimensional subspace of $\mathbb{F}_q^m$ to a random subspace of dimension at most λt, by Proposition 4.1. We get the lower bound on the failure probability by showing the following:

1. $\mathsf{wt}_R(\mathbf{e}_1 P^{-1}) = t$ with probability $p(q, n, t)$;
2. $\mathsf{wt}_R(-\mathbf{e}_2 S^{-1}) = \lambda t$ with probability $p(q, n-k, t) p(q, n-k, \lambda t)$;
3. under the conditions in (1) and (2), $\langle \mathbf{e}_1 P^{-1} \rangle \not\subset \langle -\mathbf{e}_2 S^{-1} \rangle$ with probability

$$1 - \frac{\begin{bmatrix} \lambda t \\ t \end{bmatrix}_q}{\begin{bmatrix} m \\ t \end{bmatrix}_q}.$$

Note that (1) follows directly from Corollary 4.4 and the fact that P is an isometry of the space w.r.t the rank metric.

Likewise, $\mathsf{wt}_R(-\mathbf{e}_2) = t$ with probability $p(q, n-k, t)$. The proof of Proposition 4.1 shows that for $\mathbf{e}_2$ with $\mathsf{wt}_R(-\mathbf{e}_2) = t$ we have that $-\mathbf{e}_2 S^{-1}$ is contained in a λt-dimensional subspace of $\mathbb{F}_q^m$. Again by Corollary 4.4 we obtain that $\langle -\mathbf{e}_2 S^{-1} \rangle$ spans the entire space with probability $p(q, n-k, \lambda t)$, proving 2.

To prove 3 we will compute the opposite, i.e. the probability that $\langle \mathbf{e}_1 P^{-1} \rangle$ is a subspace of $\langle -\mathbf{e}_2 S^{-1} \rangle$. As mentioned at the beginning of the proof, we treat $\langle \mathbf{e}_1 P^{-1} \rangle$ as a random t-dimensional subspace of $\mathbb{F}_{q^m}$. Thus we can compute this probability as the ratio between the number of t-dimensional subspaces of $\langle -\mathbf{e}_2 S^{-1} \rangle$ and of $\mathbb{F}_q^m$, that is,

$$\frac{\begin{bmatrix} \lambda t \\ t \end{bmatrix}_q}{\begin{bmatrix} m \\ t \end{bmatrix}_q}.$$

Combining the probabilities and observing that (1 – 3) imply $\mathsf{wt}_R(\mathbf{e}') \geq \lambda t + 1$ gives the result. □

Remark 4.7. There are more ways to get $\mathsf{wt}_R(\mathbf{e}') \geq \lambda t + 1$ by relaxing the first two requirements in the proof of Theorem 4.6 and studying the dimension of the union in the third, but p_{fail} is large enough for the parameters in REDOG to prove the point.

Remark 4.8. The proof of property (3) relies on $\mathbf{e}_1 P^{-1}$ being a random subspace of dimension t. We note that for $\alpha \in \mathbb{F}_q$ we have $\langle \mathbf{e}_1 \rangle = \langle \mathbf{e}_1 P^{-1} \rangle \subset \langle \mathbf{e}_2 S^{-1} \rangle$ for $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ and $1 \in \Lambda$. The latter constraint is stated in [KHL+22a] and [LTP21] and it is possible that the authors were not aware of the full generality of isometries. See also Section 7 for further observations on [LTP21] which are consistent with this misconception.

Corollary 4.9. *Let $(n, k, \ell, q, m, r, \lambda, t)$ be the parameters of a instance of REDOG with $r = \lambda t$. Then REDOG will produce decryption failures with probability at least $p_{\mathsf{fail}}(n, k, q, m, \lambda, t)$.*

Proof. Recall that the decoder Φ can only correct errors up to rank weight $r = \lambda t$. By Theorem 4.6 we have that $\mathbf{e}'$ has rank weight $\geq \lambda t + 1$, hence producing decoding failure, with probability at least $p_{\mathsf{fail}}(n, k, q, m, \lambda, t)$.

Note that a $[2n-k, n]$ Gabidulin code has minimum distance $d_R = 2n - k - n + 1 = n - k + 1$ and can thus correct at most $\lfloor (n-k)/2 \rfloor$ errors and that all instances of REDOG in Table 1 satisfy $\lfloor (n-k)/2 \rfloor = r = \lambda t$.

Example 4.10. As in Example 4.5, consider the suggested parameters for 128 bits of security. Then Theorem 4.6 states that $\mathsf{wt}_R(\mathbf{e}') \geq 19$ with probability at least

$$\begin{aligned} p_{\mathsf{fail}}(44, 8, 2, 83, 3, 6) &= p(2, 44, 6)p(2, 36, 6)p(2, 36, 18)\left(1 - \frac{\begin{bmatrix} 18 \\ 6 \end{bmatrix}_2}{\begin{bmatrix} 83 \\ 6 \end{bmatrix}_2}\right) \\ &= 0.999996184401789. \end{aligned}$$

Table 2 reports the value of p_{fail} for each set of security parameters given in Table 1. This shows that REDOG's decryption process fails almost always.

Security parameter	p_{fail}
128	0.999996184401789
192	0.999999940394453
256	0.999999999068677

Table 2. Value of decryption failure probability p_{fail} per suggested parameters.

5 Message recovery attack on REDOG's implementation

Theorem 4.6 and the numerical examples show that, with probability almost 1, REDOG will fail decrypting. However, it is not exactly 1 and there exist some choices of $\mathbf{e}$ for which decryption still succeeds. One extreme way to avoid decryption failures, chosen in the refenrence implementation of REDOG, is to build errors as follows:

Algorithm 5.1 *(REDOG's error generator)*

1. *Pick* $\beta_1, \ldots, \beta_t \in \mathbb{F}_{q^m}$ *being* $\mathbb{F}_q$*-linearly independent.*
2. *Pick random permutation* π *on* $2n-k$ *symbols.*
3. *Set* $\mathbf{e}_{\mathsf{init}} = (\beta_1, \ldots, \beta_t, 0, \ldots, 0) \in \mathbb{F}_{q^m}^{2n-k}$.
4. *Output* $\mathbf{e} = \pi(\mathbf{e}_{\mathsf{init}})$.

Error vectors in REDOG's reference implementation[1], whose performance is analyzed in [KHL+22b], are generated in an equivalent way to Algorithm 5.1. Indeed, $\mathbf{e}'$ has rank weight

$$\mathsf{wt}_R(\mathbf{e}') = (\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1}) \leq \lambda t$$

and can therefore be decoded using Φ.

Remark 5.2. Algorithm 5.1 produces an error vector $\mathbf{e}$ such that $\mathsf{wt}_H(\mathbf{e}) = \mathsf{wt}_R(\mathbf{e}) = t$ as only t coordinates of $\mathbf{e}$ are nonzero.

We are ready to give the description of an efficient message recovery algorithm.

Algorithm 5.3 *(Message recovery attack)*
Input*: REDOG's public key* pk *and a REDOG's ciphertext* $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2) = \mathit{Encrypt}(\mathbf{m}, \mathsf{pk})$ *generated by the reference implementation.*
Output*:* $\mathbf{m}$

[1] https://www.kpqc.or.kr/images/zip/REDOG.zip

1. *Let C' be the linear $[2n-k,\ell]$-code in the Hamming metric generated by $G = (\mathsf{pk}_1 \mid \mathsf{pk}_2)$.*
2. *Put $f = 0$.*
3. *While $f = 0$:*
   (a) *Randomly select ℓ columns of G to form the matrix A. Let $\mathbf{c}_A$ be the matching positions in $\mathbf{c}$.*
   (b) *If A is invertible*
      i. *Compute $B = A^{-1}$ and $\bar{\mathbf{m}} = \mathbf{c}_A B$.*
      ii. *Compute $\bar{\mathbf{c}}_1 = \bar{\mathbf{m}}\mathsf{pk}_1$.*
      iii. *If $\mathsf{wt}_H(\mathbf{c}_1 - \bar{\mathbf{c}}_1) = t_1 \leq t$*
         A. *Compute $\bar{\mathbf{c}}_2 = \bar{\mathbf{m}}\mathsf{pk}_2$.*
         B. *If $\mathsf{wt}_H(\mathbf{c}_2 - \bar{\mathbf{c}}_2) = t - t_1$*
            Put $\mathbf{m}' = \bar{\mathbf{m}}, \mathbf{e} = (\mathbf{c}_1, \mathbf{c}_2) - (\bar{\mathbf{c}}_1, \bar{\mathbf{c}}_2)$ and $f = 1$.
4. *Compute $\mathbf{m} = \mathbf{m}' - \mathsf{hash}(\mathbf{e})$.*

The inner loop is Prange's information-set decoding algorithm [Pra62] in the generator-matrix form with early aborts. If the chosen ℓ positions are not all error free then $\bar{\mathbf{m}}$ equals $\mathbf{m}$ with one or more rows of B added to it. Then $\bar{\mathbf{m}}\mathsf{pk}_1$ will be random vector and thus differ from $\mathbf{c}_1$ in more than t positions. If the initial check succeeds there is a high chance of the second condition succceeding as well leading to $\mathbf{e}$ with $\mathsf{wt}_H(\mathbf{e}) = t$.

We now analyze the success probability of each iteration of the inner loop of Algorithm 5.3. The field $\mathbb{F}_{q^m}$ is large, hence A very likely to be invertible. The algorithm succeeds if the ℓ positions forming A are chosen outside the positions where $\mathbf{e}$ has non-zero entries. This happens with probability

$$\frac{\binom{2n-k-t}{\ell}}{\binom{2n-k}{\ell}}.$$

Each trial costs the inversion of an $\ell \times \ell$ matrix and up to three matrix-vector products, where the vector has length ℓ and the matrices have ℓ, n, and $n-k$ columns respectively, in addition to minor costs of two vector differences and two weight computations.

We implemented the attack in Algorithm 5.3 in Sagemath 9.5; see Appendix A for the code. We perform faster early aborts, testing $\bar{\mathbf{m}}$ on only $t+3$ columns of pk_1. The probability that a coordinate matches between $\mathbf{c}_1$ and $\bar{\mathbf{c}}_1$ for $\bar{\mathbf{m}} \neq \mathbf{m}$ is q^{-m} and thus negligible for large m. Hence, most candidate vectors $\bar{\mathbf{m}}$ are discared after $(t+3)\ell^2$ multiplications in $\mathbb{F}_{q^m}$. Running the attack on a Linux Mint virtual machine we broke the KAT ciphertexts included in the submssion package for all the proposed parameters. We also generated a bunch of ciphertexts corresponding to randomly chosen public keys and messages and measured the average running time of our algorithm.

As can be seen from Table 3, the attack on outputs of the reference implementation succeeds in few steps and is very fast to execute for all parameter sets.

Security parameter	$\log_2(\mathsf{Prob})$	Time_{KAT} (sec.)	$\text{Time}_{100}(sec.)$
128	-5.62325179726894	~ 8.01	~ 9.17
192	-7.51182199577027	~ 108.13	~ 112
256	-9.40052710879827	~ 167.91	~ 133.43

Table 3. Prob is the probability of success of one iteration of the inner loop of Algorithm 5.3. Time_{KAT} is the average timings of message recovery attack over entries in the KAT file (30 for 128 bits, 15 for 192 bits, 13 for 256 bits). Time_{100} is the average timings of message recovery attack over 100 ciphertext generated by REDOG's encryption.

6 Recomputing attacks costs

In this section we deal with the computation of complexities of general attacks against cryptosystems relying on the rank decoding problem. We noticed that the official REDOG submission [KHL+22a], as well as [LTP21] do not consider attack algorithms proposed in [BBC+20].

Our computations are reported in Tables 4, 5 and 6. The tables show that parameters suggested for REDOG provide significantly less security than expected. The tables also confirm that the parameters do provide the claimed security under attacks prior to [BBC+20] when using a realistic exponent for matrix multiplication. Note that the computations in these tables ignore all constants and lower-order terms in the big-$\mathcal{O}$ complexities. This is in line with how the authors of the attack algorithms use their results to determine the security of other systems, but typically constants are positive and large.

6.1 Overview of rank decoding attacks

Recall that the public code is generated by the $\ell \times 2n - k$ matrix $(M \mid F)$ over $\mathbb{F}_{q^m}$. The error vector added to the ciphertext is chosen to have rank t. In the description of the attacks we will give formulas for the costs using the notation of this paper, i.e., the dimension is ℓ and the error has rank t; we denote the length by N for reasons that will become clear later. The complexity of algorithms in [BBB+20] and [BBC+20] also depends on the matrix multiplication exponent which we denote as ω.

The GRS [GRS16] algorithm is a combinatorial attack on the rank decoding problem. The idea behind this algorithm is to guess a vectorspace containing the space spanned by the error vector. In this way the received vector can be expressed in terms of the basis of the guessed space. The last step is to solve the linear system associated to the syndrome equations. This has complexity

$$\mathcal{O}\left((N-\ell)^3 m^3 q^{\min\{t\lfloor \ell m/N \rfloor, (t-1)\lfloor (\ell+1)m/N \rfloor\}}\right). \tag{5}$$

The second attack, introduced in [GRS16], which we denote GRS-alg, is an algebraic attack. Under the condition that $\ell > \lceil ((t+1)(\ell+1) - N - 1)/t \rceil$ the

decoding problem can be solved in

$$\mathcal{O}\left(t^3\ell^3 q^{t(\lceil((t+1)(\ell+1)-N-1)/t\rceil)}\right). \tag{6}$$

The attack AGHT [AGHT18] is an improvement over the GRS combinatorial attack. The underlying idea is to guess the space containing the error in a specific way that provides higher chance of guessing a suitable space. It has complexity

$$\mathcal{O}\left((N-\ell)^3 m^3 q^{t(\ell+1)m/N-m}\right). \tag{7}$$

The BBB+ attack [BBB+20] translates the rank metric decoding problem into a system of multivariate equations and then uses Gröbner-basis methods to find solutions. Much of the analysis is spent on determining the degree of regularity, depending on the length, dimension, and rank of the code and error. If the condition $m\binom{N-\ell-1}{t}+1 \geq \binom{N}{t}$ is fulfilled then the problem can be solved in

$$\mathcal{O}\left(\left(\frac{((m+N)t)^t}{t!}\right)^{\omega}\right). \tag{8}$$

If the condition is not satisfied then the complexity of solving the decoding problem becomes

$$\mathcal{O}\left(\left(\frac{((m+N)t)^{t+1}}{(t+1)!}\right)^{\omega}\right) \tag{9}$$

or the same for $t+2$ in place of $t+1$. The authors of [BBB+20] use (9) in their calculations and thus we include that as well.

The BBC+-1,BBC+-2,BBC+-3 and BBC+-4 improvements that will follow are all introduced in [BBC+20]. They make explicit the use of extended linearlization as a technique to compute Gröbner bases. For solving the rank-decoding problem it is not necessary to determine the full Gröbner basis but to find a solution to this system of equations. Extended linearlization introduces new variables to turn a mulitvariate quadratic system into a linear system. The algorithms and complexity estimates differ in how large the resulting systems are and whether they are overdetermined or not, dependent on the system parameters.

BBC+-1 applies to the overdetermined case, which matches $m\binom{N-\ell-1}{t}+1 \geq \binom{N}{t}$, and permits to solve the system in

$$\mathcal{O}\left(m\binom{N-\ell-1}{t}\binom{N}{t}^{\omega-1}\right). \tag{10}$$

These costs match matrix computations on a matrix with $m\binom{N-\ell-1}{t}$ rows and $\binom{N}{t}$ columns.

In case of an undetermined system, BBC+-2 is a hybrid attack which fixes some of the unknowns in a brute-force manner to produce to an overdetermined system in the remaining variables. The costs are testing all possible values for j positions, where j is the smallest non-negative integer such that $m\binom{N-\ell-1}{t}+1 \geq$

$\binom{N-j}{t}$, and for each performing the same matrix computations as in BBC on j columns less. This leads to a total complexity of

$$\mathcal{O}\left(q^{jt}m\binom{N-\ell-1}{t}\binom{N-j}{t}^{\omega-1}\right). \tag{11}$$

The brute-force part in BBC+-2 quickly becomes the dominating factor. The BBC+-3 algorithm introduces terms of larger degrees first and then linearizes the system. This consists in multiplying the equations by some homogeneous monomials of degree b so as to obtain a system of homogeneous equations. However, for the special case of $q = 2$ the equations in the system might not be homogeneous. In this case, homogeneous equations coming from smaller values of b are considered. To state the conditions for this and the next algorithms we first introduce some notation from [BBC+20].

$$\begin{aligned}
A_b &:= \sum_{j=1}^{b}\binom{N}{t}\binom{m\ell+1}{j},\\
B_b &:= \sum_{j=1}^{b}\left(m\binom{N-\ell-1}{t}\binom{m\ell+1}{j}\right) \quad \text{and}\\
C_b &:= \sum_{j=1}^{b}\sum_{s=1}^{j}\left((-1)^{i+1}\binom{N}{t+s}\binom{m+s-1}{s}\binom{m\ell+1}{j-s}\right).
\end{aligned}$$

The degree of the equations formed in BBC+-3 depends on b, where $0 < b < 2+t$ is minimal such that $A_b - 1 \leq C_b$ if such a b exists. In this case the problem can be solved with complexity

$$\mathcal{O}\left((m\ell+1)(t+1)A_b^2\right). \tag{12}$$

The BBC+-4 algorithm produces a system of equations in a slightly different way than BBC+-3, where some extra equations are generated by modeling the problem in an additional way. Then as for BBC+-3, the equations are multiplied by monomials of degree b where $0 < b < t+2$ is minimal such that $A_b - 1 \leq B_b + C_b$. The resulting linear system is then solved in

$$\mathcal{O}\left((B_b + C_b)A_b^{\omega-1}\right). \tag{13}$$

The above approaches all need to perform matrix computations on systems that are too dense to be modeled as sparse system. Hence, the typical choices to consider for ω are $\omega = 2.807$ matching Strassen's algorithm which is a realistic choice given the sizes encountered, and $\omega = 2.37$ for asymptotically faster algorithms such as Coppersmith-Vinograd or more recent advances such as [DWZ22]. Note that none of these algorithms achieves exponent 2.37 but rather 2.371866 and that multiplicative constants are typically not known but very large.

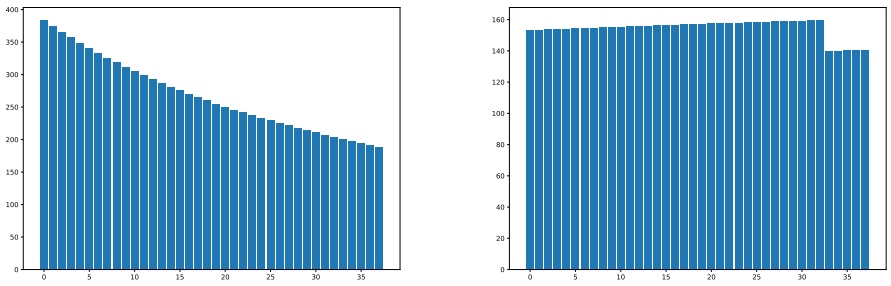

Fig. 1. Plots showing the $\log_2$ of the costs for AGHT and BBB+ for the parameters at the 128–bit security level for different choices of code length.

As a final algorithm, [BBC+20] presents an approach that produces a larger but sparse system of equations which can be solved using Wiedemann's algorithm [Wie86] which implies $\omega = 2$. The BBC+-Wiedemann algorithm will turn out to produce the lowest attack costs for some of the parameter sets suggested for REDOG. For b fulfilling the condition $A_b - 1 \leq B_b + C_b$ the problem can be solved with complexity

$$\mathcal{O}\left(D_b A_b^2\right), \tag{14}$$

where

$$D_b := \frac{B_b\binom{\ell+t+1}{t} + C_b(m\ell+1)(r+1)}{B_b + C_b}.$$

6.2 Lowering the attack costs beyond the formulas stated

The combinatorial attacks GRS and AGHT perform best for longer codes, however, algebraic attacks that turn each column into a new variable perform best with fewer variables. For each attack strategy we search for the best number of columns that we should consider in order to obtain the cheapest cost of a successful break of REDOG. This is why we presented the above formulas using N rather than the full length of the code $2n-k$. The conditions given above determine the minimum length required relative to dimension and rank of the error.

We then evaluate the costs for each algorithm for each choice of length $N = \ell + t + i$, for every value of $i = 0, 1, \ldots, 2n - k - \ell - t$ satisfying the conditions of the attacks. In case multiple versions of the algorithm apply, namely for BBC+2 - BBC+4 in the non-overdetermined case, we choose the lowest costs over the options.

Figure 1 shows the different behavior of the algorithms for fixed ℓ and t and increasing i. The jump in the BBB+ plot is at the transition between the two formulas.

We point out that [BBC+20] also considered decreasing the length of the code for the case of overdetermined systems, see [BBC+20, Section 4.2] on puncturing the code in the case of "super"-overdetermined systems. We perform a systematic scan for all algorithms as an attacker will use the best possible attack.

6.3 The recomputed values

We computed complexity costs for all the attacks introduced in the previous subsection, taking into consideration two values of matrix multiplication exponent, namely $\omega = 2.807$ and $\omega = 2.37$. For each possible length $N+i$ for $N = \ell+t$ and $i = 0, 1, \ldots, 2n-k-\ell-t$ we computed the costs for each attack strategy, keeping the lowest value per strategy. For the two cases of BBB+ and the four strategies described for the BBC+-* algorithms, we selected the best complexity among them. For the sake of completeness, we report the value of i in Tables 4–6 as well. All the values are stated as the $\log_2$ of the costs resulting from the complexity formulas. The lowest costs of a realistic algorithm are stated in blue. Note the above-mentioned caveats regarding evaluating big-$\mathcal{O}$ estimates for concrete parameters.

Algorithm	Complexity formula	$\log_2$ of cost			i
		general	$\omega = 2.807$	$\omega = 2.37$	
GRS [GRS16]	5	230	-	-	36
GRS-alg [GRS16]	6	209	-	-	37
AGHT [AGHT18]	7	189	-	-	37
BBB+ [BBB+20]	8 and 9		140	118	33
BBC+ [BBC+20]	10,11,12 and 13		78	66	33
BBC+-Wiedemann [BBC+20]	14	87	-	-	7

Table 4. Values of the $\log_2$ of attack costs for REDOG's suggested parameters for 128-bit security (see Table 1). Costs of attacks not depending on ω are reported in column "general".

As shown in the tables, for all security levels the suggested parameters for REDOG do not resist attacks BBC+ and BBC+-Wiedemann for any choice of ω, while they do resist BBB+ for $\omega = 2.807$. In Section 9 we propose a solution to the decryption failures that also boosts the security of the suggested parameters of REDOG.

7 Further results on attacking REDOG

In this section we consider the key-recovery attack, as stated in [LTP21, Section 4] and show that it does not apply in the full generality as claimed but only works for $P \in \mathsf{GL}_n(\mathbb{F}_q)$ while [LTP21] states it for P being an isometry. This is

Algorithm	Complexity formula	$\log_2$ of cost			i
		general	$\omega = 2.807$	$\omega = 2.37$	
GRS [GRS16]	5	395	-	-	48
GRS-alg [GRS16]	6	370	-	-	49
AGHT [AGHT18]	7	340	-	-	49
BBB+ [BBB+20]	8 and 9		210	178	0
BBC+ [BBC+20]	10,11,12 and 13		125	106	8
BBC+-Wiedemann [BBC+20]	14	107	-	-	8

Table 5. Values of the $\log_2$ of attack costs for REDOG's suggested parameters for 192-bit security (see Table 1). Costs of attacks not depending on ω are reported in column "general".

Algorithm	Complexity formula	$\log_2$ of cost			i
		general	$\omega = 2.807$	$\omega = 2.37$	
GRS [GRS16]	5	607	-	-	60
GRS-alg [GRS16]	6	578	-	-	61
AGHT [AGHT18]	7	539	-	-	61
BBB+ [BBB+20]	8 and 9		270	227	0
BBC+ [BBC+20]	10,11,12 and 13		151	128	10
BBC+-Wiedemann [BBC+20]	14	129	-	-	10

Table 6. Values of the $\log_2$ of attack costs for REDOG's suggested parameters for 256-bit security (see Table 1). Costs of attacks not depending on ω are reported in column "general".

another indication that the authors may not have understood the full generality of isometries, see also Remark 4.8.

The key-recovery attack is described for $S \in \mathsf{GL}_{n-k}(\mathbb{F}_q)$ and is the main reason for choosing S from a larger set. The attack computes an alternative parity-check matrix for the secret Gabidulin code by solving a system of equations. This matrix can be used in place of the secret key to decrypt any ciphertext The attack relies on the fact that, given M, the second part of the public key, i.e. $F = MP^{-1}H_1^T \left(H_2^T\right)^{-1} S$, can be written as a system of linear equations over $\mathbb{F}_q$ in terms of the coordinates of $H_1' = H_1(P^{-1})^T$ and $H_2' = H_2(S^{-1})^T$. If H_1' and H_2' are Moore matrices the number of unknowns is reduced so that the system becomes overdetermined and [LTP21] states that they are Moore matrices without giving a proof. The same claim is used in the plaintext-recovery attack in [LTP21, Section 5, Proposition 1] to show that the public code is a subcode of a Gabidulin code.

We show that $H_1(P^{-1})^T$ is a Moore matrix if $P \in \mathsf{GL}_n(\mathbb{F}_q)$ but not in the general case where P is an isometry. The first part also shows that $H_2(S^{-1})^T$ is a Moore matrix for $S \in \mathsf{GL}_n(\mathbb{F}_q)$

The following lemma shows that the product of a $k \times n$ Moore matrix A and an isometry P is an $\mathbb{F}_{q^m}$-multiple of a Moore matrix.

Lemma 7.1. *Let A be a $k \times n$ Moore matrix over $\mathbb{F}_{q^m}$ and let P be an isometry. Then $AP = \gamma B$ for $\gamma \in \mathbb{F}_{q^m}^*$ such that $P = \gamma Q$ for $Q \in \mathsf{GL}_n(\mathbb{F}_q)$ and B a $k \times n$ Moore matrix. This representation is unique up to $\mathbb{F}_q^*$ factors in γ and Q.*

Proof. Theorem 2.5 states that P is of the form $P = \gamma Q$ for some $\gamma \in \mathbb{F}_{q^m}^*$ and $Q \in \mathsf{GL}_n(\mathbb{F}_q)$. Any other factorization $P = \gamma' Q'$ satisfies $(\gamma/\gamma')I_n = Q'Q^{-1}$ which implies $\gamma/\gamma' \in \mathbb{F}_q^*$.

For this choice of γ write $AP = \gamma AQ$. It is then enough to prove that AQ is a Moore matrix. Write $Q = (Q_{i,j})$ for $i, j = 1, \ldots, n$. Let A_ℓ be the ℓ-th row of A and write $A_1 = (a_1, \ldots, a_n)$. Then $A_\ell = \left(a_1^{q^{\ell-1}}, \ldots, a_n^{q^{\ell-1}}\right)$ for $\ell = 1, \ldots, k$. We can write the entry $(AQ)_{i,j}$ as

$$(AQ)_{i,j} = \sum_{h=1}^{n} a_h^{q^{i-1}} Q_{h,j} = \sum_{h=1}^{n} (a_h Q_{h,j})^{q^{i-1}} = \left(\sum_{h=1}^{n} a_h Q_{h,j}\right)^{q^{i-1}} = (AQ)_{1,j}^{q^{i-1}}$$

for $i = 1, \ldots, k$ and $j = 1, \ldots, n$ using that $Q_{h,j} \in \mathbb{F}_q$ and that q-th powers are linear over $\mathbb{F}_{q^m}$. This proves that AQ is a Moore matrix. Setting $B = AQ$ we obtain the result in the statement. □

For $\gamma = 1$ we get the following corollary.

Corollary 7.2. *If $P = \mathsf{GL}_n(\mathbb{F}_q)$ and A is a $k \times n$ Moore matrix then AP is a Moore matrix.*

The attack proposed in [LTP21, Section 4.1] claims that given a $k \times n$ Moore matrix A over $\mathbb{F}_{q^m}$ and an isometry P, there exists a Moore matrix A' such that $A' = AP$. However, this statement is false except for the special case in Corollary 7.2.

Lemma 7.3. *Under the conditions of Lemma 7.1, AP is not a Moore matrix unless $P \in \mathsf{GL}_n(\mathbb{F}_q)$, or $A = 0$, or $k = 1$.*

Proof. By Lemma 7.1 $AP = \gamma B$ for $P = \gamma Q$ with $Q \in \mathsf{GL}_n(\mathbb{F}_q), \gamma \in \mathbb{F}_{q^m}^*$, and B a $k \times n$ Moore matrix. If $A = 0$ then $B = 0$ and the result holds trivially. Similarly, if $k = 1$ then AP has only a single row so that there is no constraint.

As P is invertible, $A \neq 0$ implies $B \neq 0$. Let B_ℓ denote the ℓ-th row of B. As B is a Moore matrix, we have for $B_1 = (b_1, \ldots, b_n)$ that $B_\ell = \left(b_1^{q^{\ell-1}}, \ldots, b_n^{q^{\ell-1}}\right)$ for $\ell = 1, 2, \ldots, k$. Because $B \neq 0$ there is at least one b_i that is nonzero.

The matrix γB for $\gamma \in \mathbb{F}_{q^m}^*$ has rows $\gamma B_1 = (\gamma b_1, \ldots, \gamma b_n)$ and $\gamma B_\ell = \left(\gamma b_1^{q^{\ell-1}}, \ldots, \gamma b_n^{q^{\ell-1}}\right)$ for $\ell = 1, 2, \ldots, k$. Hence, γB is a Moore matrix if and only if $\gamma b_i^{q^{\ell-1}} = (\gamma b_i)^{q^{\ell-1}}$ for $i = 1, 2, \ldots, n$ and $\ell = 1, 2, \ldots, k$. For $k > 1$ and $b_i \neq 0$ this holds if and only if $\gamma = \gamma^q$, which is equivalent to $\gamma \in \mathbb{F}_q$. Hence, $\gamma \in \mathbb{F}_q^*$ and thus $P = \gamma Q \in \mathsf{GL}_n(\mathbb{F}_q)$. □

The key recovery attack in [LTP21] builds a system of linear equations from the public key $(M \mid F)$ with $F = MP^{-1}H_1^T \left(H_2^T\right)^{-1} S$ rewriting it as

$$F\left(H_2(S^{-1})^T\right)^T = M\left(H_1\left(P^{-1}\right)^T\right)^T, \tag{15}$$

where the entries of $(H_1', H_2') = \left(H_1\left(P^{-1}\right)^T \mid H_2\left(S^{-1}\right)^T\right)$ are considered as unknowns and M and F are known. At first sight this is a system of $\ell(n-k)$ equations in $(n-k)(2n-k)$ variables and ℓ is much less than $2n-k$, hence, the system is severely underdetermined. Considering the system over $\mathbb{F}_q$ instead of over $\mathbb{F}_{q^m}$ just multiplies the number of equations and the number of variables by m. This is where [LTP21] uses that H_1' and H_2' are Moore matrices. From the above considerations this holds for H_2' but not for H_1'. The next proposition shows that the $\mathbb{F}_q$-linear system of equations obtained by the public key of REDOG is underdetermined if $P \notin \mathsf{GL}_n(\mathbb{F}_q)$, even if $S \in \mathsf{GL}_{n-k}(\mathbb{F}_q)$.

Proposition 7.4. *Let M be a full-rank $\ell \times n$ matrix over $\mathbb{F}_{q^m}$, $(H_1 \mid H_2)$ a $(n-k) \times (2n-k)$ Moore matrix over $\mathbb{F}_{q^m}$, P a rank-metric isometry of the space $\mathbb{F}_{q^m}^n$, and $S \in \mathsf{GL}_{n-k}(\mathbb{F}_q)$. Consider the linear system of equations defined by $FS^{-1}H_2^T = MP^{-1}H_1^T$, where the entries of $\left(H_1\left(P^{-1}\right)^T \mid H_2\left(S^{-1}\right)^T\right)$ are considered as unknowns. Use an explicit basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$ to turn this into a system of linear equations over $\mathbb{F}_q$. For $P \notin \mathsf{GL}_n(\mathbb{F}_q)$ this system has $m\ell(n-k)$ equations and $m^2n + m(n-k)$ variables.*

Proof. By Lemma 7.1, $H_1\left(P^{-1}\right)^T$ is an $\mathbb{F}_{q^m}$-multiple of a $(n-k) \times n$ Moore matrix, say γB, with $\gamma \in \mathbb{F}_{q^m}^* \setminus \mathbb{F}_q^*$ unless $P \in \mathsf{GL}_n(\mathbb{F}_q)$. Moreover, $H_2(S^{-1})^T$ is an $(n-k) \times (n-k)$ Moore matrix by Corollary 7.2.

Consider the right-hand side of (15). Fix a normal basis $\{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Write $\gamma = \sum_{i=1}^m \gamma_i \alpha^{q^{i-1}}$ with $\gamma_i \in \mathbb{F}_q$. Let $(b_1, \ldots, b_n)$ be the first row of B, and write its h-th coordinate as $b_h = \sum_{i=1}^m b_{h,i}\alpha^{q^{i-1}}$ with $b_{h,i} \in \mathbb{F}_q$. Then

$$\gamma b_h = \left(\sum_{i=1}^m \gamma_i \alpha^{q^{i-1}}\right)\left(\sum_{j=1}^m b_{h,j}\alpha^{q^{j-1}}\right) = \sum_{i,j=1}^m \gamma_i b_{h,j}\alpha^{q^{i-1}+q^{j-1}}.$$

The h-th entry of the ℓ-th row of γB becomes

$$\gamma\left(b_h\right)^{q^{\ell-1}} = \sum_{i,j=1}^m \gamma_i b_{h,j}\alpha^{q^{i-1}+q^{j+\ell-2}}.$$

By setting the variables as $x_{i,h,j} = \gamma_i b_{h,j}$, we obtain m^2n variables from the $M\left(H_1\left(P^{-1}\right)^T\right)^T$ part of the system. If $\gamma \in \mathbb{F}_q$ then $\gamma_1 = \gamma_2 = \cdots = \gamma_m$ and the m^2 term collapses to m. For a general isometry P no such assumptions can be made.

Consider now the left hand side of (15). By following the same steps as above, but now using that $H_2\left(S^{-1}\right)^T$ is a Moore matrix, there are $m(n-k)$ unknowns for the equations corresponding to the first row of $F\left(H_2(S^{-1})^T\right)^T$ part, and all the other equations will share the same unknowns. □

For the parameters used in REDOG, $m > 2n-k$ and $n > \ell$. By Proposition 7.4 there are $u = m(mn-(\ell-1)(n-k)) > mn$ more variables than equations. Solving such an underdetermined system of equations requires trying all $q^u > (q^m)^n$ possibilities which makes the attack complexity exponential.

Remark 7.5. Note also that since REDOG considers $S \in \mathsf{GL}_{n-k}(\mathbb{F}_{q^m})$ (via $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$), these attacks could not be automatically applied even if P were limited to $\mathsf{GL}_n(\mathbb{F}_q)$. This can be shown by slightly tweaking the proof of Proposition 7.4 to having the left side of (15) require more variables. Note that in general the λ-dimensional space Λ is not invariant under the q-power Frobenius map, leading to the full $m^2(n-k)$ variables instead of $m(n-k)$ in Proposition 7.4. However, if $\mathbb{F}_{q^m}$ has m divisible by λ then it is possible to choose $\Lambda \cong \mathbb{F}_{q^\lambda}$ which would lead to only $\lambda m(n-k)$ variables for that part. The 128- and 192-bit parameters for REDOG chose prime m but the 256-bit parameters have $m = 135 = 3^3 \cdot 5$ and $\lambda = 3$, permitting this choice. While it is unlikely that a random choice of Λ picks this case and P being a general isometry also avoids having too few variables, we recommend choosing prime values of m to completely rule out this concern.

Remark 7.6. Despite Proposition 7.4 showing that the mentioned attacks do not apply to the case where P is an isometry of $\mathbb{F}_{q^m}^n$ and $S^{-1} \in \mathsf{GL}_{n-k}(\mathbb{F}_q)$, we do not advise to choose this combination to instantiate REDOG.

8 Solving decryption failures

The key point of REDOG's decryption failures is given by point (3) of the proof of Theorem 4.6. Indeed, the crucial step for showing decoding failure of the decoder Φ, is that $\langle \mathbf{e}_1 P^{-1} \rangle \not\subset \langle -\mathbf{e}_2 S^{-1} \rangle$.

In order to solve the issue of decryption failures in REDOG, we propose an alternative that keeps the random choice of an error vector $\mathbf{e}$ with $\mathsf{wt}_R(\mathbf{e}) = t$ and changes the public key. The idea is to retain the method introduced in [Loi17], but also to make sure that $\mathsf{wt}_R(\mathbf{e}') \leq \lambda t$. We suggest to pick $P^{-1} \in \mathsf{GL}_n(\Lambda)$ randomly instead of it being an isometry of the space $\mathbb{F}_{q^m}^n$.

The proof of the next result is an adaptation of the proof of Proposition 4.1.

Proposition 8.1. *Let $\Lambda \subset \mathbb{F}_{q^m}$ be a λ-dimensional subspace of $\mathbb{F}_{q^m}$ and $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ a random vector with $\mathsf{wt}_R(\mathbf{e}) = t$ with $\mathbf{e}_1 \in \mathbb{F}_{q^m}^n$ and $\mathbf{e}_2 \in \mathbb{F}_{q^m}^{n-k}$. Let $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ and $P^{-1} \in \mathsf{GL}_n(\Lambda)$. Then*

$$\langle \mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1} \rangle \subseteq V \tag{16}$$

for some λt-dimensional $\mathbb{F}_q$-linear vectorspace V.

Proof. Let $\Gamma = \langle \mathbf{e} \rangle$ be the $\mathbb{F}_q$-linear subspace of $\mathbb{F}_{q^m}$ generated by $\mathbf{e}$. As before we can write $\Gamma = \langle y_1, \ldots, y_t \rangle$. Write also $\Lambda = \langle \alpha_1, \ldots, \alpha_\lambda \rangle$. As in the proof of Proposition 4.1 we can express the j-th coordinate of $\mathbf{e}_1 P^{-1}$ as a linear combination of the λt elements $y_h \alpha_k$ for $h = 1, \ldots, t$ and $k = 1, \ldots, \lambda$ as

$$(\mathbf{e}_1 P^{-1})_j = \sum_{h=1}^{t} \sum_{k=1}^{\lambda} c_{h,k} y_h \alpha_k.$$

The same can be done for each coordinate of $-\mathbf{e}_2 S^{-1}$. Hence both subspaces are contained in the space $V = \langle y_h \alpha_k \rangle$ generated by these λt elements. □

Remark 8.2. The condition in (16) is a relaxation of $\langle \mathbf{e}_1 P^{-1} \rangle \subset \langle -\mathbf{e}_2 S^{-1} \rangle$ implied by point (3) in the proof of Theorem 4.6. Indeed, what we will require is that the space spanned by the entries of $\mathbf{e}_1 P^{-1}$ and $-\mathbf{e}_2 S^{-1}$ has dimension at most λt, which will correspond to a decodable error, as stated in the next corollary.

Corollary 8.3. *Let* $\mathbf{e}' = \left(\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1}\right)$ *with* $\mathbf{e}, P^{-1}$ *and* S^{-1} *as in Proposition 8.1. Then*

$$\mathsf{wt}_R(\mathbf{e}') \leq \lambda t.$$

Proof. Since $\langle \mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1} \rangle$ is a subspace of a vectorspace V of dimension λt by Proposition 8.1, then $\mathsf{wt}_R(\mathbf{e}') = \mathsf{wt}_R\left(\left(\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1}\right)\right) \leq \lambda t$.

The only change to the specification of REDOG is in the KeyGen algorithm in Step 3; encryption and decryption remain unchanged as in Section 3. The specification of the updated version of REDOG with no decryption failures follows.

KeyGen:

1. Select $H = (H_1 \mid H_2)$, $H_2 \in \mathsf{GL}_{n-k}(\mathbb{F}_{q^m})$, a parity check matrix of a $[2n-k, n]$ Gabidulin code, with syndrome decoder Φ correcting r errors.
2. Select a full rank matrix $M \in \mathbb{F}_{q^m}^{\ell \times n}$.
3. Select a λ-dimensional subspace $\Lambda \subset \mathbb{F}_{q^m}$, seen as $\mathbb{F}_q$-linear space, and select $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ and $P^{-1} \in \mathsf{GL}_n(\Lambda)$.
4. Compute $F = MP^{-1}H_1^T \left(H_2^T\right)^{-1} S$ and publish the public key $\mathsf{pk} = (M, F)$.
5. Store the secret key $\mathsf{sk} = (P, H, S, \Phi)$.

Theorem 8.4. *The updated version of REDOG is correct.*

Proof. The correctness of the updated version of REDOG follows from the correctness of the original version, except for decryption correctness, which is proven by Corollary 8.3.

9 Solving decryption failures and boosting security

Our second idea of how to deal with REDOG not decrypting correctly is to change how $\mathbf{e}$ is sampled. While the approach in Section 8 works and preserves all considerations regarding parameter sizes, in Section 6 we have shown that these are too small to offer security against the best known attacks. The approach in this section provides a functioning system and increases the security offered by the parameters.

Recall that the public key is $(M \mid F)$, where M has dimension $\ell \times n$ and F has dimension $\ell \times (n-k)$ and both, M and F, have full rank. The relative sizes in REDOG are such that $n - k = \ell - 1$, so F is just one column short of being square, and $n = \ell + t + 1$. The parameters are chosen so that the decryption step can decode errors of rank up to r, while encryption in REDOG adds only an error vector of rank t with $r \geq t\lambda$. All parameter sets have $\lambda = 3$ and $r = \lambda t = (n-k)/2$.

Encryption is computed as $\mathbf{c} = \mathbf{m}'(M \mid F) + \mathbf{e}$, for $\mathbf{m}' \in \mathbb{F}_{q^m}^{\ell}$. Decryption requires decoding in the Gabidulin code for error $(\mathbf{e}_1 P^{-1}, -\mathbf{e}_2 S^{-1})$, where P is an isometry and $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$. We have shown in Theorem 4.6 that this $\mathbf{e}'$ typically has rank larger than r, which causes incorrect decoding, for REDOG's choice of $\mathbf{e}$ with $\mathsf{wt}_R(\mathbf{e}) = t$. Where we proposed changing the definition of P in the previous section to reach a system which has minimal changes compared to REDOG, we now suggest changing the way that $\mathbf{e}$ is chosen.

In particular, we redefine $\mathbf{e}$ to have different rank on the first n positions and the last $n-k$ positions. Let $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2)$ with $\mathsf{wt}_R(\mathbf{e}_1) = t_1$ and $\mathsf{wt}_R(\mathbf{e}_2) = t_2$. This can be achieved by sampling t_1 random elements from $\mathbb{F}_{q^m}$, testing that this achieves rank t_1 and taking the n positions in $\mathbf{e}_1$ as random $\mathbb{F}_q$-linear combinations of these t_1 elements. Because m is significantly larger than t_1, this finds an $\mathbf{e}_1$ of rank t_1 on first try with high probability. Similarly, we pick t_2 random elements from $\mathbb{F}_{q^m}$ and use their $\mathbb{F}_q$-linear combinations for $\mathbf{e}_2$.

We keep P being an isometry and $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ as in REDOG. Then the decoding step needs to find an error of rank $t_1 + \lambda t_2$, namely $\mathbf{e}_1 P^{-1}$ on the first n positions and $\mathbf{e}_2 S^{-1}$ on the last $n-k$ positions. This will succeed if

$$r \geq t_1 + \lambda t_2. \tag{17}$$

Hence, we can consider different splits of r to maximize security.

9.1 Considerations for extreme choices of t_1 and t_2

As already explained in Section 6.2, the attacker can consider parts of $\mathbf{c}_1$ and $\mathbf{c}_2$, for example, the extreme choice of $t_1 = 0$ would mean that $\mathbf{c}_1$ is a codeword in the code generated by M and thus $\mathbf{m}'$ would be trivially recoverable from $\mathbf{c}_1 = \mathbf{m}'M$ by computing the inverse of an $\ell \times \ell$ submatrix of M. Because $\mathbb{F}_{q^m}$ is large, almost any choice of submatrix will be invertible.

The other extreme choice, $t_2 = 0$, does not cause such an obvious attack as for the REDOG parameters F has one column fewer than it has rows, meaning that

$\mathbf{c}_2 = \mathbf{m}'F$ cannot be solved for $\mathbf{m}'$. Hence, at least one position of $\mathbf{c}_1$ needs to be included, but that means that we do not have a codeword in the code generated by that column of M and F but a codeword plus an error of rank 1. However, a brute-force attack on this system still succeeds with cost q^m as follows:

Let $\bar{F} = (M_i|F)$ be the square matrix obtained from taking M_i, the i-th column of M, for a choice of i that makes $\bar{F}$ invertible. Most choices of i will succeed. Let $\bar{\mathbf{c}} = (c_{1i}, \mathbf{c}_2)$, the i-th coordinate of $\mathbf{c}_1$ followed by $\mathbf{c}_2$.

For each $a \in \mathbb{F}_{q^m}$ compute $\bar{\mathbf{m}} = (\bar{\mathbf{c}} - (a, 0, 0, \ldots, 0))\bar{F}^{-1}$. Then compute $\bar{\mathbf{e}} = \mathbf{c} - \bar{\mathbf{m}}(M \mid F)$ and check if $\mathsf{wt}_R(\bar{\mathbf{e}}_1) = t_1$. If so put $\mathbf{m}' = \bar{\mathbf{m}}$ and $\mathbf{e} = \bar{\mathbf{e}}$.

The matrix operations in this attack are cheap and can be made even cheaper by observing that $\bar{\mathbf{m}} = \bar{\mathbf{c}}\bar{F}^{-1} - a\mathbf{f}$, for $\mathbf{f}$ the first *row* of $\bar{F}^{-1}$, and $\bar{\mathbf{e}} = \mathbf{c} - (\bar{\mathbf{c}}\bar{F}^{-1})(M \mid F) + a\mathbf{f}(M \mid F)$, where everything including $\mathbf{f}(M \mid F) \in \mathbb{F}_{q^m}^{2n-k}$ is fixed and can be computed once per target $\mathbf{c}$. Note also that only the $\mathbf{c}_1$ and $\mathbf{e}_1$ parts need to be computed as by construction $\mathbf{e}_2 = 0$. This leaves just n multiplications and additions in $\mathbb{F}_{q^m}$ and the rank computation for each choice of a. The search over $a \in \mathbb{F}_{q^m}$ is thus the main cost for a complexity of q^m. For all parameters of REDOG this is less than the desired security.

9.2 Generalizations of the brute-force attack

For $t_1 = 1$, a brute-force attack needs to search over all $a \in \mathbb{F}_{q^m}$, up to scaling by $\mathbb{F}_q$-elements, and over all choices of error patterns, where each position of the error is a random $\mathbb{F}_q$-multiple of a. We need ℓ positions from $\mathbf{c}_1 = \mathbf{m}'M + \mathbf{e}_1$ to compute a candidate $\bar{\mathbf{m}}'$ as in the attack on $t_1 = 0$. Hence, for each $a \in \mathbb{F}_{q^m}$ we need to try at most the q^ℓ patterns for those ℓ positions of $\mathbf{e}_1$ for a cost of $(q^m - 1)q^\ell/(q-1)$. For the REDOG parameters, $q = 2$ and $m + \ell$ is significantly smaller than the security level. Hence, $t_1 = 1$ is also a bad choice.

Starting at $t_1 = 2$, when there are two elements $a, b \in \mathbb{F}_{q^m}$ and error patterns need to consider random $\mathbb{F}_q$-linear combinations of these two elements, the attack costs of $(q^m - 1)(q^m - 2)q^{2\ell}/(2(q-1)^2)$ grow beyond the more advanced attacks considered in Section 6.2.

Lemma 9.1. *In general, the brute-force attack on the left side takes*

$$\binom{q^m - 1}{t_1} q^{t_1 \ell}/(q-1)^{t_1}$$

steps.

Proof. The error vector on the left, $\mathbf{e}_1$, has rank t_1, this means that there are t_1 elements $a_1, a_2, \ldots, a_{t_1} \in \mathbb{F}_{q^m}$ which are $\mathbb{F}_q$-linearly independent. There are $\binom{q^m-1}{t_1}/(q-1)^{t_1}$ such choices up to $\mathbb{F}_q$ factors.

Each of the ℓ positions takes a random $\mathbb{F}_q$-linear combination. For a fixed choice of the a_i there are $q^{t_1\ell}$ choices for these linear combinations. Combining these quantities gives the result. □

Similarly, for $t_2 = 1$ the brute-force attack is no longer competitive, yet less clearly so than for $t_1 = 2$ because a and b appear in separate parts. There are q^m candidate choices for e_{1i} and $(q^m - 1)q^{\ell-1}/(q-1)$ candidates for $\mathbf{e}_2$. For $q = 2$ this amounts to roughly $2^{2m+\ell-1}$ and $2m + \ell - 1$ is larger than the security level for all parameters in REDOG.

Lemma 9.2. *In general, the brute-force attack on the right side takes*

$$q^m \binom{q^m - 1}{t_2} q^{t_2(\ell-1)}/(q-1)^{t_2}$$

steps.

Proof. There are q^m choices for e_{1i}. The result follows by the same arguments as for Lemma 9.1, and taking into account that $\mathbf{e}_2$ has length $\ell - 1$. □

We do not consider other combinations of columns from the left and right as those would lead to higher ranks than these two options. Depending on the sizes of t_1 and t_2, Lemma 9.1 or 9.2 gives the better result, but apart from extreme choices these costs are very high.

9.3 Finding good choices of t_1 and t_2

We now turn to the more sophisticated attacks and try to find optimal splits of the decoding budget r between t_1 and t_2 satisfying (17), to make the best attacks as hard as possible. For any such choice, we consider attacks starting from the left with (parts of) $\mathbf{c}_1$ and M or from the right with $\mathbf{c}_2, F$, and parts of $\mathbf{c}_1$ and M. The attacks and sub-attacks differ in how many columns they require, depending on the dimension and rank, and we scan the whole range of possible lengths from both sides.

Since $n = \ell + t + 1$, for the t parameter in REDOG, for small choices of $t_1 \leq t$ the attack may take a punctured system on $\mathbf{c}_1$ and M to recover $\mathbf{m}'$, similar to the attacks considered in Section 6, or include part of $\mathbf{c}_2$ and F, while accepting an error of larger rank including part of t_2. Hence, the search from the left may start with puncturing of $\mathbf{c}_1$. Once parts of $\mathbf{c}_2$ are included, the rank typically increases by one for each extra position, again because m is much larger than t_1 and t_2, until reaching $t_1 + t_2$, after which the rank does not increase with increasing length.

If $t_1 > t + 1$ parts of $\mathbf{c}_2$ need to be considered in any case, with the corresponding increases in the rank of the error, in turn requiring more positions to deal with the increased rank, typically reaching $t_1 + t_2$ before enough positions are available.

Starting from the right, the attacker will always need to include parts from $\mathbf{c}_1$ to even have an invertible system. Hence, the attack is hardest for t_1 maximal in (17) provided that the brute-force attack is excluded. This suggests choosing $t_2 = 1, t_1 = r - \lambda$, as then the attacker is forced to decode an unstructured code with an error of rank $t_1 + t_2 = r - \lambda + 1$.

parameter set	$\log_2(\text{cost})$	$N+i$	t_1	t_2	m	n	k	ℓ
128-bit	128.3841	55	15	1	83	44	8	37
192-bit	163.9988	73	21	1	109	58	10	49
256-bit	199.0283	91	27	1	135	72	12	61

Table 7. Best parameter choices and achieved security using the original values for ℓ, k, m, and n and splitting the decoding capacity r according to (17). The stated costs are achieved by BBC+-Wiedemann at length $N+i$.

A computer search, evaluating all attacks considered in Section 6 for all choices of $t_2 \in \{1, 2, \ldots, r/\lambda - 1\}$ and considering both directions as starting points for the attacker confirms that $t_2 = 1$ is optimal. See Appendix B for the Sage code used for the search. For this choice, the best attack turned out to be BBC+-Wiedemann for all three sizes. The original parameters choices for REDOG then provide the attack costs in Table 7.

This means that the parameter set for 128-bit security, combined with this second idea, provides enough security. However, the parameter sets for 192 and 256-bits are not guaranteed to provide enough security.

Note that, as pointed out before, these computations use big-$\mathcal{O}$ complexity estimates and put all constants to 1 and lower-order terms to 0. This is in line with how estimates are presented in the papers introducing BBB+ [BBB+20] and BBC+ [BBC+20] but typically underestimates the security. However, it is unlikely that this underestimate can close the gaps of 29 and 57 bits for levels 192 and 256, respectively.

Remark 9.3. After we developed this idea but before posting it, the REDOG authors informed us that they fixed the decryption issue in a manner similar to the approach in this section, namely by having different ranks for $\mathbf{e}_1$ and $\mathbf{e}_2$. Their choice of $t_1 = r/2$ and $t_2 = r/(2\lambda)$ satisfies (17) but provides less security against attacks.

The Sage script in Appendix B gives the results in Table 8 as a byproduct of computing the costs for all values of t_2. For each parameter set, again BBC+-Wiedemann is the best attack.

10 Conclusions and further considerations

In this paper we showed several issues with the REDOG proposal but also some ways to repair it. One other issue is that REDOG has rather large keys for a

Intended security in bits	128	192	256
Achieved security in bits	114.8425	143.3622	173.2914
Number of columns $(N+i)$	53	69	86

Table 8. Results for the modified parameter for REDOG using $t_1 = t/2$ and $t_2 = r/(2\lambda)$. The stated costs are achieved by BBC+-Wiedemann at length $N+i$.

rank-metric-based system. A strategy used by many systems in the NIST post-quantum competition is to generate parts of the secret and public keys from seeds and storing or transmitting those seeds instead of the matrices they generated. Implementations written in C always need to define ways to take the output of a random-number generator and this strategy includes the use of a fixed such generator into the KeyGen, encryption, and decryption steps. For REDOG, this approach permits to reduce the size of the secret key sk and, at the same time, moderately shrink the size of the public key pk.

Let $f : \{0,1\}^{256} \to \{0,1\}^*$ be such a generator, where $\{0,1\}^*$ indicates that the output length is arbitrary, in a use of f the output length N must be specified. Most recent proposals use SHAKE-256 or SHAKE-512. The idea is to pick a random 256-bit seed s and initialize f with this seed, the output bits of $f(s)$ are then used in place of the regular outputs of the random-number generator to construct elements of the public or secret key. This method is beneficial if s is much smaller than the key element it replaces. The downside is that any use of that key element then incurs the costs of recomputing that element from s.

As one of the more interesting cases, we show how to build the isometry P form $f(s)$ for some seed s. Let $(n, k, \ell, q, m, \lambda)$ denote the same quantities as in REDOG.

Example 10.1. Let $N = (n^2+m)\lceil \log_2(q) \rceil + 256$ and let $\{\alpha_1, \ldots, \alpha_m\}$ be a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Choose a random seed s and produce the N-bit string $f(s)$. Use the first $n^2\lceil \log_2(q) \rceil$ bits of $f(s)$ to determine n^2 elements in $\mathbb{F}_q$ and build an $n \times n$ matrix Q with these elements. The matrix Q is invertible with probability roughly 0.29. If this is not the case, use the last 256 bits of the output as a new seed s', discard s, and repeat the above with $f(s')$ (an average of 3 trials produces an invertible matrix).

Once an invertible Q has been constructed, use the middle $m\lceil \log_2 q \rceil$ bits of $f(s)$ to define m coefficients in $\mathbb{F}_q$ and to determine an element $\gamma \in \mathbb{F}_{q^m}$ as the $\mathbb{F}_q$-linear combination of the α_i. Then compute $P = \gamma Q$ which, by Theorem 2.5 is an isometry for the rank metric.

As a second example we show how to select S.

Example 10.2. We first observe that $\mathbb{F}_{q^m}$ is a large finite field, so any choice of λ elements for $\lambda \ll m$ will be $\mathbb{F}_q$-linearly independent with overwhelming probability. Using $N = (m + (n-k)^2)\lambda\lceil \log_2(q) \rceil$ we can determine λ random elements from $\mathbb{F}_{q^m}$ which define the subspace $\Lambda \subset \mathbb{F}_{q^m}$. We then define the $(n-k)^2$ entries of $S^{-1} \in \mathsf{GL}_{n-k}(\Lambda)$ as $\mathbb{F}_q$-linear combinations over those λ elements, using the next $(n-k)^2\lambda\lceil \log_2 q \rceil$ bits. The resulting matrix is almost certainly invertible and permits computing $S = (S^{-1})^{-1}$.

Similar strategies can be applied to compute the matrices M, H_1 and H_2. Let $s_P, s_S, s_M, s_{H_1}, s_{H_2}$ be the seeds corresponding to the matrices P, S, M, H_1 and H_2, respectively. Then we can set $\mathsf{sk} = (s_P, s_S, s_{H_1}, s_{H_2})$ and $\mathsf{pk} = (s_M, F)$ where $F = MP^{-1}H_1^T \left(H_2^T\right)^{-1} S$. This approach cannot be used to compress F as it

depends on the other matrices. In this way we reduced the private key size of REDOG to 1024 bits and public key of size of REDOG to $256+\ell(n-k)m\lceil\log_2(q)\rceil$. For the 128-bit-security level, we obtain a secret key size of 0.13 KB compared to the original 1.45 KB and a public key size of 13, 85 KB, compared to the original 14, 25KB (which was obtained by choosing M to be a circulant matrix) at the expense of having to recompute the matrices from their seeds when needed. Given that matrix inversion over $\mathbb{F}_{q^m}$ is not fast, implementations may prefer to include S and S^{-1} in sk and use seeds for the other matrices. To save even more space, it is possible to replace $s_P, s_S, s_M, s_{H_1}, s_{H_2}$ by a single seed s and generating those five seeds as a call to $f(s)$. The public key then includes the derived value s_M but the secret key consists only of s. Note that in that case each non-invertible Q will be generated for each run expanding the secret seed, before finding the Q and P that were used in computing pk. In summary, this strategy provides a tradeoff between size and computing time.

References

AAB+17a. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Adrien Hauteville, and Gilles Zémor. Ouroboros-R. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

AAB+17b. Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillippe Gaborit, and Gilles Zémor. RQC. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

AASA+20. Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, and Daniel Smith-Tone. Status report on the second round of the NIST post-quantum cryptography standardization process. NIST IR 8309, 2020. https://doi.org/10.6028/NIST.IR.8309.

ABD+17a. Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. LAKE. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

ABD+17b. Nicolas Aragon, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Adrien Hauteville, Olivier Ruatta, Jean-Pierre Tillich, and Gilles Zémor. LOCKER. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

AGHT18. Nicolas Aragon, Philippe Gaborit, Adrien Hauteville, and Jean-Pierre Tillich. A new algorithm for solving the rank syndrome decoding problem. In *ISIT*, pages 2421–2425. IEEE, 2018.

BBB+20. Magali Bardet, Pierre Briaud, Maxime Bros, Philippe Gaborit, Vincent Neiger, Olivier Ruatta, and Jean-Pierre Tillich. An algebraic attack on rank metric code-based cryptosystems. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 64–93, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.

BBC+20. Magali Bardet, Maxime Bros, Daniel Cabarcas, Philippe Gaborit, Ray A. Perlner, Daniel Smith-Tone, Jean-Pierre Tillich, and Javier A. Verbel. Improvements of algebraic attacks for solving the rank decoding and MinRank problems. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 507–536, Daejeon, South Korea, December 7–11, 2020. Springer, Heidelberg, Germany.

Ber03. Thierry P. Berger. Isometries for rank distance and permutation group of Gabidulin codes. *IEEE Trans. Inf. Theory*, 49(11):3016–3019, 2003.

Del78. Philippe Delsarte. Bilinear forms over a finite field, with applications to coding theory. *Journal of Combinatorial Theory, Series A*, 25(3):226–241, 1978.

DGZ17. Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Ouroboros: A simple, secure and efficient key exchange protocol based on coding theory. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 18–34, Utrecht, The Netherlands, June 26–28, 2017. Springer, Heidelberg, Germany.

DWZ22. Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. *arXiv preprint arXiv:2210.10173*, 2022.

Gab85. Ernst M. Gabidulin. Theory of codes with maximum rank distance. *Problems of Information Transmission*, 21(1):1–12, 1985.

GKK+17. Lucky Galvez, Jon-Lark Kim, Myeong Jae Kim, Young-Sik Kim, and Nari Lee. McNie. Technical report, National Institute of Standards and Technology, 2017. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-1-submissions.

GPT91. Ernst M. Gabidulin, A. V. Paramonov, and O. V. Tretjakov. Ideals over a non-commutative ring and thier applications in cryptology. In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 482–489, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

GRS16. Philippe Gaborit, Olivier Ruatta, and Julien Schrek. On the complexity of the rank syndrome decoding problem. *IEEE Transactions on Information Theory*, 62(2):1006–1019, 2016.

HTMR15. Anna-Lena Horlemann-Trautmann, Kyle Marshall, and Joachim Rosenthal. Extension of Overbeck's attack for Gabidulin-based cryptosystems. *Designs, Codes and Cryptography*, 86:319–340, 2015.

KHL+22a. Jon-Lark Kim, Jihoon Hong, Terry Shue Chien Lau, YounJae Lim, Chik How Tan, Theo Fanuela Prabowo, and Byung-Sun Won. REDOG. Submission to KpqC Round 1, 2022.

KHL+22b. Jon-Lark Kim, Jihoon Hong, Terry Shue Chien Lau, YounJae Lim, Chik How Tan, Theo Fanuela Prabowo, and Byung-Sun Won. REDOG and its performance analysis. Cryptology ePrint Archive, Report 2022/1663, 2022. https://eprint.iacr.org/2022/1663.

KKGK21. Jon-Lark Kim, Young-Sik Kim, Lucky Erap Galvez, and Myeong Jae Kim. A modified Dual-Ouroboros public-key encryption using Gabidulin codes. *Appl. Algebra Eng. Commun. Comput.*, 32(2):147–156, 2021.

Loi17. Pierre Loidreau. A new rank metric codes based encryption scheme. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017*, pages 3–17, Utrecht, The Netherlands, June 26–28, 2017. Springer, Heidelberg, Germany.

LT18. Terry Shue Chien Lau and Chik How Tan. Key recovery attack on McNie based on low rank parity check codes and its reparation. In Atsuo Inomata and Kan Yasuda, editors, *IWSEC 18*, volume 11049 of *LNCS*, pages 19–34, Sendai, Japan, September 3–5, 2018. Springer, Heidelberg, Germany.

LTP21. Terry Shue Chien Lau, Chik How Tan, and Theo Fanuela Prabowo. On the security of the modified Dual-Ouroboros PKE using Gabidulin codes. *Appl. Algebra Eng. Commun. Comput.*, 32(6):681–699, 2021.

Ove05. Raphael Overbeck. A new structural attack for GPT and variants. In *Mycrypt*, volume 3715 of *Lecture Notes in Computer Science*, pages 50–63. Springer, 2005.

Ove08. R. Overbeck. Structural attacks for public key cryptosystems based on Gabidulin codes. *Journal of Cryptology*, 21(2):280–301, April 2008.

Pra62. E. Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

S+21. W. A. Stein et al. *Sage Mathematics Software (Version (9.3))*. The Sage Development Team, 2021. `http://www.sagemath.org`.

Sta11. Richard P. Stanley. *Enumerative Combinatorics*, volume 1 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 2 edition, 2011.

Wie86. Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986.

A Sage code for ISD attack in Section ??

The following Sage script can be used for breaking the reference implementation for the 128-bit-security parameters. The file reads from the KAT file `rsp_128.rsp`. See also https://gitlab.tue.nl/tlange/kpqc-public/-/tree/master/redog for the code.

```
from sage.doctest.util import Timer
q=2
m=83
Fqm = GF(q^m)
ran_len = 30

def Hash_function(error_vec, length_n, length_k, length_l , seed_num = 0) :
    import random
    random.seed(seed_num)
    index_list = range(2*length_n - length_k)
    index = random.sample(index_list, length_l)
    Field = error_vec.base_ring()
    Hash_error = zero_matrix(Field, 1, length_l)
    for i in range(length_l):
        Hash_error[0,i] = error_vec[0, index[i]]
    return matrix(Hash_error)

def pis(G,y):
Ψ M = copy(G)
Ψk,n = M.dimensions()
Ψp = list(Permutations(n).random_element())
Ψindexes = p[:k]
```

```
Ψindexes.sort()
ΨcolsG = [M.columns()[i-1] for i in indexes]
Ψcolsy = [y.columns()[i-1] for i in indexes]
ΨpisG = matrix(Fqm, colsG)
Ψpisy = matrix(Fqm, colsy)
Ψreturn pisG.transpose(), pisy.transpose()
Ψ
Ψ
def Prange(pubKey, y, t):
ΨM = pubKey[0].augment(pubKey[1])
Ψkpr,npr = M.dimensions()
Ψyleft = y[0][:n]
Ψyright = y[0][n:]
Ψwhile True:
ΨΨM1,y1 = pis(M,y)
ΨΨwhile not M1.is_invertible():
ΨΨΨM1,y1 = pis(M,y)
ΨΨU = M1.inverse()
ΨΨmsg =y1*U
ΨΨy1left = msg*pubKey[0].matrix_from_columns(list(range(t+3)))
ΨΨwtleft = len([i for i in range(t+3) if y1left[0][i]!=yleft[i]])
ΨΨif wtleft <= t:
ΨΨΨx = msg*M
ΨΨΨwt = len([i for i in range(2*n-k) if x[0][i]!=y[0][i]])
ΨΨΨif wt <= t:
ΨΨΨΨe = y -x
ΨΨΨΨreturn msg, e
ΨΨΨ
def string_to_mat(s,nrows,ncols):
Ψsbin = [list(reversed(Integer(ch,base=16).digits(base=2,padto=4))) for ch in s]
Ψsbin = flatten(sbin)
Ψmeta = []
Ψfor i in range(0,len(sbin),m):
ΨΨmeta.append(Fqm(sbin[i:i+m]))
Ψreturn matrix(Fqm,nrows,ncols,meta[:ncols*nrows])
Ψ
def el_to_string(polynomial): #copied from original submission
    p_coeff = matrix(ZZ(polynomial.integer_representation() ).digits(base=q, padto=m))
    p_bin = p_coeff[0]
    return p_bin

with open('rsp_128.rsp', 'r') as f:
Ψpk, cipher, pk0, pk1 = '', '', '', ''
Ψstart = False # used to read pk which is on 2 different lines
Ψtime = 0
Ψtimer = Timer()
Ψfor line in f:
ΨΨline = line.split()
ΨΨif line[:2] == ['msg', '=']:
ΨΨΨmsg = string_to_mat(line[2],1,37)
ΨΨif line[:2] == ['pk', '=']:
ΨΨΨstart = True
ΨΨΨpk = line[2]
ΨΨΨtmp = pk
ΨΨΨpk0 = string_to_mat(pk,37,44)
ΨΨelif start and line[0] == ',':
ΨΨΨpk= line[1]
ΨΨΨpk1 = string_to_mat(pk,37,36)
ΨΨΨstart = False
ΨΨelse: start=False
ΨΨif line[:2] == ['c','=']:
ΨΨΨcipher = line[2]
ΨΨΨcipher = string_to_mat(cipher,1,80)
ΨΨΨend=True
ΨΨΨtimer.start()
ΨΨΨm_prange,e= Prange([pk0,pk1], cipher, 6)
ΨΨΨtimer.stop()
ΨΨΨtime += timer.walltime
```

```
ΨΨΨprint(m_prange - Hash_function(e,44,8,37)==msg)
Ψprint(time/ran_len)Ψ
ΨΨΨ
#ΨΨif line[:2] == ['c', '='] : cipher = line[2]
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
```

The KAT files for the 192- and 256-bit-security parameters use a different encoding of the public key. This file is for `rsp_192.rsp` and can be used for `rsp_256.rsp` as well.

```
from sage.doctest.util import Timer

level = 192
(n, k, l, q, m, r, t) = (58, 10, 49, 2, 109, 24, 8)
ran_len = 15
Fqm.<z> = GF(q^m)

def Hash_function(error_vec, length_n, length_k, length_l , seed_num = 0) :
    import random
    random.seed(seed_num)
    index_list = range(2*length_n - length_k)
    index = random.sample(index_list, length_l)
    Field = error_vec.base_ring()
    Hash_error = zero_matrix(Field, 1, length_l)
    for i in range(length_l):
        Hash_error[0,i] = error_vec[0, index[i]]
    return matrix(Hash_error)

def pis(G,y):
ΨM = copy(G)
Ψk,n = M.dimensions()
Ψp = list(Permutations(n).random_element())
Ψindexes = p[:k]
Ψindexes.sort()
ΨcolsG = [M.columns()[i-1] for i in indexes]
Ψcolsy = [y.columns()[i-1] for i in indexes]
ΨpisG = matrix(Fqm, colsG)
Ψpisy = matrix(Fqm, colsy)
Ψreturn pisG.transpose(), pisy.transpose()
Ψ
Ψ
def Prange(pubKey, y, t):
ΨM = pubKey[0].augment(pubKey[1])
Ψkpr,npr = M.dimensions()
Ψyleft = y[0][:n]
Ψyright = y[0][n:]
Ψwhile True:
ΨΨM1,y1 = pis(M,y)
ΨΨwhile not M1.is_invertible():
ΨΨΨM1,y1 = pis(M,y)
ΨΨU = M1.inverse()
ΨΨmsg =y1*U
ΨΨy1left = msg*pubKey[0].matrix_from_columns(list(range(t+3)))
ΨΨwtleft = len([i for i in range(t+3) if y1left[0][i]!=yleft[i]])
ΨΨif wtleft <= t:
ΨΨΨx = msg*M
ΨΨΨwt = len([i for i in range(2*n-k) if x[0][i]!=y[0][i]])
ΨΨΨif wt <= t:
```

```
ΨΨΨe = y -x
ΨΨΨreturn msg, e

def string_to_mat(s,nrows,ncols):
Ψsbin = [list(reversed(Integer(ch,base=16).digits(base=2,padto=4))) for ch in s]
Ψsbin = flatten(sbin)
Ψmeta = []
Ψfor i in range(0,len(sbin),m):
ΨΨmeta.append(Fqm(sbin[i:i+m]))
Ψreturn matrix(Fqm,nrows,ncols,meta[:ncols*nrows])
Ψ
def el_to_string(polynomial): #copied from original submission
    p_coeff = matrix(ZZ(polynomial.integer_representation() ).digits(base=q, padto=m))
    p_bin = p_coeff[0]
    return p_bin

def string_to_Fqm_tuple(s):
Ψs = s.split()
Ψtup = []
Ψel = ''
Ψj=1
Ψwhile j<len(s):
ΨΨif s[j]!='+':
ΨΨΨel += s[j-1]
ΨΨΨtup.append(Fqm(el))
ΨΨΨel = ''
ΨΨΨj +=1
ΨΨelse :
ΨΨΨel += s[j-1]+s[j]
ΨΨΨj+=2
Ψif j ==len(s):tup.append(Fqm(el+s[j-1]))
Ψif j ==len(s)+1 : tup.append(Fqm(el+s[j-1]))
Ψreturn tup

with open(f'rsp_{level}.rsp', 'r') as f:
Ψcipher, pk0, pk1 = None, [], []
ΨstartM = False # used to read pk which is on different lines
ΨstartF = False
Ψi = 1 # used to read the public key 0 <= i < l
Ψtime = 0
Ψtimer = Timer()
Ψfor line in f:
ΨΨif startM and i<l:
ΨΨΨi = i+1
ΨΨΨpkl = line[1:-2]
ΨΨΨpk0.append(string_to_Fqm_tuple(pkl))

ΨΨif startF and i < l:
ΨΨΨi += 1
ΨΨΨpkl = line[1:-2]
ΨΨΨpk1.append(string_to_Fqm_tuple(pkl))
ΨΨΨ
ΨΨif line[:5] == 'msg =':
ΨΨΨmsg = string_to_mat(line[6:-1],1,l)
ΨΨif line[:4] == 'pk =':
ΨΨΨstartM = True
ΨΨΨpkl = line[6:-2]
ΨΨΨpk0.append(string_to_Fqm_tuple(pkl))
ΨΨif startM and line[0] == ',':
ΨΨΨpkl = line[3:-2]
ΨΨΨpk1.append(string_to_Fqm_tuple(pkl))
ΨΨΨstartM = False
ΨΨΨstartF = True
ΨΨΨi = 1
ΨΨif startF and i==l:
ΨΨΨstartF=False
ΨΨΨi=1
ΨΨif line[:3] == 'c =':
```

```
ΨΨΨpk0 = matrix(Fqm,l,n,pk0)
ΨΨΨpk1 = matrix(Fqm,l,n-k,pk1)
ΨΨΨcipher = line[4:-1]
ΨΨΨcipher = string_to_mat(cipher,1,2*n-k)
ΨΨΨtimer.start()
ΨΨΨm_prange,e= Prange([pk0,pk1], cipher, t)
ΨΨΨtimer.stop()
ΨΨΨtime += timer.walltime
ΨΨΨprint(m_prange - Hash_function(e,n,k,l)==msg)
ΨΨΨpk0,pk1=[],[]
ΨΨΨi=1
ΨΨΨstartM=False
ΨΨΨstartF=False
Ψprint(time/ran_len)Ψ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
ΨΨ
```

B Sage code for Section 9

The following is the Sage script for searching the best value for t_2 for the REDOG parameters suggested for 128 bit security.

Changing line 31 to

`n = 58; k = 10; l = 49; m = 109; r = 24; lam = 3; t = 8`

or

`n = 72; k = 12; l = 61; m = 135; r = 30; lam = 3; t = 10`

respectively covers the parameters proposed for 192 and 256 bit security. See also https://gitlab.tue.nl/tlange/kpqc-public/-/tree/master/redog for the code.

```
# AGHT: Aragon, Gaborit, Hauteville, Tilich
# GRS: main case in Gaborit, Ruatta, Schrek
# fast: fast case from Gaborit, Ruatta, Schrek
# BBB: Bardet, Briaud, Bros, Gaborit, Neiger, Ruatta, Tillich,
# BBC: Bardet, Bros, Cabarcas, Gaborit, Perlner, Smith-Tone, Tillich, Verbel
# brute: brute-force search as in section 9

q=2
w_win = 2.37
w_stras = 2.807

def log2(x):
  assert x > 0
  return log(RR(x),2.0)


def A_b(NF,RF,MF,KF,b):
  return sum(binomial(NF,RF)*binomial(MF*KF + 1,j) for j in range(1,b+1))

def B_b(NF,RF,MF,KF,b):
  return sum(MF*binomial(NF - KF - 1,RF)*binomial(MF * KF + 1,j) for j in range(1,b+1))

def C_b(NF,RF,MF,KF,b):
  return sum(sum((-1)^s+i * binomial(NF, RF+s) * binomial(MF + s - 1, s) * binomial(MF * KF + 1, j - s) for s in range(1,j + 1)) for j in range(1, b + 1))

def D_b(NF,RF,MF,KF,b):
  return  (B_b(NF,RF,MF,KF,b)*binomial(KF + RF + 1, RF) + C_b(NF,RF,MF,KF,b)*(MF * KF + 1)*(RF + 1))/(B_b(NF,RF,MF,KF,b) + C_b(NF,RF,MF,KF,b))

print('Security level 128')

n = 44; k =  8; l = 37; m =  83; r = 18; lam = 3; t = 6

K = l; M = m

AGHT = []; GRS = []; fast = []; BBB_win = []; BBB_stras = []
```

```
BBC_win = []; BBC_stras = []; BBC_wied = []; brute = []

for t2 in range((r//lam) + 1): # we can handle brute-force attacks
  t1 = r - t2 * lam
  N = K + min(t1,t2)
  # initializing all values  (counter, cost, and direction)
  i_AGHT = 0; i_GRS = 0; i_fast = 0; i_BBB_win = 0; i_BBB_stras = 0;
  i_BBC_win = 0; i_BBC_stras = 0; i_BBC_wied = 0;
  c_AGHT = oo; c_GRS = oo; c_fast = oo; c_BBB_win = oo; c_BBB_stras = oo;
  c_BBC_win = oo; c_BBC_stras = oo; c_BBC_wied = oo;
  dir_AGHT = 'l'; dir_GRS = 'l'; dir_fast = 'l'; dir_BBB_win = 'l'; dir_BBB_stras = 'l';
  dir_BBC_win = 'l'; dir_BBC_stras = 'l'; dir_BBC_wied = 'l';
  # first we look from the left
  for i in range (2*n - k - N + 1):
    # we normally get rank s taking s < t columns
    if (N + i) in range(K + t1, n+1): R = t1 # purely in c1 part, only rank t1
    elif (N + i) >= n + t2: R = t1 + t2 # reached full rank
    elif (N + i) < K + t1: continue # not enough information to decode
    else: R = t1 + N + i - n # taking N + i - n columns from right part
    try:
      if (R*(K+1)*M/(N+i) -M) >= 0:
        cost = log2((N + i -K)^3*M^3*q^(R*(K+1)*M/(N+i) -M))
        if cost < c_AGHT: c_AGHT = cost; i_AGHT = N + i; dir_AGHT = 'l'
    except:
      pass
    try:
      if R > 0:
        cost = log2((N+i-K)^3*M^3*q^(min(R*floor(K*M/(N+i)),(R-1)*floor((K+1)*M/(N+i)))))
        if cost < c_GRS: c_GRS = cost; i_GRS = N + i; dir_GRS = 'l'
    except:
      pass
    if R > 0:
      if K > ceil(((R+1)*(K+1)-(N+i)-1)/R) and ceil(((R+1)*(K+1)-(N+i)-1)/R) >= 0:
        try:
          cost = log2(R^3 * K^3 * q^(R*ceil(((R+1)*(K+1)-(N+i)-1)/R)))
          if cost < c_fast: c_fast = cost; i_fast = N + i; dir_fast = 'l'
        except:
          pass
    if (N + i - K - 1) > R and M*binomial((N+i) - K - 1, R) + 1 >= binomial((N+i), R):
      try:
        cost = log2(((((M+N+i)*R)^R)/factorial(R))^w_win)
        if cost < c_BBB_win: c_BBB_win = cost; i_BBB_win = N + i; dir_BBB_win = 'l'
      except:
        pass
      try:
        cost = log2(M*binomial((N+i) - K - 1, R) * binomial((N+i), R)^(w_win-1))
        if cost < c_BBC_win: c_BBC_win = cost; i_BBC_win = N + i; dir_BBC_win = 'l overdetermined'
      except:
        pass
      try:
        cost = log2(((((M+N+i)*R)^R)/factorial(R))^w_stras)
        if cost < c_BBB_stras: c_BBB_stras = cost; i_BBB_stras = N + i; dir_BBB_stras = 'l'
      except:
        pass
      try:
        cost = log2(M*binomial((N+i) - K - 1, R) * binomial((N+i), R)^(w_stras-1))
        if cost < c_BBC_stras: c_BBC_stras = cost; i_BBC_stras = N + i; dir_BBC_stras = 'l overdetermined'
      except:
        pass
    else:
      try:
        cost  =  log2(((((M+N+i)*R)^(R+1))/factorial(R+1))^w_win)
        if cost < c_BBB_win: c_BBB_win = cost; i_BBB_win = N + i; dir_BBB_win = 'l'
      except:
        pass
      try:
        cost =  log2(((((M+N+i)*R)^(R+1))/factorial(R+1))^w_stras)
        if cost < c_BBB_stras: c_BBB_stras = cost; i_BBB_stras = N + i; dir_BBB_stras = 'l'
      except:
        pass
      j = 1
      if (N + i - K -1) >= R:
        while (M*binomial((N+i) - K - 1, R) + 1 < binomial((N+i-j),R)): j = j+1
        if min(N + i-K - 1, N + i -j) >= R:
          try:
            cost = log2(q^(j*R)*M*binomial((N+i) - K - 1, R)*binomial((N+i-j), R)^(w_win-1))
            if cost < c_BBC_win: c_BBC_win = cost; i_BBC_win = N + i; dir_BBC_win = 'l hybrid'
          except:
            pass
          try:
            cost = log2(q^(j*R)*M*binomial((N+i) - K - 1, R)*binomial((N+i-j), R)^(w_stras-1))
            if cost < c_BBC_stras: c_BBC_stras = cost; i_BBC_stras = N + i; dir_BBC_stras = 'l hybrid'
          except:
            pass
      b = 1
      while ((sum(binomial(N+i,R)*binomial(M*K,j)) for j in range(1,b+1))  -  1 > sum(sum((-1)^(s+1)*binomial(N+i,R+s)*binomial(M+s-1,s)*binomial(M*K,j-s)) for s in range
        b = b + 1
      if b < R+2:
        try:
          cost = log2(K*M*(R+1)*(sum(binomial(N+i,R)*binomial(K*M,j)) for j in range(1,b+1))^2)
```

```
        if cost < c_BBC_wied: c_BBC_wied = cost; i_BBC_wied = N + i; dir_BBC_wied = 'l 26, 27'
      except:
        pass
    b = 1
    while A_b(N+i,R,M,K,b) - 1 > B_b(N+i,R,M,K,b) + C_b(N+i,R,M,K,b) and b < R + 2: b = b + 1
    if b < R + 2:
      try:
        cost = log2((B_b(N+i,R,M,K,b) + C_b(N+i,R,M,K,b))*A_b(N+i,R,M,K,b)^(w_win - 1))
        if cost < c_BBC_win: c_BBC_win = cost; i_BBC_win = N + i; dir_BBC_win = 'l 28 , 29'
      except:
        pass
      try:
        cost = log2((B_b(N+i,R,M,K,b) + C_b(N+i,R,M,K,b))*A_b(N+i,R,M,K,b)^(w_stras - 1))
        if cost < c_BBC_stras: c_BBC_stras = cost; i_BBC_stras = N + i; dir_BBC_stras = 'l 28 , 29'

      except:
        pass
      try:
        cost =  log2(D_b(N+i,R,M,K,b) * A_b(N+i,R,M,K,b)^2)
        if cost < c_BBC_wied: c_BBC_wied = cost; i_BBC_wied = N + i; dir_BBC_wied = 'l 30'
      except:
        pass


# now do the right part
# no new initialization, so 'l' has some priority
for i in range (2*n - k - N + 1):
  # we normally get rank s taking s < t columns
  if (N + i) in range(K + t2, n - k + 1): R = t2 # this case is empty for the REDOG parameters
  elif (N + i) >= n - k + t1: R = t1 + t2 # reached full rank
  # all other cases are empty for REDOG
  elif (N + i) < K + t2: continue # not enough information to decode
  else:
    if (N + i) > (K + t2 + N + i - (n - k)) : R = t2 + N + i - (n - k)
    else: continue
  try:
    if (R*(K+1)*M/(N+i) -M) >= 0:
      cost = log2((N + i -K)^3*M^3*q^(R*(K+1)*M/(N+i) -M))
      if cost < c_AGHT: c_AGHT = cost; i_AGHT = N + i; dir_AGHT = 'r'
  except:
    pass
  try:
    if R > 0:
      cost = log2((N+i-K)^3*M^3*q^(min(R*floor(K*M/(N+i)),(R-1)*floor((K+1)*M/(N+i)))))
      if cost < c_GRS: c_GRS = cost; i_GRS = N + i; dir_GRS = 'r'
  except:
    pass
  if R > 0:
    if K > ceil(((R+1)*(K+1)-(N+i)-1)/R) and ceil(((R+1)*(K+1)-(N+i)-1)/R) >= 0:
      try:
        cost = log2(R^3 * K^3 * q^(R*ceil(((R+1)*(K+1)-(N+i)-1)/R)))
        if cost < c_fast: c_fast = cost; i_fast = N + i; dir_fast = 'r'
      except:
        pass
  if (N + i - K - 1) >= R and M*binomial((N+i) - K - 1, R) + 1 >= binomial((N+i), R):
    try:
      cost = log2(((((M+N+i)*R)^R)/factorial(R))^w_win)
      if cost < c_BBB_win: c_BBB_win = cost; i_BBB_win = N + i; dir_BBB_win = 'r'
    except:
      pass
    try:
      cost = log2(M*binomial((N+i) - K - 1, R) * binomial((N+i), R)^(w_win-1))
      if cost < c_BBC_win: c_BBC_win = cost; i_BBC_win = N + i; dir_BBC_win = 'r overdetermined'
    except:
      pass
    try:
      cost = log2(((((M+N+i)*R)^R)/factorial(R))^w_stras)
      if cost < c_BBB_stras: c_BBB_stras = cost; i_BBB_stras = N + i; dir_BBB_stra = 'r'
    except:
      pass
    try:
      cost = log2(M*binomial((N+i) - K - 1, R) * binomial((N+i), R)^(w_stras-1))
      if cost < c_BBC_stras: c_BBC_stras = cost; i_BBC_stras = N + i; dir_BBC_stras = 'r overdetermined'
    except:
      pass
  else:
    try:
      cost  =  log2(((((M+N+i)*R)^(R+1))/factorial(R+1))^w_win)
      if cost < c_BBB_win: c_BBB_win = cost; i_BBB_win = N + i; dir_BBB_win = 'r'
    except:
      pass
    try:
      cost =  log2(((((M+N+i)*R)^(R+1))/factorial(R+1))^w_stras)
      if cost < c_BBB_stras: c_BBB_stras = cost; i_BBB_stras = N + i; dir_BBB_stras = 'r'
    except:
      pass
    j = 1
    if (N + i - K -1) >= R:
      while (M*binomial((N+i) - K - 1, R) + 1 < binomial((N+i-j),R)): j = j+1
      if min(N + i-K - 1, N + i -j) >= R:
```

```
            try:
              cost = log2(q^(j*R)*M*binomial((N+i) - K - 1, R)*binomial((N+i-j), R)^(w_win-1))
              if cost < c_BBC_win: c_BBC_win = cost; i_BBC_win = N + i; dir_BBC_win = 'r hybrid'
            except:
              pass
            try:
              cost = log2(q^(j*R)*M*binomial((N+i) - K - 1, R)*binomial((N+i-j), R)^(w_stras-1))
              if cost < c_BBC_stras: c_BBC_stras = cost; i_BBC_stras = N + i; dir_BBC_stras = 'r hybrid'
            except:
              pass
        b = 1
        while ((sum(binomial(N+i,R)*binomial(M*K,j)) for j in range(1,b+1))  -  1 > sum(sum((-1)^(s+1)*binomial(N+i,R+s)*binomial(M+s-1,s)*binomial(M*K,j-s)) for s in range
          b = b + 1
        if b < R+2:
          try:
            cost = log2(K*M*(R+1)*(sum(binomial(N+i,R)*binomial(K*M,j)) for j in range(1,b+1))^2)
            if cost < c_BBC_wied: c_BBC_wied = cost; i_BBC_wied = N + i; dir_BBC_wied = 'r 26 , 27'
          except:
            pass
        b = 1
        while A_b(N+i,R,M,K,b) - 1 > B_b(N+i,R,M,K,b) + C_b(N+i,R,M,K,b) and b < R + 2: b = b + 1
        if b < R + 2:
          try:
            cost = log2((B_b(N+i,R,M,K,b) + C_b(N+i,R,M,K,b))*A_b(N+i,R,M,K,b)^(w_win - 1))
            if cost < c_BBC_win: c_BBC_win = cost; i_BBC_win = N + i; dir_BBC_win = 'r 28 , 29'
          except:
            pass
          try:
            cost = log2((B_b(N+i,R,M,K,b) + C_b(N+i,R,M,K,b))*A_b(N+i,R,M,K,b)^(w_stras - 1))
            if cost < c_BBC_stras: c_BBC_stras =cost; i_BBC_stras = N + i; dir_BBC_stras = 'r 28 , 29'

          except:
            pass
          try:
            cost =  log2(D_b(N+i,R,M,K,b) * A_b(N+i,R,M,K,b)^2)
            if cost < c_BBC_wied: c_BBC_wied = cost; i_BBC_wied = N + i; dir_BBC_wied = 'r 30'
          except:
            pass
    AGHT.append((c_AGHT, i_AGHT, dir_AGHT))
    GRS.append((c_GRS, i_GRS, dir_GRS))
    fast.append((c_fast, i_fast, dir_fast))
    BBB_win.append((c_BBB_win, i_BBB_win, dir_BBB_win))
    BBB_stras.append((c_BBB_stras, i_BBB_stras, dir_BBB_stras))
    BBC_win.append((c_BBC_win, i_BBC_win, dir_BBC_win))
    BBC_stras.append((c_BBC_stras, i_BBC_stras, dir_BBC_stras))
    BBC_wied.append((c_BBC_wied, i_BBC_wied, dir_BBC_wied))
    brutel = log2(binomial(q^m - 1, t1)* q^(t1*l)/(q-1)^t1)
    bruter = log2(q^m * binomial(q^m - 1, t2)* q^(t2*(l-1))/(q-1)^t2)
    if brutel <= bruter: brute.append((brutel,0,'l'))
    else: brute.append((bruter,0,'r'))

algos = [AGHT,GRS,fast,BBB_win,BBB_stras,BBC_win,BBC_stras,BBC_wied,brute]
algo_names = ['AGHT','GRS','fast','BBB_win','BBB_stras','BBC_win','BBC_stras','BBC_wied','brute']

best = []
for i in range((r//lam)+ 1):
  local = [x[i] for x in algos]
  best.append((min(local),algo_names[local.index(min(local))]))

print('best attack per position')

print(best)
print(' ')
print(' ')

print('best parameter choices have attack costs', max(best), 'at t2 = ', best.index(max(best)))
print(' ')
print(' ')

print('Updated choice in REDOG has:' )

t2 = r//(2*lam)

for i in range(len(algos)):
  print(algo_names[i], ': ', algos[i][t2])
```