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 multi-tree 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 8*n* 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.