From:	pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Sunday, December 6, 2020 7:11 PM
To:	pqc-forum
Subject:	[pqc-forum] ROUND 3 OFFICIAL COMMENT: SIKE -- CCA Bug

Hello,

Regarding SIKE Round 3 submission code -- and the current code at sike.org.

File "fpx.c" -- which is a part of the optimized, arm64, and x86 implementations has this function:

```
20: int8_t ct_compare(const uint8_t *a, const uint8_t *b, unsigned int len)
21: { // Compare two byte arrays in constant time.
22:   // Returns 0 if the byte arrays are equal, -1 otherwise.
23:     uint8_t r = 0;
24:
25:     for (unsigned int i = 0; i < len; i++)
26:         r |= a[i] ^ b[i];
27:
28:     return (-(int8_t)r) >> (8*sizeof(uint8_t)-1);
29: }
```

Due to the unfortunate cast to a signed 8-bit integer on line 28 (rather than a larger type -- as required for the arithmetic shift trick) this function does not work correctly and will usually return 0 even when the strings are unequal; If bit 7 of r is set, then -r is a positive value, yielding a 0. For random len-byte strings, a *nonzero* return value has prob 2^-len -- vanishingly small. Furthermore the function has a third return value, 0x01 e.g. when a ^ b == 0x80.

This ct_compare() implementation is used in the Fujisaki-Okamoto step of "sike.c" in optimized SIKE implementations -- the output is used directly as an argument for cmov (conditional move) for implicit rejection. Not sure what the full implications to are in this particular case, but quite often flaws in re-encryption verification and rejection are exploitable. In any case, it would appear that the submitted optimized SIKE implementations are not CCA -- as rejection is basically never performed?

Above flaw is not a timing attack. It should be noted that memcmp() is used throughout in the compressed implementations; "dlog.c", "ec_isogeny.c", "sidh_compressed.c", "torsion_basis.c". So these compressed implementations are not constant time (the specification seems to suggest that the optimized implementations would be).

Best Regards,
- markku

Dr. Markku-Juhani O. Saarinen <mjos@pqshield.com> PQShield, Oxford UK.

From:	pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Monday, December 7, 2020 6:48 AM
To:	pqc-forum
Cc:	Markku-Juhani O. Saarinen
Subject:	[pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: SIKE -- CCA Bug

Hi Again,

The compressed SIKE variants do things differently but are not CCA either.

To avoid recompression the Fujisaki-Okamoto implementation is based on comparing curve points. There are two bugs here -- both of them kill the CCA claim.

Here's "fpx.c" of Optimized_Implementation/portable/SIKEp434_compressed, submitted to NIST:

```
int8_t cmp_f2elm(const f2elm_t x, const f2elm_t y)
{ // Comparison of two GF(p^2) elements in constant time.
 // Is x != y? return -1 if condition is true, 0 otherwise.
   f2elm_t a, b;
   uint8_t r = 0;

   fp2copy(x, a);
   fp2copy(y, b);
   fp2correction(a);
   fp2correction(b);

   for (int i = NWORDS_FIELD-1; i >= 0; i--)
       r |= (a[0][i] ^ b[0][i]) | (a[1][i] ^ b[1][i]);

   return (-(int8_t)r) >> (8*sizeof(uint8_t)-1);
}
```

Here f2elm is of native word size (32- or 64-bit), but only its low 8 bits are compared as "r" is 8 bits. So the comparator will return 0 either if the low 8 bits of the words match but also if the high bit of r is 1 after the loop (meaning that bit 7 differs somewhere).

My colleague noticed that there's a patch for it in Microsoft code base for it from last night; r is still 8 bit and the bug is not fixed: https://github.com/microsoft/PQCrypto-SIDH/commit/23cb2260a0196e4921cd2586be16df7e9163ff0b

It's actually kind of surprising how such clever cryptographers stumble over this particular issue over and over again. The comparison in FO should be a well-known spot to check especially after that [QuJoNi20] at CRYPTO last summer. I thought it was slightly unfair to mention FrodoKEM in the title but the paper of course had an interesting attack/exploit in it, which justified its publication. However, I also think that it's easier for everyone if we just report such errors in submitted implementations on this mailing list without having to go through the whole exploit-publication cycle.

[QuJoNi20] Qian Guo and Thomas Johansson and Alexander Nilsson: "A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM", IACR-CRYPTO 2020, https://eprint.iacr.org/2020/743

Cheers,
- markku

Dr. Markku-Juhani O. Saarinen <mj...@pqshield.com> PQShield, Oxford UK.

On Monday, December 7, 2020 at 12:11:19 AM UTC Markku-Juhani O. Saarinen wrote:

> Hello,
>
> Regarding SIKE Round 3 submission code -- and the current code at sike.org.
>
> File "fpx.c" -- which is a part of the optimized, arm64, and x86 implementations has this function:
>
> ```
> 20: int8_t ct_compare(const uint8_t *a, const uint8_t *b, unsigned int len)
> 21: { // Compare two byte arrays in constant time.
> 22:   // Returns 0 if the byte arrays are equal, -1 otherwise.
> 23:     uint8_t r = 0;
> 24:
> 25:     for (unsigned int i = 0; i < len; i++)
> 26:         r |= a[i] ^ b[i];
> 27:
> 28:     return (-(int8_t)r) >> (8*sizeof(uint8_t)-1);
> 29: }
> ```
>
> Due to the unfortunate cast to a signed 8-bit integer on line 28 (rather than a larger type -- as required for the arithmetic shift trick) this function does not work correctly and will usually return 0 even when the strings are unequal; If bit 7 of r is set, then -r is a positive value, yielding a 0. For random len-byte strings, a *nonzero* return value has prob 2^-len -- vanishingly small. Furthermore the function has a third return value, 0x01 e.g. when a ^ b == 0x80.
>
> This ct_compare() implementation is used in the Fujisaki-Okamoto step of "sike.c" in optimized SIKE implementations -- the output is used directly as an argument for cmov (conditional move) for implicit rejection. Not sure what the full implications to are in this particular case, but quite often flaws in re-encryption verification and rejection are exploitable. In any case, it would appear that the submitted optimized SIKE implementations are not CCA -- as rejection is basically never performed?
>
> Above flaw is not a timing attack. It should be noted that memcmp() is used throughout in the compressed implementations; "dlog.c", "ec_isogeny.c", "sidh_compressed.c", "torsion_basis.c". So these compressed implementations are not constant time (the specification seems to suggest that the optimized implementations would be).
>
> Best Regards,
> - markku
>
> Dr. Markku-Juhani O. Saarinen <mj...@pqshield.com> PQShield, Oxford UK.

From: pqc-forum@list.nist.gov on behalf of David Jao <djao@uwaterloo.ca>
Sent: Monday, December 7, 2020 8:50 PM
To: pqc-forum
Subject: Re: [pqc-forum] Re: ROUND 3 OFFICIAL COMMENT: SIKE -- CCA Bug

Hi Markku,

Thank you for this report.

We are fixing two bugs, one in the field element comparison in the compression code, and another one when casting the final result in the constant-time comparison used during decapsulation. We are also adding tests to make sure we catch changes in the ciphertext (i.e., to make sure the CCA mechanism works). All these changes are now available in the SIDH library: https://github.com/microsoft/PQCrypto-SIDH/

> My colleague noticed that there's a patch for it in Microsoft code base for it from last night; r is still 8 bit and the bug is not fixed: https://github.com/microsoft/PQCrypto-SIDH/commit/23cb2260a0196e4921cd2586be16df7e9163ff0b

One of the consequences of doing development is the open is that everyone can see what we're doing. Notwithstanding this openness, we will be sure to announce changes on this list.

> Above flaw is not a timing attack. It should be noted that memcmp() is used throughout in the compressed implementations; "dlog.c", "ec_isogeny.c", "sidh_compressed.c", "torsion_basis.c". So these compressed implementations are not constant time (the specification seems to suggest that the optimized implementations would be).

The SIKE spec specifically addresses this issue already, in Section 1.5, on page 19: "Note that, since all of the operations in both algorithms are performed on public data, side-channel countermeasures (e.g., constant-time routines) are irrelevant except for the last step of the decompression algorithm in which we compute the last kernel generator using Ladder3pt (Algorithm 8)."

I believe that when someone claims that the implementation of an algorithm is constant-time, it implicitly means that the private data processing is constant-time, which is where it is really required. We are not claiming that every function is constant time everywhere, even when it is not needed. In case our meaning was ambiguous, this paragraph hereby clarifies our intended meaning.

-David