Practical Attack on RaCoSS-R

Keita Xagawa

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

Abstract. RaCoSS is a signature scheme based on the syndrome decoding problem over the random linear code and proposed by Fukushima, Roy, Xu, Kiyomoto, Morozov, and Takagi. This scheme is cryptanalyzed Bernstein, Hülsing, Lange, and Panny (pqc-forum on 23 Dec. 2017).
Roy, Morozov, Fukushima, Kiyomoto, and Takagi recently gave a patch and call the patched scheme as RaCoSS-R (ISEC Conf. on 25 Jul. 2018). This short note describes how to break RaCoSS-R by modifying the forgery attack against RaCoSS.
keywords: NIST PQC, post-quantum digital signatures, cryptanalysis, coding-based cryptography

1 Introduction

RaCoSS[1] is a signature scheme based on the syndrome decoding problem over a random linear code [FRXKMT17]. It is one of NIST PQC standardization candidates in Round 1 on 21 Dec. 2017. On 23 Dec. 2017, Bernstein, Hülsing, Lange, and Panny rapidly cryptanalyzed RaCoSS and reported implementation bug, weakness of the dedicated hash function, mathematical breaking, and no-hope [BHLP17].

Recently, Roy, Morozov, Fukushima, Kiyomoto, and Takagi recently gave a patch and call the patched scheme as RaCoSS-R [RMFKT18]. They changed several parameters of RaCoSS in order to prevent the attacks of Bernstein et al. [BHLP17].

This short note describes a mathematical attack on RaCoSS-R. Our attack is a variant of the mathematical breaking of RaCoSS by Bernstein et al. [BHLP17].

2 Review of RaCoSS and the BHLP Attack

Preliminaries: Let $\mathbb{F} := \mathrm{GF}(2)$. Ber_τ denotes the Bernoulli distribution with parameter τ, that is,

$$\Pr_{x \leftarrow \mathrm{Ber}_\tau}[x = 1] = \tau \text{ and } \Pr_{x \leftarrow \mathrm{Ber}_\tau}[x = 0] = 1 - \tau.$$

For two positive integers n, m, $\mathrm{Ber}_\tau^{n \times m}$ denotes the distribution of matrices in $\mathbb{F}^{n \times m}$ where each entry of the matrix is sampled independently from Ber_τ.

2.1 RaCoSS

RaCoSS employed several parameters defined as follows: $n = 2400$, $k = 2060$, $n - k = 340$, $\omega = 48$, and $\gamma = 0.07$. We have $\rho \approx 0.057$, $\mathrm{th} \approx 1548.03$, and $w = 3$.[2] Their hash function Hash is modeled as a random oracle that on input a string outputs an n-bit string of Hamming weight w uniformly at random.

The signature scheme is described as follows:

- Setup(): output $\mathsf{p} := H \leftarrow \mathbb{F}^{(n-k) \times n}$.
- Gen(p): sample $S \leftarrow \mathrm{Ber}_{\omega/n}^{n \times n}$ and compute $T := HS \in \mathbb{F}^{(n-k) \times n}$.
  Output $sk = S$ and $vk = T$.

[1] an acronym of Random Code-based Signature Scheme.

[2] The proposer defined

$$\rho := \left\lfloor \frac{1}{2}\left(1 - \left(1 - \frac{2\omega}{n}\right)^{\lfloor \gamma\omega \rfloor}\right) \cdot 1000 \right\rfloor / 1000$$

and $\mathrm{th} := 12n\rho(1 - \rho)$. They also define $w := \lfloor \gamma\omega \rfloor$.

– Sign(p, sk, μ):
  1. $y \leftarrow \mathsf{Ber}^n_{\omega/n}$ and compute $v := Hy$
  2. $c := \mathsf{Hash}(v, \mu, H)$.
  3. compute $z = Sc + y$
  4. output $\sigma = (z, c)$
– Vrfy(p, vk, μ, σ):
  1. if $\mathsf{Hash}(Hz + Tc, \mu, H) \neq c$, then return $\perp$
  2. if $\mathsf{wt}(z) \geq \mathsf{th}$, then return $\perp$
  3. else, return $\top$

Notice that $Hz + Tc = H(Sc + y) + Tc = HSc + Hy + Tc = Tc + Hy + Tc = Hy$. In addition, the expected value of z' weight is $w \cdot n \cdot (\omega/n) + n \cdot (\omega/n) = (w+1)\omega = 48 \cdot 4 = 192$, much smaller than $\mathsf{th} \approx 1548.03$. Therefore, the signature scheme is (statistically) correct.

2.2 The BHLP Attack

We review "math" attack in Lange's mail [BHLP17].

Assume that $H = [H_1 \mid H_2]$ with $H_1 \in \mathrm{GL}(n-k, \mathbb{F})$. For any message μ,

1. choose arbitrary $y \in \mathbb{F}^n$ and compute $v = Hy$
2. compute $c := \mathsf{Hash}(v, \mu, H)$
3. compute $z_1 := H_1^{-1} \cdot (Hy + Tc)$
4. fill $z := (z_1, 0, \ldots, 0) \in \mathbb{F}^n$
5. output (z, c) as a forgery.

Apparently, $\mathsf{wt}(z) \leq n - k = 340 < \mathsf{th}$. By the definition of z, we have

$$Hz + Tc = [H_1 \mid H_2] \cdot \begin{pmatrix} z_1 \\ 0 \end{pmatrix} + Tc = H_1 z_1 + Tc = (Hy + Tc) + Tc = v$$

and $\mathsf{Hash}(Hz + Tc, \mu, H) = c$ as we wanted. Thus, (z, c) is a valid signature on μ.

3 Review of RaCoSS-R

In order to prevent the BHLP attack, Roy et al. changed several parameter values as follows: Let $n = 3072$, $k = 1536$, $\omega = 215$, and $\beta = 1386$. Their hash function $\mathsf{Hash}\colon \{0,1\}^* \to \{0,1\}^n$ is modeled as a random oracle. The modified scheme is described as follows:

– Setup(): output $\mathsf{p} := H \leftarrow \mathbb{F}^{(n-k)\times n}$.
– Gen(p): sample $S \leftarrow \mathsf{Ber}^{n\times n}_{\omega/n}$ and compute $T := HS \in \mathbb{F}^{(n-k)\times n}$. Output $sk = S$ and $vk = T$.
– Sign(p, sk, μ):
  1. $y \leftarrow \mathsf{Ber}^n_{\omega/n}$ and compute $v := H \cdot y$
  2. $c := \mathsf{Hash}(v, \mu, H)$
  3. compute $z = Sc + y$
  4. if $\mathsf{wt}(z) < \beta$ or $\mathsf{wt}(z) \geq n/2$, then go to step 1
  5. output $\sigma = (z, c)$
– Vrfy(p, vk, μ, σ):
  1. if $\mathsf{Hash}(Hy + Tc, \mu, H) \neq c$, then return $\perp$
  2. if $\mathsf{wt}(z) < \beta$ or $\mathsf{wt}(z) \geq n/2$, then return $\perp$
  3. else, return $\top$

Roy et al. discussed that their changes prevent the BHLP attack in [RMFKT18, Sect. 4.3].

4 Our Attack on RaCoSS-R

In order to mount a forgery attack, we are required to compute z whose weight is in between β and $n/2$. We observe that the weight of z_1 is roughly $(n-k)/2$ in the BHLP attack and we fill z_1 with a random vector z_2 instead of 0.

Let us assume that $H = [H_1 \mid H_2]$ with $H_1 \in \mathrm{GL}(n-k, \mathbb{F})$ and introduce a parameter l that will control the weight of z_2. The attack follows:

1. choose arbitrary $y \in \mathbb{F}^n$ and compute $v = Hy$
2. compute $c := \mathsf{Hash}(v, \mu, H)$
3. set $z_2 := (\overbrace{0, \ldots, 0}^{k-l}, \overbrace{1, \ldots, 1}^{l}) \in \mathbb{F}^k$
4. compute $z_1 := H_1^{-1} \cdot (Hy + Tc + H_2 z_2)$
5. set $z := (z_1, z_2) \in \mathbb{F}^n$
6. if $\mathrm{wt}(z) < \beta$ or $\mathrm{wt}(z) \geq n/2$, then go to step 1
7. output (z, c) as a forgery

This z satisfies

$$Hz + Tc = H_1 z_1 + H_2 z_2 + Tc = (Hy + Tc + H_2 z_2) + H_2 z_2 + Tc = Hy$$

as we wanted.

Experiment: We implemented the attack in the computer algebra system SageMath [Sage18] using the code in section A. We set $l = 700$ in our experiment, because $\mathrm{wt}(z_1)$ falls in the range $[730, 800]$ in our preliminary experiment.

We then ran the attack on 10 keys. On each key, we generate random 10 messages and try to forge. In our experiment, we succeed to forge on all messages and keys and the attack took an average CPU time of 11 seconds per key and 5.4 seconds per message on a single core of a 2.3 GHz Intel Xeon server machine.

Acknowledgment

The author would like to thank Partha Sarathi Roy and Kazuhide Fukushima for their kindness and fruitful discussions.

References

BHLP17. Daniel J. Bernstein, Andreas Hülsing, Tanja Lange, and Lorenz Panny. Comments on RaCoSS, a submission to NIST's PQC competition. 23 Dec. 2017. Available at https://helaas.org/racoss/. 1, 2

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

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. 17–22, July 2018. See also https://www.ieice.org/ken/paper/20180725L1FF/eng/. 1, 2

Sage18. The Sage Developers. SageMath, the Sage Mathematics Software System (Version 8.3), 2018. http://www.sagemath.org. 3

A Implementation

Listing 1.1: attack.sage

```
# forgery attack against RaCoSS-R

import hashlib
```

```
n = 3072; k = int(n/2); w = 215; b = 1386
F = GF(2)
def Bernoulli(w,n): return 1 if randint(0,n) < w else 0

# This hash funciton returns n-dim vector such that
# ded_hash(dat,n) = sha512(0,dat) || sha512(1,dat) || ... || sha512(n/512,dat)
# We assume 512 | n.
def ded_hash(dat,n):
    h = hashlib.sha512()
    t = ""
    for i in range(int(n/512)):
        h.update((str(i)+dat).encode())
        t += h.hexdigest()
    z = ('{:b}'.format(int(t,16))).zfill(n)
    v = vector(F,map(int,list(z)))
    return v

def ded_str(H,M):
    h = hashlib.sha512()
    h.update(str(H))
    pkdigest = h.hexdigest()
    ded_digest = M+pkdigest
    return ded_digest

# We ignore a transpose op.
def pargen(): return Matrix(F,n-k,n,lambda i,j: randint(0,2))
def skgen(): return Matrix(F,n,n,lambda i,j: Bernoulli(w,n))
def pkgen(H,S): return H*S

def inner_sign(S,H,ded_digest):
    y = vector(F,[Bernoulli(w,n) for _ in range(n)])
    c = ded_hash(str(H*y)+ded_digest,n)
    z = S*c+y
    return z, c

def sign(S,T,H,M):
    ded_digest = ded_str(H,M)
    z = vector(F,[0 for _ in range(n)])
    while z.hamming_weight() < b or n/2 <= z.hamming_weight():
        z, c = inner_sign(S,H,ded_digest)
    return z, c

def vrfy(H,T,M,z,c):
    ded_digest = ded_str(H,M)
    ctilde = ded_hash(str(H*z+T*c)+ded_digest,n)
    if c != ctilde:
        return False
    elif z.hamming_weight() < b:
        return False
    elif z.hamming_weight() >= n/2:
        return False
    else:
        return True

##########
# forgery attack against RaCoSS-R
##########


# return P, H1, H2 s.t. H * P = [H1 | H2] with invertible H1
# we assume that H.rank() = n-k.
```

```
def find_PH(H):
    if H.rank() != n-k:
        raise ValueError("error! H.rank() != n-k")
    L = list(H.pivots())
    P = identity_matrix(n)
    for i,v in enumerate(L):
        P.swap_columns(i,v)
    H1 = (H*P)[:,0:n-k]
    H2 = (H*P)[:,n-k:n]
    H1inv = H1.inverse()
    return P, H1, H2, H1inv


def attack(H,T,P,H1,H2,H1inv,M):
    # this l is hueristic
    l = 700
    ded_digest = ded_str(H,M)
    z = vector(F,[0 for _ in range(n)])
    t = 0
    while z.hamming_weight() < b or n/2 <= z.hamming_weight():
        y = vector(F,[randint(0,2) for _ in range(n)])
        c = ded_hash(str(H*y)+ded_digest,n)
        z21 = vector(F,[0 for _ in range(k-l)])
        z22 = vector(F,[1 for _ in range(l)])
        z2 = vector(F,list(z21)+list(z22))
        z1 = H1inv * (H*y + T*c + H2*z2)
        # print "wt(z1)=", z1.hamming_weight()
        z = P * vector(F,list(z1)+list(z2))
        t += 1
    return z, c, t

def attack_test(keys = 10, messages = 10, debug = false):
    tot_time_ext = 0.0
    tot_time_forge = 0.0
    tot_keys = 0
    tot_tries = 0.0
    nkey = 0
    tm = cputime(subprocesses=True)

    print "----- Start -----"
    while tot_keys <= keys:
        print "----- Key pair %d -----" % (nkey)
        H = pargen()
        S = skgen()
        T = pkgen(H,S)
        nkey += 1
        if H.rank() != n-k:
            break
        else:
            tot_keys += 1
        print "attack"
        tm = cputime(subprocesses=True)
        P,H1,H2,H1inv = find_PH(H)
        tot_time_ext += float(cputime(tm))

        for message in range(messages):
            M = "RaCoSS-R is broken" + str(randint(0,2**10))
            tm = cputime(subprocesses=True)
            z,c,t = attack(H,T,P,H1,H2,H1inv,M)
            tot_time_forge += float(cputime(tm))
            tot_tries += t
```

```
            if debug:
                print "M = ", M
                print "t = ", t
                print "vrfy(p,M,(z,c)) = ", vrfy(H,T,M,z,c)

    print "===== Results ====="
    print "Total time for extraction: %f seconds." % (tot_time_ext)
    print "Average time for extraction: %f seconds." % (tot_time_ext/keys)
    print "----- Forgery -----"
    print "Total time: %f seconds." % (tot_time_forge)
    print "Average time: %f seconds." % (tot_time_forge/messages/keys)
    print "Average tries: %f tries." % (tot_tries/messages/keys)

attack_test(10,10,False)
```