Stateful Hash-Based Signatures

Public Comments on Misuse Resistance—Request Issued Feb. 4, 2019

Larry Marks

From: "Marks, Larry" <larry.marks@bbh.com>
Date: Monday, February 4, 2019 at 1:04 PM

For the guidance for b) I understand that it may be recommended that the user should switch to SHA-3 for 512-bit hashes and to use SHA-512/224 and SHA-512/256 instead of SHA-224 and SHA-256 as a compensating control to make applications more secure.

Larry

Evan Clendening

From: Even Clendening evan@clendening.com
Date: Thursday, February 7, 2019 at 11:02 PM

Hi,

I saw the request for comments and thought I would give one for XMSS.

I have followed a cryptocurrency project that uses XMSS along with wots+ called QRL. It is one way for XMSS to be applied in the real world correctly against attackers with classic and quantum computers.

I also think the team and there code should be included in an example for guidance on how to correctly apply.

It was just a thought since I know of a group using XMSS.

Thanks,
Evan

ISARA Corporation (1)

From: Mike Brown <Mike.Brown@isara.com>
Date: Wednesday, February 13, 2019 at 1:40 PM

Good Afternoon,

Please find enclosed comments from ISARA on the request related to Stateful Hash Based Signatures.

Thanks,

Mike Brown

CTO, ISARA Corporation

ISARA Comment on NIST HBS Certification Process

As organizations globally begin to understand the significant undertaking required to prepare their systems for quantum-safe security, we endorse the dual-track approach to the standardization of stateful hash-based signatures recommended by others.

Our experience has been that many organizations are taking an overly cautious approach to preparing for the quantum threat by doing nothing at all until NIST approves standards – even preliminarily testing and starting proof-of-concept projects.

Stateful hash-based signatures are already used by manufacturers to sign software/firmware to ensure their products are protected from malicious updates in the future. The number of connected devices is growing by the millions every year making the ability to manage and update them over-the-air essential. ISARA is currently working with leading hardware security module (HSM) vendors to implement stateful hash-based signatures into their products, including reliable private key management mechanism design to ensure stateful hash-based signatures are used securely.

Recently, NIST asked the cryptography community to submit responses to two questions regarding how hash-based signatures should and should not be used. The ISARA team believes that:

1. **How should NIST's specification characterize the applications for which such signatures are, or are not, appropriate?**
    - We believe that NIST should focus on applications that are have high security requirements and/or applications that utilize keys for long life cycles. Two examples of these would be code signing, especially in the context of durable IoT devices that may be difficult to update in-field, and PKI root certificates. We do not believe that HBS are appropriate for high volume environments such as authentication of web traffic.
2. **What requirements and guidance for protecting against misuse should NIST include beyond what is provided in the IETF specifications?**
    - We believe that it would be helpful for NIST to focus on private key management techniques as reusing the same one-time-use signature key is the greatest concern. This should also take into account how this is impacted in a hardware security module environment where disaster recovery and continuity of operations must be considered. For example, how should the private key for a root certificate be managed in hardware across multiple, geographically disparate, data centers.

ISARA believes that the essential migration toward new post-quantum algorithms combined with cryptographic agility that the onset of large-scale quantum computing requires is best facilitated by pursuing multiple viable schemes simultaneously.

For that reason, ISARA supports the standardization of both XMSS and LMS.

Crypto4a

From: Jim Goodman, <jimg@crypto4a.com>
Date: Tuesday, February 19, 2019 at 9:36 AM

Hello,

Crypto4a is very keen on seeing this work be standardized and is eager to help in any way it can to move this effort forward so please continue to post requests for input. That being said, here are some comments intended to address the most recent request for input on stateful hash based signatures.

Take care.

Jim

--------------------- NIST Stateful HBS Comments ---------------------

o A request for clarification: the request mentions only XMSS/LMS, but is the intent to consider XMSS-MT/HSS too given they provide additional degrees of freedom and benefits that might be of great interest to the community? We think it should be the latter due to the increased flexibility and benefits that can be derived from being able to tailor the tree structure to the specific application being considered. In particular, HSS' ability to define heterogenous parameter sets for all of the sub-trees should prove to be a very powerful tool when designing for specific use cases. In addition, we've collaborated with partners previously to demonstrate the suitability of full XMSS-MT/HSS to even tightly constrained IoT environments/devices (i.e., Cortex M0/M3/M4-based MCUs), the results of which are available via Github (https://github.com/google/quark), so we think they merit consideration in this discussion.

o NIST could provide a list of characteristics of applications that are well-suited to stateful HBS (e.g., centralized signature generation such as FW update signing, etc.). This could be canvassed from the user community based on their experiences, and then vetted by NIST, or the community to identify applications that there is a general consensus are good use cases for stateful HBS. Our company looked at a number of applications based on criteria such as total required signature capacity, signature size, key generation times, signature generation/verification times, and key management requirements (e.g., backup requirements, key access controls, and hardware protection needs). Using these criteria we arrived at an initial group of four use cases that we thought would be well-suited to stateful HBS:

1) FW signing
2) Root CA (or similar high trust scenarios which have a reduced frequency of signature demands, or whose key usage can be metered out in a controlled fashion)
3) Device configuration management schemes (e.g., MUD-based schemes)
4) Trust Anchor Management Protocol (TAMP) message signing and management

o NIST should provide some "best practice" guidelines related to how stateful HBS should be used in terms of generation, distribution, protection, and recovery. This would identify likely concerns based on feedback from the community and provide suggestions for, or at the very least just draw attention to, how to best deal with these concerns. As a result it would help raise the bar for stateful HBS implementations by giving users a basic checklist of things they should be doing/considering.

o We're not sure if NIST wants to go so far as to provide specific recommendations for methods to address concerns identified in either of the above points as this could end up being prescriptive, which may not be the end goal. However, the results could address specific use cases such as implementations that are completely contained within a single implementation point (e.g., all-in-one-box); implementations that are distributed across geographically-distinct points via redundancy or partitioning of the signature space; and integrating a TAMP-based architecture for managing multiple instances that are co-operatively implementing a stateful HBS generation process. All have unique requirements which may be of interest to some, or all of the community at large, and each may require very different design considerations/decisions to deliver a viable implementation option.

Gemalto

From: GOUGET Aline <Aline.Gouget@gemalto.com>
Sent: Saturday, March 30, 2019 7:41 AM

Dear NIST

On behalf of the Gemalto team working on HBS, please find attached our contribution to the NIST request for public comments.

Best regards
Aline

Gemalto contribution to NIST RPC on Stateful Hash-Based Signature

NIST has announced its intent to standardize on both families of schemes LMS and XMSS. This initiative is expected to foster the deployment of quantum-safe signature in the coming years.

As a developer of cryptographic schemes, agnostic between LMS and XMSS, we are aware of the importance of ensuring backward compatibility with existing implementations. We also believe that:

- An alignment of both specification would be very helpful for developers to facilitate the implementation of both schemes with similar choices in terms of parameters, and
- Some extensions of the current specifications would be very helpful to enable the implementation of both families of schemes in constrained devices using available speed-up accelerator.

We believe that a close collaboration between academic cryptographers and industrials would be very beneficial in getting a suitable standard on HBS schemes in order to foster deployment of quantum-safe security in the coming years.

We are then taking this opportunity to share 3 types of comments to explain our view on the current specification of both LMS and XMSS:

1. Generic comments
2. Alignment of specifications
3. Extension of specifications

[GTO 1, Generic comment] - Highlighting similarities in both LMS and XMSS constructions for helping developers to make suitable technical choices.

Both families of schemes LMS and XMSS share many similarities, as already explained in the paper *"LMS vs XMSS: Comparison of two Hash-Based Signature Standards"* available on eprint IACR at https://eprint.iacr.org/2017/349.pdf. Even if these similarities may look obvious for cryptographers, it may be not that obvious for developers when implementing specifications as currently described in both RFC-8391 and IETF draft.

The use of similar notation and naming of sub-functions in both description should help developers:

- To make fair comparison when selecting a family of schemes depending on their specific constrained (platform, RAM size, NVM size, specific constraints of the use-case,…)
- Select equivalent parameters in both families when willing to implement both families of schemes.

In this document, we provide several technical comments on how we suggest to improve the current description such that similarities and differences could be easier to get for developers.

[GTO 2, Generic comment] - Importance of multi-tree structure for enabling flexibility for developers.

The possibility to use a multi-tree structure is key in the flexibility offer to adapt to different types of constraints for both back-end side management of hash-based signature keys and for implementing hash-based signature on constrained devices.

Indeed:

- On a key management system, it happens that the same signature key can be used by several Hardware Security Modules (HSM). The goal is to provide signatures that can be verified with the same public key and that fit real-time constraints. The use of a multi-tree structure is significantly helpful in that case to be able to manage securely the “state” (or counter) of the hash-based signature. As an example, in a 2-layer multi-tree structure, it is possible to segregate the use of the secret key by dedicating one sub-tree per HSM. In that case, every HSM has the responsibility to manage its own counter and there is no risk of a concurrent use of the same value of counter for the same Winternitz One Time Signature (WOTS) key. The benefit of providing a signature that can be verified with the same public key is then kept while removing the risk of concurrent use of state value by 2 different HSMs.

- On both a HSM or in a constrained device, using a multi-tree structure is helpful to manage the time dedicated to key generation and to split it in several phases that can occur at different times during the life-cycle of the secret signature key.
- For constrained devices with strong limitation on RAM size, *e.g.* <4kB, the use of a multi-tree structure enables to split the signature verification in several phases (1 phase per level). In that case, it becomes possible to verify securely a signature with a higher capacity in terms of number of signatures.

Our generic comment is that the multi-tree structure is very important and it is not only a nice-to-have option in the final standard.

[GTO 3, Generic comment] – Highlighting the possibility to provide the forward-security property

The property of forward-security in hash-based signature is related to the method used to generate the secret keys. Several methods to generate the key could be recommended in the final standard by explaining/comparing advantages and costs. At least, the developer should be aware of the fact that there is several types of security properties that can be achieved depending on the choice of the key generation method.

[GTO 4, Alignment of specifications] - Security strength

In the call for contribution to NIST Post-quantum process, 5 strength categories are defined:

I. Security comparable to AES-128
II. Security comparable to SHA-256/SHA3-256
III. Security comparable to AES-192
IV. Security comparable to SHA-384/SHA3-384
V. Security comparable to AES-256

"NIST recommends that submitters primarily focus on parameters meeting the requirements for categories I, II and/or III, since these are likely to provide sufficient security for the foreseeable future."
https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf

In Table 1, we summarize what is currently defined in both documents (RFC-8391 and IETF draft) based on the defined strength categories.

Table 1: Security strength

NIST security level	I	II	III	IV	V	>V
LMS family	no	yes	no	no	no	no
XMSS family	no	yes	no	no	no	yes

Similar security strengths should be specified for both families of schemes or at least a clarification on the recommended level of strength should be included in the standard.

[GTO 5, Alignment of specifications] - Notation

In the documents RFC-8391 and IETF draft, several parameters with same meaning are used but with different notation. This might be confusing and it may cause misunderstanding during selection of parameters for developers willing to implement both constructions.

In Table 2, we summarize the current notation used in both documents together with the notation we have selected for preparing our comments. There is difference in the meaning of the parameter *w* between LMS and XMSS specification. In XMSS, it is actual length of the Winternitz chain while for LMS it is the length of the chain in bits. A unification in the naming and wording in the final standard for all parameters would be very helpful for developers.

Table 2: Notation

	LMS Family	XMSS family	Notation used in our comment
Winternitz parameter w	length of one chain in bits	actual length of one chain	length of one chain in bits
number of layers in multi-tree	L	d	d
height of one Merkle tree	h	h/d	h
height of the multi-tree	-	h	h_{MT}
number of chains in WOTS	p	len	ℓ
number of chains in WOTS from hash	u	len_1	ℓ_{hash}
number of chains in WOTS from checksum	v	len_2	ℓ_{check}

[GTO 6, Alignment of specifications] - Sets of parameters

The defined values for several similar parameters of both LMS and XMSS are different and there is no technical explanation to justify these differences. As an example, for some values of number of signature, it is not possible to implement both schemes.
Moreover, for developer willing to implement both schemes with similar parameters, it is not straightforward to select the corresponding parameters.

In Table 3, recommended/allowed values from both drafts are summarized.

Table 3: recommended/allowed values

	LMS family	XMSS	XMSS^MT
Winternitz parameter w (in bits)	1, 2, 4, 8	(2[1]), 4[2]	(2[1]), 4[2]
d (number of layers in multi-tree)	1-8	N.A.	1, 2, 3, 4, 6, 8, 12
h (height of one Merkle tree)	5, 10, 15, 20, 25	10, 16, 20	5, 10, 20

[1] Only in the description. It does not appear in any proposed set of parameters.
[2] In RFC-8391 there are used values 4, 16 instead of 2, 4, because the meaning of the w is the actual length of the Winternitz chain.

h_{MT} (height of the multi-tree)	Flexible, minimum is 5 and maximum is 85.[3]	N.A.	20, 40, 60

Since the selection of parameter determines performance (e.g. execution time) and usability of the scheme (e.g. number of signature), not providing the same possibility for recommended/allowed values strongly influence the selection of one scheme. Since there is no obvious technical reason to justify that, similar parameters for both schemes should be provided.

In Table 4, we summarize values that can be reached using one of the proposed parameter set when taking into account parameter dependencies.

Table 4

	LMS family	XMSS	XMSS^MT
ℓ (number of chains in WOTS)	265, 133, 67, 34	131[4], 67	131[4], 67
ℓ_{hash} (number of chains in WOTS from hash)	256, 128, 64, 32	128[4], 64	128[4], 64
ℓ_{check} (number of chains in WOTS from checksum)	9, 5, 3, 2	3	3
number of signatures	Semi-flexible, the smallest values are 2^5, 2^{10}, 2^{15} and the biggest value is 2^{85}.	2^{10}, 2^{16}, 2^{20}	2^{20}, 2^{40}, 2^{60}

According to the LMS Internet-Draft, one has to build his own multi-tree structure by selecting parameters for every layer which gives a good flexibility. For XMSS, there is less flexibility and every layer in the multi-tree has the same parameters.

From our perspective, we would prefer to be able to select similar sets of parameters for both designs and to have the possibility of selecting different parameters per level of subtree.

[GTO 7, Alignment of specifications] - Context identifiers

Context identifiers are used in both XMSS and LMS using different names. On the one hand, prefixes that depend on location within the structure are used in LMS. A prefix consists of several identifiers specifying Merkle tree and index within the tree as well as constants

[3] The number of subtrees has to be smaller than 2^{64} .
[4] For block length $n = 512$.

specifying intended usage of the hash function. On the other hand, *Address scheme* which serves similar purpose are used in XMSS. Address has certain structure, which is very similar to the structure of LMS prefix. It depends on the location of the corresponding hash function in the structure of the scheme. In XMSS, the address is not used directly as a prefix, it serves as an input for PRNG together with public SEED. The output of the PRNG is called *key*. The wording is not very lucky in this case as this information is not secret value and it may be misleading. Final usage of this *key* is the same as in LMS case, because the specified hash function doesn't support additional input beside the message to be hashed. Consequently, it would be wise to use less confusing term for the prefix, as for example $context\_id\_B$, and unify it for XMSS and LMS.

We also mention that there is *masking value* used in XMSS design, which is generated very similarly to the *key*. Thus, we proposed to use unified naming, where the *masking value* would be called $context\_id\_A$.

We then suggest to align naming and description of $context\ id\ A$ and $context\ id\ B$ for both designs so that there can be one global description of both schemes as they are very close in design.

[GTO 8, Alignment of specifications] - Specifying similar sub-functions

Since both schemes rely on similar phases of computations, it would be helpful for developers to have same naming of these different phases, especially for developers willing to implement both schemes.

We provide below a (tentative) description of stateful hash-based signatures using 7 phases denoted by $F_0, F_1, F_2, F_3, F_4, F_5$ and F_6 that could be used to describe both LMS and XMSS.

Phase F_0 – hash of the original message to a constant size message digest c.f. Figure 1

- F_{01}: derivation of an identifier $context\_id\_B$ which depends on the location in the structure; an input to F_{02}.
- F_{02}: this function takes an input of arbitrary length and it returns $message\ digest$ for the next step (e.g. for XMSS, $F_{02} = \mathrm{H\_msg}$)
- F_{03}: this function takes $message\ digest$ as an input and generates ℓ_{check} values which are later appended to the ℓ_{hash} values of the $message\ digest$.

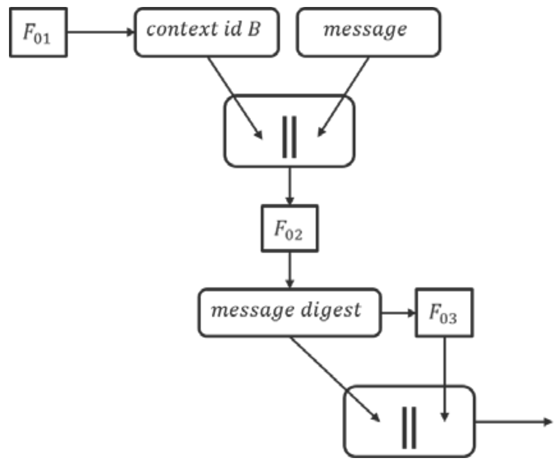

Figure 1: Phase F_0

Phase F_1 – within the Winternitz chains a.k.a. chaining function. It takes intermediate values as an input and returns next intermediate value in the chain. The whole procedure is repeated until we get chain of length w. For better understanding, we enclose pictures of this phase (Figure 2 and Figure 3).

- F_{10}: generation of the identifier $context\_id\_A$, which is XORed with the previous intermediate value (mask generation in XMSS and a value 0 for LMS).
- F_{11}: derivation of an identifier for both LMS and XMSS denoted here by $context\ id\ B$, and that is called keys in XMSS.
- F_{12}: concatenation of $context\ id\ B$ and XOR of $context\ id\ A$ with previous intermediate value as an input and outputs new intermediate value.

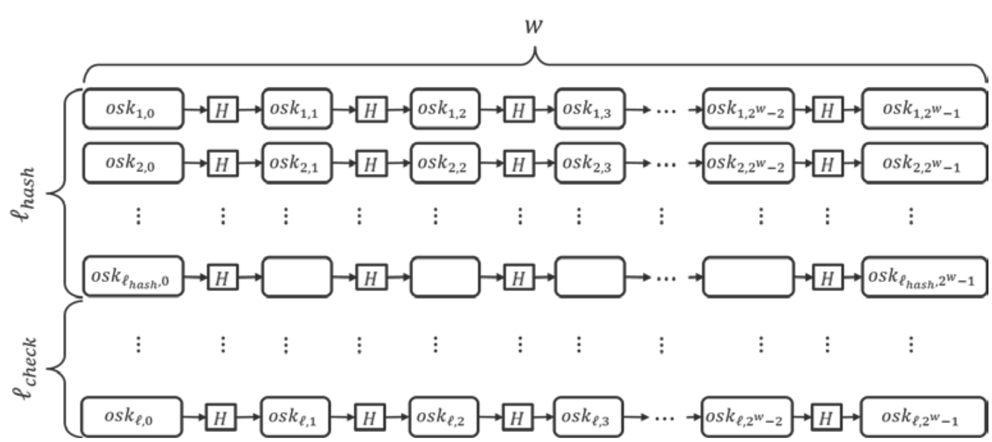

Figure 2: Phase F_1; value $osk_{i,j}$ denotes j-th chaining value in i-th chain; H denotes one chaining step (details in Figure 3)

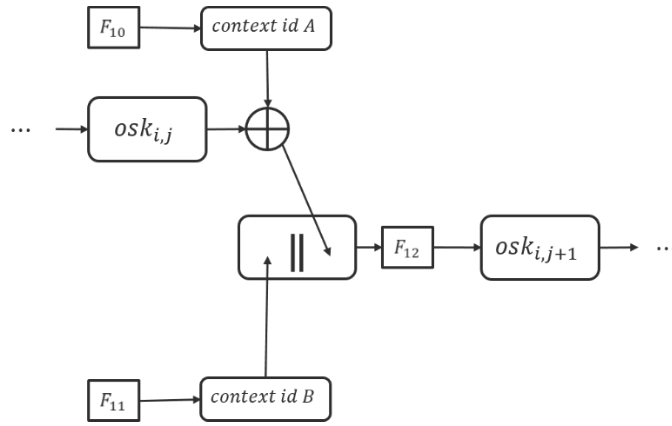

Figure 3: Detail of chaining function H in phase F_1.

Phase F_2 – Combination of last values of Winternitz chains (c.f. Figure 4, Figure 5)
For each layer, two successive nodes $node_{2i,j}$ and $node_{2i+1,j}$ are combined into one node on a new layer $node_{i,j+1}$. For all layers except for the final one, we use functions F_{20}, F_{21} and F_{22}. For the final layer, function F_{22} is replaced by function F_{23}.

- F_{20}: generation of the identifier $context\_id\_A$ (similar to function F_{10})
- F_{21}: generation of the identifier $context\_id\_B$ (similar to function F_{11}). Note that for LMS, function F_{21} returns empty value except for the last call.
- F_{22}: concatenation of two given values. In the end, all final values of the Winternitz chains will be combined using this function except for the last merge. For XMSS, this function corresponds to the computation of hash of previously created value. In the case of LMS, it is simple identity.
- F_{23}: computation of the output denoted by opk. It is also used for the public key of the one-time signature.

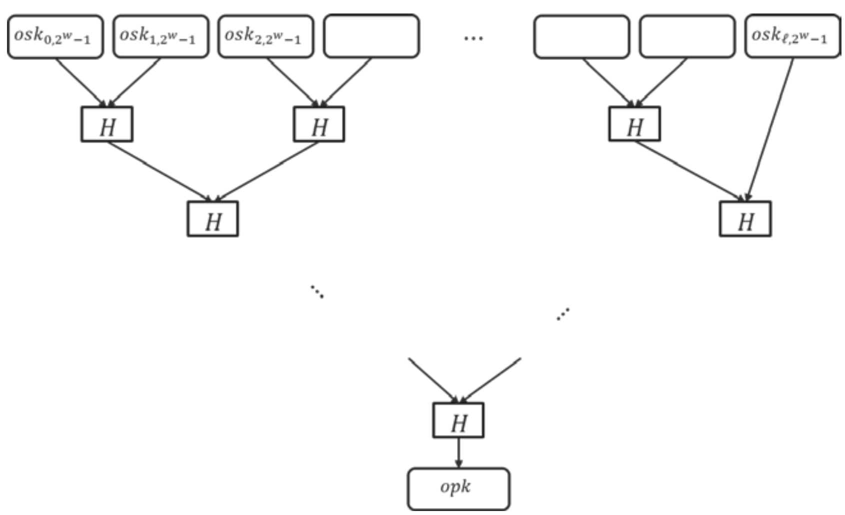

Figure 4: Phase F_2; H denotes merging step (detail can be found in Figure 5)

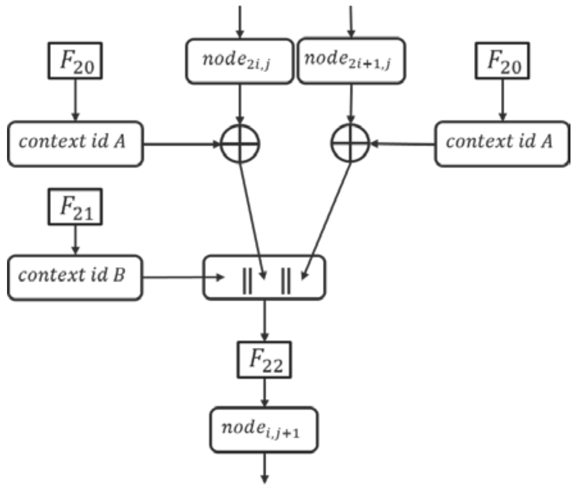

Figure 5: Detail of merging function H in phase F_2.

Phase F_3 – Merkle tree computation (different in XMSS and LMS) (Figure 6, Figure 7 and Figure 8)

- F_{30} – Function similar to F_{20}. Note that it returns value zero for LMS and this function is not used in the first layer with leaves of the Merkle tree.
- F_{31} – Function similar to F_{21}.
- F_{32} – This function is executed only for the leaves of the Merkle tree. For XMSS, this function is identity.
- F_{33} – Function F_{33} takes as an input concatenation of $context\_id\_B$ and two values which were formed as XOR of the node and the corresponding value of $context\_id\_A$. The output of the function is denoted mpk.

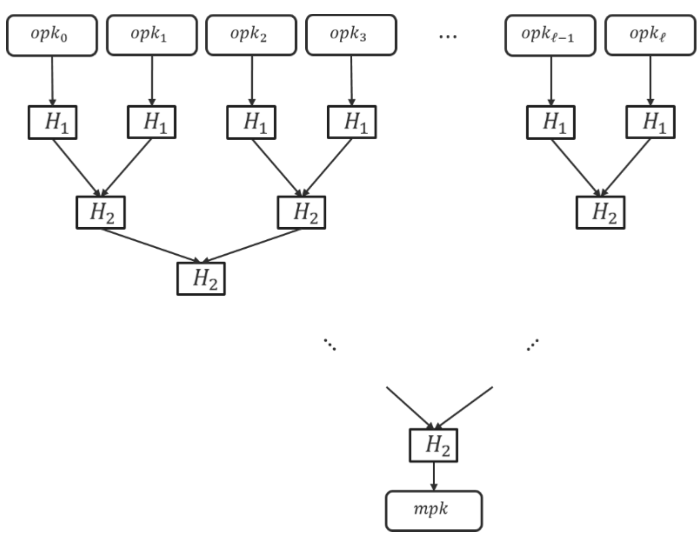

Figure 6): Phase F_3; H_1 denotes preprocessing step (details in Figure 7); H_2 denotes merging step (details in Figure 8)

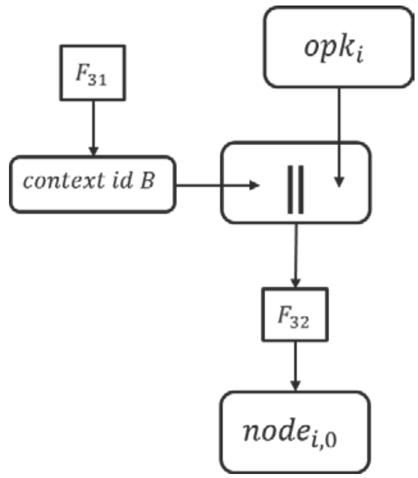

Figure 7: Detail of preprocessing function H_1 in phase F_3.

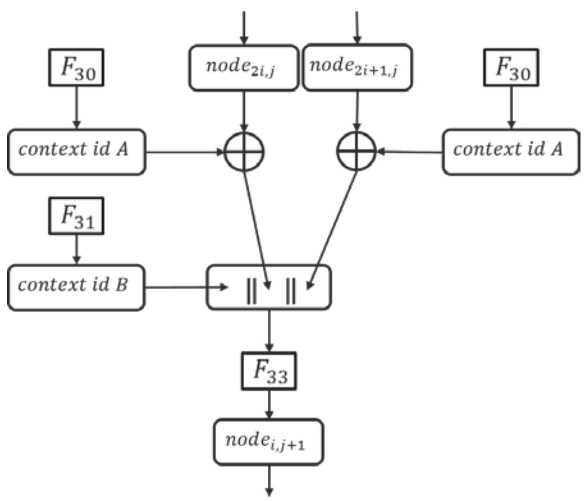

Figure 8: Detail of merging function H_2 in phase F_3.

Phase F_4 – Formation of the public key.

Phase F_5 – Connection of trees in multi-tree variant.

This function takes public key of the upper tree and signs it using parent tree.

Phase F_6 – Formation of the final signature.

[GTO 9, Alignment of specifications] - Specifying similar crypto primitives

While there are several options for the selection of the underlying hash function in XMSS, there is only one possibility for LMS. In LMS, the selection of the hash function is done for the best performance results.

It would be good to align the possibilities for both schemes.

LMS family	SHA-256	
XMSS family	SHA-256, SHAKE-128	SHA-512, SHAKE-256

[GTO 10, Alignment of specifications] - Recommendation on performance improvement

For both designs, there are several possibilities for the implementation choices. Usage of the Merkle tree traversal is recommended in RFC-8391, but there is nothing mentioned in LMS Internet-Draft. As this is a key point for the usability of the scheme, it would be good to have some recommendation for both designs.

[GTO 11, Extension of specifications] - Sets of parameters

On constrained device, there is a tradeoff (using parameter w) between performance and resources (such as RAM consumption). Based on our experimentations, the value $w \in \{2,4\}$ seem to be a good trade-off for devices with limited resources. We, also see possible benefits when using the value $w = 8$.

We believe that standardizing parameter sets with small values for the number of signatures could be suitable for specific use cases and very useful for constrained devices, e.g. all power of 2 between 2^3 and 2^{16}.

In multi-tree structure, the flexibility provided in LMS by enabling trees of different heights in a multi-tree structure is important for developers. For 2 layers and the suggested small number of signatures, we would propose to have height of each tree in set $h \in \{3, \dots, 13\}$. For 3 layers, we would recommend $h \in \{4, \dots, 8\}$. Based on our study, we observe different advantages in choosing to have higher three close to the root of the multi-tree structure, or the opposite for specific cases as well.

More flexibility could be added by enabling different value of w for each layer.

Then, using different OIDs to describe the structure of the multi-tree seems to be most suitable solution compared to using a single OID for the full multi-tree structure.

In Appendix A, we include a table describing parameters we would like to see in the final standard.

[GTO 12, Extension of specifications] - First hashing in hash-and-sign paradigm

The first hashing (phase F_0) of the message is not done in the same way in both LMS and XMSS.
In LMS, we have:

$$F_{01,LMS} = I \parallel q \parallel D\_MESG \parallel C$$

where I and q are two identifiers, D_MESG is a constant and C a randomization element. Output of the function is $context\_id\_B$ and it is concatenated with the message to be signed. Result of the concatenation is used as an input for F_{02} function. Function F_{02} takes input of arbitrary length and hash it using specified primitive (for further details see [GTO 8, Alignment of specifications] - Specifying similar crypto primitives). The output message digest is later processed by the last function F_{03} which computes checksum of the message digest (which is necessary for the following phase). Finally, the message digest is concatenated with its checksum and the results is outputted.

$$F_{0,LMS} = F_{02,LMS}(F_{01,LMS}) \parallel F_{03,LMS}\left(F_{02,LMS}(F_{01,LMS})\right)$$

In the case of XMSS, we have:

$$F_{01,XMSS} = 2 \parallel r \parallel root \parallel idx\_MT$$

where r is a randomization element, $root$ is a constant identifier and idx_MT is an identifier of the location. All values are concatenated together with constant 2.
Second function F_{02} is equivalent to the function of LMS. The value $context\_id\_B$, generated by the function $F_{01,XMSS}$, is concatenated with the message to be signed and results serves as an input for the function F_{02}. Function F_{02} takes input of arbitrary length and hash it using specified primitive (for further details see [GTO 8, Alignment of specifications] - Specifying similar crypto primitives). The output message digest is later processed by the last function F_{03} which computes checksum of the message digest (which is necessary for the following phase). Finally, the message digest is concatenated with its checksum and the result is outputted.

$$F_{0,XMSS} = F_{02,XMSS}(F_{01,XMSS}) \parallel F_{03,XMSS}\left(F_{02,XMSS}(F_{01,XMSS})\right)$$

As we can see, the structure is very similar. In both cases, there might be problem with devices with limited resources. Typically, these devices work in three possible modes:

1) hash fully done off card: only the hash transferred to the device
2) hash partially done outside: intermediate hash transferred to the device with the last block and final hash computation is done by the device
3) hash fully computed on card

For both designs, it is possible to use only the last mode, which may cause some problems in case of long messages. It would be nice if it is possible to change order of the concatenated values in function F_{02}.

$$message\ digest = F_{02}(message \parallel context\ id\ B)$$

This modification would allow to use mode 2) as well. In that case partial hash of the message could be provided instead of the full message to be signed.
Furthermore, in RFC-8391, there is a specification of the r computation as $r = PRF(SK_{PRF}, idx\_MT)$. Is it necessary to derive value of r in this manner? There would not be any difference if the value of r was generated uniformly distributed.

[GTO 13, Extension of specifications] - Implementation on constrained devices / adding new crypto-suite

For both schemes, there is the biggest performance impact based on the many computations of the crypto primitive. Therefore, there is a huge impact on the performance of the whole scheme even for very small differences in the primitive computation.
As there is a hardware accelerator for AES computation available on many devices, it would be nice to have the possibility to use it for speeding up the overall performance.
As mentioned in SPHINCS+ submission to the NIST post-quantum project available at https://sphincs.org/data/sphincs+-specification.pdf, it might be beneficial also to consider Haraka function as an underlying primitive.
One can focus on the computation of the OTS public key/signature as this is clearly the most consuming part of the scheme. Chaining function does not necessarily have to be based on the same primitive as the rest of the scheme.
It might be worthy to consider KDF in Counter Mode as specified in NIST SP 800-108 for function F_{12} while using either CMAC-AES-256 or CBC-MAC-AES-256 for the PRF function (since the length of the input message is a fixed value).
In this case intermediate value of the Winternitz chain would be considered as a key derivation key K_I from the NIST standard.

[GTO 14, Extension of specifications] - Context identifiers

In LMS scheme, specific choices of the identifier lengths seems to be done based on specification of the used hash function (SHA-256). It would be interesting for improving performances to define the format of the context identifiers depending on the choice of the underlying cryptographic primitive. Currently, there are identifiers $context\_id\_A$ and $context\_id\_B$ of various lengths for both designs and for different usage.

	LMS family	**XMSS family**
F_{10}	0 – 0 bytes	$\mathrm{PRF}(3 \parallel \mathrm{SEED} \parallel \mathrm{ADRS})$ – n bytes[5]
F_{20}	0 – 0 bytes	$\mathrm{PRF}(3 \parallel \mathrm{SEED} \parallel \mathrm{ADRS})$ – n bytes[5]
F_{30}	0 – 0 bytes	$\mathrm{PRF}(3 \parallel \mathrm{SEED} \parallel \mathrm{ADRS})$ – n bytes[5]

Table 5: $\mathbf{context\ id\ A}$; PRF function is described in RFC-8391; length of $\mathbf{SEED}$ is equal to $\mathbf{n}$; length of $\mathbf{ADRS}$ is 32 bytes

	LMS family	**XMSS family**
F_{01}	$I \parallel q \parallel D\_MESG \parallel C$ – 54 bytes	$2 \parallel r \parallel root \parallel idx\_MT$ – $(3n + 4)$ bytes[5]
F_{11}	$I \parallel q \parallel i \parallel j$ – 23 bytes	$0 \parallel \mathrm{PRF}(3 \parallel \mathrm{SEED} \parallel \mathrm{ADRS})$ – $(n + 4)$ bytes[5]
F_{21}	$I \parallel q \parallel D\_PBLC$ – 22 bytes	$1 \parallel \mathrm{PRF}(3 \parallel \mathrm{SEED} \parallel \mathrm{ADRS})$ – $(n + 4)$ bytes[5]
F_{31}	$I \parallel r \parallel D\_LEAF/D\_INTR$ – 22 bytes	$2 \parallel \mathrm{PRF}(3 \parallel \mathrm{SEED} \parallel \mathrm{ADRS})$ – $(n + 4)$ bytes[5]

Table 6: $\mathbf{context\ id\ B}$; PRF function is described in RFC-8391; length of $\mathbf{SEED}$ is equal to $\mathbf{n}$; length of $\mathbf{ADRS}$ is 32 bytes

[5] Parameter n denotes size of the hash function block in bytes. For XMSS, it can be either 32 or 64.

It may be possible to reduce the number of the identifiers by removing some of them. This would require a security analysis but it would help to improve performance of the resulting scheme.
Also, clarifying the sizes of identifiers indeed needed to guarantee the security, depending on the number of signatures (or other selected parameters) would be very helpful to improve performance by removing the need to perform some calculations in the case of smaller parameters.

I. Appendix A – extension of parameter sets

In the following table, we summarize the set of parameters that seem to us specifically relevant for implementation on constrained devices.

In multi-tree structure with $d \geq 2$, the values w and h are not necessarily equal for each layer. Values h has to be selected in such a way that sum of those values equal to the value of h_{MT}. The approach used in LMS draft (hierarchical description of the structure) is well suited for the specification of the parameters as it gives the necessary flexibility in parameter selection. This gives parameter choices of the one-time signature for each layer independence of the parameter choices for Merkle tree and multi-tree variant.

w	d	h_{MT}	h	number of signatures
from $\{2,4,8\}$	1	3	3	2^3
from $\{2,4,8\}$	1	4	4	2^4
from $\{2,4,8\}$	1	5	5	2^5
from $\{2,4,8\}$	1	6	6	2^6
from $\{2,4,8\}$	1	7	7	2^7
from $\{2,4,8\}$	1	8	8	2^8
from $\{2,4,8\}$	1	9	9	2^9
from $\{2,4,8\}$	1	10	10	2^{10}
from $\{2,4,8\}$	1	11	11	2^{11}
from $\{2,4,8\}$	1	12	12	2^{12}
from $\{2,4,8\}$	1	13	13	2^{13}
from $\{2,4,8\}$	1	14	14	2^{14}
from $\{2,4,8\}$	1	15	15	2^{15}
from $\{2,4,8\}$	1	16	16	2^{16}
from $\{2,4,8\}$	2	6	3	2^6
from $\{2,4,8\}$	2	7	from $\{3, 4\}$	2^7
from $\{2,4,8\}$	2	8	from $\{3, \ldots ,5\}$	2^8
from $\{2,4,8\}$	2	9	from $\{3, \ldots ,6\}$	2^9

from $\{2,4,8\}$	2	10	from $\{3,\ldots,7\}$	2^{10}
from $\{2,4,8\}$	2	11	from $\{3,\ldots,8\}$	2^{11}
from $\{2,4,8\}$	2	12	from $\{3,\ldots,9\}$	2^{12}
from $\{2,4,8\}$	2	13	from $\{3,\ldots,10\}$	2^{13}
from $\{2,4,8\}$	2	14	from $\{3,\ldots,11\}$	2^{14}
from $\{2,4,8\}$	2	15	from $\{3,\ldots,12\}$	2^{15}
from $\{2,4,8\}$	2	16	from $\{3,\ldots,13\}$	2^{16}
from $\{2,4,8\}$	3	12	4	2^{12}
from $\{2,4,8\}$	3	13	from $\{4,5\}$	2^{13}
from $\{2,4,8\}$	3	14	from $\{4,5,6\}$	2^{14}
from $\{2,4,8\}$	3	15	from $\{4,5,6,7\}$	2^{15}
from $\{2,4,8\}$	3	16	from $\{4,5,6,7,8\}$	2^{16}

Andreas Huelsing

From: "A. Huelsing" <ietf@huelsing.net>
Date: Friday, February 22, 2019 at 5:44 AM

Hello NIST team,

I largely agree with the comment by Panos Kampanakis. I would add another dimension to distinguish applications:

In my opinion stateful HBS are especially well suited for scenarios where the signature algorithm is implemented on some kind of secure execution environment, like a smart card, a hardware security module, or a TPM chip. For such applications it is strictly easier to keep a state. For that reason, I think it would be worthwhile considering HBS for CAs in general, especially when using forward secure key generation this has additional benefits beyond post-quantum security (see our EuroPKI'12 publication https://huelsing.net/wordpress/wp-content/uploads/2013/05/paper.pdf).

Best regards,

Andreas

Panos Kampanakis

From: “P. Kampanakis” <ietf@pkampana>
Date: Wednesday, February 20, 2019, at 18:01 schrieb Panos Kampanakis (pkampana):

Hello NIST PQC team,

This email is to provide feedback on the request for comments at https://www.nist.gov/news-events/news/2019/02/request-public-comments-stateful-hash-based-signatures-hbs

- How should NIST's specification characterize the applications for which such signatures are, or are not, appropriate?

As we know, stateful HBS' concern is keeping state. The most evident categorization of applications that are not friendly to stateful HBS are live / on-line applications. Such applications usually include high rates of signatures in a live environment where messages are exchanged and singed on-the-fly. Examples are TLS, IKEv2 etc. Using a stateful HBS scheme to sign / authenticate TLS messages on a server that terminates thousands of multi-threaded TLS sockets is probably more prone to state sync issues that could compromise security. So, live vs offline applications could be one characterization with the offline ones being more suitable for stateful HBS.

Another characterization is signature frequency. Applications that sign hundreds of messages per second are less friendly to state synchronization than applications that sign a message per day or week. Examples include a VPN termination endpoint vs software signing.

- What requirements and guidance for protecting against misuse should NIST include beyond what is provided in the IETF specifications?

In 2016, we had tried to summarize some of the challenges of state management and propose some solutions in https://eprint.iacr.org/2016/357.pdf that was presented in SSR 2016. In the document, we lay out the challenges and propose some countermeasures. Additionally, Section 4 includes some OS specific thoughts on keeping state.

Best practices include a state reservation strategy where you load many states in advance and if there is crash or a failure you lose x amount of signatures that you can't generate, but at least you don't need to update state one at a time. Frequent state updates increase the risk of state sync mess-ups. We also propose hierarchical HBS trees where the top is kept in non-volatile memory and the bottom in volatile memory. Such practices can be used so that the state in volatile memory does not need to be updated often and a failure in volatile memory is not detrimental.

Also, I believe the NIST document should address VM cloning. VM cloning could practically deem stateful HBS broken if state is cloned with the VM. NIST should provide thoughts on VM cloning and HBS state. It could provide recommendations like after a VM clone the stateful HBS counter could be refreshed to a future state to ensure that the cloned VM starts from an unused instantiation. But that assumes that the parent VM of the clone no longer signs with the same tree. Or ideally, a new tree is instantiated every time a clone is kicked off.

Signing parallelization should also be addressed. If I sign multiple things at the same time there is more chance state will be messed up and reused which deems the HBS scheme insecure. Careful state locking and protections are possible there, and that should be carefully considered by implementers.

Additionally, I think it would be worth for NIST to address if stateful HBS is suitable for X.509 certs. I believe they are just fine for root level CAs. These CAs stay offline after just signing only a small set of subCA keys, so state synchronization is not a concern. Someone could use a short height HBS tree for a root CA that can sign subCA very quickly and with very small sig size. Stateless schemes cannot lower the sig size like that. The disadvantage of course will be that the cert chain will have use a different algorithm in the root of a cert chain and the rest of the certs in the chain.

Thank you,
Panos

Adam Langley

On 3/27/19, 1:38 PM, "Adam Langley" <agl@imperialviolet.org> wrote:

(Submitted in an individual capacity in response to https://csrc.nist.gov/News/2019/stateful-hbs-request-for-public-comments—not speaking for my employer.)

Abstract: The contexts in which stateful signatures are useful are currently limited. Competent entities who are exploring such contexts may be able to productively deploy stateful signature schemes, but such entities will do so with or without NIST's approval. Thus NIST's approval of stateful schemes at this time will only inappropriately grant the imprimatur of the US Government and embolden misapplications of these designs.

Hash-based signature schemes are attractive because they avoid resting on mathematical assumptions that are vulnerable to analytic breakthroughs or quantum attacks. Stateful, hash-based signature primitives are a building block in hash-based signature schemes and, as such, research into them is valuable. By themselves, these building blocks are faster and smaller than a fully fledged hash-based signature scheme, but collapse if the signer doesn't keep perfect state from past operations. But these performance advantages are leading people to wonder whether building a full (and thus stateless, like everything deployed so far) scheme is necessary.

Short-lived keys that can be held exclusively in memory mitigate many of the disadvantages of stateful primitives, but eliminate most of the advantages of hash-based signatures too. The contexts in which cryptanalytic/quantum resistance is most valuable are those that involve long-lived keys. Using stateful primitives directly trades a great deal of operational risk for this improved resistance and whether that's a good deal remains an unresolved question. However, we have quite a lot of evidence from the operation of long-lived keys in the WebPKI that the operational risk should not be downplayed.

The processes around long-lived keys must be robust to the erosion experienced by all organisations: the cumulative effect of decades of human error; the dissolution of the original design team; loss of knowledge; reorganisation & acquisition; deprioritisation & neglect as economic conditions change. The last couple of decades have provided concrete examples of all these in the WebPKI, against a generous backdrop of audited organisations.

The most obvious parallel between the needs of stateful signatures and the WebPKI are X.509 serial numbers. These serial numbers are required to be unique and revocation mechanisms are ineffective if they are not. However, the space of serial numbers is vast and so it's much easier to manage them compared to the dense nonce space of stateful signatures: just generate them randomly.

Nonetheless, it is not difficult to find cases where intermediate certificates have been issued with duplicate serial numbers[1][2][3]. In an additional case, a CA issued an intermediate to an inappropriate entity who then misissued certificates. The CA was unable to revoke the intermediate because it shared a serial with a different, active intermediate. Intermediate certificates should have the highest level of scrutiny given that they typically carry the full authority of the root and such limited issuance is the sort of situation where stateful signatures are most often suggested.

Outside of intermediates, the situation is worse. We find 314 leaf certificates[4] sharing the same 128-bit serial, for example.

Stateful signature schemes require perfect knowledge of previously used nonces and a recent rule requiring the revocation of certain non-conformant certificates tested CAs' record-keeping. Although this only covered unexpired certificates and thus only the past few years, and was non-adversarial, this was a challenge in several cases[5][6][7][8]. Record keeping has been found to be inadequate while investigating more serious problems[9] and has failed under adversarial conditions[10].

The ravages of the real world can also take more extreme forms: CAs have lost track of intermediates that they've signed, which were only discovered from bulk datasets of certificates.

Human processes are clearly insufficient. Thus the safe operation of long-term, stateful signatures will require new forms of HSMs and new operational practices, with which the ecosystem has little or no experience and which have not yet proven themselves. More adventurous organisations will undoubtedly experiment in this direction, and will do so whether NIST approves these algorithms or not. If their experience is positive, and the necessary infrastructure moves from being bespoke to being a commodity, then stateful primitives might be appropriate for wider use. NIST is the leading certification body for cryptographic modules and, as such, it would seem obvious that approval of stateful algorithms could be coupled with new standards for cryptographic modules that can provide for their unique needs.

Absent that, however, NIST's approval would falsely communicate that stateful primitives are suitable for widespread use outside of being components in stateless schemes.

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=1405817
[2] https://bugzilla.mozilla.org/show_bug.cgi?id=1405815
[3] https://bugzilla.mozilla.org/show_bug.cgi?id=1404403
[4] https://crt.sh/?serial=056d1570da645bf6b44c0a7077cc6769&iCAID=1662&p=1&n=100
[5] https://bugzilla.mozilla.org/show_bug.cgi?id=1526154
[6] https://bugzilla.mozilla.org/show_bug.cgi?id=1521520
[7] https://bugzilla.mozilla.org/show_bug.cgi?id=1462844
[8] https://bugzilla.mozilla.org/show_bug.cgi?id=1524815
[9] https://security.googleblog.com/2015/10/sustaining-digital-certificate-security.html
[10] https://security.googleblog.com/2014/07/maintaining-digital-certificate-security.html

ISARA Corporation (2)

From: Philip Lafrance <Philip.Lafrance@isara.com>
Date: Monday, April 1, 2019 at 11:46 AM
Greetings good people of NIST,

Attached are our comments regarding stateful hash-based signature algorithms and their potential standardization.

We look forward to hearing NIST's conclusions on the matter!

Out of curiosity, will these comments be made public at some point, or are they expected to be kept by NIST?

Warm regards and all the best,
Philip Lafrance and the ISARA team

Comments on Stateful Hash-based Signature

Schemes

April 1, 2019 ISARA Corporation

The present document is a response to NIST's call for feedback on the standardization of stateful hash-based signature algorithms. This document discusses some high-level concerns with the CFRG's Internet-Draft on the Hierarchical Signature System (HSS), and RFC 8391 on the Multi-Tree variant of the eXtended Merkle Signature Scheme (XMSS-MT). Several concerns around the secure management of state, as well as side-channel attacks on these schemes are highlighted. Further, this document discusses several general issues for stateful hash-based signature algorithms such as: (in)appropriate use cases for such algorithms, problems with parameter selection and analysis, issues surrounding state management, disaster recovery, the existence of fault attacks, and so on.

The reader should be aware that this document does not attempt to resolve each of the issues discussed, but rather serves to inform the reader of the many considerations NIST must make for the development of standards for stateful hash-based signature schemes that facilitate secure implementations with maximum interoperability.

1 Usage Limitation

• Due to particular characteristics of Stateful Hash-based Signature Schemes, these schemes do not appear to be suitable for general purpose signatures, but rather should be used for specific use cases. For example, stateful signature schemes seem to be well suited to code signing or signing of digital certificates, where state can be managed within a closed system. Such signing procedures are relatively infrequent and are performed in highly controlled and secure environments.

- On the other hand, stateful signature schemes may not be appropriate for use in TLS or IKEv2 handshakes, where such signatures are generated with high frequency and in less secure environments.
- It is recommended that any NIST Stateful HBS standard explicitly limits the deployment options of these algorithms to low frequency signing cases, preferably in highly secured environments or devices.

2 Issues in LMS-HSS IETF Draft

- The current CFRG LMS-HSS Internet-Draft defines parameter sets for the OTS and for LMS independently. This gives a number of parameter choices multiplicative in the number of layers in the hyper-tree. Concretely, there are 4 distinct Winternitz parameter sets, 5 LMS parameter sets, and up to 8 layers are permitted. Together this allows for up to $\sum^{)}(4*5)^{\wedge}i$ possibleparametersets.Thisisapproximately2^35possibilities.
- Most of these parameter options are suboptimal; they can be reduced to around 150,000 sets of similar choices (e.g. H10+H5 vs H5+H10 might be considered the same), but even in those similar choices there can be performance differences depending on implementation strategy.

- This excessively large number of parameter choices puts the implementations of this proposed standard at risk of being non-interoperable, or at least increases the difficulty of successful parameter negotiation.
- Furthermore, it will add complexities in encodings of the parameter sets and might make APIs difficult to use. It is highly recommended that NIST limits the parameter set choices to a reasonably sized set of carefully selected parameters. It should be noted that a number of distinct parameter choices yield very similar speed and size trade-offs for a given security level, but depending on the state management strategies employed, these

*

parameter sets may have different performance characteristics.

- Currently the draft standard is hard-coded for SHA-256. Supporting additional hash functions such as SHA-3 may be desirable. Such an extension will expand the list of parameter choices linearly. The current draft does not easily allow other hash algorithms to be used, and depending on the use case and implementation, it is conceivable that other hash functions will be desirable in the future for use in hash-based systems.
- Parameters are encoded in the public key and signature; current CMS-related IETF drafts require this and explicitly forbid encoding parameters outside of the key or signature. As a result, a malicious actor can easily cause buffer overruns by manipulating the signature data.
- It is recommended that NIST introduce domain parameters similar to DSA and ECDSA. Moreover, parameters should be negotiated or transported either externally or bundled within the public key.
- HSS permits different choices of parameters (Winternitz and tree height) per different tree layer. However, each tree on any given layer is supposed to have the same parameters. For example, there should not be both a height 10 and a height 15 LMS tree on the same multi-tree layer. The current draft has the following text:

o *"A close reading of the HSS verification pseudocode would show that it would allow the parameters of the non-top LMS public keys to change over time; for example, the signer might initially have the 1-th LMS public key to use LMS_SHA256_M32_H10, but when that tree is exhausted, the signer might replace it with LMS_SHA256_M32_H15 LMS public key. While this would work with the example verification pseudocode, the signer MUST NOT change the parameter sets for a specific level. This prohibition is to support verifiers that may keep state over*

the course of several signature verifications."

• This is the only place in the current draft where this problem is observed; there is nothing inherent to the algorithms to prevent this sort of behaviour, and the warning above may be missed by an inattentive implementor. Further, if this restraint is not abided by, then the possible parameter choices increases dramatically, causing yet more interoperability concerns. There may be security concerns if this warning is not heeded as well, such as the inability to detect this violation on the verification end. It is recommended that any Stateful HBS standard produced by NIST clearly and explicitly address this issue.

3 Mitigation of Fault Attacks

• Multi-tree schemes (HSS, $XMSS^{MT}$, SPHINCS+) are known to be vulnerable to fault attacks, leading to information leakage and signature forgery attacks (see *Grafting Trees: a Fault Attack against the SPHINCS framework* by Castelnovi, Martinelli, Prest, and Sud-Ouest). Systems implementing these schemes need to address at a minimum the following issues:

o The use of ECC memory is encouraged to help prevent possible state corruption. o Prevention of malicious or accidental physical access; this requirement restricts

the possible environments these algorithms can be securely used in.

• Note that these are issues beyond algorithm specification. A better place might be in a more system and operational scope standard such as the FIPS 140 series. In such a framework, the standard points out the issues to be addressed, while the vendors must provide evidence that the issues are addressed in the manner of the design documents for the system, implementation specifications, and by the security policies for the operational matters. These must also be audited in the process of module validation.

4 Challenges in State Management

Secure state management is critical for the security of stateful hash-based signature schemes. State management is a system dependent issue that can be addressed by implementation specifics and operational guidelines such as security policies. As discussed in Section 3, perhaps the issues are better addressed in the scope of a standard such as FIPS 140-2, where system issues, especially key and CSP management issues, are discussed based on specific vendor implementations rather than an algorithm standard.

The issues to be addressed should include, but are not limited to, the following: ***Issue:*** State is an unknown concept to existing protocols and APIs.

- Current protocols and APIs may not know what state is, nor how it should be processed. This is particularly true in key generation and signing operations. If the state is part of the private key for example, then problems occur with APIs that treat private keys as read-only objects that cannot be changed during operations such as signing. Additionally, some key stores are severely limited for space; they may be able to store a private key, but the private key and state together may be much too large to fit. Existing IT infrastructures where keys are backed up and stored at creation may not be easily modifiable to accommodate state.
- Furthermore, widely adopted public key standards such as PKCS#11 do not have a place for state; this promotes the need for custom work around the state object.
- Due to this, it remains unclear how stateful schemes could be incorporated into current handshake protocols such as those in TLS or IKEv2. It may require considerable amount of work to update such protocols to accommodate statefulness.

  Issue: State must be managed for computing clusters or across (virtual) machines and devices.

• Load balancing multiple stateful signing machines that have the same key pair can be difficult as clustered signing machines may exhaust state faster than single machines, and state must be synchronized between all the clustered machines.

- Considerations required here include how state is re-synchronized when a machine comes back online and how to ensure a previous state, which was current at the time the machine went offline is not usable or exploitable before it is re-synchronized with the rest of the cluster. Moreover, real-time state synchronization between live machines must also be done.
- Backups for these systems must also be considered. It may be possible to use a single machine as a backup solution for a cluster or network, but that backup will also need to know about state. However, a retrieved backup with an already used state represents state re-use concerns. Further concerns around state storage are discussed below.
- For devices deployed across either a network or a large geographic area, or in a computing cluster, additional issues include:

o Compensatingfortimedelaysinsynchronizingpairwisedifferentmachines.
o Syncingtoacentral,standardizedtimesource.
o Gracefulhandlingofsynchronizationfailure.
o State synchronization of machines coming online either for the first time or

otherwise.
Issue: State is implementation-defined.

- There is no general standardized method for the handling of state. In fact, the representation of state is a local matter. Depending on the platform architecture (e.g., 8-bit vs 64-bit systems, or big- vs little-endian machines) the handling of state must be considered and developed for each different state-requiring process.
- The representation of state and the specification of state transition is implementation dependent. The specification of the representation of a state may assist the transfer of

private keys, including their state, between different implementations which will require each vendor to implement “transform” or “interpret” layers. However, since it is not an interoperability issue, it is unclear if this mixture of different implementations for a single private key is beneficial. It rather creates chances of misinterpretation, causing security risks.

Issue: State must be safely stored.

- Applications need to *guarantee* that the updated state has been written to storage. For else they run the risk of re-using state, and thus allowing for forged signatures. Writing state to disk may not be guaranteed on most platforms because hardware caches might cache data prior to writing to physical media. For example, Linux's sync() function merely schedules things to be written to disk. To help prevent improper storage or corruption of state, every component between the CPU and storage needs to have power backup to avoid loss during power failure. This requires custom hardware or infrastructure.
- Stateful hash-based signature schemes are able to produce an exponential number of signatures, each requiring a unique state. The state has to be updated for each new signature, and hence, exponentially many writes to the storage medium are required. Depending on the medium of storage then, this can represent a major problem. Writing updated state to Flash

storage can easily wear out the Flash. For example, cheap Flash allows on the order of 100,000 writes before wearing out. Using higher-reliability Flash, or massively over- provisioning Flash storage on hardware may be required. This is also a problem for key generation.

- State must not only be backed up securely but must be retrievable from storage securely.
- State must be synchronized between volatile and non-volatile memory. For example, if state is securely retrieved from non-volatile storage and is used in temporary volatile memory, where it may or may not be updated, then it must be ensured that any difference in the

state between volatile memory and non-volatile memory is not exploitable.

- Escrow of stateful keys also represents a problem. If an organization has their stateful keys held in escrow by a trusted third party, then the stored key's state will fall behind the live state being used by the organization. When the key is retrieved from escrow or storage, the state will be different from the current live state. Unless this retrieved state is synchronized securely, there is a risk of state (key) re-use. This is the same problem as in key storage.
- In order to achieve secure deployment of state management, the following items must be handled:

o full control over the software and hardware platform
o reliable protection against power outages or other physical problems o reliable (disaster) recovery mechanisms

Issue: Multi-tree variants are vulnerable to fault attacks corrupting state.

• There exist practically exploitable fault injection attacks against algorithms in the "SPHINCS framework", which includes HSS and XMSS-MT. The successful execution of these attacks requires physical access to the device wherein the algorithm is run, and thus restriction from access to physical devices may be required. This was discussed in Section 3 above but is repeated here as it is a state management issue as well.

Issue: State may be (un)intentionally cloned in Virtual Machines.

• Algorithm instances running in Virtual Machines or other logical environments have a risk of having their state or related data (un)intentionally cloned. Virtual Machine environments often provide functionality for live cloning. For algorithms which maintain state, this represents a risk of state re-use.

Issue: State must not be corrupted.

• A single flipped bit in the state could lead to state corruption, or in the worst case, re-using one-time signatures. It is recommended that hardware platforms use ECC memory to avoid corruption.

Issue: State must not be re-used.

• Operational standards for backups, load balancing, and disaster recovery need to ensure that stale state is never re-used. If state has been reserved for future use, it must be guaranteed that that state has never been used to sign.

Issue: Even encrypting state on the machine using it is insufficient to protect it.

• An attacker with access to the machine may copy the state, wait for a signature to be created, then restore the copied state. The next signature generated will reuse the state without the attacker ever knowing the state contents. This furthers the argument that physical access to devices running stateful hash-based signature algorithms should be restricted.