From:	pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Sunday, November 1, 2020 5:52 PM
To:	pqc-forum
Subject:	[pqc-forum] ROUND 3 OFFICIAL COMMENT: HQC (SHA-2 and SHA-3 in a KEM, Buffer Overflows)

Dear HQC Authors,

First of all, I am puzzled by the requirement of HQC.KEM to implement both SHA-2 and SHA-3. This is not a very good way to perform domain separation -- as that is presented as the rationale in Section 2.3.2 of the submission document. Authors may want to look at NIST SP 800-185, which provides a standard way of performing domain separation via SHA-3 derived functions.

Even though HLS translated C-to-HDL should not be confused with actual hardware implementations (which is correctly noted in Section 3.3), even such numbers should make it clear that requiring two completely separate hash functions in a KEM greatly increases implementation size. The situation would be similar in embedded software implementations; Especially SHA-512 requires a large chunk of firmware on microcontroller targets.

--

We'd expect that some attention would have been paid to the implementation robustness of Round 3 submissions. I'm afraid that HQC is one of several submissions that may continue to avoid "advanced implementation attacks" solely because they have to be bug-fixed to hold together and not crash even in basic KAT verification :(

So, HQC round 3 reference implementations have buffer overflows. Even though typically exhibited as program crashes, it is well known that vulnerabilities of this type can also have very serious consequences, including privilege escalation and remote code execution.

Brief diagnostics (first-aid bugfixes):

- gf2x.c:vect_mul() has a tmp variable

```
uint64_t tmp[VEC_N_SIZE_64 << 1];
```

The size of tmp is therefore ceil(n/64) * 2 64-bit words; in case of HQC-256 this means 16 * 901 = 14416 bytes. The function fast_convolution_mult() is called which writes to "o", overwriting the table when s=3601 or larger; a frequent occurrence since 901*4=3604 16-bit words. The table is in the stack so potentially anything can happen, including overwriting of the return address and resulting in an unauthorized branch on some platforms. This overwriting occurs because the following loop has one extra iteration:

```
for (int32_t j = 0 ; j < VEC_N_SIZE_64 + 1 ; j++) {
```

That same loop will also crash many targets as it is directly casting non-aligned values to 64-bit pointers and writing to those pointers. This is not allowed in portable C. (It is also apparent that this reference code does not run on big-endian platforms. I could not see any specification about the encoding of endianness in the supporting documentation.)

- gf.c:gf_inverse(): a lot of calls come from reed_solomon.c:compute_error_values() with input parameter a = 0

```
uint16_t gf_inverse(uint16_t a) {
	return exp[PARAM_GF_MUL_ORDER - log[a]];
}
```

Since log[0] = 256 and PARAM_GF_MUL_ORDER = 255, the return value is _something_ at index -1 before this static location. Fortunately it is typically just alignment zeros.

Here "log" and "exp" These are static tables defined in a header file gf.h, shadowing well-known standard math library primitives. This represents very poor programming style.

- The Optimized implementation, which targets Intel AVX2 only (rather than generic ANSI C -- typically such explicitly non-portable implementations are considered as "additional implementations"), has several additional programming errors. This is apparent in trying to use it; removing the link-time optimization flag "-flto" causes the "hqc-128-kat" code in the submission itself to segfault (Ubuntu 20.04, GCC 9.3.0). gdb points to karat_mult9() as the crash point. This function is directly casting 64-bit pointers as 256-bit vector pointers and performing arithmetic on them; this is of course an illegal operation if pointers happen to be unaligned. Nothing is preventing them from being unaligned; hence crash.

Best Regards,
- markku

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

From: pqc-forum@list.nist.gov on behalf of John Schanck <jschanck@uwaterloo.ca>
Sent: Monday, November 2, 2020 8:09 AM
To: Markku-Juhani O. Saarinen
Cc: pqc-forum
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: HQC (SHA-2 and SHA-3 in a KEM, Buffer Overflows)

For what it is worth, all of the bugs that Markku mentions were present in earlier versions of the HQC code and they were caught and fixed by integrating HQC into PQClean [1, 2].

One other potentially exploitable issue (that is fixed in PQClean) is the use of short-circuiting boolean operators in the FO re-encryption test
(kem.c:149):

```
> result = (vect_compare(u, u2, VEC_N_SIZE_BYTES) == 0 &&
> vect_compare(v, v2, VEC_N1N2_SIZE_BYTES) == 0 && vect_compa
> re((uint64_t *)d, (uint64_t *)d2, SHA512_BYTES) == 0);
```

The code currently in PQClean was produced using the "package.sh" script at [3]. That repository contains a set of patches [4] on the upstream HQC code, which is easier to review than the full diff between upstream and PQClean.

Cheers,
John

[1] https://github.com/PQClean/PQClean/pull/324
[2] https://github.com/PQClean/PQClean/pull/348
[3] https://github.com/jschanck/package-pqclean/tree/main/hqc
[4] https://github.com/jschanck/package-pqclean/tree/main/hqc/patches

From:	pqc-forum@list.nist.gov on behalf of Gaborit <gaborit@unilim.fr>
Sent:	Tuesday, November 3, 2020 12:36 PM
To:	pqc-forum
Cc:	Philippe Gaborit; mjos.crypto@gmail.com
Subject:	Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: HQC (SHA-2 and SHA-3 in a KEM, Buffer Overflows)

Dear Markku,

Thank you for your comments in our implementations.

Concerning the reference implementation, we agree on what you mention, but as mentionned in our specifications, the reference version is not secure (and is not supposed to be), for instance it is not constant time. As mentionned by John from PQClean, it was already the case for our previous versions. For Round 3 we focused on our constant time version. We will nonetheless include the suggested changes from PQClean in the future.

Regarding the optimized version, thank you for pointing this out, as far as we know, optimized implementation are not required to be portable, and we optimized it for NIST reference platform. It should not be a problem to propose a portable version if requested by NIST.

If, for some reason, someone wanted to alter the compilation flags as you mention, there exist several possibilities, one of them is to use GCC's aligned attribute,_attribute__((aligned(32))),
for all arguments of the vect_mul function. We checked it and it didn't alter performances.

best,

Philippe for the HQC team

Le 01/11/2020 à 23:51, Markku-Juhani O. Saarinen a écrit :
> Dear HQC Authors,
>
> First of all, I am puzzled by the requirement of HQC.KEM to implement
> both SHA-2 and SHA-3. This is not a very good way to perform domain
> separation -- as that is presented as the rationale in Section 2.3.2
> of the submission document. Authors may want to look at NIST SP
> 800-185, which provides a standard way of performing domain separation
> via SHA-3 derived functions.
>
> Even though HLS translated C-to-HDL should not be confused with actual
> hardware implementations (which is correctly noted in Section 3.3),
> even such numbers should make it clear that requiring two completely
> separate hash functions in a KEM greatly increases implementation
> size. The situation would be similar in embedded software
> implementations; Especially SHA-512 requires a large chunk of
> firmware on microcontroller targets.

From: pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent: Tuesday, November 3, 2020 4:36 PM
To: pqc-forum
Cc: Gaborit; Philippe Gaborit
Subject: Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: HQC (SHA-2 and SHA-3 in a KEM, Buffer Overflows)

On Tue, Nov 3, 2020 at 5:36 PM Gaborit <gaborit@unilim.fr> wrote:

> Dear Markku,
>
> Thank you for your comments in our implementations.

Hello,

My main comment was really about the internal requirement of HQC to implement both SHA-2 (FIPS 180-4) and SHA-3 (FIPS 202), which makes HQC more unnecessarily unattractive from hardware or embedded implementation viewpoint.

It seems that additionally AES (FIPS 197) is required by this KEM as it is not possible to implement an interoperable version of HQC without it due to its use of AES seed expanders (with seeds -- AES keys -- encoded into keys and messages).

All of these functions (G/H/XOF) could have been achieved with FIPS 202 SHAKE alone -- this is a very easy change to make, significantly improves implementation footprint, but obviously results in an incompatible implementation. HQC is the only Round 3 candidate left that is using the AES seed expander from NIST .. well, the Rainbow code goes even further and modifies the NIST test vector DRNG code into an ad hoc XOF for no reason. Rainbow internally requires a constant-time AES too.

> Concerning the reference implementation, we agree on what you mention,
> but as mentionned in our specifications, the reference version is not
> secure (and is not supposed to be), for instance it is not constant
> time. As mentionned by John from PQClean, it was already the case for
> our previous versions.

This is an interesting statement and probably not a very well-considered one. Surely a reference implementation, the only portable implementation, needs to be at least safe to run and free of buffer overflow vulnerabilities, which are usually considered more critical in production code than timing attacks. The behavior of the submitted reference code is not even deterministic as it was reading memory out-of-bounds.

NIST can only react to published attacks, so I guess it is our duty to report on vulnerabilities on all of the candidates, even if they appear trivial. Trivial bugs usually have the most devastating consequences, but make less interesting scientific writeups.

> For Round 3 we focused on our constant time version.

I think John Schanck already mentioned this, but the optimized version does not appear to be constant time.

The decapsulation function crypto_kem_dec() (kem.c in the optimized implementation) computes its (explicit-rejection) Fujisaki-Okamoto result via:

result = (vect_compare(u, u2, VEC_N_SIZE_BYTES) == 0 && vect_compare(v, v2, VEC_N1N2_SIZE_BYTES) == 0 && vect_compare((uint64_t *)d, (uint64_t *)d2, SHA512_BYTES) == 0);

Such a conjunction is evaluated from left to right; the 2nd and 3rd comparisons are evaluated only if the first one yields a zero, and the third one only if first and second == 0. This is known as short-circuit evaluation -- it is a part of the C language, and some programmers rely on it.

Here u and v are two parts of the ciphertext; so the timing oracle will reveal if half of it matches in re-encryption, and again if the full re-encryption matches. The oracle will additionally reveal if d=hash of m will match that contained in the ciphertext (but that is also clear from the explicit rejection out).

> there exist several possibilities, one of them is to use GCC's aligned
> attribute,_attribute__((aligned(32))),
> for all arguments of the vect_mul function. We checked it and it didn't
> alter performances.

The current optimized code crashes if a different compiler or link-time optimization setting is used. I think it would best that HQC team tries to fix illegal loads and store instructions in their code themselves. Having correct prototypes and attributes is indeed a good start for writing robust SIMD programs. Perhaps the community will look more seriously into side-channel issues once at least somewhat portable, robustly working code is available. Good luck!

Best Regards,
- markku

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

> best,
>
> Philippe for the HQC team
>
>
> Le 01/11/2020 à 23:51, Markku-Juhani O. Saarinen a écrit :
> > Dear HQC Authors,
> >
> > First of all, I am puzzled by the requirement of HQC.KEM to implement
> > both SHA-2 and SHA-3. This is not a very good way to perform domain
> > separation -- as that is presented as the rationale in Section 2.3.2
> > of the submission document. Authors may want to look at NIST SP
> > 800-185, which provides a standard way of performing domain separation
> > via SHA-3 derived functions.
> >
> > Even though HLS translated C-to-HDL should not be confused with actual
> > hardware implementations (which is correctly noted in Section 3.3),
> > even such numbers should make it clear that requiring two completely
> > separate hash functions in a KEM greatly increases implementation
> > size. The situation would be similar in embedded software
> > implementations; Especially SHA-512 requires a large chunk of

From:	pqc-forum@list.nist.gov on behalf of Carlos Aguilar Melchor <carlos.aguilar@isae-supaero.fr>
Sent:	Thursday, November 5, 2020 10:14 AM
To:	pqc-forum
Cc:	mjos....@gmail.com; carlos.aguilar@isae-supaero.fr
Subject:	Re: [pqc-forum] ROUND 3 OFFICIAL COMMENT: HQC (SHA-2 and SHA-3 in a KEM, Buffer Overflows)

Dear Markku,

Thank you for your remarks, we will take them into account when making the next release, which will include our hardware implementation.

About your main remark, the preliminary hardware implementation announced in the Round 3 documentation already uses a SHA3 core as a unique basis for domain separation and randomness generation. This hardware implementation was announced as preliminary in particular because the software was not yet doing the same. It will in next release.

>> The current optimized code crashes if a different compiler or link-time optimization setting is used.

Being robust to compiler or flag changes is not a NIST requirement and will not be taken into consideration until it is. We will however improve load/store instructions directly in the code, in next release, as it requires little work and it does improve code quality.

best,

Carlos, for the HQC team.

Le mardi 3 novembre 2020 à 22:36:20 UTC+1, mjos....@gmail.com a écrit :

> On Tue, Nov 3, 2020 at 5:36 PM Gaborit <gab...@unilim.fr> wrote:
>
>> Dear Markku,
>>
>> Thank you for your comments in our implementations.
>
> Hello,
>
> My main comment was really about the internal requirement of HQC to implement both SHA-2 (FIPS 180-4) and SHA-3 (FIPS 202), which makes HQC more unnecessarily unattractive from hardware or embedded implementation viewpoint.
>
> It seems that additionally AES (FIPS 197) is required by this KEM as it is not possible to implement an interoperable version of HQC without it due to its use of AES seed expanders (with seeds -- AES keys -- encoded into keys and messages).
>
> All of these functions (G/H/XOF) could have been achieved with FIPS 202 SHAKE alone -- this is a very easy change to make, significantly improves implementation footprint, but obviously results in an incompatible implementation. HQC

From:	pqc-forum@list.nist.gov on behalf of Markku-Juhani O. Saarinen <mjos.crypto@gmail.com>
Sent:	Sunday, November 8, 2020 1:56 PM
To:	pqc-forum
Subject:	[pqc-forum] ROUND 3 OFFICIAL COMMENT: GeMSS reference code buffer overflow, constant-time claims?

1. Buffer Overflow

The reference implementation of GeMSS exhibits buffer overflows at least for one of the variants, WhiteGeMSS192. This is a read overflow at src/convMQ_gf2.c:285 (convMQ_last_uncompressL_gf2) which reads at least 8 bytes past the end of the public key when decoding it.

This is a GCC 9.3.0 / Ubuntu 20.04 but the bug is not platform-dependent. Test vectors do match. I diagnosed this as an error in pointer calculation in public key decoding, but perhaps the authors have some other explanation (public key too short?). In any case, buffer overflow errors are always serious, as are malfunctions in signature verification.

Address calculations at this part of the code are admittedly exceedingly (and unnecessarily) complex. Auditing this code for security is made nontrivial by the fact the reference version has 10,398 lines of C source and further 11,117 lines of headers -- even if one discounts duplicate files. For the optimized version the C code and header files reach 13,911 + 27,325 = 41,236 lines without duplicate files or libraries.

This is not necessarily the fault of the GeMSS algorithm itself -- it's just that only a research code type implementation is available, where almost historical options have not been removed (code related to QUARTZ and even SHA-1 is still in there). A production-quality implementation would look quite different, but someone would need to invest quite a lot of time in it.

2. Constant-time claims?

I appreciate that a lot of effort has been put into making GeMSS resilient against timing attacks, but perhaps the authors can clarify which implementation is claimed to have this property (if any).

From the remarks in the code, it appears that the optimized implementation has more constant-time code than the reference implementation. However the actual executed signature-generating function signHFE_FeistelPatarin() has this remark:

src/signHFE.c:428
" * @remark A part of the implementation is not in constant-time."

This function is also in use in the alternative implementation with the same remark.

This is followed by two functions with the same name -- the second one appears to be in use and in casual inspection does *not* appear to be constant time. [Overall the construct has at least six nested levels of #if/#endif conditional compiling logic, so understanding the intention or security claim is difficult.]

Similar remarks ("part of this implementation is not costant-time") are made for findRootsHFE_gf2nx() and chooseRootHFE_gf2nx() which are also in use in the optimized implementations.

In any case, I'll look more closely if there is a clear claim for a leakage-free implementation. Thanks!

Cheers,
- markku

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

From:	Carlos Aguilar Melchor <carlos.aguilar@isae-supaero.fr>
Sent:	Monday, June 7, 2021 4:17 AM
To:	pqc-forum
Cc:	pqc-comments
Subject:	ROUND 3 OFFICIAL COMMENT: HQC

Dear all,
we are happy to announce a new release for HQC which can be found here: https://pqc-hqc.org/implementation.html

This release features a change in the primitives used, as they are all now based on a KECCAK core with domain separation.

We also provide a C HLS-compatible implementation with two flavors: perfor-
mance oriented and compactness oriented. This implementation is translated by automatic tools into VHDL code that implements in hardware the full library with exactly the same outputs as our C reference implementation. The automatically generated VHDL is then post-processed to reduce the amount of KECCAK cores to two instances.

The results highlight that HQC is hardware friendly enough to have at the same time compacity, high throughput, and easy maintainability with an HLS implementation. Here are some figures for the performance oriented implementation on an Artix-7.

HQC L1 size (Keygen+Encaps+Decaps): 6.6k slices (20k LUTs, 16k FFs, 12.5 BRAMs, 0 DSPs)
HQC L1 cycles and times (at ~150Mhz):
Keygen 40k cycles, 0.27 ms
Encaps 89k cycles, 0.59 ms
Decaps 190k cycles, 1,2 ms

Best,

Carlos, for the HQC team.