Post-Quantum Software Updates

A case study on Code Signing with Hash-based Signatures

Stefan-Lukas Gazdag[1] Markus Friedl[1] Daniel Loebenberger[2]

Abstract: Due to the progress in building quantum computers and the risk of attacks on cryptographic primitives based on quantum algorithms emerging, the development and analysis, but also the deployment of resistant schemes is an important research area. Hash-based signatures are a very promising candidate since they have been analyzed and improved for years. Nevertheless, there are some peculiarities that need consideration when using hash-based signatures in practice, for example the statefulness of some of the primitives. Fortunately, by now more and more experience is gained in real-world scenarios. In this paper we detail the troubles we encountered when using hash-based signatures in practice and study the most important use case for hash-based signatures: software or code signing.

Keywords: Post-Quantum Cryptography, Hash-based Signatures, XMSS, Software Update, ssh

1 Introduction

With the threat of quantum computing and its effect on today's cryptographic solutions arising, more and more companies and agencies are looking for alternatives to secure our current infrastructure. Well known schemes like RSA or the signature schemes DSA and its elliptic curve based equivalent ECDSA couldn't provide confidentiality, integrity nor authenticity when scalable quantum computer exist. For a survey on this topic see e. g. [BL17].

In the need for long-term security the transition to post-quantum schemes seems inevitable. Efforts have been made not only by the post-quantum cryptography community contributing and committing different schemes and providing security analyses but also by governmental and standardization agencies: In February 2017 the NSA [In17] released a recommendation to increase the security levels of currently used schemes and announced their plan to change to post-quantum schemes in the near future. NIST [Ch16; NI16] introduced a post-quantum crypto standardization project, the IETF has two RFCs on hash-based signatures (HBS) [Hu18; MCF19] as well as protocol-related efforts e. g. regarding IKEv2, and also ETSI [ET] is seeking for new post-quantum standards.

[1] genua GmbH, Kirchheim b. München, Germany, {stefan-lukas_gazdag,markus_friedl}@genua.eu
[2] Fraunhofer AISEC, Weiden i.d.Opf., Germany, daniel.loebenberger@aisec.fraunhofer.de

doi:10.18420/inf2019_63

The time needed for this transition is a major problem everyone has to face as the schedule is tight. Introducing new schemes is not only affiliated with building them and scrutinizing their security. They have to be standardized—which is currently a major effort—as well as integrated into software and tested in practical use cases.

As securing our whole infrastructure may take longer than the time it takes to build a large enough quantum computer (for a treatise of this issue we refer to [Mo15]), users long for introducing post-quantum cryptography to the first use cases where it is applicable. One of the use cases we can safeguard today is code signing and software updates. These are critical for a secure migration to a fully post-quantum secure system. If quantum attacks occur earlier than expected and systems are not yet equipped with a quantum-resistant update mechanism, there is no easy way to secure the systems afterwards. In the worst case a manufacturer would have to deliver updates with a secure carrier.

But even in the case that quantum computers can never be built due to foundational problems not being known or understood today, work motivated by the transition to post-quantum schemes will have great benefit for our understanding of applied cryptography. Furthermore, a diverse range of cryptographic schemes based on different kinds of problems would enhance our knowledge, since a versatile pool of well-analyzed, practically relevant and deployed schemes would supply enough alternatives to switch to another scheme if trouble arises for one of the currently used ones. For this a lot of questions are yet to be answered, e. g. regarding modular design of protocols, hybrid solutions for using cryptography and testing protocols and implementations while being confronted with so many different combinations of cryptographic schemes.

Thus, it makes sense today to experimentally employ selected post-quantum primitives in practice. We are in good company here: Google announced in mid 2016 that they would run first real world experiments with post-quantum cryptography. In the announcement they note [In16]:

> "Our aims with this experiment are to highlight an area of research that Google believes to be important and to gain real-world experience with the larger data structures that post-quantum algorithms will likely require."

They have pursued their efforts since and others have followed.

Motivation. But which post-quantum primitives come into consideration for such experiments? As stated above, we need a scheme which is well analyzed and standardized. What we mostly do have are experimental schemes which are in a process of standardization, notably the candidates [NI17; NI19] of the NIST call [NI16] for post-quantum cryptographic algorithms and the hash-based signatures from the informational RFCs, namely XMSS [Hu18], Leighton-Micali Signatures, and HSS [MCF19].

In this article we discuss a specific use case which is using hash-based signatures in the context of software updates. The cryptographic foundations of these signatures were invented quite early by Merkle in 1979 [Me79] and advanced in works like [BDH11; BDS08; Bu07; HRB13]. Unfortunately, the schemes suffer some peculiarities (the size of cryptographic keys, statefulness of the private key, etc.) which ruled them out for classical applications for some time. On the other hand they now look promising in the post-quantum context. Most other post-quantum schemes are comparatively recent academic developments which clearly have not went through thorough analyses by many people over several decades, though a lot of work is invested to enhance our knowledge. Still, the trust in those schemes is not fully established yet and it seems too early to apply them broadly in practice, though in some cases we already have to do so (e. g. by using hybrid solutions) to gain experience and trust.

Additionally, for hash-based signatures there are also first results on practical implementation issues [HBB12; KMN14]. For LMS there exists RFC 8554 [MCF19] and XMSS was released as RFC 8391 [Hu18]. The work of several authors showed how crucial a secure environment is to the handling of the private key [CMP18; FG18; Ge18; Ka18], yet we know how to deal with this in the case examined in this article. What is still missing, is work on the correct use of these algorithms in real-world settings. To do so we want to exhaustively describe the most suitable use case for hash-based signatures and the surrounding settings in which to use these. We also evaluate the features and stress the limitations of this kind of schemes.

2 Peculiarities of hash-based signatures

2.1 Hash-based signatures

We first give a brief introduction to hash-based signatures partly following [Mc16]. Unlike most other signature schemes, HBS 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 construction.

One-time signature schemes. Hash-based signatures use one-time signature (OTS) schemes as a fundamental building block. Common examples are the seminal one by Lamport [La79], the Winternitz scheme [DSS05], and its recent variant W-OTS$^+$ [Hü13]. For descriptions of current one-time signatures we refer to the respective section of [Hu18; MCF19]. For one-time signatures, the private key is usually generated randomly 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. One-time schemes are inadequate on their own in practice, since each private key can only be used to securely sign a single message.

Hash-based or N-time signature schemes. Merkle introduced a way to manage a confined number of OTS key pairs in 1979 [Me79] by using a binary tree to administer the one-time keys. A tree with height H may hold up to $N = 2^H$ key pairs and sign this number of messages. While OTSs use hash functions just as well, it is this construct that is referred to as *hash-based signatures*. It consists of the classical algorithms of key generation, signing and verification. Several improvements have been made for hash-based signatures, e. g.[BDH11; BDS08; Bu07; LM95]. For each signature generation a different OTS key pair is used. An integer counter has to keep track of the advancement of the key. A simple way to reduce the size of an N-time key is to define it to be a short string and then use a cryptographically secure pseudorandom function to generate the actual keys of the underlying one-time scheme.

Hierarchical Signatures. A *hierarchical signature scheme* is an N-time signature scheme that uses other hash-based signatures in its construction. In literature it is mostly referred to as *multi-tree* or *hyper-tree* variant of classical hash-based signatures. A scheme of this kind uses layers of trees meaning the root of a single tree is signed by an OTS key of a different tree. That way trees of large height may be build with relatively small cost in performance and memory demand. Concrete examples of hierarchical hash-based signatures include XMSS^{MT} [HRB13], a scheme by Leighton and Micali [LM95] and SPHINCS [Be15]. XMSS^{MT} and SPHINCS define a parameter d as the number of layers in the hierarchical structure. Additionally, the LMS [MCF19] specification describes a hierarchical hash-based signature variant based on Leighton and Micali's scheme, called HSS.

2.2 Statefulness

Hash-based signatures introduce some rare properties. The most important one is the *statefulness* of the private key of most hash-based signature schemes: A secret key has to be updated with the generation of each signature, since the underlying one-time signature scheme must only use its private keys exactly once to preserve its security features. In the most basic case it is the index of the next OTS key pair within the Merkle tree which has to be updated. In advanced implementations using pseudo-random generation of the OTS key pairs and an improved tree traversal algorithm also the seed for the pseudo-random number generator and nodes within the tree have to be updated in addition.

Another issue is the limited number of OTS key pairs. A private key exhausts due to the limited number of signing keys. Therefore a decision about the required number of signing keys must be made before key generation. And for the case that the HBS key has run out of OTS keys, some form of warning and key exchange system has to be installed.

The private key therefore needs special consideration and attention. No matter if the key is stored unguarded e. g. on a hard disk (which we definitely do not recommend) or if some form of key management was established the private key has to be seen as a critical resource.

An exclusive access to the private key must be enforced. Multiple processes or instances must not access the private key at the same time. Indeed, these issues regarding statefulness were resolved or easened, e. g. with a reservation approach [Mc16].

Yet some issues can't be addressed in common ways. Any copy of the secret key may be problematic. As [Mc16] show it is quite some effort already to make sure no unwanted copies are left within a modern computer system. The wish for having a backup of the secret key is tough as this would mean one would have to update the backup key as well. Some approaches as for state management might be applied to a backup as well, but still lead to complex scenarios. We rather recommend to provide several stateful keys and to deploy their public keys altogether. Then all but one of the secret keys may remain unused and stored in a secure place. If there's a problem with the currently used key one may switch to the "backup" key(s). Traditional backup mechanism may only be applied to stateless schemes.

Stateless schemes do exist, the first being SPHINCS [Be15], followed by the two submissions to the NIST standardization process [AE17; Be17]. Instead of one-time signature schemes so-called few-time signature schemes are used. With a single key pair multiple signatures may be generated with each signature generation reducing the security. That way the index of the key pair used can be chosen at random. Signatures of such kind have bigger sizes, while the need for handling a state is exchanged with a probability calculation.

2.3 API incompatibility

The need for an update of the private keys implies that HBS typically don't match common interfaces of signing operations of state of the art cryptographic tools and libraries. Even more, stateful HBS don't fit the definition of cryptographic signature schemes in general, since those traditionally only consist of `keygen` for key generation, `sign` for signing a message with a private key and `verify` for verification of a signature given a public key. Now another operation is needed: the operation `update` which evolves the key. One solution is to implement the function `update` implicitly as part of the signing operation `sign`.

McGrew et al. [Mc16] propose a state reservation method which grants access to a specific state of the key. Unfortunately, every way of implementing a secure form of HBS key management will always have some overhead due to writing operations or special key update tactics.

2.4 Storage and speed requirements

For different use cases various key sizes and different forms of key management might be suitable. Even a pool of HBS keys would be conceivable. This in turn has a vast effect on the amount of memory needed for the key material, the signature itself and many temporary

data structures which are employed during the signing/verification/update procedures. Also, the time required for all operations might vary considerably.

In the case of hash-based signatures, there is a natural trade-off between storage size and life-time of the secret key. A small key will have a short life-time, while a large key can be used more often. Also, the time needed for different cryptographic operations depends on the size of the key: Typically, the time increases when the keysize grows.

3 Software Update Authentication

We now explore one of the most suitable use cases for HBS: the authentication of software updates or firmware. When delivering IT systems or software, manufacturers usually provide means of authenticity. This could be a sealed package handed to the customer or a software being installed only if it was signed by a key an operating system already knows. Once the user trusts the system, the different applications take care of further updates.

A non-specified attacker (e. g. an intelligence agency) may be able to forge signatures for a malicious product update and include a backdoor. To prevent attacks, software update mechanisms as well as the updates themselves must be secured. This is especially important when systems need to run over a long period of time. Many servers and machines are working for years, sometimes even decades. For instance, satellites may be updated via a wireless connection. However, while improving parts of a system via updates may be straightforward, modifying currently used update mechanisms on older machines or systems remotely, as satellites sometimes remain active for a long time, can be tricky.

Therefore, a mechanism and parameter settings have to be used that provide security for at least the expected period of service. Of course advances in cryptanalysis and software bugs may require a change of the currently used system. Therefore it is always advisable to design the update mechanism as modular as possible and to allow easy exchange or update of keys or the mechanism itself.

As noted, hash-based signatures are particularly well suited for software updates: On one hand, updates are often released in an interval of at least days (e. g. to apply security fixes), weeks (patches for newly released computer games) or even months (major product updates). Thus, comparatively few signatures need to be generated. Even a small key with a tree of height 10 lasts for 85 years assuming monthly updates. A key using a tree of height 20 would last 29 years, even with a hundred signatures per day (e. g. including test builds). Another way to deal with this is to e. g. use one key pair per major release if this suits the update policy.

On the other hand, in many update settings, neither key generation nor signature generation or verification are too time-critical. Only the last may be limited e. g. on resource-restricted devices, but a tailored solution with minimal verifier (only providing mandatory or necessary parameter sets) with minimal code base should offer adequate performance and compatibility.

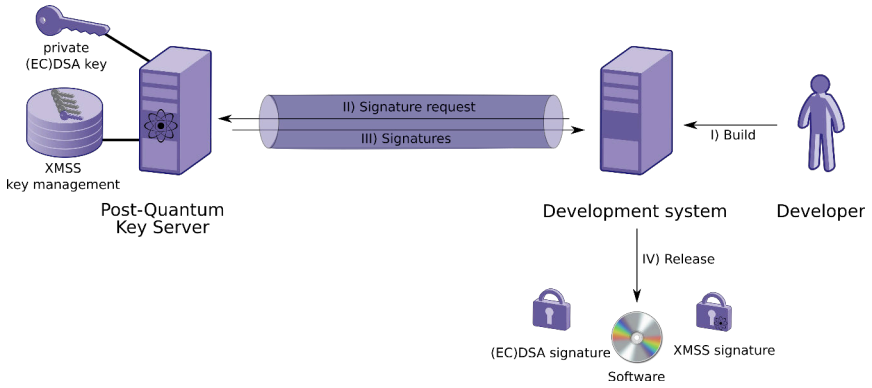

Fig. 1: A simple but secure setup for software development. A key server connects to selected development machines and handles requests for signing code.

A key pair may be generated in advance, so the process could take hours without causing trouble. Usually it does not matter if the signature generation or verification takes a few hundred (milli-)seconds. There might be some restrictions in size, depending on the specific update mechanism, the used data structures and possibly memory restrictions on devices with restricted resources. Luckily the public key of most HBS is quite small. Only the signature size might be problematic in few cases. Also, secure updates are universally relevant, since they may serve as a basis for realizing secure cryptographic algorithm-agility.

3.1 Software Update Environments

We now examine the process of releasing a signature for a software update. Numerous ways of implementing a software development environment and update mechanisms exist, one particular way is depicted in Figure 1. A typical infrastructure consists of a server (or a distributed net of servers) holding the current version of the code, a dedicated build server (which is sometimes in union with the code server) and a key server. Especially large companies use more advanced systems and tools to control and manage huge amounts of code and software, where lots of different products or services are managed, instead of a single product (version) as described here. Ideally, changes to the code base involve at least three persons: A software developer working on the actual change (two considering pair programming), at least one reviewer checking the code for bugs, conceptual errors as well as backdoors or other malicious changes and an integrator who includes the change (ideally after another review) into the current code base. Finally, a build server takes the cleared source code and builds a final package to be released or for test purposes. Such a software package is to be cryptographically secured e. g. by signing the content of the package.

To this end, the build server needs a signing key which resides on a distinct key server. The key server may hold diverse keys, but each key should be only used for one single product or

a specific purpose. Ideally, each key is stored on a dedicated smartcard or at least provided by a different security anchor like a hardware security module. The communication between those systems is to be secured, even if it appears to be a separated network. This can be done easily via TLS, SSH or other protocols securing the connection. To build a code package, the build server takes the latest code base, builds the software, asks the key server to sign e. g. a message digest respectively a hash value of the package using a particular signing key and hands out the package to be released as well as the signature. The targeted system is now able to verify the integrity and issuer of the package before installing the update.

A pleasant feature of update mechanisms is the fact that in many cases, it is possible to establish a hybrid solution which may help increasing confidence in the use of HBS. An update mechanism could use ECDSA [In13] or EDDSA [JL17] to verify the software and double-check by using an XMSS key and signature. This allows us to add signatures that are believed to be quantum-safe while still relying on well-established cryptographic primitives such as (EC)DSA.

In some cases companies sign the same update with several keys to offer compatibility for upgrading from older machines, though this may contain steps meaning you may have to install a version in-between first before going to the targeted version. In such an environment HBS could be simply another rollout of an advanced key. No matter whether a hybrid or a successive solution is used, in a pre-quantum era, it is possible to only trust the classical key at first and observe the behavior of the post-quantum part of the update mechanism. In the unlikely event of a deviation in the verification process, a skeptical user (and with the assistance and allowance of the user the manufacturer) could check for the cause. This way HBS can be deployed, show their qualification as successor technology and be switched to on the way before attacks by quantum computers emerge.

3.2 Our Implementation

We now discuss the integration of HBS in a currently used update mechanism and its implications, bearing stepping-stones that have been shown before. The main drawback of HBS is to be found in the statefulness of some schemes. Fortunately, for update mechanisms, this disadvantage dissipates. Even relatively small tree sizes (e. g. a tree of height 20) suffice to sign daily test builds and official software or patch releases. Typical software updates in our case have sizes of a few to many megabytes. Often it does not matter if a signature requires a few kilobytes. Speed is not that relevant either, as a user installing the update will not notice a few more hundred milliseconds. As a consequence, issues typical for HBS have less impact here. The statefulness mainly affects the key server which has to offer a key management instance that controls the access to the actual key, like the reservation approach proposed by McGrew et al. [Mc16]. While in a regular case one would directly call the signing function, for HBS the reservation function is called first. It then returns the current state of the key.

Though some quirks of HBS seem not to be relevant, some aspects still need consideration. One does have to think about dependencies by (automatic) test setups. While testing the signature generation and verification may not be troublesome considering the runtime, the generation of a key might be. This can take several minutes even for fairly small parameter sets and result in the delay of other tests.

Another issue is providing a secure environment, which must not be neglected for classical schemes as well. It is tricky to actually enforce such an environment on off-the-shelf hardware. One option is to employ smartcards, which are an important security anchor for providing cryptographic services while protecting the key material. In most cases, where a central secure key server is not suitable, a smartcard would be the only really secure way to use HBS. But the nature of a smartcard is not well-disposed for the use of HBS. In conventional cases, a secret key on a smartcard only has to be read after it was written to the memory. With HBS, some amount of data has to be rewritten, which might cause trouble. The first constraint is that a smartcard with re-writable memory must be used, but the memory wears out over time with each writing operation. As a result, the rewriting of data has to be managed to avoid that some sections of the memory are used too often and may get broken. Hülsing et al. [HBB12] have shown how smartcards can handle HBS schemes like XMSS^{MT}. If a smartcard is not applicable, yet a secure environment has to be used. First Hardware Security Modules (HSMs) are available with experimental support for quantum-resistant schemes. At least a hardened server with corresponding restrictions has to be used, so an attacker has no influence especially on the key generation.

We selected XMSS as described RFC [Hu18] for our implementation, since it suits our security requirements well. To do so, a parameter set should be picked from the options provided by the IETF specification according to personal needs: A total tree height of 20 was sufficient for most applications in our setup, independently of whether XMSS or XMSS^{MT} is used, though bigger tree height could be used easily. Relative to other official parameter sets, this is a large height for XMSS but a small one for XMSS^{MT}. We recommend XMSS^{MT} for performance reasons, but XMSS can also be used and has the advantage of a simpler implementation. In practice, we recommend

- `XMSS-SHA2_20_256` or `XMSS-SHAKE_20_256` for XMSS and
- `XMSSMT-SHA2_20/2_256` or `XMSSMT-SHAKE_20/2_256` for XMSS^{MT} for a more efficient implementation.

For a higher security level and being well prepared for a post-quantum era one should opt for

- `XMSS-SHA2_20_512` or `XMSS-SHAKE_20_512` for XMSS
- `XMSSMT-SHA2_20/2_512` or `XMSSMT-SHAKE_20/2_512` for XMSS^{MT}.

Note that some of these are optional parameter sets in RFC 8391.

To secure the connection between the systems involved, we decided to access the servers via OpenSSH and implemented experimental XMSS support[1]. Also other post-quantum suites are tested to establish a quantum-resistant connection.

For the purpose of signing data, several open-source applications exist, e. g. in the OpenBSD context `signify` or its predecessor `gzsig`. We opted for the latter due to its greater flexibility and simplicity. These tools are easily adaptable by introducing the use of XMSS via X.509 certificates or SSH keys and including the XMSS functionality by either using an adapted cryptographic library like LibreSSL[2] or forks with a focus on quantum-resistant algorithms like the Open Quantum Safe Project[3] or at least by some verified standalone solution. A standalone solution would make it possible to offer a minimal verifier for the updated systems, but also means that it is an individual solution for that product. In our case we use OpenSSH to offer all cryptographic functionality.

On start-up a secure key server connects to several development machines. Access to the key server is restricted by digital as well as physical means. As soon as a signature is needed for a new patch, the key server is asked to sign the code package by the signing tool. As usual instead of the whole code package a hash value is signed. The key server contains the reservation-based key management and controls the access to the XMSS keys and other private keys. A signing agent awaits the requests and though only one signature per signature scheme used is needed, the signing agent uses the reservation function to reserve an interval of keys. In this case the software solution or cryptographic library used to supply XMSS functionality on the build server or in software itself (verifier) does not need to provide the state management instance, as an instance on the key server takes care of this. In addition, digests (e.g. SHA2-512 or SHA3-512) of the package or signatures may be provided via other channels than the package itself. One alternative is to provide package digest values to users via a web interface. A software system to be updated holds a public key that can be used to verify the new software or update package (e.g. in the X.509, SSH or PKCS8/12 format).

As described before, the transition to post-quantum cryptography is best done by implementing hybrid solutions. We highly recommend to do so in the case of adapting or extending update mechanisms. In many cases, it should be straightforward to verify multiple signatures of different signature schemes and check for the correct verification of all tested signatures. Nevertheless some settings might include resource-restricted environments or deployed software that can't be updated easily. At least it is possible with most HBS to offer a minimal verifier while also having a quite small public key. This easens for example the requirements of the IoT or embedded devices. We use ECDSA as a pre-quantum solution which might resist first attacks based on early quantum computers as standard case to trust, while also verifying an XMSS signature. During an initial transition phase, both ECDSA and XMSS signatures could be verified in parallel with only the result of the ECDSA

[1] https://www.openssh.com/txt/release-7.7; git repository https://github.com/openssh/openssh-portable
[2] https://www.libressl.org/
[3] https://openquantumsafe.org/

Device	Type
Processor	AMD Geode LX800
Speed	500MHz
Memory	256MB SDRAM
Operating System	OpenBSD 6.1

Object/Operation	Requirements
Secret key	4363 Byte
Public key	190 Byte
Signature	2820 Byte
Key generation	25 hours
Signing	0.91s
Verification	0.75s

Tab. 1: Some characteristics of our key-server (left) as well as the corresponding benchmarking results (right). Shown are values for the variant `XMSS_SHA2_20_256` following RFC 8391. The keys were stored in the ssh key format. Only approximate timing results are given, since the verification time is client-dependent. Note that the results show sizes including the encoding overhead and times include any software overhead.

verification actually being taken into account. Nevertheless we opted for trusting both signatures, classical as well as quantum-resistant, so no update can be installed if a single one of them fails verification. Introducing this approach to an update mechanism can mean that no downgrade or fallback to a former software release is possible unless the update mechanism would not check the new signatures if an old version is to be installed. This is a serious attack vector as an adversary might forge a software patch masqueraded as an old version. Thus we recommend to only accept with both signatures verifying positively and in our case we have not experienced or heard about any problems even on live systems in the field. We expect that even with sticking to an optional transition face confidence in HBS will increase gradually and remove any impact on actual verification in the unlikely event of an XMSS verification failure.

Also, a hybrid approach constitutes a secure solution to comply to conflicting governmental requirements. Imagine various agencies asking for different cryptographic signature schemes to be used in specific settings. Thanks to the hybrid approach several schemes may be used targeting diverse specifications while each provider of requirements can be sure that each update is checked by means of own liking.

If small trees shall be used or backup keys are wished for other reasons a key management may provide several keys. Each major release may be shipped with several keys, so there's one active signing and verification key while there's e. g. two backup keys available. If no unforeseen events enforce the exchange of more than one key, the active key may be replaced with the next major update. This also works for a backup key server. If a problem with the running key server occurs one might switch to a backup system or restore the actual key server which can use the unused, securely (and depending on the solution independently) stored extra keys.

4 Conclusion

We presented a working implementation of a post-quantum secure software update mechanism based on hash-based signatures. The code-signing mechanism is currently run in a hybrid fashion employing ECDSA and XMSS signatures at the same time.

This allows us to experimentally work with novel cryptographic algorithms resistant to quantum computers in practice without compromising the well-established security of the update mechanism. Our implementation was carried out on an OpenBSD-based kernel and a version of OpenSSH providing basic XMSS support for the purpose of post-quantum secure software updates.

Summarizing, we were able to realize post-quantum secure crypto-agility in practice, offering real-world software signatures that cannot be broken by a scalable quantum computer—as far as we know. Such a protection for software updates can and should be used today in components for high-security areas with long expected deployment times.

Acknowledgments

The work underlying this article was mainly done when the third author worked with the first two authors at genua GmbH, Kirchheim, Germany. We thank Hans-Jörg Hoexer for collecting the benchmark-data given in Table 1 and Tobias Heider for his review.

References

[AE17] Aumasson, J.-P.; Endignoux, G.: Gravity-SPHINCS, Submission to the NIST standardization process, 2017.

[BDH11] Buchmann, J.; Dahmen, E.; Hülsing, A.: XMSS — A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: PQCrypto. Vol. 7071. LNCS, Springer, pp. 117–129, 2011, ISBN: 978-3-642-25404-8.

[BDS08] Buchmann, J.; Dahmen, E.; Schneider, M.: Merkle Tree Traversal Revisited. In: PQCrypto. Vol. 5299. LNCS, Springer, pp. 63–78, 2008, ISBN: 978-3-540-88402-6.

[Be15] 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: EUROCRYPT 2015. Vol. 9056. LNCS, Springer, pp. 368–397, 2015, ISBN: 978-3-662-46799-2.

[Be17] Bernstein, D. J.; Dobraunig, C.; Eichlseder, M.; Fluhrer, S.; Gazdag, S.-L.; Hülsing, A.; Kampanakis, P.; Kölbl, S.; Lange, T.; Lauridsen, M. M.; Mendel, F.; Niederhagen, R.; Rechberger, C.; Schwabe, P.: SPHINCS+, Submission to the NIST standardization process, 2017.

[BL17] Bernstein, D. J.; Lange, T.: Post-quantum cryptography. Nature 549/, pp. 188–194, 2017.

[Bu07] Buchmann, J.; Dahmen, E.; Klintsevich, E.; Okeya, K.; Vuillaume, C.: Merkle Signatures with Virtually Unlimited Signature Capacity. In: ACNS. Vol. 4521. LNCS, Springer, pp. 31–45, 2007, ISBN: 978-3-540-72737-8.

[Ch16] Chen, L.; Jordan, S.; Liu, Y.-K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D.: Report on Post-Quantum Cryptography (NISTIR 8105), http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf, Accessed 2017-02-20., 2016.

[CMP18] Castelnovi, L.; Martinelli, A.; Prest, T.: Grafting Trees: a Fault Attack against the SPHINCS framework. In: International Conference on Post-Quantum Cryptography. Springer, pp. 165–184, 2018.

[DSS05] Dods, C.; Smart, N. P.; Stam, M.: Hash Based Digital Signature Schemes. In (Smart, N. P., ed.): Cryptography and Coding. Vol. 3796. LNCS, Springer, pp. 96–115, 2005, ISBN: 3-540-30276-X.

[ET] ETSI: Quantum-Safe Cryptography, http://www.etsi.org/technologies-clusters/technologies/quantum-safe-cryptography, Last access on 2017-04-18.

[FG18] Fan, J.; Gierlichs, B., eds.: Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings, vol. 10815, Lecture Notes in Computer Science, Springer, 2018, ISBN: 978-3-319-89640-3, URL: https://doi.org/10.1007/978-3-319-89641-0.

[Ge18] Genêt, A.; Kannwischer, M. J.; Pelletier, H.; McLauchlan, A.: Practical Fault Injection Attacks on SPHINCS, Cryptology ePrint Archive, Report 2018/674, https://eprint.iacr.org/2018/674, 2018.

[HBB12] Hülsing, A.; Busold, C.; Buchmann, J. A.: Forward Secure Signatures on Smart Cards. In: SAC. Vol. 7707. LNCS, Springer, pp. 66–80, 2012, ISBN: 978-3-642-35998-9.

[HRB13] Hülsing, A.; Rausch, L.; Buchmann, J.: Optimal Parameters for XMSS^{MT}. In: Security Engineering and Intelligence Informatics — CD-ARES 2013 Workshops: MoCySEn and SeCIHD. Vol. 8128. LNCS, Springer, pp. 194–208, 2013, ISBN: 978-3-642-40587-7.

[Hü13] Hülsing, A.: W-OTS$^+$ — Shorter Signatures for Hash-Based Signature Schemes. In: AFRICACRYPT. Vol. 7918. LNCS, Springer, pp. 173–188, 2013, ISBN: 978-3-642-38552-0.

[Hu18] Huelsing, A.; Butin, D.; Gazdag, S.-L.; Rijneveld, J.; Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme, RFC 8391, IRTF, May 2018, 74 pp., URL: https://rfc-editor.org/rfc/rfc8391.txt.

[In13] Information Technology Laboratory: Digital Signature Standard (DSS), en, tech. rep. NIST FIPS 186-4, National Institute of Standards and Technology, July 2013, URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, visited on: 04/12/2019.

[In16] Inc., G.: Experimenting with Post-Quantum Cryptography, https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html, July 2016.

[In17] Information Assurance Directorate at the National Security Agency: Commercial National Security Algorithm Suite, https://www.iad.gov/iad/programs/iad-initiatives/cnsa-suite.cfm, Accessed 2017-02-20., 2017.

[JL17] Josefsson, S.; Liusvaara, I.: Edwards-Curve Digital Signature Algorithm (EdDSA), RFC 8032 (Informational), RFC, Fremont, CA, USA: RFC Editor, Jan. 2017, URL: https://www.rfc-editor.org/rfc/rfc8032.txt.

[Ka18] Kannwischer, M. J.; Genêt, A.; Butin, D.; Krämer, J.; Buchmann, J.: Differential Power Analysis of XMSS and SPHINCS. In (Fan, J.; Gierlichs, B., eds.): Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings. Vol. 10815. Lecture Notes in Computer Science, Springer, pp. 168–188, 2018, ISBN: 978-3-319-89640-3, URL: https://doi.org/10.1007/978-3-319-89641-0_10.

[KMN14] Knecht, M.; Meier, W.; Nicola, C. U.: A space- and time-efficient Implementation of the Merkle Tree Traversal Algorithm. CoRR abs/1409.4081/, 2014.

[La79] Lamport, L.: Constructing Digital Signatures from a One Way Function, tech. rep., https://www.microsoft.com/en-us/research/publication/constructing-digital-signatures-one-way-function/ Accessed 2017-02-20., SRI International Computer Science Laboratory, 1979.

[LM95] Leighton, T.; Micali, S.: Large provably fast and secure digital signature schemes from secure hash functions, U.S. Patent 5,432,852, 1995.

[Mc16] McGrew, D. A.; Kampanakis, P.; Fluhrer, S. R.; Gazdag, S.-L.; Butin, D.; Buchmann, J. A.: State Management for Hash-Based Signatures. In: SSR. Vol. 10074. LNCS, Springer, pp. 244–260, 2016, ISBN: 978-3-319-49099-1.

[MCF19] McGrew, D.; Curcio, M.; Fluhrer, S.: Leighton-Micali Hash-Based Signatures, RFC 8554, IRTF, Apr. 2019, 61 pp., URL: https://rfc-editor.org/rfc/rfc8554.txt.

[Me79] Merkle, R. C.: Secrecy, Authentication and Public Key Systems, PhD thesis, Dept. of Electrical Engineering, Stanford University, 1979.

[Mo15] Mosca, M.: Cybersecurity in an era with quantum computers: will we be ready?, Cryptology ePrint Archive, Report 2015/1075, https://eprint.iacr.org/2015/1075, 2015.

[NI16] NIST: Federal Register Vol. 81, No. 244, Dec. 2016.

[NI17] NIST: Post-Quantum Cryptography: Round 1 Submissions, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions, Last access on 2018-03-23, Mar. 2017.

[NI19] NIST: Post-Quantum Cryptography: Round 2 Submissions, https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-2-Submissions, Last access on 2019-04-11, Jan. 2019.