From: vadim1980@gmail.com on behalf of Vadim Lyubashevsky <vadim.lyubash@gmail.com>
Sent: Tuesday, January 02, 2018 11:34 AM
To: pqc-comments
Cc: pqc-forum@list.nist.gov
Subject: OFFICIAL COMMENT: CRYSTALS-DILITHIUM

Dear all,

We are very grateful to Peter Pessl for notifying us of an implementation error in a randomness generator of our NIST submission. The bug was in the function rej_gamma1m1 in poly.c and consisted of accidentally overwriting a variable prior to using it. This function is used for sampling the masking vector y (line 13 of Figure 4 in the supporting documentation), and the result of the bug was that the same randomness ended up being used for pairs of consecutive coefficients, whereas the specification demands that all the coefficients be independent.

This reuse of randomness can easily be exploited to recover the secret key and we thus emphasize that the software, in the state submitted to NIST, should not be used in any real application.

We fixed the bug in an updated version of the software, which is available from the CRYSTALS website at https://pq-crystals.org/dilithium/resources.shtml. On the site, we also re-packaged the NIST submission package to include the updated KAT vectors.

Sincerely,

The CRYSTALS Team

From: 4akolzinaolga@gmail.com
Sent: Monday, March 26, 2018 11:03 AM
To: pqc-forum
Cc: pqc-comments
Subject: [pqc-forum] Re: OFFICIAL COMMENT: CRYSTALS-DILITHIUM

Dear all!

Typically, the standards (such as X9.98, SHA - 3) use Converting Between Bit Strings and Right-Padded Octet Strings.
Why do you use Converting Between Bit Strings and Left-Padded Octet Strings?

24 bits instead of 23 bits usage for A matrix generation needs to generate extra 256 pseudo random bits for each matrix element. Is this for security reasons or for code simplicity?

Digital signature verification
It isn't clear, why send the public key in the field before hashing, and don't hash it directly?

I.e. operators:

```
for(i = 0; i < CRYPTO_PUBLICKEYBYTES; ++i)
    m[CRYPTO_BYTES - CRYPTO_PUBLICKEYBYTES + i] = pk[i];
```

and

```
shake256(m + CRYPTO_BYTES - CRHBYTES, CRHBYTES,
           m + CRYPTO_BYTES - CRYPTO_PUBLICKEYBYTES,
CRYPTO_PUBLICKEYBYTES);
```

replace with operators:

```
shake256(m + CRYPTO_BYTES - CRHBYTES, CRHBYTES,
           pk, CRYPTO_PUBLICKEYBYTES);
```

Thank you! Best regards,

I. Gorbenko, E. Kachko, M. Yesina, O. Akolzina

From: Gregor Seiler <gseiler@inf.ethz.ch>
Sent: Thursday, April 05, 2018 9:38 AM
To: 4akolzinaolga@gmail.com; pqc-forum
Cc: pqc-comments
Subject: Re: [pqc-forum] Re: OFFICIAL COMMENT: CRYSTALS-DILITHIUM

Dear all,

On 26.03.2018 17:03, 4akolzinaolga@gmail.com wrote:
> Typically, the standards (such as X9.98, SHA - 3) use Converting
> Between Bit Strings and Right-Padded Octet Strings.
>
> Why do you use Converting Between Bit Strings and Left-Padded Octet Strings?

We don't entirely understand where you are referring to when you say we are converting between bit strings and "left-padded octet strings". We repeatedly pack vectors of unsigned b-bit integers into byte strings for various different b. But since the integer vectors always have 256 coefficients there is no padding needed. We store the bytes of the integers in little-endian order as most modern CPUs use this byte ordering. So if we would for example convert a single integer to a byte string then the most significant bits of the last byte would be zero.
See Section 5.2 in the specification for an explanation of our data formats.

> 24 bits instead of 23 bits usage for A matrix generation needs to
> generate extra 256 pseudo random bits for each matrix element. Is this
> for security reasons or for code simplicity?

Using 23 random bits per coefficient in the sampling of A would be much more difficult to implement and would also not save any randomness because one would still need 5 blocks of SHAKE-128 output per polynomial. Notice that we sample each polynomial in the matrix A from a separate output stream of SHAKE-128 corresponding to a distinct input. Although this means we throw away more randomness compared to sampling all polynomials from the same stream, it is necessary in order to take advantage of a vectorized SHAKE implementation, which we do in our optimized implementation which uses AVX2 instructions.

> Digital signature verification
>
> It isn’t clear, why send the public key in the field before hashing,
> and don’t hash it directly?
>
> I.e. operators:
>
> for(i = 0; i < CRYPTO_PUBLICKEYBYTES; ++i)
>
> m[CRYPTO_BYTES- CRYPTO_PUBLICKEYBYTES+ i] = pk[i];
>
> and
>
> shake256(m+ CRYPTO_BYTES- CRHBYTES, CRHBYTES,
>
> m+ CRYPTO_BYTES- CRYPTO_PUBLICKEYBYTES, CRYPTO_PUBLICKEYBYTES);
>
> replace with operators:
>

> shake256(m+ CRYPTO_BYTES- CRHBYTES, CRHBYTES,
>
> pk, CRYPTO_PUBLICKEYBYTES);

We could directly hash the public key as you suggest without first copying it to the message buffer, but it wouldn't make verification any faster. We might change this in a more optimized version of our AVX2 code together with omissions of other copying that make the reference code more readable.

Regards,
Gregor

From: 'James Howe' via pqc-forum <pqc-forum@list.nist.gov>
Sent: Tuesday, September 18, 2018 5:43 AM
To: pqc-forum
Subject: [pqc-forum] OFFICIAL COMMENT: Dilithium

Dear All,

We would like to bring to notice a recent work of ours on a side-channel assisted attack on Dilithium, the lattice-based signature scheme. The main motivation of this paper is to highlight the fact that signatures can be forged with only the partial knowledge (i.e., s1) of the secret-key which in Dilithium consists of the components (rho,tr,s1,s2,t0,K), wherein s1 and s2 are the secret and error components, respectively, of the public-key's LWE instance. Once s1 is known in previous lattice-based signature schemes like
BLISS, Ring-Tesla, and GLP, (i.e., based on the Fiat-Shamir with aborts framework) they broke down upon knowing s1. However, Dilithium only publishes the compressed version of the LWE instance as the public-key and thus does not make it trivial to recover the other secrets just upon knowing s1. We propose a very simple forgery procedure that forges signature with only the knowledge of s1, thus showing that s1 appears to be the most important component of the secret-key. We do not claim any major break of Dilithium, but just highlight the fact that secrecy of s1 is very crucial for its security. Moreover, we also acknowledge a parallel work by Pessl and Bruienderink [1] who also proposed a similar technique to forge signatures with only knowing s1. Thus, we stress on the fact that it is crucial to protect s1 from possible side-channel attacks.

Moreover, we also see a possibility that our forgery procedure (that uses only s1) could be used as an alternative signature scheme for Dilithium, since we observed a speed-up of around 2.67x compared to the original signing procedure. But our forgery signing procedure comes with a certain error probability, since it is not possible to perform some checks for the rejection conditions which ensure correctness of Dilithium. We have not performed a mathematical analysis of its failure probability, but tried to validate our claim through experiments. We observed that we were able to output around 2^28 valid signatures without a single invalid signature. We feel it might be possible to concretely derive the failure probability of our forgery signing procedure so that it can be considered as a faster alternative signing procedure, provided it does not leak any information about the secret-key.

Please find our paper at: https://eprint.iacr.org/2018/821

[1] Leon Groot Bruinderink and Peter Pessl, Differential Fault Attacks on Deterministic Lattice Signatures, IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018.

Regards,
James Howe and co-authors

From: Vadim Lyubashevsky <vadim.lyubash@gmail.com>
Sent: Wednesday, September 19, 2018 8:15 AM
To: jameshowe007@googlemail.com
Cc: pqc-forum@list.nist.gov
Subject: Re: [pqc-forum] OFFICIAL COMMENT: Dilithium

Dear all,

> Once s1 is known in previous lattice-based signature schemes like
> BLISS, Ring-Tesla, and GLP, (i.e., based on the Fiat-Shamir with aborts framework) they broke down upon
> knowing s1. However, Dilithium only publishes the compressed version of the LWE instance as the public-key and thus
> does not make it trivial to recover the other secrets just upon knowing s1.

This statement is not correct. Just like in the other LWE schemes, knowing the full secret s_1 in Dilithium is enough to trivially recover all the other necessary secret information from the public key -- so no side channel attacks are needed to break the scheme if s_1 is known.

The public key / secret key relationship in Dilithium is As_1+s_2 = N*t_1+t_0, where N is 2^14. The public key that's given out is A and t_1, while the signer uses s_1,s_2,t_0 to create signatures.

If the attacker knows the entire s_1, he can compute As_1 - N*t_1, which is equal to t_0 - s_2. The vector t_0 has all its coefficients of absolute value < 2^13 and s_2 has coefficients at most 5. It is now trivial to obtain another solution t_0' and s_2' such that t_0' - s_2' = t_0 - s_2 that have the same range constraints on t_0' and s_2' -- note that this will not necessarily be the same solution t_0,s_2 that the real signer has, but it doesn't matter. Now that the adversary has s_1,s_2',t_0' all in the correct range that satisfy As_1+s_2' = N*t_1+t_0', he can use them to produce valid signatures since his secret key is as good as the real signer's.

So in summary, s_1 is really the only secret key that the adversary or the signer, if he's willing to recompute s_2',t_0' before doing the signatures every time, need in the signature scheme. And so the side-channel attack in the paper really assumes initial knowledge of the entire secret key.

Best,
-Vadim