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.