includes exporting private keying material in encrypted form, even if no copies of the decryption key are available outside of the cryptographic module.

For most cryptographic algorithms storing private keying material outside of the cryptographic module in encrypted form would not pose a security vulnerability. However, in a stateful HBS scheme allowing private keying material to be stored outside of the module could lead to problems with state management that could result in one-time key reuse. For example, when the key is loaded into the cryptographic module for use, the module may have no way to verify that the key being loaded represents the current state of the key rather than being an older copy of the key that is being loaded. The only safe way for the cryptographic module to ensure that state information is being properly maintained is to store this information internally.

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.

NIST Response: As with other NIST standards, SP 800-208 will not specify encoding methods (other than the XDR that is included). Other standards organizations, such as the IETF, are in a better position to specify encoding methods.

# 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.

NIST Response: Section 2.3 was extended to include definitions of the variables from RFCs 8391 and 8554 that are referenced in the SP, specifically noting instances in which a single variable is used in both RFCs, but with different meanings. NIST does not believe there is a need to have separate documents for LMS/HSS and XMSS/XMSS$^{MT}$. While it is true that RFC 8391 and RFC 8554 sometimes assign different meanings to the same identifier, we believe that implementers understand that the definition of an identifier in one context does not apply to the use of that identifier in a different context. For example, in FIPS 186-4 the identifiers *p* and *q* are used to identify prime numbers in both RSA and DSA, but in RSA they represent the private factors of the modulus *n*, whereas in DSA they represent public domain parameters. The identifiers *n* and *d* are used in the descriptions of both RSA and DSA, but with different meanings in each. Readers of FIPS 186-4 understand that the definitions of identifiers such as *p*, *q*, *n* and *d* in the description of RSA do not apply to DSA and vice versa.

- 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.

  NIST Response: Noted. As is the case with other NIST algorithm publications, SP 800-208 specifies the requirements that implementations must meet, not how to meet them. Those who are trying to implement these algorithms will read the RFCs. While RFC 8391 does not describe tree management strategies itself, it does refer readers to papers that provide this information. We strongly encourage implementers to review the existing literature for guidance on how to most efficiently and safely implement these schemes.

- 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.

  NIST Response: We believe that if there are details in these RFCs that are ambiguous, then it would be better for them to be addressed in the RFCs (either via errata or updates). Implementers may also refer to the reference implementations, when necessary. If NIST were to attempt to address potential ambiguities in SP 800-208, then there would be the risk of NIST unintentionally deviating from the RFCs. Note that details about such issues as how to format public keys and signatures are out of scope for NIST algorithm specifications, except in the case of data objects that are to be digitally signed (e.g., the root of a lower-level tree). Other encoding details are better addressed elsewhere (e.g., RFC 8708).

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.

NIST Response: As noted above, issues related to encoding and efficient implementation techniques are out of scope for SP 800-208. Implementers are strongly encouraged to review the

existing literature on hash-based signatures for information about how to implement these schemes efficiently.

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.

NIST Response: While we agree that an instance of HSS with 8 layers of trees each having a height of 25 would not be practical, we do not believe there is a practical way to prohibit its use, as parameter sets are only defined for LMS, not HSS. As noted in Section 7.1 of SP 800-208, a cryptographic module that implements LMS signing has no control over how LMS instances are combined to create HSS instances. On the verification side, we expect verifiers to process each LMS public key and signature in a chain individually, so that the overall height of the multi-tree will not be relevant.

We also believe that the primary use case for stateful hash-based signatures will be scenarios such as firmware signing, in which the same entity is in control of both the signing process and the verification software, so that interoperability between arbitrary signing cryptographic modules and arbitrary verification cryptographic modules will not be necessary.

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

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.

NIST Response: NIST agrees that it is important to clearly specify encodings and identifiers (e.g., OIDs), but does not believe that SP 800-208 is the appropriate place to specify this information. Just as the underlying signature schemes are defined in IETF publications RFC 8391 and RFC 8554, work is underway in the IETF to specify how to use these signature schemes in various protocols (see RFC 8708, https://tools.ietf.org/html/draft-ietf-cose-hash-sig, https://tools.ietf.org/html/draft-vangeest-x509-hash-sigs, and https://tools.ietf.org/html/draft-mu-curdle-ssh-xmss).

2) 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.

NIST Response: The differences in the parameter set options between LMS/HSS and XMSS/XMSS$^{MT}$ in SP 800-208 are the result of their differences in RFC 8391 and RFC 8554. We do not believe there is a substantial benefit in defining new parameter sets for LMS/HSS or XMSS/XMSS$^{MT}$ just for the sake of making the parameter sets of the two schemes more similar. Allowing for any tree height to be chosen would making testing more difficult, and there does not seem to be a compelling need. In practice, working with a tree of height 10 instead of one of height 8 would not result in substantially longer signatures, or signing and verification times. The additional tree height would also not significantly increase the amount of memory required to implement the schemes.

3) 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.

NIST Response: Algorithm 10' in Section 7.2.1 of SP 800-208 does not specify XMSS key generation, so it is not semantically identical to Algorithm 10 in RFC 8391. Algorithm 10' in

Section 7.2.1 of SP 800-208 is intended for use in implementing XMSS$^{MT}$ in a distributed manner. Algorithm 10 in RFC 8391 could not be used for this purpose, as it only works for XMSS (i.e., it assumes that L=0, t=0, and d=1).

NIST had no involvement in the development of either RFC 8391 or the C reference implementation. However, RFC 8391 explicitly notes that the algorithms in that document are written for simplicity, not efficiency, and so recommends against using them in implementations.

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.

> NIST Response: OIDs have been specified in RFC 8708 for LMS/HSS. While this is not sufficient, NIST believes that it would be more appropriate for other standards organizations, such as the IETF, to specify OIDs and encoding formats rather than NIST.

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.

> NIST Response: While the fixed limit in the number of signatures that can be created is a difference from traditional signature schemes, the concern in this section is mainly the security risk resulting from the need to maintain state. We believe that for applications that have the characteristics described in Section 1.1, the number of signatures that will need to be created will be relatively small, so it shouldn't be too difficult to choose a parameter set that can generate enough signatures.

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.

> NIST Response: The referenced text specifically notes that stateful HBS may be applicable if the implementation will have a long lifetime and it would not be practical to transition to a different digital signature scheme once the implementation has been deployed. In most cases of software update, we believe it would be practical to update the software update mechanism itself in order to be able to work with a newer digital signature scheme.
>
> While it is true that not all systems are updated as quickly as they should be, this is becoming less of an issue as automatic updating becomes more prevalent. In addition, while we cannot predict when quantum computers that will be capable of breaking current signature algorithms will become available, it is likely that it will be possible to start deploying updated software update mechanisms with new (stateless) post-quantum signature algorithms years before the current signature algorithms become insecure.

Line 436:
Please use the $\oplus$ symbol for exclusive-or

> NIST Response: This has been corrected in the final version. Thank you.

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

> NIST Response: This has been corrected in the final version. Thank you.

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.

> NIST Response: Thank you for noting this. The final version of SP 800-208 notes that this is an option.

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

NIST Response: This issue has been corrected in the final version of the document.

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

NIST Response: As noted in Sections 7 and 8, cryptographic modules conforming to SP 800-208 cannot export private keying material. So, the cryptographic module cannot output SK. The cryptographic module must generate and store SK, but must not output it.

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.

NIST Response: We believe that for the applications described in Section 1.1, signing will be infrequent, and so reserving multiple OTS keys at a time will not be beneficial.

Line 844:
s/196/192/

NIST Response: This has been corrected in the final version. Thank you.

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".<br>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".<br><br>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.<br><br>NIST Response: The final version of SP 800-208 has been modified to forbid using an OTS key more than once, even to sign the same message a second time (e.g., recomputing the signature on the root of a lower-level tree). This avoids the potential mistake described here and also protects against fault injection attacks. |
| 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".<br><br>NIST Response: We believe this level of detail is not needed for the high-level description. The checksum formula itself may be unnecessary, but it was included in case it helps some people understand the sentence that follows: “The checksum is designed so that the value is non-negative and any increase in a digit in the message digest will result in the checksum becoming smaller.” |
| 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". |

NIST Response: Accepted.

| | | |
|---|---|---|
| 506 | Editorial | Replace SHA2 with either SHA-256 (to match earlier in the draft) or SHA2-256 (to match [1]). |

NIST Response: Noted. The current text aligns with RFC 8391 [1], which says: “To implement the keyed hash functions, the following is used for SHA2 with n = 32”

## 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.

   NIST Response: As noted in the response to Crypto4A, Section 7 of SP 800-208 was written under the assumption that all of the operational and spare cryptographic modules would be instantiated with keys at the beginning so that the cryptographic module holding the top-level key would not be a single point of failure.

   While there may be many mechanisms available to protect traditional private keys (e.g., secret sharing) while allowing the keys to be copied, these mechanisms do not apply to stateful HBS, since they cannot address the state management issue. As long as copies of keys can be made, there is the risk that the same key (with the same state) will be loaded onto more than one cryptographic module and that this will result in one-time key reuse. While we understand that many organizations would be able to put procedural mechanisms in place to prevent this one-time key reuse, past experience has shown that we cannot rely on all organizations that may choose to use stateful HBS to be able to do this reliably.

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.

   NIST Response: Noted. The requirement for both I and SEED is that they "**shall** be generated using an approved random bit generator (see the SP 800-90 series of publications [6]) where the instantiation of the random bit generator supports at least 128 [8*n*] bits of security strength" This includes the option to use any of the deterministic random bit generators from SP 800-90A.

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