Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

Post-Quantum FIDO2 Security Keys using Hash-Based Signatures

Masterarbeit

zur Erlangung des akademischen Grades
Master of Science (M. Sc.)

eingereicht von:	Quentin M. Kniep
geboren am:	[redacted]
geboren in:	[redacted]
Gutachter/innen:	Prof. Dr. Jens-Peter Redlich
	Prof. Dr. Ernst-Günter Giessmann

eingereicht am: .... verteidigt am: ....

Abstract

Security keys used in two-factor authentication are based on digital signatures. These signatures are currently based on elliptic curves or other problems which will no longer be hard to solve on quantum computers. This thesis looks at Google's OpenSK specifically, assessing steps for making it post-quantum secure. Sequentially-updatable Merkle trees and Merkle tree forests are presented as theoretical generalizations of Merkle trees. A novel hash-based few-time signature scheme is then proposed, which is based on sequentially-updatable Merkle trees and a variant of W-OTS$^+$. An implementation of this scheme is analyzed and compared to lattice-based and classical cryptography regarding its feasibility for use in FIDO authenticators and similar applications.

Contents

Acronyms

2FA two-factor authentication

CSPRNG cryptographically-secure pseudorandom number generator

CTAP Client to Authenticator Protocol

DLP discrete logarithm problem

DoS denial-of-service

DSA Digital Signature Algorithm

ECDSA Elliptic-Curve DSA

FTS few-time signature

KDF key derivation function

KEM key encapsulation mechanism

MAC message authentication code

MFA multi-factor authentication

MITM man-in-the-middle

MSS Merkle signature scheme

NIST National Institute of Standards and Technology

OTP one-time password

OTS one-time signature

PQ post-quantum

PQC post-quantum cryptography

PRF pseudo-random function

PSK pre-shared key

RP relying party

SoC system on a chip

SVP shortest vector problem

W3C World Wide Web Consortium

XMSS eXtended Merkle Signature Scheme

1. Introduction

Security keys provide a great alternative to — or extension of — password-based authentication. They are usually hardware tokens (for example USB sticks or NFC smart cards) that store private keys of asymmetric cryptography. In asymmetric cryptography both communicating parties do not know the same secret, instead the authenticating party has sole ownership of a private key and the verifying party only needs a public key, which does not need to be kept secret. Advantages of security keys over password-based authentication are:

- high entropy (keys are generated from secure random numbers by the hardware, not thought of and remembered by the user)
- phishing resistance (in asymmetric cryptography secrets are never transmitted to the server)
- replay-attack resistance (challenge-response mechanisms are used where the same authentication message can only be used for one session)

FIDO is the most widely used standard for security-key based authentication. It is an open standard developed by the industry consortium FIDO Alliance, which includes companies such as Google, Microsoft, and Mozilla. The FIDO Alliance is also supported by government organizations, such as the American National Institute of Standards and Technology (NIST) and the German Federal Office for Information Security (BSI). Later in section 2.4, FIDO's relevant protocols are presented in more detail.

It is long known though [41], that quantum computers will be able to solve the discrete logarithm problem (DLP) and the prime factorization problem, which are number theoretical problems for which no efficient algorithm for classical (non-quantum) computers exists. The hardness of these problems is at the heart of current asymmetric cryptography. Therefore, quantum computers are able to break the security of RSA, Digital Signature Algorithm (DSA) and Elliptic-Curve DSA (ECDSA), the asymmetric cryptosystems used in essentially all cryptographic applications today, including FIDO security keys. Across all systems and applications a lot of work needs to be done replacing these asymmetric cryptosystems with quantum-secure alternatives.

1.1. Motivation

FIDO authentication is currently based on asymmetric cryptographic primitives, like ECDSA. To provide secure authentication even in a future where adversaries with large quantum computers exist, the FIDO protocols need to be extended. Especially, new cryptographic primitives need to be employed, for example hash- or lattice-based signature schemes could replace current signatures.

Furthermore, all currently available PQ asymmetric cryptography is much more resource intensive than their classical counterparts. Different schemes also give very different tradeoffs between, for example, signature size and computation time needed. Thus, there is not one straightforward way of making current protocols PQ secure. Especially, there is no single primitive that is optimal in every way and can always be substituted in for the current classical primitive. The different tradeoffs have to be weighed carefully, possibly even novel ways of combining cryptographic primitives have to be found for optimally crafted PQ security protocols.

1.2. Contributions

The goal of this thesis is to analyze the possible ways of integrating PQ secure cryptographic primitives into the FIDO protocols, compare the different primitives in their relevant performance characteristics. For this OpenSK, an open-source implementation of a FIDO security key developed at Google, and its supported hardware are used as a reference and benchmarking platform.

Specifically, the focus is on how hash-based few-time signatures can be used efficiently. These cryptosystems have been disregarded so far because of the usually very large public key and signature sizes. In chapter 4 new ways of using hash-based signature schemes for the specific use-cases of FIDO authentication are presented. Finally, in chapter 5, we evaluate the proposed solutions regarding their security properties, and communication- and computation-efficiency. We compare them to NIST standardization candidates for PQ cryptography and the current classical primitives, thus giving an indication towards the viability of PQ secure user authentication on current embedded hardware.

2. Background

2.1. Quantum Algorithms

2.1.1. Shor

Shor presented efficient quantum algorithms [41] for integer factorization and the (elliptic curve) discrete logarithm. The running times are in $\mathcal{O}((\log n)^3)$, where n is the number to be factored. Thus, they run in time polynomial in the input length, and are therefore almost exponentially faster than the fastest algorithms available on classical computers [36, 1]. Consequently, an adversary with a large quantum computer can break RSA, DSA, and ECDSA, by for example cryptanalyzing the public key, reconstructing the private key.

2.1.2. Grover

Grover's algorithm [17] can be used to speed up search for an input to a black-box function for a specific output by a quadratic factor. That is, a quantum computer can find such an input in an average of $\mathcal{O}(\sqrt{n})$ steps, whereas any classical algorithm will always need on average $\mathcal{O}(n)$ time. This is because the classical computer would on average have to check half the possible inputs. It can be used to speed up hash preimage/collision search and symmetric cryptography key recovery attacks. Because the speed-up is only quadratic, searching a search space of size 2^n still takes $\mathcal{O}(2^{n/2})$ time. For hash-collision search the theoretical speed-up is even less, improving from $\mathcal{O}(2^{n/2})$ (using classical birthday attacks) to $\mathcal{O}(2^{n/3})$. So, symmetric cryptography and hash functions are not broken by quantum computers. Doubling hash function output lengths and key lengths of symmetric encryption is enough to eliminate any theoretical advantage. Also, Grover's algorithm is shown to be asymptotically optimal on quantum computers [44]. So, further results improving upon Grover's work are not to be expected.

There are also arguments [13, 2] that Grover's algorithm, in practice, has even less of an effect. Specifically, this means 192-bit primitives actually provide around 128-bit PQ security against preimage attacks — not just 96 bits. These arguments are based on the limited parallelization possibilities of Grover's algorithm, which scales only by $\sqrt{n}$ when using n parallel instances [44]. This is relevant because in most attack models the attacker runs their attack highly parallelized. In their call for proposals [32], the NIST also asked researchers to focus on the lower security levels, suggesting they estimate these to be secure for the near future. This mostly means levels I and III, which are equivalent to 128-bit preimage search and 192-bit preimage search respectively. Throughout this thesis we will use these as our PQ security levels.

2.2. Preliminaries

Here we establish some common definitions from cryptography, which are needed for the rest of the thesis.

2.2.1. Cryptographic Hash Functions

A hash function is a deterministic function $h : \{0,1\}^* \to \{0,1\}^n$, i.e. it maps arbitrary length bit strings to bit strings of fixed length n. To be considered a (strong) cryptographic hash function it needs to fulfill these security properties:

- **Preimage resistance:** Given $h(m)$ it is hard to find any m' (possibly $m' = m$), such that $h(m) = h(m')$.
- **Second preimage resistance:** Given m_1, and thus $h(m_1)$, it is hard to find an $m_2 \neq m_1$, such that $h(m_1) = h(m_2)$.
- **Collision resistance:** It is hard to find any m_1, m_2 (with $m_1 \neq m_2$), such that $h(m_1) = h(m_2)$. This is only required for *strong* cryptographic hash functions.

Cryptographic hash functions usually provide n-bit security against preimage attacks, and $\frac{n}{2}$-bit security against collision attacks, both of which is optimal. Another property that is often wanted of (cryptographic) hash functions is that they behave like pseudorandom functions when concatenating the inputs to a key. For this we first define keyed cryptographic hash function families:

- **Keyed cryptographic hash function family:** A set of functions F, where each function $f_k \in F$ for key k (usually $k \in \{0,1\}^n$, where n is the hash function output length) is a cryptographic hash function.

- **Pseudorandom function family:** A keyed function family F where if we pick a function $f_k \in F$ at random based on its key k it is impossible to distinguish any outputs of f_k from random outputs (regardless of whether inputs are chosen randomly or predictably).

2.2.2. Digital Signature Schemes

A digital signature scheme provides three algorithms:

- **genkp(seed):** Takes an n-bit random seed (usually from a cryptographically-secure pseudorandom number generator (CSPRNG)) and returns a keypair, i.e. a private key (with an entropy of n bits) and a corresponding public key.

- **sign(kp, msg):** Generates a signature for message *msg* using keypair *kp*.

- **verify(pk, msg, sig):** Verifies whether *sig* is a valid signature for the message *msg* under the public key *pk*, i.e. was created using the corresponding private key. Returns true/false accordingly.

Security of digital signature schemes is formalized in the following three notions [16]. For all, assume the public key *pk*, some messages $(m_0, m_1, \ldots, m_k)$ and corresponding valid (under *pk*) signatures $(s_0, s_1, \ldots, s_k)$ to be public information. This list is in weak to strong order (where each stronger notion implies all weaker ones):

- **Total break resistance:** Using only the public information, it is hard to find the secret key *sk* that corresponds to *pk*.

- **Universal forgery resistance:** Given a new message m_{k+1} (generated by the owner of the secret key) and using only the public information, it is hard to find a corresponding signature s_{k+1} which is valid under *pk*.

- **Selective forgery resistance:** Given a message m_{k+1} (generated by the adversary independent on the messages and signatures they learned) and using only the public information, it is hard to find a corresponding signature s_k+1 which is valid under *pk*.

- **Existential forgery resistance:** Using only the public information, it is hard to find any message-signature pair (m_{k+1}, s_{k+1}) which is valid under *pk*.

2.2.3. One-/Few-Time Signatures

A digital signature scheme is called an one-time signature (OTS) scheme if we can only sign one message without (partially) compromising security. Let n be the security level of the one-time signature (OTS) scheme. If a second signature is ever created under the same keypair, the security of the scheme may become less than n bits (possibly rendering all signatures untrustworthy).

A digital signature scheme is called an few-time signature (FTS) scheme if for some $k \in \mathbb{N}$ we can only sign k message without (partially) compromising security. Let n be the security level of the few-time signature (FTS) scheme. As long as k or fewer signatures have been created under a given keypair, the security of the scheme is (at least) n bits. As soon as a total of $k + 1$ or more signatures have been created under the same keypair, the security of the scheme may become less than n bits (possibly rendering all signatures untrustworthy). Usually k is a parameter which can be increased at the cost of more storage-, communication-, and computation-overhead.

2.3. Post-Quantum Cryptography

2.3.1. Lattice-based Cryptography

These cryptosystems are based on mathematical properties of lattices and specific problems regarding these. The underlying computational problems, such as the shortest vector problem (SVP) are currently believed to be intractable to solve, even on quantum computers. Falcon [14] is an example of lattice-based cryptography; it is a signature scheme based on NTRU lattices [18]. Another notable example is SWIFFT [27], a cryptographic hash function based on lattices. There is a security reduction of its collision resistance to worst-case ideal lattice problems, which is unusual for cryptographic hash functions. Also unusual is its homomorphic property ($h(a + b) = h(a) + h(b)$), which can be useful for certain applications (as will be seen in subsection 3.3.1) but necessarily makes it not a pseudorandom function. One more example is the first ever fully homomorphic encryption scheme [15], which is also made possible by lattice-based cryptography.

The NIST post-quantum cryptography (PQC) standardization process has so far shown that lattice-based schemes offer a good middle ground. Under the finalists, two of the three signature schemes and three of the four key encapsulation mechanisms

(KEMs) are lattice-based [1]. Other schemes [4, 29] go very far to either extreme of time-space tradeoffs. Whereas those based on linear codes are fast but have key sizes in the megabytes, those based on supersingular isogenies over elliptic curves are up to three orders of magnitude slower but have the smallest ciphertexts [38].

2.3.2. One-time Passwords

Lamport [26] first proposed the idea of using a chain of hash values as single-use passwords, so-called one-time passwords (OTPs). The idea is that of a finite chain $(x_0, x_1, \ldots, x_n)$, where x_0 is a secret seed value and $x_i = h(x_{i-1})$ for all $i > 0$, for some cryptographic hash function h. Initially, after generating the hash chain, the verifying party is given x_n. After that the authenticating party can authenticate themselves by sending x_{n-1}.

One main shortcoming of this scheme is that it needs regular re-registration of the password because the hash chain only has a finite number of elements. Recently, an improved hash-chain OTP was proposed [34] which solves exactly this problem while keeping the other advantages of the Lamport OTP. As opposed to other schemes which generate unbounded numbers of OTPs this does not require any shared secrets. It works by using many short hash chains, which are linked in a specific pattern. Each hash chain in theory has its own seed, though the seeds can all be generated from a single seed via a CSPRNG. Where this infinite OTP scheme fails though is when transmission failures occur. It can not recover from more than one failure, whereas Lamport's scheme can always recover from $r - 1$ failures, where r is the number of unused OTPs remaining. In contrast to signatures OTPs are not bound to a specific message. Especially, they can not be used to prove to a third party that a specific message was approved.

2.3.3. Hash-based Signature Schemes

Hash-based signature schemes build upon the seminal papers of Lamport [25] and Merkle [31]. Lamport first introduced the idea of using hash values as public keys and selectively publishing their inputs to sign messages. The original scheme requires one collision resistant hash value per bit of the message to be signed. Assuming 256 bit long hashes (to achieve 128-bit security against collisions), this gives signature sizes of

[1] https://csrc.nist.gov/Projects/post-quantum-cryptography/round-3-submissions (Accessed Oct 17 2021)

around 8 KiB ($= 256 \cdot 256$ *bits*) because the signed message should also be a collision resistant hash. This is a one-time signature (OTS) scheme.

Hash-based signatures are especially interesting as (depending on their construction) they may require only one-way functions, which is the minimal assumption you need for digital signature schemes to exist at all [37]. The next scheme we look at, W-OTS$^+$, actually achieves this. Also, because hash functions are already central to much of practical cryptography, it might be easier to trust signature schemes built from them than other PQ secure schemes, which often rely on new assumptions which are not yet applied in any practically used schemes.

W-OTS/W-OTS$^+$

Nowadays there are much more efficient hash-based OTS schemes exist, most prominently the Winternitz improvement [30]. It extends Lamport's idea by introducing a space-time tradeoff parameter w. With $w = 2$ it is essentially equivalent to Lamport's scheme, whereas with $w > 2$ a single hash value in the signature maps to $\log_2 w$ bits (instead of just one bit) of the message. Thereby the signature size decreases linearly in $\log_2 w$, the number of hash function evaluations needed for key generation, signing, and validation on the other hand increase (almost) linearly in w (and thus exponentially in $\log_2 w$). Central to the scheme are hash chains, such as seen in Figure 2.1. In W-OTS the chaining function is just plain applying of a hash function h, i.e. $e_{i,j} = h(e_{i,j-1})$ for all $i > 0$. Further improvements have been made to the Winternitz OTS scheme, especially the W-OTS$^+$ variant [20], which is shown to be secure when instantiated with any cryptographic one-way function (without requiring collision resistance). This enables signature sizes to be half the length they need to be for plain W-OTS for the same security parameter. Also, whereas security for W-OTS decreases linearly in w, for W-OTS$^+$ it only decreases logarithmically. W-OTS$^+$ achieves all this by replacing plain calls to the hash function with keyed calls and using random bitmasks on the input, i.e. $e_{i,j} = h_{key_{i,j-1}}(e_{i,j-1} \oplus bitmask_{i,j-1})$ for all $i > 0$.

In the following the exact variant of W-OTS$^+$ used in this thesis is explained in detail. Let h_k be functions from keyed cryptographic one-way function family, w be the Winternitz parameter, n be the security parameter, and f_k be pseudorandom functions. The output size of h_k needs to be at least n bits (if it is longer it can be truncated). The number of hash chains is $l = l_1 + l_2$, with message length $l_1 = 2n/\log_2 w$ and checksum length $l_2 = \lfloor \log_w (l_1 \cdot (w - 1)) \rfloor + 1$. The secret key technically is $(s_0, s_1, \ldots, s_{l-1})$, instead we can generate these from a single seed using a key derivation function (KDF).

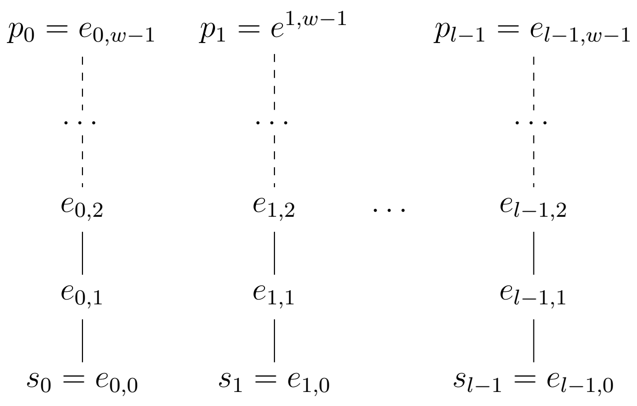

Figure 2.1.: Hash chains as used in the W-OTS one-time signature scheme.

Then, only a single value s needs to be stored by defining $s_i = f_s(i)$. The public key technically is the tuple of hash-chain end values $(p_0, p_1, \dots, p_{l-1})$. This can be reduced by hashing because the values p_i can later be calculated from the signature. So, the published public key is just the single hash value $pk = h(p_0, p_1, \dots, p_{l-1})$ instead of the whole tuple of l hash values.

To sign a message, it is hashed and the hash is converted into base-w numbers $(m_0, \dots, m_{l_1-1})$. Additionally a checksum is calculated as chk $= \sum_{i=0}^{l_1-1} w - 1 - m_i$, also converted to a base-w representation $(m_{l_1}, m_{l_1+1}, \dots, m_{l-1})$, and then appended to the message. The signature for the final message $(m_0, m_1, \dots, m_{l-1})$ is then $(e_{0,m_0}, e_{1,m_1}, \dots, e_{l-1,m_{l-1}})$. Signing and verifying thus use an expected number of $\frac{wl}{2}$ hash function evaluations, whereas public key generation always requires wl evaluations. Signature size is $wl + 1$ hash values ($(wl + 1) \cdot n$ bits), the one additional hash value being for the public seed from which the keys for and bitmasks for chaining are derived. Effective secret and public key sizes are, as argued before, just a single hash value (n bits).

Without the additional checksum it would be possible to calculate signatures for different messages from a given signed message. Given a message $(m_0, m_1, \dots, m_{l-1})$ and a corresponding valid signature $(c_0, c_1, \dots, c_{l-1})$. Assuming for some i the message has $m_i < w - 1$, an attacker could construct the message $(m_0, \dots, m_{i-1}, m_i + 1, m_{i+1}, \dots, m_{l-1})$ and the corresponding valid signature $(c_0, \dots, c_{i-1}, h(c_i), c_{i+1}, \dots, c_{l-1})$.

One interesting property of W-OTS (and W-OTS$^+$) is that the public key can be derived from the signature. Usually the actual public key would not be published, instead a compact hash value would be published, defined for example as: $h_1 = h(pk_0 \parallel pk_1 \parallel \dots \parallel pk_{l-1})$.

Algorithm 1 W-OTS$^+(w, n, l, h, f)$

```
1:  procedure KeyGen($seed$)
2:      $seed_{\text{public}} \leftarrow f_{seed}(l)$
3:      $(s_0, \dots, s_{l-1}) \leftarrow (f_{seed}(0), \dots, f_{seed}(l-1))$
4:      $(p_0, \dots, p_{l-1}) \leftarrow (\text{Chain}(s_0, w-1, 0, 0), \dots, \text{Chain}(s_{l-1}, w-1, l-1, 0))$
5:      $pk \leftarrow h(p_0 \parallel p_1 \parallel \dots \parallel p_{l-1})$
6:      return $(seed, pk)$
7:  end procedure

8:  procedure Sign($msg$, $seed$)
9:      $(c_0, \dots, c_{l-1}) \leftarrow \text{CyclesForMsg}(msg)$
10:     $(s_0, \dots, s_{l-1}) \leftarrow (f_{seed}(0), \dots, f_{seed}(l-1))$
11:     $(\sigma_0, \dots, \sigma_{l-1}) \leftarrow (\text{Chain}(s_0, c_0, 0, 0), \dots, \text{Chain}(s_{l-1}, c_{l-1}, l-1, 0))$
12:     return $(o_0, o_1, \dots, o_{l-1})$
13: end procedure

14: procedure Verify($msg$, $sig = (\sigma_0, \dots, \sigma_{l-1})$, $pk$)
15:     $(c_0, \dots, c_{l-1}) \leftarrow \text{CyclesForMsg}(msg)$
16:     $(p_0, \dots, p_{l-1}) \leftarrow (\text{Chain}(\sigma_0, w-1-c_0, 0, c_0), \dots, \text{Chain}(\sigma_{l-1}, w-1-c_{l-1}, l-1, c_{l-1}))$
17:     $pk_{\text{Calc}} \leftarrow h(p_0 \parallel \dots \parallel p_{l-1})$
18:     return $pk = pk_{\text{Calc}}$
19: end procedure

20: procedure CyclesForMsg($msg$)
21:     $hash \leftarrow h(pk \parallel h(msg))$
22:     $(c_0, \dots, c_{l_1-1}) \leftarrow \text{ToBase}(hash, w)$
23:     $chk \leftarrow 0$
24:     for all $x \in \{c_0, \dots, c_{l_1-1}\}$ do
25:         $chk \leftarrow chk + (w-1-x)$
26:     end for
27:     $(c_{l_1}, \dots, c_{l-1}) \leftarrow \text{ToBase}(chk, w)$
28:     return $(c_0, \dots, c_{l-1})$
29: end procedure

30: procedure Chain($input$, $cycles$, $chain$, $i$)
31:     if $cycles = 0$ then
32:         return $input$
33:     else
34:         $key \leftarrow f_{seed_{\text{public}}}(\text{"}key\text{"} \parallel chain \parallel i)$
35:         $bitmask \leftarrow f_{seed_{\text{public}}}(\text{"}bm\text{"} \parallel chain \parallel i)$
36:         return $\text{Chain}(h_{key}(input \oplus bitmask), cycles-1, chain, i+1)$
37:     end if
38: end procedure
```

To further reduce the signature size, during signing h_1 can be included in the message hash, that is to sign message m the input to W-OTS$^+$ should be $h(m \parallel h_1)$ instead of simply $h(m)$ [35]. The authenticator would then publish $h_2 = h(h_1)$ instead of h_1, to keep h_1 hidden until the signature is created. By doing this the hash function h also does not need to be collision-resistant. Therefore, the input to W-OTS$^+$ can be again half its original length, resulting in almost halving the total signature length. Publishing the original public key hash, before the additional round of hashing is then necessary though, increasing the signature size by one hash value. So signatures are now $wl+2$ hash values ($n(wl+2)$ bits) long, still with $l = l_1 + l_2$ but now $l_1 = n \cdot \log_2 w$. This adaptation to W-OTS$^+$ is also used in our implementation.

MSS/XMSS

Merkle [31] showed how OTSs can be combined into an FTS scheme, by building a binary tree of hash values. In general a Merkle tree is a binary tree of hash values, where each node is the hash over the concatenation of its two children. When used as an FTS scheme this is called Merkle signature scheme (MSS). The leaf nodes are then OTS public keys (or hash values thereof, in which case the actual public keys need to be provided together with the signature). The root of the tree serves as the FTS public key, and serves as a pre-commitment to all the OTS public keys. To sign a message under this scheme the signer first signs it using the OTS keypair from one of the leaves. Additionally, they need to provide the authentication path, which is the sequence of sibling nodes when traversing the path from leaf to root node. A verifier can then check the OTS signature and calculate all hashes along the authentication path, finally comparing the calculated root hash with the root value they have stored as FTS public key.

This is a rather efficient scheme, in terms of communication overhead: Signature size overhead over the OTS scheme is only logarithmic in the number of signatures possible with the same public key (i.e. the number of leaf nodes). On the other hand, the public key generation can become rather expensive, as the whole tree needs to be calculated (including one OTS public key for each leaf node). Public and secret keys can still be just a single value of the same length as a hash output. This is because the OTS seeds can all be generated from the same FTS seed via a KDF, analogous to how we then generate all the hash chain start values from the single OTS seed.

A variant of this scheme called eXtended Merkle Signature Scheme (XMSS) [8] (but instantiated with W-OTS$^+$, as in [20]) is standardized in RFC 8391 [19].

2.4. FIDO

FIDO is a collection of standards defining algorithms and protocols (or ceremonies) for user authentication based on so-called FIDO authenticators. These authenticators are recommended to be so-called roaming authenticators, i.e. external hardware that is not part of the platform running the client software, for example a USB security key (NFC and Bluetooth Low Energy are other common connectivity methods). It can be used as a single factor to provide password-less authentication or in two-factor authentication (2FA) in addition to a password. First of all, we define some specific terms from the FIDO standard that will be used in this section:

- **Relying Party:** The platform running the WebAuthn and application servers is called the relying party (RP).
- **Client:** Software that communicates with WebAuthn server and the authenticator. For authentication against web servers, this is usually done directly by the browser.
- **Credential:** One keypair that is registered with a specific RP and is used for creating assertions. A special case are *discoverable* credentials (also known as *resident keys*), which are stored persistently on the authenticator or generated deterministically from the RP ID. Usually stored encrypted on the server and thus not *discoverable*.
- **Assertion:** A confirmation created by an authenticator specifying that a certain action has been authenticated, with a specific credential and possibly with a user presence check or further verification (e.g. PIN or fingerprint).
- **Attestation:** The process in which the authenticator provides a certificate and signature, indicating that it is genuine and provides specific capabilities.
- **User presence:** A simple check on the authenticator which serves to verify the user is using the device (e.g. by button press), to prevent fully-automated access to the authenticator.
- **User verification:** A further check performed directly on the authenticator to verify the user to the authenticator. This can be a user-specified PIN code or biometric check (e.g. fingerprint).

2.4.1. WebAuthn

WebAuthn is an authentication API designed by the World Wide Web Consortium (W3C). The WebAuthn/FIDO2 protocol is the challenge-response authentication protocol that is part of the FIDO standards. It is used for authenticator-based authentication on the web and serves as a standard, which many independently developed implementations adhere to. Web application developers can use a WebAuthn server library to support authenticators as password-less or second-factor authentication for their users. In the case of discoverable credentials even username-less login is possible. For this the client *discovers* the credential on the authenticator based on the RP ID. Then, the server can look up the correct account based on the public key (serving as the user's identifier).

2.4.2. CTAP

Client to Authenticator Protocol (CTAP) is the protocol used for communications between the FIDO authenticator and the client platform. CTAP 2.0/2.1 are the versions of this protocol which are part of the FIDO2 standard currently under development. They replace their predecessor CTAP 1.0, also known as U2F.

Central to the CTAP specification are two procedures, one for registering a new credential (called `authenticatorMakeCredential`) and one for authenticating with an already registered credential (called `authenticatorGetAssertion`). These are called *ceremonies* — not *protocols* — in FIDO because of the explicit involvement of the human user (for example via user presence checks).

The credential registration ceremony (see Figure 2.2) is used to register a credential with a user account. Usually it will be performed only once to add the authenticator to the user's account, but can also be done again, for example to add another authenticator. During this ceremony the credential's public key is signed with an attestation key, which is static and independent of the credential. The attestation key lets the RP identify the authenticator's manufacturer and model, they may then use this information to assess the risk associated with this type of authenticator, for example no longer accepting authenticators from a compromised series.

The authentication ceremony (see Figure 2.3) is a challenge-response authentication protocol. Depending on the application it may be performed on every login, only when signing in to a new device, or only when specific actions — such as a password reset — should be performed. The RP sends a random challenge which serves to provide freshness and thus prevent replay attacks. This challenge (together with some data from

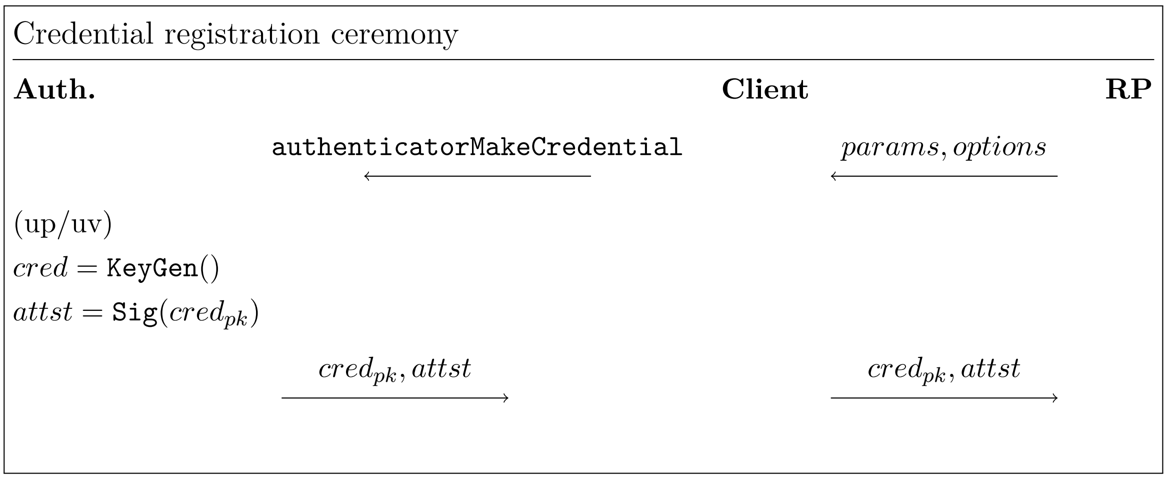
Credential registration ceremony
Auth.
Client
RP
authenticatorMakeCredential
$params, options$
(up/uv)
$cred = \texttt{KeyGen}()$
$attst = \texttt{Sig}(cred_{pk})$
$cred_{pk}, attst$
$cred_{pk}, attst$

Figure 2.2.: Simplified sequence diagram of the credential registration ceremony in WebAuthn/FIDO2. The abbreviations up and uv mean *user presence* and *user verification* respectively.

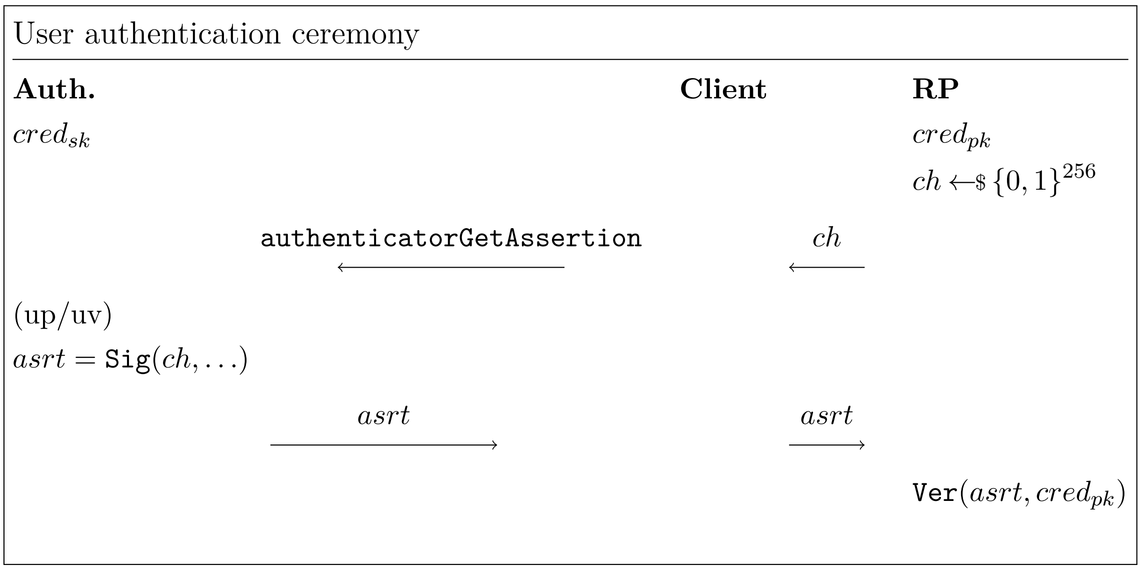
User authentication ceremony
Auth.
Client
RP
$cred_{sk}$
$cred_{pk}$
$ch \leftarrow\$ \{0,1\}^{256}$
authenticatorGetAssertion
ch
(up/uv)
$asrt = \texttt{Sig}(ch, \ldots)$
$asrt$
$asrt$
$\texttt{Ver}(asrt, cred_{pk})$

Figure 2.3.: Simplified sequence diagram of the user authentication ceremony in WebAuthn/FIDO2. The abbreviations up and uv mean *user presence* and *user verification* respectively.

the authenticator) is signed on the authenticator using the private key that is registered with this RP. This way many of the usual attack vectors against passwords do not apply. For example, phishing by a third party website is no longer possible because the authenticator only responds with the keypair registered for the RP. Though, it is essential that the authenticator gets authentic data about the RPs identity from the client, otherwise phishing is possible again. Assume the client is compromised and can provide the authenticator with wrong RP IDs. If the malware and the phishing site are operated by the same adversary, they can cooperate to trick the authenticator to issue a signature.

CTAP requires the authenticators to at least support a global signature counter, and preferably one for each credential. Allowing the RP, who stores this counter, to detect and prevent attackers from logging in with cloned authenticators.

In chapter 4 the focus will be on replacing the credential (keypair) and assertion (signature). Conversely, we will not focus on attestation, user PIN protocol, and WebAuthn encryption and authentication (TLS). These parts are less interesting, in the sense that they are more well studied and less FIDO specific. Attestations for example can be implemented with any public key infrastructure solution, such as is needed for TLS anyways. Also, attestations are only sent once on credential registration, not on every user authentication. Thus, the resource constraints are less harsh here as well. On the other hand the user PIN protocol is (potentially) executed on every user authentication, but only between client and authenticator, without involving the RP. It is largely based on symmetric cryptography, though, only a single key agreement is needed, which can be done with any PQ KEM.

3. Related Work

There is currently no published work directly analyzing the FIDO protocols regarding their PQ security. Furthermore, there is no published work so far making propositions on how to adapt the protocols to guarantee some level of PQ security. Thus, the work presented in this thesis is the first analysis, that we are aware of, directly regarding FIDO and PQ cryptography. On the other hand, a lot of work already exists analyzing the security of the FIDO protocols and PQ cryptography, including hash-based signature schemes (even in the setting of embedded systems), separately.

3.1. Formal Analysis of FIDO

There are published works formalizing the security properties of WebAuthn and FIDO protocols, including CTAP1/UAF and CTAP2, even providing (tool-assisted) formal proofs [5, 21, 12]. These are important as they are further support for the soundness of the protocols. On the other hand, they assume the cryptographic primitives to be secure, or at least the underlying number theoretic problems to be hard. That is, they do not look at quantum-capable adversaries. Therefore, they also don't look at how these primitives need to be replaced and what specific candidates there are.

3.2. PQ Cryptography on Embedded Devices

Also a lot of literature already exists [38, 28, 3, 43, 10, 42] on the general question of PQ cryptographic primitives' performance on mobile and embedded devices. Topics already covered in the literature include pre-computation [3], hardware-software co-design [43], and relative viability of different types of primitives [42, 3, 38]. Some [28, 10] also came to the conclusion that hash-based signature schemes are viable on embedded systems. What is new in this thesis is the focus on FIDO as a specific use case and the analysis of a novel more efficient hash-based FTS scheme for this use case.

3.3. Efficient Hash-based Signatures

3.3.1. K2SN-MSS (SWIFFT)

K2SN-MSS [24, 22] is a special case, as it is a hash-based FTS scheme but SWIFFT, the hash function it is based on, uses lattice-based cryptography internally. The interesting property of SWIFFT that is needed here is its homomorphic property, i.e. if we have inputs x_1, x_2 for which x_1+x_2 is also a valid input, it satisfies $h(x_1+x_2) = h(x_1)+h(x_2)$. This property enables more efficient constructions than the Winternitz scheme. It is also efficiently implemented in software, taking similar times for key generation, signing, and verification as W-OTS$^+$ implementations.

Unfortunately, the hash outputs of SWIFFT are much larger than the output sizes of the SHA family, which are optimal for their respective security parameters. For example for security parameter 112 the hash output is at least 512 bits long [27]. This results in OTS public keys of around 15 KB because the public key can also not be calculated from the signature as in W-OTS$^+$, and thus not reduced by hashing it.

3.3.2. SDS-OTS

Very recently, Shahid et al. [39] [40] proposed a new OTS scheme, which aims to improve upon W-OTS$^+$ regarding signature size and computation time.

Whereas W-OTS$^+$ builds one hash chain per w-bit symbol of the message hash, in SDS-OTS the parameter w is used in practically the inverse way. They build w hash chains, where the length of each chain is equal to the number of w-bit symbols in the message hash.

Their claimed performance of the signature scheme would make it a very attractive alternative to W-OTS$^+$. If the parameters they provide in the paper would indeed give a PQ security level of 128 bits, SDS-OTS would be about two times faster at signing and key generation and also have slightly smaller signatures. Unfortunately, the scheme as they describe it does not seem to give the claimed security level.

The signature is determined entirely by the sums of hex digit indices, these are 16 numbers in the range $0, 1, \ldots, 127$. Therefore, there are only a total of $128^{16} = 2^{112}$ different signatures. By the pigeon-hole principle an attacker can find two messages with the same signature, and thereby an existential forgery, by exhaustive search of $2^{112} + 1$ messages. Thus, SDS-OTS does not even achieve a *classical* security level of 128 bits. The security proof, which claims existential forgery of SDS-OTS can be reduced to onewayness of the underlying hash function has to be flawed then. Flawed assumptions in the proof include: uniform randomness of the forged messages and even allowing the adversary to calculate the inverse of the one-way function at one point.

We can make three adaptations to the SDS-OTS scheme to at least achieve 128 bit classical security, see Appendix A. One of the necessary changes makes key generation, signing, and verification twice as computationally expensive. By then it is no longer any faster than W-OTS$^+$ with $w = n = m = 256$, which even achieves the 256 bit classical security level (and PQ level V). In conclusion, SDS-OTS does not need to be considered further because it can be seen as inferior to the state-of-the-art W-OTS$^+$.

3.3.3. SPHINCS$^+$

SPHINCS$^+$ [6, 7] was one of the candidates in round 2 of the NIST's PQC competition [33], though in the ongoing round 3 it is only listed under "Alternate Candidates", not under "Round 3 Finalists" [1]. It combines FTSs into a general signature scheme which can sign a practically unlimited number of messages, similarly to how FTS schemes combine many instances of an OTS scheme.

We do not consider SPHINCS$^+$ further as it gives very large signatures and public keys, of at least 8 KiB each. The other NIST candidate schemes offer better tradeoffs between signature size and computational effort [38, 42]. The only upside of general hash-based signature schemes at the moment is that hash functions are already well studied and widely used, compared to lattice-based cryptography for example. In chapter 4 it will be shown how much can be gained in terms of performance, when using FTSs directly and dropping the aim of signing (practically) unlimited numbers of messages.

[1] https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

4. Methodology

4.1. Hardware and Software

4.1.1. OpenSK

OpenSK [1] is an open-source implementation of a FIDO authenticator developed by Google. It currently implements the CTAP 2.0 protocol, with version 2.1 being worked on. OpenSK is written in Rust and based on the embedded operating system Tock [2], also written in Rust. We used OpenSK and the nRF52840 board (explained in subsection 4.1.2) it is supposed to be used with as a reference FIDO authenticator. Of the actual OpenSK implementation only the cryptography benchmark was used and extended to provide easy benchmarking on the board, and to perform direct performance comparisons with OpenSK's cryptographic primitives.

4.1.2. nRF52840

Nordic Semiconductor's nRF52840 is a system on a chip (SoC) and the only hardware currently officially supported by the OpenSK project. It features a single 32-bit ARM processor, clocking at 64 MHz, 256 KiB RAM and 1 MiB flash memory. It supports all usual FIDO connectivity methods: USB, NFC, and Bluetooth Low Energy. As opposed to more advanced FIDO authenticator hardware it has neither a display nor input capabilities for a user PIN. This chip has an embedded ARM TrustZone CryptoCell 310 security subsystem, which offers hardware accelerated cryptographic primitives, including AES and SHA-256. Unfortunately, it is currently not directly accessible from Tock, though there is a work-in-progress development branch for this [3] on OpenSK developer Picod's fork of Tock. For benchmarking, we more specifically used the development board nRF52840-DK.

[1] https://github.com/google/OpenSK (Accessed Oct 17 2021)
[2] https://github.com/tock/tock (Accessed Oct 17 2021)
[3] https://github.com/jmichelp/tock/tree/cryptocell (Accessed Oct 17 2021)

4.2. Threat Model

FIDO authenticators are meant to protect against scenarios which are not considered by the usual adversary model, in which the adversary has a read/write man-in-the-middle position between the two (or more) communicating parties. Additionally, it tries to reduce the impact of certain user errors, such as skipping necessary checks, or falling victim to phishing. Malware with certain capabilities is also part of the adversary model, at the minimum an adversary under this model should be able to read any messages on the device where the FIDO client software runs This adversary is therefore at least as strong as an adversary employing a keylogger. Evidently, password-only authentication is not strong enough against this kind of adversary.

Regarding denial-of-service (DoS), we further make a limiting assumption about our attacker. We assume they are not able to disturb the communication at a specific point in the message flow, i.e. between two specific packets, with higher than 50% probability. This restriction in the attacker's capabilities will allow us to argue, that a few-time signature scheme will with near certainty never result in the user running out of signatures. Also, this assumption is quite reasonable, as the communication between FIDO/WebAuthn client and RP are usually TLS-secured channels. The attacker needs to solve the non-trivial task of identifying one specific packet in a TLS-encrypted data stream. An adversary with both, a man-in-the-middle position able to disrupt the TLS connection and some read-only malware able to eavesdrop on client-to-authenticator communication, however might be able to use information gathered from the malware to time the connection disruption. To make it harder for attackers the implementation could also introduce padding into messages, split packets, and send additional decoy messages, to more reliably achieve this bound.

This attack capability is not inherent to FTS schemes. The same type of DoS attack is possible against FIDO using ECDSA today. Though, when using FTSs we have to take extra care because we could run out of signatures permanently locking the user out of their account.

Under this model, the probability of a successful DoS attack is thus: $p_{\text{DoS}} = 0.5^n$. Here, n is the number of so far unused OTS keypairs. The expected number of attempts necessary to perform this attack successfully is thus exponential in n. Therefore, the number of unused OTS keypairs is a security parameter for security against such DoS attacks. More directly, we can control the number of total OTS keypairs to adapt this security parameter.

4.3. Changes to the FIDO Protocols

In this section the different ways of adapting and extending the WebAuthn/FIDO2 and CTAP2 protocols for PQ security are presented. First explaining possibilities that are not further considered and then depicting our proposed schemes using hash-based signatures.

4.3.1. General Hybrid Signatures

This way of making FIDO post-quantum secure is the most straightforward. In addition to the ECDSA signature, assertions would also be signed with a post-quantum secure signature scheme. Adding a second signature instead of replacing the ECDSA signature completely is preferable because the PQ cryptosystems used are based on less well studied mathematical properties than their classical counterparts. This approach is called hybrid-security and recommended by the German Federal Office for Information Security (BSI) [9]. Accordingly, during credential registration two public keys need to be sent by the authenticator. This greatly increases all transmission and computation costs. Possibly the only type of PQ signature schemes we are confident enough in to deploy without using them in hybrid with ECDSA, are the hash-based cryptosystems. For this reason, especially hash-based FTS schemes are looked at further below.

4.3.2. Key Encapsulation Mechanisms

One could also consider using KEMs instead of signatures as the cryptographic building block for adding PQ security into the protocol. This is attractive because among the NIST competition candidates the KEMs are more compact and faster than the signature schemes [38].

The naive way to do this would be to perform a key exchange only once, upon credential registration. Then authenticator and RP have a shared symmetric key and can authenticate themselves via message authentication codes (MACs). There are two problems with this approach, namely: A read-only compromise of the server allows an attacker to impersonate users, and a backup alone is enough for an attacker to do so. The former is inherent in that symmetric cryptography is used.

The latter problem can be solved by performing ephemeral PQ key exchanges for each authentication event. This would give forward secure authentication, a backup of the server would no longer be enough for an attacker because they could at best obtain an old ephemeral key. There is also another reason why the FIDO standard requires signatures instead, which will be seen below.

4.3.3. One-time Passwords

Generally, for authenticator-based user authentication OTPs are an interesting possibility. Especially regarding PQ cryptography, because OTPs are very efficient, requiring only a few hash function calls. The main goal is only to prove the user's identity to the RP (authentication). If this is all we need to achieve, this requirement is strictly weaker than that of a signature, which could also be used to prove the identity to a third party. In FIDO however the standard requires that signatures are used. One reason for this is that FIDO attestations are not only for user authentication but also let the user approve specific actions [4]. And in certain applications, for example banking, it should probably not be possible for the RP to sign these in the name of the user.

4.3.4. Hash-based One-time Signatures

Because most PQ signature schemes are very expensive when it comes to both computation and data transmission, the option of adapting FTS schemes is also explored. The hope here is, that few-time signatures can be cheaper in both ways than regular signature schemes.

Sequentially-Updatable Merkle Tree

First of all, we introduce a sequentially-updatable Merkle tree, which is then used as a building block of the schemes explained below. The tree is initialized with a secret-key seed s, its height h, and a KDF f. The internal state of the tree is then defined by two counters c and c_{new}, which are both at first initialized to 0. One such counter would need to be persisted anyways to prevent reuse of the OTS keypairs. The leaf nodes of the tree are then always the OTS public keys corresponding to the OTS secret keys: $f_s(c), f_s(c+1), \dots, f_s(c+2^h-1)$. Internal nodes and the root are calculated in the usual way using a hash function. Messages are always signed with the left-most unused keypair, that is the keypair at position c_{new}. Upon signing a new message, firstly c_{new} is incremented, then a new OTS is added based on the secret key $f_s(c_{new}+2^h-1)$, finally the new root can be calculated. Signatures then not only include the OTS

[4]One such proposal: https://fidoalliance.org/white-paper-fido-transaction-confirmation (Accessed Oct 17 2021)

		$pk_{\text{FTS,new}}$				
pk_{FTS}						
$pk_{\text{OTS},c}$	$\cdots$	$pk_{\text{OTS},c_{new}}$	$\cdots$	$pk_{\text{OTS},c+2^h-1}$	$\cdots$	$pk_{\text{OTS},c_{new}+2^h-1}$
$f_s(c)$	$\cdots$	$f_s(c_{new})$	$\cdots$	$f_s(c+2^h-1)$	$\cdots$	$f_s(c_{new}+2^h-1)$

Figure 4.1.: This shows the inner workings of a sequentially-updatable Merkle tree. The lower two rows show the leaf nodes, i.e. the OTS public keys, and the secrets they are generated from. The upper two rows show over exactly which leaves each of the two concurrently existing FTS keys is defined.

signature and authentication path, but also the new root hash (basically equivalent to an updated public key). When the verifier responds with success, that the new root hash has been persisted, the sequentially-updatable Merkle tree can be *reconciliated*. Reconciliation simply means the old part of the tree, before c_{new} is no longer needed. So, c can be set to c_{new} and all leaf nodes before c_{new} can be deleted.

Shallow-deep Merkle Tree (sd-MSS)

Next we present an unbalanced Merkle tree, which may also be useful for building similar authentication schemes based on OTSs or other FTSs. The idea of unbalanced Merkle trees was already proposed before [23]. This similar idea however was focused on streaming applications and not the highly synchronous setting of user authentication. Also, there is not much focus on signature sizes there.

We call this new unbalanced Merkle tree a shallow-deep Merkle tree (sd-Merkle tree). It has a simple structure, consisting of a single Merkle root node, combining two subtrees of (possibly) different heights. The left subtree, parameterized by its height s is called the shallow subtree. The right subtree, parameterized by its height d is called the deep subtree. In total the tree is thus parameterized by a pair (s, d) with $s \leq d$, for $s = d$ it is equivalent to a Merkle tree of height $s + 1$ (or equivalently $d + 1$).

An sd-Merkle tree can also be built from sequentially-updatable Merkle trees, as explained above. It then needs to keep track of two counters for each of the two subtrees. We call these counters $c_s, c_{s,new}$ and $c_d, c_{d,new}$ respectively. When signing a message, the shallow subtree is used whenever there are unused OTS keypairs left (i.e. $c_{s,new} < c_s + 2^h$). Once all keypairs in the shallow subtree are used up, the deep subtree is used for signing new messages.

We would usually cache the shallow subtree completely on the server. This reduces the signature size overhead in the best case from $s + 1$ to just a single hash value,

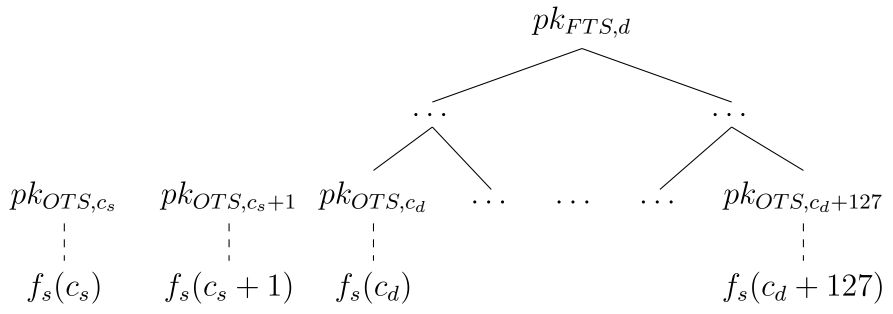

Figure 4.2.: Simplified drawing of a cached sd-Merkle tree (with $s = 1, d = 7$). Because of the server-side caching, the upper levels of the shallow (left) subtree are not needed. To fully benefit from the server-side caching we also omit the root node.

at the cost of increasing server-side storage from a single hash value to $2^s + 1$ hash values. Assuming a small s of 3 or less, this is still very acceptable, coming out to only 144 Bytes for a 128-bit hash function. To fully benefit from the server-side caching we also omit the root node, which combines shallow and deep tree.

In addition to this server-side caching of the shallow tree, the client (in our case the FIDO authenticator) could also cache (parts of) the deep tree. Here we argue that even caching the whole deep tree might be feasible. Assuming $s = 3, d = 7, n = 128$ and available storage of 1 MiB, we could store around

$$\frac{1024^2\ B}{2 * 128 * 16\ B + 2 * 8 * 16\ B} \simeq 240$$

credentials on the authenticator with full caching of both trees (including new and old part of the sequentially-updatable Merkle trees).

We can thus see that an sd-Merkle tree defined by (s, d) has the following properties:

- Total number of faults that can be tolerated: $2^s + 2^d - 1$
- Size overhead for the ith signature (with $i \leq 2^s$): i hash values
- Size overhead for all other signatures: $2^s + d + 1$ hash values
- Server-side storage needed: $2^s + 1$ hash values

The only disadvantage over a normal Merkle tree is the additional server-side storage needed for the shallow subtree. And also, a slight increase in public key generation

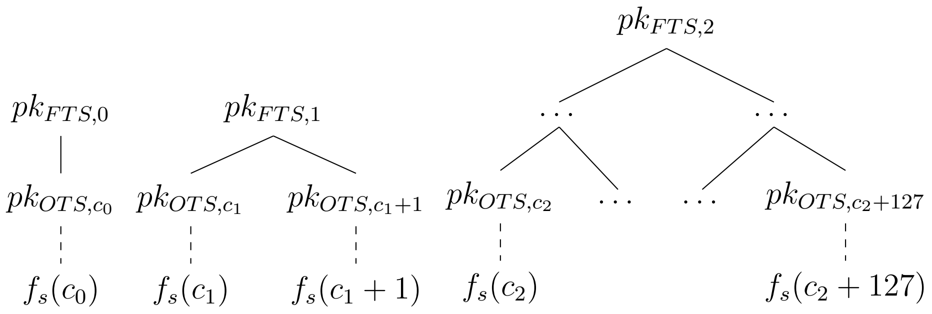

Figure 4.3.: Simplified drawing of a 0-1-7-MTF.

time, which is however directly proportional in the additional number of signatures possible.

In analogy to MSS, we call the signature scheme based on a such an sd-Merkle tree *sd-MSS*. Also, the pseudocode of how the scheme works is shown in 2. In chapter 5 we will look at our implementation of sd-MSS in comparison to other post-quantum and classical signature schemes, and comparing it to a basic sequentially-updatable MSS implementation. But before that, we present a theoretical generalization of this idea.

Merkle Tree Forest

As a generalization of Merkle trees and (cached) sd-Merkle trees we present Merkle tree forests (MTFs). An h_1-h_2-...-h_t-MTF is a forest of t trees of heights h_1, h_2, ..., h_t respectively. Normally we would expect: $h_1 \leq h_2 \leq \ldots \leq h_t$. Therefore, any non-cached sd-Merkle tree with parameters (s, d) is an s-d-MTF, or if cached an s_c-d-MTF (which is) instead. This analogy of course only works using the modified definition of sd-Merkle trees with the root node removed.

We can see that an h_1-h_2-...-h_t-MTF has the following properties (assuming $h_1 \leq h_2 \leq \ldots \leq h_t$):

- Total number of faults that can be tolerated: $2^{h_1} + 2^{h_2} + \cdots + 2^{h_t} - 1$
- Overhead for the signatures in the ith tree: $h_i + i$ hash values
- Min. overhead: $h_1 + 1$ hash values
- Max. overhead: $h_t + t$ hash values
- Server-side storage needed: t hash values

Algorithm 2 sd-MSS(w, n, s, d, h, f)

```
procedure KeyGen($seed$)
    $(c_s, c_{s,new}, c_d, c_{d,new}) \leftarrow (0, 0, 0, 0)$
    $(s_{s,0}, \ldots, s_{s,2^s-1}) \leftarrow (f_{seed}(``shallow'', 0), \ldots, f_{seed}(``shallow'', 2^s - 1))$
    $(p_{s,0}, \ldots, p_{s,2^s-1}) \leftarrow (\text{WotsPK}(s_{s,0}), \ldots, \text{WotsPK}(s_{s,2^s-1}))$
    $(s_{d,0}, \ldots, s_{s,2^d-1}) \leftarrow (f_{seed}(``deep'', 0), \ldots, f_{seed}(``deep'', 2^d - 1))$
    $(p_{d,0}, \ldots, p_{s,2^d-1}) \leftarrow (\text{WotsPK}(s_{d,0}), \ldots, \text{WotsPK}(s_{d,2^d-1}))$
    $pk_d \leftarrow \text{MerkleRoot}(p_{d,0}, \ldots, p_{d,2^d-1})$
    return $(seed,\ p_{s,0},\ \ldots,\ p_{s,2^s-1},\ pk_d)$
end procedure

procedure Sign($msg,\ c_{s,remote},\ c_{d,remote},\ seed$)
    if $c_{s,remote} > c_s$ or $c_{d,remote} > c_d$ then
        $(c_s, c_d) \leftarrow (c_{s,remote}, c_{d,remote})$
        $pk_d \leftarrow pk_{d,new}$
    end if
    if $c_{s,new} < c_s + 2^s$ then
        $c_{s,new} \leftarrow c_{s,new} + 1$
        $s_{s,c_{s,new}+2^s-1} \leftarrow f_{seed}(``shallow'', c_{s,new} + s^2 - 1)$
        $p_{s,c_{s,new}+2^s-1} \leftarrow \text{WotsPK}(s_{s,c_{s,new}+2^s-1})$
        $\sigma \leftarrow \text{MerkleSign}(msg)$
        return $(i = c_{s,new} - c_s - 1, \sigma, p_{s,c_s+2^s}, \ldots, p_{s,c_{s,new}+2^s-1})$
    else
        $c_{d,new} \leftarrow c_{d,new} + 1$
        $s_{d,c_{d,new}+2^d-1} \leftarrow f_{seed}(``deep'', c_{s,new} + s^2 - 1)$
        $p_{d,c_{d,new}+2^d-1} \leftarrow \text{WotsPK}(s_{s,c_{s,new}+2^s-1})$
        $pk_{d,new} \leftarrow \text{MerkleRoot}(p_{d,c_{d,new}}, \ldots, p_{d,c_{d,new}+2^d-1})$
        $\sigma \leftarrow \text{MerkleSign}(msg)$
        return $(i = 2^s,\ \sigma,\ p_{s,c_s+2^s}, \ldots, p_{s,c_{s,new}+2^s-1}, pk_{d,new})$
    end if
end procedure

procedure Verify($msg,\ sig = (i, \sigma, p_0, \ldots, p_i, pk_{d,new}),\ c_{s,remote},\ c_{d,remote}$)
    if $i < 2^s$ then
        $r \leftarrow \text{WotsVerify}(\sigma, p_{s,c_s+i})$
    else
        $r \leftarrow \text{MerkleVerify}(\sigma, pk_d)$
    end if
    $(p_{s,c_s+2^s}, \ldots, p_{s,c_{s,new}+2^s-1}) \leftarrow (p_0, \ldots, p_i)$
    $pk_d \leftarrow pk_{d,new}$
    $(c_s, c_d) \leftarrow (c_{s,remote}, c_{d,remote})$
    return $r$
end procedure
```

The signature size is variable and depends on how often in a row authentication is expected to fail, leading to no reconciliation of the updatable Merkle tree between signing of messages. When given an estimate for the failure probability p_{fail} of a single authentication attempt, we can even calculate the expected average signature size overhead $\delta\sigma_{\text{avg}}$ as:

$$\mathbb{E}[\delta\sigma_{\text{avg}}] = \sum_{i=1}^{t} p_{\text{tree},i}(h_i + i) \text{ [hash values]}$$

where $p_{\text{tree},i} = p_{\text{fail}}^{(2^{h_0}+2^{h_1}+\dots+2^{h_i-1})}(1 - p_{\text{fail}}^{(2^{h_i})})$ for each $i \in \{0, 1, \dots, t\}$ is the probability of being in the state where tree i is being used.

4.4. Implementation

All code for the cryptographic primitives, benchmarks, performance estimates, and evaluation calculations is available in the following GitHub repository:

```
URL:         https://github.com/qkniep/PQ-FIDO_sd-MSS
Commit hash: 2f85f1a408503dff4b5c1bc510e315bf2cb87622
```

The `commit hash` refers to the version of the repository that this thesis is based on. Specifically, all measurement in chapter 5 were taken with this exact version of the source code.

The `signature_benchmark` directory contains implementations of the cryptographic primitives, all written in Rust. This includes implementations of our variant of W-OTS$^+$, the sequentially updatable Merkle tree, and the sd-Merkle tree. Not included is an implementation of Merkle tree forests. In the `OpenSK` directory there is a fork of Google's OpenSK with W-OTS$^+$ and sd-MSS added to the cryptography benchmark. This can be used for running the benchmarks directly on the nRF52840 board, whereas the `signature_benchmark` project is only meant to compile on non-embedded x86 systems.

Furthermore, there are some Python scripts available for creating graphs and tables based on estimated data. These are located in the `scripts` directory. The estimates are based on the formulas for size and time used by the different schemes. For example `shallow_deep_merkle_tree.py` generates an overview table, comparing sd-Merkle trees with different values for s and d to Merkle tree forests. Also, it generates graphs for signature size and signing time of sd-Merkle tree, which are also shown in chapter 5.

Then there is `ctss.py`, which generates a simple overview table comparing different FTS schemes, including XMSS, sd-Merkle trees, and two schemes CTSS and XCMSS which were predecessors of the sd-Merkle tree but are not discussed in this thesis. Finally, `sds_ots.py` is a small script used to estimate SDS-OTS security levels of the plain scheme, and with the adaptations presented in Appendix A.

5. Evaluation

In this chapter we analyze the security and practical applicability of the presented signature scheme. We compare sd-MSS to other post-quantum schemes, specifically Dilithium [11] and Falcon [14], and the classical state-of-the art signature schemes, represented by ECDSA-p256, and also to sequentially updatable Merkle trees without the shallow-deep tree construction.

5.1. Security Analysis

Here, the security levels of different schemes are compared. As quantum security levels we use the levels defined by NIST for the PQC competition [32], which are defined by equivalence to symmetric security levels of well-known primitives.

Primitives	Classical	Post-Quantum
ECDSA-p256	128	—
Dilithium2	≥ 128	I (AES-128)
Dilithium4	≥ 192	III (AES-192)
Falcon-512	≥ 128	I (AES-128)
Falcon-1024	≥ 256	V (AES-256)
W-OTS$^+$($n = 128$)	$\sim 128 - \log w^2$	I (AES-128)
W-OTS$^+$($n = 192$)	$\sim 192 - \log w^2$	III (AES-192)
W-OTS$^+$($n = 256$)	$\sim 256 - \log w^2$	V (AES-256)
sd-MSS ($n = 128$)	$\sim 128 - d - \log w^2$	I (AES-128)
sd-MSS ($n = 192$)	$\sim 192 - d - \log w^2$	III (AES-192)
sd-MSS ($n = 256$)	$\sim 256 - d - \log w^2$	V (AES-256)

Table 5.1.: Security levels (classical and quantum) of the different PQ cryptographic primitives.

The Winternitz OTS scheme's security is reduced to preimage attacks on the underlying hash function, and therefore offers a security level that is similar to key recovery on AES with keys of the same length as the hash output. If instantiated with XMSS trees as shallow and deep trees, our scheme's theoretical security is the same as the minimum of the two XMSS security levels. Using the formula from [8] together with the formula in [20] this gives us an upper bound on the security level b in bits:

$$b \leq n - 2 - \min \{s - \log_2 (w_s^2 l + w_s), d - \log_2 (w_d^2 l + w_d)\}$$

Or, with $w_s = w_d$ and assuming $d > s$ we have simply:

$$b \leq n - 2 - d - \log_2 (w^2 l + w)$$

As a specific example for $n = 128, w = 64, d = 7$, this comes out to:

$$128 - 2 - 7 - \log_2(64^2 * 24 + 64) \simeq 102 \text{ (bits)}$$

5.2. Performance Evaluation

All code for the benchmarks is also included in the GitHub repository that was linked in section 4.4. Wherever possible the benchmarks here have been performed directly on the nRF52840 board, its hardware specifications were presented in subsection 4.1.2.

5.2.1. W-OTS$^+$ Parameters

On most modern hardware one would implement the Winternitz OTS scheme using AES. A benchmark by the wolfSSL developers though suggests that SHA-256 is faster than AES on the CryptoCell 310 (see Appendix B).

Because we use a custom implementation of W-OTS$^+$, we first look at the performance of this on its own. Table 5.2 shows these results as measured on the nRF52840. These measurements will later be used to estimate the overhead of the FTS scheme over the pure W-OTS$^+$ calculations. They also serve as an estimate for the performance difference between security levels $n = 128$ (PQ: I), $n = 192$ (PQ: III), and $n = 256$ (PQ: V).

In the following benchmarks we will no longer look at verification times, as those are not performed on the authenticator. Instead verification runs on the RP server, which is usually orders of magnitude more powerful. Also, the server probably features a processor with hardware support for SHA-256 anyways.

w	n	$s_{sk}[B]$	$s_{pk}[B]$	$s_{sig}[B]$	$t_{gen}[ms]$	$t_{sig}[ms]$	$t_{ver}[ms]$
16	128	16	16	592	160	72.0	86.7
16	192	24	24	1,272	365	173	194
16	256	32	32	2,208	594	288	306
64	128	16	16	405	443	200	241
64	192	24	24	864	977	460	517
64	256	32	32	1,493	1,638	764	874
256	128	16	16	320	1,328	548	783
256	192	24	24	672	3,044	1,369	1,679
256	256	32	32	1,152	4,980	2,338	2,649

Table 5.2.: W-OTS$^+$ benchmark performed on the nRF52840 board for different parameters w and n. Showing key sizes, signature size, and timings of the single function calls.

5.2.2. sd-MSS Parameters

In Figure 5.1 we see the signature times of sd-MSS for some reasonable values of s and for different expected failure rates. In this setting we can understand lower failure rates to mean normal operation, without active attacks and unusually bad network conditions. High failure rates on the other hand represent scenarios where an attacker is actively running a DoS attack or with unexpectedly bad network conditions where the TLS connection is interrupted frequently. We see that even quite small shallow trees ($s = 2$ or $s = 3$) drastically reduce the average signature size, moving it very close to the minimum. On the other hand, for larger values (such as $s = 4$) there is basically no further improvement for failure rates below 50%. We should remind ourselves here that, as part of our attack model, we assume a maximum average failure rate of 50% anyways. If we didn't, DoS attacks would be a bigger hindrance than large signatures. The maxima are the same for all parameters because building the shallow tree does not significantly influence the total time, or not at all if we assume the calculated public keys to be cached on the authenticator.

In Figure 5.2 we see a similar graph as we saw above for the signing times. This time we look at signature sizes of sd-MSS for the same values of s and for all the expected failure rates. The most notable difference to the time graph is that the maxima are now significantly different (signature size grows exponentially in s). Because of the server-side caching of the shallow tree, the authenticator needs to send all the new OTS public keys from the shallow tree in addition to the OTS, authentication path, and new deep tree root hash. This is further reason not to use larger values for s because for $s = 4$ the maximum is already more than two times the minimum. Also,

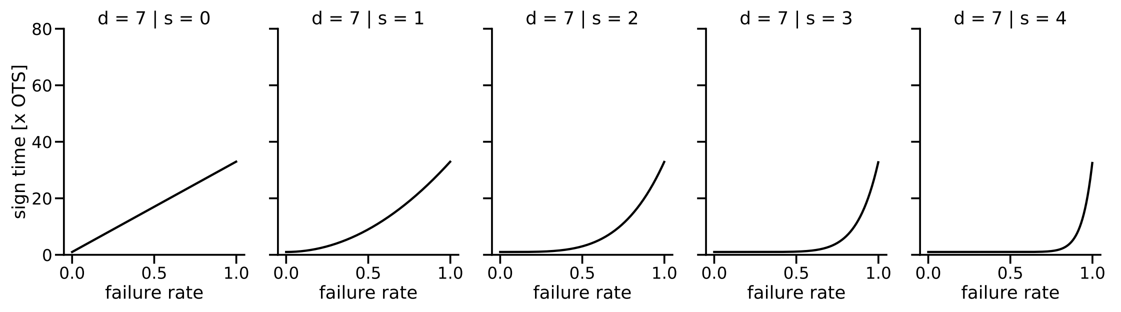

Figure 5.1.: Average time for signing a message under sd-MSS (with $w = 64, n = 128$) for different failure rates. This assumes two layers of the deep tree are cached client-side (shrinks the graph by factor ~ 4).

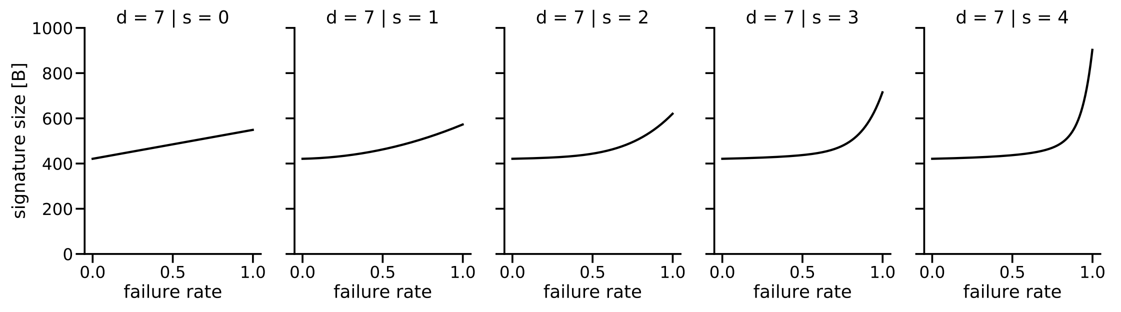

Figure 5.2.: Average signature size under sd-MSS (with $w = 64, n = 128$) for different failure rates.

this increase might even be reason to chose $s = 2$ over $s = 3$.

Here, we present a first benchmark of the FTS schemes (MSS and sd-MSS) compared to each other, to Falcon and Dilithium (as two of the NIST PQ standardization candidates), and to ECDSA. Table 5.3 shows the results of this benchmark which was run on a modern work laptop, not (as the other benchmarks) on the nRF52840. MSS here means our implementation of a sequentially-updatable Merkle tree, sd-MSS is our proposed scheme, Falcon and Dilithium are the `liboqs` [1] implementations of the two schemes, and ECDSA-p256 is the OpenSK's internal implementation of said scheme. We include this benchmark to provide at least some comparison to NIST PQ cryptography candidates, even though we could not get the available Falcon and Dilithium implementations to compile in the embedded environment of the nRF52840. It can be seen that keypair generation is three to four orders of magnitude more expensive, whereas for signing the difference is less extreme.

[1]https://github.com/open-quantum-safe/liboqs

Scheme	$s_{Reg}[B]$	$t_{Reg}[\mu s]$	$s_{Auth}[B]$	$t_{Auth}[\mu s]$
ECDSA-p256	33	20.0	72	19.4
Falcon512	897	14.1	690	4.1
Dilithium2	1,312	27.5	2,420	71.6
MSS ($h = 7, w = 16$)	16	13,049	704	189
MSS ($h = 7, w = 64$)	16	37,956	517	477
MSS ($h = 7, w = 256$)	16	114,150	432	1,285
sd-MSS ($s = 2, d = 7, w = 16$)	80	13,979	630.7	159
sd-MSS ($s = 3, d = 7, w = 16$)	144	14,321	624.4	160
sd-MSS ($s = 2, d = 7, w = 64$)	80	38,894	443.7	441
sd-MSS ($s = 3, d = 7, w = 64$)	144	40,611	437.5	412
sd-MSS ($s = 2, d = 7, w = 256$)	80	117,380	358.7	1,298
sd-MSS ($s = 3, d = 7, w = 256$)	144	120,630	352.4	1,269

Table 5.3.: Performance characteristics (data to transfer and timings) of the different cryptographic primitives, all with security parameter $n = 128$ (and PQ security level I, except for ECDSA, which offers no PQ security). Measurements are split into the two protocol steps: Registration and Authentication. The time measurements were taken on a laptop with an AMD Ryzen 5 4600H processor. s_{Auth} and t_{Auth} are averages, assuming 50% failure rate (which is the worst case in our model). Important to note: The time measurements here are in μs whereas most others in this thesis are in ms!

Next, we present a benchmark of the FTS schemes (MSS and sd-MSS) compared to each other and ECDSA, performed on the nRF52840 board. Table 5.4 shows results of this benchmark of just the cryptographic schemes (without FIDO functionality). MSS here means our implementation of a sequentially-updatable Merkle tree, sd-MSS is our proposed scheme, and ECDSA-p256 is the OpenSK's internal implementation of said scheme. Comparing these to the measurements of just W-OTS$^+$ (see Table 5.2) suggests the overhead in key generation is almost non-existent, signing on the other hand takes about three to four times as long, mostly due to the generation of a new OTS keypair.

5.2.3. CryptoCell

Table 5.5 compares the sd-MSS and ECDSA timings from above with estimates based on a CryptoCell 310 benchmark. The benchmarks were run on the nRF52840 hardware, except for the CryptoCell ones (marked by "CC310"), which are instead based on the other benchmarks (which use a software implementation of SHA-256) and comparative

Scheme	$s_{Reg}[B]$	$t_{Reg}[ms]$	$s_{Auth}[B]$	$t_{Auth}[ms]$
ECDSA-p256	33	104	72	144
MSS ($h = 7, w = 16$)	16	20,201	704	270
MSS ($h = 7, w = 64$)	16	57,204	517	679
MSS ($h = 7, w = 256$)	16	172,228	432	1,932
sd-MSS ($s = 2, d = 7, w = 16$)	80	20,887	630.7	234
sd-MSS ($s = 3, d = 7, w = 16$)	144	21,519	624.4	233
sd-MSS ($s = 2, d = 7, w = 64$)	80	58,991	443.7	645
sd-MSS ($s = 3, d = 7, w = 64$)	144	61,488	437.5	646
sd-MSS ($s = 2, d = 7, w = 256$)	80	177,611	358.7	1,896
sd-MSS ($s = 3, d = 7, w = 256$)	144	185,148	352.4	1,882

Table 5.4.: Performance characteristics (data to transfer and timings) of the different cryptographic primitives, all with security parameter $n = 128$ (and PQ security level I, except for ECDSA, which offers no PQ security). Measurements are split into the two protocol steps: Registration and Authentication. The time measurements were taken on the nRF52840-DK board. s_{Auth} and t_{Auth} are averages, assuming 50% failure rate (which is the worst case in our model).

benchmarks found in Appendix B. The wolfSSL benchmark was used to find the estimate factor of 50, which we assume to be the speedup the CryptoCell would offer to SHA-256 hash calculation. This factor was derived as:

$$\frac{25.903 \text{ MB/s}}{1024 \text{ B}/1.989 \text{ ms}} \approx 50$$

because the wolfSSL benchmark gave the speed of SHA-256 on the CryptoCell as 25.903 MB/s running on 1,024-byte data blocks, whereas our measurements showed that the software implementation of SHA-256 takes 1.989 ms per 1,024-byte block. Also, the running time of sd-MSS is almost entirely based on W-OTS$^+$ calculation and authentication path calculations, both of which are sped up by a faster implementation of the underlying hash function. However, since the benchmark used 1,024 B blocks (as opposed to the small 16-byte block used in W-OTS), the estimate could be somewhat optimistic.

It can be seen from the data that the authentication step can be performed very quickly; even the pure software implementation could be considered fast enough (below one second) for the use case of a FIDO authenticator. And the CryptoCell implementation is likely competitive to or even faster than ECDSA on the CryptoCell. Only the registration step which involves the keypair generation is the bottleneck.

Scheme	$s_{Reg}[B]$	$t_{Reg}[ms]$	$s_{Auth}[B]$	$t_{Auth}[ms]$
ECDSA-p256	33	104	72	144
ECDSA-p256, CC310*	33	18.5	72	20.1
sd-MSS ($s = 3, d = 7, w = 16$)	144	21,519	624.4	233
sd-MSS, CC310 ($s = 3, d = 7, w = 16$)**	144	430	624.4	4.7
sd-MSS ($s = 3, d = 7, w = 64$)	144	61,487	437.5	652
sd-MSS, CC310 ($s = 3, d = 7, w = 64$)**	144	1,230	437.5	13
sd-MSS ($s = 3, d = 7, w = 256$)	144	185,148	352.4	1,882
sd-MSS, CC310 ($s = 3, d = 7, w = 256$)**	144	3,703	352.4	38

Table 5.5.: Comparison of the cryptographic primitives' software implementations to implementations using the ARM CryptoCell 310, all with security parameter $n = 128$. Measurements are split into the two protocol steps: Registration and Authentication. (*) This measurement was performed independently by the wolfSSL developers (see Appendix B). Therefore, this can not be considered a comparative benchmark. (**) These measurements are based on a simple conversion factor of 50, derived from the same wolfSSL benchmark. Therefore, these values should only serve as a rough estimate.

5.2.4. Further Improvements

Further measures to improve the performance (especially regarding keypair generation) are however possible:

Firstly, one could try to combine the advantages of different w parameter values, namely small signatures (for large w) and fast computation (for small w). To do so, the shallow tree could work with a large w (e.g. $w = 256$) and the deep tree with a small w (e.g. $w = 16$ or smaller). Then, during key generation only a small number (2^s) of expensive long-chained W-OTS$^+$ instances needs to be calculated. The minimum size of signatures would be as small as if the whole sd-Merkle tree had used $w = 256$. And, as we already saw, the average signature is very close to the minimum anyways, even for small s. It can be estimated from Table 5.5 that this could achieve signature sizes below 360 B with key generation times below 500 ms. The computation overhead over ECDSA is thus estimated to be lower than it is for Falcon, where it could be around 30× for keypair generation at PQ security level I [38]. At the same time signature sizes would on average be a lot lower. This even keeps the same security level, compared to the parameters given in section 5.1:

$$128 - 2 - \min\{3 - \log_2(256^2 * 16 + 256), 7 - \log_2(16^2 * 35 + 16)\} \simeq 102 \text{ (bits)}$$

Furthermore, the number of keypairs in the deep tree could be reduced significantly (e.g. using $s = 2, d = 4$). Then another backup mechanism would be needed, to prevent the user being locked out permanently (which with $s = 2, d = 4$ would happen after 20 failures). The naive solution would be to use an ECDSA and Falcon (or Dilithium) hybrid signature as backup solution.

Also, at least in the case of USB-powered authenticators, which are constantly powered and have lots of idle CPU cycles, offline computation can be used to offload some of the key generation times. The authenticator could use downtime to generate spare keypairs, which are stored in a central buffer and later (during `authenticatorMakeCredential`) assigned to a specific RP. This is not really feasible with NFC or Bluetooth authenticators as these might not be permanently powered on.

Another property that is rather specific to our FTS-based signature scheme is that it is stateful. Both client and server need to persistently store and update certain state variables. Most importantly, the client needs to immediately persist the counters before using a signature, to prevent ever using the same OTS keypair twice. The server needs to regularly update the stored public key, overwriting it with the new one generated from the sequentially-updatable Merkle trees. This step is less critical, as the scheme is supposed to handle failures and DoS attacks anyways, and to this end is resistant to temporary failures. The server can use the same mechanism to put off persisting the updated public keys to a later time, by simply not updating the counters even on a successful authentication. This should be limited to a fixed number of times though, to prevent using up to many of the keys. When using the sd-MSS for example, the server could allow up to 2^s authentications without updating the public key, only persisting the new public key once the deep subtree starts being used. This would still keep us mostly using the shallow subtree. If this parameter is part of the protocol the client could also directly refrain from sending the updated shallow public keys, only submitting updated public keys once they know the server might accept them. In terms of performance (e.g. signature size) this somewhat undermines our analysis from above because the server artificially increases the effective failure rates. In settings like globally distributed and replicated authentication servers, the lower rate of persisting data may however be well worth the (on average) larger signature sizes.

6. Conclusion

This thesis looked at post-quantum cryptographic primitives for the use-case of FIDO authenticators. The main-focus was on hash-based cryptography and how it may be used efficiently, possibly serving as a post-quantum replacement for ECDSA as the signature scheme of choice in WebAuthn/FIDO authentication and other similar applications. To achieve this efficiency, we took a step back from general signature schemes, and looked at few-time signature schemes based on Merkle trees instead.

Sequentially-updatable Merkle trees and sd-Merkle trees were presented as novel applications of Merkle trees, which can be used whenever signatures are used in a synchronous setting between only two parties, for example in challenge-response authentication protocols. The additional risks of being only able to sign finitely many messages before syncing public keys with the server again were assessed, as was the additional cost of server-side storage incurred.

Feasibility of an actual implementation of sd-MSS was further analyzed on the hardware of a specific FIDO authenticator. The current implementation based on software-implemented SHA-256 was shown to be too resource intensive for this restricted platform. On the other hand, we gave an outlook on what is possible once hardware-accelerated SHA-256 is available (for example via the CryptoCell). It was shown that this reduces signing and verification times far below 100 ms and keypair generation in the low seconds range (possibly down to 500 ms). This makes the scheme generally usable. Also further adaptations or tradeoffs were discussed that could be made to try to reduce or mitigate the long key generation times. At the same time our scheme has very small signature sizes (and even smaller public keys) compared to other post-quantum schemes, competitive even with supersingular isogeny-based cryptography. Even though hash-based signature schemes were previously disregarded specifically because of their large signatures. This suggests that hash-based signatures are more flexible than previously thought.

6.1. Future Work

Open for the future is the task of more accurately analyzing the security level of our proposed scheme, sd-MSS. Also more possible applications of sequentially-updatable Merkle trees, sd-MSS, or even the more general (sequentially-updatable) Merkle tree forests could be explored.

Regarding the implementation there are open tasks in analyzing and improving its memory usage. Specifically, there are known problems with running out of memory on the nRF52840 for parameters with particularly high memory usage.

In general, this thesis could help encourage future work to consider hash-based cryptography as a viable alternative to lattice-based cryptography. For less computation-restricted environments it could be especially interesting, as the bottleneck part (keypair generation) is almost perfectly parallelizable.

Bibliography

[1] Leonard M Adleman and Jonathan DeMarrais. A subexponential algorithm for discrete logarithms over all finite fields. *Mathematics of Computation*, 61(203):1–15, 1993.

[2] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *International Conference on Selected Areas in Cryptography*, pages 317–337. Springer, 2016.

[3] Aydin Aysu and Patrick Schaumont. Precomputation methods for faster and greener post-quantum cryptography on emerging embedded platforms. *IACR Cryptol. ePrint Arch.*, 2015:288, 2015.

[4] Reza Azarderakhsh, Matthew Campagna, Craig Costello, LD Feo, Basil Hess, Amir Jalali, David Jao, Brian Koziel, Brian LaMacchia, Patrick Longa, et al. Supersingular isogeny key encapsulation. *Submission to the NIST Post-Quantum Standardization project*, 152:154–155, 2017.

[5] Manuel Barbosa, Alexandra Boldyreva, Shan Chen, and Bogdan Warinschi. Provable security analysis of FIDO2. In *Annual International Cryptology Conference*, pages 125–156. Springer, 2021.

[6] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. SPHINCS: practical stateless hash-based signatures. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 368–397. Springer, 2015.

[7] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pages 2129–2146, 2019.

[8] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. XMSS-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[9] Bundesamt für Sicherheit in der Informationstechnik (BSI). Migration zu post-quanten-kryptografie. `https://www.bsi.bund.de/SharedDocs/Downloads/DE/`

`BSI/Krypto/Post-Quanten-Kryptografie.pdf`, Aug. 2020. Accessed October 17, 2021.

[10] Kevin Bürstinghaus-Steinbach, Christoph Krauß, Ruben Niederhagen, and Michael Schneider. Post-quantum tls on embedded systems: Integrating and evaluating kyber and sphincs+ with mbed tls. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 841–852, 2020.

[11] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 238–268, 2018.

[12] Haonan Feng, Hui Li, Xuesong Pan, and Ziming Zhao. A formal analysis of the FIDO UAF protocol. In *Proceedings of the Annual Network & Distributed System Security Symposium (NDSS)*, 2021.

[13] Scott R Fluhrer. Reassessing Grover's algorithm. *IACR Cryptol. ePrint Arch.*, 2017:811, 2017.

[14] Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. Falcon: Fast-fourier lattice-based compact signatures over NTRU. *Submission to the NIST's post-quantum cryptography standardization process*, 36, 2018.

[15] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[16] Shafi Goldwasser, Silvio Micali, and Ronald L Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on computing*, 17(2):281–308, 1988.

[17] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[18] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. NTRU: A ring-based public key cryptosystem. In *International Algorithmic Number Theory Symposium*, pages 267–288. Springer, 1998.

[19] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018.

[20] Andreas Hülsing. W-OTS+–shorter signatures for hash-based signature schemes. In *International Conference on Cryptology in Africa*, pages 173–188. Springer, 2013.

[21] Charlie Jacomme and Steve Kremer. An extensive formal analysis of multi-factor authentication protocols. *ACM Transactions on Privacy and Security (TOPS)*, 24(2):1–34, 2021.

[22] Kassem Kalach and Reihaneh Safavi-Naini. An efficient post-quantum one-time signature scheme. In *International Conference on Selected Areas in Cryptography*, pages 331–351. Springer, 2015.

[23] Thivya Kandappu, Vijay Sivaraman, and Roksana Boreli. A novel unbalanced tree structure for low-cost authentication of streaming content on mobile and sensor devices. In *2012 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, pages 488–496. IEEE, 2012.

[24] Sabyasachi Karati and Reihaneh Safavi-Naini. K2SN-MSS: An efficient post-quantum signature. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 501–514, 2019.

[25] Leslie Lamport. Constructing digital signatures from a one-way function. *Report SRI Intl. CSL 98*, 1979.

[26] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[27] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for fft hashing. In *International workshop on fast software encryption*, pages 54–72. Springer, 2008.

[28] Soundes Marzougui and Juliane Krämer. Post-quantum cryptography in embedded systems. In *Proceedings of the 14th International Conference on Availability, Reliability and Security*, pages 1–7, 2019.

[29] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[30] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[31] Ralph Charles Merkle. *Secrecy, authentication, and public key systems.* Stanford university, 1979.

[32] National Institute of Standards and Technology. 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`, Dec. 2016. Accessed October 17, 2021.

[33] National Institute of Standards and Technology. Round 2 submissions. `https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`, 2017. Accessed October 17, 2021.

[34] Chang-Seop Park. One-time password based on hash chain without shared secret and re-registration. *Computers & Security*, 75:138–146, 2018.

[35] Geovandro CCF Pereira, Cassius Puodzius, and Paulo SLM Barreto. Shorter hash-based signatures. *Journal of Systems and Software*, 116:95–100, 2016.

[36] John M Pollard. Factoring with cubic integers. In *The development of the number field sieve*, pages 4–10. Springer, 1993.

[37] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 387–394, 1990.

[38] Markku-Juhani O Saarinen. Mobile energy requirements of the upcoming NIST post-quantum cryptography standards. In *2020 8th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 23–30. IEEE, 2020.

[39] Furqan Shahid, Iftikhar Ahmad, Muhammad Imran, and Muhammad Shoaib. Novel one time signatures (NOTS): A compact post-quantum digital signature scheme. *IEEE Access*, 8:15895–15906, 2020.

[40] Furqan Shahid and Abid Khan. Smart digital signatures (SDS): A post-quantum digital signature scheme for distributed ledgers. *Future Generation Computer Systems*, 111:241–253, 2020.

[41] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[42] Teik Guan Tan, Pawel Szalachowski, and Jianying Zhou. SoK: Challenges of post-quantum digital signing in real-world applications.

[43] Wen Wang, Bernhard Jungk, Julian Wälde, Shuwen Deng, Naina Gupta, Jakub Szefer, and Ruben Niederhagen. Xmss and embedded systems. In *International Conference on Selected Areas in Cryptography*, pages 523–550. Springer, 2019.

[44] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 1999.

A. Possible adaptations to SDS-OTS

Contrary to the claims by SDS-OTS authors Shadid et al. [40], arguments presented in subsection 3.3.2 suggest security levels of the scheme may be lower. To nevertheless achieve 128-bit classical security we can make certain adaptations to the scheme, which will be presented in the following.

Here we make the security level of SDS-OTS as specified in [40] more concrete than the rather simplified arguments in subsection 3.3.2. The Python script `sds_ots.py` in our GitHub repository implements a simulation of SDS-OTS with message length 512 bits and $w = 16$. The simulation generates random messages and looks at how often each possible number of hash cycles appears. This is central to the security of SDS-OTS because the number of cycles for each position uniquely identifies a signature under a specific keypair. Therefore, if there are fewer than 2^n possible signatures, reaching n-bit security is impossible. And if any signature is significantly more likely to occur than 2^{-n}, then finding two messages for this signature is significantly less expensive than 2^n attempts.

Running this simulation for 10,000,000 rounds in one example gave the following estimates:

```
Expected probability: 0.0078125
Max probability: [66] 0.015820000000007023
Median probability: 0.007159999999999268
Sum of probabilities: 1.0000000000002345
WC Security Level: 95.71370543941407
Max theoretical SL: 112
```

We see that the practically achieved security level is even significantly worse than the theoretically possible 112 bits. Instead, only about 96 bits of security is achieved. This is because signatures are not uniformly distributed over the messages. Specifically values close to 64 are more likely to appear for the number of hash cycles, making certain signatures more likely than others. For example, here we see the value 66 is more than twice as likely as if values were uniformly distributed.

If instead of summing the digits, we sum the numbers directly, we see that values are much more uniformly distributed. One example, again running for 10,000,000 rounds, gave us:

```
Expected probability: 0.0078125
Max probability: [1] 0.008124999999999327
Sum of probabilities: 0.9999999999998797
WC Security Level: 111.09466354614003
Median Security Level: 112.00000000000291
Max theoretical SL: 112
```

Now the worst case security starts to approach the theoretically possible security level. The one bit still missing in security can most likely be regarded as a small constant factor, and ignored. The bigger problem now is that the theoretical maximum of 112 is still much less than the 128 bits of security they wanted to achieve.

This final problem can be fixed by using 256 as modulus, instead of 128. One example, also running for 10,000,000 rounds, gave us the following numbers:

```
Expected probability: 0.00390625
Max probability: [209] 0.004071250000000465
Sum of probabilities: 1.0000000000001317
WC Security Level: 127.0449995206519
Median Security Level: 127.99999999999687
Max theoretical SL: 128
```

We get a theoretical maximum security level of 128 bits ($= \log_2 256^{16}$) now, which as achieved within a factor of 2 (127 bit security in the worst case). This can finally be considered as 128 bit classical security level. Though, now because we doubled the modulus in the calculation for the number of cycles, we unfortunately also doubled the computational effort necessary for keypair generation, signing, and verification.

B. CryptoCell 310 Benchmark by wolfSSL developers

Benchmark results taken from the following git repository:

```
https://github.com/wolfSSL/wolfssl/tree/master/IDE/CRYPTOCELL
Commit hash: c729318dddd33303ce62af2887c8f84df1836008
```

According to the included `README` file this benchmark data stems from a Nordic nRF52840 development board (the same hardware used in this thesis) running the Nordic nRF5-SDK of version `nRF5_SDK_15.2.0_9412b96`:

```
Benchmark Test Started
-----------------------------------------------------------
 wolfSSL version 3.15.7
-----------------------------------------------------------
wolfCrypt Benchmark (block bytes 1024, min 1.0 sec each)
RNG                  5 MB took 1.000 seconds,    4.858 MB/s
AES-128-CBC-enc     17 MB took 1.001 seconds,   17.341 MB/s
AES-128-CBC-dec     17 MB took 1.000 seconds,   17.285 MB/s
SHA                425 KB took 1.040 seconds,  408.654 KB/s
SHA-256             26 MB took 1.000 seconds,   25.903 MB/s
HMAC-SHA           425 KB took 1.049 seconds,  405.148 KB/s
HMAC-SHA256         24 MB took 1.000 seconds,   23.877 MB/s
ECC   256 key gen  55 ops took 1.017 sec,     avg 18.491 ms
ECDHE 256 agree    56 ops took 1.017 sec,     avg 18.161 ms
ECDSA 256 sign     50 ops took 1.004 sec,     avg 20.080 ms
ECDSA 256 verify   48 ops took 1.028 sec,     avg 21.417 ms
Benchmark Test Completed
```

Results for RSA and the additional measure `ops/sec` for the ECC schemes have been omitted for brevity. Formatting, specifically spacing, was also adapted.

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 22. Dezember 2021 ....