

Post-Quantum Use In Protocols  
Internet-Draft  
Intended status: Informational  
Expires: 11 August 2026

T. Wiggers  
PQShield  
K. Bashiri  
BSI  
S. K~~u~~hlbl  
Google  
J. Goodman  
Crypto4A Technologies  
S. Kousidis  
BSI  
7 February 2026

Hash-based Signatures: State and Backup Management  
draft-ietf-pquip-hbs-state-03

Abstract

Stateful Hash-Based Signature Schemes (Stateful HBS) such as LMS, HSS, XMSS and XMSS<sup>MT</sup> combine Merkle trees with One-Time Signatures (OTS) to provide signatures that are resistant against attacks using large-scale quantum computers. Unlike conventional stateless digital signature schemes, Stateful HBS have a state to keep track of which OTS keys have been used, as double-signing with the same OTS key allows forgeries.

This document provides guidance and catalogs security considerations for the operational and technical aspects of deploying systems that rely on Stateful HBS. Management of the state of the Stateful HBS, including any handling of redundant key material, is a sensitive topic. This document describes some approaches to handle the associated challenges. It also describes the challenges that need to be resolved before certain approaches should be considered.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at <https://hbs-guidance.github.io/draft-hbs-state/draft-ietf-pquip-hbs-state.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-pquip-hbs-state/>.

Discussion of this document takes place on the Post-Quantum Use In Protocols Working Group mailing list (<mailto:pgc@ietf.org>), which is archived at <https://mailarchive.ietf.org/arch/browse/pgc/>. Subscribe at <https://www.ietf.org/mailman/listinfo/pgc/>.

Source for this draft and an issue tracker can be found at  
<https://github.com/hbs-guidance/draft-hbs-state>.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 11 August 2026.

## Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. When are Stateful HBS Appropriate? . . . . .	4
2. Specific Terminology in the Context of Stateful HBS . . . . .	5
2.1. Private Key Components . . . . .	5
2.2. State Management . . . . .	6
2.3. Backup Management . . . . .	6
2.4. Key Export, Key Import and Key Transfer . . . . .	7
3. Operational Considerations . . . . .	7
4. Requirements for Secure State Management . . . . .	9
5. Potential Solutions . . . . .	11
5.1. Multiple Public Keys (SP-800-208) . . . . .	11
5.2. Distributed Multi-trees (SP-800-208) . . . . .	12

5.3. Sectorization . . . . .	13
5.4. Key/State Transfer . . . . .	14
5.5. Key Rotation . . . . .	14
5.6. Variable-length Signature Chains . . . . .	15
5.7. Pre-assigning States . . . . .	15
5.8. Time-based State Management . . . . .	16
5.9. Interval-based Approaches . . . . .	18
6. Backup Management Beyond NIST SP-800-208 . . . . .	18
7. Security Considerations . . . . .	21
8. IANA Considerations . . . . .	21
9. Informative References . . . . .	21
Acknowledgments . . . . .	23
Contributors . . . . .	23
Authors' Addresses . . . . .	23

## 1. Introduction

Stateful Hash-Based Signature Schemes (Stateful HBS) such as LMS, HSS, XMSS and XMSS<sup>MT</sup> combine Merkle trees with One-Time Signatures (OTS) in order to provide digital signature schemes that remain secure even when large-scale quantum computers become available. The theoretic security of Stateful HBS is well understood and depends only on the security of the underlying hash function. As such, Stateful HBS can serve as an important building block for quantum-resistant information and communication technology. Stateful HBS are specified in [RFC8391], [RFC8554], and NIST [SP-800-208].

The private key of a Stateful HBS is a finite collection of OTS keys (typically generated on-demand from a seed) and an associated data structure which keeps track of which OTS keys have been used. This data structure is typically a simple counter and often called an index; we refer to it as the *\*state\** of the private key. Each Stateful HBS private key can be used to sign a finite number of messages, and the state must be updated with each generated signature.

One must not reuse any OTS key that is part of a Stateful HBS private key. If an attacker is able to obtain signatures for two different messages created using the same OTS key, it is computationally feasible for that attacker to create forgeries [BH16] [Fluhrer23]. As noted in [MCGREW] and [ETSI-TR-103-692], extreme care should be taken in order to avoid the risk that an OTS key will be reused accidentally. Whereas [MCGREW] identifies the fundamental failure modes of Stateful HBS and proposes architectural strategies such as a reservation approach, and [ETSI-TR-103-692] provides a broad analysis of state management challenges and risks, this document complements both by cataloging concrete operational patterns in Section 5 and by addressing backup and recovery considerations Section 6 not covered in prior work.

In particular, the challenges below highlight why careful state and backup management are essential in Stateful HBS:

- \* Implementers must ensure that each creation of a signature updates the state correctly.
- \* If the Stateful HBS private key is distributed to multiple signers at the same time, implementers must ensure that they never use the same OTS key. This may require synchronization between all signers.
- \* Additional operational complexity arises when part of the available OTS signatures are allocated to different devices (partial state transfer), or when state from different devices needs merging; these introduce risks of overlap, failure, and require careful coordination.
- \* If key backups are required, implementers must ensure that any backup mechanism can not lead to re-using a previously used OTS key.

The following sections present, recall, and discuss various strategies for a correct state and backup management for Stateful HBS.

### 1.1. When are Stateful HBS Appropriate?

The issues with state management described above, as well as (for most parameter sets) the limited number of signatures, lead to new requirements that most developers will not be familiar with and that require careful handling in practice: Stateful HBS are not general-purpose signature schemes. Most applications, especially those that may produce unrestricted numbers of signatures, should use `_stateless_` hash-based signature schemes like SLH-DSA [FIPS205],

which use the same security assumptions, or schemes based on other assumptions, such as ML-DSA [FIPS204], instead. However, if run time, implementation size, or signature or key sizes of stateless alternatives are prohibitive, and the specific use case allows a very tight control of the signing environment, using Stateful HBS may be an appropriate solution. It seems likely that in many scenarios, it is only possible to meet the requirements set out in Section 4 when using purpose-designed hardware, such as hardware-security modules.

Stateful HBS are already profiled or discussed in several deployment-focused specifications and guidance documents. For example, [RFC9802] discusses suitable use cases for stateful HBS in X.509 (including firmware/software signing and CA certificates). The SUIT Mandatory-to-Implement algorithms specification [I-D.draft-ietf-suit-mti] defines an asymmetric profile that uses HSS-LMS, providing an interoperability target for software/firmware update IoT ecosystems. Additionally, the NSA [CNSA2.0] allows LMS (and XMSS) in specific application scenarios such as firmware/software signing.

## 2. Specific Terminology in the Context of Stateful HBS

In this section we specify certain notions which are important in the context of Stateful HBS.

### 2.1. Private Key Components

This section describes the two conceptual components that make up the private key material used in Stateful HBS.

**private key** the static, long-lived secret(s) from which OTS private keys are derived. This material is stateless: given the scheme parameters, it deterministically defines the set of OTS private keys but does not change over time.

**state** the dynamically updated data structure that records which OTS key indices have been consumed (often a monotone counter). This material is mutable and must change on every successful signature.

Conceptually, the private key and the state are distinct and should be handled accordingly: the private key is a static secret, while the state is mutable, evolves with each signature, and must be maintained with integrity and correctness. In some implementations, these two components may be packaged together and not directly separable; in such cases, this document 襄沔 guidance applies to the combined artifact.

## 2.2. State Management

In this document, `_state management_` refers to the handling and implementation of the state of the private key.

This includes mechanisms, which aim:

- \* to securely update the state before the signature is released,
- \* to set up Stateful HBS where the state is separated in distinct, non-overlapping parts, so that signatures can be generated from either part without risk of state reuse,
- \* to enable partial transfer of unused signature capacity between devices, and optionally merging state fragments without overlap,
- \* to enable effective but secure handling of private key and state backup material,
- \* to guarantee the availability of both the private key and its state across the lifetime of the key.

Note that in particular implementations of Stateful HBS, or in alternative signature mechanisms, the state and private key might be inseparable. For example, puncturable schemes [BSW16] represent such an alternative; they are research-level constructions and are not currently standardized or deployed in practice. However, even in these scenarios, this document's guidance should still apply.

## 2.3. Backup Management

In order to mitigate failure of, e.g., devices storing key material and to facilitate other types of disaster recovery, backups of private keys and their associated states should be considered as part of a security architecture.

In this document, `_backup management_` refers to all mechanisms surrounding the goal to guarantee the availability of the private key and state, but with special care to avoid state reuse by rolling back to a state in which already-used OTS keys are still available.

These mechanisms include procedures and protocols, which aim:

- \* to securely store this private key and state material outside the in-use signing device,
- \* to import an externally stored private key and state to a newly initiated signing device,

- \* allow practicing with backup recovery and to ensure backups are valid.

Backup management can be viewed as a more specific type of state management; we make this distinction to clarify the aims of our recommendations.

#### 2.4. Key Export, Key Import and Key Transfer

As part of state and backup management, we will discuss mechanisms to export, import or transfer private key and state material. In order to avoid misunderstandings we now specify these notions more precisely.

key export mechanism of exporting secret data, which yields (partial) private key and state material, from the signing device to external storage. This external storage may be given in digital or non-digital form.

key import mechanism of importing secret data, which loads (partial) private key and state material, from external storage to the signing device.

key transfer a cryptographically protected transfer of ownership of private key and state material from one signing device to another.

Systems and architectures relying on key transfer are generally expected to require fewer operational and manually-executed steps and checks to avoid state reuse.

Note that, at times, secure variants of the aforementioned primitives may be required (e.g., securely importing/exporting the key). In these situations cryptographic mechanisms should be utilized to provide assurances related to the confidentiality (e.g., utilizing symmetric/asymmetric encryption mechanisms) and/or integrity/authenticity (e.g., utilizing digital signatures, hash functions, and keyed message authentication codes) of the associated operations.

#### 3. Operational Considerations

An important aspect of the evaluation of various HBS state and backup management options is to consider the operational costs associated with the option(s) being evaluated. In the past, a traditional trust infrastructure solution could utilize straightforward archival procedures to make copies of the keys, which could then be distributed geographically to ensure their availability and deliver a sufficiently resilient solution, all the while enforcing whatever security protocols and procedures were required. Unfortunately,

Stateful HBS introduce an additional constraint in that they need to ensure the state is never re-used. Hence, archival procedures used for traditional trust infrastructures have to be amended/redesigned to be used as viable options.

One of the most problematic aspects of providing a long-lived resilient solution is simply managing the physical media on which the keys/state are stored externally (i.e., outside of the signing device) throughout the course of their lifetime. Physical media/devices degrade over time, and the more complex the media/device, the more likely it is to fail at some point in time (e.g., data stored on a CD vs. data stored on a USB drive vs. data stored in a Hardware Security Module). Combine that fact with the long lifetimes associated with Stateful HBS keys (e.g., 10-20+ years) and the difficulties associated with transferring keys between devices, and one finds them self with a perplexing set of challenges that needs to be accounted for in any state selection process of a proper state and backup management solution. Compounding these complexities is the fact any resilient state management system should also provide some means to verify the integrity of these long-lived backups to ensure they will be valid when they are required, and to ensure the operators know how to execute the necessary recovery procedure(s).

Similarly, many of the prescribed state management options require a high degree of operator involvement which means one should consider the costs associated with training the operator element to ensure processes and procedures are adhered to and failures caught early and corrected before a catastrophic loss of security can occur (e.g., accidentally instantiating multiple instances of a Stateful HBS key/state). Note that training is not a fixed one-time cost either as long lifetimes will necessitate succession planning amongst the operator element, and training of each successive generation of participants. Mechanisms also should be put in place to mitigate the ever-present insider threat via mechanisms such as M-of-N controls, ensuring least-privileges amongst participants, and enforcing a segregation of duties to ensure multiple parties are required to collude to undermine a solution's security. Note that the segregation of duties must persist across successive generations to ensure participants do not acquire multiple roles over time, thereby undermining the intended segregation.



In addition to the state management, implementers may consider implementing mechanisms to prevent abrupt signature exhaustion. Implementations may consider providing a configurable warning threshold, *M*, which is triggered when *M* signatures remain. When the number of available signatures reaches this threshold, the system should return a 'signatures nearing exhaustion' warning. This warning condition should require explicit acknowledgment from the user through a mechanism that cannot be trivially skipped.

Another important consideration in deploying Stateful HBS is the selection of an appropriate parameter set. Given the flexibility of these schemes 譬如 such as adjustable tree heights or Winternitz parameters — there exists a large variety of possible configurations. The availability of these different configurations offers many trade-offs between signature generation/verification time, signature size, and key generation time. Hence, careful attention during the design phase is essential to ensure that the chosen parameter set aligns optimally with the specific requirements of the intended use case.

Lastly, costs associated with any external dependencies required by a particular solution (e.g., access to a public ledger or transparency log, providing accurate time references and synchronization mechanisms, access to attestation facilities, etc.) must be accounted for as well, particularly if a system is operating in an offline mode that makes delivering these additional capabilities all the more complicated and expensive.

#### 4. Requirements for Secure State Management

A system deploying Stateful HBS should fulfill certain requirements to allow securely handling the state. The system must ensure that no two signing operations can ever be issued from the same state. In addition, the generation of a signature and update of the state should appear to be an `_atomic transaction_`. This means that the system must not release a signature without irrevocably and correctly updating the state.

State management systems should satisfy all `_ACID_` properties:

- \* `_Atomicity_`: each operation on the state must be indivisible — such that it either commits completely or leaves the state unchanged.
- \* `_Consistency_`: the state before and after each update must reflect a valid progression of available OTS indices, and no invalid or conflicting state is ever observable.

- \* \_Isolation\_: concurrent or overlapping operations (e.g., from separate processes or devices) must not interfere in ways that could lead to state reuse.
- \* \_Durability\_: once a state transition (e.g., before issuing a signature) is committed, that transition must survive crashes, power loss, or device failure.

These requirements impose significant restrictions on the underlying technical approach and a careful implementation of how the state will be updated or synchronized. The abstraction layers of modern systems can make it particularly difficult to guarantee that no two versions of the same state are present. The main concerns here are

- \* how the actual storage for the state is implemented,
- \* how it is modified,
- \* how an accidental/intentional failure/glitch might affect the state security.

A system may have a version of the private key stored in non-volatile memory (e.g. a disk) and will load it into volatile memory (e.g. RAM) while processing. Here, an implementer must ensure that these are always perfectly synchronized [MCGREW], meaning that no parts of the system are allowed to read any version of the key during procedures which load, write or modify keys. This can be particularly challenging if there are additional abstraction layers present in the system, like additional caches which may affect reading/writing the state and its potential existence in multiple locations.

Cloning is another concern, as it can easily lead to re-using the same state. This can happen for instance if a process which issues a signing operation is forked, and no proper synchronization is enforced in the implementation to guarantee correct state update. Virtual machine (VM) cloning is another potential security risk here, as both backing up a VM into non-volatile memory or live cloning of a VM can easily lead to a state re-use [MCGREW]. With users shifting workloads to cloud service providers, the issue of VM cloning may become more prevalent.

Using dedicated cryptographic hardware is recommended to enforce these requirements, ensure correct behavior and handle the complexity of state management. In particular, this enables implementing rollback resistant counters which can be difficult to achieve in a software-only fashion.

On the verifier side, no state management is required. However, the verifier needs to trust the signer to not have re-used the state. A verifier may want to check that no state re-use happened in the past by a signer, before accepting a signature.

In practice, this can be done if the verifier has access to all signatures issued by the signer. As the signatures contain the index of the OTS key used, detecting if an index was used more than once becomes trivial. In practice, such a (public) data structure which contains all signatures may already be present in some use cases (e.g. certificate transparency [RFC9162]) or could be built. It is worth noting that while trusting the signer to not re-use the state is a strong assumption, other signature schemes like ECDSA introduce similar assumptions for the verifier, by requiring the signer to never re-use the nonce.

## 5. Potential Solutions

A variety of potential solutions have been proposed both within the [SP-800-208] specification, as well as from external sources. This section describes a number of approaches and their potential advantages/disadvantages.

### 5.1. Multiple Public Keys (SP-800-208)

[SP-800-208] proposes generating multiple Stateful HBS keypairs and configuring devices and clients to accept signatures created by any of these keys. Secondary Stateful HBS keys can be kept in storage until the first keypair is exhausted or lost.

Accepting multiple public keys negatively impacts one of the advantages of using Stateful HBS by increasing the public key footprint within the client, which can be problematic if it has limited public key storage capacity. (Though public keys are typically equivalently sized to ECDSA rather than larger classical RSA keys often currently found.) [SP-800-208] addresses storage capacity concerns by suggesting using a mechanism such as that proposed in [RFC8649] to update the stored public key by having the current key endorse the next key that is to be installed. Unfortunately, for many constrained devices the public key is embedded in immutable ROM or fuses due to security reasons, so it cannot be updated in this manner.

The proposal of using multiple Stateful HBS keypairs for a single instance also generates questions as to how to establish that approach in existing public key infrastructures. For example, issuing multiple certificates adds the storage needs of the certificate material to the public key footprint. In order to

alternatively issue multiple public keys encoded inside a single certificate one would need a standardized format if interoperability is a concern.

## 5.2. Distributed Multi-trees (SP-800-208)

[SP-800-208] also proposes creating multiple Stateful HBS keys across multiple cryptographic modules using a distributed multi-tree approach that is a variant of the standard hyper-tree based Stateful HBS schemes HSS and XMSS<sup>MT</sup>. In this approach, trees are instantiated on a root device (HSM<sub>root</sub>), as well as one or more subordinate devices (HSM<sub>(sub<sub>{i}</sub>)</sub>), and the root tree is used to sign the root nodes of the subordinate trees to synthesize a multi-level Stateful HBS key. The root device is only ever used to sign subordinate device root nodes, while the subordinate device(s) is(are) used to sign messages. This is relatively straightforward to do using HSS, and [SP-800-208] describes the necessary algorithmic modifications when using XMSS<sup>MT</sup>.

One drawback of this approach is the increased signature size as an additional OTS needs to be generated, effectively doubling the overall signature size. Another concern is the single point of failure nature of relying on the root tree module to sign all of the subordinate trees; if the root tree device fails then no new subordinate trees can be signed. [SP-800-208] suggested that as many subordinate trees as possible be generated during the initial root key generation and subordinate-signing procedure. Unfortunately, this can incur a large capital expenditure to procure all of the necessary devices, many of which may not be used for a long period of time, if at all. The subordinate tree root node signing process must also be carefully managed to ensure top level trees are only ever used to sign the root nodes of trusted/approved subordinate trees to ensure that no malicious signing request is accepted, which would effectively give a rogue entity the ability to generate valid signatures, thereby undermining the security of the entire system.

[SP-800-208] also suggests combining distributed multi-trees with multiple root public keys as a means to mitigate some of the concerns regarding having a single point of failure in the root tree. However, even if a system operator does everything right, use cases with exceptionally long lifetimes of 10-20+ years (e.g., automotive and aerospace/satellite applications) will require system operators to rely on devices well beyond their expected lifetimes of 5-10 years, which may constitute an unacceptable business risk.

### 5.3. Sectorization

Distributed multi-trees attempt to partition a Stateful HBS signing space amongst multiple cryptographic modules by breaking up the signing space along the boundaries of the subordinate trees generated during the multi-tree key generation process. An alternative approach would be to use only a single tree, and partition its signature space along some power-of-2 less than the total number of leaves in the tree (e.g.,  $2^s$  for a tree of height  $h > s$ ), creating  $N = 2^{(h-s)}$  partitions or sectors, which are instantiated as  $N$  height- $s$  Merkle trees whose root nodes are considered interior nodes of the overall height- $h$  Merkle tree. Hence, there is no additional OTS required to sign their root nodes; their values are used as-is in the tree ascent mechanism of the underlying Stateful HBS scheme, yielding a common public key (i.e., root node) for all sectors. Care must be taken to ensure that each sector uses the same root tree identifier (i.e., the "I" value for HSS/LMS and "root" value for XMSS/XMSS<sup>MT</sup>).

Each of the  $N$  sectors' OTS private key values can be generated pseudo-randomly from a unique seed value generated from an appropriate source of randomness. The private keys from different sectors are independent when generated by this process. This requires that the path information for the root node of each sector (i.e., all off-path nodes between the sector root node and the top level node value) be stored with each sector's private key at key generation time since a sector will not know the seed information required to compute any of the other sectors' root nodes during the tree ascent phase of a signature generation operation. During signature generation the signer appends the stored path information to the path information it computes to ascend from the leaf OTS to the sector's root node (which it can compute given that it knows its own seed value).

Hence, sectorized key generation results in a single public key value and  $2^{(h-s)}$  private key values, each capable of generating  $2^s$  signatures, after which the sectorized key is exhausted.

In addition to avoiding an increased signature size; when unique seeds are utilized sectorization breaks a given Stateful HBS key/state into multiple independent fragments that can be managed as independent objects. As a result, system operators may distribute sectors to multiple cryptographic devices, providing scalability through parallelization and improved resiliency/availability. This approach offers isolation between sectors, ensuring that a compromise in one does not extend to others, thereby supporting damage containment. At the same time, it simplifies operational robustness by removing the need for cross-device state coordination, since each sector is restricted to its own signature space.

#### 5.4. Key/State Transfer

Stateful HBS key/state transfer between cryptographic modules entails having a means to migrate one instance of a Stateful HBS key/state on a source device to a separate destination device, while ensuring that any copy of the key/state is deleted from the source device.

This capability may help alleviate the aforementioned concern regarding operating devices beyond their expected lifetimes by allowing operators to migrate Stateful HBS key/state to a newer device when the original device begins to approach its end-of-life. However, it still leaves the operator vulnerable to having the source device fail before the key/state can be transferred, effectively causing the loss of the key/state. Hence, it will not be of much help addressing the single point of failure issue identified for root trees, but may be useful for managing subordinate trees.

In addition to complete key/state transfer, a device holding part of the total available OTS signatures may transfer some unused capacity to another device (partial state transfer). In more advanced deployments, state fragments from two devices may be merged to reconstruct or continue signature operations. These operations carry risk: ensuring no overlap in used indices, ensuring atomicity of transfer/merge operations, managing consistency, possible conflicts, and durability of state across devices. Such approaches require robust synchronization, auditability, and appropriate backup mechanisms to avoid double-signing or loss of capacity.

A more elaborate variant of key transfer, going beyond what [SP-800-208] allows, can be found described in Section 6 where key transfer is accomplished using a two-step export and import process with hash-based transfer validation to yield a more robust transfer mechanism.

#### 5.5. Key Rotation

Key rotation, such as that defined in [RFC8649], would generate new Stateful HBS keys on an as-needed basis, and provide a means to transition the system on to using this new Stateful HBS key, while generating the next key in the chain in preparation of a future rotation/update. However, this just shifts the problem to the PKI and certificate handling.

Key rotation is not foolproof since in most use cases it will require redundancy to ensure there is at least one Stateful HBS signing key available to attest to newly generated keys. In addition, for many applications the device keys cannot be updated due to engineering constraints or security reasons.

## 5.6. Variable-length Signature Chains

A variant of the key rotation approach is to have an available signing tree endorse a new subordinate tree when it is about to become exhausted (e.g., use its final OTS to sign the root node of a new subordinate tree, creating a  $\{n+1\}$ -layer multi-tree from an  $\{n\}$ -layer multi-tree). This process can in theory be repeated as many times as necessary. However, this entails having a multi-tree scheme with a variable number of levels, and hence, variable length signatures. Such dynamically extensible constructions are research-class and are not currently standardized or deployed.

In addition to departing quite significantly from the current Stateful HBS specifications and [SP-800-208], this approach has a number of significant challenges on both the engineering and operational fronts. Firstly, the variable length nature of the signature can lead to variable length verification of signatures, which may cause significant issues for use cases with strict time constraints (e.g., secure booting of a semiconductor device). From an operational perspective, the ability of a subordinate tree to sign either messages or new subordinate trees leads to severe security implications as the rigor around authorizing those two types of operations will vary dramatically, leading to either a much more onerous message signing operation, or a much more risky subordinate tree signing operation. This may put the system operator in an untenable situation where no users are satisfied with the resulting solution, and hence, should not be considered as a viable solution.

## 5.7. Pre-assigning States

In some applications, individual one-time signatures (or states) can be pre-assigned to the to-be-signed objects. This may for example be possible if the signed objects are monotonically increasingly numbered. One example of such a use case may be software signing. This solution basically externalizes the state management to the to-be signed messages.

Expanding on the given example, for software that is released with strictly increasing, simple single-position version numbers (i.e., versions 1, 2, 3...), this can be trivially implemented. As versions have a one-to-one correspondence to a Stateful HBS signing state, operators must ensure that versions can only be minted a single time. This may require skipping version numbers if a release process failed, to avoid double-signing.

This scheme can be adapted to more complicated release schemes: for example, minor update-releases 1.0 to 1.99 can be accommodated by assigning signatures 1-100 for these version numbers, while release

2.0-2.99 would get signatures 101-200. The assignments must be fixed as the scheme is set up, and operators should take into account that they are strictly limiting the number of update releases. In the described solution to state management, one must move up a major release number after 99 minor releases, even if this would break, e.g., semantic versioning conventions.

A variant of pre-assigning signatures is doing this on the basis of time, which we describe in the next section.

#### 5.8. Time-based State Management

As a variant of pre-assigning one-time signatures based on external counters, it is in theory possible to base the selection of one-time signature indexes on the current date and time. For example, if a given Stateful HBS instance offers 1024 total signatures, they could be broken up into 8 groups of 128 OTS instances each, with the first 128 allowed to be used in the first time window, the second 128 in the second time window, and so on, until the signature space is effectively exhausted after 8 time windows. Note that a time-based approach to state management will "waste" any OTS keys that are unused in past time windows. One must not attempt to use these keys after the time window has gone by.

Any time-based approach has a very strict reliance on accurate time-keeping and synchronization of clocks. In particular, we identify that at least the following engineering-related challenges need to be considered:

- \* Signing devices must have accurate timekeeping (which is a very challenging engineering problem [TIMEFALSEHOODS]).
- \* Time on signing devices must not be allowed to ever move backwards, as this can cause double-signing.
- \* Within time windows, signers must track the number of signatures produced to ensure it does not exceed the number allowed within the window.
- \* Signing devices must still operate consistently with the requirements of state keeping for Stateful HBS: the signature index within a time window should still appear to be updated atomically, and signatures must not be released before state changes have been recorded.



- \* A system should be robust against exhaustion of the number of signatures available in a time window, as in this case it is required to wait until the next time window starts before new messages can be signed.
- \* Time on signing devices should not be allowed to be moved forward maliciously or accidentally, which would allow for a simple denial-of-service attack by skipping over portions of the signature space.
- \* If a signing device needs to be replaced, the replacement device must be set up with its time in sync with or ahead of the device it is to replace. This implies the current time on signing devices should be continuously recorded.
- \* Rate limiting may need to be considered, as exhausting the available signatures in a given time window may otherwise be easy.
- \* It may be necessary for signers to keep a separate clock for time-based state management, and one for not necessarily monotonically increasing "wall-time", e.g., if signed artifacts are expected to be time-stamped with real-world time.

If these concerns can not be sufficiently addressed, time-based state management as described in this paragraph should not be used. Note that this list of concerns is not exhaustive, and other, unmentioned, concerns may also be relevant to the security of a time-based solution.

Time-based systems can be backed up by simply recording the private keys and the configuration of the time windows. In case of loss of a signing device, a time-based state management system can be recovered by using this information to bring online a new device in the next time window. This approach may also be used as a recovery mechanism in the case of (suspected) state consistency problems during a time window. However, the operator must not allow new signatures to be produced before the new time window starts, unless they know the exact state at which the previous device became unavailable and are able to set up the new device accordingly. Waiting until the start of the next time window avoids double signing, as the OTS keys assigned to future time windows are guaranteed to have not yet been used. However, this might incur significant downtime of the signing systems. Downtime may be avoided by forcibly moving the signing device to the next time window by incrementing its clock; however, this induced clock drift will then need to be accounted for in the future. If clock drift is to be avoided, this approach should account for availability considerations.

### 5.9. Interval-based Approaches

The State Reservation Strategy described in section 5 of [MCGREW] provides another means of managing the state by allowing users to reserve intervals of the signing space, marking the interval's associated OTS keys as being used in the overall HBS state, which is then written back to non-volatile memory prior to their usage. The OTS keys within the reservation interval are then consumed as-needed, without having to update the state again until they have all been consumed and additional OTS keys are required. Note that the reserved OTS keys are kept in dynamic memory so they will be lost if the signing device loses power or is reset, resulting in a reduction in the number of usable signatures for a given HBS instantiation.

Over provisioning can be used to ensure a sufficient number of signatures can be provided in the presence of unexpected losses due to power loss or resets. Over provisioning will cause a minor increase between 2% and 12% on signature length as the MTS validation paths increase to accommodate the increased Merkle tree height. However, reservation eliminates the need to update the state after each OTS key is used, minimizing the likelihood of state reuse due to state update failures and coherency issues.

Multiple signing devices can in theory utilize reservation intervals to carve out portions of signing space so that a single S-HBS key can be shared amongst multiple devices, leading to potential performance and disaster-recovery benefits. However, great care must be taken to manage the reservations to ensure there is no overlap or repeated reservation of a given interval, either in part or in whole.

## 6. Backup Management Beyond NIST SP-800-208

In this section, an alternative backup mechanism for Stateful HBS is presented in a generic form, which makes the strategy applicable for both multi-tree instances XMSS<sup>MT</sup> and HSS. However, following the same arguments as in Section 5.3, with minor modifications, the presented strategy is also applicable for single-tree instances such as XMSS and LMS.

The strategy presented in this section builds upon the multi-tree variant approach from [SP-800-208], and aims to mitigate its limitations described in Section 5.2. Thus, it is assumed that already a top-level Merkle tree (for signing the root-nodes of sub-trees) and several bottom-level Merkle trees (for signing messages) are initiated. These bottom-level trees may be implemented on different hardware modules in order to obtain redundancy and improve availability. Let  $R$  be the number of these already initiated bottom-level trees. Let  $h_0$  be the height of the top-level-tree. It is assumed that  $R + 1$  is strictly smaller than  $2^{(h_0)}$ , the number of leaves of the top-level tree.

In this new strategy, after the completed key generation procedure from the multi-tree variant approach from [SP-800-208], further bottom-level trees are generated, one by one, in one of the hardware modules. These new bottom-level trees are each generated from a different seed, which is chosen uniformly at random. For the sake of clarity, let us introduce some notation:

- \*  $S$  denotes the number of these newly generated bottom-level trees. Note that at most  $2^{(h_0)} - R$  new bottom-level trees can be generated, i.e.  $S$  is lower or equal to  $2^{(h_0)} - R$ . In the following we suppose that  $S$  is strictly smaller than  $2^{(h_0)} - R$ .
- \*  $I_{\text{new}}$  denotes the set of indices that belong to these newly generated bottom-level trees, i.e.  $I_{\text{new}} = \{R, R+1, \dots, R+S-1\}$ .  $I_{\text{new}}$  is zero-indexed here.

For each new bottom-level tree, after it has been generated, the following steps must be performed:

- \* sign the corresponding root node with an unused OTS key from the top-level tree,
- \* securely key export (as described in Section 2.4) the seed, which was used to generate the bottom-level tree,
- \* export the signature of the root node, the corresponding OTS key index and finally the hash of the seed, using appropriate domain separation (i.e. ensuring there is no domain overlap with the hashes in the Stateful HBS scheme, and the hash of the seed includes the public key and leaf index to mitigate multi-target attacks),
- \* irreversibly delete the seed and the bottom-level tree from the hardware module.

The newly generated bottom-level trees (i.e. those bottom-level trees, whose indices belong to `I_new`) are only used in order to guarantee availability in the `_worst-case scenario_`, where at the same time both

- \* none of the `R` bottom-level Merkle trees (which were generated according to the multi-tree variant approach from [SP-800-208]) are available for signing messages and
- \* the top-level Merkle tree (which is used for signing the root-nodes of sub-trees) is also not available anymore.

This scenario may, for example, happen if all hardware modules are broken at the same time.

As soon as this worst-case scenario occurs, the newly generated bottom-level trees (i.e. those bottom-level trees, whose indices belong to `I_new`) need to be initiated in order to ensure availability. In order to do this the following steps must be performed:

- \* initiate a new hardware module
- \* securely `_key import_` (as described in Section 2.4) the first unused seed into this hardware module
- \* generate the bottom-level tree corresponding to the seed
- \* irreversibly delete the seed from the backup medium
- \* perform a correctness check by letting the hardware module output the hash of the seed

Now this bottom-level tree can be used to sign messages. As soon as no more OTS on the bottom-level tree are available or as soon as the hardware module is broken, the above steps with a new seed from the backup medium can be repeated.

Note that the resulting signatures generated from these backed up seeds do not require any special processing on the verifier side. The signature stored alongside the backed up seed, and the signature generated from the bottom-level trees created from the backed up seed can be combined to match the format of a signature over the complete tree.

## 7. Security Considerations

Security considerations are given throughout this document. Further security considerations, which are not already covered in this document, are given in [SP-800-208], [MCGREW], [FIPS205], [RFC8391] and [RFC8554].

## 8. IANA Considerations

This document has no IANA actions.

## 9. Informative References

- [BH16] Bruinderink, L. and A. Hlsing, "Oops, I did it again Security of One-Time Signatures under Two-Message Attacks.", 2016, <<https://eprint.iacr.org/2016/1042.pdf>>.
- [BSW16] Bellare, M., Stepanovss, I., and B. Waters, "New Negative Results on Differing-Inputs Obfuscation", n.d., <[https://link.springer.com/chapter/10.1007/978-3-662-49896-5\\_28](https://link.springer.com/chapter/10.1007/978-3-662-49896-5_28)>.
- [CNSA2.0] National Security Agency (NSA), "Commercial National Security Algorithm Suite 2.0 (CNSA 2.0) Cybersecurity Advisory (CSA)", 7 September 2022, <[https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA\\_CNSA\\_2.0\\_ALGORITHMS\\_.PDF](https://media.defense.gov/2022/Sep/07/2003071834/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS_.PDF)>.
- [ETSI-TR-103-692] European Telecommunications Standards Institute (ETSI), "State management for stateful authentication mechanisms", November 2021, <[https://www.etsi.org/deliver/etsi\\_tr/103600\\_103699/103692/01.01.01\\_60/tr\\_103692v010101p.pdf](https://www.etsi.org/deliver/etsi_tr/103600_103699/103692/01.01.01_60/tr_103692v010101p.pdf)>.
- [FIPS204] National Institute of Standards and Technology, "FIPS 204: Module-Lattice-Based Digital Signature Standard", Federal Information Processing Standards , 13 August 2024, <<https://doi.org/10.6028/NIST.FIPS.204>>.
- [FIPS205] National Institute of Standards and Technology, "FIPS 205: Stateless Hash-Based Digital Signature Standard", Federal Information Processing Standards , 13 August 2024, <<https://doi.org/10.6028/NIST.FIPS.205>>.
- [Fluhrer23] Fluhrer, S., "Oops, I did it again revisited; another look at reusing one-time signatures", 2023.

- [HBSX509] Bashiri, K., Fluhrer, S., Gazdag, S., Van Geest, D., and S. Kousidis, "Internet X.509 Public Key Infrastructure: Algorithm Identifiers for Hash-based Signatures", n.d., <<https://www.ietf.org/archive/id/draft-gazdag-x509-hash-sigs-02.html>>.
- [I-D.draft-ietf-suit-mti]  
Moran, B., Rnningstad, O., and A. Tsukamoto,  
"Cryptographic Algorithms for Internet of Things (IoT) Devices", Work in Progress, Internet-Draft, draft-ietf-suit-mti-23, 22 July 2025,  
<<https://datatracker.ietf.org/doc/html/draft-ietf-suit-mti-23>>.
- [MCGREW] McGrew, D., Kampanakis, P., Fluhrer, S., Gazdag, S., Butin, D., and J. Buchmann, "State Management for Hash-Based Signatures", Security Standardization Research 2016 , 2 November 2016,  
<<https://eprint.iacr.org/2016/357.pdf>>.
- [RFC8391] Huelensing, A., Butin, D., Gazdag, S., Rijneveld, J., and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme", RFC 8391, DOI 10.17487/RFC8391, May 2018,  
<<https://www.rfc-editor.org/rfc/rfc8391>>.
- [RFC8411] Schaad, J. and R. Andrews, "IANA Registration for the Cryptographic Algorithm Object Identifier Range", RFC 8411, DOI 10.17487/RFC8411, August 2018,  
<<https://www.rfc-editor.org/rfc/rfc8411>>.
- [RFC8554] McGrew, D., Curcio, M., and S. Fluhrer, "Leighton-Micali Hash-Based Signatures", RFC 8554, DOI 10.17487/RFC8554, April 2019, <<https://www.rfc-editor.org/rfc/rfc8554>>.
- [RFC8649] Housley, R., "Hash Of Root Key Certificate Extension", RFC 8649, DOI 10.17487/RFC8649, August 2019,  
<<https://www.rfc-editor.org/rfc/rfc8649>>.
- [RFC9162] Laurie, B., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", RFC 9162, DOI 10.17487/RFC9162, December 2021, <<https://www.rfc-editor.org/rfc/rfc9162>>.
- [RFC9802] Van Geest, D., Bashiri, K., Fluhrer, S., Gazdag, S., and S. Kousidis, "Use of the HSS and XMSS Hash-Based Signature Algorithms in Internet X.509 Public Key Infrastructure", RFC 9802, DOI 10.17487/RFC9802, June 2025,  
<<https://www.rfc-editor.org/rfc/rfc9802>>.

[SP-800-208]

Cooper, D., Apon, D., Dang, Q., Davidson, M., Dworkin, M.,  
and C. Miller, "NIST SP 800-208: Recommendation for  
Stateful Hash-Based Signature Schemes", NIST Special  
Publication , October 2020,  
<<https://doi.org/10.6028/NIST.SP.800-208>>.

[TIMEFALSEHOODS]

Vise, T., "Falsehoods programmers believe about time",  
n.d., <[https://gist.github.com/timvisee/  
fcda9bbdff88d45cc9061606b4b923ca](https://gist.github.com/timvisee/fcda9bbdff88d45cc9061606b4b923ca)>.

#### Acknowledgments

This document was inspired by discussions at the 2nd Oxford Post-Quantum Cryptography Summit 2023.

We gratefully acknowledge Melissa Azouaoui for her input to this document.

The abstract and the introduction are based on the introduction in [HBSX509]. Thanks go to the authors of this document. "Copying always makes things easier and less error-prone" - [RFC8411].

#### Contributors

- \* Jeff Andersen (Google)
- \* Bruno Couillard (Crypto4A Technologies)
- \* Stefan-Lukas Gazdag (genua GmbH)

#### Authors' Addresses

Thom Wiggers  
PQShield  
Email: [thom@thomwiggers.nl](mailto:thom@thomwiggers.nl)

Kaveh Bashiri  
BSI  
Germany  
Email: [kaveh.bashiri.ietf@gmail.com](mailto:kaveh.bashiri.ietf@gmail.com)

Stefan Klbl  
Google  
Switzerland

Email: kste@google.com

Jim Goodman  
Crypto4A Technologies  
Canada  
Email: jimg@crypto4a.com

Stavros Kousidis  
BSI  
Germany  
Email: kousidis.ietf@gmail.com