Preventing timing attacks against RQC using constant time decoding of Gabidulin codes

Slim Bettaieb*, Loïc Bidoux*, Philippe Gaborit†, and Etienne Marcatel*

*Worldline, ZI Rue de la pointe, 59113 Seclin, France
†University of Limoges, XLIM-DMI, 123, Av. Albert Thomas, 87060 Limoges, France

Abstract. This paper studies the resistance of the code-based encryption scheme RQC to timing attacks. We describe two choosen ciphertext timing attacks that relies on a correlation between the weight of the error to be decoded and the running time of Gabidulin code's decoding. These attacks are of theoretical interest as they outperform the best known algorithm to solve the rank syndrome decoding problem in term of complexity. Nevertheless, they are quite impracticable in real situations as they require a huge number of requests to a timing oracle. We also provide a constant-time algorithm for the decoding of Gabidulin codes that prevent these attacks without any performance cost for honest users.

Keywords: RQC, Gabidulin decoding, Timing attack, Rank metric

1 Introduction

RQC [2, 3] is a code-based IND-CCA2 public key encryption scheme submitted to the NIST's post-quantum cryptography standardization project. It features attractive parameters and its security only relies on the rank syndrom decoding problem without any additionnal assumption regarding the indistinguishability of the considered family of codes. RQC relies on Gabidulin codes which were introduced in 1985 in [6]. The latter are the analogs of the Reed-Solomon codes for the rank metric and can be thought as the evaluation of q-polynomials of bounded degree on the coordinates of a vector over $\mathbb{F}_{q^m}$. Gabidulin decoding can be performed efficiently using the Welch-Berlekamp like algorithm proposed by Loidreau [9]. Hereafter, we study the resistant of RQC to timing attacks.

Contributions. In this paper, we present two timing attacks against RQC. In addition, we also describe a constant time decoding algorithm for Gabidulin codes that prevent these attacks without any performance cost for honest users.

Paper organisation. In section 2, we introduce the rank metric, Loidreau's algorithm for the decoding of Gabidulin codes as well as the RQC cryptosystem. Next, in section 3, we highlight the correlation between the rank of the error to be decoded and the decoding time of Loidreau's algorithm. This correlation is the keystone of the timing attacks described in section 4. To finish, countermeasures preventing these attacks are presented in section 5.

2 Preliminaries

In this section, we present some preliminaries regarding the rank metric (section 2.1), Gabidulin codes (section 2.2) and the RQC cryptosystem (section 2.3).

2.1 Rank metric

Let q be a power of a prime p, m an integer, $\mathbb{F}_{q^m}$ a finite field and $\beta = (\beta_1, \cdots, \beta_m)$ a basis of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$. Any vector $\mathbf{x} \in \mathbb{F}_{q^m}^n$ can be associated to the matrix $\mathbf{M}_\mathbf{x} \in \mathcal{M}_{m,n}(\mathbb{F}_q)$ by expressing its coordinates in β.

Definition 1 (Rank weight). *Let $\mathbf{x} = (x_1, \cdots, x_n) \in \mathbb{F}_{q^m}^n$ be a vector, the rank weight of $\mathbf{x}$, denoted $\omega(\mathbf{x})$, is defined as the rank of the matrix $\mathbf{M}_\mathbf{x} = (x_{i,j})$ where $x_j = \sum_{i=1}^{m} x_{i,j}\beta_i$. The set of words of weight w in $\mathbb{F}_{q^m}^n$ is denoted $\mathcal{S}_w^n$.*

Definition 2 (Support). *The support of $\mathbf{x} \in \mathbb{F}_{q^m}^n$, denoted Supp$(\mathbf{x})$, is the $\mathbb{F}_q$-linear space of $\mathbb{F}_{q^m}$ spanned by the coordinates of $\mathbf{x}$. Formally, Supp$(\mathbf{x}) = \langle x_1, \ldots, x_n \rangle_{\mathbb{F}_q}$.*

Definition 3 ($\mathbb{F}_{q^m}$-linear code). *An $\mathbb{F}_{q^m}$-linear $[n, k]$ code $\mathcal{C}$ of length n and dimension k is a linear subspace of $\mathbb{F}_{q^m}^n$ of dimension k.*

Definition 4 (Generator Matrix). *A matrix $\mathbf{G} \in \mathbb{F}_{q^m}^{k \times n}$ is a generator matrix for the $[n, k]$ code $\mathcal{C}$ if $\mathcal{C} = \left\{\mathbf{xG} \mid \mathbf{x} \in \mathbb{F}_{q^m}^k\right\}$.*

Definition 5 (Parity-Check Matrix). *A matrix $\mathbf{H} \in \mathbb{F}_{q^m}^{(n-k) \times n}$ is a parity-check matrix for the $[n, k]$ code $\mathcal{C}$ if $\mathcal{C} = \left\{\mathbf{x} \in \mathbb{F}_{q^m}^n \mid \mathbf{Hx}^\top = \mathbf{0}\right\}$. The vector $\mathbf{Hx}^\top \in \mathbb{F}_{q^m}^{n-k}$ is called the syndrome of $\mathbf{x}$.*

2.2 Gabidulin codes

Gabidulin codes were introduced in 1985 in [6]. They can be seen as the evaluation of q-polynomials of bounded degree on the coordinates of a vector over $\mathbb{F}_{q^m}$. The notion of q-polynomial was introduced by Ore in [10].

Definition 6 (q-polynomials). *A q-polynomial over $\mathbb{F}_{q^m}$ is a polynomial defined as $P(X) = \sum_{i=0}^{r} p_i X^{q^i}$ with $p_i \in \mathbb{F}_{q^m}$ and $p_r \neq 0$. The q-degree of a q-polynomial P is denoted $\deg_q(P)$.*

Definition 7 (Gabidulin codes). *Let $k, n, m \in \mathbb{N}$ such that $k \leqslant n \leqslant m$. Let $\mathbf{g} = (g_1, \ldots, g_n)$ be a $\mathbb{F}_q$-linearly family of vectors of $\mathbb{F}_{q^m}$. The Gabidulin code $\mathcal{G}_\mathbf{g}(n, k, m)$ is the $[n, k]_{q^m}$ code defined as $\left\{P(\mathbf{g}) \mid \deg_q(P) < k\right\}$ where $P(\mathbf{g})$ denotes the evaluation of the coordinates of $\mathbf{g}$ by the q-polynomial P.*

Gabidulin codes can efficiently decode up to $\lfloor \frac{n-k}{2} \rfloor$ errors [6]. In such cases, the algorithm considered hereafter in this paper features no decoding failure. It has been proposed by Loidreau in [9] and later improved in [5]. It is based on the resolution of the Linear Reconstruction Problem (see [9, 5] for further details).

Definition 8 (Decoding$(\mathbf{y}, \mathcal{G}_\mathbf{g}, t)$). *Find, if it exists,* $\mathbf{c} \in \mathcal{G}_\mathbf{g}$ *and* $\mathbf{e}$ *with* $\omega(\mathbf{e}) \leq t$ *such that* $\mathbf{y} = \mathbf{c} + \mathbf{e}$.

Definition 9 (Reconstruction$(\mathbf{y}, \mathbf{g}, k, t)$). *Find a tuple* (V, N) *where* V *is a non-zero* q*-polynomial with* $\deg_q(V) \leq t$ *and* N *is a* q*-polynomial with* $\deg_q(N) \leq k + t - 1$ *such that* $V(y_i) = N(g_i)$ *with* $1 \leq i \leq n$.

Theorem 1 ([9]). *If* (V, N) *is a solution of* **Reconstruction**$(\mathbf{y}, \mathbf{g}, k, t)$ *and* $t \leq \lfloor \frac{(n-k)}{2} \rfloor$, *then* $(\mathbf{c}, \mathbf{e}) = (f(\mathbf{g}), \mathbf{y} - \mathbf{c})$ *with* f *defined as the left euclidean division of* N *by* V *in the ring of* q*-polynomials is a solution of* **Decoding**$(\mathbf{y}, \mathbf{g}, k, t)$.

As stated in [2], one can solve **Reconstruction**$(\mathbf{y}, \mathbf{g}, k, t)$ by constructing by recurrence two pairs of q-polynomials (N_0, V_0) and (N_1, V_1) satisfying the interpolation conditions of the problem $V(y_i) = N(g_i), 1 \leq i \leq n$ at each step i and such that at least one of the pairs satisfies the final degree conditions $\deg_q(V) \leq t$ and $\deg_q(N) \leq k + t - 1$. See algorithm 5 (from [5], section 4, algorithm 5) hereafter for additionnal details.

Theorem 2 ([5]). *The complexity of solving the* **Decoding**$(\mathbf{y}, \mathcal{G}_\mathbf{g}, t)$ *problem using algorithm 5 is* $\mathcal{O}(n^2)$ *operations in* $\mathbb{F}_{q^m}$.

2.3 The RQC public key encryption scheme

RQC [2, 3] is a code-based IND-CCA2 encryption scheme whose security relies on the rank syndrom decoding problem [7, 4] without any additionnal assumption regarding the indistinguishability of the family of codes used. It is based on an IND-CPA PKE construction (described in figure 1) on top of which the HHK transformation [8] is applied in order to obtain an IND-CCA2 KEM. Standard transformations are then applied in order to get an IND-CCA2 encryption scheme. RQC uses a Gabidulin code of generator matrix $\mathbf{G}$ denoted $\mathcal{C}$ and a random double-circulant $[2n, n]$ code of parity-check matrix $(\mathbf{1}, \mathbf{h})$.

- $\mathsf{Setup}(1^\lambda)$: Generates and outputs the global parameters $\mathsf{param} = (n, k, \delta, w, w_\mathbf{r}, w_\mathbf{e})$.
- $\mathsf{KeyGen}(\mathsf{param})$: Samples $\mathbf{h} \xleftarrow{\$} \mathbb{F}_{q^m}^n$, $\mathbf{G} \in \mathbb{F}_{q^m}^{k \times n}$ a generator matrix of $\mathcal{C}$, $\mathbf{x} \xleftarrow{\$} \mathcal{S}_w^n$, $\mathbf{y} \xleftarrow{\$} \mathcal{S}_w^n$ such that $\mathsf{Supp}(\mathbf{x}) = \mathsf{Supp}(\mathbf{y})$. Sets $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$, $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$ and returns $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Encrypt}(\mathsf{pk}, \mathbf{m})$: Generates $\mathbf{r_1} \xleftarrow{\$} \mathcal{S}_{\omega_r}^n$, $\mathbf{r}_2 \xleftarrow{\$} \mathcal{S}_{\omega_r}^n$, $\mathbf{e} \xleftarrow{\$} \mathcal{S}_{\omega_r}^n$ such that $\mathsf{Supp}(\mathbf{r}_1) = \mathsf{Supp}(\mathbf{r}_2) = \mathsf{Supp}(\mathbf{r}_3)$. Sets $\mathbf{u} = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$ and $\mathbf{v} = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{e}$ and returns $\mathbf{c} = (\mathbf{u}, \mathbf{v})$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathbf{c})$: Returns $\mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$.

Fig. 1. Description of the IND-CPA version of RQC [2].

RQC correcteness relies on the decoding capability of the Gabidulin code $\mathcal{C}$. Indeed, $\mathsf{Decrypt}\left(\mathrm{sk}, \mathsf{Encrypt}\left(\mathrm{pk}, \mathbf{m}\right)\right) = \mathbf{m}$ when $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ is correctly decoded namely whenever $\omega\left(\mathbf{x} \cdot \mathbf{r}_2 - \mathbf{y} \cdot \mathbf{r}_1 + \mathbf{e}\right) \leq \lfloor \frac{(n-k)}{2} \rfloor$.

3 Correlation between decoding time and error rank

In this section, we show that there exists a correlation between the rank of the error to be decoded and the running time of algorithm 5. This observation is summarized in theorem 3. We start by introducing a simpler version of Loidreau's algorithm (section 3.1) and then we prove the aforementioned theorem (section 3.2). Next, we describe an oracle that computes the rank of the error to be decoded using the running time of the decoding algorithm (section 3.3).

3.1 A simpler decoding algorithm

In order to solve the **Reconstruction**$(\mathbf{y}, \mathbf{g}, k, t)$ problem, Loidreau's algorithm performs a q-polynomial interpolation. We denote by *nominal case*, *dummy interpolation case* and *early end case* the three scenarios that may occur during the interpolation step (see algorithm 5). The *early end case* is quite subtle as it performs two operations simultaneously. First, it checks the discrepancy vector to detect if the current q-polynomials are an admissible solution which can happen whenever the rank of the error to be decoded is inferior to the decoding capacity of the code. In addition, if a nominal interpolation can't be performed using the i^{th} coordinate of the discrepancy vector (see *nominal case* below) but can be performed using one of its j^{th} coordinate where $j > i$, then the i^{th} and j^{th} coordinates of the discrepancy vector are swapped. The *nominal case* corresponds to the expected interpolation which requires to inverse $u_{1,i}$ to be performed. If both $u_{1,i} = 0$ and $u_{0,i} = 0$, a *dummy interpolation case* will be performed.

As both the *dummy interpolation case* and the *early end case* handle situations where $u_{1,i} = 0$, the considered algorithm can be simplified by merging them together. Indeed, one can see that the dummy interpolation is using $\lambda_0 = \lambda_1 = 0$ which mean that no interpolation is actually performed at this step even if the q-degrees of the q-polynomials are increased. As a consequence, by modifying the *early end case* condition to $u_{1,j} = 0$ only (see algorithm 6), one can handle these two cases simultaneously. In fact, the *dummy interpolation cases* will be delayed to the end of the algorithm during the swap step but will never be performed as an admissible solution will be found as some point before we had to handle these cases. This is due to the fact that the dummy interpolation only increase the q-degrees of the q-polynomials without making any progress with respect to error correction. Therefore, our simpler algorithm always returns the q-polynomials of *minimal q-degrees* solving the reconstruction problem while the original algorithm may return any admissible solution.

The constant time decoding algorithm proposed in section 5 is based on our simpler algorithm. Hereafter, the term decoding algorithm refers to algorithm 6.

3.2 From decoding time to error weight

The decoding algorithm performs successive interpolations until the solution is found. As the *early end case* may end the main loop prematurely, the running

time of the algorithm may vary. Theorem 3 formalizes this observation as it shows that there exists a correlation between the rank of the error to be decoded and the decoding time of the Gabidulin code whenever the rank of the considered error is smaller than the error correcting capacity $\lfloor \frac{n-k}{2} \rfloor$.

Theorem 3. *Let* $\mathbf{G}$ *be the generator matrix of a Gabidulin code* $\mathcal{G}_{\mathbf{g}}(n,k,m)$, $\mathbf{m} \in \mathbb{F}_{q^m}^k$, $\mathbf{e} \in \mathbb{F}_{q^m}^n$ *such that* $\omega(\mathbf{e}) = t$ *with* $t \leq \lfloor \frac{n-k}{2} \rfloor$ *and* $\mathbf{y} = \mathbf{mG} + \mathbf{e}$. *Then, algorithm 6 will perform exactly* $2t$ *interpolation steps when solving* $\mathbf{Decoding}(\mathbf{y}, \mathcal{G}_{\mathbf{g}}, t)$.

Proof. The proof of theorem 3 follows from lemmas 1 and 2.

Lemma 1. *Under the same hypotheses than theorem 3, algorithm 6 will perform at least* $2t$ *interpolation steps when solving* $\mathbf{Decoding}(\mathbf{y}, \mathcal{G}_{\mathbf{g}}, t)$.

Proof. In order to retrieve an error $\mathbf{e}$, one need to find a q-polynomial V_1 such that $V_1(\mathbf{e}) = 0$. If $\omega(\mathbf{e}) = t$, then $\deg_q(V_1) \geq t$. Therefore, it follows from propostion 12 of [5] that algorithm 6 will perform at least $2t$ interpolation steps.

Lemma 2. *Under the same hypotheses than theorem 3, algorithm 6 will perform at most* $2t$ *interpolation steps when solving* $\mathbf{Decoding}(\mathbf{y}, \mathcal{G}_{\mathbf{g}}, t)$.

Proof. Let $n' = k + 2t$ and $\mathbf{e}' = (e_1, \ldots, e_{n'})$ be a shortened error such that $\mathsf{Supp}(\mathbf{e}') = \mathsf{Supp}(\mathbf{e})$. It is always possible to construct $\mathbf{e}'$ from $\mathbf{e}$ using a coordinates permutation followed by a truncation. Let $\mathcal{G}_{\mathbf{g}'}(n',k,m)$ be the shortened Gabidulin code generated by the matrix $\mathbf{G}'$ using the vector $\mathbf{g}' = (g_1, \ldots, g_{n'})$. As the error decoding capacity of $\mathcal{G}_{\mathbf{g}'}(n',k,m)$ is equal to $t = \lfloor \frac{n'-k}{2} \rfloor$, the vector $\mathbf{y}' = (y_1, \ldots, y_{n'}) = \mathbf{mG}' + \mathbf{e}'$ can be decoded using algorithm 6 in at most $2t$ interpolation steps. Let (N_1', V_1') be the solution retuned by algorithm 6, then every vector in $\mathsf{Supp}(\mathbf{e}')$ is a root of V_1' as well as every vector in $\mathsf{Supp}(\mathbf{e})$ because $\mathsf{Supp}(\mathbf{e}') = \mathsf{Supp}(\mathbf{e})$. It follows that (N_1', V_1') is a solution of the decoding problem induced by $\mathcal{G}_{\mathbf{g}}(n,k,m)$ and $\mathbf{y}$. As algorithm 6 outputs the q-polynomials of minimal q-degrees solving the reconstruction problem, decoding $\mathcal{G}_{\mathbf{g}}(n,k,m)$ is equivalent to decoding $\mathcal{G}_{\mathbf{g}'}(n',k,m)$ therefore algorithm 6 will perform at most $2t$ interpolation steps when solving $\mathbf{Decoding}(\mathbf{y}, \mathcal{G}_{\mathbf{g}}, t)$.

Corollary 1. *Let* $\mathbf{G}$ *be the generator matrix of a Gabidulin code* $\mathcal{G}_{\mathbf{g}}(n,k,m)$, $\mathbf{m} \in \mathbb{F}_{q^m}^k$, $\mathbf{e} \in \mathbb{F}_{q^m}^n$ *such that* $\omega(\mathbf{e}) = t$ *with* $t \leq \lfloor \frac{n-k}{2} \rfloor$ *and* $\mathbf{y} = \mathbf{mG} + \mathbf{e}$, *then it is possible to find* $\omega(\mathbf{e})$ *from the running time of algorithm 6.*

3.3 Error weight oracle for Gabidulin codes and RQC

Let $O_{\mathsf{Time}}^{\mathrm{Gab}}$ and $O_{\mathsf{Time}}^{\mathrm{RQC}}$ denote two timing oracles that return the running time of either the Gabidulin decoding algorithm or the RQC $\mathsf{Decapsulate}$ step. Following corollary 1, we now explain how to construct two oracles denoted $O_{\omega(\mathbf{e})}^{\mathrm{Gab}}$ and $O_{\omega(\mathbf{e})}^{\mathrm{RQC}}$ that return the rank $\omega(\mathbf{e})$ of the decoded error using respectively $O_{\mathsf{Time}}^{\mathrm{Gab}}$ and $O_{\mathsf{Time}}^{\mathrm{RQC}}$. The oracle $O_{\omega(\mathbf{e})}^{\mathrm{Gab}}$ takes as input a Gabidulin code $\mathcal{G}$ and a vector $\mathbf{y}$

while the oracle $O^{\text{RQC}}_{\omega(\mathbf{e})}$ takes as input an RQC public key pk (which implicitely defines a Gabidulin code) and a ciphertext ct.

Each oracle features an initialization step Init (see algorithm 1) and an evaluation step Eval (see algorithm 2). The Init step computes the expected running times required to decode an error $\mathbf{e}$ of given weight w for all $w \in [0, t]$. To this end, requests $O^{\text{Gab}}_{\mathsf{Time}}(\mathcal{G}, \mathbf{e})$ (respectively $O^{\text{RQC}}_{\mathsf{Time}}(\mathsf{pk}, (0, \mathbf{e}))$ are made using the message $\mathbf{m} = 0$ (respectively $\mathbf{m} = \mathbf{r}_1 = \mathbf{r}_2 = 0$) along with errors $\mathbf{e}$ of weight $i \in [0, t]$. The Eval step uses these expected running times $\mathbf{T}$ to output the rank of the error $\omega(\mathbf{e})$ by returning the index i such that $|\mathsf{time} - \mathbf{T}_i|$ is minimal where time denotes the result given by $O^{\text{Gab}}_{\mathsf{Time}}(\mathcal{G}, \mathbf{y})$ or $O^{\text{RQC}}_{\mathsf{Time}}(\mathsf{pk}, \mathsf{ct})$. The complexity of a $O^{\text{Gab}}_{\omega(\mathbf{e})}$ (respectively $O^{\text{RQC}}_{\omega(\mathbf{e})}$) request is equal to the complexity of a Gabidulin decoding (respectively an RQC decapsulation) namely $O(n^2)$ operations in $\mathbb{F}_{q^m}$.

Algorithm 1 Init step of $O^{\text{Gab}}_{\omega(\mathbf{e})}$ and $O^{\text{RQC}}_{\omega(\mathbf{e})}$

Input: $\begin{cases} \text{A Gabidulin code } \mathcal{G}(n, k, m) \text{ and access to } O^{\text{Gab}}_{\mathsf{Time}} \text{ for } O^{\text{Gab}}_{\omega(\mathbf{e})} \\ \text{A public key } \mathsf{pk} \text{ and access to } O^{\text{RQC}}_{\mathsf{Time}} \text{ for } O^{\text{RQC}}_{\omega(\mathbf{e})} \end{cases}$
A precision parameter param

Output: An array $\mathbf{T}$ of expected running times

$\mathbf{T} \longleftarrow (0, \cdots, 0) \in \mathbb{R}^{t+1}$
for $i \in \{0, \cdots, t\}$ **do**
  for $j \in \{1, \cdots, \mathsf{param}\}$ **do**
    $\mathbf{e} \xleftarrow{\$} \mathcal{S}_i^n$
    $\mathsf{time} \longleftarrow \begin{cases} O^{\text{Gab}}_{\mathsf{Time}}(\mathcal{G}, \mathbf{e}) \text{ for } O^{\text{Gab}}_{\omega(\mathbf{e})} \\ O^{\text{RQC}}_{\mathsf{Time}}(\mathsf{pk}, (0, \mathbf{e})) \text{ for } O^{\text{RQC}}_{\omega(\mathbf{e})} \end{cases}$
    $\mathbf{T}_{i+1} \longleftarrow \mathbf{T}_{i+1} + \mathsf{time}$
  $\mathbf{T}_{i+1} \longleftarrow \mathbf{T}_{i+1} \,/\, \mathsf{param}$
return $\mathbf{T}$

Algorithm 2 Eval step of $O^{\text{Gab}}_{\omega(\mathbf{e})}$ and $O^{\text{RQC}}_{\omega(\mathbf{e})}$

Input: $\begin{cases} \text{A Gabidulin code } \mathcal{G}(n, k, m) \text{ and a vector } \mathbf{y} \text{ for } O^{\text{Gab}}_{\omega(\mathbf{e})} \\ \text{A public key } \mathsf{pk} \text{ and a ciphertext } \mathsf{ct} \text{ for } O^{\text{RQC}}_{\omega(\mathbf{e})} \end{cases}$
An array $\mathbf{T}$ of expected running times from the Init step

Output: The rank $\omega(\mathbf{e})$ of the decoded error

$\mathsf{time} \longleftarrow \begin{cases} O^{\text{Gab}}_{\mathsf{Time}}(\mathcal{G}, \mathbf{y}) \text{ for } O^{\text{Gab}}_{\omega(\mathbf{e})} \\ O^{\text{RQC}}_{\mathsf{Time}}(\mathsf{pk}, \mathsf{ct}) \text{ for } O^{\text{RQC}}_{\omega(\mathbf{e})} \end{cases}$
return i such that $|\mathsf{time} - \mathbf{T}_i|$ is minimum

In order for these oracles to be usefull, each difference $\mathbf{T}_{i+1} - \mathbf{T}_i$ have to be large enough to be accurately measured. Experimental results (see section 5, figure 2) shows that for the considered machine, $\mathbf{T}_{i+1} - \mathbf{T}_i$ amounts for 6.6×10^4 CPU cycles (approximately 0.02 ms) for $O^{\text{RQC}}_{\omega(\mathbf{e})}$ in average. Such values allow timing attacks to be performed locally but would hardly be sufficient to allow an adverdary to perform a remote attack due to the variability of the network transfer times. Nevertheless, we assume hereafter that the existence of such an oracle is a potential threat for RQC and thus we choose to address it properly.

4 Timing attacks against RQC

In this section, we present two side-channel choosen ciphertext attacks against RQC. These attacks outperform the best known algorithm to solve the rank syndrome decoding problem [4] in term of complexity. Nonetheless, they require a huge number of requests to $O^{\text{RQC}}_{\omega(\mathbf{e})}$ therefore are quite unpracticable in real situations. We start by giving an overview of the attacks (section 4.1) then we describe two support recovery algorithms (section 4.2 and 4.3) that relies on $O^{\text{RQC}}_{\omega(\mathbf{e})}$ in order to bring the aforementionned improvement. Next, we present the complexity and the bandwidth cost of these attacks with respect to RQC parameters (section 4.4).

4.1 Overview of the attacks

The two attacks presented hereafter follow the same pattern. First, a support recovery algorithm is used to find $\mathbf{F} = \mathsf{Supp}(\mathbf{x}) = \mathsf{Supp}(\mathbf{y})$ then a linear system is solved in order to retrieve $\mathbf{x}$ and $\mathbf{y}$ thus revealing the secret key.

The support recovery algorithm makes several requests to $O^{\text{RQC}}_{\omega(\mathbf{e})}$ in order to find the support of $\mathbf{y}$. All these requests are constructed such that $\mathbf{m} = 0$, $\mathbf{r}_1 = 1$ and $\mathbf{r}_2 = 0$ namely the considered ciphertexts are of the form $(1, \mathbf{e})$. Recall from section 2.3 that decrypting a RQC ciphertext implies to decode $\mathbf{mG} + \mathbf{x} \cdot \mathbf{r}_2 - \mathbf{y} \cdot \mathbf{r}_1 + \mathbf{e}$. In this case, this will reduce to decoding $\mathbf{e} - \mathbf{y}$. The support recovery algorithm uses this particular form in order to retrieve $\mathbf{F} = \mathsf{Supp}(\mathbf{y})$.

Once the support $\mathbf{F}$ is known, one only need to solve the linear system $(1\ \mathbf{h}) \cdot (\mathbf{x}\ \mathbf{y})^\top = \mathbf{s}$ to find $\mathbf{x}$ and $\mathbf{y}$ as explained in [4]. This system can be obtained from the public key and features nm equations over $\mathbb{F}_q$ as well as $2wn$ unknowns over $\mathbb{F}_q$ because $\mathsf{dim}(\mathbf{F}) = w$. Given the RQC parameters, this system is always solvable and the secret key $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$ is its unique solution.

4.2 Simple support recovery algorithm

The simple support recovery strategy (see algorithm 3) tests all the elements $\alpha \in \mathbb{F}_{q^m}$ and checks whether they belong to the support $\mathbf{F}$ in order to retrieve one of its basis $(F_1, \cdots, F_w)$. To this end, a function $\psi : \mathbb{F}_{q^m} \longrightarrow \mathbb{F}_{q^m}$ that

deterministically enumerates the elements of $\mathbb{F}_{q^m}$ is defined. In addition, errors of the form $\mathbf{e} = (\alpha, 0, \cdots, 0) \in \mathbb{F}_{q^m}^n$ are considered. One can see that if $\mathsf{Supp}(\mathbf{e}) \subset \mathsf{Supp}(\mathbf{y})$, then $\omega(\mathbf{e} - \mathbf{y}) = w$ otherwise $\omega(\mathbf{e} - \mathbf{y}) = w + 1$. Using $O^{\text{RQC}}_{\omega(\mathbf{e})}$, one can retrieve the rank of $\mathbf{e} - \mathbf{y}$ thus learning if $\alpha \in \mathbf{F}$ or not.

Algorithm 3 Simple support recovery

Input: A public key pk and access to $O^{\text{RQC}}_{\omega(\mathbf{e})}$
The oracle precision parameter param

Output: $\mathbf{F} = \mathsf{Supp}(\mathbf{x}) = \mathsf{Supp}(\mathbf{y})$

$\mathbf{T} \longleftarrow O^{\text{RQC}}_{\omega(\mathbf{e})}.\mathsf{Init}(\mathsf{pk}, \mathsf{param})$
$\mathbf{F} \longleftarrow \langle 0 \rangle_{\mathbb{F}_q}$
$\alpha \longleftarrow 0 \in \mathbb{F}_{q^m}$
while $\mathsf{dim}(\mathbf{F}) < w$ **do**
    $\alpha \longleftarrow \psi(\alpha)$
    $\mathbf{e} \longleftarrow (\alpha,\ 0,\ \cdots,\ 0) \in \mathbb{F}_{q^m}^n$
    $\omega \longleftarrow O^{\text{RQC}}_{\omega(\mathbf{e})}.\mathsf{Eval}(\mathbf{T}, \mathsf{pk}, (1, \mathbf{e}))$
    if $\omega = w$ **then**
        $\mathbf{F} \longleftarrow \mathbf{F} + \langle \alpha \rangle_{\mathbb{F}_q}$
return F

Algorithm 3 requires $O(q^m)$ requests to the $O^{\text{RQC}}_{\omega(\mathbf{e})}$ oracle therefore its complexity is $O(n^2 q^m)$ operations in $\mathbb{F}_{q^m}$.

4.3 Advanced support recovery algorithm

The advanced support recovery strategy (see algorithm 4) is a generalization of the simple one in which we no longer consider errors of weigth $\omega(\mathbf{e}) = 1$ but rather errors of weight $\omega(\mathbf{e}) = t - w$. Instead of only checking if $\alpha \in \mathsf{Supp}(\mathbf{y})$, we look for any linear combination of the error's coordinates belonging to $\mathsf{Supp}(\mathbf{y})$ therefore speeding-up the algorithm. Without loss of generality, we only consider the case $q = 2$ since it matches the parameters used in RQC.

Given $\mathbf{a} \in \mathbb{F}_{q^m}^n$, let $(\overline{a}_1, \cdots, \overline{a}_{\omega(\mathbf{a})}) \in \mathbb{F}_{q^m}^{\omega(\mathbf{a})}$ denotes a basis of $\mathsf{Supp}(\mathbf{a})$. As $\omega(\mathbf{e}) = t - w$ and $\omega(\mathbf{y}) = w$, if $\omega(\mathbf{e} - \mathbf{y}) < t$ then there exists at least one non trivial linear combination of the vectors $(\overline{e}_i)_{i \in [1, t-w]}$ such that:

$$\sum_{i=1}^{t-w} \lambda_i \overline{e}_i = \sum_{j=1}^{w} \mu_j \overline{y}_j \in Supp(\mathbf{y})$$

The remaining of the algorithm compute the λ_i of such expressions thus retrieving a vector in the support $\mathbf{F}$. Each oracle request may lead to the discovery

of $\Delta = \omega(\mathbf{e}) + \omega(\mathbf{y}) - \omega(\mathbf{e}-\mathbf{y}) = \mathsf{dim}(\mathsf{ker}(\overline{y}_1 \ \cdots \ \overline{y}_w \ \overline{e}_1 \ \cdots \ \overline{e}_{t-w}))$ elements of $\mathbf{F}$ although Δ will be equal to 1 with overwhelming probability.

Let $\mathbf{M}_i \in \mathcal{M}_{m,w+i}(\mathbb{F}_q)$ be the matrices defined as $\mathbf{M}_i = \big(\overline{y}_1 \ \ldots \ \overline{y}_w \ \overline{e}_1 \ \ldots \ \overline{e}_i\big)$ for $i \in [1, t-w]$ and $d = \min\{i \mid \mathsf{rank}(\mathbf{M}_i) < w+i\}$. By construction, $\lambda_d = 1$. For $i \in [1,d]$, let $\mathbf{M}_{d,i} \in \mathcal{M}_{m,w+d}(\mathbb{F}_q)$ be the matrices defined as $\mathbf{M}_{d,i} = \big(\overline{y}_1 \ \cdots \ \overline{y}_w \ \overline{e}_1 \ \cdots \ \overline{e}_{i-1} \ 0 \ \overline{e}_{i+1} \ \cdots \ \overline{e}_d\big)$. If $\mathsf{rank}(\mathbf{M}_d) = \mathsf{rank}(\mathbf{M}_{d,i})$, then $\lambda_i = 1$. By performing this test for all $i \in [1,d]$, one can retrieve $\sum_{i=1}^{d} \lambda_i \overline{e}_i \in \mathsf{Supp}(\mathbf{y})$.

Algorithm 4 Advanced support recovery

Input: A public key pk and access to $O^{\mathrm{RQC}}_{\omega(\mathbf{e})}$
The oracle precision parameter param

Output: $\mathbf{F} = \mathsf{Supp}(\mathbf{x}) = \mathsf{Supp}(\mathbf{y})$

$\mathbf{T} \longleftarrow O^{\mathrm{RQC}}_{\omega(\mathbf{e})}.\mathsf{Init}(\mathsf{pk}, \mathsf{param})$
$\mathbf{F} \longleftarrow \langle 0 \rangle_{\mathbb{F}_q}$
while $\mathsf{dim}(\mathbf{F}) < w$ **do**
  $\mathbf{e} \xleftarrow{\$} \mathcal{S}^n_{t-w}$
  $\omega \longleftarrow O^{\mathrm{RQC}}_{\omega(\mathbf{e})}.\mathsf{Eval}(\mathbf{T}, \mathsf{pk}, (1, \mathbf{e}))$
  if $\omega < t$ **then** ▷ Collision detected
    $\Delta \longleftarrow t - \omega$
    $d \longleftarrow 0$
    $\omega' \longleftarrow 0$
    for $k \in \{1, \cdots, \Delta\}$ **do**

      ▷ Compute d
      repeat
        $d \longleftarrow d + 1$
        $\omega' \longleftarrow \omega' + 1$
        $\mathbf{e}' \longleftarrow (\overline{e}_1, \cdots, \overline{e}_d, 0, \cdots, 0) \in \mathbb{F}^n_{q^m}$
      until $O^{\mathrm{RQC}}_{\omega(\mathbf{e})}.\mathsf{Eval}(\mathbf{T}, \mathsf{pk}, (1, \mathbf{e}')) < w + \omega'$

      ▷ Compute λ
      $\lambda_d \longleftarrow 1$
      for $i \in \{1, \cdots, d-1\}$ **do**
        $\mathbf{e}' \longleftarrow (\overline{e}_1, \cdots, \overline{e}_{i-1}, 0, \overline{e}_{i+1}, \cdots, \overline{e}_d, 0, \cdots, 0) \in \mathbb{F}^n_{q^m}$
        if $O^{\mathrm{RQC}}_{\omega(\mathbf{e})}.\mathsf{Eval}(\mathbf{T}, \mathsf{pk}, (1, \mathbf{e}')) = w + \omega' - 1$ **then**
          $\lambda_i \longleftarrow 1$
        else
          $\lambda_i \longleftarrow 0$

      $\mathbf{F} \longleftarrow \mathbf{F} \ + \langle \sum_{i=1}^{d} \lambda_i \overline{e}_i \rangle_{\mathbb{F}_q}$

      $\overline{e}_d \longleftarrow 0$
      $\omega' \longleftarrow \omega' - 1$
return $\mathbf{F}$

Hereafter, we assume for simplicity that $\mathsf{rank}(\mathbf{e}) = t - w$ as it happens with high probability and can be enforced at no cost by tweaking the algorithm. The complexity of algorithm 4 is $O\left(wn^2/p\right)$ where p denotes the probability to find a non trivial intersection between $\mathsf{Supp}(\mathbf{e})$ and $\mathsf{Supp}(\mathbf{y})$ namely $p = P\left(\omega(\mathbf{e}-\mathbf{y}) < t \mid \omega(\mathbf{y}) = w \ \wedge \ \omega(\mathbf{e}) = t - w\right)$. The quantity $1-p$ represents the probability to pick the coordinates of $\mathbf{e}$ linearly independant from the coordinates of $\mathbf{y}$ knowing that $\omega(\mathbf{e}) = t - w$ and $\omega(\mathbf{y}) = w$. For each coordinate e_i, one have $q^m - q^{w+i}$ ways to pick it correctly amongst $q^m - q^i$ potential choices therefore:

$$1 - p = \prod_{i=0}^{t-w-1} \frac{q^m - q^{w+i}}{q^m - q^i} = \prod_{i=0}^{w-1} \frac{1}{q^m - q^i} \times \prod_{i=t-w}^{t-1} \frac{q^m - q^i}{1}$$

When considering the RQC parameters, one can approximate the complexity of algorithm 4 as $O\left(wn^2 q^{m-t}\right)$ operations in $\mathbb{F}_{q^m}$.

4.4 Attacks complexity and bandwith cost

As the linear system solving step of the attack is negligible with respect to the support recovery one, the attacks complexity is equal to the complexity of algorithms 3 and 4. Hereafter, we briefly describe a small improvement for these algorithms relying on the fact that $1 \in \mathsf{Supp}(\mathbf{y})$ in RQC. Indeed, one should note that if $a \notin \mathsf{Supp}(\mathbf{y})$, then $\forall \lambda \in \mathbb{F}_q, a + \lambda \notin \mathsf{Supp}(\mathbf{y})$. Thus, by setting $\mathbf{F} = \langle 1 \rangle_{\mathbb{F}_q}$ at the begining of the algorithms, one can choose error's coordinates from $\mathbb{F}_{q^m}/\langle 1 \rangle_{\mathbb{F}_q}$ instead of $\mathbb{F}_{q^m}$. Consequently, the simple attack has a complexity of $O(n^2 q^{m-1})$ operations in $\mathbb{F}_{q^m}$ and requires $O(q^{m-1})$ requests to the $O^{\mathrm{RQC}}_{\omega(\mathbf{e})}$ oracle. Similarly, the advanced attack has a complexity of $O(wn^2 q^{m-t-1})$ operations in $\mathbb{F}_{q^m}$ and requires $O(q^{m-t-1})$ requests to the $O^{\mathrm{RQC}}_{\omega(\mathbf{e})}$ oracle.

Table 1 presents the complexity and number of requests required to perform the attacks with respect to RQC parameters. One can see that both attacks outperform the best known algorithm to solve the rank syndrome decoding problem in term of complexity [4]. Nevertheless, they both require a huge number of requests to the $O^{\mathrm{RQC}}_{\omega(\mathbf{e})}$ oracle therefore are quite unpracticable in real situations.

Table 1. Attacks complexity and bandwith cost against RQC

	Complexity			Requests		
	128	192	256	128	192	256
Simple attack (§4.2)	2^{101}	2^{126}	2^{152}	2^{88}	2^{112}	2^{138}
Advanced attack (§4.3)	2^{73}	2^{86}	2^{106}	2^{58}	2^{70}	2^{90}

5 Preventing timing attacks against RQC

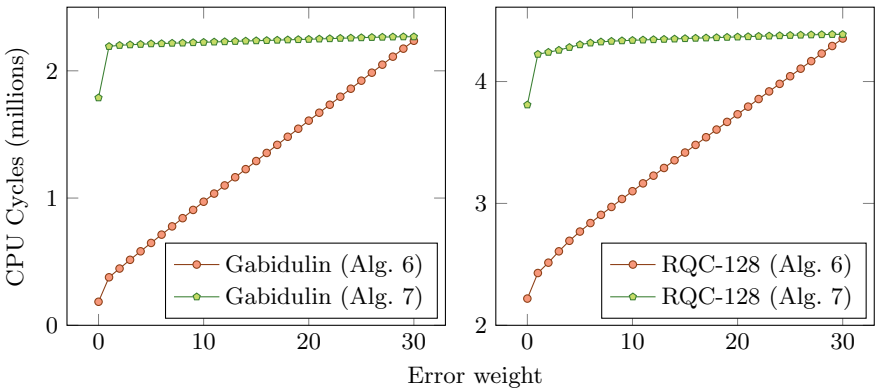
In this section, we explain how to prevent timing attacks against RQC using either a constant time decoding algorithm for Gabidulin codes (section 5.1) or a countermeasure based on the IND-CCA2 property of RQC (section 5.2). Interestingly, these two strategies can be implemented without any additional performance cost for honest users.

5.1 Constant time decoding of Gabidulin codes

Algorithm 7 provides a constant-time implementation of the reconstruction algorithm and as such can be used to prevent timing attacks against RQC. The main idea is to perform operations on dummy q-polynomials (lines 2-5) whenever required (lines 6-17) while ensuring that every operations is performed on a q-polynomial of correct q-degree with respect to a nominal case (lines 25-34). As a result, given a Gabidulin code $\mathcal{G}_{\mathbf{g}}(n,k,m)$, and an error $\mathbf{e} \in \mathbb{F}_{q^m}^n$ such that $\omega(\mathbf{e}) = t$ with $t \leq \lfloor \frac{n-k}{2} \rfloor$, algorithm 7 will perform exactly $2 \times \lfloor \frac{n-k}{2} \rfloor$ interpolation steps whatever the value of t is.

Figure 2 compares the running time of algorithms 6 and 7 when they are respectively used to decode Gabidulin codes or used as part of the Decapsulate step of RQC. We have performed 10 000 tests for each error weight using a computer equiped with an Intel Core i7-7820X CPU @ 3.6 GHz and 16GB of memory. On average (excluding the case $\omega(\mathbf{e}) = 0$ which is discussed below), $\mathbf{T}_{i+1} - \mathbf{T}_i$ is reduced from 6.6×10^4 to 5.6×10^3 CPU cycles (approximately 2 μs) for $O^{\text{RQC}}_{\omega(\mathbf{e})}$. This is sufficient to prevent timing attacks as an adversary cannot realize reliable timing measurements anymore. Indeed, the average standard deviation to the running time observed for each error weight is equal to 1.4×10^4 CPU cycles thus rendering the aforementioned timing attacks impracticable.

Fig. 2. Running time (CPU cycles) of Gabidulin code decoding and RQC-128 Decapsulate step with respect to different error weights $\omega(\mathbf{e})$ using algorithms 6 and 7



By analyzing figure 2, one immediately sees that the special case $\omega(\mathbf{e}) = 0$ is an outlier with respect to the running time of algorithm 7. This is presumably due to the fact that the involved q-polynomials have many coefficients equal to zero which speeds the q-polynomial update step of the decoding. This case does not appear to be concerning as it seems hard to retrieve information regarding the support $\mathbf{F}$ whenever $\omega(\mathbf{x} \cdot \mathbf{r}_2 - \mathbf{y} \cdot \mathbf{r}_1 + \mathbf{e}) = 0$. One may exploit this special case by trying to find errors $\mathbf{e}$ such that $\mathbf{e} - \mathbf{y} = 0$ (with $\mathbf{r}_1 = 1$ and $\mathbf{r}_2 = 0$) nonetheless such an attack would have a complexity of $O(q^{\omega n})$ operations in $\mathbb{F}_{q^m}$ which is worse than solving the rank syndrome decoding problem.

The running time of algorithm 7 is similar to the running-time required to decode an error of weight $\omega(\mathbf{e}) = \lfloor \frac{n-k}{2} \rfloor$ using algorithm 6. As the weight of the error $\mathbf{x} \cdot \mathbf{r}_2 - \mathbf{y} \cdot \mathbf{r}_1 + \mathbf{e}$ used in RQC is equal to the error correction capacity of the considered Gabidulin code, our constant time algorithm can be used without any additional performance cost for honest users.

5.2 Countermeasure based on RQC IND-CCA2 property

RQC being an IND-CCA2 encryption scheme, any attempt to modify one of its ciphertexts will be detected and the Decapsulate step will end-up by an Abort. By correctly implementing the Abort behaviour, the aforementioned timing attacks can be prevented. Indeed, one may choose to not respond to invalid requests therefore preventing the adversary to perform any time measurement. Alternatively, one can wait a randomly chosen amount of time before sending its response thus forcing an adversary to perform a huge number of requests in order to get any reliable time measurement. As both of these strategies intervene after an Abort case is detected, they can be implemented without any additional performance cost for honest users.

6 Conclusion

In this paper, we have highlighted a correlation between the rank of the error to be decoded and the running time of Loidreau's decoding algorithm for Gabidulin codes. We have also described two choosen ciphertext timing attacks against RQC that are based on this correlation. In addition, we have provided two countermeasures preventing the aforementionned attacks. The first one relies on a constant time decoding algorithm for Gabidulin codes and second one uses the IND-CCA2 property of RQC. As both of these countermeasures can be deployed without additional performance cost for honest users, we suggest to implement both of them. In a future work, we will conduct a similar analysis on the HQC [1] encryption scheme in order to study its resistance to timing attacks. Indeed, as the latter shares the same framework than RQC in the Hamming setting, it might be threatened by similar attacks.

References

1. Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. Hamming Quasi-Cyclic (HQC).
2. Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Rank Quasi-Cyclic (RQC).
3. Carlos Aguilar-Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient Encryption from Random Quasi-Cyclic Codes. *IEEE Transactions on Information Theory*, 64(5):3927–3943, 2018.
4. Nicolas Aragon, Philippe Gaborit, Adrien Hauteville, and Jean-Pierre Tillich. A New Algorithm for Solving the Rank Syndrome Decoding Problem. In *2018 IEEE International Symposium on Information Theory (ISIT)*, pages 2421–2425, 2018.
5. Daniel Augot, Pierre Loidreau, and Gwezheneg Robert. Generalized Gabidulin codes over fields of any characteristic. *Designs, Codes and Cryptography*, 86(8):1807–1848, 2018.
6. Ernest Mukhamedovich Gabidulin. Theory of codes with maximum rank distance. *Problemy Peredachi Informatsii*, 21(1):3–16, 1985.
7. Philippe Gaborit and Gilles Zémor. On the hardness of the decoding and the minimum distance problems for rank codes. *IEEE Transactions on Information Theory*, 62(12):7245–7252, 2016.
8. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.
9. Pierre Loidreau. A Welch–Berlekamp like algorithm for decoding Gabidulin codes. In *Coding and cryptography*, pages 36–45. Springer, 2006.
10. Oystein Ore. On a special class of polynomials. *Transactions of the American Mathematical Society*, 35(3):559–584, 1933.

A Original reconstruction algorithm

Algorithm 5 Original reconstruction algorithm [9, 5]

Input: $k, n \in \mathbb{N}, k \leq n$
$\mathbf{g} = (g_1, \cdots, g_n) \in \mathbb{F}_{q^m}^n$, $\mathbb{F}_q$-linearly independent elements
$\mathbf{y} = (y_1, \cdots, y_n) \in \mathbb{F}_{q^m}^n$

Output: (N_1, V_1), solution to **Reconstruction**$(\mathbf{y}, \mathbf{g}, k, t)$.

▷ Initialization step
$N_0(X) \longleftarrow \mathcal{A}_{\langle g_1, \cdots, g_k \rangle_{\mathbb{F}_q}}$
$V_0(X) \longleftarrow 0$
$N_1(X) \longleftarrow \mathcal{I}_{[g_1, \cdots, g_k],[y_1, \cdots, y_k]}$
$V_1(X) \longleftarrow 1$
$\mathbf{u}_0 \longleftarrow N_0\{\mathbf{g}\} - V_0\{\mathbf{y}\}$
$\mathbf{u}_1 \longleftarrow N_1\{\mathbf{g}\} - V_1\{\mathbf{y}\}$

▷ Interpolation step
for $i \in \{k+1, \cdots, n\}$ **do**
  $j \longleftarrow i$ ▷ Early-end case
  while $j \leq n$ **and** $u_{1,j} = 0$ **and** $u_{0,j} \neq 0$ **do**
    $j \longleftarrow j + 1$
  if $j = n + 1$ **then**
    break
  else
    $u_{0,i} \longleftrightarrow u_{0,j}$
    $u_{1,i} \longleftrightarrow u_{1,j}$

  ▷ q-polynomials update
  if $u_{1,i} \neq 0$ **then** ▷ Nominal case
    $\lambda_1 \longleftarrow \frac{\theta(u_{1,i})}{u_{1,i}}$
    $\lambda_0 \longleftarrow \frac{u_{0,i}}{u_{1,i}}$
  else if $u_{0,i} = 0$ **then** ▷ Dummy interpolation case
    $\lambda_1 \longleftarrow 0$
    $\lambda_0 \longleftarrow 0$
  $N_1' \longleftarrow (X - \lambda_1) \cdot N_1$
  $V_1' \longleftarrow (X - \lambda_1) \cdot V_1$
  $N_0' \longleftarrow N_0 - \lambda_0 \cdot N_1$
  $V_0' \longleftarrow V_0 - \lambda_0 \cdot V_1$

  ▷ q-polynomials swap
  $N_0 \longleftarrow N_1'$
  $V_0 \longleftarrow V_1'$
  $N_1 \longleftarrow N_0'$
  $V_1 \longleftarrow V_0'$

  ▷ Discrepancies update
  for $j \in \{i+1, \cdots, n\}$ **do**
    $u_{0,j}' \longleftarrow \theta(u_{1,j}) - \lambda_1 \cdot u_{1,j}$
    $u_{1,j}' \longleftarrow u_{0,j} - \lambda_0 \cdot u_{1,j}$

return (N_1, V_1)

B Simpler reconstruction algorithm

Algorithm 6 Simplier reconstruction algorithm (§3.1)

Input: $k, n \in \mathbb{N}, k \leq n$
$\mathbf{g} = (g_1, \cdots, g_n) \in \mathbb{F}_{q^m}^n, \mathbb{F}_q$-linearly independent elements
$\mathbf{y} = (y_1, \cdots, y_n) \in \mathbb{F}_{q^m}^n$

Output: (N_1, V_1), solution to **Reconstruction**$(\mathbf{y}, \mathbf{g}, k, t)$.

▷ Initialization step
$N_0(X) \longleftarrow \mathcal{A}_{\langle g_1, \cdots, g_k \rangle_{\mathbb{F}_q}}$
$V_0(X) \longleftarrow 0$
$N_1(X) \longleftarrow \mathcal{I}_{[g_1, \cdots, g_k],[y_1, \cdots, y_k]}$
$V_1(X) \longleftarrow 1$
$\mathbf{u}_0 \longleftarrow N_0\{\mathbf{g}\} - V_0\{\mathbf{y}\}$
$\mathbf{u}_1 \longleftarrow N_1\{\mathbf{g}\} - V_1\{\mathbf{y}\}$

▷ Interpolation step
for $i \in \{k+1, \ldots, n\}$ **do**
  $j \longleftarrow i$ ▷ Early-end case
  while $j \leq n$ **and** $u_{1,j} = 0$ **do**
    $j \longleftarrow j + 1$
  if $j = n + 1$ **then**
    break
  else
    $u_{0,i} \longleftrightarrow u_{0,j}$
    $u_{1,i} \longleftrightarrow u_{1,j}$

  ▷ q-polynomials update
  $\lambda_1 \longleftarrow \frac{\theta(u_{1,i})}{u_{1,i}}$ ▷ Nominal case
  $\lambda_0 \longleftarrow \frac{u_{0,i}}{u_{1,i}}$
  $N_1' \longleftarrow (X - \lambda_1) \cdot N_1$
  $V_1' \longleftarrow (X - \lambda_1) \cdot V_1$
  $N_0' \longleftarrow N_0 - \lambda_0 \cdot N_1$
  $V_0' \longleftarrow V_0 - \lambda_0 \cdot V_1$

  ▷ q-polynomials swap
  $N_0 \longleftarrow N_1'$
  $V_0 \longleftarrow V_1'$
  $N_1 \longleftarrow N_0'$
  $V_1 \longleftarrow V_0'$

  ▷ Discrepancies update
  for $j \in \{i+1, \cdots, n\}$ **do**
    $u_{0,j}' \longleftarrow \theta(u_{1,j}) - \lambda_1 \cdot u_{1,j}$
    $u_{1,j}' \longleftarrow u_{0,j} - \lambda_0 \cdot u_{1,j}$

return (N_1, V_1)

C Constant-time reconstruction algorithm

Algorithm 7 Constant-time reconstruction algorithm (§5)

Input: $k, n \in \mathbb{N}, k \leq n$
$\mathbf{g} = (g_1, \cdots, g_n) \in \mathbb{F}_{q^m}^n$, $\mathbb{F}_q$-linearly independent elements
$\mathbf{y} = (y_1, \cdots, y_n) \in \mathbb{F}_{q^m}^n$

Output: (N_1, V_1), solution to **Reconstruction**$(\mathbf{y}, \mathbf{g}, k, t)$.

1: ▷ Classical initialization step (see algorithm 6, lines 1 - 6)

▷ Constant-time initialization step
2: $d \longleftarrow ((n-k)/2 \equiv 0 \bmod 2)\ ?\ k+t-1\ :\ k+t$
3: $N_2, N_3, V_2, V_3 \xleftarrow{\$}$ { q-polynomials of q-deg d }
4: $c_0, c_1 \xleftarrow{\$} \mathbb{F}_{q^m} \backslash \{0\}$
5: $b \longleftarrow 0$

▷ Interpolation step
6: **for** $i \in \{k+1, \cdots, n\}$ **do**
7: $\quad i' \longleftarrow n+1$ ▷ "Early-end" case
8: $\quad$ **for** $j \in \{i, \cdots, n\}$ **do**
9: $\quad\quad r \longleftarrow \mathsf{isZero}(u_{1,j})$
10: $\quad\quad i' \longleftarrow (1-r)j + ri'$
11: $\quad$ **if** $i' = n+1$ **or** $b = 1$ **then**
12: $\quad\quad b \longleftarrow 1$
13: $\quad\quad u_{0,i} \longleftarrow c_0$
14: $\quad\quad u_{1,i} \longleftarrow c_1$
15: $\quad$ **else**
16: $\quad\quad u_{0,i} \longleftrightarrow u_{0,i'}$
17: $\quad\quad u_{1,i} \longleftrightarrow u_{1,i'}$

▷ q-polynomials update
18: $\quad \lambda_1 \longleftarrow \frac{\theta(u_{1,i})}{u_{1,i}}$
19: $\quad \lambda_0 \longleftarrow \frac{u_{0,i}}{u_{1,i}}$
20: $\quad$ **if** $b = 0$ **then** ▷ Classical nominal case
21: $\quad\quad N_1' \longleftarrow (X - \lambda_1) \cdot N_1$
22: $\quad\quad V_1' \longleftarrow (X - \lambda_1) \cdot V_1$
23: $\quad\quad N_0' \longleftarrow N_0 - \lambda_0 \cdot N_1$
24: $\quad\quad V_0' \longleftarrow V_0 - \lambda_0 \cdot V_1$
25: $\quad$ **else if** $i - k \equiv 0 \bmod 2$ **then** ▷ Constant-time nominal case
26: $\quad\quad N_1 \longleftarrow (X - \lambda_1) \cdot N_1$
27: $\quad\quad V_1 \longleftarrow (X - \lambda_1) \cdot V_1$
28: $\quad\quad N_0' \longleftarrow N_2 - \lambda_0 \cdot N_3$
29: $\quad\quad V_0' \longleftarrow V_2 - \lambda_0 \cdot V_3$
30: $\quad$ **else**
31: $\quad\quad N_1' \longleftarrow (X - \lambda_1) \cdot N_3$
32: $\quad\quad V_1' \longleftarrow (X - \lambda_1) \cdot V_3$
33: $\quad\quad N_0' \longleftarrow N_2 - \lambda_0 \cdot N_3$
34: $\quad\quad V_0' \longleftarrow V_2 - \lambda_0 \cdot V_3$

35: $\quad$ ▷ Classical q-polynomials swap (see algorithm 6, lines 22 - 25)
36: $\quad$ ▷ Classical discrepancies update (see algorithm 6, lines 26 - 28)

37: **return** (N_1, V_1)