State Management for Hash-Based Signatures

David McGrew[1], Panos Kampanakis[1], Scott Fluhrer[1],
Stefan-Lukas Gazdag[2], Denis Butin[3], and Johannes Buchmann[3]

[1] Cisco Systems, USA
{mcgrew,pkampana,sfluhrer}@cisco.com
[2] genua GmbH, Germany
stefan-lukas_gazdag@genua.eu
[3] TU Darmstadt, Germany
{dbutin,buchmann}@cdc.informatik.tu-darmstadt.de

Abstract. The unavoidable transition to post-quantum cryptography requires dependable quantum-safe digital signature schemes. Hash-based signatures are well-understood and promising candidates, and the object of current standardization efforts. In the scope of this standardization process, the most commonly raised concern is statefulness, due to the use of one-time signature schemes. While the theory of hash-based signatures is mature, a discussion of the system security issues arising from the concrete management of their state has been lacking. In this paper, we analyze state management in N-time hash-based signature schemes, considering both security and performance, and categorize the security issues that can occur due to state synchronization failures. We describe a state reservation approach that loosens the coupling between volatile and nonvolatile storage, and show that it can be naturally realized in a hierarchical signature scheme. To protect against unintentional copying of the private key state, we consider a hybrid stateless/stateful scheme, which provides a graceful security degradation in the face of unintentional copying, at the cost of increased signature size. Compared to a completely stateless scheme, the hybrid approach realizes the essential benefits, with smaller signatures and faster signing.

Keywords: Post-quantum cryptography, Hash-based signatures, Statefulness, System integration

1 Introduction

Security protocols routinely rely on digital signatures for authentication. Common examples are code signing for software updates, server authentication for TLS, and S/MIME for secure email. The most common cryptographic schemes for digital signatures (RSA [26], DSA [8], and ECDSA [17]) are all susceptible to quantum computer cryptanalysis using Shor's algorithm [28]. While the concrete realization of quantum computers still is an object of ongoing research, substantial efforts in this area are ongoing [23, 27].

Independently of the actual realization of quantum computing, governmental and standardization organizations are encouraging the transition to post-quantum cryptography, i.e. cryptographic schemes not known to be vulnerable to quantum computer attacks. Notably, the NSA recently announced its transition from the Suite B set of cryptographic algorithms towards post-quantum cryptography [16]. Standardization efforts are also underway, for instance by NIST [6] and ETSI [9]. As more stakeholders are required to heed official recommendations, the deployment of post-quantum cryptography becomes inevitable.

Fortunately, post-quantum cryptographic schemes exist, and some are already well-understood. In particular, hash-based signatures have been thoroughly analyzed [2, 3, 7, 13, 19, 22] and are currently undergoing standardization [14, 21].

Motivation One major obstacle to the widespread use of hash-based signatures is the fact that the signing algorithm is stateful. That is, with each message being signed, the private key must change. There are stateless hash-based signature schemes [1, 12] that address state concerns, but their signature sizes are significantly higher. Tab. 1 shows the signature size of two stateful schemes (LMS [21], XMSS^{MT} [15]) and stateless SPHINCS [1] for parameters that lead to 2^{60} signed messages and a security level of 128-bits. For LMS we are considering LMS_SHA256_N32_H10 with LMOTS_SHA256_N32_W16 with a supertree height of $l = 6$. For XMSS, we use XMSSMT_SHA2-256_M32_W16_H60_D6 and XMSSMT_SHA2-256_M32_W16_H60_D12 with $n = 32$ respectively. For SPHINCS, the parameters $h = 60$, $d = 12$, $w = 16$ and $n = 32$ are considered. It is clear that stateless SPHINCS has significantly larger signatures, which could make it impractical in some scenarios. The public key size of the stateful schemes are in the range of 70B. The SPHINCS public key is in the 1000B range.

	Stateful		Stateless
	LMS	XMSS^{MT}	SPHINCS
Signature size (KB)	5	15 | 28	41

Table 1. Signature sizes (in KB) of hash-based signature schemes (128-bit security level, 2^{60} messages).

In a stateful scheme, when a private key is long-lived, it must be stored in nonvolatile memory, and the version of the private key in memory must continuously be synchronized with that in volatile memory (e.g. Random Access Memory, or RAM). State synchronization is especially important because it is critical to the security of the system; if two distinct messages are signed with the same private key, then an attacker can use those signatures to construct a forgery. Thus, after signing one message, the signer must update the state so the same key is not reused. Key synchronization also requires a time delay between signatures that can lead to a significant performance penalty.

In this paper, we consider the design of N-time signature schemes and the system engineering considerations needed to ensure that they avoid the problems outlined above. To the best of our knowledge, no work addressing state management strategies for stateful hash-based signatures exists so far. We describe a simple state management scheme for hierarchical signature schemes that minimizes synchronization delay and reduces the chance of synchronization failure. It works by storing only the root (or topmost) level(s) of the signature hierarchy in nonvolatile storage, and having the remaining levels exist only in volatile memory. However, this scheme does not address the cloning problem, which occurs when a system, e.g. a virtual machine may be copied and reveal sensitive cryptographic data; to address that point, we then consider a hybrid scheme: a hierarchical signature scheme in which the root level consists of a stateless N-time scheme, while the other levels are stateful.

Outline The remainder of this paper is organized as follows. We recall one-time signature schemes as well as N-time and hierarchical signature schemes, and establish some notation (Sec. 2). We then review the security and performance issues with stateful signature schemes, notably the impact of volatile and nonvolatile cloning (Sec. 3). Next, we cover the basics of synchronization between volatile and nonvolatile storage (Sec. 4), including possible mitigation for caching mechanisms, illustrated with concrete examples. We proceed by describing a strategy for loosening the coupling between those data stores by having the signer 'reserve' the state needed to sign multiple messages (Sec. 5). We show that hierarchical signature schemes naturally support this state reservation strategy, in a way that benefits security and performance. These techniques provide the best possible security for stateful signature schemes, but they do not address scenarios in which a private key may be unintentionally cloned. To address those cases, we outline a hybrid approach, in which the root level scheme of a hierarchical scheme is stateless — ensuring inadvertent nonvolatile cloning does not compromise the whole structure — but the lower levels are stateful (Sec. 6). We then offer our conclusions and sketch future work (Sec. 7).

2 Stateful Hash-Based Signature Schemes

One-time signature schemes Hash-based signatures use one-time signature schemes as a fundamental building block. One-time signature schemes, unlike most other signature schemes, require only a secure cryptographic hash function and no other hardness assumption (about a number-theoretic problem) and are not known to be vulnerable to Shor's algorithm. *Secure* here refers to either collision resistance or mere second-preimage resistance, depending on the specific one-time signature used. Common examples of one-time signature schemes are the seminal one by Lamport [19], the Winternitz scheme [7], and its recent variant W-OTS$^+$ [13]. In one-time signatures, the private key is usually randomly generated and the public key is a function of the private key, involving the underlying hash function. Advanced one-time signature schemes feature a parameter

enabling a time/memory trade-off, e.g. the Winternitz parameter. These schemes are inadequate on their own in practice, since each one-time private key can only be used to securely sign a single message.

N-time signature schemes Stateful N-time signature schemes, introduced by Merkle [22] and often improved since (e.g. [2, 3, 20]), are built out of one-time signature schemes. They make one-time signatures practical by combining $N = 2^H$ of them in a single structure — a complete binary tree of height H. Let $\Upsilon_N = (K, S, V)$ denote an N-time signature scheme that consists of a key generation algorithm K, a signing algorithm S, and a signature validation algorithm V. The private key Ψ for the N-time scheme Υ_N consists of the set of N private keys of the underlying one-time scheme. An integer counter has to keep track of the advancement of the key, as with each signature generation another one-time private key has to be used. A simple way to reduce the size of that N-time key is to instead define it to be a short string, and then use a cryptographically secure pseudorandom function to generate the keys of the underlying one-time scheme [22]. Formally speaking, this strategy changes the algorithms K and S by creating an additional preprocessing step. The private state is thus reduced down to the short private key and the integer counter, simplifying state management somewhat through a reduction in scale. Nonetheless, the correct management of the counter across multiple invocations of S is critical to security, as any one-time private key must not be used twice.

Hierarchical Signatures A *hierarchical signature scheme* is an N-time signature scheme that uses other hash-based signatures in its construction. Let $\Gamma = (\Upsilon_{N_0}, \Upsilon_{N_1}, \ldots, \Upsilon_{N_{l-1}})$ denote a hierarchical signature scheme with l levels. The public key Z for Γ is the output of K_0 (that is, the key generation algorithm of the top level). The private key for Γ consists of the private keys of each level: $\Psi_0, \Psi_1, \ldots, \Psi_{l-1}$. A signature for Γ consists of the public keys $Z_1, \ldots, Z_{l-1}$ of levels 1 through $l-1$, along with the signatures $Y_0(Z_1), Y_1(Z_2), \ldots, Y_{l-2}(Z_{l-1})$ of the i^{th} level's public key by the $(i-1)^{th}$ level private key, and the signature $Y_{l-1}(M)$ of the message M with the private key of the last level. If the signature scheme Υ_{N_i} at the i^{th} level of Γ can sign N_i signatures, then Γ is an N-time signature scheme with $N = \prod_{i=0}^{l-1} N_i$. Hierarchical signatures allow for shorter signing time of a message M by a N_{l-1} -time (instead of $\prod_{i=0}^{l-1} N_i$-time) signature while offering a higher total number ($\prod_{i=0}^{l-1} N_i$) of signed messages. Concrete examples of hierarchical hash-based signatures include XMSS^{MT} [15], a scheme by Leighton and Micali [20] and SPHINCS [1]. XMSS^{MT} and SPHINCS define parameter d as the number of layers in the hierarchical structure. Additionally, the LMS [21] specification describes a hierarchical hash-based signature variant based on Leighton and Micali's scheme.

3 State Synchronization Security Risks

We identified several distinct issues with stateful signature schemes, which we consider here. Ensuring the correct synchronization of the private key between a

storage unit and the execution unit requires a carefully engineered system. In most cases, the synchronization cost is likely to add to the time required for signature generation. We call this additional latency the *synchronization delay*. We say that a *synchronization failure* occurs if the private key or counter in nonvolatile storage fails to advance at or before the time that the private key in RAM advances. This could be caused by the crash of an application or an operating system, by a power outage, by the corruption of the nonvolatile state, or by a software bug. Another issue with statefulness is the *cloning problem*: the situation arising when a private key is copied and then used without coordination by two or more execution units. This can for instance happen through Virtual Machine (VM) image cloning, or by the restoration of a key file to a previous state from a backup system. Cloning will cause multiple signatures to be generated from the same system state, thus undermining security. The most important issue is *nonvolatile cloning*, as outlined above. A related but distinct problem is that of *volatile cloning*, which is the copying of a private key from volatile memory, as discussed below.

In a well-designed stateful signature system, synchronization failure can be avoided, but also synchronization delay will greatly impact performance. Nonvolatile cloning may not be a consideration on a system that is dedicated to signature generation, but it is a significant risk on general-purpose software systems.

Volatile cloning in VM environments There is a low risk of volatile cloning, except in VM environments. Most contemporary VM environments (e.g. VMware Virtual Center, Oracle Virtual Box, KVM or Xen) support live cloning. This is usually achieved by first capturing a system snapshot. Such environments introduce several other risks for cryptography and security protocols. Pseudorandom number generators are at risk, since their state can be cloned, too. The exact risk to pseudorandom number generation depends on the method of entropy collection (possibly causing state divergence), but the risk is great enough that volatile cloning should be avoided. Generally speaking, values that may only be used once are at risk from live VM cloning. In addition to random numbers, this includes "initialization vectors, counters for encryption, seeds for digital signatures, and one-time passwords" [10]. Issues with such primitives can result in catastrophic failures for classic (pre-quantum) digital signature schemes such as DSA, and even on the level of security protocols such as TLS [25]. These vulnerabilities relate to random number generators caching randomness far in advance. Many other systems are at risk from live virtual machine cloning, including the S/KEY one-time password system and the TCP protocol [11], for which initial sequence numbers could be reused for hijacking. Volatile cloning therefore appears problematic to such an extent that the vulnerability of hash-based signature to this scenario is by no means a special case.

In our view, stateful hash-based signature techniques can be safely implemented in some scenarios, such as dedicated cryptographic hardware, which can then benefit from their smaller signatures and signing times. On the other hand, there is also a need for techniques that are secure even when nonvolatile cloning

occurs, for use in general purpose software environments. The hybrid approach of Sec. 6 addresses the latter need.

4 State Synchronization Considerations

State synchronization can be delayed by caching, or interfered with by other processes on the used system. Of course, it is neither possible nor reasonable to avoid caching and therefore the resulting delay completely. Multiple levels of caches coexist on typical systems, both in hardware and software, helping the system to function quickly and properly, by making frequently used data easily accessible. Nevertheless, partially mitigating measures can be applied by forcing cache flushing, or by deactivating it. As dealing with the state synchronization is highly dependent on several factors like the used operating system or hardware specifics, no universal approach exists. In the following, we present diverse options to cope with state synchronization.

The access to nonvolatile storage deserves more detailed consideration as the essential operation in our case is storing the new key state held in the main or volatile memory on that nonvolatile storage (as shown in Fig. 1). To minimize I/O operations e.g. on hard disks for frequently used or written data, caches provide recently used or repeatedly accessed data based on prior operations. So data supposed to be written to the disk may stay in the cache rather than being written to the actual disk memory. Hard disks and other nonvolatile storage devices often feature a (disk) cache that holds the data before actually writing to disk. Most modern operating systems offer a page cache which behaves similarly to the memory caches of hard disks but remains in the main memory of the system. Usually data that is read or written for the first time is held in the page cache which gets synchronized to the disk or disk cache in fixed intervals, by exceeding thresholds e.g. a limit for non-synchronized data or due to system calls. Data sometimes can be swapped between both caches. In most cases the use of those caches is transparent and therefore unknown to applications running in userspace. In matters of key state synchronization, a write-through is ideal. In such a scenario using stateful schemes it is desirable to use state-of-the-art techniques to overwrite the sections of the main memory with random values if they hold cryptographic data that is not needed after writing to disk. Mechanisms that come with the operating system (and may be applied automatically) should be preferred over self-made implementations which need special attention i.e. regarding compiler optimizations.

In terms of file I/O, on POSIX environments, the `O_SYNC` flag (for the `open()` system call) will cause invocations of the `write()` system call to block the calling process until the data has been written to the underlying hardware. Equivalent flags (e.g. `FILE_FLAG_WRITE_THROUGH` for `CreateFile()` on Win32 API) exist on other platforms. Modern Linux systems offer writeback or flusher threads (formerly `pdflush`) for the page cache that may be optimized. Synchronization may be triggered manually by the `sync` or `fsync` system calls. However, if hardware has its own memory cache, it must be separately dealt with using an

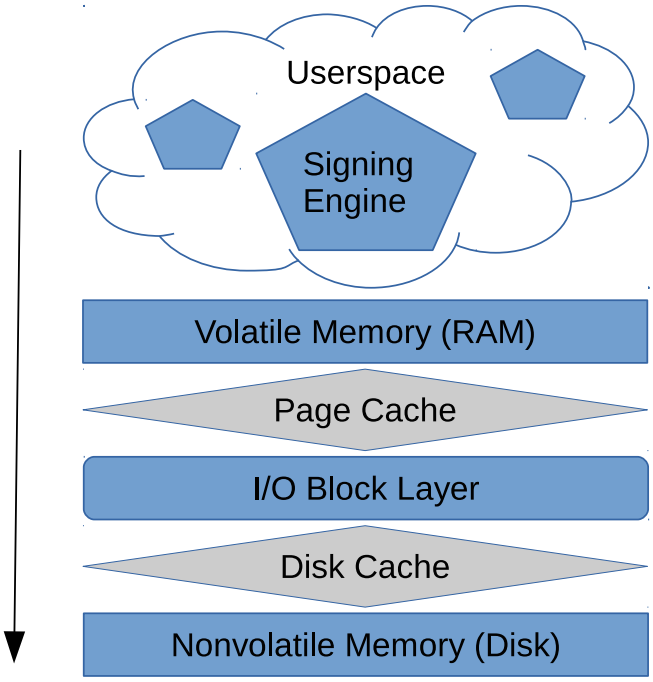

Fig. 1. A simplified view of the data flow from signing engine to nonvolatile storage, via cache layers.

operating system- or device-specific tool such as `hdparm` to flush the on-drive cache, or to deactivate write caching for that drive. The secure way to ensure that the data was written is to read it again and compare it to what was expected, which requires the data to be written to disk without cache interruption and would add to the synchronization delay.

4.1 Overhead for hash-based signatures

The synchronization latency introduced earlier depends on the chosen hash-based signature scheme and its implementation. A typical example following the specification of XMSS [14] merely requires updating the leaf index in the secret key, a 4-byte value; or an 8-byte value for XMSS^{MT}. This assumes that all data including the one-time keys and all nodes are stored in the global secret key. The LMS [21] specification also requires updating a 16 or 32-byte random private key value. Of course, generating 16 or 32-byte values can be avoided by generating them from a seed and index with a pseudorandom number generator (PRNG), which would reduce the state update of a few byte long index.

A PRNG is often used to help generating the one-time signature (OTS) keys [5]. That way one is able to reduce the size of the scheme's secret key by storing only a part of it and rebuilding the necessary information on the fly. A

sequence of 2^h seeds is computed from a single initial seed, iteratively (Fig. 2). In that case, the one-time key pairs (2^h in total) are not all stored in the overall secret key but are generated successively instead. This generation takes place twice: for the generation of the overall public key and for signing[4]. The very basic data to be synchronized are the index and the succeeding seed. This would be just 36 bytes with $n = 32$ and XMSS or 72 bytes with $n = 64$ and XMSS^{MT}.

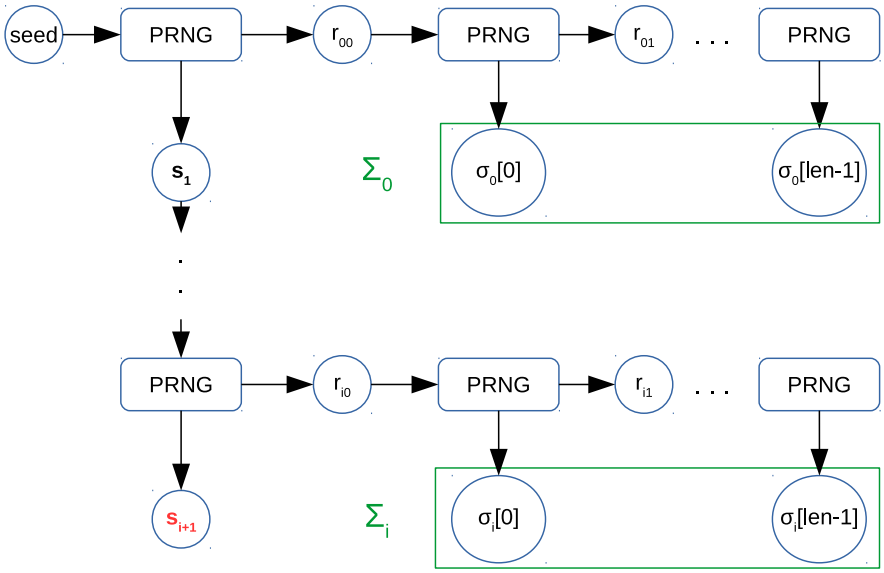

Fig. 2. Iterative generation of successive PRNG seeds (s_i) from an initial one (seed), and OTS signing key generation. The OTS signing keys are $\Sigma_i = \sigma_i[0] \ldots \sigma_i[len-1], i \leq 2^h$, where *len* is the number of n-byte string elements in a W-OTS$^+$ secret key, public key, and signature. As part of the state synchronization, after the generation of the ith OTS signing key, the next seed s_{i+1} (in red) must be stored.

The total size of the state data to be updated then depends on the used parameters and implementation choices. In particular, different methods to update the authentication path[5] involve the storage of different elements (e.g. nodes for the next authentication path or precomputed nodes) as part of the state.

[4] This allows for forward-secure constructions if used with the right schemes, e.g. special instantiations of XMSS using a forward-secure PRNG as shown by [2]. That way an attacker may get access to the secret key on a system but is not able to forge signatures using previous keys. A hash-based secret key is then to be seen just as secure as any other signing key that an attacker gets access to.

[5] The authentication path is the sequence of tree nodes that a verifier needs to reconstruct the path to reach the root of the tree from a leaf.

We now consider this case where a PRNG is used. When nodes are computed on the fly like this, an important concern is the worst case running time. Without precomputation, the large variation in node computing time (depending on how many already known nodes can be reused) would lead to unbalanced node computations and unacceptable delays in signing speed. By precomputation, modern tree traversal algorithm proposals strive to mitigate these variations by balancing the number of leaves computed in each computation of an authentication path [4]. Besides the index and the seed, the next authentication path and some nodes, which are needed in the near future, must be stored in the meantime (while some older nodes may be deleted). As more data has to be written one should also consider the impact of interrupts to the storing procedure. A generated signature must not be handed out before a new state, in this case including additional data by tree traversal algorithms, is available in storage. That way the old key state does not get compromised in case of an outage or other harmful events.

To give a sense of the size of the state to be synchronized in this case, we consider the following typical (plain) XMSS parameter set: tree height $h = 16$, message length $m = 32$ bytes, node length $n = 32$ bytes, Winternitz parameter $w = 16$[6]. For LMS [21], the parameter set are equivalent. They are provided in the specification's Table 1.

A number of different tree traversal algorithms exist (e.g. [4, 18]), and the choice of this algorithm defines the exact node precomputation method. In turn, the total size of the state depends on the chosen node precomputation method. The choice of a tree traversal algorithm does not affect the security of the overall signature scheme, but has a massive impact on performance. As an example, consider the BDS tree traversal algorithm [4]. We assume the value $K = 2$ for the BDS parameter. The state of the BDS algorithm comprises the current authentication path, an array of nodes used for the efficient computation of left authentication nodes, a single right authentication node, tail nodes on the stack and nodes for the treehash algorithm. Buchmann, Dahmen and Schneider showed [4] that the total space requirement of this algorithm depends only on the height of the tree and the size of the nodes: it is bounded by $(3.5h - 4)n$ bytes for that algorithm, i.e. 1664 bytes for our chosen concrete parameter set, which have to be written to the nonvolatile storage in addition. The algorithm computes $\frac{h-1}{2}$ leaves and $\frac{h-3}{2}$ inner nodes on average for each authentication path.

Storing therefore adds to each signing operation by several milliseconds, but using the techniques introduced in the following section helps reducing the need for storing each update of the key.

5 A State Reservation Strategy

When an N-time signature system's private key is read from nonvolatile storage into RAM, it can use a *state reservation* approach by writing back into storage

[6] Recall that the Winternitz parameter is used as a trade-off setting for the underlying one-time signature scheme.

the private key that is u signatures ahead of the current signature. In this way, the execution unit reserves the next u signatures for use, and avoids the need to write the updated private key u times into nonvolatile storage.

We formalize this idea by introducing a new *reserve* operation $R : [0, N] \rightarrow \{0,1\}^K$, which takes as input the number of private keys to reserve for use, and returns their values after it has updated the nonvolatile storage. We define an *N-time signature system with reservation* as an N-time system $\Upsilon = (K, R, S, V)$ in which the signing algorithm cannot directly access any private key information, and must first call the reserve function to obtain that key. For example, to sign the next X messages with $X \leq N$ the signer will receive X one-time signature keys from the reserve function R (or i.e. a key that may be advanced for up to X signatures, depending on the implementation). These keys are not sequentially synchronized in nonvolatile storage. They are delivered to the signing function only after the key state of the last key in the reservation is synchronized by R. If the signer crashes while it is using the X keys to sign the messages, the remaining keys in the reservation will not be used, which reduces the maximum number of signed messages, but there is no change of key reuse since the key state has already been updated in nonvolatile storage.

Given any N-time signature system, it is easy to define a signature system with reserve by introducing the appropriate function and a counter. Multiple signing engines can be accommodated by calling the reserve function to obtain a distinct private key range for each signing engine. The key still is a critical resource with this approach, but access to it is accelerated, as an engine calling the reserve function does not have to wait for another engine or process to finish signing first.

The reserve strategy minimizes the number of write operations to nonvolatile storage, which significantly reduces synchronization delay. The larger the interval u is, the less storage overhead is necessary. It also incurs a small penalty. Interrupting the update process of the reserve function is not a problem, as the key may not be written to nonvolatile storage but also will not be handed to a signing engine. However, any interruption of the signing process — including a crash, a graceful shutdown or the signing process simply not using all the signatures available — will reduce the number of messages that can be signed by the long-term key.

To offer an optimal coverage on diverse keys for addressing different use cases, for example with keys using different parameter sets or to provide backup keys if the current key's compromised somehow, a key provider tool may be used, holding a key pool $P = (\Psi_0, \Psi_1, \ldots, \Psi_{p-1})$ where p is the number of different keys. Each private key Ψ can be accessed via a reserve function.

Even though a reservation function introduces extra processing to generates a series of keys, key generation is less costly than signing. When decoupled from signing, a reservation function does not introduce a performance high risk factor. Several ways to improve the performance of N-time signature schemes in practice exist. As mentioned above, one common strategy is to generate the one-time signature key pairs by using a PRNG, instead of creating each key

pair randomly following a uniform distribution. That way a seed s is used to feed the PRNG, which returns output for the key generation as well as the successor seed s'. In this case the nature of N-time signature schemes can be used in favor of more security. Instead of creating all one-time signature key pairs using one initial seed that is employed iteratively by the PRNG, one may reseed the PRNG for every fixed interval u and limit the reserve function to a maximum limit of u signatures per call of the function by an engine. That way a signing engine receives several signing keys, but cannot get any information about the remainder of the long-term key. One then has to consider the compatibility with the used tree traversal algorithm and call that one correctly. As mentioned above, a tree traversal algorithm may precalculate further nodes to allow for a balanced average signing time. If using different seeds, a tree traversal algorithm used by the signing engine must be compatible with that behavior. To simplify implementation and streamline operation, the interval u should be a divisor of the number of leafs of a (bottom-layer) subtree so subtree boundaries are not crossed. The reserve function does have access to all secret key information and therefore has no problem updating the secret key using several seeds.

Another way to improve security is to introduce volatile and nonvolatile levels within the signature scheme Υ. This approach is detailed next.

5.1 Volatile and nonvolatile levels

A hierarchical signature scheme very naturally supports a reserve operation of all $r = N_{l-1}$ signatures associated with level $l - 1$. We formalize this notion in this section with the idea of a *volatile level*. In a hierarchical signature system, the first $k \leq l$ levels can be maintained in nonvolatile storage, while the remaining $l - k$ levels can be volatile (Fig. 3).

Without essential loss of generality, we consider a signature chain with exactly two levels[7]; Υ_0 is nonvolatile, and Υ_1 is volatile. The overall scheme is an N_0N_1 time signature scheme that naturally supports a reserve of exactly $r = N_1$. The only time that Υ_0 needs to sign is when the reserve operation is invoked and when Υ_1 is exhausted (that is, it has produced all N_1 of its signatures). Each invocation of the reserve operation invokes the top level signing algorithm and the bottom level key generation algorithm. The key generation process may take a long time, but the system could generate bottom-level private keys in advance of when they are needed, and bring them into Υ at that time.

When a bottom level public/private key pair is generated, any public data (such as the components of a Merkle hash tree) can be cached in nonvolatile storage without raising any security issues. Furthermore, a private key could be kept in nonvolatile storage for some period of time pending its use in a volatile level, as long as it is erased from nonvolatile storage before it is used. That is to say, the private key of a volatile level may be temporarily stored in nonvolatile memory between the time that it is generated and the time that it is linked into Υ by signing its public key with the top level. We emphasize,

[7] Note that either of these two levels could themselves be hierarchical signature schemes.

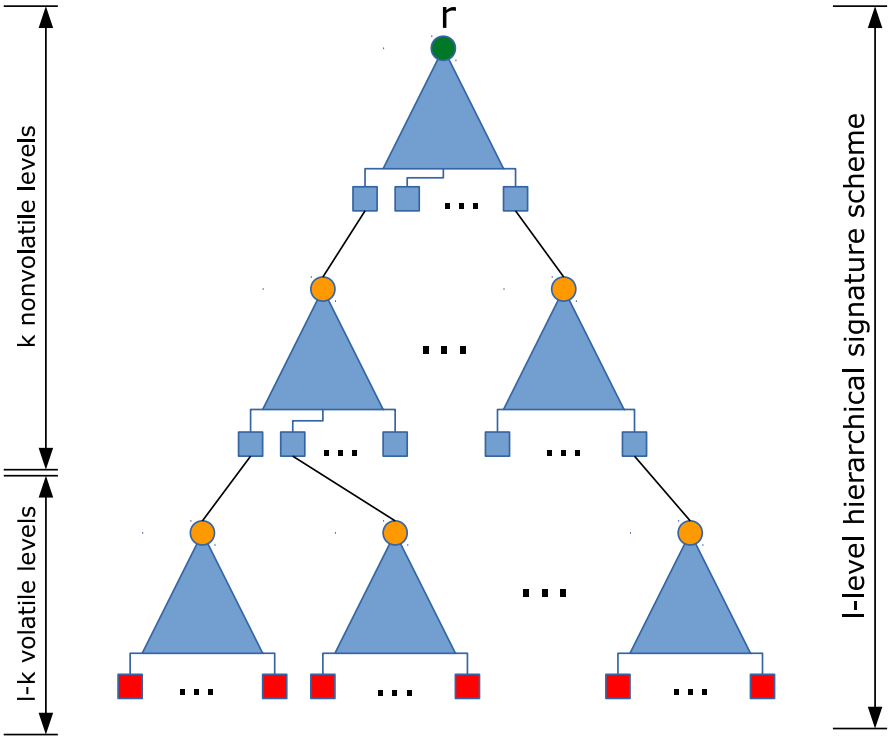

Fig. 3. Combined volatile/nonvolatile hierarchical signature scheme with $l = 3$ and $k = 2$. r is the multitree root, i.e. the public key of the entire scheme. Each subtree can be a stateful $N - time$ signature scheme like XMSS or LMS.

however, that all private keys at the volatile level must not be synchronized with nonvolatile storage. A system that uses this *volatile key precomputation* strategy must carefully avoid synchronization issues that might cause a cached private key to be used in multiple levels.

A hierarchical signature scheme with a volatile bottom level enforces the reservation property; since the private key of the volatile level is not synchronized in nonvolatile storage, it avoids synchronization problems. Of course, the root level must be nonvolatile, but it only rarely performs signing, and thus rarely changes its state. This is a benefit because synchronization delay is no longer an issue, and there is less chance of a synchronization problem (in the reasonable model that problems occur when there is a crash or power outage during a write operation).

We conjecture that a nonvolatile level with 2^{20} signatures is sufficient to provide security in general-purpose software environments, since that number of keys can be used to sign once per hour for over a century, or once per second for nearly two years. This is enough time to allow for almost all types of system

failures to be detected and corrected. Even 2^{16} signatures may be sufficient if a signature is associated with an hourly re-initialization, considering that it is enough to last for seven years.

However, the reservation strategy and hierarchical signature schemes do not address the nonvolatile cloning problem. If a root level private key is copied and then used by multiple execution units, an attacker could forge a signature on a bottom level public key, and thus perpetrate N_{l-1} forgeries. This limits the scenarios in which signature scheme could be used.

6 Stateless and Hybrid Approaches

The SPHINCS hash-based signature scheme [1] is stateless, and thus avoids the synchronization delay and cloning problems outlined above. It can sign a nearly arbitrary number of messages, but unfortunately, it is less efficient than stateful hash-based signatures; its signatures are over 40KB in length, and they take a relatively long time to generate.

Ideally, we would like to have a signature scheme that has the shorter signatures and quicker signing of a stateful method, but which has the advantages of the stateless methods. A reasonable compromise is the following hybrid scheme: a hierarchical signature scheme with a stateless N_0-time scheme at the root level of Υ, while the other levels are stateful (i.e. XMSS, LMS). Such a structure is shown in Fig. 4 (which is similar to Fig. 3) with $k = 1$ and the top tree (nonvolatile level) being a stateless signature scheme. This hybrid approach provides security against nonvolatile cloning, because it does not require any state synchronization at the root level. It also avoids synchronization delay.

A stateless N-time signature scheme can sign up to N signatures at a certain security level. If more than N signatures are generated, its security will degrade, but this degradation can be graceful. The Hash to Obtain Random Subset (HORS) scheme [24] is a good example of this type of system. Each distinct message that is signed reveals a small subset of the private state, i.e. a few-time signature scheme is used instead of a one-time signature scheme as a cornerstone of the system. In order to perpetrate a forgery, an attacker would need to collect the private state that is revealed from many different signatures. HORS is further improved by HORST which is used in SPHINCS.

The hybrid approach described above can bring worthwhile advantages, because it can make use of a stateless signature scheme that can sign only a limited number of signatures, e.g. 64K or 1M. It does not entirely eliminate all of the issues surrounding state management. Even with a stateless N-time signature scheme, there is still a need to limit the number of signatures created with the scheme, and the enforcement of that limit might require some state synchronization. Of course, this need to restrict the number of signatures is fundamental to *any* N-time signature scheme. However, there may be some natural bounds on the number of signatures that a system could create with a nonvolatile private key. If those signatures are associated with the initialization of an application or an operating system, for example, then even if a restart or reboot occurs every

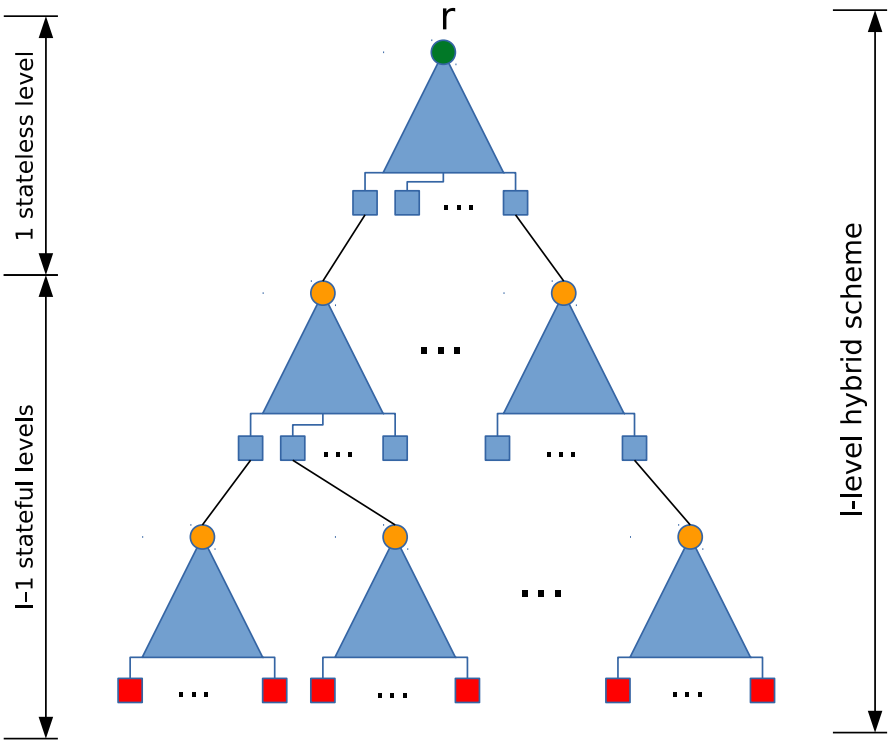

Fig. 4. A hybrid approach combining a stateless signature scheme like HORST [1] at the root level and stateful schemes like LMS and XMSS at the lower levels.

hour, it would take seven years to reach $N = 2^{16}$ root-level signatures and over a century to reach $N = 2^{20}$ signatures. In a backup scenario, either of those root level schemes would be adequate, since the restoration of an operating system or application from backup does not change the number of execution units. In a VM scenario, however, each cloning would produce multiple execution units; we do not recommend the hybrid approach for the volatile part of the hybrid structure for such environments. As discussed earlier (Sec. 3), there are several other security issues with the cloning of VMs, so it is not clear how important this limitation is in practice.

If volatile private keys are cached in nonvolatile storage, it is essential to make sure that they are not cloned. For instance, it is acceptable to have the private key stored in swap memory (assuming that there are adequate security protections in place), but it is not acceptable to cache those keys in a file that might be cloned in a VM or backed up.

In practice, with a hybrid scheme, if one never performs more than N restarts (for some reasonable N), all issues with state management are eliminated. One would regenerate the volatile state after every reboot (and generate a signature

of that volatile state based on the stateless scheme); one would then generate signatures based on the volatile state. If the volatile signature scheme is designed to be able to generate enough signatures, one can avoid regenerating that state until the next reboot.

7 Conclusions

As the transition to post-quantum cryptography moves forward — notably with standardization efforts at NIST, the IETF, and other organizations — existing schemes must be evaluated for real-world scenarios. In particular, hash-based signature schemes such as LMS and XMSS/XMSS^{MT} are good candidates for the post-quantum era, with minimal security requirements, but their statefulness is often mentioned when adoption is discussed.

We have taken a closer look at concrete cloning and state management scenarios, considered examples with typical parameter sets, and shown that the security-critical issues can be avoided using relatively simple measures. We discussed the effects of caching on state synchronization delays, and measures to mitigate these effects. It was shown how the size of the state to be synchronized depends not only on the chosen scheme and its parameters, but also on implementation choices — such as the tree traversal algorithm selection — which directly impact performance. A trade-off between performance and the (quantifiable) risk of slower state synchronization is therefore available to signing engines.

The state reservation strategy is essential for practical implementations. It makes sense to formalize this fact by defining it into the interface. The only way to avoid the reserve scheme is to accept the inefficiency of synchronizing state with nonvolatile storage, or to have N be so large that it could not be exceeded by the signing systems in use. Hierarchical signatures are a practical and reasonably simple scheme. The hybrid approach could be useful in general-purpose software environments, and deserves further exploration.

Currently specifications of hash-based signature schemes do not allow reserve parameters to be set. Since the reserve operation is not scheme-dependent, it can be specified generically. To offer an optimal granularity of reserve parameter choices for end users, feedback from stakeholders on this matter would be beneficial. Since hash-based signature schemes normally already require a large number of parameters to be set, additional complexity may impede usability.

While the statefulness of hash-based signature schemes must be carefully managed, it can be seen that a number of trade-offs are available to accelerate state synchronization if need be. This flexibility underscores the need to select parameters according to use case constraints, instead of always simply opting for maximal speed. We consider a detailed analysis of use case requirements for widespread security protocols relying on digital signatures as the next step in this direction. This study of use case requirements would pave the way for the standardization of state management strategies for typical applications.

References

1. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: Practical Stateless Hash-Based Signatures. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology — EUROCRYPT 2015 — 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques. Lecture Notes in Computer Science, vol. 9056, pp. 368–397. Springer (2015)
2. Buchmann, J., Dahmen, E., Hülsing, A.: XMSS — A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: Yang, B. (ed.) Post-Quantum Cryptography — 4th International Workshop, PQCrypto 2011. Lecture Notes in Computer Science, vol. 7071, pp. 117–129. Springer (2011)
3. Buchmann, J., Dahmen, E., Klintsevich, E., Okeya, K., Vuillaume, C.: Merkle Signatures with Virtually Unlimited Signature Capacity. In: Katz, J., Yung, M. (eds.) Applied Cryptography and Network Security, 5th International Conference, ACNS 2007. Lecture Notes in Computer Science, vol. 4521, pp. 31–45. Springer (2007)
4. Buchmann, J., Dahmen, E., Schneider, M.: Merkle Tree Traversal Revisited. In: Buchmann, J., Ding, J. (eds.) Post-Quantum Cryptography — Second International Workshop, PQCrypto 2008. Lecture Notes in Computer Science, vol. 5299, pp. 63–78. Springer (2008)
5. Buchmann, J., García, L.C.C., Dahmen, E., Döring, M., Klintsevich, E.: CMSS — An Improved Merkle Signature Scheme. In: Barua, R., Lange, T. (eds.) Progress in Cryptology — INDOCRYPT 2006, 7th International Conference on Cryptology in India. Lecture Notes in Computer Science, vol. 4329, pp. 349–363. Springer (2006)
6. Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R., Smith-Tone, D.: Report on Post-Quantum Cryptography (NISTIR 8105 Draft). `http://csrc.nist.gov/publications/drafts/nistir-8105/nistir_8105_draft.pdf` (2016), Accessed 2016-06-06.
7. Dods, C., Smart, N.P., Stam, M.: Hash Based Digital Signature Schemes. In: Smart, N.P. (ed.) Cryptography and Coding, 10th IMA International Conference. Lecture Notes in Computer Science, vol. 3796, pp. 96–115. Springer (2005)
8. ElGamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Blakley, G.R., Chaum, D. (eds.) Advances in Cryptology — CRYPTO'84, Lecture Notes in Computer Science, vol. 196, pp. 10–18. Springer (1985)
9. ETSI: White Paper No. 8: Quantum Safe Cryptography and Security. An introduction, benefits, enablers and challenges. `http://www.etsi.org/images/files/ETSIWhitePapers/QuantumSafeWhitepaper.pdf` (2015), Accessed 2016-06-06.
10. Everspaugh, A.C., Bose, B.: Virtual Machine Reset-Atomicity in Xen. Tech. rep., University of Wisconsin-Madison (2013), `http://pages.cs.wisc.edu/~ace/reset-atomic/reset-paper.pdf` Accessed 2016-06-06.
11. Garfinkel, T., Rosenblum, M.: When Virtual Is Harder than Real: Security Challenges in Virtual Machine Based Computing Environments. In: Proceedings of HotOS'05: 10th Workshop on Hot Topics in Operating Systems. USENIX Association (2005)
12. Goldreich, O.: Foundations of Cryptography: Volume 2, Basic Applications. Cambridge University Press (2004)
13. Hülsing, A.: W-OTS$^+$ — Shorter Signatures for Hash-Based Signature Schemes. In: Youssef, A., Nitaj, A., Hassanien, A.E. (eds.) Progress in Cryptology —

AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa. Lecture Notes in Computer Science, vol. 7918, pp. 173–188. Springer (2013)
14. Hülsing, A., Butin, D., Gazdag, S., Mohaisen, A.: XMSS: Extended Hash-Based Signatures. https://datatracker.ietf.org/doc/draft-irtf-cfrg-xmss-hash-based-signatures/ (2016), Internet-Draft. Accessed 2016-06-06.
15. Hülsing, A., Rausch, L., Buchmann, J.: Optimal Parameters for XMSS^{MT}. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E.R., Xu, L. (eds.) Security Engineering and Intelligence Informatics — CD-ARES 2013 Workshops: MoCrySEn and SeCIHD. Lecture Notes in Computer Science, vol. 8128, pp. 194–208. Springer (2013)
16. Information Assurance Directorate at the National Security Agency: Commercial National Security Algorithm Suite. https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm (2015), Accessed 2016-06-06.
17. Johnson, D., Menezes, A., Vanstone, S.: The Elliptic Curve Digital Signature Algorithm (ECDSA). International Journal of Information Security 1(1), 36–63 (2001)
18. Knecht, M., Meier, W., Nicola, C.U.: A space- and time-efficient Implementation of the Merkle Tree Traversal Algorithm. CoRR abs/1409.4081 (2014)
19. Lamport, L.: Constructing Digital Signatures from a One Way Function. Tech. rep., SRI International Computer Science Laboratory (1979), http://research.microsoft.com/en-us/um/people/lamport/pubs/dig-sig.pdf Accessed 2016-06-06.
20. Leighton, T., Micali, S.: Large provably fast and secure digital signature schemes from secure hash functions. U.S. Patent 5,432,852 (1995)
21. McGrew, D., Curcio, M.: Hash-Based Signatures. https://datatracker.ietf.org/doc/draft-mcgrew-hash-sigs/ (2016), Internet-Draft. Accessed 2016-06-06.
22. Merkle, R.C.: A Certified Digital Signature. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO '89, 9th Annual International Cryptology Conference. Lecture Notes in Computer Science, vol. 435, pp. 218–238. Springer (1989)
23. Monz, T., Nigg, D., Martinez, E.A., Brandl, M.F., Schindler, P., Rines, R., Wang, S.X., Chuang, I.L., Blatt, R.: Realization of a scalable Shor algorithm. Science 351(6277), 1068–1070 (2016)
24. Reyzin, L., Reyzin, N.: Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In: Batten, L.M., Seberry, J. (eds.) Information Security and Privacy, 7th Australian Conference, ACISP 2002. Lecture Notes in Computer Science, vol. 2384, pp. 144–153. Springer (2002)
25. Ristenpart, T., Yilek, S.: When Good Randomness Goes Bad: Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography. In: Proceedings of the Network and Distributed System Security Symposium (NDSS). The Internet Society (2010)
26. Rivest, R., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21, 120–126 (1978)
27. Saeedi, K., Simmons, S., Salvail, J.Z., Dluhy, P., Riemann, H., Abrosimov, N.V., Becker, P., Pohl, H.J., Morton, J.J.L., Thewalt, M.L.W.: Room-Temperature Quantum Bit Storage Exceeding 39 Minutes Using Ionized Donors in Silicon-28. Science 342(6160), 830–833 (2013)
28. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. on Computing 26(5), 1484–1509 (1997)