From: Lorenz Panny <l.s.panny@tue.nl>
Sent: Saturday, December 23, 2017 9:11 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: RaCoSS

Dear designers, dear all,

In looking for a signature forgery we noticed that in some cases, almost any message passes as valid for a given signature when using the code in Reference_Implementation. This is because of a confusion between bits and bytes, thus only 1/8th of the entries of c get compared:

```
  for( i=0 ; i<(RACOSS_N/8) ; i++ )
    /* compare ith bit and fail on mismatch */
```

Since c has very low weight, this succeeds with high probability:
The signature of the first KAT "signs" about 67% of all messages because c starts with 300 zero bits. If c has {1, 2, 3} of the first 300 bits set, the probability that a random message is accepted for a given signature drops to about {2^-10, 2^-20, 2^-31}.

Moreover, we also noticed memory leaks in wrhf() and crypto_sign_open(); in both cases the arrays obtained via malloc() are not free()d.

Regards
    Andreas, Dan, Lorenz, and Tanja

Andreas Huelsing, Daniel J. Bernstein, Lorenz Panny, Tanja Lange

From: A. Huelsing <ietf@huelsing.net>
Sent: Saturday, December 23, 2017 9:16 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov; authorcontact-racoss@box.cr.yp.to
Subject: OFFICIAL COMMENT: RaCoSS

Dear designers, dear all,

The low-weight hash function used in RaCoSS is not secure. Please see below for a message colliding with the first KAT message when hashed together with the data used in RaCoSS, hence the signature of the first KAT is also a signature for this second-preimage. This does not use the implementation flaw we observed before. It exploits the fact that the size of the image of the wrhf hash function is small, thus (second-)preimages can easily be found by brute force.

The message we found is (without quotes):

"NISTPQC is so much fun! 10900qmmP"

To check this, replace the original message of the first KAT by this message and leave the rest of the signed message (sm) unchanged; it will still verify for the same public key.

The hash function wrhf takes as input a message m, the desired weight w, and a length n and outputs a string c of length n having exactly w non-zero entries. The internals of wrhf do not matter, only the size of the range of wrhf is relevant. For the proposed parameters n=2400,
w=3 there are only (2400 choose 3) = 2301120800 ~ 2^31.09 possible outputs.

This means that a (second-)preimage can be found in roughly 2^31 runs of wrhf, allowing for a forgery under random message attacks. The hash function is relatively slow, but the above preimage attack was done overnight. It is also possible to select a particular message by varying other parts of the hash input.

Chosen message attacks are even faster. A collision can be found in
2^15.5 runs of wrhf.

Regards
Andreas, Dan, Lorenz, and Tanja

Andreas Huelsing, Daniel J. Bernstein, Lorenz Panny, Tanja Lange

From: Tanja Lange <tanja@hyperelliptic.org>
Sent: Saturday, December 23, 2017 9:16 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: RaCoSS

Dear designers, dear all,

We have identified a serious flaw in the design of RaCoSS and can quickly sign any message for any public key with the specified RaCoSS parameters, without knowing the secret key.

Here is the setup.

The system uses a public, shared matrix H which is a 340x2400 low-weight binary matrix.

The secret key is a low-weight 2400x2400 matrix S^t.

The public key is H*S^t.

The signer picks a random, low-weight 2400-bit vector y; computes v = H * y; computes a hash function on v, m, and H to get a low-weight 2400-bit vector c; and computes z = S^t * c + y.

The signature on m is the pair (z,c), where z has 2400 bits and c has 2400 bits and only very few non-zero entries.

The verifier checks that z has not too large weight. If so the verifier computes v1 = H * z, v2 = T * c, v = v1 + v2 and verifies that c matches the output of the hash function on v, m, and H.

This works for a valid message because

v2 = T * c = H * S^t * c and

v1 = H * z = H * (S^t * c + y) = H * S^t * c + H * y = v2 + v.

Here is the attack.

The attacker takes a subset of 340 linearly independent columns of H.
Note that any set has about a 29% chance of being invertible, and there are (2400 choose 340) subsets to choose from.
For ease of exposition assume that H = (H1 | H2) with H1 an invertible 340x340 matrix; this is true for the particular matrix H used in the RaCoSS reference software.

To forge a signature on a message m, follow the steps as above up to the computation of c. Then compute

z1 = H1^(-1) * (H * y + T * c),

a 340 bit vector, and put z = (z1|00...0) to fill up to 2400 bits.

This vector z passes all the computational checks.
Each entry of z1 is 1 with probability 1/2, so verification of the weight restriction works if 170 errors are allowed. If the bound is more restrictive, the attacker can try with other splits of H.
However, for functionality reasons the bound cannot be much smaller than this, because a properly generated vector z is likely to have weight at least this large.

Note: Computing H1^(-1) is a one-time effort that works for _all_ public keys. The costs for the forgery are essentially the same as the costs for a regular signature; the only additional operation is the multiplication by the 340x340 matrix H1^(-1).

See https://helaas.org/racoss-20171222.tar.gz for an implementation of the attack.

Regards
Andreas, Dan, Lorenz, and Tanja

Andreas Huelsing, Daniel J. Bernstein, Lorenz Panny, Tanja Lange

From: D. J. Bernstein <djb@cr.yp.to>
Sent: Saturday, December 23, 2017 9:26 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: RaCoSS

Dear designers, dear all,

The RaCoSS design specifies an upper bound for the weight of z. The designers want to ensure that all properly generated signatures are valid. They chose to use the Chernoff bound to compute a bound on the weight of z. We point out that the Chernoff bound is a very weak bound:
for the parameters suggested here the 2400-bit vector z is accepted if it has weight no larger than 1564, which means that more than half of the bits of z are allowed to be 1. For comparison, the z in the first KAT has weight 275. This illustrates that a better analysis would lead to better bounds while keeping a negligible probability of failure.

The designers could further tighten the bound and push a check whether (z,c) verifies into the signature generation. The algorithm could iterate over different choices of y to find one satisfying the bound.

For the given parameters this will not stop the attack laid out in our previous message. Any reasonable choice of bound that will not require too many iterations of choosing y will allow the attacker's z to pass and the attacker similarly has the freedom to change y and on top of that to change the subset of columns in H1.

The bounds will become worse with larger weights in c, but those are necessary to avoid preimage attacks on the hash functions.

Maybe there are some very large choices of n and k for which one can achieve a separation of the weight of the attacker's z and the legitimately computed values, but we see no chance for the given n and k.

Regards
Andreas, Dan, Lorenz, and Tanja

Andreas Huelsing, Daniel J. Bernstein, Lorenz Panny, Tanja Lange

From: Kazuhide Fukushima <ka-fukushima@kddi-research.jp>
Sent: Monday, January 08, 2018 2:09 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov; Shinsaku Kiyomoto; Partha Sarathi Roy
Subject: OFFICIAL COMMENT: RaCoSS

Dear Dr. Andreas Huelsing, Prof. Daniel Julius Bernstein, Dr. Lorenz Panny and Prof. Tanja Lange:

First of all, we appreciate your thorough analysis of our proposal: RaCoSS.

Overall, we believe that the main problem is our choice of parameters as you pointed out in Comment (4); specifically, a too low weight of the vector "c."
We are planning to address it by providing a different set of recommended parameters with increased "n."
Since our design is based on the Fiat-Shamir transformation with abort, we do not expect forgery attacks, once the appropriate parameter settings will be found.
We are now in the process of re-evaluating the security of our scheme, and we will send you the updated document, once this process will have been completed and the necessary and sufficient parameter setting is found.

Let us now specifically address your comments below.

Comment (1):
In looking for a signature forgery we noticed that in some cases, almost any message passes as valid for a given signature when using the code in Reference_Implementation. This is because of a confusion between bits and bytes, thus only 1/8th of the entries of c get compared:

```
  for( i=0 ; i<(RACOSS_N/8) ; i++ )
    /* compare ith bit and fail on mismatch */
```

Since c has very low weight, this succeeds with high probability:
The signature of the first KAT "signs" about 67% of all messages because c starts with 300 zero bits. If c has {1, 2, 3} of the first 300 bits set, the probability that a random message is accepted for a given signature drops to about {2^-10, 2^-20, 2^-31}.

Moreover, we also noticed memory leaks in wrhf() and crypto_sign_open(); in both cases the arrays obtained via malloc() are not free()d.

Answer:
This security issue is due to a programing bug. We can deal with the issue by replacing "for(i=0 ; i<(RACOSS_N/8) ; i++)" with "for(i=0 ; i<(RACOSS_N) ; i++)."

Furthermore, we will add calls to free() in wrhf() and cryopt_sign_open().

Comment (2):
The low-weight hash function used in RaCoSS is not secure. Please see below for a message colliding with the first KAT message when hashed together with the data used in RaCoSS, hence the signature of the first KAT is also a signature for this second-preimage. This does not use the implementation flaw we observed before. It exploits the fact that the size of the image of the wrhf hash function is small, thus (second-)preimages can easily be found by brute force.

The message we found is (without quotes):

"NISTPQC is so much fun! 10900qmmP"

To check this, replace the original message of the first KAT by this message and leave the rest of the signed message (sm) unchanged; it will still verify for the same public key.

The hash function wrhf takes as input a message m, the desired weight w, and a length n and outputs a string c of length n having exactly w non-zero entries. The internals of wrhf do not matter, only the size of the range of wrhf is relevant. For the proposed parameters n=2400,
w=3 there are only (2400 choose 3) = 2301120800 ~ 2^31.09 possible outputs.

This means that a (second-)preimage can be found in roughly 2^31 runs of wrhf, allowing for a forgery under random message attacks. The hash function is relatively slow, but the above preimage attack was done overnight. It is also possible to select a particular message by varying other parts of the hash input.

Chosen message attacks are even faster. A collision can be found in
2^15.5 runs of wrhf.

Answer:
This security issue is related to the security of hash function.
The attack is due to the low weight image space. More concretely, it is related to the low weight property of "c."
Thus, we need to increase the security parameter "n" to increase the hamming weight of "c."
The attack is not related to the structure of the hash function.
Thus, changing the hash function will not work to solve the security issue.

Comment (3):
We have identified a serious flaw in the design of RaCoSS and can quickly sign any message for any public key with the specified RaCoSS parameters, without knowing the secret key.

Here is the setup.

The system uses a public, shared matrix H which is a 340x2400 low-weight binary matrix.

The secret key is a low-weight 2400x2400 matrix S^t.

The public key is H*S^t.

The signer picks a random, low-weight 2400-bit vector y; computes v = H * y; computes a hash function on v, m, and H to get a low-weight 2400-bit vector c; and computes z = S^t * c + y.

The signature on m is the pair (z,c), where z has 2400 bits and c has 2400 bits and only very few non-zero entries.

The verifier checks that z has not too large weight. If so the verifier computes v1 = H * z, v2 = T * c, v = v1 + v2 and verifies that c matches the output of the hash function on v, m, and H.

This works for a valid message because

v2 = T * c = H * S^t * c and

v1 = H * z = H * (S^t * c + y) = H * S^t * c + H * y = v2 + v.

Here is the attack.

The attacker takes a subset of 340 linearly independent columns of H.
Note that any set has about a 29% chance of being invertible, and there are (2400 choose 340) subsets to choose from.
For ease of exposition assume that H = (H1 | H2) with H1 an invertible 340x340 matrix; this is true for the particular matrix H used in the RaCoSS reference software.

To forge a signature on a message m, follow the steps as above up to the computation of c. Then compute

z1 = H1^(-1) * (H * y + T * c),

a 340 bit vector, and put z = (z1|00...0) to fill up to 2400 bits.

This vector z passes all the computational checks.
Each entry of z1 is 1 with probability 1/2, so verification of the weight restriction works if 170 errors are allowed. If the bound is more restrictive, the attacker can try with other splits of H.
However, for functionality reasons the bound cannot be much smaller than this, because a properly generated vector z is likely to have weight at least this large.

Note: Computing H1^(-1) is a one-time effort that works for _all_ public keys. The costs for the forgery are essentially the same as the costs for a regular signature; the only additional operation is the multiplication by the 340x340 matrix H1^(-1).

See https://helaas.org/racoss-20171222.tar.gz for an implementation of the attack.

Answer:
This security issue is due to the low randomness of the signature string.
We need to increase the security parameter "n" to increase the hamming weight of "c."
Our design of signature scheme follows Fiat-Shamir transformation with abort.
So, there should not have any forgery attack for appropriate parameter settings.
It is possible only when the signature string has less randomness, and our scheme with specified parameters has this problem.
The proposed attack utilizes the huge number of zero bits of "c,"
which propagate in the expression of "z."

Comment (4):
The RaCoSS design specifies an upper bound for the weight of z. The designers want to ensure that all properly generated signatures are valid. They chose to use the Chernoff bound to compute a bound on the weight of z. We point out that the Chernoff bound is a very weak bound:
for the parameters suggested here the 2400-bit vector z is accepted if it has weight no larger than 1564, which means that more than half of the bits of z are allowed to be 1. For comparison, the z in the first KAT has weight 275. This illustrates that a better analysis would lead to better bounds while keeping a negligible probability of failure.

The designers could further tighten the bound and push a check whether (z,c) verifies into the signature generation. The algorithm could iterate over different choices of y to find one satisfying the bound.

For the given parameters this will not stop the attack laid out in our previous message. Any reasonable choice of bound that will not require too many iterations of choosing y will allow the attacker's z to pass and the attacker similarly has the freedom to change y and on top of that to change the subset of columns in H1.

The bounds will become worse with larger weights in c, but those are necessary to avoid preimage attacks on the hash functions.

Maybe there are some very large choices of n and k for which one can achieve a separation of the weight of the attacker's z and the legitimately computed values, but we see no chance for the given n and k.

Answer:
This security issue is due to the low hamming weight of "c."
We need to increase the security parameter "n" to increase the hamming weight of "c."

Best regards,
Partha Sarathi Roy, Kazuhide Fukushima, Shinsaku Kiyomoto, Kirill Morozov and Tsuyoshi Takagi

From: Tanja Lange <tanja@hyperelliptic.org>
Sent: Wednesday, March 07, 2018 10:49 AM
To: Kazuhide Fukushima
Cc: pqc-comments; pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: RaCoSS

Dear Dr. Fukushima, dear all,
Coming back to your message from earlier this year.

> Overall, we believe that the main problem is our choice of parameters
> as you pointed out in Comment (4); specifically, a too low weight of
> the vector "c."
> We are planning to address it by providing a different set of
> recommended parameters with increased "n."
>
I agree that low weight and small n are reasons for the break, but my assessment is that parameters would need to be very large if you want to create a gap between the weight an attacker can create and the weight that a legitimate user creates following the protocol.

> We are now in the process of re-evaluating the security of our scheme,
> and we will send you the updated document, once this process will have
> been completed and the necessary and sufficient parameter setting is found.
>
We would be happy to give feedback if you have updated parameters. I was hoping to discuss this at the NISTPQC workshop but I do not see a RaCoSS talk on the schedule.

All the best
Tanja