Stateful Hash-Based Signatures

Public Comments on Draft SP 800-208
(February 28, 2020 deadline)

Karsten Klein

From: Karsten Klein
Date: Wednesday, January 15,2020, at 3:19pm

Hi there,

Concerning - Draft NIST SP 800-208.

I just finished a first read (I've extracted some items for further follow up) and have a general and a minor comment:

Line 486 - With respect to how approved parameter sets are specified and footnote 3 in particular:
In general, an overview of all schemes (approved by NIST and existing in the referenced materials) with an outcome (approved, approved with restrictions, not approved, not in scope) and a reasoning (inefficient, ineffective, less secure due to...) could be used to avoid confusion of which parameter sets are approved and which are not. This would also allow to unify the naming scheme and map the parameter set naming used in the referenced RFCs (as it appears to be not homogeneous).
Eventually, this allows to omit the footnote. It really confused me, as it mixes scheme and parameter set level)

In short: please consider how not approved and approved parameter sets are represented to avoid confusion.

Line 502 - Numeric Identifier of XMSS-SHA2_20_256:
The Numeric Identifier for XMSS-SHA2_20_256 is 0x00000003 instead of 0x00000002; see RFC 8391 - Table 7.

Best regards,
Karsten Klein

AMD

From: Don Matthews
Date: Thursday, January 23, 2020 at 4:39pm

NIST SP 800-208 Draft Review Comments

General Comments

This is a summary of requested modification found by reviewers from AMD.

Commonality of Parameter Sets for Both Algorithms

LMS and XMSS are similar algorithms that as defined in their respective RFCs contain different parameter sets. We believe that there is some benefit to unifying the LMS and XMSS parameter sets as much as possible. The specific parameters are: W value, tree height, and hierarchical definition (HSS and XMSSMT).

- For the W value, LMS support 1, 2, 4, and 8 where XMSS only supports 16 (equivalent to LMS w=4). We would expect that most devices would use 1, 2, or 4 trading off key size for performance, while 8 would be used for interface constrained devices with a subsequent drop in performance.
- For tree height, LMS supports 5, 10, 15, 20, and 25 while XMSS supports 10, 16, and 20. XMSS^{MT} has support for a height 5 tree along with heights of 10 and 20.
- For the hierarchical versions, XMSS^{MT} has a complete parameter set with a variety of options but HSS has no parameter set associated with it and therefore, leaves the use up to an individual creator's definition.

We expected that NIST would use the two IETF standards and create similar parameter sets for both. As an example, having both LMS and XMSS support tree heights of 5, 10, and 20.

By creating a common set of parameters, NIST can allow the implementors to choose an algorithm based on their analysis of implementation rather than choosing an algorithm that has parameters that best fits their requirements.

It is understood that any changes to the parameter sets may create additional work over what would be required to meet the IETF standard. Although, most implementations should be written such that new parameter sets should work with existing code for algorithm implementation. An LMS implementation should allow for heights of 5, 10, and 20. An XMSS + XMSSMT algorithm also should allow for heights of 5, 10, and 20. Therefore, a common parameter set that allowed for heights of 5, 10, 20 for both LMS and XMSS would be possible.

Distributed Multi-Tree Hash-Based Signatures

We like this proposal that helps to alleviate concerns about system issues for solutions that require longevity of key signing capabilities.

Concerns

- Large number of cryptographic modules are required if each tree from level 0 and level 1 is a different module
    - The smallest LMS and XMSS^{MT} solution has a top-level tree of height 5
        - would require 33 cryptographic modules
    - The cryptographic module associated with the top level is a single point of failure.
        - Recommendations below make It possible for the signature system to still be functional even with the loss of the cryptographic module associated with the top level.

Recommendations

- Suggested changes to limit the number of cryptographic modules
    - Allow the top level to be based on LMS or XMSS with a height of 2, 3, or 4 and the lower levels to be based on 4, 8, or 16 (respectively) cryptographic.
    - Allow for the second layer to be implemented with LMS, HSS, XMSS, or XMSS^{MT} algorithms.
        - helps alleviate any concern about the total number of signatures that can be performed with a low height (height <= 5) top level tree.
    - Create a new system with associated parameters.
        - It would require parameters for the top level (LMS or XMSS with low height) plus parameters for the lower level (LMS, HSS, XMSS, XMSS^{MT}) and parameters for the OTS (LM-OTS, WOTS+).
        - Full parametrization of the complete system could get into long names.
- Suggested change to allow for loss of the cryptographic module associated with the top level
    - Have the signature of each lower level tree stored with the public key of the lower level tree.
        - Top level cryptographic module is only required for initial set up of sub level cryptographic modules allowing for the cryptographic module to be decommissioned.
    - Since the top-level module is only used at system initialization it prevents glitch attacks against the top level (as discussed in section 8).

FIPS Requirement

NIST has defined a process for algorithm validation (CAVP) and module validation (CMVP). SP 800-208 is defining an algorithm that should fall under CAVP but has a mandate that it only runs on a CMVP validated module.

In the past, FIPS has not posted a certification requirement for the solutions using NIST approved cryptographic algorithms. Many government contracts require FIPS certification, at different levels, but individual customers could determine if there was value in having a certified implementation. With the

FIPS certification requirements as specified in line 741-743 in this draft, it mandates FIPS testing on the processing module to be able to implement these algorithms. This leads to two different concerns

- It is not possible for a company to implement 800-208, even for internal uses, without getting FIPS validation on their cryptographic module(s) or purchasing a module from an outside company
- It will lead to confusion over algorithm names. Referencing any of the algorithms (LMS, HSS, XMSS, XMSS^{MT}) doesn't indicate if it is compliant with 800-208 since it may only be compliant with the RFCs
    - AES, and other algorithms, are defined by NIST and is always NIST compliant no matter where used for cryptographic operations
    - AES may or may not be CAVP tested

Some of the impetus for approval of Stateful Hash-Based Signatures was that companies may not be able to wait for the PQC algorithm selection process. Adding a FIPS 140 level 3 requirement for all implementations of 800-208 (see lines 741-743) could delay companies from using 800-208 as a solution. This requirement could be especially problematic for any company that has not been involved with a previous FIPS validation.

Specific items

Line 358 (Figure 1) – Figure 1 is representing both the hash chain but also the signature and verification operation simultaneously. It may be easier for some to understand if this was broken into two different figures. The new Figure 1 would consist of (using shorthand):

X -> HASH -> H(X) -> HASH -> H(H(X)) -> HASH -> H(H(H(X))) = pub

The new figure 2 would consist of (using shorthand):

X -> HASH -> H(X) = S S -> HASH -> H(S) -> HASH -> H(H(S)) = pub

|----------| |----------|

Signing Operation Verification Operation

Lines 449-450 (related to HSS) – "Shall be used for every LMS tree at that level" –implies that one can have an HSS signature design that utilized a different LMS parameter set at each level. The only requirement is that they use the same hash algorithm. Is your intention to allow for that type of design?

Line 491 (and others) – As discussed in the general comment previously, for XMSS, it may make sense to allow for other W values (1, 2, 4, 8, 32) like what has been provided by LMS. This would allow from performance/signature size tradeoffs and allow for similar configurations between LMS and XMSS.

LMS & HSS RNG requirements:

If a TRNG is available, should it not be possible to use the TRNG for all private keys? This capability is prevented in line 561-562 "shall be generated using the pseudorandom key generation method"

If TRNG is not allowed for private key generation (as currently written), then the content in parentheses should be removed from line 567 "and SEED (if using the pseudorandom key generation method)"

Line 577 – “generated using the pseudorandom key generation” – as with LMS comment above, if a TRNG is available, this specification prevents it from being used for private key generation.

Andreas Huelsing

From: Andreas Huelsing
Date: Wednesday, January 29 at 10:58am

Dear NIST team,

Thanks for your work. I highly appreciate the current draft of SP 800-208. I only have a few remarks.

a) As you do define a key generation mechanism, it might be worthwhile to define a forward-secure one (as for example in the original XMSS paper). XMSS and LMS with forward-secure key generation lead a forward-secure signature scheme. Forward-security can add a strong guarantee for old signatures in case of key-compromise and essentially comes for free in this setting.

b) I understand that there is no decision made about the NIST post-quantum standardization project. However, if NIST is even considering to keep SPHINCS+ it might be helpful to synchronize the addressing schemes of XMSS & SPHINCS+ as well as considering the tree-less WOTS-PK compression. This would allow to treat XMSS as a sub-step of SPHINCS+, requiring the same code-base. Especially, the code for XMSS signature verification would be almost a full sub-set of the SPHINCS+ verification code.

c) While I essentially do agree with your accessment of the security proofs there are a few nits:

   - Line 1459: Should say second-preimage resistance.

   - On the whole paragraph starting at 1457: Following the analysis in our recent publication "The SPHINCS+ Signature Framework" we additionally require h_k to be post-quantum, multi-function, multi-target decisional second preimage resistant. Alternatively, one needs a statistical assumption about h_k which does not hold for random functions (see the discussion about the tight security proof for SPHINCS+ on the PQC mailing list).

   - You could mention that LMS and XMSS are (non-tightly) secure in the standard model if we are willing to assume collision resistance of the used hash function. In this case, all the bitmasks and the prepended values can be arbitrary bit strings.

d) Regarding parameters: While I do think that limiting the choice of w and the used hash function is a good idea, I do not see any benefit in limiting the number of options for the total tree height and the number of layers. All implementations that I have seen are generic with regard to these values. This allows users to adapt the schemes to their constraints. What would be necessary in this case is defining upper bounds on both values.

Best wishes,

Andreas

Thales DIS

From: Aline Gouet
Date: Tuesday, February 18, 2020 at 8:59am

Hello

Please find below comments on NIST draft 800-208 as a contribution of Thales DIS.

Best regards
Aline

Comment 1: In sub-section 1.1, the statement from line 273 to 275 discourages the use of stateful HBS: "Stateful HBS schemes are only suitable for particular uses, as they require careful state management. The recommendations are summarized in section 1.2 and described in detail in [8]". We believe that stateful HBS schemes can be efficient, secure and useful when some implementation conditions are met. As the document itself is meant to be a general recommendation, we would suggest to rephrase the sentence in a more assertive manner, e.g. highlighting the importance of securely manage the state/counter in the implementation whatever or independently from the use-case.

Comment 2: In sub-section 1.1, lines 276 to 279, recommendation 2) "the implementation will have a long lifetime" seems to be different compared with and maybe contradict in some extend with initial answer from NIST on Gemalto comments: "we are keen to discourage the use of stateful hash-based signatures except in scenarios where signing is infrequent" (from https://csrc.nist.gov/CSRC/media/Projects/Stateful-Hash-Based-Signatures/documents/stateful-HBS-misuse-resistance-public-comments-April2019.pdf).
We believe that stateful HBS are suitable for long term, frequent usage, as long as the security recommendations are taken care of. Could you please clarify NIST's position on this point?
It would also make sense to add a fourth recommendation: "4) the implementation relies on hardware cryptographic modules, as described in section 8.1."

Comment 3: Section 3 on General Discussion describes mainly similarities of LMS and XMSS (in subsections 3.1, 3.2, 3.3) and only few differences between LMS and XMSS, i.e. mainly in subsection 3.4 on bitmasks and prefixes. It would be useful for the developer to describe in a similar way differences in final hashing of the Winternitz scheme and signature structure.

Comment 4: In section 3 on General discussion, there is no guidance on how to select one signature scheme or the other one based on different criteria, such as for example the total number of hashing (including the hashes used for bitmask generation) for comparable parameter sets.

Comment 5: In sections 4 and 5, the parameter sets of LMS and XMSS are described using original notation from RFC 8554 and RFC 8391. Since the naming for both schemes are not unified, that would be helpful to inform the reader in Section 3 and highlight some equivalences or differences in Notation. For example, it might be worth mentioning that p in LMS description

is equal to len parameter in XMSS. Another example is w that has different meanings in XMSS and in LMS, w in LMS corresponds to logarithm of w in XMSS.

Comment 6: The approved parameter sets for both LMS and XMSS are described in Section 4 and Section 5. For some parameter sets of XMSS, there are no equivalent parameters for LMS and vice versa. For example, there are no XMSS parameters for 32 signatures while it is possible for LMS. We believe that it is important to maintain the possibility to sign 32 messages which is suitable for implementation on constrained secured elements and it would be good to either provide similar parameter sets for both schemes or to explain the rationale of not having similar parameters for both schemes.

Comment 7: The four approved hash functions are defined in the beginning of Section 4 and Section 5. Since the most time consuming part of the signature is the OTS computation, it might be beneficial to have the possibility to use a function based on a block-cipher for this part, e.g. based on NIST SP 108 with PRF = CMAC-AES-256.

Comment 8: In Section 4, Tables 1, 3, 5 and 7, the parameter ls is indicated. Is there any reason for mentioning this parameter? We believe that it is used only for specific implementation described in original proposal, but it is not mandatory for different implementation, therefore it might be confusing placing it among the structure influencing parameters.

Comment 9: In Section 3, Figure 5, the symbol $\otimes$ is used for XOR operation. Maybe, it would be better to use classical symbol $\oplus$ instead.

Comment 10: In section 8, subsection 8.1, it is mentioned that "*The cryptographic module shall update the state of the private key in non-volatile storage before exporting a signature value or accepting another request to sign a message*". Could you please clarify whether this requirement also unable the possibility to use external memory to store the encrypted private key. (The private key would be encrypted by a key of cryptographic module.)

Comment 11: General comment: beyond the algorithm standardization, there is a need to address the need for a standardized key parameter encoding. This applies not only to state full HBS schemes, but any new HBS scheme in general. A general recommendation is that implementations should rely on standardized key encoding techniques, which should be referenced.

ETSI TC CYBER WG QSC

From: ETSI CyberSupport
Date: Wednesday, February 19, 3:39am

LIAISON STATEMENT

Title:	Responses to NIST's call for comments on Draft SP 800-208: Recommendation for Stateful Hash-Based Signature Schemes
Date:	
From (source):	TC CYBER WG QSC
Contact(s):	cybersupport@etsi.org
To:	NIST
Copy to:	
Response to: (if applicable)	NIST's call for comments on Draft SP 800-208
Attachments: (if applicable)	

TC CYBER WG QSC – Responses to NIST's call for comments on Draft SP 800-208: Recommendation for Stateful Hash-Based Signature Schemes

This document contains a non-exhaustive collection of comments from ETSI TC CYBER WG QSC on NIST's draft Special Publication 800-208: Recommendation for Stateful Hash-Based Signatures Schemes.

The draft specifies approved profiles for the LMS/HSS and XMSS/XMSS^MT stateful hash-based signature schemes. This means that it lists parameter sets for the schemes, but it relies on RFC 8391 and RFC 8554 for detailed descriptions of the algorithms. This is problematic for several reasons:

- Although LMS and XMSS are very similar, the two RFCs use different and sometimes conflicting notation. The NIST draft keeps the same notation as the RFCs, which will inevitably cause confusion for readers who are not already familiar with the schemes. Harmonising the notation is preferable, but not straightforward. One possible, but imperfect, solution would be to add a section that defines the mappings between notations. An alternative may be to consider producing two separate documents, one profiling LMS/HSS, and the other profiling XMSS/XMSS^MT.

- The RFCs were only intended to describe the schemes "with enough specificity to ensure interoperability between implementations". Neither RFC gives a full description of signature generation. Indeed, RFC 8391 provides example pseudocode for computing the authentication path for XMSS, but strongly recommends that a different method is used. Further, both the RFCs, as well as the draft SP, omit discussion on tree management strategies; RFC 8391 mentions it briefly, but general discussion is omitted. While algorithms such as the Buchmann-Dahmen-Schneider (BDS) algorithm are not required for interoperability, some mention of them may be beneficial for prospective implementors.

- There are some places where the RFCs are ambiguous. For example: when RFC 8554 refers to the LM-OTS or LMS public key it is not always clear whether it means the full public key including the typecodes and identifiers, or just the final hash values; RFC 8391 does not describe what should happen when idx_sig, which is incremented with each signature, exceeds the number of available one-time signatures.

Consequently, without further guidance it would be difficult for a non-expert to implement the signature schemes correctly and efficiently from the NIST draft and the RFCs.

More detailed comments follow:

Line 131: *"NIST would like feedback on whether there would be a benefit in reducing the number of parameter sets…"*

There are currently 80 LMS parameter sets, 12 XMSS parameter sets, and 32 XMSS^MT parameter sets. This seems excessive. Fewer choices of parameters generally increases interoperability of implementations, especially as there are now different choices of hash functions. In general, the choice of which parameter sets to eliminate and which to include is not straight-forward: parameter choices require different trade-offs, and those trade-offs may be compounded by other implementation choices, such as tree management strategies. However, certain parameter sets are impractical and can easily be eliminated. For example, RFC 8554 allows for up to 8 layers in an HSS hierarchy, and each layer can be of height at most 25, giving a maximum total tree height of 200. Time and compute resources for such a parameter set may not be readily available, and the benefits of using such large constructions are not clear. Conversely, it seems unlikely that the improved verification times are worth the increased signature sizes for the LMS parameters where $w = 1$ or $w = 2$. Therefore, we recommend NIST reduce the approved parameter sets to those that are practical or feasible to use.

There an issue of redundancy in parameter sets: there exist multiple parameter sets that offer the same signature size but require a varying number of hash function invocations. Such parameter sets could be pruned down to the most performant options while the rest are discarded, perhaps based on the number of signatures required, or for obtaining specific trade-offs. Unfortunately, no closed-form formula currently exists that would exclude non-optimal parameter sets.

Line 143: *"NIST would like feedback on whether there is a need to be able to create one-level XMSS or LMS keys in which the one-time keys are not all created or stored on same cryptographic module…"*

Resilience can already be provided by distributing a two-level HSS or XMSS^MT instance over different cryptographic modules. Distributing a single-level LMS or XMSS tree would

likely require more significant changes to the interfaces for key generation, but only saves the cost of an intermediate one-time signature.

Line 273: "*Stateful HBS schemes are not suitable for general use because they require careful state management that is often difficult to assure…"*

Another feature of HBS schemes that makes them less suitable for general use is that a given key pair can only sign a limited number of messages, and once that limit has been reached the long-term signing key is no longer useable.

Line 276: "*Instead, stateful HBS schemes are primarily intended for applications with the following characteristics…*"

It is also necessary to estimate the maximum number of messages that will need to be signed over the lifetime of the implementation, as this determines which parameter set should be used. This may be straightforward for some applications, but difficult for others; of course, it may be possible to be conservative and use a significant overestimate, but at the cost of reduced performance and increased signature sizes.

Further, there is also the notion of signature "loss" over the lifetime of the long-term key pair, depending on how state is managed. For example, an implementation may partition the state and advance it in distinct, non-overlapping blocks, accepting the risk that a system restart would lose the number of signatures in a single block. Over time, with large enough blocks, or with enough reboots, a significant portion of the total signatures may be lost.

It should be noted that the longer a hardware cryptographic module is in use, the greater the probability of device failure becomes. In such a case, existing signatures can still be verified, but no new signatures can be created under that same long-term key pair. As key back-up and recovery is restricted by the draft, the eventuality of no longer being able to generate signatures under a long-term key pair should be considered before deployment.

Footnote 2: *"HSS allows for up to eight levels of trees and XMSS^MT allows for up to 12 levels of trees."*

This restriction on the number of layers is important enough that it should be included in the main body of the text, as it could easily be missed. Implementors will select parameter sets from the tables within the SP, therefore parameter set restrictions should be explicit.

Line 427: *"…which uniquely identifies where a particular hash invocation occurs within the scheme."*

As per the comment below regarding Line 576, the addressing scheme used in RFC 8391 does not uniquely identify where every hash invocation occurs within the scheme.

Line 428: "*This address is then hashed along with a unique identifier for the long-term public key (SEED) to create the prefix.*"

There is an unhelpful (and potentially dangerous) conflict of notation between the use of SEED in XMSS, where it is a public identifier, and in LMS, where it is a private value used to derive the one-time private keys (see line 563).

Line 436: *Figure 5*

The diagram should use the symbol ⊕ to denote exclusive or instead of ⊗.

Line 438: *"This Special Publication approves the use of LMS and HSS…"*

The use of the word "and" implies that NIST approves stand-alone LMS implementations that are not themselves HSS with L=1. Section 6 of RFC 8554 states that "Since HSS with L=1 has very little overhead compared to LMS, all implementations MUST support HSS in order to maximize interoperability"; the somewhat ambiguous language "all implementations" is taken to mean "all implementations of LMS". NIST should make it explicit if they wish to allow non-HSS implementations of LMS. However, as *LMS* is often used interchangeably with *HSS* (which could lead to undue confusion) it is recommended that NIST only allow HSS, where single-layer LMS is explicitly HSS.

Line 444: "*… the hash function used for the LMS system **shall** be the same as the hash function used in the LM-OTS keys*."

RFC 8554 allows the use of different hash functions in LM-OTS and the LMS tree. If this restriction is intended to be enforced by verifiers, then Section 8.2 needs to mandate an explicit check of the typecodes in the public key, with the public key being rejected if they do not correspond to the same hash function.

Line 447: "*If the HSS instance has more than one level, then the hash function used for the tree at level 0 **shall** be used for every LMS tree at every other level.*"

As expressed in Section 6.1 of RFC 8554, the HSS public key only includes the typecodes for the LMS and LM-OTS signatures at level 0. The general HSS process described in RFC 8554 specifically allows the use of different parameter sets, and hence different hash functions, at different levels. If this restriction is to be enforced by verifiers, then Section 8.2 of the draft needs to mandate an explicit check of the typecodes contained in each signature; signatures are to be rejected if the typecodes do not correspond to the hash function specified in the HSS public key.

Because the long-term public key only includes the typecodes for the LMS and LM-OTS signatures at level 0, the signer could change the parameters used at other levels over time; that is, different signatures could use different parameters. Although Section 6 of RFC 8554 makes the explicit requirement *"…the signer MUST NOT change the parameter sets for a specific level"*, there is no way to detect or forbid this from the perspective of a verifier, without storing extra state. Therefore, complete parameter sets (for all levels) should also be included in, or derivable from the public key.

Considering the above comment regarding Line 438, if an LMS instance is defined as an HSS instance with L=1, and if parameter sets are validated, there may be additional difficulty with signature verification if using the distributed method described in Section 7.1 of the draft, as each distinct module will use "L=1", although the "virtual hierarchy" is larger.

Section 5.3 of RFC 8554 (LMS public key) does not set explicit requirements for the LMS public key format. The language used is *"the LMS public key can be represented as the byte string u32str(type) || u32str(otstype) || I || T[1]".* In addition to the comments given above, NIST could make the public key formats explicit requirements. Similarly, there is a lack of

requirements expressed for the LM-OTS or HSS public key formats.

There is also an unhelpful (and potentially dangerous) conflict of indexing conventions between HSS, where level 0 corresponds to the "root" tree used to compute the HSS public key, and XMSS^MT, where level 0 corresponds to the "leaf" trees used to sign messages.

Line 449: "*For each level, the same LMS and LM-OTS parameters set **shall** be used for every LMS tree at that level.*"

For clarity, it may be worth explicitly stating that different levels may use different LMS and LM-OTS parameters; e.g., they are allowed to have different tree heights. However, as mentioned above, the verifier cannot check whether this statement has been adhered to.

Line 452: "*The parameters n, w, ls, m, and h specified in the tables are defined in Sections 4.1 and 5.1 of [2].*"

There is an unhelpful (and potentially dangerous) conflict of notation between the use of w in XMSS, where the Winternitz chains have length w, and in LMS, where they have length 2^w. If the parameters are not explained, then there should at least be a warning that they represent different things for the two schemes. Similarly, there is a conflict of notation between the use of h in HSS, where it is the height of the trees in a single level, and in XMSS^MT, where it is the total height of the hypertree.

Line 459: *Table 1*

Although the signature lengths for LM-OTS are taken directly from RFC 8554, they are rather misleading when taken in isolation, as the one-time signature scheme will never be used by itself. It would be useful to have a separate table listing the public key and signature sizes for the different LMS parameters.

Line 516: "*For the parameter sets in this section, the functions F, H, H_msg, and PRF are defined as follows:*"

In RFC 8391, the SHA-256 and SHA-512 parameter sets pad the key so that it completely fills a SHA-2 message block for F, H and PRF, or two SHA-2 message blocks for H_{msg}. If the same approach is used for the truncated SHA-256/192 parameter sets, then the functions should be defined as:

$$\begin{aligned} F(KEY, M) &= T_{192}(\text{SHA-256}(\text{toByte}(0, 40) \mid\mid KEY \mid\mid M)) \\ H(KEY, M) &= T_{192}(\text{SHA-256}(\text{toByte}(1, 40) \mid\mid KEY \mid\mid M)) \\ H_{msg}(KEY, M) &= T_{192}(\text{SHA-256}(\text{toByte}(2, 56) \mid\mid KEY \mid\mid M)) \\ PRF(KEY, M) &= T_{192}(\text{SHA-256}(\text{toByte}(3, 40) \mid\mid KEY \mid\mid M)) \end{aligned}$$

In the current draft the text reads *"toByte(i, 4)"*, representing integer *i* only in 4 bytes.

Line 545: "*For the parameter sets in this section, the functions F, H, H_msg, and PRF are defined as follows:"*

In RFC 8391 it is explained that although a shorter identifier could be used with SHA3, n bytes are used for consistency with the SHA2 implementations. The draft appears to stick

with this convention in the case where $n = 32$, in lines 533 to 536, so it is recommended that the functions defined in lines 547 to 550 pad their identifiers to 24 bytes.

Line 566: "*If more than one LMS instance is being created (e.g., for an HSS instance), then a separate key pair identifier I, and SEED (if using the pseudorandom key generation method)* ***shall*** *be generated for each LMS instance.*"

The previous paragraph of Section 6.1 mandates the use of the pseudorandom key generation method.

Line 569: *"When generating a signature, the n-byte randomizer C (see Section 4.5 of [2])* ***shall*** *be generated…"*

The LM-OTS signatures are not deterministic because of the randomizer C. Therefore, if a leaf node on a higher level signs a root node on a lower level more than once, the resulting signatures will be different, which could allow an attacker to forge signatures. Section 6 of RFC 8554 implicitly addresses this issue by stating that "*It is expected that the above arrays are maintained for the course of the HSS key."* NIST should make storage of these arrays a requirement, or propose an alternative, deterministic, signing method.

Line 576: "*The private n-byte strings in the WOTS+ private keys ($sk[i]$ in Section 3.1.3 of [1])* ***shall*** *be generated using the pseudorandom key generation method specified in Section 3.1.7 of [1]:*"

There is a serious flaw in the pseudorandom key generation process described in RFC 8391 and mandated in the NIST draft. The private key value $sk_{i,j}$ for one-time signature instance j is derived from the private seed $seed_j$ via

$$sk_{i,j} = PRF\left(seed_j, toByte(i, 32)\right)$$

The private key index i acts as the address for the PRF, but this address does not depend on the index j of the one-time signature. Consequently, after observing r one-time signatures there is a multi-target attack that recovers a private seed with around $2^{8n+4}/r$ calls to the PRF. This reduces the classical security of XMSS and XMSS^MT with tree height h by around $h - 4$ bits.

Expanding on the above, suppose we observe a WOTS+ signature $S = (s_0, s_1, \dots, s_{l-1})$ on the message $M = (m_0, m_1, \dots, m_{l-1})$, where we implicitly include the checksum. For most values of i the message word m_i can be viewed as a uniformly random element of $\{0, 1, \dots, w-1\}$, with the obvious exception being the most significant word of the checksum. The probability that $m_i = 0$, and so the probability that s_i reveals the private value sk_i , will be $1/w$.

Now suppose that we observe r WOTS+ signatures $S_1, S_2, \dots, S_r$ on the messages $M_1, M_2, \dots, M_r$. For a fixed i we expect r/w of the messages to have $m_{i,j} = 0$, so we expect the r signatures to reveal r/w private values; that is, we expect there to be r/w values where $s_{i,j} = sk_{i,j}$. In general, we can choose the index i that reveals the most private values, which will be higher than r/w, but not significantly so.

Because the r/w private values all have the form $sk_{i,j} = H(seed_j || i)$ for private one-time seeds $seed_j$ and a fixed index i, we can try to guess a seed by choosing a putative n-byte value $seed'$, computing $sk' = H(seed' || i)$, and then comparing sk' with our r/w target values $sk_{i,j}$. The probability that our guess will match one of the targets is $r/w2^{8n}$, so we would expect to recover one of the seeds after $w2^{8n}/r$ guesses.

For XMSS, the Winternitz parameter is always chosen to be $w = 2^4$. Given a tree of height h, the maximum number of one-time signatures that can be observed is $r = 2^h$. Consequently, the attack requires 2^{8n+4-h} guesses.

A possible fix to this attack is to adopt the addressing method used for XMSS^MT in the NIST PQC Round 2 SPHINCS+ submission.

Line 586: *"Distributed Multi-Tree Hash-Based Signatures"*

The methods described in this section of the draft effectively describe "virtual hypertree" schemes, distributed across multiple hardware cryptographic modules, where no keying material is exported from any module. To use this approach in practice will require a significant amount of supporting software to facilitate communication between hardware modules, keep track of which trees belong to which device, prevent malicious re-routing of requests to inauthentic modules, and other operational requirements.

Consequently, such techniques will be difficult to deploy or use practically. With that in mind, NIST may want to consider relaxing the constraints on exporting private data. Below are some options NIST may consider that would allow for secure key backup and recovery:

- Backup and recovery should happen between two distinct machines that share the same code (e.g., both are HSMs).
- This communication should be supported by a KEM, where the shared secret is ephemeral and securely deleted after one use; this prevents redeployment.
- The state must be deleted from source machine after it has been exported to the other device. This prevents redeployment as well.

Line 620: "*Distributing the implementation of an XMSS^MT instance across multiple cryptographic modules requires each cryptographic module to implement slightly modified versions of the XMSS key and signature generation algorithms provided in [1].*"

Distributing HSS across multiple cryptographic modules is reasonably straightforward, as each intermediate signature is an independent instance of LMS. However, in XMSS^MT the intermediate signatures are instances of a reduced variant of XMSS, which are all implicitly viewed as being part of the same hypertree of total height h; e.g., the hash function addresses are given in terms of their locations in this hypertree.

The method of distributing XMSS^MT across multiple cryptographic modules suggested in Section 7.2 preserves interoperability with RFC 8391 by modifying the standard XMSS key generation and signing algorithms but is significantly more complicated to implement and use. Further, if the process for provisioning a bottom-level cryptographic module fails for some reason (see line 719) then this wastes a valuable signature from the top-level module.

A simpler approach would be to adapt the approach from HSS and use independent instances of (full) XMSS for the intermediate signatures. The disadvantages of doing this are that it would increase the length of the signatures, and the scheme would not be interoperable with XMSS^MT as specified by RFC 8391.

Given that NIST is allowing additional parameter sets and hash functions for both HSS and XMSS^MT, RFC-compliant implementations may not be able to verify all NIST-compliant signatures. This raises the question of how much interoperability should be preserved? NIST may want to break away from the RFCs entirely and set their own, distinct, requirements.

Line 641: *"7.2.1 Modified XMSS Key Generation and Signature Algorithms"*

The LMS and XMSS RFCs both contain explicit return statements in their pseudocode, which improves clarity, but the pseudocode in the NIST draft does not. This is particularly confusing in, for example, lines 708 and 712 where assignments are made to public key values using information returned from calls to `XMSS'_keygen`.

It may be worth stating explicitly that Algorithm 10' is a modified version of Algorithm 10 in RFC 8391; the same applies to Algorithm 12'. Similarly, it may be worth stating explicitly that `XMSS^MT external device keygen` replaces Algorithm 15, and that `XMSS^MT external device sign` replaces Algorithm 16.

There is a lack of clarity about where the structure `SigPK` lives in relation to the provisioned cryptographic modules, and whether it needs to be protected.

Line 647: "`Output: XMSS public key PK`"

There may be scope for confusion here, as in RFC 8391 the output of Algorithm 10 is the XMSS public key and the XMSS private key.

Line 651: "`wots_sk[i] = WOTS_genSK();`"

In RFC 8391, `WOTS_genSK()` (as described in Algorithm 3) sets each element of `wots_sk` to a uniformly random *n*-byte string, but the NIST draft mandates the use of the pseudorandom key generation method described in Section 3.1.7 of RFC 8391. This has the potential to cause confusion as the `WOTS_genSK()` function requires access to a uniformly random *n*-byte string `S` that should be stored as part of the private key.

Line 679: "`SK = L || t || idx || wots_sk || SK_PRF || root || SEED`"

No terminating semicolon. The same comment applies to lines 681, 683, 696, and 697.

This definition also conflicts with the use of "setter methods" in lines 657, 669, and 670.

Line 683: "`PK = OID || root || SEED`"

The format of the OID is not defined in RFC 8391, and it is not entirely clear how it relates to the identifiers in Section 5 of the NIST draft. There may be some confusion between the identifiers for XMSS and XMSS^MT as they appear to overlap.

Line 719: "`if ( getIdx(SigPK[t]) ≠ t ) {`"

This should be a `while` loop rather than an `if` statement. This process probably deserves more detailed explanation in the surrounding text.

Line 729: "*`// Send XMSS'_sign() command to one of the bottom-level key pairs`*"

In the example XMSS^MT signing algorithm described in RFC 8391, when one bottom-level key pair is exhausted a new key pair is generated automatically for the next signature. The method of external device operations presented in Section 7.2.2 suggest that the bottom-level cryptographic modules are provisioned first during key generation, and then one of the available modules is chosen for use during each signing call. In practice, there will likely need to be a mechanism for switching between modules and dynamically re-provisioning them when their key pairs have been exhausted.

Line 815: "*The faulted signature remains a valid signature, so checking that the signature verifies is insufficient to detect or prevent this attack.*"

The faulted signature is highly likely to be valid, but it depends where the fault occurs. If it is during one of the hash function calls that needs to be recomputed for verification, then the signature will not be valid.

Line 816: "*The only reliable way to prevent this attack is to compute each one-time signature once, cache the result, and output it whenever needed.*"

There are alternative mitigations. For example, one approach is to use redundancy: compute the full signature twice, compare the results and only release a signature if the results match; an attacker would need to induce two identical faults in order to obtain an exploitable signature.

Line 841: "*The randomized hashing process does not, however, impact the ability for a signer to create a generic collision since the signer, knowing the private key, could choose the random value to prepend to the message.*"

It is not entirely clear why this discussion is included, since, as pointed out on line 851, this should not really be considered an attack on the signature scheme. Randomised hashing is intended to prevent someone other than the signer preparing a pair of colliding messages; see, for example, the discussion in NIST SP 800-106. This is only a threat if the values r in RFC 8391 and C in RFC 8554 are not sufficiently random.

Line 844: "*The 196-bit hash functions in this recommendation…*"

They are 192-bit hash functions.

Line 898: "`union lmots_signature switch`"

The indenting of the `case` statements is inconsistent.

The same comment holds for `case` statements beginning on lines 947 and 982.

Line 1452: "*However, in the* current *version of XMSS^MT [1], the security analysis differs somewhat. In the standard model, [17] shows that XMSS^MT is EUF-CMA. Further, [16] shows that XMSS^MT is post-quantum existentially unforgeable under adaptive chosen message attacks with respect to the QROM.*"

Appendix C.4 somewhat overstates the provable security results for XMSS^MT. The standard model result by Malkin et al in [17] holds for a general signature framework which covers both XMSS^MT and HSS. It shows that hierarchical signature schemes are secure provided that the underlying one-time signature schemes are secure, but with a significant tightness gap.

The tight QROM proof by Hülsing et al from [16] does not apply to XMSS^MT as described in [1]. Firstly, the result from [16] requires an assumption about the hash function family F that is almost certainly not satisfied by any NIST approved cryptographic hash function; a recent paper presented by Bernstein and Hülsing at ASIACRYPT 2019 replaces this with a brand-new security notion which they call *(multi-target) decisional second-preimage resistance* and which they believe should be difficult to attack. Secondly, the scheme analysed in [16] differs from the version of XMSS^MT described in [1] in a few important details; for example, the method for generating one-time private keys in [16] involves the address of the one-time signature, which prevents the attack described above.

Line 1469: "*The main difference between these schemes' security analyses comes down to the use (and the degree of use) of the random oracle model or quantum random oracle models.*"

It is also arguable that the complexity of the security reduction and the number of assumptions involved are also important. A simpler argument gives more confidence in the correctness of the result.

NSA's Center for Cybersecurity Standards

From: Sharon Ehlers
Date: Monday, February 24, 2020, at 1:13pm

Comments for SP 800-208.

- The option of using SHA384 or SHA512 could be useful.

- The parameter sets for LMS and XMSS use similar but different notation and this could cause some confusion. For example, w has two different meanings between the two schemes and SEED is a private value in LMS and a public value in XMSS. Consider making these differences clear.

- Section 7.1, page 20 line 618: Unable to find an Algorithm 9 in [2].

- Sections 7.2.1 and 7.2.2:
    - Calls to XMSS' sign
      need to know to which module it's being sent so layer/tree can to be tracked in the external device keygen and external device sign.

    - Lines 716-723: It is not clear what the purpose of this if statement is. Please Clarify.

- Line 732: The definition of t is misleading. In the RFC, it is
  h-(h/d) most significant bits of idx_sig. Here, since d=2, t=h/d most significant bits is correct, but using t=h-(h/d) or t=h/2 most significant bits would be clearer. Furthermore, the definition from the RFC, t=h-(h/d) most significant bits of idx_sig, is misleading as well.
  If idx_sig has exactly h bits, this is fine, but idx_sig has ceil(h/8) bytes, which is not always h bits. In that case, the definition of t might not be grabbing the intended bits of idx_sig. This definition comes up in the XMSS^MT sign and verify algorithms.

- p26: 196's should be 192's

Crypto4A

From: Jim Goodman
Date: Monday, February 24, 2020 at 3:25pm

Crypto4A's Comments on NIST SP800-208 Draft Specification

Crypto4A's comments are provided in two distinct parts: first we provide editorial comments regarding the draft's proposed language, and then we provide comments regarding the concepts being proposed within the draft itself.

Editorial Comments

First, our editorial comments:

- **Line 266:** replace "some but not all of" with "some, but not all, of"
- **Line 268:** consider adding references for SHA-256 and SHAKE256 (i.e., [3] and [5] respectively)
- **Line 280:** change "is firmware" to "is authenticating firmware"
- **Line 342:** consider changing "public keys." to "public keys using a Merkle tree construction."
- **Line 348:** consider deleting ", as follows"
- **Line 358:** consider changing figure title to "A sample Winternitz chain for b = 4"
- **Line 376:** fix formatting to avoid CRLF's in H**i(x_j) elements in the figure
- **Line 385-386:** consider changing "value, which will" to "value at the root of the tree, which will"
- **Line 389:** consider changing "public keys." to "public keys (k_i, $i \in [0, 7]$)."
- **Line 390:** consider changing "the tree." to "the tree (h_j, $j \in [0, 7]$)."
- **Line 391:** consider changing "the tree." to "the tree (i.e., h_{01}, h_{23}, h_{45}, and h_{67})."
- **Line 419:** change "different values" to "different prefix values"
- **Line 436:** the symbol for XORing $\mathbf{x_k}$ and the bitmask looks an awful lot like some form of multiplication, perhaps there's a more "XOR-like" symbol that could be used instead?
- **Line 489:** change "functions is specified" to "functions are specified"
- **Line 502:** change XMSS-SHA2_20_256 entry's Numeric Identifier from "0x00000002" to "0x00000003"
- **Line 518:** change "toByte(0, 4)" to "toByte(0, 24)" (or perhaps you'd prefer to stay with 32?)
- **Line 519:** change "toByte(1, 4)" to "toByte(1, 24)" (or perhaps you'd prefer to stay with 32?)
- **Line 520:** change "toByte(2, 4)" to "toByte(2, 24)" (or perhaps you'd prefer to stay with 32?)
- **Line 521:** change "toByte(3, 4)" to "toByte(3, 24)" (or perhaps you'd prefer to stay with 32?)
- **Line 547:** change "toByte(0, 4)" to "toByte(0, 24)" (or perhaps you'd prefer to stay with 32?)

- **Line 548:** change “toByte(1, 4)” to “toByte(1, 24)” (or perhaps you’d prefer to stay with 32?)
- **Line 549:** change “toByte(2, 4)” to “toByte(2, 24)” (or perhaps you’d prefer to stay with 32?)
- **Line 550:** change “toByte(3, 4)” to “toByte(3, 24)” (or perhaps you’d prefer to stay with 32?)
- **Line 587:** consider changing “of time and” to “of time, and”
- **Line 683:** consider adding additional line after 683 that states “return (PK)”
- **Line 685:** consider changing “Message M” to “Message M, XMSS private key SK”
- **Line 686:** consider changing “signature Sig” to “Updated SK, XMSS signature Sig”
- **Line 703:** consider adding additional line after 703 that states “return (SK || Sig)”
- **Line 907:** consider adding additional space at start of line for proper alignment
- **Line 915:** consider adding additional space at start of line for proper alignment
- **Line 952:** consider adding additional space at start of line for proper alignment
- **Line 958:** consider adding additional space at start of line for proper alignment
- **Line 961:** consider adding additional space at start of line for proper alignment
- **Line 964:** consider adding additional space at start of line for proper alignment
- **Line 995:** consider adding additional space at start of line for proper alignment
- **Line 1279:** consider adding two additional spaces at start of line for proper alignment

Qualitative Comments Regarding Concepts

In addition to the aforementioned editorial comments, we have identified several primary concerns with the document, as well as just some general comments regarding various sections of the document:

- There is no disaster recovery (DR) option given the manner NIST is proposing to generate HBS private keys, and the restrictions you’re imposing in Section 8.1. On line 745 you clearly state that the cryptographic module **shall not** allow for the export of private keying material. While we don’t expect NIST to have to provide guidance on DR, we also don’t believe it should be explicitly precluding options by putting this sort of restriction on the cloning/exporting of HBS private keys. Yes, state management is difficult to do, but processes can be put in place to manage the activity (more on this later), and the benefits of being able to archive keys to avoid having the entire hierarchy come crashing down if the top level HSM were to fail. Your proposed solution attempts to mitigate this by distributing the private key generation across multiple devices such that the top level HSM signs public keys presented by other HSMs (more on this in a later comment) which have generated private keys for lower layers of the hierarchy. This approach is still dependent on the top level HSM being present and operational so that it can sign new public keys as they come online, which could be difficult for a long-lived keying hierarchy. One way to overcome that is to have all of the subordinate HSMs present and accounted for soon after the top level HSM has generated its HBS private key, so that they can all request their public keys get signed before the top level HSM fails. Unfortunately, you’re just moving the problem around as now those subordinate HSMs need to survive long enough

to carry out their roles as HBS signing authorities, and the amount of capital expenditure to finance the bulk purchase of HSM devices may prove prohibitive. Hence, we think it would be best for NIST to **not** preclude exporting private key materials, but rather focus on devising best practices related to managing the risks associated with that operation, so that operators can devise their own DR solutions.

- Over the past 25 years of handling DR principles around critical PKI root keys, we have evolved very strong procedures for the secure extraction and re-injection of critical root key material in HSMs. This has provided us with a high guarantee of having preserved the integrity, confidentiality and availability of the keys by enforcing the tracking of private key material whether it's within an HSM or some form of secure external storage such as a safe or vault. This was possible as the RSA/ECC keys were complete objects with no additional state that needed to be maintained. Unfortunately, HBS introduces state to the management equation so attempting to distribute HBS private key material across multiple HSMs is tantamount to scattering the private key in both space and time. Hence, the proposed multi-HSM approach for implementing a distributed multitree HBS (Section 7) is concerning to us from a security perspective in its current form. What guarantees does the top level HSM have regarding the validity of the signing request it receives from parties looking to have the public key of the HSS private key they've generated on their HSM devices? Mechanically anyone could present a public key for signing, thereby introducing the possibility of rogue parties now being able to generate valid signatures. In a PKI CA world, they would manage this with revocation to punish the bad actors who managed to fool the CA into signing their illegitimate certificate. In the proposed 2-level HBS scheme there are no such revocation methods to save us after the fact, so we need to do everything we can to prevent this situation from happening. Hence, there needs to be some robust mechanism in place to validate requests BEFORE they are signed, which we have found to be a very difficult problem to solve unless very rigid procedures are put in place to eliminate the possibility (e.g., force the subordinate HSM to be brought into the room where the root HSM is so that the root HSM operators can witness the HSS key generation process and perform some sort of attestation that the HBS public key the subordinate HSM generates corresponds to a private key generated on that subordinate HSM). This is likely to prove to be a very onerous process akin to a full-on traditional root key generation ceremony in a conventional PKI, so this needs to be considered and addressed somehow (e.g., guidance on procedures, introduction of requirements to guarantee attestation of the authenticity of the signing request, etc.).

- The existing hash-sigs github repository that provides a reference implementation for LMS-HSS includes functions to pseudo-randomly generate LMS subtree {I, SEED} values from a master seed value for a given LMS-HSS instance (i.e., hss_generate_root_seed_I_value() and hss_generate_child_seed_I_value() in hss.c) , which allows the implementor to optimize the private key data storage requirements by eliminating the need to store discrete pairs of {I, SEED} for each layer of the tree since we can just recompute them from a single master seed value. This method of pseudo-random value generation for I in particular was identified as an option in RFC 8554 Section 7.1, so we don't believe it represents a security compromise of any proposed solution. Lines 566-568 of Section 6.1 appears to preclude this sort of implementation option by forcing the implementer to generate a separate {I, SEED} pair for each LMS instance. However, this requirement is itself quite vague as you put no requirements on how those values are generated (i.e., can they be pseudo-random or do we need to generate using a random bit

generator that supports at least 8n bits of security strength)? We would prefer to be able to continue using a pseudo-random method, but if that isn't acceptable then perhaps the language of the requirement can be made more precise to remove the aforementioned ambiguity.

- Section 6.1 also enforces the requirement that the same SEED value shall be used to generate every private element in a single LMS instance (line 563). We feel this is overly restrictive, and an implementor should be able to use one or more values/SEEDs provided they are generated in a manner that meets the stated security criteria (i.e., using an approved random bit generator where the instantiation of the random bit generator supports at least 8n bits of security strength). Relaxing this constraint opens up the possibility of proposing novel DR-compatible solutions, one of which we describe below.

- Would NIST consider a mechanism whereby the top-level LMS instance (we're applying things to LMS-HSS in the interest of simplicity, but the comments should extend to XMSS/XMSS^{MT} as well) is sectorized into cryptographically-isolated segments, each of which shares the same I value but which has its own SEED value that was generated using a manner similar to the pseudo-random generation of LMS-OTS private keys (but using a unique format to ensure it doesn't collide with that pseudo-random process, or any of the processes used in hash-sigs to generate {I, SEED} pairs, and which can't be used to guess another sector's SEED value). Sectorization would segment the 2^h leaves of the top-level tree into 2^s groups (a.k.a., sectors), each containing 2^{h-s} leaves. Each sector's SEED value allows a device to generate signatures from that sector's set of leaves and NOT any other sectors' leaves. Hence, you have cryptographically-enforced state reuse protection if you assign different sectors to different cryptographic modules (i.e., HSM_i can't generate valid signatures from the sector assigned to HSM_j). However, the sector generation process can ensure that all sectors share the top-level public key value, so all sectors are part of the same HBS signing authority. These sectors can then safely be exported from the top-level HSM and stored in a secure fashion using the same techniques and procedures that have been proven over the years to handle the secure extraction and handling of any regular private keys so that they can be loaded onto other HSMs (once and only once) when needed (e.g., the existing HSM(s) fail and we need to recover the signing capability for the given HBS public key, we use up all of the existing allocated sectors' signatures and need to load new sectors into the HSM many years down the road, or we want to load unique sectors into multiple HSMs in parallel to allow higher signing throughput). We believe this will yield a feasible means of providing DR for HBS on HSMs (albeit with potential over-allocation of the total tree size in order to accommodate the redundancies that facilitate DR). Note that this approach can be used to create a one-layer tree with OTS keys being created and stored on different HSMs as per the request made in the paragraph on lines 143-146 within the Note to Reviewers section. In that use case, each sector would be loaded into a different HSM, where the resulting unique SEED values would facilitate the generation of unique OTS keys on each device.

- An additional note on revocation as per the proposed 2-level scheme described in Section 7. Our interpretation is that the subordinate cryptographic modules are generating a single certificate that verifies back to the primary cryptographic module's top-level public key. In a typical PKI the root CA would sign a subordinate CA's public key, generating a certificate for that subordinate CA public key that the user/application could validate. In the proposed approach we'd have the root CA (i.e., top-level CM) sign the subordinate CA's (i.e., subordinate

CM) public key, but that result would just appear as part of any HSS/XMSS^{MT} signature the subordinate CA generates (i.e., the first LMS/XMSS signature component that precedes the subordinate CA's pubic key element, and LMS/XMSS signature on the message). Hence there is no discrete certificate that could be checked and revoked. Furthermore, if another subordinate CA has been stood up, and it hasn't been compromised, then will it be affected as a consequence of revoking the other subordinate CA given it shares the same root CA public key as all other subordinate CA's in this stratified approach, and we don't have a discrete top-level certificate to use to achieve finer-grained revocation. We've kicked around ideas related to atypical revocation mechanisms based on longest prefix-matching against portions of the HBS and its components, but these are all custom hacks that don't lend themselves well to a standardization effort. How does NIST envision revocation working with the proposed 2-level scheme? Is it an all-or-nothing sort of thing?

- A general comment regarding Figure 4, and the differences between HSS and XMSS^{MT}: Figure 4 shows the top-level tree being marked as level 0, with the level value increasing as we progress form top-to-bottom of the multi-level tree. This approach is fine for HSS, where a similar numbering convention is used, but in XMSS^{MT} we believe the standard numbers the top-level tree as level (d-1) and proceeds to decrease the level value as we progress from top-to-bottom. This may lead to confusion later on, and we think the difference merits some form of mention in the text.

- The description/pseudocode for XMSS^{MT} external device key generation is confusing to us. Under what conditions would the IF statement in line 719 evaluate to true given the generation calls on Lines 712 and 715, thereby necessitating us to essentially repeat the generation calls using lines 721 and 722 respectively? Would the given code not just adjust the incorrect t value by at most 1 given the correction is not iterative, but just a one-off? This confusion is somewhat compounded by what seems to us to be under-specified inputs/outputs for Algorithms 10' and 12' in section 7.2.1. which are used extensively in Section 7.2.2.

- In Appendix A and Appendix B, the text indicates we're extending the XDR syntax for [2] and [1] respectively, but the subsequent descriptions in Lines 859-1002 and Lines 1007-1391 read like they are the entire XDR specifications. Would it make sense to add comments into the XDR elements to remind the reader that you're supposed to also include all existing XDR specification code into each definition? For example, for LMS-OTS algorithm type (lmots_algorithm_type), add a new line between Lines 861 and 862 that says something along the lines of "`/* includes all existing lmots_algorithm_type values */`" or some similar language to remind the reader that existing definitions are retained as well.

Marc Stöttinger

From: Marc Stöttinger
Date: Friday, February 28, 2020, at 3:43am

Dear Author team of the document SP800-208,

as consortium members of the German nationally funded research project “QuantumRISC”, we would like to provide you feedback on the draft NIST Special Publication 800-208 (SP 800-208).

The QuantumRISC project is funded by the German Federal Ministry of Education and Research (BMBF) and brings together partners from both academia and industry. The project partners jointly develop and improve post-quantum secure cryptographic schemes for low-end devices with severe limitations on memory usage and power consumption while maintaining a high level of security. The practical implementation of such schemes highly depends on their operability on embedded devices. The main focus of the project is the development of quantum secure solutions for the automotive domain; however, research findings will be transferable to other domains and use cases. We investigate the interaction between existing vehicle systems and architectures as well as the integration of PQC into the vehicle while allowing a future exchange of cryptographic primitives (crypto agility).

The project consortium consists of the following partners: Continental AG, Elektrobit Automotive GmbH, Fraunhofer Institute for Secure Information Technology SIT, RheinMain University of Applied Sciences, MTG AG, Ruhr-University Bochum and Technical University of Darmstadt.

We have the following three feedback comments to the current draft version:

1) Past experience has shown that developers find it difficult to deploy cryptography if the specifications are distributed among different standards or if ambiguous representations exist (e.g. RSA parameters with explicit NULL or empty). In order to improve interoperability and to be able to use algorithms between different applications, object identifiers and standardized representations of public keys are necessary. Therefore, object identifiers (OID) should be specified for the two algorithms XMSS and LMS and for the signatures and public keys. Public keys should be uniquely represented in ASN.1 to make it possible to issue interoperable certificates that contain public XMSS or LMS keys.

   For example, a public key could be represented as:

   SubjectPublicKeyInfo ::= SEQUENCE {

   algorithm AlgorithmIdentifier,

   subjectPublicKey BIT STRING }

The 'algorithm' field could specify an OID and an explicit statement regarding the parameters and the 'subjectPublicKey' field could provide a concrete specification of the encoded public key (e.g., a 1-to-1 mapping to the specifications of the RFC). With SP 800-208, there is a chance to specify OIDs and representations in one document to facilitate the use of XMSS and LMS.

What is the reason that XMSS and LMS variants are not harmonized to provide parameter sets with the same tree heights? Different usage scenarios have different requirements and more flexibility for the maximum number of signatures should be provided. Hence, we would like to see similar parameter sets for XMSS and LMS with respect to the tree heights and ideally with a smaller step size in the tree height in order to choose a number of 2^5, 2^8, 2^10, 2^15, 2^16, 2^20, 2^25, 2^32, 2^40, ... signatures. Alternatively, the tree height could not be specified in the parameter set but freely chosen (in a certain range) for each key pair.

SP 800-208 references RFC 8391, which also provides a description of the XMSS algorithm. Alongside the RFC document, there is also a C reference implementation of XMSS. We note that each of these documents provides different algorithm definitions. For example, algorithm 10 in SP 800-208 and algorithm 10 in RFC 8391 both specify the XMSS key generation; yet they provide different implementations. Though the algorithms are semantically identical, a uniformly standardized basis of algorithms would likely prevent misunderstandings and implementation flaws. Similarly, the implementation of algorithms in the C reference implementation does not follow the pseudocode from RFC 8391. For example, algorithm 2 (WOTS Chaining) is defined recursively in the RFC but implemented iteratively in the reference code. Having a unified definition of algorithms throughout the provided documents would presumably ease understanding and implementation.

Best regards,

Marc Stöttinger

Stefan-Lukas Gazdag

From: Stefan-Lukas Gazdag
Date: Friday, February 28, 2020, at 12:50pm

Hi,

thanks to NIST for all the great work regarding the PQC standardization process! Please find enclosed some comments on draft SP 800-208.

We (genua GmbH) provide hybrid signatures (ECDSA and XMSS) for our latest software updates. Both signatures have to be verified as valid, otherwise the update is rejected. Key generation and signing is done on a secure key server. Authorized build servers in a restricted development network may ask for a signature via an OpenSSH connection. First updates have been applied to machines in the field. We look forward to HBS being used more widely by others.

Open topic: OIDs
For the use in practice (explicitly taking a look at X.509 certificates) object identifiers (OIDs) are needed. This far there are no OIDs defined by any organization (neither by any agency, corporation, university or the IETF/IRTF). Without going into details about former discussions on who should publish OIDs I just want to raise awareness that this should be dealt with. Software using HBS so far uses "temporary" or private OIDs (that have somewhat been agreed on between some software projects) or use software specific identifiers.

Line 273-275:
Yet another peculiarity is that you should choose a proper parameter set suiting your specific use case (e. g. which signature size is still ok, while maintaining a specific security level). This also means how many signatures will be written as the key has a limited life-time. Whereas classical keys have an implicit life-time (forced by a validity date or due to the need of increasing the security level due to advances in supercomputing, cryptanalysis, …), for HBS maybe a small key writing e.g. a million keys would be enough (or may be exchanged in time) for a specific use case while other scenarios would require a huge multi-level tree. All in all decisions that have to be made beforehand in a different way than with classical schemes.

Line 278/279:
I'd argue that using HBS now is important in many other, probably most use cases of software updates and code signing. History shows that software runs for way longer in the field than often expected as users stick to their running systems. Thus old systems are likely to be found running pre-quantum update mechanisms once a large enough quantum computer exists. Therefore it is recommendable to apply HBS now to existing systems even it is "just" to ensure a proper transition to other quantum-safe signature schemes later on. Also implementing and distributing update mechanisms using hybrid signatures now might help having somewhat modular mechanisms where exchanging a single scheme might be easier.

Line 436:
Please use the ⊕ symbol for exclusive-or

Line 502:
The correct numeric identifier of XMSS-SHA2_20_256 is 0x00000003

Line 587:
Not the most sophisticated solution but practicable: as the public keys for all the schemes are quite small, a specific device or software might be provided with several HBS public keys.

Ling 641 and following:
Some pseudo-code lines are missing semicolons. Also, sometimes setter / getter methods are used as in the RFC but sometimes they are omitted

Line 647:
Algorithm 10' / XMSS'_keyGen should also output the secret key SK

Line 774 and following:
In some uses cases performance might improve by the reservation approach described in [8], which we've tried in practice. Reserving an interval of OTS keys, meaning writing an updated secret key according to the interval chosen to non-volatile memory before signing alleviates performance issues in practice. In case of any interrupt, some OTS keys stay unused, which in most scenarios should not be a problem with somewhat stable cryptographic modules / key servers.

Line 844:
s/196/192/

Kind Regards,
Stefan-Lukas Gazdag

Canadian Centre for Cyber Security

From: David E. Smith
Date: Friday, February 28, 2020, at 3:58pm

Please find below our editorial and technical comments on the Draft SP 800-208 issued for comment in December 2019.

David Smith
Canadian Centre for Cyber Security

Line	Type	Comment
288, 775, 809	Technical	Starting at line 288, "If an attacker were able to obtain digital signatures for two different messages created using the same OTS key, then it would become computationally feasible for that attacker to forge signatures on arbitrary messages". Similarly, starting at line 775 and line 809 "...this is acceptable since it just involves using an OTS key multiple times to sign the same message". Comment: Per Section 6.1, 9.3 and [2], it seems that in LMS the OTS generates a random prefix for every message to be signed (Algorithm 3 in Section 4.5 of [2]). In particular, a forgery would be possible given two distinct signatures even if they were for the same message. Also, it would not be acceptable to use an OTS key multiple times, even for the same message, unless the random prefix was forced to be the same. XMSS also generates a random prefix before signing, but it appears to be deterministically derived from the private key and signature index (Algorithm 12 of Section 4.1.9 of [1]), so signing the same message with the same OTS would result in the same signature.
368	Editorial	Replace "checksum is computed as sum_{k=0, n-1}(b-1-N_k)" with "checksum is computed as sum_{k=0, n-1}(b-1-N_k), which requires ceil(log_b(n*(b-1))) digits".
389	Editorial	Replace "Figure 3 depicts a hash tree containing eight OTS public keys." with "Figure 3 depicts a hash tree containing eight OTS public keys k_0, ..., k_7".
506	Editorial	Replace SHA2 with either SHA-256 (to match earlier in the draft) or SHA2-256 (to match [1]).

Panos Kampanakis

From: Panos Kampanakis
Date: Friday, February 28, 2020 at 4:49pm

Dear Quynh, NIST,

I would like to provide some more feedback regarding the SP 800-208 Draft for HBS after discussing with some of our HSM peers implementing HBS. They pointed out to us some practical concerns:

1) Section 8.1 mandates that private keys should not be extractable. Today HSMs allow for extracting a classical private key using some Shamir sharing scheme so that key can be reconstructed and reused in case of an HSM failure. I don't think LMS is different. In a hierarchical scenario where a top level HSM signs subordinate LMS trees, the top HSM would need to survive for a long time (30 years for a traditional CA root) in order to be able to sign any new subordinate tree coming online. That may not always be practical. We should allow for the OTS private keys to be extractable using similar methods (Shamir secret sharing or so) so someone could reconstruct the top HBS tree and sign new messages in case of failure.

2) Section 6.1 requires a separate I and SEED value for each LMS instance. If someone wanted to generate I with a PRF he should be able to, so that the subtrees of a hypertree can be generated by using a master value instead of storing separate (I, SEED) pairs for each tree in the hypertree. Generating I in a deterministic pseudorandom could point to SP800-90A.

3) Section 6.1 requires one SEED per LMS tree. By allowing more SEED values, HSMs can use them to be able to generate non-overlapping sections of the tree in order to prevent state reuse in a DR scenario. Using different SEEDs in some of the LM-OTS leaves does not compromise the security of LMS tree.

Panos
Cisco

From: Panos Kampanakis
Date: Friday, February 28, 2020 at 10:26pm

We would also like to propose for the SP to include the following parameters that are suitable for all our (Cisco and probably many more vendor) image signing usecases

~~~~~~~~~~

- LMS_SHA256_M16_H5 with LMOTS_SHA256_N16_W8
- LMS_SHA256_M24_H5 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H5 with LMOTS_SHA256_N32_W8

- LMS_SHA256_M16_H10 with LMOTS_SHA256_N16_W8
- LMS_SHA256_M24_H10 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H10 with LMOTS_SHA256_N32_W8

- LMS_SHA256_M16_H15 with LMOTS_SHA256_N16_W8
- LMS_SHA256_M24_H15 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H15 with LMOTS_SHA256_N32_W8

- LMS_SHA256_M16_H20 with LMOTS_SHA256_N16_W8
- LMS_SHA256_M24_H20 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H20 with LMOTS_SHA256_N32_W8

- HSS (with 2-4 levels) with any of the above LMS trees at any level.

~~~~~~~~~~

For N=M=16 we realize that that would provide 64-bit PQ security, but given NIST's stance with AES-128 (Grover not being parallelizable and thus AES-128 is considered secure) we could use it when needing very small signatures at acceptable security.

Thank you,
Panos
Cisco Systems

Google

From: Stefan Kölbl, Roy D'Souza
Date: Friday, February 28, 2020, at 6:18pm

Google's Comments on the NIST SP800-208 Draft Specification

Stefan Kölbl, Roy D'Souza
February 28, 2020

Google anticipates deployment of post-quantum hash-based signature schemes for verified boot, and over-the-air updates, for a range of hardware modules. These modules vary significantly in available power, computational capabilities and related resources.

When deciding between stateless and stateful schemes, for scenarios that are amenable to the larger signature sizes of stateless schemes we would leverage a NIST-recommended scheme, such as the anticipated SPHINCS+. Whereas for other contexts, where it is an imperative to limit signature sizes, we would deploy a NIST-recommended stateful scheme such as LMS/HSS.

Deployment Scenarios

The following three deployment scenarios would most likely be constrained to usage of a stateful scheme:

- Google Security Chips: All Chromebooks are deployed with an embedded Google Security Chip that is candidate for being a quantum-ready hardware root of trust. It would probably have computational abilities similar to an ARM Cortex M3, with limited memory and flash.

- Battery Operated IoT Sensors: These include sensor devices such as Nest Detect, the motion and perimeter sensors used by the Nest Guard secure alarm system. This class of devices has the resource constraints of the previous category, and also needs to operate on the equivalent of an AAA battery for over two years.

- Powered IoT Devices and Chromebooks: These are powered devices based on Intel/AMD and ARM chips, and these lower cost devices have space and other resource constraints that would benefit from compact signatures.

Our choice of stateful hash-based standardization candidates is LMS/HSS, and the following two categories of parameters would be important for addressing the resource constraints of the scenarios outlined above.

Variable (Sub-)Trees

It would be beneficial to have different parameters depending on the level of a multi-tree. The cryptographic modules at a lower level might be deployed in more constrained environments,

while a higher-level tree, perhaps belonging to a more trustworthy third party, could afford more expensive computations.

The cadence of firmware updates to devices, even within each category, could differ significantly. A Chromebook might be updated every six weeks, while some IoT devices might only be updated occasionally. Therefore it would be useful to have a choice of parameters for LMS/HSS:

- LMS_SHA256_M24_H5 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H5 with LMOTS_SHA256_N32_W8

- LMS_SHA256_M24_H10 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H10 with LMOTS_SHA256_N32_W8

- LMS_SHA256_M24_H15 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H15 with LMOTS_SHA256_N32_W8

- LMS_SHA256_M24_H20 with LMOTS_SHA256_N24_W8
- LMS_SHA256_M32_H20 with LMOTS_SHA256_N32_W8

- HSS (with 2-4 levels) with any of the above LMS trees at any level.

Security Targets

In the ongoing NIST post-quantum cryptography standardization process five security levels have been defined and the proposed schemes seem to fall into NIST security level 3 and 5, as they do not rely on the collision resistance of the underlying hash function.

In some of our scenarios it might be useful to have variants of LMS/XMSS that target NIST security level 1, as this would provide security comparable to ECDSA with P-256 or Ed25519, while still providing a buffer against quantum adversaries given the limitations of Grover's algorithm (e.g., limited parallelization or that the quantum circuit of the hash functions will be fairly large). Introducing new variants with n = 16 would reduce the signature size for the OTS by over 50%:

- LMOTS_SHA256_N16_W1: 2196 bytes
- LMOTS_SHA256_N16_W2: 1108 bytes
- LMOTS_SHA256_N16_W4: 580 bytes
- LMOTS_SHA256_N16_W8: 308 bytes