To Be, or Not to Be Stateful: Post-Quantum Secure Boot using Hash-Based Signatures

Alexander Wagner*
Fraunhofer AISEC
Garching, Germany
alexander.wagner@aisec.fraunhofer.de

Felix Oberhansl*
Fraunhofer AISEC
Garching, Germany
felix.oberhansl@aisec.fraunhofer.de

Marc Schink*
Fraunhofer AISEC
Garching, Germany
marc.schink@aisec.fraunhofer.de

ABSTRACT

While research in post-quantum cryptography (PQC) has gained significant momentum, it is only slowly adopted for real-world products. This is largely due to concerns about practicability and maturity. The secure boot process of embedded devices is one scenario where such restraints can result in fundamental security problems. In this work, we present a flexible hardware/software co-design for hash-based signature (HBS) schemes which enables the move to a post-quantum secure boot today. These signature schemes stand out due to their straightforward security proofs and are on the fast track to standardisation. In contrast to previous works, we exploit the performance intensive similarities of the stateful LMS and XMSS schemes as well as the stateless SPHINCS$^+$ scheme. Thus, we enable designers to use a stateful or stateless scheme depending on the constraints of each individual application. To show the feasibility of our approach, we compare our results with hardware accelerated implementations of classical asymmetric algorithms. Further, we lay out the usage of different HBS schemes during the boot process. We compare different schemes, show the importance of parameter choices, and demonstrate the performance gain with different levels of hardware acceleration.

KEYWORDS

post-quantum cryptography, hash-based signatures, LMS, XMSS, SPHINCS$^+$, secure boot, hardware/software co-design

ACM Reference Format:
Alexander Wagner, Felix Oberhansl, and Marc Schink. 2022. To Be, or Not to Be Stateful: Post-Quantum Secure Boot using Hash-Based Signatures. In *Proceedings of the 2022 Workshop on Attacks and Solutions in Hardware Security (ASHES '22), November 11, 2022, Los Angeles, CA, USA*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3560834.3563831

1 INTRODUCTION

The boot process plays an important role to guarantee the security and trustworthiness of modern electronic devices. The first piece of software that is executed is stored in read-only memory (ROM). This boot code is the first step of a process called *secure boot* which ensures that only trusted and genuine software is executed from the very beginning. The importance of this role combined with the inability to update requires foresighted design decisions and carefully developed software. This is especially relevant, but not limited, to the used cryptographic primitives. Today's state-of-the-art implementations rely on common asymmetric algorithms like RSA or ECC [23, 27]. Through the development of quantum computers these algorithms are at risk as attacks based on Shor's algorithm may become feasible in the future [33].

In order to prepare for this threat, a process to standardise quantum-resistant public key cryptographic algorithms was initiated by the National Institute of Standards and Technology (NIST) in 2016 [30]. At the end of the third round, NIST selected the stateless hash-based signature (HBS) scheme SPHINCS$^+$ for standardization [29]. It is the only selected algorithm not relying on the security of structured lattices. Stateless schemes can be used in the same manner as common digital signature algorithms based on RSA and ECC. In contrast, stateful schemes require the signer to keep track of the already used keys as only a limited amount of signatures can be generated per key pair [25]. Any failure to do so seriously degrades the security [13]. The advantage of stateful schemes over stateless schemes is the smaller signature size and faster runtime. For the two stateful HBS schemes, Leighton-Micali Hash-Based Signature (LMS) and extended Merkle signature scheme (XMSS), IETF RFCs are available [14, 26]. Based on these documents, NIST published a recommendation for the use of stateful HBSs in 2020 [10]. In 2022, recommendations for deployment of HBSs were published by the ANSSI and for stateful HBSs by the BSI. The soundness of HBSs relies only on the properties of the underlying hash functions. As hash functions are well understood, this makes HBS schemes a very conservative choice, in particular when compared to other post-quantum cryptography (PQC) algorithms [5, 24]. Due to this and their maturity, hybridation is not required [2, 7]. This makes HBSs a perfect fit for secure boot.

While maturity is not an area of concern for HBS schemes, practicability is. In case of secure boot, the startup time and accordingly the signature verification is of major importance. The verification time of hash-based signatures is mostly determined by the underlying hash function. To enable HBS schemes for secure boot, we propose a hardware/software co-design with minimal additional hardware overhead. Thus, allowing an immediate drop-in replacement of hash hardware accelerators. For evaluation purposes, we integrate our hardware accelerator in the OpenTitan . OpenTitan is a reference design for open source security controller and is based on a 32-bit RISC-V processor. We use the secure boot process of the OpenTitan to compare our implementations against the existing hardware-accelerated signature verifications based on RSA and ECC. Further, we lay out why a singular focus on either stateful or

*The authors contributed equally to this work.

ASHES '22, November 11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9884-8/22/11.
https://doi.org/10.1145/3560834.3563831

stateless schemes is a blocker for real world applications. In short, this is due to the fact that for most products different constraints are applicable. In the context of the secure boot, this is even more evident, as the successive boot stages mean that different entities and their respective constraints are involved. We overcome this issue with our flexible co-design accelerating stateless as well as stateful schemes from boot up.

In recent years, stateful schemes were evaluated for a usage in secure boot [19, 21] or for general purpose usage with efficient implementations [9, 12, 34, 35]. These implementations range from software evaluations including comparisons of different schemes [9, 18] to System-on-Chips (SoCs) with different levels of hardware acceleration [12, 19, 35] and full hardware designs [21, 34].

A flexible HBS solution for secure boot is still missing from state-of-the-art literature. Specifically, our paper provides the following contributions:

- A flexible hardware accelerator with support for stateful and stateless schemes, namely LMS, XMSS and SPHINCS$^+$ (also referred to as SPX$^+$ in the following)
  - Modular design approach to enable different levels of acceleration
  - Utilization of algorithmic similarities to achieve low hardware overhead
  - Drop-in replacement/extension for generic hash cores
- Evaluation of algorithmic accelerations and trade-offs for hardware/software co-designs
- Exploration of HBS parameter trade-offs for the secure boot scenario
- Benchmark results in the context of secure boot with respect to code size, signature and key size, speed and area requirements
- Rust software implementations for all evaluated algorithms

Our implementations are open source under an Apache 2 license at https://github.com/Fraunhofer-AISEC/hbs-pq-secure-boot.

2 HASH-BASED SIGNATURE SCHEMES

Hash-based signatures are digital signature schemes that use hash functions as the main cryptographic primitive.

2.1 Classification of Hash-Based Signature Schemes

The way how and which of these cryptographic primitives are combined allows to build either a so-called *stateful* or *stateless* HBS scheme. The usage of a stateful HBS algorithm differs for the signer compared to common asymmetric cryptographic algorithms. In essence, the number of signatures that can securely be generated is limited by the total number of key pairs available. Each key pair can only be used once and any reuse would result in a security compromise [13]. Thus, the signer has to keep track of the already used key pairs, effectively storing a state and updating it after every signature generation, i.e. stateful [25]. Contrarily the usage of stateless HBSs is in line with classical asymmetric cryptographic algorithms.

Aside from stateful and stateless, HBS schemes can be divided into the two variants, "simple" and "robust", for which different security arguments in relation to the used hash function can be made.

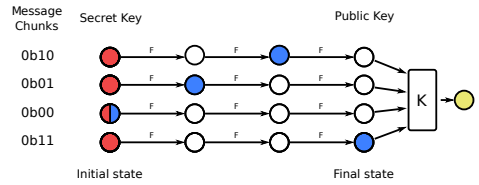


Figure 1: The Winternitz OTS with the secret key 🔴, the compressed public key 🟡, and the signature 🔵.

The "simple" instantiations have a less conservative security argument but a better performance. In contrast, "robust" instantiations meet more conservative security requirements and consequently require more hash operations. The reader is referred to [10, 15] for more information on the security arguments and its details.

2.2 One-Time Signatures

The foundation for contemporary HBS algorithms are the one-time signature (OTS) schemes. LMS, XMSS and SPHINCS$^+$ each use variants of the Winternitz OTS (WOTS). The core idea is to have a certain amount of function chains, i.e. repeatedly applying a function F to the prior output. For the sake of convenience, we assume that the function F only consists of a single call to a cryptographic hash function with the prior output as single input. Thus, in the following we simply use the term hash chain for this specific construct. The working principle of such OTS schemes is depicted in Figure 1. The starting point of a hash chain is a random value and corresponds to one OTS secret key 🔴. An intermediate value of a hash chain is an OTS signature 🔵. The end point of a hash chain is an OTS public key. The function K is applied to these end points to generate the compressed OTS public key 🟡. For the compression function K, LMS and SPHINCS$^+$ use a tweakable hash function and XMSS a so-called L-tree. The signing and verifying operations are inherently similar. To sign or verify a message, its digest is split into chunks of $log_2(w)$ bits and each chunk is interpreted as value a. For **signing** 🔴→🔵 and **verifying** 🔵→🟡 each hash chain is advanced. For signing it is advanced by a and for verifying by $w-1-a$. In the case of verification, a signature is valid, if the public key candidate is equal to the public key.

The function chain length and the chunk bit-size is defined by the Winternitz parameter w and $log_2(w)$, respectively. In the example in Figure 1, the message is split into chunks of 2 bit which corresponds to a Winternitz parameter w of 4. For more details with respect to implementation the reader is referred to [11].

2.3 Few-Time Signatures

In contrast to the OTS scheme a few-time signature (FTS) scheme allows the reuse of a key pair for a few times. The FTS scheme is only used in the stateless HBS scheme SPHINCS$^+$. As a result the total tree height of SPHINCS$^+$ can be reduced significantly, making it applicable in practice [3]. In the context of SPHINCS$^+$, the forest of random subsets (FORS) scheme is used to sign message digests. FORS consists of k Merkle trees each having a tree height of a. To generate the FORS public key 🟡 all k Merkle tree root nodes are compressed with a hash. A single tree authenticates $t = 2^a$

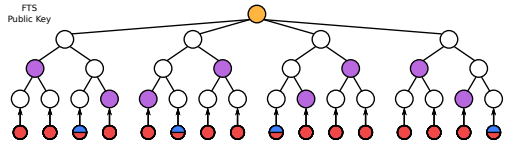


Figure 2: Overview of the forest of random subsets scheme with the secret key ●, the public key ●, the signature ●, and the authentication path nodes ●.

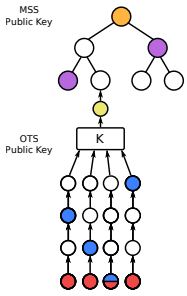


Figure 3: Merkle signature scheme.

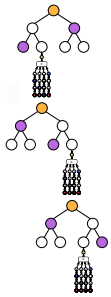


Figure 4: Generalized Merkle signature scheme.

FORS secret keys ●. Thus, the leaves are the hashes of the secret keys. To generate a signature the message digest is split into k a-bit chunks, as shown in Figure 2. Each chunk is interpreted as an integer which is used as an index to select a secret key as signature node ●. This is done for all k trees and chunks. The selected nodes are aggregated together with the respective authentication path nodes ●. For verification the signature is used to generate a public key candidate similar to the WOTS signature verification [15]. To allow for comparison, the same message is signed in the OTS and the FORS example, depicted in Figure 1 and Figure 2, respectively.

2.4 Merkle Signature Scheme

With an OTS or FTS scheme, one key pair can be used once or a few times to sign, respectively. In order to overcome this limitation, the Merkle signature scheme (MSS) is used, depicted in Figure 3. It applies a Merkle tree to authenticate multiple OTS public keys. Every leaf node in the Merkle tree corresponds to a single hashed OTS public key. The root node of the tree corresponds to the MSS public key, which is used to authenticate the OTS public keys. A Merkle tree with a tree height h authenticates 2^h OTS key pairs [28].

A MSS signature consists of an OTS signature, introduced in Section 2.2, and, in the context of HBS, a so called *authentication path*. Starting at the bottom of the figure, the OTS public key ● is calculated from the signature ●. Then the authentication path nodes ● are used to generate the public key candidate ● for the respective signature. The signature is valid, if the public key candidate is equal to the known public key.

With the MSS an OTS can be extended to a many-time signature (MTS). Such a HBS construct is applicable to real world use cases, but still impractical for a high number of key pairs, i.e. required signatures. The tree height h is limited by the runtime of key generation and signing. To overcome this limitation the generalized Merkle signature scheme (GMSS) was introduced in [8]. Its core idea is to build a so-called certification tree with multiple MSSs. Instead of having a single MSS with a large Merkle tree, it is split into d MSSs each being a smaller Merkle tree. At the top of this certification tree is the root MSS, which signs its child MSSs. At the bottom is the leaf MSS, which is used to sign the message.

For SPHINCS$^+$, GMSS is especially relevant, as the total required tree height h is vast [15]. Different to the exemplary overview in Figure 4, SPHINCS$^+$ uses a FTS scheme instead of an OTS scheme in the bottom layer, as explained in Section 2.3.

3 SECURE BOOT

In this section, we describe the different stages and involved entities of a secure boot process and show how real world applications benefit from our hardware/software co-design and its flexibility to support stateful and stateless signature schemes. Further, we derive parameters for all considered HBS schemes tailored for the secure boot use case.

A secure boot process consists of multiple stages which are executed one after the other until the application is started. To ensure that only genuine software is executed, every boot stage verifies the next stage before it hands over execution. Each stage is associated with an individual role belonging to a different entity. The stages and roles are based on the specification of the boot process for the OpenTitan project, but apply to most use cases. The involved roles in the boot process are *silicon creator*, *silicon owner*, and *provider*.

The *silicon creator* is responsible for the first stage of the boot process and thus the root of trust and security for the entire device. As the software for this boot stage is stored in ROM, it cannot be updated after the chip production. Thus, foresighted decisions with respect to the chosen signature scheme and a secure implementation are of utmost importance. The *silicon owner* is the entity that uses the hardware and provides, for example, the kernel or operating system (OS) for the silicon. This boot stage is usually stored on an updateable non-volatile memory. The application of the product is the last stage of the boot process. The *provider* is responsible for this boot stage.

Each entity is responsible to provide valid signatures for their respective boot stages. Thus, the key generation, maintaining the key material and signing needs to be handled individually by each entity. Depending on the constraints and capabilities each entity may decide to use stateful or stateless HBSs.

3.1 To Be, or Not to Be Stateful

Stateful HBS schemes allow for fast verification with small signatures. The drawback of a stateful HBS scheme is the requirement to maintain its state. Best practices for state management were evaluated in [25]. In essence, three different approaches were proposed and outlined. Two of them requiring dedicated cryptographic hardware, like hardware security modules (HSMs), which are capable of securely synchronizing the state. The third approach uses a combination of a stateful and a stateless HBS scheme. This approach does not require any cryptographic hardware, but has the drawback that the verifier must perform a stateless and a stateful signature verification. Hence, this is not of general interest for the

secure boot use case. In general, state management is expensive and any implementation must guarantee high assurance. Depending on the respective application, entities cannot generally bear this overhead. Thus, the question on whether the HBS scheme should be stateful or not must be answered differently, depending on the application, but also the entity. Since every boot stage is controlled by an individual entity and *owner* and *provider* may have different constraints with respect to boot time and application security, some devices need to verify stateful signatures at one stage and stateless ones at another. Thus, our architecture that supports both types is the most promising approach to enable the transition to a post-quantum secure boot using hash-based signatures.

3.2 Choice of Hash-Based Signature Parameters

The general impact of the HBS parameters are formally described in [18]. Within the scope of this work, we select the parameters in accordance with the constraints of secure boot. The SPHINCS$^+$ submission document defines fixed parameter sets. This is in contrast to stateful schemes, where parameters can be selected more freely. In the following section, we provide an overview over the parameters and set forth the rationale behind our parameter choice.

Table 1: Signature and public key sizes and the reached NIST security level, in comparison to RSA and ECC, for LMS, XMSS and SPHINCS$^+$ with SHA-256 as underlying hash function and an output size of 32 bytes.

Algorithm	Parameter		Signature size	Public key size	NIST security level
LMS	$h = 15$	$w = 4$	4.7 KiB	32 B	5
	$h = 15$	$w = 16$	2.7 KiB	32 B	5
	$h = 15$	$w = 256$	1.6 KiB	32 B	5
XMSS	$h = 16$	$w = 16$	2.6 KiB	32 B	5
SPX$^+$	256*s*		29 KiB	64 B	5
RSA	3072		384 B	384 B	⚠
ECC	$P - 256$		64 B	64 B	⚠

Hash Algorithm. LMS, XMSS and SPHINCS$^+$ have support for both SHA-2 and SHA-3. Due to the widespread use of SHA-2, we select the SHA-256 hash function with an untruncated output size of 32 bytes. This guarantees a level of security equivalent to an exhaustive key search for AES-256 [15], thus reaching NIST's highest security level five. If Grover's attack is feasible, this equals a level of 128 bits in a pre-quantum world. We select for ECC a 256-bit curve and for RSA 3072-bit integers (Table 1) for comparison with commonly used asymmetric algorithms.

LMS and XMSS. Due to the nature of stateful HBSs the ability to sign firmware images is limited to a fixed count, which is defined once at key generation (Section 2.4). For the secure boot use case we estimate the required number of firmware updates including a security margin. We estimate the maximum lifetime of a security controller to be 40 years and the amount of required updates to at most two updates each day during its lifetime. This totals to up to 29200 required signatures, where one signature is used for each firmware update. From this we can derive the required tree height

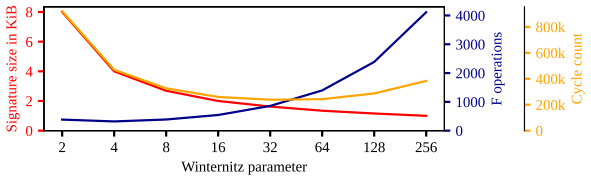


Figure 5: Cycle count estimation for a signature verification of a WOTS accelerated by our hardware/software co-design depending on different Winternitz parameters and in comparison to the signature size and the required F operations.

parameter for LMS to be $h = 15$ and XMSS to be $h = 16$, with respect to the parameters listed in [10]. For XMSS, the Winternitz parameter w is limited to $w = 16$. For LMS, the Winternitz parameter is $W \in [1, 2, 4, 8]$, which maps to $w \in [2, 4, 16, 256]$. The notation of w is used for SPHINCS$^+$ and XMSS, so we will use this notation as well for LMS. As shown in Figure 5, the Winternitz parameter allows a trade-off between signature size and overall performance. A higher Winternitz parameter generates smaller signatures, but has the drawback of worse performance. This is not the case for $w = 2$, as for both $w = 2$ and $w = 4$ the required hash compress calls add up to the same count, while the signature for $w = 2$ is larger. Therefore, the original Winternitz parameter set can be limited to $w \in [4, 16, 256]$. The resulting signature sizes of the selected parameters for LMS and XMSS are listed in Table 1.

SPHINCS$^+$. In contrast to XMSS and LMS, the SPHINCS$^+$ parameters are described with certain sets of parameter combinations. This is due to the fact, that the stateless property of SPHINCS$^+$ is a result of carefully combining parameters. Thus, we restrict our evaluations to the provided parameter sets. The available list of parameters can be split into the two variants, "small" and "fast", which are denoted with "s" and "f", respectively. The "small" variant has the drawbacks of slower key generation and signing. However, it achieves smaller signature sizes and faster verification. As this is desirable for the secure boot scenario, we select the "small" variant. The respective signature and public key size for the selected parameter set is listed in Table 1. The "simple" and "robust" construction that can be used in SPHINCS$^+$ only influence the security proof and runtime but not signature or public key sizes. They are referred to as SPX$^+$-s and SPX$^+$-r in the following.

4 HARDWARE/SOFTWARE CO-DESIGN

To ensure the practicability of HBS schemes during runtime, we propose a flexible hardware/software co-design to enhance the performance of signature verification. In this chapter we lay out and evaluate our approach.

4.1 Software implementation

Within the following section, we lay out our design methodology starting with a peak into the general performance characteristics of LMS with $w = 16$, XMSS and SPHINCS$^+$. As depicted in Figure 6, we differentiate the operations of the algorithms into the three classes: hash chain, authentication path, and unclassified. The unclassified

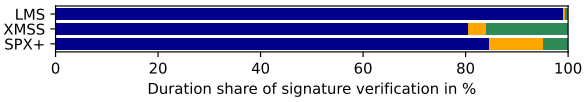


Figure 6: Performance of software implementations impacted by hash chain, merkle tree, and unclassified operations.

part was not further split as it takes up less than 10 % on average. Hence, it is less important for hardware acceleration. The hash chain computation is responsible for over 80 % of the overall latency during a signature verification. Therefore it is the most interesting part for acceleration by dedicated hardware. The authentication path takes up to 15 % of performance for SPHINCS$^+$, making it a possible second target for acceleration.

Hash Hardware Accelerators. In general, hashing dominates the execution time of the HBS algorithms. Thus, any acceleration within the hash computation has a meaningful impact on the overall performance. SHA-2/3 hardware cores are found in many microcontrollers, as the algorithm is frequently required and their permutation logic can be implemented efficiently in hardware. The usage of such an accelerator shifts the bottleneck from computation to communication with the accelerator. On our target platform, the OpenTitan, a SHA-256 compress takes 65 clock cycles, while writing the data and reading the digest raises the latency to around 1400 clock cycles. For digesting a high amount of data this is irrelevant, as the transfer interleaves with computation and the compress function is executed multiple times. However, for a step within a hash chain 55 (LMS) to at most 96 bytes (XMSS) are digested at once. Therefore, a generic hash accelerator is not ideal for usage in hash chains.

4.2 Hardware Hash-Based Signature Accelerators

Acceleration that can be achieved with generic hash cores is limited due to a high communication overhead. Due to the tree and chain structures in HBS schemes, data structs are often accessed subsequently. Dedicated hardware components can manage this data flow and consequently reduce interaction with the main processor. Therefore, we propose a set of HBS top modules that support the computation of a hash chain, the computation of a tree root, or both. We use the open source SHA-2/HMAC core from the OpenTitan project as hash backend. Throughout the rest of this paper we refer to our proposed design as SHA-2$^+$ core.

Winternitz parameter exploration for HW/SW Co-Designs. To assess the impact of our approach, we estimate the resulting cycle count for an OTS signature verification on our design in Figure 5. As stated in Section 2, OTS verification consists mainly of advancing hash chains. The estimation shows that a hash chain module allows to use higher Winternitz parameters without significantly degrading the overall performance. As the chain length increases, the number of required I/O operations reduces, so runtime improves even if more hash operations are required. This is in direct contrast to software implementations where an increase of required F operations implies a linear increase in runtime. Interestingly, our estimate shows that a hardware/software co-design approach enables to reduce the signature size without affecting the performance. For example, a Winternitz parameter of $w = 256$ can be selected instead of $w = 16$, to cut the OTS signature size into half without a significant performance regression.

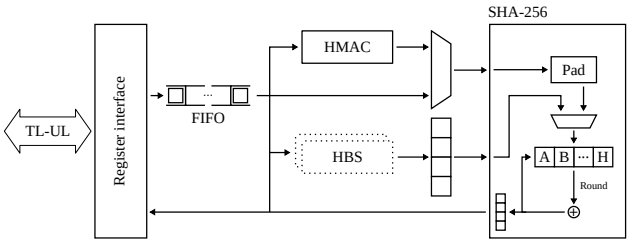
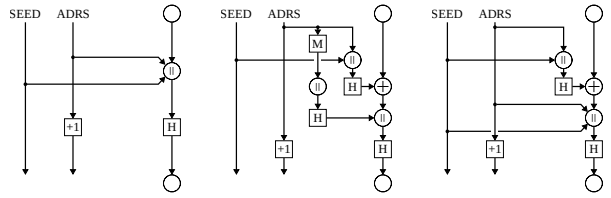


Figure 7: Block diagram of the SHA-2$^+$ core.

SHA-2$^+$ core. Figure 7 shows our hardware design. The original OpenTitan SHA-256 accelerator consists of a SHA-256 backend, which can be accessed either transparently or through the HMAC toplevel, to compute a SHA-256 or a MAC, respectively. By reusing the SHA-256 logic, we minimize the additional cost for manufacturers. At synthesis time one of six HBS modules can be plugged into the design: (i) LMS, (ii) XMSS, (iii) SPX$^+$-s, (iv) SPX$^+$-r, (v) SPX$^+$-s + LMS, (vi) SPX$^+$-r + XMSS. In theory, these modules can be fitted for arbitrary hash cores with minor modifications. Our changes to the original hash accelerator include a chain register which is connected to the SHA-256's initial state register, additional control logic to switch between operation modes, and a digest feedback path into the HBS module. The HBS cores implement the respective behaviour by means of simple state machines. Our complete design can be directly integrated into any chip that supports the TileLink Uncached Lightweight (TL-UL) interface.

Combined stateful and stateless accelerators. The rationale for supporting both stateless and stateful signatures in one design was laid out in Section 3.1. The obvious choices for combination are LMS and SPX$^+$-s and XMSS and SPX$^+$-r, as they are respectively "simple" and "robust" instantiations of HBSs. Figure 8 shows the steps in a "simple" hash chain (LMS and SPX$^+$-s), a "robust" hash chain in XMSS, and a "robust" hash chain in SPX$^+$-r. The only difference between LMS and SPX$^+$-s is that the former uses 23 address bytes, whereas the latter uses 22 bytes and the initial SHA-256 state. In theory, both the "simple" and "robust" SPHINCS$^+$ variant would require to hash a public seed before each hash operation. This seed is padded such that this compress only needs to be done once and the resulting SHA-256 state can be reused for subsequent hash operations. All SPHINCS$^+$ cores support this behaviour. Apart from that, advancing a simple hash chain requires a single compress with an incrementing iterator and the output of the previous step. The differences in SPX$^+$-r and XMSS are more severe. WOTS$^+$ in XMSS requires to first compute a unique key and mask, where the hash function acts as a pseudorandom function, and finally to update the chain digest, for which the hash function acts as a keyed hash function. The SPX$^+$-r construct omits the calculation of a unique key. In XMSS, 32 address bytes are required, SPHINCS$^+$ uses 22



(a) LMS & SPX$^+$-s **(b) XMSS** **(c) SPX$^+$-r**

Figure 8: Schematic depiction for a "simple" hash chain (LMS & SPX$^+$-s) and a "robust" hash chain in XMSS and SPX$^+$-r.

bytes. Therefore, in straightforward implementations, six compress iterations are required in XMSS, two each for the mask, key, and advancing the chain, and three compress iterations are required in SPHINCS$^+$, two for the mask, and one for advancing the chain. While the two "robust" instantiations in XMSS and SPHINCS$^+$ do not map as good as the two simple instantiations, the buffer registers and some control logic can be reused.

XMSS precomputation. The amount of compress iterations in an XMSS WOTS$^+$ hash chain step can be reduced via precomputation [35]. Three data structures are computed in a step, the mask, the key, and the data for the chain itself. For all three data structures, a public seed and an address must be compressed first. Since the public seed is constant, and the address is constant within a step, this compress only needs to be done once instead of three times. The resulting state can be used to overwrite the SHA-256 state in the subsequent hashes, thus lowering the number of compress iterations from six to four. This comes at the cost of a buffer register and overwrite logic for the SHA-256. As SPHINCS$^+$ requires this behaviour as well, this applies only to the standalone XMSS core.

Accelerating root computation in Merkle trees. In addition to accelerating hash chains, we also explore hardware optimizations for the *compute root* operation in Merkle trees. We limit this exploration to the FORS and MSS scheme in SPHINCS$^+$. Due to the usage of multiple tree levels in SPHINCS$^+$, more authentication path calculations than in LMS and XMSS are required (Figure 6). Further, related work has shown that hardware features beyond hash chain computation offer little to no additional benefit for verification in XMSS [35], and as the performance of SPHINCS$^+$ lacks behind the stateful schemes in general, it is most relevant for additional acceleration. Therefore our SPX$^+$-s and SPX$^+$-r modules can be extended with a *MerkleTree* submodule. The *compute root* operation calculates the root of a tree from a leaf and the respective authentication path. Through all tree levels, two child nodes are combined to obtain their parent node with a hash function. In SPX$^+$-s, this corresponds to calculating one digest by compressing both child nodes with two compress iterations. For SPX$^+$-r, two compress iterations to obtain a 64 byte mask are required. Afterwards, the masked data is compressed within two more iterations. The order in which the nodes are hashed depends on whether the node from the authentication path is a left or right neighbour. Our core continuously absorbs nodes from the authentication path, reorders them with the node buffered in hardware and computes the hash to obtain the parent node. Only the root node is read from the SHA-2$^+$ core, thus dispensing all but one read operation.

Synthesis results. In Table 2, we report FPGA and ASIC synthesis results for all configurations of the SHA-2$^+$ core. As FPGA target, we chose an Artix-7, synthesis and implementation are done with default settings in Vivado 2020.2. For ASIC synthesis, we use the SG13s 130 nm process by IHP and the Cadence Genus synthesis tool in version 21.10-p002_1. The OpenTitan SHA-256/HMAC core we use as basis consumes around 56.3 kGE. The SHA-256 logic and its registers amount to slightly more than 50 % of that. The next largest blocks are the register interface, the HMAC logic, and the FIFO, with approx. 20 %, 13 %, and 8 %, respectively. We prove that integrating a HBS accelerator for LMS can be as cheap as 11 kGE additional gates, an approximate overhead of 20 %. Extending this to a design that also supports SPHINCS$^+$ costs an additional 5 kGE, mainly due to the buffer register for the initial SHA-256 state. Additionally integrating the *MerkleTree* accelerator is comparatively expensive, as the behaviour differs fundamentally and registers to parse leafs and buffer a digest are required. The "robust" instances are more expensive, as mask and key registers are required. Integrating SPX$^+$-r requires no additional sequential logic, but 5 kGE in combinatorial logic, as the differences in the hash chain step are more severe. Supporting the *MerkleTree* computation in the "robust" variant on top requires even more buffer registers and additional control logic. This becomes even more obvious if a standalone SPX$^+$-r core is built, as the hash chain core in SPHINCS$^+$ requires less registers than the XMSS core.

Table 2: Synthesis results for different configurations of the SHA-2$^+$ core on FPGA on Artix7 and ASIC using the IHP 130 nm process SG13S [17]

	FPGA		ASIC		Overhead
	LUTs	FFs	Area (mm^2)	GE	(%)
SHA2, HMAC	3480	2400	0.319	56,300	-
+ LM-OTS	4400	3080	0.381	67,200	20
+ SPX$^+$-s	5060	3360	0.414	73,000	30
+ MerkleTree-s	5920	4070	0.485	85,600	52
+ WOTS+	4840	4000	0.464	81,800	45
+ SPX$^+$-r	5950	4000	0.492	86,800	55
+ MerkleTree-r	6210	4170	0.575	101,400	80
+ SPX$^+$-s	4870	3350	0.407	71,800	28
+ MerkleTree-s	5850	4050	0.476	84,000	49
+ SPX$^+$-r	5510	3860	0.458	80,800	44
+ MerkleTree-r	6050	4120	0.560	98,800	75

Our synthesized ASIC can be clocked at 125 MHz at most. The target frequency of the OpenTitan is 100 MHz. Even with the most complex SHA-2$^+$ synthesis configuration the critical path of the design is still determined by the SHA-256 round logic. The FPGA synthesis supports this finding, as the length of the critical paths deteriorates only minimally. On a medium-sized Artix-7 board with almost 50k LUTs, all our designs can be clocked at 100 MHz.

Table 3: State-of-the-art HBS implementations.

	Algorithm	Approach	Area		Performance	
			FPGA		MHz	ms
[9]	LMS $w = 16, h = 10$	SW ARM Cortex-M4	-		24	110
[19]	LMS $w = 265, h = 15$	FPGA	968 517 2 1	LUTs FFs BRAMs DSP	250	6.3
[9]	XMSS $w = 16, h = 10$	SW (rap. verif.) ARM Cortex-M4	-		24	273
[35]	XMSS $w = 16, h = 10$	HW/SW RISC-V RV32IM	2580 1700	Registers ALMs	95.2	5.68
[20]	SPX$^+$ $256s - simple$	SW ARM Cortex-M4	-		24	729
[20]	SPX$^+$ $256s - robust$	SW ARM Cortex-M4	-		24	2070

5 BENCHMARK RESULTS

We evaluate our flexible hardware accelerator in general and with respect to the secure boot use case defined in Section 3. As the verification time for hash-based signatures is not constant and depends on the signed message, we evaluate each parameter set and implementation with 1000 different messages. For a baseline, we start the comparison with our open source software implementations of LMS, XMSS and SPHINCS$^+$. Within this comparison all implementations are written in Rust and are not optimized for the RISC-V target architecture. We extend this by benchmarking the implementations accelerated by the available SHA-256 core as well.

5.1 LMS and XMSS

First, we evaluate the performance of signature verification for the stateful signature schemes LMS and XMSS. For this, we benchmark the verification in software running on the Ibex processor, accelerated by the general-purpose SHA-256 core, and accelerated by our SHA-2$^+$ core. The parameters of interest are listed in Table 1. The results are shown in Figure 9.

For LMS, the relative performance improvements for our approach differ from the relative ranking of a software implementation with or without general-purpose SHA-256 core. Our results show that the benefit to use a hardware/software co-design depends on the Winternitz parameter as estimated in Figure 5. Instead of exponentially increasing cycle counts for larger w, our architecture is fastest for $w = 16$. Signature verification with $w = 256$ performs only 25 % slower than for $w = 16$ and even faster than for $w = 4$. Overall, signature verification with $w = 4$ is slowest and signature size is largest. Hence, we further restrict the Winternitz parameter for LMS to $w \in [16, 256]$.

Comparing the XMSS results with the LMS results for $w = 16$ leads to approximately the same relative performance increase. With our accelerator, the performance of XMSS signature verification is increased in comparison to software and the SHA-256 core by a factor of 12 and 4, respectively. Due to the "robust" construction, XMSS is slower by a factor of approximately 6. For LMS with

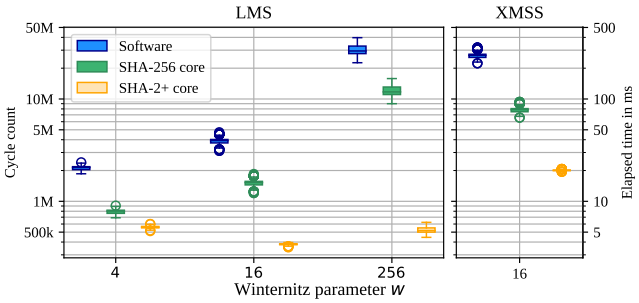


Figure 9: Signature verification time for 1000 different messages with LMS and XMSS. Both are implemented in software and accelerated by a SHA-256 and SHA-2$^+$ core.

$w = 256$, software and the SHA-256 core are outperformed by a factor of 80 and 20, respectively.

The relationship between different Winternitz parameters and performance for hardware/software co-design was never reported in detail. While it is straightforward for standalone hardware or software implementations (e.g. shown in [19]), co-designs require to consider costs for data transfers. For our design, the performance of LMS verification is effectively the same for all w, while the signature size is reduced by up to 50 %. For XMSS this can be estimated to have even a bigger impact as the compression of the OTS public keys with K is performed by an L-tree. Increasing the Winternitz parameter from $w = 16$ to $w = 256$ effectively halves the required node operations within the calculation of the L-tree. As the L-tree generation is computationally expensive, halving the required operations impacts the overall performance. Therefore using Winternitz parameters $w > 16$ in XMSS would be desirable not only for secure boot, but as the standardisation is quite progressed, parameter changes are unlikely to happen.

Rapidly Verifiable Signatures. As it can be seen in Figure 9, the execution of a signature verification is not performed in constant time. Based on this [6, 31] introduced a general algorithmic optimization applicable for stateful HBSs. The approach does not reduce the total number of operations in signing and verifying but shifts computations from the verifier to the signer. The signer searches for a rapidly verifiable signature by appending a random counter to the message. If the resulting signature requires the verifier to perform less F operations to reach the final state, i.e. public key (see Figure 1) in comparison to the original signature, a *rapidly verifiable* signature is found. This is repeated for a distinct number of trials. The random counter that requires the verifier to do the least F operations is used to generate a *rapidly verifiable* signature. For software implementations it has been proven that the performance benefits significantly and thus signer's additional effort is well spent. In the following section, we extend our implementations to give an insight into the impact of this optimization on hardware/software co-designs. Further, we extend the existing knowledge base by applying this optimization technique to LMS, as this was not yet performed.

The results for the rapidly verifiable signatures are plotted in Figure 10. Performance improvements are evaluated up to a maximum

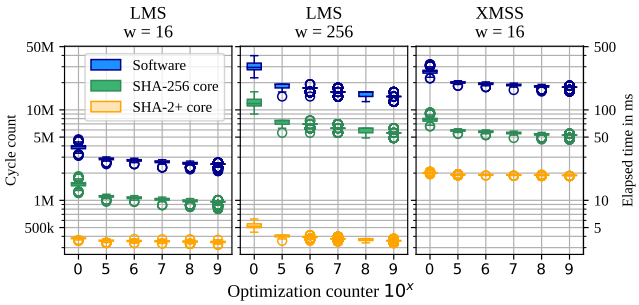


Figure 10: Signature verification time for 1000 different messages with LMS and XMSS. Optimization with the *rapidly verifiable* approach with different optimization counters. Both algorithms are implemented in software and accelerated by a SHA-256 and SHA-2$^+$ core.

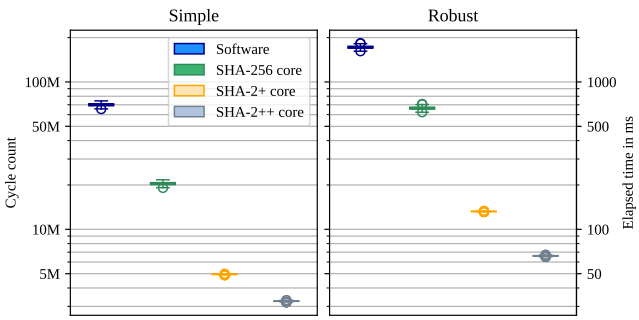


Figure 11: Signature verification time for 1000 different messages with SPX$^+$-s and SPX$^+$-r implemented in software and accelerated by a SHA-256, our SHA-2$^+$ core, and our extended SHA-2$^+$ listed as SHA-2^{++}.

optimization counter equal to 10^9. For LMS and XMSS with $w = 16$ the relative performance improvement is comparable. For our software implementation and the acceleration with the general-purpose SHA-256 core, the optimization decreases the runtime for signature verification by approximately 30 %. In contrast, the runtime of the implementations accelerated by our SHA-2$^+$ core is only decreased by 10 %. For LMS with $w = 256$ comparable observations can be made as the software and the SHA-256 variants have a decrease in runtime by 50 % and the SHA-2$^+$ variant by 30 %. In general, the optimization reduces the runtime. However, the relative improvement is reduced for our hardware/software co-design. This is due to the fact that advancing within the hash chains with our design is not as expensive compared to the rest of the algorithm.

Further, we extend the results reported by [6] for Winternitz parameters $w > 16$ with our benchmark results. A larger Winternitz parameter allows for bigger improvement of the performance. It can be seen that the benefit of using the rapidly verifiable approach for LMS and XMSS with $w = 16$, accelerated by our SHA-2$^+$ core, is neglectable and the effort may not be required by the signer. For larger Winternitz parameters of $w = 256$ it still is a decent improvement of the performance.

5.2 SPHINCS$^+$

The performance of the SPHINCS$^+$ signature verification was evaluated similar to the stateful HBS benchmarks in software, accelerated by a SHA-256 accelerator and by our SHA-2$^+$ core. As presented in Section 4, our hardware accelerator comes in two different variants for the acceleration of SPHINCS$^+$. The basic SHA-2$^+$ core accelerating the hash chain calculation and the extended SHA-2^{++} core including the *MerkleTree* module accelerating the *compute root* calculation in Merkle trees as well. The SHA-2$^+$ core speeds up the signature verification for "simple" and "robust" by approximately a factor of 14. Similar to the stateful HBS benchmarks the execution time varies. The SHA-2^{++} core improves the performance compared to a software implementation for "simple" and "robust" by roughly a factor of 21 and 27, respectively. With respect to the SHA-2$^+$ core the extended SHA-2^{++} reduces the latency by 34 % for "simple" and 45 % for "robust". Considering the relative high hardware utilizations listed in Section 4 makes the extended core only suitable for scenarios with strict timing requirements. As our evaluation in Section 4 has shown, further hardware accelerations are not suitable as the latency is not heavily influenced by the not yet accelerated calculations. This makes our SHA-2$^+$ core a very efficient implementation with low overhead and our SHA-2^{++} core more performant with the drawback of higher hardware costs.

Similar to benchmarks with LMS it would be interesting to evaluate the impact of a higher Winternitz parameter with $w = 256$. As shown with the LMS benchmarks, the performance can be expected to be constant. This is due to the fact that FORS, which is the main design difference, does not depend on the Winternitz parameter. The update in parameters would reduce the signature size from 29 KiB down to 21 KiB. We did not further investigate $w = 256$ for SPHINCS$^+$ and leave it open for future work.

5.3 Comparison with Related Work

Table 3 summarizes the state of the art relevant to this work. The reported hardware utilizations do not include the SHA-256, but only the relative overhead for supporting HBSs. Complete FPGA implementations such as [1, 4, 21, 34] are out of scope as their use case differs from ours. The software benchmarks in [6, 9, 20] provide interesting comparisons, as the performance of the ARM Cortex-M4 is similar to that of the Ibex. However, in these works, assembly optimized SHA-256 functions are used, thus outperforming our plain Rust implementations. In addition, the authors of [6] use the rapidly verifiable approach for XMSS to lower verification time, thus outperforming the straightforward XMSS implementation of [9] by factor two. The LMS FPGA implementation in [19] supports the verification in hardware with a compact design. In comparison to our hardware/software co-design, their verification takes 20 % longer in terms of absolute latency. The authors of [35] propose different hardware extensions for XMSS but conclude that any acceleration beyond the hash chain computation does not lead to improved performance. A comparison of area utilization is not applicable, as they use a different FPGA architecture. They highly optimize their architecture for XMSS and achieve a latency which is lower by a factor of three.

As stated before, hardware/software co-designs for SPHINCS$^+$ and combined stateful and stateless HBS schemes have not been reported. The comparison with related work demonstrates that our designs improves the state-of-the-art in performance of HBSs and in their adaptability for applications.

5.4 Secure Boot

In this section, we evaluate the applicability of HBSs for a post-quantum secure boot. To put our results into perspective, we include the time required to hash an application firmware image into our comparison. We use the - at time of writing - most recent Git master version[1] of the TockOS kernel as reference firmware [22] and compile it in release mode, which results into a binary size of 124 KiB. To take a minimal application into account, we set the code size to 128 KiB.

We compare both software only and hardware accelerated implementations. We use available open source Rust implementations for RSA[2] and ECC [3]. For comparison with hardware accelerated RSA and ECC we use the OpenTitan BigNumber accelerator (OTBN). The results of these comparisons are listed in Table 4.

Table 4: Cycle count [cc] in MCycles of the signature verification executed on an Ibex processor and overhead relative to hashing a 128 KiB firmware.

	RSA 3072	ECC P-256	LMS w=256	XMSS w=16	SPX$^+$-s 256s	SPX$^+$-r 256s
SW	37.0	50.0	30.1	26.6	70.0	172
$cc_{verif}/cc_{sw-hash}$	141 %	190 %	114 %	101 %	266 %	654 %
OTBN or SHA-2$^+$	1.07	0.617	0.524	2.01	4.95	13.2
$cc_{verif}/cc_{hw-hash}$	75 %	43 %	37 %	142 %	349 %	930 %
SHA-2^{++}	-	-	-	-	3.26	6.61
$cc_{verif}/cc_{hw-hash}$	-	-	-	-	230 %	465 %

In software, stateful HBSs allow for a slightly better performance for signature verification than ECC or RSA. Using a stateless HBS would mean a slight performance degradation. For implementations where the boot timing for classical asymmetric algorithms is not required to be accelerated, it is also not required for HBSs.

The hardware accelerated LMS implementation exceeds the performance of RSA and ECC run on the OTBN. In a secure boot, hashing the firmware would be the crucial part, as the verification time can be reduced to only 37 % of the firmware digest time. For XMSS, the duration for signature verification doubles in comparison to RSA, but still is in a comparable range of execution time. For SPHINCS$^+$ the performance degrades for both our accelerator designs in comparison to the classical asymmetric algorithms performed on the OTBN.

To conclude, the performance of the stateful HBS schemes is comparable to that of classical asymmetric algorithms accelerated by the OTBN. It should be noted that the hardware footprint of our SHA-2$^+$ core is significantly smaller than that of the OTBN. The SPHINCS$^+$ variants degrade the performance of signature verification. However, our accelerator makes the overhead bearable. Digesting the firmware and verification with SPX$^+$-s can be achieved below 5 MCycles.

[1] https://github.com/tock/tock/commit/db7cb5fc815ba3c5fa45310dab730b6e5ffa4243
[2] https://github.com/RustCrypto/RSA
[3] https://github.com/RustCrypto/elliptic-curves/tree/master/p256

Table 5: Code size of our Rust libraries compiled for the Ibex and optimized for runtime.

	LMS	XMSS	SPX$^+$-s	SPX$^+$-r
Size in KiB	4.8	11.5	15.9 (+ 40.1)[a]	20.8 (+ 40.1)[a]

[a] SPHINCS$^+$ with SHA-256 as hash function additionally requires SHA-512 to reach NIST security level 5 [16, 32]. This is due to a shortcoming in the initial SPHINCS$^+$ specification.

Code size. In general, the boot ROM size is constrained. Within the OpenTitan the boot ROM has a size of 32 KiB. To further allow to evaluate the applicability of HBSs, we list the required code size for our software libraries in Table 5. All HBS libraries are compiled with optimizations for runtime.

6 CONCLUSION AND OUTLOOK

In this work, we show that the transition to a post-quantum secure boot using HBS schemes is feasible for today's designs. In contrast to other works, we provide a flexible hardware/software co-design to support both stateful as well as stateless schemes from boot up. We demonstrate that by exploiting similarities of LMS or XMSS and SPHINCS$^+$ low hardware overheads can be achieved. Hence, making the discussion to choose between stateless and stateful HBSs one indifferent to the underlying hardware. Further, our design allows to easily incorporate updates with respect to the parameters without changes to the hardware design.

Regarding the parameter sets that should be chosen for secure boot with HBSs, we come to the following conclusions: As NIST views both "simple" and "robust" constructs as secure [10] we recommend the usage of LMS and SPHINCS$^+$-s, due to their advantageous performance. Our design demonstrates that both can be implemented with a minimal hardware footprint. The synergy between LMS and SPHINCS$^+$-s makes them ideal for a flexible architecture. Although, it should be noted that the small differences between the OTS in LMS and SPHINCS$^+$-s make the design more complicated than it needs to be. For XMSS and SPHINCS$^+$-r this is even more obvious. From an implementers perspective, a uniform approach for all "simple" and all "robust" HBS constructs would be desirable. We established that our architecture allows to choose $w = 256$ for LMS without a significant performance penalty. Due to the small signature size, this is our Winternitz parameter of choice.

We would like to highlight that our design is suited for fast transition, as well as being a starting point for further research, in particular for designs and parameter explorations of SPHINCS$^+$. NIST even requested public feedback for new SPHINCS$^+$ parameter sets [29]. While their focus is a lower number of maximum signatures, we suggest to revisit higher Winternitz parameters. As shown in this work, on hardware/software co-designs this does not degrade the performance but reduces the signature size. With this paper we put forward that hardware/software co-designs are

the most relevant solution for problems like secure boot. Software only implementations will not be suitable for embedded devices due to timing requirements and full hardware implementations are very expensive in terms of overhead and cost. Dedicated hardware/software co-designs for SPHINCS$^+$ are largely unexplored. With our flexible HBS design for secure boot we hope to motivate more research in this direction.

Acknowledgements. This work was partly funded by the German Ministry of Education, Research and Technology in the context of the project Aquorypt (reference number 16KIS1018). We thank Chris Gourley, Yongkui Han, Steve Rich, Chris Shenefiel, and Chirag Shroff from CISCO Systems, Inc. for the valuable discussions and feedback, especially on LMS.

REFERENCES

[1] Dorian Amiet, Lukas Leuenberger, Andreas Curiger, and Paul Zbinden. 2020. FPGA-based SPHINCS+ Implementations: Mind the Glitch. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, Kranj, Slovenia, 229–237. https://doi.org/10.1109/DSD51259.2020.00046

[2] ANSSI. 2022. ANSSI views on the Post-Quantum Cryptography transition. Retrieved April 6, 2022 from https://www.ssi.gouv.fr/en/publication/anssi-views-on-the-post-quantum-cryptography-transition/

[3] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. 2015. SPHINCS: Practical Stateless Hash-Based Signatures. In *Advances in Cryptology – EUROCRYPT 2015*, Elisabeth Oswald and Marc Fischlin (Eds.). Vol. 9056. Springer Berlin Heidelberg, Berlin, Heidelberg, 368–397. https://doi.org/10.1007/978-3-662-46800-5_15 Series Title: Lecture Notes in Computer Science.

[4] Quentin Berthet, Andres Upegui, Laurent Gantel, Alexandre Duc, and Giulia Traverso. 2021. An Area-Efficient SPHINCS$^+$ Post-Quantum Signature Coprocessor. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)* (2021-06). IEEE, Portland, OR, USA, 180–187. https://doi.org/10.1109/IPDPSW52791.2021.00034

[5] Ward Beullens. 2022. Breaking Rainbow Takes a Weekend on a Laptop. https://eprint.iacr.org/2022/214

[6] Joppe W. Bos, Andreas Hülsing, Joost Renes, and Christine van Vredendaal. 2020. Rapidly Verifiable XMSS Signatures. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021, 1 (Dec. 2020), 137–168. https://doi.org/10.46586/tches.v2021.i1.137-168

[7] BSI. 2022. BSI – Technische Richtlinie: Kryptographische Verfahren: Empfehlungen und Schluessellaengen. Retrieved April 6, 2022 from https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile

[8] Johannes Buchmann, Erik Dahmen, Elena Klintsevich, Katsuyuki Okeya, and Camille Vuillaume. 2007. Merkle Signatures with Virtually Unlimited Signature Capacity. In *Applied Cryptography and Network Security*, Jonathan Katz and Moti Yung (Eds.). Vol. 4521. Springer Berlin Heidelberg, Berlin, Heidelberg, 31–45. https://doi.org/10.1007/978-3-540-72738-5_3 Series Title: Lecture Notes in Computer Science.

[9] Fabio Campos, Tim Kohlstadt, Steffen Reith, and Marc Stöttinger. 2020. LMS vs XMSS: Comparison of Stateful Hash-Based Signature Schemes on ARM Cortex-M4. In *Progress in Cryptology - AFRICACRYPT 2020*, Abderrahmane Nitaj and Amr Youssef (Eds.). Vol. 12174. Springer International Publishing, Cham, 258–277. https://doi.org/10.1007/978-3-030-51938-4_13 Series Title: Lecture Notes in Computer Science.

[10] David A. Cooper, Daniel C. Apon, Quynh H. Dang, Michael S. Davidson, Morris J. Dworkin, and Carl A. Miller. 2020. Recommendation for Stateful Hash-Based Signature Schemes. https://doi.org/10.6028/NIST.SP.800-208

[11] C. Dods, N. P. Smart, and M. Stam. 2005. Hash Based Digital Signature Schemes. In *Cryptography and Coding* (2005), Nigel P. Smart (Ed.). Springer, Berlin, Heidelberg, 96–115. https://doi.org/10.1007/11586821_8

[12] Santosh Ghosh, Rafael Misoczki, and Manoj R Sastry. 2019. Lightweight Post-Quantum-Secure Digital Signature Approach for IoT Motes. https://eprint.iacr.org/2019/122

[13] Leon Groot Bruinderink and Andreas Hülsing. 2018. "Oops, I Did It Again" – Security of One-Time Signatures Under Two-Message Attacks. In *Selected Areas in Cryptography – SAC 2017*, Carlisle Adams and Jan Camenisch (Eds.). Vol. 10719. Springer International Publishing, Cham, 299–322. https://doi.org/10.1007/978-3-319-72565-9_15 Series Title: Lecture Notes in Computer Science.

[14] A. Huelsing, D. Butin, S. Gazdag, J. Rijneveld, and A. Mohaisen. 2018. XMSS: eXtended Merkle Signature Scheme. https://datatracker.ietf.org/doc/html/rfc8391.

[15] A. Hülsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, J.-P. Aumasson, B. Westerbaan, , and W. Beullens. 2020. SPHINCS+ - Submission to the NIST post-quantum project, v.3. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[16] A. Hülsing, D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, P. Kampanakis, S. Kolbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, P. Schwabe, J.-P. Aumasson, B. Westerbaan, , and W. Beullens. 2021. SPHINS+ round 3 presentation. https://csrc.nist.gov/CSRC/media/Presentations/sphincs-round-3-presentation/images-media/session-1-sphincs-plus-hulsing.pdf.

[17] IHP. [n. d.]. SiGe-BiCMOS- und Siliziumphotonik-Technologien. Retrieved March 14, 2022 from https://www.ihp-microelectronics.com/de/leistungen/forschungs-und-prototyping-service/mpw-prototyping-service/sigec-bicmos-technologien

[18] Panos Kampanakis and Scott Fluhrer. 2017. LMS vs XMSS: Comparion of two Hash-Based Signature Standards. https://eprint.iacr.org/2017/349

[19] Panos Kampanakis, Peter Panburana, Michael Curcio, and Chirag Shroff. 2021. Post-quantum Hash-Based Signatures for Secure Boot. In *Silicon Valley Cybersecurity Conference*, Younghee Park, Divyesh Jadav, and Thomas Austin (Eds.). Springer International Publishing, Cham, 71–86.

[20] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. 2018. PQM4: Post-quantum crypto library for the ARM Cortex-M4. https://github.com/mupq/pqm4.

[21] Vinay B. Y. Kumar, Naina Gupta, Anupam Chattopadhyay, Michael Kasper, Christoph Krauß, and Ruben Niederhagen. 2020. Post-Quantum Secure Boot. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*. IEEE, Grenoble, France, 1582–1585. https://doi.org/10.23919/DATE48585.2020.9116252

[22] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Levis. 2017. Multiprogramming a 64kB Computer Safely and Efficiently. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP'17)*. ACM, New York, NY, USA, 234–251. https://doi.org/10.1145/3132747.3132786

[23] lowRISC. 2022. OpenTitan Documentation. Retrieved April 7, 2022 from https://docs.opentitan.org/

[24] MATZOV. 2022. Report on the Security of LWE: Improved Dual Lattice Attack. https://doi.org/10.5281/zenodo.6493704

[25] David McGrew, Panos Kampanakis, Scott Fluhrer, Stefan-Lukas Gazdag, Denis Butin, and Johannes Buchmann. 2016. State Management for Hash-Based Signatures. In *Security Standardisation Research*, Lidong Chen, David McGrew, and Chris Mitchell (Eds.). Vol. 10074. Springer International Publishing, Cham, 244–260. https://doi.org/10.1007/978-3-319-49100-4_11 Series Title: Lecture Notes in Computer Science.

[26] D. McGrew and S. Fluhrer M. Curcio. 2019. Leighton-Micali Hash-Based Signatures. https://datatracker.ietf.org/doc/html/rfc8554.

[27] MCUboot. 2022. MCUboot Documentation. Retrieved April 7, 2022 from https://docs.mcuboot.com/

[28] Ralph C. Merkle. 1990. A Certified Digital Signature. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, Gilles Brassard (Ed.). Vol. 435. Springer New York, New York, NY, 218–238. https://doi.org/10.1007/0-387-34805-0_21 Series Title: Lecture Notes in Computer Science.

[29] Dustin Moody, Gorjan Alagic, Daniel C Apon, David A Cooper, Quynh H Dang, John M Kelsey, Yi-Kai Liu, Carl A Miller, Rene C Peralta, Ray A Perlner, Angela Y Robinson, Daniel C Smith-Tone, and Jacob Alperin-Sheriff. 2022. Status report on the third round of the NIST post-quantum cryptography standardization process. , NIST IR 8413 pages. https://doi.org/10.6028/NIST.IR.8413

[30] NIST. 2016. Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[31] Lucas Pandolfo Perin, Gustavo Zambonin, Douglas Marcelino Beppler Martins, Ricardo Custódio, and Jean Everson Martina. 2018. Tuning the Winternitz hash-based digital signature scheme. In *2018 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, Natal, Brazil, 00537–00542. https://doi.org/10.1109/ISCC.2018.8538642

[32] Ray Perlner, John Kelsey, and David Cooper. 2022. Breaking Category Five SPHINCS+ with SHA-256. https://eprint.iacr.org/2022/1061

[33] Peter W. Shor. 1997. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. , 1484–1509 pages. https://doi.org/10.1137/S0097539795293172 arXiv:quant-ph/9508027

[34] Jan Philipp Thoma and Tim Guneysu. 2021. A Configurable Hardware Implementation of XMSS. https://eprint.iacr.org/2021/352

[35] Wen Wang, Bernhard Jungk, Julian Wälde, Shuwen Deng, Naina Gupta, Jakub Szefer, and Ruben Niederhagen. 2020. XMSS and Embedded Systems. In *Selected Areas in Cryptography – SAC 2019*, Kenneth G. Paterson and Douglas Stebila (Eds.). Springer International Publishing, Cham, 523–550.