Cryptanalysis of a New Code-based Signature Scheme with Shorter Public Key in PKC 2019

Keita Xagawa

NTT Secure Platform Laboratories
3-9-11, Midori-cho Musashino-shi, Tokyo 180-8585 Japan
keita.xagawa.zv@hco.ntt.co.jp

Abstract. Song, Huang, Mu, and Wu proposed a new code-based signature scheme, the Rank Quasi-Cyclic Signature (RQCS) scheme (PKC 2019, Cryptology ePrint Archive 2019/053), which is based on an IND-CCA2 KEM scheme, RQC, proposed by Aguilar Melchor et al. (NIST PQC Standardization Round 1). Their scheme is an analogue to the Schnorr signature scheme.
In this short note, we investigate the security of RQCS. We report a key-recovery known-message attack by following the discussion in Aragon, Blazy, Gaborit, Hauteville, and Zémor (Cryptology ePrint Archive 2018/1192) and an experimental result. The key-recovery attack requires only one signature to retrieve a secret key and recovers a secret key within 10 seconds.
keywords: NIST PQC, post-quantum digital signatures, cryptanalysis, coding-based cryptography

1 Introduction

RQC is an efficient KEM scheme proposed by Aguilar Melchor et al. [AMAB+17, AMBD+18], which is based on hard problems related to rank-metric quasi-cyclic codes. Song, Huang, Mu, and Wu proposed the Rank Quasi-Cyclic Signature (RQCS) scheme [SHMW19] upon key's structure of RQC, which can be considered as analogue to the Schnorr signature scheme [Sch90]. Song et al. insisted that RQCS is existentially unforgeable under adaptive chosen-message attacks (EUF-CMA secure) in the random oracle model if the syndrome decoding problem related to the key pair's structure in the rank metric is hard.

Our contribution: We give a key-recovery known-message attack by following the discussion in Persichetti [Per12] and the discussion in Aragon, Blazy, Gaborit, Hauteville, and Zémor [ABG+18, Section 3.1]. Persichetti discussed applicability of the Schnorr identification to the code setting in his thesis [Per12, Section 7.3.2]. He concluded that sufficient number of transcripts leak a whole secret in the rank-metric setting if challenges are chosen from $\mathbb{F}_q$. Aragon et al. proposed Durandal, a new code-based signature scheme based on a syndrome decoding problem in the rank metric. They designed Durandal by following the Schnorr signature scheme but gave some tweaks, because a signature will leak a secret key if they follow the Schnorr signature scheme directly and a challenge is chosen from $\mathbb{F}_{q^m}^{n-k}$ with low-rank.[1] However, RQCS *directly* follows the Schnorr signature scheme. Thus, the attacks in [Per12, Section 7.3.2] and [ABG+18, Section 3.1] can be applied to RQCS. The key-recovery attack requires only one signature to retrieve a secret key and recovers a key less than 10 seconds even on SageMath.

Related Works: Concurrently, Lau and Tan [LT19] propose an experimental attack on Magma. Their attack is the same as the attack in this paper.

In the Hamming-metric setting, Persichetti discussed applicability of the Schnorr identification to the code setting in his thesis [Per12, Section 7.3.2]. He concluded that a transcript leaks information of the secret and thus, the direct application of the Schnorr identification (and signature) is not secure for ordinal signature scheme.

RaCoSS is a new signature scheme based on the syndrome decoding problem of a random parity-check matrix in the Hamming-metric setting [FRXKMT17]. It can be a direct application of the Schnorr signature scheme. It is broken because of the small parameter values [BHLP17], patched [RMFKT18], and broken again [Xag18].

Persichetti proposed an analogue of the Schnorr signature in the Hamming-metric code setting as *one-time* signature [Per18]. Santini, Baldi, and Chiaraluce [SBC18] cryptanalyzed Persichetti's one-time signature scheme by discussing the corresponding decoding problem is easier than Persichetti thought. Moreover, Deneuville and Gaborit [DG18] cryptanalyzed the scheme by using an efficient bit-flipping decoding algorithm for random moderate-density parity-check codes.

[1] We note that, in the lattice-based signature context, Lyubashevsky also gave a tweak, the rejection-sampling technique, in his Schnorr-like signature scheme [Lyu08].

Organization: Section 2 reviews basic notions and notations. Section 3 reviews the signature scheme, RQCS. Section 4 gives the attack and the experimental result.

2 Preliminaries

For a positive integer n, we define $\mathcal{R} := \mathbb{F}_{q^m}[X]/(X^n - 1)$, the quotient ring of $\mathbb{F}_{q^m}$-coefficient polynomial modulo $X^n - 1$. We will identify an element $\boldsymbol{a}$ of $\mathcal{R}$ as a row vector $(a_1, \dots, a_n) \in \mathbb{F}_{q^m}^n$ and a polynomial $a_1 + a_2X + \cdots + a_nX^{n-1}$.

Rank Metric: We review the rank metric.

Definition 2.1 (Rank metric over $\mathbb{F}_{q^m}^n$). *Let $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{F}_{q^m}^n$. Let $\beta_1, \dots, \beta_m \in \mathbb{F}_{q^m}$ be a basis of $\mathbb{F}_{q^m}$ viewed as an m-dimensional vector space over $\mathbb{F}_q$. Each coordinate x_j is associated to a vector of $\mathbb{F}_q^m$ as $x_j = \sum_{i=1}^m x_{ij}\beta_j$ with $x_{ij} \in \mathbb{F}_q$. We associate $\boldsymbol{x}$ with a matrix $\boldsymbol{M_x}$ defined as*

$$\boldsymbol{M_x} := \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mn} \end{bmatrix} \in \mathbb{F}_q^{m \times n}.$$

The rank weight $\|\boldsymbol{x}\|$ of $\boldsymbol{x}$ is defined as

$$\|\boldsymbol{x}\| := \mathrm{rank}(\boldsymbol{M_x}).$$

The distance $d_R(\boldsymbol{x}, \boldsymbol{y})$ between elements $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_{q^m}^n$ is defined by

$$d_R(\boldsymbol{x}, \boldsymbol{y}) := \|\boldsymbol{x} - \boldsymbol{y}\|.$$

We omit the definition of $\mathbb{F}_{q^m}$-linear codes. See the textbooks, e.g., [Rot06].

Definition 2.2 (Support of a word). *Let $\boldsymbol{x} = (x_1, \dots, x_n) \in \mathbb{F}_{q^m}^n$. The support of $\boldsymbol{x}$, denoted by* $\mathrm{Supp}(\boldsymbol{x})$*, is the $\mathbb{F}_q$-subspace of $\mathbb{F}_{q^m}$ generated by the coordinates of $\boldsymbol{x}$: that is,*

$$\mathrm{Supp}(\boldsymbol{x}) := \langle x_1, \dots, x_n \rangle_{\mathbb{F}_q}.$$

We have $\dim \mathrm{Supp}(\boldsymbol{x}) = \|\boldsymbol{x}\|$.

The number of supports of dimension w of $\mathbb{F}_{q^m}$ is denoted by the Gaussian binomial coefficient;

$$\binom{m}{w}_q = \prod_{i=0}^{w-1} \frac{q^m - q^i}{q^w - q^i}.$$

Definition 2.3 (Product of supports). *Let E and F be $\mathbb{F}_q$-linear subspaces of $\mathbb{F}_{q^m}$. Their* product *is defined as $\mathbb{F}_q$-linear subspace spanned by*

$$\{e \cdot f \mid e \in E, f \in F\}.$$

Quasi-Circulant Codes: The circulant matrix induced by $\boldsymbol{x} \in \mathcal{R}$ is defined as

$$\mathrm{rot}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{x}X \\ \dots \\ \boldsymbol{x}X^{n-1} \end{bmatrix} = \begin{bmatrix} x_0 & x_1 & \dots & x_{n-1} \\ x_{n-1} & x_0 & \dots & x_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ x_1 & x_2 & \dots & x_0 \end{bmatrix} \in \mathbb{F}_{q^m}^{n \times n}.$$

Definition 2.4 (Quasi-circulant codes). *For positive integers n, ℓ, m and a power of prime q, a $[\ell n, n]_{q^m}$ linear code C is said to be* quasi-circulant *if it has a generator matrix $\boldsymbol{G}$ of the form $\boldsymbol{G} := [\boldsymbol{G}_1 \mid \cdots \mid \boldsymbol{G}_\ell]$ where each $\boldsymbol{G}_i$ is circulant matrix of size n. If ℓ is two, we say double-circulant.*

Problems and assumptions: Let us rephrase our syndrome decoding problem as follows:

Definition 2.5 (Rank Syndrome Decoding (RSD) Problem). *The input is a parity-check matrix $\boldsymbol{H} \in \mathbb{F}_{q^m}^{(n-k)\times n}$ of an $[n,k]$ $\mathbb{F}_{q^m}$-linear code, a syndrome $\boldsymbol{s} \in \mathbb{F}_{q^m}^{n-k}$, and a positive integer r. The $\mathsf{RSD}_{q,m,n,k,r}$ problem is finding $\boldsymbol{e} \in \mathbb{F}_{q^m}^{n}$ such that $\|\boldsymbol{r}\| = w$ and $\boldsymbol{s} = \boldsymbol{r} \cdot \boldsymbol{H}^\top$.*

The rank quasi-cyclic syndrome decoding (RQCSD) problem is defined as the special case of the RSD problem.

Definition 2.6 (Rank Quasi-Cyclic Syndrome Decoding (RQCSD) Problem). *The input is $\boldsymbol{h} \in \mathcal{R}$ which defines a systematic parity-check matrix $\boldsymbol{H} = [\boldsymbol{I}_n \mid \mathrm{rot}(\boldsymbol{h})^\top] \in \mathbb{F}_{q^m}^{n\times 2n}$ of a double-circulant $[2n,n]_{q^m}$ code C, a syndrome $\boldsymbol{s} \in \mathbb{F}_{q^m}^{n}$, and a positive integer w. The problem is finding $(\boldsymbol{x}_1,\boldsymbol{x}_2) \in \mathbb{F}_{q^m}^{2n}$ such that $\boldsymbol{x}_1 + \boldsymbol{h}\cdot\boldsymbol{x}_2 = \boldsymbol{s}$ and $\|\boldsymbol{x}_1\| = \|\boldsymbol{x}_2\| = w$, where $\boldsymbol{x}\cdot\boldsymbol{H}^\top = \boldsymbol{s}$.*

Gaborit and Zémor showed that the RSD problem is difficult with a probabilistic reduction to the Hamming setting in [GZ16]. If the parity-check matrix is low-rank, there are efficient decoding algorithms [ABG+13, AGHT18]

3 Review of RQCS

Song, Huang, Mu, and Wu proposed RQCS [SHMW19]. For a positive integer w, we define $\mathcal{R}_w$ as a set of elements in $\mathcal{R}$ whose weight is w, that is, $\mathcal{R}_w := \{\boldsymbol{a} \in \mathcal{R} \mid \|\boldsymbol{a}\| = w\}$. The scheme employs a hash function $\mathsf{Hash}\colon \mathcal{R} \times \{0,1\}^* \to \mathcal{R}_{w_g}$, which is modeled as a random oracle. RQCS is described as follows:

- $\mathsf{Setup}(1^\kappa) \to pp$: It outputs $pp = (n, w, w_r, w_g)$.
- $\mathsf{Gen}(pp) \to (vk, sk)$: Choose $\boldsymbol{h} \leftarrow \mathcal{R}$ and $\boldsymbol{x},\boldsymbol{y} \leftarrow \mathcal{R}_w$. Compute $\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h}\cdot\boldsymbol{y}$. Output $vk = (\boldsymbol{h},\boldsymbol{s})$ and $sk = (\boldsymbol{x},\boldsymbol{y})$.
- $\mathsf{Sign}(sk, M) \to \sigma$: Choose $\boldsymbol{r} = (\boldsymbol{r}_1,\boldsymbol{r}_2) \leftarrow \mathcal{R}_{w_r} \times \mathcal{R}_{w_r}$. Compute $I = \boldsymbol{r}_1 + \boldsymbol{h}\cdot\boldsymbol{r}_2$. Compute $\boldsymbol{g} = \mathsf{Hash}(I, M) \in \mathcal{R}_{w_g}$. Compute $\boldsymbol{u} = (\boldsymbol{u}_1,\boldsymbol{u}_2) = \boldsymbol{g}\cdot(\boldsymbol{x},\boldsymbol{y}) + \boldsymbol{r} \in \mathcal{R}^2$. Output $\sigma = (\boldsymbol{g},\boldsymbol{u})$.
- $\mathsf{Vrfy}(vk, M, \sigma) \to 0/1$: Compute $I' = \boldsymbol{u}_1 + \boldsymbol{h}\cdot\boldsymbol{u}_2 - \boldsymbol{s}\cdot\boldsymbol{g}$. Output 1 if and only if $\mathsf{Hash}(I, M) = \boldsymbol{g}$ and $\|\boldsymbol{u}_1\|, \|\boldsymbol{u}_2\| \le B$.

The parameter sets are summarized in Table 1.

Table 1: Parameter Sets for the RQCS scheme: Song et al. defined $w = w_r = w_g$.

Instance	q	n	m	w	w_r	w_g	δ	Security
RQCS-1	2	67	89	5	5	5	31	128
RQCS-2	2	97	121	6	6	6	43	192
RQCS-3	2	101	139	6	6	6	48	256

Theorem 3.1 ([SHMW19, Theorem 1]). *If the RQCSD problem is hard and* Hash *is modeled as a random oracle, then the RQCS scheme is existentially unforgeable under adaptive chosen-message attacks.*

4 Attack

Idea: We notice that Aragon et al. [ABG+18] proposed a similar signature scheme, Durandal, based on a variant of the rank syndrome decoding problem. They discussed why they do not directly follow the Schnorr signature scheme in [ABG+18, Section 3.1].

> The problem with this approach in the rank metric is that adding y to cS does not hide cS properly. Indeed, the verifier, or any witness to the protocol of Figure 1, can recover the support of the secret matrix S even after a single instance of the protocol, using techniques from the decoding of LRPC codes

[15]: [2] since the verifier has c he can choose a basis $f_1, \dots, f_d$ of $\mathrm{Supp}(c)$ and then with high probability it will occur that:

$$\bigcap_{i=1}^{d} f_i^{-1}\mathrm{Supp}(z) = \mathrm{Supp}(S)$$

and with the support of S the verifier can compute S explicitly from the linear equations $HS^T = T$.

Letting $y = \boldsymbol{r}$, $c = \boldsymbol{g}$, $S = (\boldsymbol{x}, \boldsymbol{y})$, $z = \boldsymbol{u}$, $H = (\mathbf{1}, \boldsymbol{h})$, and $T = \boldsymbol{s}$, the above argument says that if we know the basis of $\mathrm{Supp}(c) = \mathrm{Supp}(\boldsymbol{g})$, $z = \boldsymbol{u}$ will leak the support of $S = (\boldsymbol{x}, \boldsymbol{y})$. By combining this knowledge with the relation of $\boldsymbol{h} = \boldsymbol{x} + \boldsymbol{h}\boldsymbol{y}$, we can find $S = (\boldsymbol{x}, \boldsymbol{y})$.

Let us consider a signature $\sigma = (\boldsymbol{g}, \boldsymbol{u})$ with $\boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2)$. We have

$$\boldsymbol{u}_1 = \boldsymbol{g}\boldsymbol{x} + \boldsymbol{r}_1 \text{ and } \boldsymbol{u}_2 = \boldsymbol{g}\boldsymbol{y} + \boldsymbol{r}_2.$$

Let $E_1 = \langle e_{11}, \dots, e_{1w} \rangle$ and $E_2 = \langle e_{21}, \dots, e_{2w} \rangle$ be the supports of $\boldsymbol{x}$ and $\boldsymbol{y}$, respectively. Let $R_b = \langle r_{b1}, \dots, r_{bw_r} \rangle$ be the supports of $\boldsymbol{r}_b$ for $b = 1$ and 2. Let $F = \langle f_1, \dots, f_{w_g} \rangle$ be the support of $\boldsymbol{g}$. Let us define two syndrome spaces $S_1 = \mathrm{Supp}(\boldsymbol{u}_1)$ and $S_2 = \mathrm{Supp}(\boldsymbol{u}_2)$. We have

$$S_b \subseteq E_b.F + R_b,$$

where $E_b.F = \langle e_{b1} \cdot f_1, \dots, e_{bw} \cdot f_{w_g} \rangle$. If $S_b = E_b.F + R_b$, then $E_b \subseteq f_i^{-1} S_b$ for each $i = 1, \dots, w_g$. Thus, it will occur that

$$E_b = \bigcap_{i=1}^{w_g} f_i^{-1} S_b.$$

If we find the supports E_1 and E_2 for $\boldsymbol{x}$ and $\boldsymbol{y}$, then we can compute $\boldsymbol{x}$ and $\boldsymbol{y}$ by letting $x_i = \sum_j \lambda_{ij} e_{1j}$ and $y_i = \sum_j \rho_{ij} e_{2j}$ and solving the system of nm equations over $\mathbb{F}_q$ corresponding to $\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y}$ with $2nw$ unknowns λ_{ij}'s and ρ_{ij}'s.

4.1 Algorithm

The attacking algorithm is summarized as follows: The input is $vk = (\boldsymbol{h}, \boldsymbol{s})$, $\sigma = (\boldsymbol{g}, \boldsymbol{u})$ with $\boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2)$.

1. Compute two syndrome spaces $S_1 = \langle u_{11}, \dots, u_{1n} \rangle_{\mathbb{F}_q}$ and $S_2 = \langle u_{21}, \dots, u_{2n} \rangle_{\mathbb{F}_q}$.
2. Compute $F = \langle f_1, \dots, f_w \rangle$ from $\boldsymbol{g}$.
3. Compute two supports $E_1 := \bigcap_{i=1}^{w} f_i^{-1} S_1$ for $\boldsymbol{x}$ and $E_2 := \bigcap_{i=1}^{w} f_i^{-1} S_2$ for $\boldsymbol{y}$.
4. Compute $\boldsymbol{x}$ and $\boldsymbol{y}$ by by letting $x_i = \sum_j \lambda_{ij} e_{1j}$ and $y_i = \sum_j \rho_{ij} e_{2j}$ and solving the system of nm equations over $\mathbb{F}_q$ corresponding to $\boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y}$ with $2nw$ unknowns λ_{ij}'s and ρ_{ij}'s.

4.2 Experiment

We implemented the attack in the computer algebra system SageMath [Sage18] using the code in appendix A. We generate 100 keys on each parameter sets. On each key, we generate one random signature and try to find a secret key.

We run the SageMath script on Ubuntu 18.04 in WSL on Windows 10. We succeed to obtain all secret keys and the attack used an average CPU time of 2.640–6.928 seconds per a signature, on a single core of a 3.20GHtz Core i7-8700 desktop machine. Table 2 summarizes the experimental results.

Table 2: Parameter Sets for the RQCS scheme: Song et al. defined $w = w_r = w_g$.

Instance	success rate	avg time (sec)
RQCS-1	100/100	2.640
RQCS-2	100/100	5.604
RQCS-3	100/100	6.928

[2] [15] = [ABG+13]

References

ABG$^+$13. Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Low rank parity check codes and their application to cryptography. In Lilya Budaghyan, Tor Helleseth, and Matthew G. Parker, editors, *The Preproceedings of Workshop on Coding and Cryptography (WCC) 2013, Borgen, Norway*, pages 167–179, April 2013. http://www.selmer.uib.no/WCC2013/pdfs/Gaborit.pdf. 3, 4

ABG$^+$18. Nicolas Aragon, Olivier Blazy, Philippe Gaborit, Adrien Hauteville, and Gilles Zémor. Durandal: a rank metric based signature scheme. Cryptology ePrint Archive, Report 2018/1192, 2018. https://eprint.iacr.org/2018/1192. 1, 3

AGHT18. 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 2018, Vail, CO, USA, June 17-22, 2018*, pages 2421–2425. IEEE, 2018. 3

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

AMBD$^+$18. Carlos Aguilar Melchor, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, and Gilles Zémor. Efficient encryption from random quasi-cyclic codes. *IEEE Trans. Information Theory*, 64(5):3927–3943, 2018. 1

BHLP17. Daniel J. Bernstein, Andreas HÃijlsing, Tanja Lange, and Lorenz Panny. Comments on RaCoSS, a submission to NISTâĂŹs PQC competition. 23 Dec. 2017. Available at https://helaas.org/racoss/. 1

DG18. Jean-Christophe Deneuville and Philippe Gaborit Cryptanalysis of a code-based one-time signature. Cryptology ePrint Archive, Report 2018/1205, 2018. https://eprint.iacr.org/2018/1205. 1

FRXKMT17. Kazuhide Fukushima, Partha Sarathi Roy, Rui Xu, Shinsaku Kiyomoto, Kirill Morozov, and Tsuyoshi Takagi. RaCoSS. 21 Dec. 2017. Available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. 1

GZ16. Philippe Gaborit and Gilles Zémor. On the hardness of the decoding and the minimum distance problems for rank codes. *IEEE Trans. Information Theory*, 62(12):7245–7252, 2016. 3

LT19. Terry Shue Chien Lau and Chik How Tan. Key Recovery Attack on Rank Quasi-Cyclic Code-based Signature Scheme. CoRR, abs/1902.00241, 2019 https://arxiv.org/abs/1902.00241. 1

Lyu08. Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 162–179. Springer, Heidelberg, March 2008. 1

Per12. Edoardo Persichetti. *Improving the Efficiency of Code-Based Cryptography*. PhD thesis, University of Auckland, 2012. Available at https://researchspace.auckland.ac.nz/handle/2292/19803 1

Per18. Edoardo Persichetti. Efficient One-Time Signatures from Quasi-Cyclic Codes: A Full Treatment. *Cryptography*, 2(4):30, 2018. 1

Rot06. Ron Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006. 2

RMFKT18. . Partha Sarathi Roy, Kirill Morozov, Kazuhide Fukushima, Shinsaku Kiyomoto, and Tsuyoshi Takagi. Code-Based Signature Scheme without Trapdoors. IEICE Tech. Rep., vol. 118, no. 151, ISEC2018-15, pp. 17âĂŞ22, July 2018. See also https://www.ieice.org/ken/paper/20180725L1FF/eng/. 1

Sage18. The Sage Developers. Sagemath, the sage mathematics software system (version 8.3), 2018. http://www.sagemath.org. 4

SBC18. Paolo Santini, Marco Baldi, Franco Chiaraluce Cryptanalysis of a One-Time Code-Based Digital Signature Scheme. CoRR, abs/1812.03286, 2018. https://arxiv.org/abs/1812.03286. 1

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990. 1

SHMW19. Yongcheng Song, Xinyi Huang, Yi Mu, and Wei Wu. A new code-based signature scheme with shorter public key. Cryptology ePrint Archive, Report 2019/053, 2019. https://eprint.iacr.org/2019/053. 1, 3

Xag18. Keita Xagawa. Practical Attack on RaCoSS-R. Cryptology ePrint Archive, Report 2018/831, 2018. https://eprint.iacr.org/2018/831. 1

A Implementation

Listing 1.1: attack.sage

```
# Breaking RQCS \url{https://ia.cr/2019/053}
import sys

if len(sys.argv) != 2:
    print("Usage: %s <n>" % sys.argv[0])
    print("Run the attack script against RQCS")
    sys.exit(1)
```

```
if int(sys.argv[1]) == 1:
    (q,n,m,w,ww,delta) = (2, 67, 89,5,30,31) #RQCS-1
elif int(sys.argv[1]) == 2:
    (q,n,m,w,ww,delta) = (2, 97,121,6,42,43) #RQCS-2
elif int(sys.argv[1]) == 3:
    (q,n,m,w,ww,delta) = (2,101,139,6,42,48) #RQCS-3
else:
    print("%s should be 1/2/3 <n>" % sys.argv[1])
    sys.exit(1)

print "===== Start ====="
print [q,n,m,w,ww,delta]

F = GF(q^m,'a')
P.<XX> = F[]
R.<X> = QuotientRing(P, P.ideal(XX^n-1))

# Utilities
def random_F_vec_generation(F,n,t):
    B = matrix(F.base_ring(),t,m,0)
    while B.rank() != t:
        B = matrix(F.base_ring(),[vector(F.random_element()) for i in range(t)])
    C = matrix(F.base_ring(),n,m,0)
    while C.rank() != t:
        C = matrix(F.base_ring(),n,t,[F.base_ring().random_element() for _ in range(n*t)]) * B
    return vector(F,[C[i] for i in range(n)])

def F_vec_to_R_elem(v):
    z = R(0)
    for i in range(n):
        z += v[i]*X^i
    return z

def random_R_element_generation(t):
    return F_vec_to_R_elem(random_F_vec_generation(F,n,t))
def R_to_Matrix(z):
    return matrix(F.base_ring(),n,m,[vector(z[i]) for i in range(n)])
def rank_R(z):
    return matrix(F.base_ring(),n,m,[vector(z[i]) for i in range(n)]).rank()
def basis_to_F_list(B):
    return [F(x) for x in B.rows()]

# key generation and sign. We omit the verification algorithm
def sk_gen():
    return random_R_element_generation(w), random_R_element_generation(w)
def pk_gen(x,y):
    h = R.random_element()
    s = x+h*y
    return h,s
def sign(h,s,x,y):
    r1 = random_R_element_generation(w)
    r2 = random_R_element_generation(w)
    III  = r1 + h * r2
    g  = random_R_element_generation(w)
    u1 = x*g+r1
    u2 = y*g+r2
    return III, g, u1, u2, r1, r2
    # we output r1 and r2 for debugging

# compute Si = (1/Fi)*S for Fi in F0/F1 and take intersection
```

```
def find_Ei(S_Flist, Fi_Flist):
    tmp = matrix(F.base_ring(),[vector(F.gen()**i) for i in range(m)]).row_space()
    for Fi in Fi_Flist:
        FSi_list = [(1/Fi)*i for i in S_Flist]
        SSBi = matrix(F.base_ring(),[vector(i) for i in FSi_list])
        tmp = tmp.intersection(SSBi.row_space())
    return tmp

def make_A(H1,H2,E1,E2):
    A_list = []
    # For x
    for i in range(n):
        for k in range(w):
            for j in range(n):
                A_list += list(vector(H1[i,j]*E1[k]))
    # For y
    for i in range(n):
        for k in range(w):
            for j in range(n):
                A_list += list(vector(H2[i,j]*E2[k]))
    return matrix(F.base_ring(),2*n*w,n*m,A_list)

def recover_sk(h,s,g,u1,u2):
    # we have u1 = g*x+r1 and u2 = g*y+r2
    # S1 is generated by u1 and S2 is generated by u2
    # E1 is generated by x and E2 is generated by y
    # F is generated by g
    # We have S1 = E1.F + R1 and S2 = E2.F + R2.
    # Thus, we take Eb = \bigcap fi^{-1} Sb for b = 1 and 2
    # We then solve the system of eqs, x + hy = s.

    # step 1: compute S1 = Supp(u1) and S2 = Supp(u2)
    S1_basis = matrix(R_to_Matrix(u1).row_space().basis())
    S1_Flist = basis_to_F_list(S1_basis)
    S2_basis = matrix(R_to_Matrix(u2).row_space().basis())
    S2_Flist = basis_to_F_list(S2_basis)

    # step 2: compute F
    F_basis = matrix(R_to_Matrix(g).row_space().basis())
    F_Flist = basis_to_F_list(F_basis)

    # step 3: find E1 = Supp(x) and E2 = Supp(y)
    E1c_basis = matrix(find_Ei(S1_Flist,F_Flist).basis())
    E1c_Flist = basis_to_F_list(E1c_basis)
    E2c_basis = matrix(find_Ei(S2_Flist,F_Flist).basis())
    E2c_Flist = basis_to_F_list(E2c_basis)

    # step 4: recover x and y
    # step 4-1: find x_ij and yij
    S = vector(F,s)
    Sp_list = []
    for j in range(n):
        Sp_list += list(vector(S[j]))
    Sp = vector(F.base_ring(),Sp_list)

    H1 = R(1).matrix()
    H2 = h.matrix()
    A = make_A(H1,H2,E1c_Flist,E2c_Flist)
    V = A.solve_left(Sp)

    # step 4-2: compute x and y from xij and yij
    Vx = matrix(F,n,w,V[0:n*w])
    Vy = matrix(F,n,w,V[n*w:2*n*w])
    Zx = Vx * (vector(F,E1c_Flist))
    Zy = Vy * (vector(F,E2c_Flist))
    Xc = F_vec_to_R_elem(Zx)
    Yc = F_vec_to_R_elem(Zy)

    return Xc, Yc

def test(pairs=10,debug=true):
    succ = 0
    tottime = 0.0
    for npair in range(pairs):
        x,y = sk_gen()
        h,s = pk_gen(x,y)
        print "----- Key pair %d -----" % (npair)

        II, g, u1, u2, r1, r2 = sign(h,s,x,y)

        tm = cputime(subprocesses = True)
        try:
            Xc, Yc = recover_sk(h,s,g,u1,u2)
            if debug:
                print "rank(Xc), rank(Yc), Xc == x, Yc == y, Xc+h*Yc"
                print rank_R(Xc), rank_R(Yc), Xc == x, Yc == y, s == Xc+h*Yc
            if rank_R(Xc) == w and rank_R(Yc) == w and s == Xc+h*Yc:
                succ += 1
        except:
            print "Unexpected error", sys.exc_info()[0]
        zz = float(cputime(tm))
        tottime += zz
        if debug:
            print zz

    print "===== Results ====="
    print [q,n,m,w,ww,delta]
    print "Total time for extraction : %f seconds ." % ( tottime )
    print "Average time for extraction : %f seconds ." % ( tottime / pairs )
    print "Successful recoveries : %d/%d" % (succ,pairs)

test(100,True)
```