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://na01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fhelaas.org%2Fracoss-20171222.tar.gz&data=02%7C01%7Csara.kerman%40nist.gov%7C5b85c3b0497d4581676808d54a109731%7C2ab5d82fd8fa4797a93e054655c61dec%7C1%7C0%7C636496357485442983&sdata=iK%2FIo5ROnTNm8HJfyOBlaMMvknUtC6oppaIDNVGVcnE%3D&reserved=0 for an implementation of the attack.

Regards
Andreas, Dan, Lorenz, and Tanja

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