
From: Alperin-Sheriff, Jacob (Fed)
Sent: Thursday, December 28, 2017 5:17 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov; jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; mypurist@gmail.com; Yongwoo Lee
Subject: OFFICIAL COMMENT: pqsigRM

So I've been doing some basic testing via adding cpucycles calls to the PQCgenKAT_*.c files, just to see if the numbers I get are somewhere in the ballpark of what was in specifications (note that I've indeed gotten numbers somewhere in the ballpark for several so it doesn't seem to be an issue with my code).

For pqsigrm (specifically pqsigRM-4-12, I haven't checked the others), I am getting on the order of thousands of times as many cycles for each of key generation, signing and verification as what the pqsigRM team gave in their supporting documentation.

It's quite possible that the submitters meant thousands of cycles instead of total cycles, but I don't see that anywhere. If not, I'd like the discrepancy to be explained. Thanks.

(For reference, the definition of cpucycles)

```
long long cpucycles(void) {
    unsigned long long result;
    __asm__ volatile(".byte 15;.byte 49;shlq $32,%%rdx;orq %%rdx,%%rax" : "=a" (result) :: "%rdx");
    return result;
}
```

—Jacob Alperin-Sheriff

From: Yongwoo Lee <yongwool@ccl.snu.ac.kr>
Sent: Monday, January 01, 2018 10:31 AM
To: Alperin-Sheriff, Jacob (Fed); pqc-comments
Cc: pqc-forum@list.nist.gov; jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; mypurist@gmail.com
Subject: RE: OFFICIAL COMMENT: pqsigRM

Dear Dr. Alperin-Sheriff

We appreciate for your comments.

We measured the CPU clocks but mistakenly wrote 'cycles' instead of 'clocks'.

We measured the cpu cycles again using the function you sent.

The new measurement is reflected in the table below.

	security	key generation	singing	verification	
-----	-----	-----	-----	-----	
pqsigRM-4-12	128	14639777783	3971208456	139814898	
pqsigRM-6-12	196	6395769782	3275234719	198607502	
pqsigRM-6-13	256	72162115384	1087667252	956410761	

In addition, We are constantly updating the program, you can always check the latest version of our submission on the website below:

: <https://sites.google.com/view/pqsigrm>

We will reflect your comments in our documentation and update it soon.

Happy new year!

Jong-Seon No,

Wijik Lee

Young-Sik Kim

Yong-Woo Lee

From: Alperin-Sheriff, Jacob (Fed) [mailto:jacob.alperin-sheriff@nist.gov]
Sent: Friday, December 29, 2017 7:17 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: pqc-forum@list.nist.gov; jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; mypurist@gmail.com; Yongwoo Lee <yongwool@ccl.snu.ac.kr>
Subject: OFFICIAL COMMENT: pqsigRM

From: Perlner, Ray (Fed)
Sent: Tuesday, January 02, 2018 5:05 PM
To: pqc-comments
Cc: pqc-forum@list.nist.gov; Alperin-Sheriff, Jacob (Fed)
Subject: OFFICIAL COMMENT: pqsigRM

Dear pqsigRM submitters,

Jacob and I believe we have found an attack on pqsigRM. We believe the punctured columns of the public parity check matrix can be identified statistically from a few hundred signatures. E.g. When we ran the submitted code to produce signatures for parameter set 4-12, we found that the bits of the signature corresponding to punctures were set to 1 about 45% of the time, while the other bits were only set to 1 about 31% of the time.

Best regards,
Ray Perlner

From: Yongwoo Lee <yongwool@ccl.snu.ac.kr>
Sent: Monday, January 08, 2018 7:08 PM
To: Perlner, Ray (Fed); pqc-comments
Cc: pqc-forum@list.nist.gov; Alperin-Sheriff, Jacob (Fed); 노종선 교수님; 이위직; 김영식교수님
Subject: RE: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Dear Perlner, Dear All;

Thank for your valuable comments.

As you mentioned, we have checked that the probability of 1's among the punctured/inserted elements is higher than that of the unpunctured elements in our proposed pqsigRM algorithms.

As you can see Algorithm 3 in the supporting documentation, the punctured/inserted part of the signature is generated in the following way;

$$e'_p{}^T = s'_p + Re_{(n-p)}{}^T$$

where s'_p is generated from the output of SHA512.

Hence the probability of occurrence of ones in the punctured/inserted part of the signatures is about to 1/2 and the probability of occurrence of ones in the unpunctured part is about w/n .

(Precisely, since we choose e 's having Hamming weight smaller than or equal to w as signature, e_p having larger Hamming weight is likely to be discarded. Hence, the probability of the occurrence of ones in the punctured/inserted part is slightly lower than 1/2. As you mentioned, it is about 45% in pqsigRM-4-12.)

As you mentioned, using this difference of the probabilities, an attacker can figure out the punctured/inserted elements. However, we think that this is not a major threat to the security of our proposed algorithm. Even though the attacker knows which bits are punctured/inserted in the signature, he cannot figure out the exact locations of the punctured/inserted bits before permutation.

The number of possible permutation matrices Q 's becomes $p!(n-p)!$ ($= 2^{43071}$ in pqsigRM-4-12, 128-bit security) from $n!$ ($= 2^{43250}$ in pqsigRM-4-12) if the locations of the punctured/inserted elements are known and it is still very large number and secure.

Moreover, to our knowledge, it does not reduce the complexity of any known attacks on RM code-based digital signature schemes such as Minder-Shokrollahi's attack, Chizhov-Borodin's attack, Square code attack, or information set decoding.

However, in order to avoid the possible threats, we'd like to slightly modify the algorithm and the parameters such that the probabilities of ones in the unpunctured and punctured/inserted parts are the same. The colored lines are slightly modified in Algorithm 3. (<https://sites.google.com/view/pqsigrm/home/documentation>, page 9)

Further, some parameters are also modified as in Table 1.

Table 2 shows the average numbers of iterations for signing the submitted algorithm and the modified algorithm, where the probabilities of ones in the unpunctured and punctured/inserted parts are the same.

Table 1. Parameters of the modified algorithms

Algorithms	original p	modified p	w_p	q
pqsigRM-4-12	20	16	7	59/256
pqsigRM-5-11	10	8	2	220/256
pqsigRM-6-12	20	8	2	1
pqsigRM-6-13	30	16	4	23/256

Table 2. Average number of iterations for signing

Algorithms	Avg. number of iter. (submitted)	Avg. number of iter. (modified)
pqsigRM-4-12	58.165	269.886 (4.64 times)
pqsigRM-5-11	6090.298	77590.397 (12.74 times)
pqsigRM-6-12	1774.464	277153.557 (156.19 times)
pqsigRM-6-13	7.4	637.880 (84.2 times)

From: Perlner, Ray (Fed) [mailto:ray.perlner@nist.gov]
Sent: Wednesday, January 3, 2018 7:05 AM
To: pqc-comments <pqc-comments@nist.gov>
Cc: pqc-forum@list.nist.gov; Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov>
Subject: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Dear pqsigRM submitters,

Jacob and I believe we have found an attack on pqsigRM. We believe the punctured columns of the public parity check matrix can be identified statistically from a few hundred signatures. E.g. When we ran the submitted code to produce signatures for parameter set 4-12, we found that the bits of the signature corresponding to punctures were set to 1 about 45% of the time, while the other bits were only set to 1 about 31% of the time.

Best regards,
Ray Perlner

--
You received this message because you are subscribed to the Google Groups "pqc-forum" group.

To unsubscribe from this group and stop receiving emails from it, send an email to pqc-forum+unsubscribe@list.nist.gov.

Visit this group at <https://groups.google.com/a/list.nist.gov/group/pqc-forum/>.

--
You received this message because you are subscribed to the Google Groups "pqc-forum" group.

From: Jacob Alperin-Sheriff <jacobmas@gmail.com>
Sent: Monday, January 08, 2018 10:04 PM
To: Yongwoo Lee
Cc: Perlner, Ray (Fed); pqc-comments; pqc-forum@list.nist.gov; Alperin-Sheriff, Jacob (Fed); 노종선 교수님; 이위직; 김영식교수님
Subject: Re: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Quick response tonight at home, Ray may add something tomorrow if he wants.

1. "As you mentioned, using this difference of the probabilities, an attacker can figure out the punctured/inserted elements. However, we think that this is not a major threat to the security of our proposed algorithm. Even though the attacker knows which bits are punctured/inserted in the signature, he cannot figure out the exact locations of the punctured/inserted bits before permutation.

The number of possible permutation matrices Q 's becomes $p!(n-p)!$ ($= 2^{43071}$ in pqsigRM-4-12, 128-bit security) from $n!$ ($= 2^{43250}$ in pqsigRM-4-12) if the locations of the punctured/inserted elements are known and it is still very large number and secure."

The total number of permutation matrices is irrelevant. The key point is that, by using the difference of probabilities, an attacker can find some permutation matrix Q' that moves each of the punctured bits to one of the final p positions of the vector (and ensures the non-punctured bits are all in the first $n-p$ positions of the vector). Concretely, we may choose Q' to move the leftmost punctured bit to the leftmost of the final p positions, the next-leftmost punctured bit to the next leftmost of the final p positions, and so on. Obviously, we will (except with very very very very small probability) have that $Q \neq Q'$.

However, we WILL have that $Q^*(Q')^{-1}$ is a block diagonal matrix, i.e.

$$Q^*(Q')^{-1} = \begin{bmatrix} U_1 & 0 \\ 0 & U_2 \end{bmatrix}$$

where U_1 is an $(n-p) \times (n-p)$ permutation matrix and U_2 is a $p \times p$ permutation matrix.

Let $X = H^*(Q')^{-1}$, where H' is the public key.

Now, note that

$$\left[\begin{array}{c|c} [P^T \mid I_{n-k-p}]U_1 & 0 \end{array} \right]$$

$$S^{-1}X = H_m Q (Q')^{-1} = \left[\begin{array}{c|c} RU_1 & U_2 \end{array} \right]$$

If I'm not mistaken (I will check with Ray and sources of previous attacks tomorrow morning), I believe this means we can fully break the scheme.

2. As we said in the call for proposals and have reiterated on this forum, "because of limited resources, and also to avoid moving evaluation targets (i.e., modifying the submitted algorithms undergoing public review), NIST will NOT accept modifications to the submitted algorithms during this initial phase of evaluation."

The change you have proposed here is clearly a modification to the submitted algorithms, so we will not be accepting it and we will judge the algorithm as submitted (the same goes for all submissions).

On Mon, Jan 8, 2018 at 7:08 PM, Yongwoo Lee <yongwool@ccl.snu.ac.kr> wrote:

Dear Perlner, Dear All;

Thank for your valuable comments.

As you mentioned, we have checked that the probability of 1's among the punctured/inserted elements is higher than that of the unpunctured elements in our proposed pqsigRM algorithms.

As you can see Algorithm 3 in the supporting documentation, the punctured/inserted part of the signature is generated in the following way;

$$e'_p{}^T = s'_p + Re_{(n-p)}{}^T$$

where s'_p is generated from the output of SHA512.

From: Perlner, Ray (Fed)
Sent: Wednesday, January 10, 2018 5:58 PM
To: Jacob Alperin-Sheriff; Yongwoo Lee
Cc: pqc-comments; pqc-forum@list.nist.gov; Alperin-Sheriff, Jacob (Fed); 노종선 교수님; 이위직; 김영식 교수님
Subject: RE: [pqc-forum] OFFICIAL COMMENT: pqsigRM

We believe that once the punctured columns are identified, we can reconstruct the entire RM code (with permuted columns), at which point standard RM attacks like Minder-Shokrollahi and Chizhov-Borodin can be applied.

We also believe that, even if rejection sampling is applied, preventing signatures from giving away information about the punctured columns, the same information can be recovered relatively inexpensively from the public key alone. According to the estimate of Minder and Shokrollahi, the original RM generator matrix has at least $2^{(rm - r(r-1))}$ minimum weight codewords (weight $2^{(m-r)}$). These will all be orthogonal to the $n-k-p$ dimensional subcode of the parity check matrix, which lacks support on the punctured columns. They can all be modified to codewords of the public code (with modestly increased weight) by substituting the appropriate bits in the punctured columns. (This is analogous to the signature procedure of the original scheme.) Such near-minimum-weight codewords can be found by standard information set decoding techniques, at a cost which we estimate to be significantly less than the claimed security level of any of the submitted parameter sets. Moreover, the punctured columns will be overrepresented in near minimum weight code words found by this technique.

Here's the procedure for reconstructing the code once you have the punctured columns:

First take the subcode of the public parity check matrix that lacks support on the punctured columns. (Remove the all-zero punctured columns from this subcode.) Now, you have a $n - k - p \times n - p$ submatrix of a parity check matrix for the original permuted r, m reed muller code. (Note it's also a submatrix of the permuted parity check matrix if up to p columns of zeroes are appended.) The dual code of this matrix contains in its row space the truncation of all the reed muller code words from the original code.

Recall that the minimum weight codewords of the original code all have hamming 2^r . Find $k-p$ linearly independent codewords from the truncated code that have weight 2^r . This can be done by information set decoding.

Now find a word in the truncated code with weight 2^{r-1} . (This can also be done by information set decoding.) Append a 1 to this code word, and append a zero to each of the $k-p$ codewords from the previous step. These generate a $k-p+1 \times n-p+1$ submatrix of a generator matrix of the permuted r, m reed muller code. Likewise, $p-1$ columns of zeroes could be appended, and it would still be a submatrix. Repeat this process, switching generator and parity check matrices each time to fill in all the missing columns of the generator and parity matrices of the punctured RM code.

We haven't done a full complexity analysis of the above, but crude heuristic estimates suggest the cost to be somewhere around 2^{70} for the originally submitted 128 and 192 bit parameters, and 2^{100} for the 256 bit parameters.

From: Jacob Alperin-Sheriff [mailto:jacobmas@gmail.com]
Sent: Monday, January 08, 2018 10:04 PM

From: Wijik Lee <leewj422@gmail.com>
Sent: Thursday, January 18, 2018 11:23 AM
To: pqc-forum
Cc: jacobmas@gmail.com; yongwool@ccl.snu.ac.kr; pqc-comments; Alperin-Sheriff, Jacob (Fed); jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; iamyskim@chosun.ac.kr; Perlner, Ray (Fed)
Subject: Re: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Dear Perlner, Dear All;

Thank for your valuable comments.

1. By slightly modifying our proposed algorithm, we can make the probabilities of the punctured/inserted and the unpunctured bits equal.

In this case, we believe that it is hard to find the exact locations of the punctured columns from the public key H' . In your comments, the near-minimum-weight codewords can be found by standard information set decoding techniques.

In fact, the codewords generated from H' are not true codewords of RM code but the vectors replaced by the random bits in the unknown punctured locations.

Further, the Hamming weight of those vectors is larger than or equal to $d_{\min} - p$ and we don't know their weight distribution.

2. If we do not modify our proposed algorithm, we need to increase the parameters of $RM(r, m)$ to increase the security level.

In case of $RM(6, 13)$, the security level will be close to 128 bits.

We didn't calculate exact security level yet, however:

-The number of codewords with Hamming weight d_{\min} in the punctured RM codes is reduced.

-The complexity of finding $n-p$ "independent" codewords with Hamming weight d_{\min} needs more than that of finding $n-p$ codewords with d_{\min} .

2018년 1월 11일 목요일 오전 7시 58분 7초 UTC+9, Perlner, Ray (Fed) 님의 말:

We believe that once the punctured columns are identified, we can reconstruct the entire RM code (with permuted columns), at which point standard RM attacks like Minder-Shokrollahi and Chizhov-Borodin can be applied.

We also believe that, even if rejection sampling is applied, preventing signatures from giving away information about the punctured columns, the same information can be recovered relatively inexpensively from the public key alone. According to the estimate of Minder and Shokrollahi, the original RM generator matrix has at least $2^{(rm - r(r-1))}$ minimum weight codewords (weight $2^{(m-r)}$). These will all be orthogonal to the $n-k-p$ dimensional subcode of the parity check matrix, which lacks support on the punctured columns. They can all be modified to codewords of the public code (with modestly increased weight) by substituting the appropriate bits in the punctured columns. (This is analogous to the signature procedure of the original scheme.) Such near-minimum-weight codewords can be found by standard information set decoding techniques, at a cost which we estimate to be significantly less than the claimed

From: Perlner, Ray (Fed)
Sent: Friday, January 19, 2018 10:16 AM
To: Wijik Lee; pqc-forum
Cc: jacobmas@gmail.com; yongwool@ccl.snu.ac.kr; pqc-comments; Alperin-Sheriff, Jacob (Fed); jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; iamyskim@chosun.ac.kr
Subject: Re: [pqc-forum] OFFICIAL COMMENT: pqsigRM

I'm confused why you think your point 1 contradicts our claim that we can recover the locations of the punctured columns from the private key alone.

"In fact, the codewords generated from H' are not true codewords of RM code but the vectors replaced by the random bits in the unknown punctured locations."

Indeed. If the modified RM codeword in question is a minimum weight codeword, the punctured bits will have probability $1/2$ to be set to 1, while the non-punctured bits will only be 1 with probability d_{min}/n . Since there is such a modified codeword for every minimum weight codeword in the original RM code, and their weight is only expected to be larger than d_{min} by a little less than $p/2$, I don't believe it would be difficult to recover enough such modified codewords to detect the puncturing locations.

While I reiterate, we are not accepting modifications to submitted parameters at this time, I don't think making the weight distribution of signatures more uniform is sufficient to hide the locations of the punctured columns.

From: Wijik Lee <leewj422@gmail.com>
Sent: Thursday, January 18, 2018 11:22:57 AM
To: pqc-forum
Cc: jacobmas@gmail.com; yongwool@ccl.snu.ac.kr; pqc-comments; Alperin-Sheriff, Jacob (Fed); jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; iamyskim@chosun.ac.kr; Perlner, Ray (Fed)
Subject: Re: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Dear Perlner, Dear All;
Thank for your valuable comments.

1. By slightly modifying our proposed algorithm, we can make the probabilities of the punctured/inserted and the unpunctured bits equal.

In this case, we believe that it is hard to find the exact locations of the punctured columns from the public key H' . In your comments, the near-minimum-weight codewords can be found by standard information set decoding techniques.

In fact, the codewords generated from H' are not true codewords of RM code but the vectors replaced by the random bits in the unknown punctured locations.

Further, the Hamming weight of those vectors is larger than or equal to $d_{min} - p$ and we don't know their weight distribution.

2. If we do not modify our proposed algorithm, we need to increase the parameters of $RM(r,m)$ to increase the security level.

In case of $RM(6, 13)$, the security level will be close to 128 bits.

We didn't calculate exact security level yet, however:

-The number of codewords with Hamming weight d_{min} in the punctured RM codes is reduced.

-The complexity of finding $n-p$ "independent" codewords with Hamming weight d_{min} needs more than that of finding $n-p$ codewords with d_{min} .

From: Moody, Dustin (Fed)
Sent: Monday, February 05, 2018 2:12 PM
To: Wijik Lee; pqc-forum
Cc: jacobmas@gmail.com; yongwool@ccl.snu.ac.kr; pqc-comments; Alperin-Sheriff, Jacob (Fed); jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; iamyskim@chosun.ac.kr; Perlner, Ray (Fed)
Subject: RE: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Pqsigrm team,

We are working on the program for the 1st NIST PQC workshop, which is quite challenging. We received a large number of submissions, and only have 2 days.

There are several submissions which have been attacked, with the submitter(s) acknowledging (to some degree) that the attack(s) are successful. NIST will be evaluating the security of all submissions with respect to the original algorithm and parameters contained in the submission; for the 1st round, as per the call for proposals: we are not allowing changes to submitted algorithm or parameters to avoid moving targets. Any team whose submission has successfully been broken should probably consider withdrawing their submission.

pqsigrm is one of these submissions with an attack. We ask you to consider sending us slides or a video in place or presenting, as it would help us in creating the workshop program. Right now, the time constraints we have, combined with the number of submissions is making the schedule difficult. We are still encouraging you to attend, but feel the time for presentations might be best served by being used for submissions which have not yet been successfully attacked. Please let us know if you would still like to present, or will instead send us slides or a video.

Please let us know as soon as you can. Thanks,

Dustin

From: Wijik Lee [mailto:leewj422@gmail.com]
Sent: Thursday, January 18, 2018 11:23 AM
To: pqc-forum <pqc-forum@list.nist.gov>
Cc: jacobmas@gmail.com; yongwool@ccl.snu.ac.kr; pqc-comments <pqc-comments@nist.gov>; Alperin-Sheriff, Jacob (Fed) <jacob.alperin-sheriff@nist.gov>; jsno@snu.ac.kr; leewj422@ccl.snu.ac.kr; iamyskim@chosun.ac.kr; Perlner, Ray (Fed) <ray.perlner@nist.gov>
Subject: Re: [pqc-forum] OFFICIAL COMMENT: pqsigRM

Dear Perlner, Dear All;

Thank for your valuable comments.

1. By slightly modifying our proposed algorithm, we can make the probabilities of the punctured/inserted and the unpunctured bits equal.

In this case, we believe that it is hard to find the exact locations of the punctured columns from the public key H' . In your comments, the near-minimum-weight codewords can be found by standard information set decoding techniques.

In fact, the codewords generated from H' are not true codewords of RM code but the vectors replaced by the random bits in the unknown punctured locations.