z-OTS: a one-time hash-based digital signature scheme with fast verification

Amos Zheng, Marcos A. Simplicio Jr.

***Abstract*—Hash-based signature schemes are a class of post-quantum algorithms usually built upon one-time signature (OTS) solutions via hash-trees. The benefits of such schemes include small key sizes, efficient processing and the fact that they are simple to implement using a regular hash algorithm. In addition, their security properties are quite well understood, since they rely basically on the pre-image or collision resistance of the underlying hash function. Among the existing OTS schemes, W-OTS+ is among the most popular. One reason for such popularity is that the OTS public key can be recovered from the signature itself, which facilitates the construction of a multi-time signature scheme using Merkle trees. On the other hand, signature generation and verification in W-OTS+ take roughly the same time, which is not ideal for applications where each signature is expected to be verified several times, as in software stores, PKI certificate validation, and secure boot. It is also inconvenient when the devices that verify signatures have lower computational power than the signers. In such scenarios, it is desirable to design signature schemes enabling faster verification, even if such speed-ups come at the expense of a slower signature generation procedure. With this goal in mind, we hereby present and evaluate a novel OTS scheme, called z-OTS. The main interest of z-OTS is that it preserves all benefits of W-OTS+, but provides faster signature verification at the cost of a (not much) slower signature generation procedure. For example, for signature sizes equivalent to W-OTS+ with Winternitz parameter $w = 4$, our simulations show that verification can be 30.3% faster with z-OTS, while key and signature generation become, respectively, 53.7% and 137.5% slower. Larger w leads to even more expressive gains in the verification procedure, besides providing lower overheads when generating keys and signatures.**

***Index Terms*—Post-quantum cryptography, digital signature, hash-based signatures, W-OTS+.**

I. Introduction

THE area of post-quantum cryptography refers to algorithms that can be executed on classic computers and, at the same time, resist known attacks made by quantum computers [1]. The motivation for developing such cryptosystems is to face the threat posed by quantum computation, an area that is evolving increasingly fast [2]. More precisely, if large-enough quantum computers become available, the security of many classical cryptosystems, in particular asymmetric primitives such as RSA [3] and schemes based on elliptic curves [4], can be threatened by attackers using Shor's algorithm [5]. This creates the need for efficient schemes that can become drop-in replacements for conventional solutions currently employed in a variety of security-aware applications.

Amos Zheng and Marcos Simplicio Jr. are with the Department of Computer and Digital Systems Engineering, Universidade de São Paulo, São Paulo, SP, Brazil. Email: {azheng,msimplicio}@larc.usp.br

Manuscript received NNNN NN, NNNN; revised NNNN NN, NNNN.

Among the existing approaches for building post-quantum cryptosystems, a particularly promising area refers to hash-based signatures [1]. Indeed, such schemes are been considered not only for desktop and server applications but also for use in embedded systems [6]. In a nutshell, hash-based signatures use a list of random bit strings as the private key, and then hash those strings one or more times to generate the corresponding public key; the signature is then produced by revealing the pre-images associated with that public-key, depending on the data to be signed. One benefit of this approach is that the (quantum and classical) security of the obtained signatures relies basically on well-known properties of hash functions, such as pre-image and collision resistance. In addition, hash-based signatures are also quite efficient. Specifically, besides leading to small public key sizes, their computational costs come basically from calls to an underlying hash function, a primitive that is known to be processing- and memory-efficient algorithms. These properties are primarily leveraged in the design of one-time signature (OTS) schemes, such as W-OTS and W-OTS+ [7], where each private key can only be used in a single signature. By combining such OTS with a suitable data structure, such as Merkle Trees [8], one can then create many-times signature (MTS) schemes for allowing private key reuse. Examples of MTS solutions following this strategy include XMSS-MT [9] and SPHINCS+ [10], the latter being a candidate in round three of NIST's Post-Quantum Cryptography Competition [11].

Despite this interest, hash-based signatures still face some challenges. Besides leading to reasonably large signature sizes when compared to classical digital signature schemes, they usually display similar times for message signature and verification. This latter property is not ideal for applications where signatures are verified quite often when compared to their generation frequency. This is the case of software stores (e.g., Microsoft Store, Google Play Store, and Apple's App Store), where software packages are signed once and verified every time a user installs them, for ensuring authenticity and integrity. It also the case of certificates in any public-key infrastructure, where, once a certificate is generated and signed, it is verified many times within its validity period (e.g., a few years). Another example refers to secure boot mechanisms, where a firmware, BIOS, boot-loader, or kernel are signed only once and verified for integrity every time a machine boots, aiming to prevent software tampering. In such applications, the verification overhead should be as little as possible, whereas a comparatively larger cost in the signing process could be more easily tolerated.

Even though there are some works in the literature aimed at

reducing verification time, the speedups obtained are usually lower than 10% [12] [13]. Aiming to facilitate the construction of hash-based signature schemes with highly efficient verification procedures, this paper proposes a novel hash-based one-time signature scheme called z-OTS. Basically, this scheme combines randomized hashing [14] with the generalized non-adjacent form [15] to create a message encoding capable of speeding-up signature verifications. Similarly to W-OTS+, z-OTS enables trade-offs between signature size and processing time, and also allows public keys to be derived from signatures. However, by efficiently mapping groups of bits into hash chains, z-OTS provides much faster verification times than W-OTS+ for similar signature sizes with a reasonably small increase in the corresponding key and signature generation costs.

The rest of this paper is organized as follows. Section II discusses related works. Section III gives the necessary background on the z-NAF encoding, which is the basis for the proposed z-OTS scheme. Section IV describes our proposal in detail. Section V analyzes the security of z-OTS. Its performance in terms of signature size, processing time, and storage requirements are theoretically and experimentally evaluated, respectively, in Sections VI and VII. Section VIII concludes the discussion and presents our final considerations.

II. Related Works: Hash-Based Signatures

Originally, hash-based signature schemes were exclusively "one-time", i.e., for each public-private key pair, only one message could be signed. Indeed, the first publicly known one-time signature (OTS) scheme was the Lamport-Diffie (LD) OTS [16], which is basically a bit-signing scheme: each bit of the message is signed independently. However, being able to sign a single message per key pair is not very practical in most real-world scenarios. After all, the management of disclosed public keys (e.g., by provisioning digital certificates) would become quite cumbersome [1]. To address this specific limitation, Ralph Merkle proposed using a hash tree (the "Merkle Tree") with multiple OTS schemes to build a many-times signature (MTS) scheme, i.e., a scheme where many messages can be signed with a single public/private key pair [8]. Besides creating a more practical hash-based signature structure, different OTS schemes can be associated with the same hash tree, so users can switch among them as desired.

Besides being one-time, another limitation with LD-OTS is that it requires as many private and public key elements as the message length. Hence, it leads to large public and private key sizes, as well as to large signatures. One approach for this issue, originally proposed in the W-OTS scheme [8], is to allow a trade-off between signature size and processing time. Basically, this is accomplished by using hash chains instead of single hashes over the private key bitstrings (see Figure 1). The hash chain length is set to 2^w, where the Winternitz parameter w controls the desired trade-off: the signature size decreases linearly with w, at the expense of exponential growth in the signature generation and verification costs. For this reason, the recommended values of w usually range from 1 to 4.

Additional tricks also exist in the literature to reduce the size of hash-based signatures. In particular, W-OTS+ [7] and

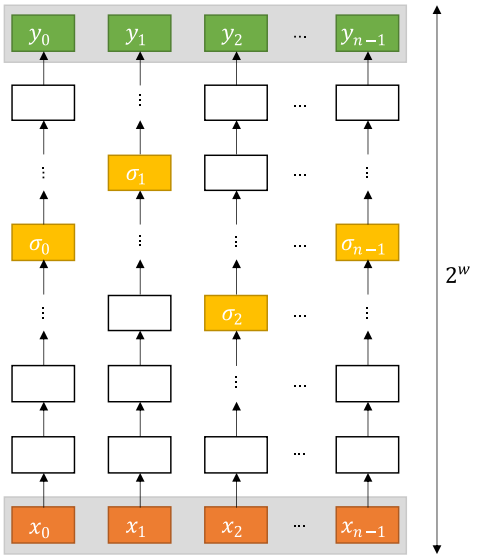


Fig. 1. W-OTS-like schemes use hash chains that start at the private key elements (bottom) and go all the way to the public key elements (top), where the arrows represent hashing or chaining function operations. The hash chain length, that is, the number of operations needed to reach the public key elements y_i from the private key element x_i, is given by 2^w, where w is the Winternitz parameter. The signature is shown as the collection of the highlighted elements somewhere in the middle of the hash chains. For the W-OTS family of one-time signature schemes, the signature elements are statistically located around the middle of the chains.

XMSS [17] are designed to require only pre-image resistant hash functions, while the need for collision resistance is eliminated in their internal modules. For example, W-OTS+ replaces regular hashing by a "chaining function": a keyed hash function whose input is XORed with different random bit-masks for each hash chain level [7]. When compared to LD-OTS, W-OTS+ leads to significantly smaller signatures, although the processing costs may be higher depending on w.

The improvements introduced in the aforementioned schemes generally target signature size reduction, while displaying similar signature generation and verification times. Nevertheless, there are also some schemes in the literature that, like our proposal, are aimed at improving the efficiency of the verification procedures, introducing trade-offs in terms of a higher generation cost, larger signature sizes, or bigger keys. Some examples of such OTS schemes are BiBa [18] and HORS [19], which are not variants of W-OTS. Actually, the latter is a few-times signature scheme, meaning that its security decline gradually if the same key is employed for signing different messages, whereas this happens abruptly with a regular OTS. Their main advantages include short signature sizes and fast verification times, at the cost of larger public keys and slower key generation. They have, however, one important limitation: their public keys cannot be derived from the corresponding signatures, which makes them harder to use in a many-times signature scheme based on Merkle trees. This issue has motivated recent schemes such as HORST [20] and FORS [10], which build upon HORS but allow public keys to be recovered from signatures. As a result, they can be easily employed as building blocks in a secure MTS scheme. The downside, however, is that the resulting signatures are

significantly larger than those obtained with W-OTS+. For example, for a 128-bit classical security level, W-OTS+ leads to signatures having less than 1kB, against 2.5kB in FORS.

There are also schemes that provide relatively short signatures and faster verification, while retaining the ability to recover public keys from a given signature. One example is the work done in [12], where the authors propose that messages should be hashed together with a counter $r \quad 0$ before being signed, similarly to what is suggested in [18]. This could then be done iteratively, until the hash value d obtained satisfies some desired condition. In particular, suppose that d has a maximum and a minimum number of non-overlapping runs of consecutive 0s and 1s, each of which having a length of at most $\grave{}$. In this case, d can be encoded into hash chains considering the run-lengths, rather than the integer values of each group of w bits. With this technique, signature verification can be made faster at the cost of longer signature generation times, because longer runs are more scarce than shorter runs. Unfortunately, the actual improvement in terms of verification time over W-OTS are quite small (less than 10%) [12]. Moreover, this approach loses some of W-OTS's flexibility, since it does not support a simple parameter for enabling signature size and processing time trade-offs.

Finally, a recent proposal by Roh et al. uses the conventional non-adjacent form (2-NAF) with the W-OTS+ scheme [13]. The scheme takes advantage of the zero imbalance of that form for accelerating signature generation at the cost of slower signature verification, or vice-versa. Albeit interesting, the gains remain quite small (around 8%). In addition, similarly to the run-length encoding solution from [12], it does not support trade-offs between signature size and processing time.

III. Background: z-NAF encoding

An important concept present in the proposed z-OTS is the z-non-adjacent form, abbreviated as z-NAF.[1] While most integer representations use unsigned digit representations (e.g., 0 and 1 in binary), z-NAF is a signed digit representation system. The representation system can, thus, have both positive and negative digits. The definition of z-NAF is provided in what follows.

Given an integer a and parameter $z \quad 2$, the z-NAF representation of a, written $a = \sum_{j=0}^{l} n_j 2^j$, is a signed integer representation that satisfies the following [15]:

Either $n_j = 0$ or n_j is odd and $jn_j j < 2^{z \quad 1}$;
If $n_j \neq 0$, then $n_{j+1} = \quad = n_{j+z \quad 1} = 0$.

For example, consider the integer $(50)_{10}$, whose binary form is $(110010)_2$. Its 2-NAF representation is $(1010010)_{2 \ NAF}$, its 3-NAF representation is $(30010)_{3 \ NAF}$, and its 4-NAF representation is $(100070)_{4 \ NAF}$, where a digit with a bar d represents the negative digit $\quad d$. This is valid because $50 = 1 \quad 2^6 \quad 1 \quad 2^4 + 1 \quad 2^1 = 3 \quad 2^4 + 1 \quad 2^1 = 1 \quad 2^6 \quad 7 \quad 2^1$.

The z-NAF representation has been independently introduced by Cohen et al. [21] and Solinas [22]. Since then, it was explored by many authors, mainly to speed up exponentiations and elliptic curve point multiplications (e.g., see [23]). In the proposed z-OTS scheme, the following properties are of particular interest:

Property 1: Every integer has a unique representation in z-NAF form (bijectivity) [15], [24], [25];
Property 2: Given an l-bit integer in ordinary binary form, the z-NAF form of that integer has no more than $l + 1$ digits in z-NAF form [25];
Property 3: An average z-NAF representation has a non-zero digit density of about $1=(z + 1)$ [25], [26].

Finally, it is worth mentioning that converting some input to its z-NAF representation is a quite fast operation. For example, [15] describes a linear-time algorithm to convert any integer in ordinary binary representation into its z-NAF form.

IV. Proposed scheme: z-OTS

This section describes the proposed z-OTS scheme and its three underlying algorithms: key generation, signature generation, and signature verification. For the convenience of the reader, Table I lists the main symbols and notation hereby adopted.

A. Parameters

Let n be the length of the data m to be signed, and b the desired (classical) security level; since the input for the signature process is commonly a hash generated with a collision-resistant hash function g, it is usual to have $n = 2b$. Also, let z denote the z-NAF parameter, and w the Winternitz parameter. In the proposed scheme, w is only used when signing the checksum of the message, since the checksum is not converted into a z-NAF form.

We define L_1 as the number of hash chains used to sign the message's z-NAF digits, and t_{max} as a parameter that limits the length of the message hash chains, len, given by

$$len = (1 + t_{max})2^{z \quad 1}$$

We use L_2 to represent the number of hash chains used to sign a checksum. It is computed as:

$$L_2 = \frac{d\log_2 L_1 + \log_2(1 + t_{max})e + z \quad 1}{w}$$

Like W-OTS+, z-OTS assumes a pre-image resistant underlying hash function. To this end, we employ a chaining function $c_k^j(x; r)$, where: $k \ 2 \ K_b$ is a function key belonging to key space K_b; and $r = (r_1; r_2; \quad ; r_{\max(len \ 1;2^w \ 1)})$ is a list of random bit-masks. Also, let

$$g : f0; 1g^* \quad ! \quad f0; 1g^n$$
$$h : f0; 1g^{b+jpj} \quad ! \quad f0; 1g^b$$

be cryptographic hash functions, where jpj is the length of a nonce p used in the signature generation algorithm. In practice, all functions c, h and g are commonly instantiated from the same underlying hash function H (e.g., a member from the SHA-2 [27] or SHA-3 [28] families), which is combined with different affixes for domain separation. Also, let PRF be a cryptographically secure pseudorandom function that, given a b-bit seed SEED_{in}, outputs a random number RAND and an updated seed SEED_{out}, both b-bits long.

[1] In the literature, this encoding is often called w-NAF, which stands for *w-width non-adjacent form*. In this paper, though, the "w" in w-NAF is replaced with "z" to avoid confusion with the Winternitz parameter w.

TABLE I
NOTATIONS USED IN THE DEFINITION OF THE z-OTS SCHEME.

Notation	Definition
b	Desired classical security level
z	The z-NAF parameter
w	The Winternitz parameter
L_1	Number of message hash chains
L_2	Number of checksum hash chains
t_{max}	Number of zeros allowed at the left of each non-zero z-NAF digit (excluding the mandatory zero digits)
len	Length of the message hash chains
x	Private key for signing messages
$(x_m)_i$	The i-th message secret string derived from private key x
$(x_c)_i$	The i-th checksum secret string derived from private key x
PRF	Pseudo-random function for generating $(x_m)_i$ and $(x_c)_i$ from x
$c_k^i(x;\boldsymbol{r})$	Chaining function to calculate the hash chains
k	Chaining function's function key
$\boldsymbol{r}$	Random bit-masks used in the chaining function
y	Main public key element
$(y_m)_i$	The i-th message public key element
$(y_c)_i$	The i-th checksum public key element
g	Hash function for generating y from $(y_m)_i$ and $(y_c)_i$
m	Message to be signed
n	Length of the message m being signed
p	Nonce hashed with m to produce m'
m'	Randomized version of m satisfying specified criteria
h	Hash function for generating m' from m and p
m'_{NAF}	Randomized message converted to z-NAF form
D_i	The i-th z-NAF digit from m'_{NAF}
$m'_{grouped}$	m'_{NAF} with mandatory zeros removed and digits grouped
D_i^0	The i-th grouped digit of $m'_{grouped}$
N_i	Non-zero digit in D_i^0
t_i	Zero-run length in D_i^0
l	Number of non-zero digits in m'_{NAF} ($l \leq L_1$).
m''	$m'_{grouped}$ with each D_i^0 digit converted to a number in $\mathbb{N}_0$
D_i''	Non-negative digits of m''
	Signature of m (comprises message and checksum parts)
$(\ \ _m)_i$	The i-th element from the message signature part of
$(\ \ _c)_i$	The i-th element from the checksum signature part of
$(y_m)_i^0$	The i-th message public key element recovered from $(\ \ _m)_i$
$(y_c)_i^0$	The i-th checksum public key element recovered from $(\ \ _c)_i$
y^0	Main public key recovered from signature

B. Key Generation

First, choose k, the randomization elements r and a b-bit private key x using a (pseudo)random number generator. Then, use x and the PRF to generate $L_1 + L_2$ secret strings with b bits, $((x_m)_{L_1\ \ 1}; \quad ;(x_m)_0)$ and $((x_c)_{L_2\ \ 1}; \quad ;(x_c)_0)$. Here, the set of $(x_m)_i$ denotes the secret strings used to sign the message, whereas $(x_c)_i$ denotes those used to sign the checksum. Subsequently, compute:

$$\begin{cases} (y_m)_i = c_k^{len\ \ 1}((x_m)_i; r) \\ (y_c)_i = c_k^{2^w\ \ 1}((x_c)_i; r) \end{cases}$$

Finally, make

$$y = g((y_m)_{L_1\ \ 1}jj \quad jj(y_m)_0jj(y_c)_{L_2\ \ 1}jj \quad jj(y_c)_0)$$

The scheme's private key is x while the public key is $(k; r; y)$.

C. Signature Generation

The goal of the proposed signature procedure is to map an n-bits message m to be signed into a verification-friendlier form m''. In a nutshell, this is done by taking advantage of the high zero density of the z-NAF encoding, mapping the fragments of the encoded message into "higher" positions of the hash chains (i.e., closer to the end of the chains, as shown in Figure 1). As a result, compared to signature generation, there are fewer hash operations to be performed during the verification phase. The mapping is such that each signature is unambiguously associated with a single original message m, as required to avoid forgeries.

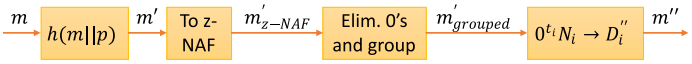


Fig. 2. Conversion chain of z-OTS in the signing algorithm, showing how m is converted to m''.

In what follows, we describe the 7 steps required by the proposed scheme, which can be grouped into two main procedures: the m to m'' conversion process (Steps 1 to 5, shown in Figure 2); and the mapping of digits from m'' into hash chains similar to W-OTS+ (Steps 6 and 7), producing the signature itself. We then give a toy example for exemplifying its application with actual numbers. We note that some of those steps can be combined at the implementation level, simplifying some operations that, for the sake of clarity, are hereby described separately.

Step 1: Calculate the randomized hash $m' = h(mjjp)$, where p is a nonce. Ideally, if one desires to eliminate the need for collision resistance of the hash function h through the use of randomized hashing [29], one should generate p (pseudo)randomly.

Step 2: Convert m' to z-NAF form, $m'_{NAF} = (D_n; D_{n\ \ 1}; \quad ; D_1; D_0)$. If m'_{NAF} has less than $n+1$ signed digits, it is padded with zeros on the left to complete $n+1$ signed digits. As mentioned in Section III, Property 1 of z-NAF guarantees that the conversion is possible, while Property 2 ensures that we need no more than $n+1$ signed digits to represent it in z-NAF form. Besides, by definition, the signed digits D_i are either zero or odd and $2^{z\ \ 1} < D_i < 2^{z\ \ 1}$. Therefore, there are $2^{z\ \ 1} + 1$ possible values for each D_i.

Step 3: Ensure that m'_{NAF} satisfy the following restrictions:

Restriction 1: The least significant digit D_0 must not be zero;

Restriction 2: The number of non-zero digits in m'_{NAF} must be equal to or less than L_1;

Restriction 3: The number of zeros on the left of each non-zero digit must be equal to or less than $z \quad 1 + t_{max}$.

These restrictions are essential so that the next steps of the key generation algorithm work properly and lead to gains in the verification procedure. Regarding the last restriction, we note that the properties of z-NAF ensure that $z \quad 1$ zeros necessarily appear on the left of each non-zero digit, except for the leftmost non-zero digit from m'_{NAF}; hence, we require that, besides those mandatory zeros, the number of additional zeros does not exceed t_{max}.

If any of those three restrictions is not satisfied, pick another nonce value and generate a new m'_{NAF} by repeating Steps 1 and 2. From our analysis, providing a parametric equation for the success probability of passing this step is a hard problem.

Nevertheless, when mimicking w-OTS signature sizes (i.e., setting $z = w - 1$ and $L_1 = n/(z+1)$), we found empirically that it takes on average from 10 to 30 calculations of h to find a m'_{NAF} satisfying these conditions for the parameters suggested in Section VII. Note that, since this procedure relies solely on public information (namely, message m and nonce p), eventual time variations resulting from such repetitions should not leak any secret information.

Step 4: Discard $z - 1$ zeros on the left of each non-zero digit from m'_{NAF}, or all of them if the number of zeros is less than $z - 1$ (which can only happen for the leftmost non-zero digit). Then, group the remaining digits in the form $m'_{grouped} = (D'_{l-1}, D'_{l-2}, \ldots, D'_1, D'_0) = (0^{t_{l-1}}N_{l-1}, 0^{t_{l-2}}N_{l-2}, \ldots, 0^{t_1}N_1, 0^{t_0}N_0)$, so that

- 0^{t_i} denotes a zero-run of length t_i, where $t_i \leq t_{max}$ as per Restriction 3;
- N_i are the non-zero digits that were present in m'_{NAF}; the total number of those digits, l, is such that $l \geq L_1$ due to Restriction 2;
- $l \leq \frac{n+1}{z}$, because, except for the leftmost non-zero digit, there are at least $z - 1$ zeros on the left of each other non-zero digit from m'_{NAF}.

This form of $m'_{grouped}$ shows why Restriction 1 is important: without requiring $D_0 \neq 0$, the rightmost sequence of zeros would not have any non-zero digit for grouping. Besides, note that the operation of eliminating the mandatory zeros does not result in ambiguity, i.e., it is such that two different m'_{NAF} result in distinct $m'_{grouped}$. The reason is that those zeros are always present in m'_{NAF}, so the removal operation can always be reverted. For a more detailed explanation, we refer the reader to the security analysis in Section V.

Step 5: With $m'_{grouped}$ at hand, generate $m'' = (D''_{L_1-1}, D''_{L_1-2}, \ldots, D''_1, D''_0)$ containing only non-negative digits. This can be done by applying the map $D'_i \mapsto D''_i$, defined as follows:

$$D''_i = \begin{cases} t_i 2^{z-1} + (N_i - 1)/2 & \text{if } N_i > 0 \text{ and } i < l \\ t_i 2^{z-1} + (2^z + N_i - 1)/2 & \text{if } N_i < 0 \text{ and } i < l \\ 0 & \text{if } i \geq l \end{cases}$$

The rationale behind this map is as follows. First, it is relatively simple to implement using only additions and bitwise shift operations. Second, it maps blocks D'_i that have less leading zeros, which are more common, into smaller values of D''_i; since a smaller D''_i is later translated into a signature element at higher positions in the hash chain, the resulting signature can be verified faster, at the cost of a more expensive signature procedure. Third, the resulting D''_i remain within the smallest possible interval in $\mathbb{N}_0$, enabling a compact binary representation. And fourth, different D'_i are mapped to distinct D''_i, ensuring bijectivity. Note also that, if the total number of blocks D'_i is less than L_1, the remaining D''_i digits are set to zero (third equation of the map) as a form of padding, so the signature size remains constant.

To give a concrete example of how this mapping works, consider $z = 3$. The non-zero digits $\{-1, 1, -3, 3\}$ would be mapped as follows: $1 \mapsto 0$; $3 \mapsto 1$; $-3 \mapsto 2$; and $-1 \mapsto 3$. Each extra leading zero adds $2^{z-1} = 4$ to D''_i, so the mapping for a D'_i with one leading zero would be: $01 \mapsto 4$; $03 \mapsto 5$; $0{-3} \mapsto 6$; and $0{-1} \mapsto 7$.

Step 6: Similar to W-OTS+, calculate the checksum as

$$\text{CHECKSUM} = \sum_{i=0}^{L_1-1} len - 1 - D''_i$$

and divide the checksum into L_2 blocks of w bits each, $(B_{L_2-1}, \ldots, B_0)$.

Step 7: Finally, once again like in W-OTS+, compute the signature $\sigma = (p, (\sigma_m)_{L_1-1}, \ldots, (\sigma_m)_0, (\sigma_c)_{L_2-1}, \ldots, (\sigma_c)_0)$ where

$$\begin{cases} (\sigma_m)_i = c_k^{len-1-D''_i}((x_m)_i, r) \\ (\sigma_c)_i = c_k^{2^w-1-B_i}((x_c)_i, r) \end{cases}$$

D. Signature Verification

Given a message m, its signature $\sigma = (p, (\sigma_m)_{L_1-1}, \ldots, (\sigma_m)_0, (\sigma_c)_{L_2-1}, \ldots, (\sigma_c)_0)$ and the public key (k, r, y), the verifier calculates m'' and the checksum CHECKSUM as in the signing algorithm. If, during the calculations, m'_{NAF} does not satisfy all three conditions given in the signing algorithm, the verifier rejects the signature as invalid. Otherwise, the verifier calculates y' via iterative hashing, "completing" the hash chains, as follows:

$$y' = g((y_m)'_{L_1-1} \| \ldots \| (y_m)'_0 \| (y_c)'_{L_2-1} \| \ldots \| (y_c)'_0)$$

where

$$\begin{cases} (y_m)'_i = c_k^{D''_i}((\sigma_m)_i, r) \\ (y_c)'_i = c_k^{B_i}((\sigma_c)_i, r) \end{cases}$$

The verifier accepts the signature if and only if $y' = y$.

E. A toy example

Suppose $n = 16$ bits, $z = 3$, $L_1 = 4$, $t_{max} = 3$ and $w = 4$ (the latter being used only for handling the signature's checksum). For those parameters, we have $L_2 = 2$ and $len = 16$.

1) Key Generation: Starting with k, r and a b-bit private key x, we derive the $L_1 + L_2 = 4 + 2$ secret b-bit strings, $((x_m)_3, (x_m)_2, (x_m)_1, (x_m)_0)$ and $((x_c)_1, (x_c)_0)$, and then compute $(y_m)_i = c_k^{15}((x_m)_i, r)$ and $(y_c)_m = c_k^{15}((x_c)_i, r)$. Finally, we compute $y = g((y_m)_3, (y_m)_2, (y_m)_1, (y_m)_0, (y_c)_1, (y_c)_0)$.

2) Signature Generation: With a nonce p, calculate $m' = h(m\|p)$. Suppose $h(m\|p_1) = 0000011011001010$, $h(m\|p_2) = 1111000110000001$ and $h(m\|p_3) = 0010001100001101$. Their corresponding 3-NAF representations are then $(0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 3, 0, 0, 3, 0)$, $(1, 0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 1)$ and $(0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 3)$ respectively. Because the first one fails *Restriction 1*, the second one fails *Restriction 3* and the last one passes all restrictions, we take p_3 and proceed with $m'_{3\,NAF} = (0, 0, 0, 1, 0, 0, 0, 0, 3, 0, 0, 0, 1, 0, 0, 0, 3)$.

By eliminating the $z - 1 = 2$ zeros on the left of each non-zero digit and grouping the remaining digits, we

get $m'_{grouped} = (01; 003; 01; 03)$. After applying the map $D'_i \mapsto D''_i$, we get $m'' = (4; 9; 4; 6)$, from which we get the checksum blocks $(B_1; B_0) = (4; 9)$. Finally, the signature is $= (p_3; c_k^{15-4}((x_m)_3; r); c_k^{15-9}((x_m)_2; r); c_k^{15-4}((x_m)_1; r); c_k^{15-6}((x_m)_0; r); c_k^{15-4}((x_c)_1; r); c_k^{15-9}((x_c)_0; r))$. This last procedure takes, thus, $6 \cdot 15 \cdot 36/54$ calls to c_k, whereas the average signing cost without the proposed technique would be $6 \cdot 15/2 = 45$ calls to c_k.

3) Signature verification: With message m and signature , we recalculate $m' = h(m; p_3) = 0010001100001101$ and $m'_{3\ NAF} = (0; 0; 0; 1; 0; 0; 0; 0; 3; 0; 0; 0; 1; 0; 0; 0; 3)$. Since $m'_{3\ NAF}$ passes all restrictions, we do not reject the signature. Repeating steps 4-6, we recover $m'' = (4; 9; 4; 6)$ and $(B_1; B_0) = (4; 9)$. Finally, we calculate $y' = g(c_k^{4-}((\ _m)_3; r); c_k^{9-}((\ _m)_2; r); c_k^{4-}((\ _m)_1; r); c_k^{6-}((\ _m)_0; r); c_k^{4-}((\ _c)_1; r); c_k^{9-}((\ _c)_0; r)) = g(c_k^{15-}((x_m)_3; r); c_k^{15-}((x_m)_2; r); c_k^{15-}((x_m)_1; r); c_k^{15-}((x_m)_0; r); c_k^{15-}((x_c)_1; r); c_k^{15-}((x_c)_0; r))$. Because we got $y' = y$, we accept the signature. The verification cost of $4 + 9 + 4 + 6 + 4 + 9 = 36$ calls to c_k is, thus, 20% lower than the average of $6 \cdot 15/2 = 45$ calls to c_k, even though it produces the same signature size and takes roughly the same signature generation time as W-OTS+ for $w = 4$. For larger n, where L_2 would be far smaller than L_1 and the $n + 1$ signed digits would be proportionately closer to the n bits in m, the difference in verification time would be even more significant for equivalent signature sizes and key generation times.

F. Correctness of the algorithm

Since the proposed algorithm consists basically in an encoding scheme for W-OTS+, its correctness derives directly from the correctness of W-OTS+. In summary: the hash chains built during the key generation process are the same ones built when combining the signing and verification procedures; therefore, the public key reconstructed from the signature matches the genuine public key when the signature is authentic.

More formally, given the message m and a nonce p chosen by the signer, both the signer and the verifier arrive at the same $m'' = (D''_{L_1-1}; D''_{L_1-2}; \ldots; D''_1; D''_0)$. After all, both of them perform exactly the same calculations from the message and the nonce to get it). Analogously, from the same m'', signer and verifier arrive at the same checksum blocks $(B_{L_2-1}; \ldots; B_0)$.

Recall that the public key in the signature is given by

$$y = g((y_m)_{L_1-1} || \ldots || (y_m)_0 || (y_c)_{L_2-1} || \ldots || (y_c)_0)$$

where

$$\begin{cases} (y_m)_i = c_k^{len-1}((x_m)_i; r) \\ (y_c)_i = c_k^{2^w-1}((x_c)_i; r) \end{cases}$$

Also, recall that the signer performs the calculation

$$\begin{cases} (\ _m)_i = c_k^{len-1-D''_i}((x_m)_i; r) \\ (\ _c)_i = c_k^{2^w-1-B_i}((x_c)_i; r) \end{cases}$$

and the verifier calculates $y' = g((y_m)'_{L_1-1} || \ldots || (y_m)'_0 || (y_c)'_{L_2-1} || \ldots || (y_c)'_0)$ where

$$\begin{cases} (y_m)'_i = c_k^{D''_i}((\ _m)_i; r) \\ (y_c)'_i = c_k^{B_i}((\ _c)_i; r) \end{cases}$$

Since both the signer and the verifier have the same values of D''_i and B_i, one can substitute the signer's equations into the verifier's and get

$$\begin{aligned} (y_m)'_i &= c_k^{D''_i}((\ _m)_i; r) \\ &= c_k^{D''_i}(c_k^{len-1-D''_i}((x_m)_i); r) \\ &= c_k^{D''_i+len-1-D''_i}((x_m)_i; r) \\ &= c_k^{len-1}((x_m)_i; r) \\ &= (y_m)_i \end{aligned}$$

In the same way,

$$\begin{aligned} (y_c)'_i &= c_k^{B_i}((\ _c)_i; r) \\ &= c_k^{B_i}(f^{2^w-1-B_i}((x_c)_i); r) \\ &= c_k^{B_i+2^w-1-B_i}((x_c)_i); r) \\ &= c_k^{2^w-1}((x_c)_i; r) \\ &= (y_c)_i \end{aligned}$$

Thus, $(y_m)'_i = (y_m)_i$ and $(y_c)'_i = (y_c)_i$ and, as a result, $y' = y$.

V. Security Analysis

The difference between z-OTS and W-OTS+ resides in how the signer maps the same message to the number of calls to c_k^i in each hash chain. Namely, while W-OTS+ divides a message m into w-bit blocks and maps them directly to the hash chains, z-OTS employs the z-NAF form to first convert a message m into m'', then map m'' to the hash chains. Therefore, since both W-OTS+ and z-OTS use the same hash chain structure with checksum for signatures, the proof of security of W-OTS+ from [7] also applies to z-OTS. More precisely, assuming that attackers cannot forge W-OTS+ signatures for any arbitrary binary message, such resistance against forgery also applies to the particular case of the encoded message m''. What remains to be proven, thus, is that the mapping scheme of z-OTS from m to m'' is secure, i.e., that it is computationally infeasible to find two distinct messages m and m_2 that are mapped to the same m''.

For the following discussion, we refer the reader to Figure 2, which depicts the four steps involved in converting m into m''.

Step 1: In the first step, $m' = h(m||p)$, where message m is combined with a nonce p via randomized hashing. As long as a second pre-image resistant hash function is adopted and an unpredictable p is picked by the signer, attackers should be unable to find a pair $(m_2; p_2)$ such that $m_2 \neq m$ and $h(m||p) = h(m_2||p_2)$, except for a negligible probability.

Step 2: In the second step, m' is converted to the z-NAF form. From the z-NAF Property 1, the z-NAF representation exists and is unique for all integers. This ensures that there is no $m'_2 \neq m'$ such that $m'_{NAF} = (m'_2)_{NAF}$ in this step.

Steps 3 and 4: In the third step, the defined restrictions ensure that we have an injective function in the fourth step, so once again no collisions exist. Specifically, from the definition of the z-NAF encoding, any two non-zero digits are necessarily separated by at least $z - 1$ zeros. This fact, together with Restriction 1, ensures that there are always $z - 1$ zero digits

on the left of each non-zero digit in the resulting m'_{NAF} to be removed in Step 4; the only possible exception is the leftmost non-zero digit, from which up to $z-1$ zeros are removed. Therefore the removal operation is a function over the domain of all m'_{NAF} that passes validation in Step 3.

Now, let u_i denote the number of zeros on the left of each non-zero digit N_i in m'_{NAF}. Per Restriction 1, we have $u_i >= z-1$ for every N_i, except for the leftmost of those non-zero digits, N_{l-1}. Then, for each sequence $0^{u_i}N_i$ in m'_{NAF}, the operation where zeros are removed in step 4 would be $0^{u_i}N_i \to 0^{u_i-(z-1)}N_i = D'_i$. Different sequences would then produce distinct D'_i, meaning that the resulting $m'_{grouped}$ would be different.

Nevertheless, one might believe that, since the leftmost non-zero digit N_{l-1} in m'_{NAF} might have a variable number of zeros removed from its left, one could take advantage of this variability to find two different m'_{NAF} having the same $m'_{grouped}$. This is not the case, however, because the number of digits in m'_{NAF} with padding is fixed at $n+1$. Consequently, a different number of zeros on the left of N_{l-1} implies that at least one zero sequence somewhere else has a different length. The resulting $m'_{grouped}$ would, thus, differ at those points.

These observations imply that the combination of Step 3 and 4 is injective: there is no $m'_{NAF_2} \neq m'_{NAF}$ satisfying $m'_{grouped_2} = m'_{grouped}$ as a result of this step.

Step 5: Finally, in the fifth step, each block $0^{t_i}N_i$ of $m'_{grouped}$ is mapped into a non-negative integer digit D''_i in the interval $[0; (1+t_{max})2^{z-1})$ according to the map

$$D''_i = \begin{cases} t_i 2^{z-1} + (N_i - 1)/2 & \text{if } N_i > 0 \text{ and } i < l \\ t_i 2^{z-1} + (2^z + N_i - 1)/2 & \text{if } N_i < 0 \text{ and } i < l \\ 0 & \text{if } i \geq l \end{cases}$$

In what follows, we prove that this map is injective for the subset of all $m_{grouped}$ generated by the proposed algorithm. In other words, assume $(m'_1)_{grouped} = ((D'_1)_{l_1-1}; \ldots; (D'_1)_0)$ generated from m_1, where $(D'_1)_i = 0^{(t_1)_i}(N_1)_i$, and $(m'_2)_{grouped} = ((D'_2)_{l_2-1}; \ldots; (D'_2)_0)$ generated from m_2, where $(D'_2)_i = 0^{(t_2)_i}(N_2)_i$; in this scenario, if $(m'_1)_{grouped} \neq (m'_2)_{grouped}$ then $m''_1 \neq m''_2$ (or, equivalently, if $(D''_1)_i = (D''_2)_i$, then $(D'_1)_i = (D'_2)_i$ for all i). To prove that, we discuss two cases separately: (1) $l_1 = l_2$; and (2) $l_1 \neq l_2$.

1) $l_1 = l_2 = l$: in that case, when $i \geq l$, the mapping from Step 5 is such that $(D''_1)_i = (D''_2)_i = 0$. Hence, we only need to evaluate the case where $i < l$, showing that $(D''_1)_i = (D''_2)_i$ whenever $(D'_1)_i = (D'_2)_i$.
From the z-NAF signed-digit definition,

$$|N_i| < 2^{z-1}$$

Then, if $N_i > 0$ we have $0 < N_i < 2^{z-1} \Rightarrow 1 < N_i - 1 < 2^{z-1} - 1 \Rightarrow 1/2 < (N_i-1)/2 < 2^{z-2} - 1/2$; since N_i is odd, $(N_i-1)/2$ will be an integer, so the inequality can be re-written as $0 \leq (N_i-1)/2 < 2^{z-2}$. Similarly, if $N_i < 0$ we have $-2^{z-1} < N_i < 0 \Rightarrow -2^z - 2^{z-1} < -1 + N_i < -1 \Rightarrow -2^{z-2} - 1/2 < (2^z+N_i-1)/2 < 2^{z-1} - 1/2$; once again, since N_i is odd, $(2^z+N_i-1)/2$ will be an integer and the inequality can be rewritten as $-2^{z-2} \leq (2^z+N_i-1)/2 < 2^{z-1}$. Hence, in summary we have:

$$\begin{cases} 0 \leq (N_i-1)/2 < 2^{z-2} & \text{if } N_i > 0 \\ -2^{z-2} \leq (2^z+N_i-1)/2 < 2^{z-1} & \text{if } N_i < 0 \end{cases}$$

Now, suppose that $(D''_1)_i = (D''_2)_i$. If $(N_1)_i; (N_2)_i > 0$, then

$$\begin{aligned} (t_1)_i 2^{z-1} + \frac{(N_1)_i - 1}{2} &= (t_2)_i 2^{z-1} + \frac{(N_2)_i - 1}{2} \\ ((t_1)_i - (t_2)_i)2^{z-1} &= \frac{(N_2)_i - (N_1)_i}{2} \\ ((t_1)_i - (t_2)_i)2^z &= (N_2)_i - (N_1)_i \end{aligned}$$

Since $0 < (N_1)_i; (N_2)_i < 2^{z-1}$, we have that

$$\begin{aligned} |(N_2)_i - (N_1)_i| < 2^{z-1} &\Rightarrow |(t_1)_i - (t_2)_i|2^{z-1} < 2^{z-1} \\ &\Rightarrow (t_1)_i = (t_2)_i \\ &\Rightarrow (N_2)_i - (N_1)_i = 0 \\ &\Rightarrow (N_2)_i = (N_2)_i \end{aligned}$$

Since these equalities are valid for all i, this means that $D'_1 = D'_2$.
Analogously, if $(D''_1)_i = (D''_2)_i$ and $(N_1)_i; (N_2)_i < 0$, then

$$\begin{aligned} (t_1)_i 2^{z-1} + \frac{2^z + (N_1)_i - 1}{2} &= (t_2)_i 2^{z-1} + \frac{2^z + (N_2)_i - 1}{2} \\ ((t_1)_i - (t_2)_i)2^{z-1} &= \frac{(N_2)_i - (N_1)_i}{2} \\ ((t_1)_i - (t_2)_i)2^z &= (N_2)_i - (N_1)_i \end{aligned}$$

Since $-2^{z-1} < (N_1)_i; (N_2)_i < 0$, then

$$\begin{aligned} |(N_2)_i - (N_1)_i| < 2^{z-1} &\Rightarrow |(t_1)_i - (t_2)_i|2^{z-1} < 2^{z-1} \\ &\Rightarrow (t_1)_i = (t_2)_i \\ &\Rightarrow (N_2)_i - (N_1)_i = 0 \\ &\Rightarrow (N_2)_i = (N_2)_i \end{aligned}$$

Once again, this means that $D'_1 = D'_2$.
Now, if $(N_1)_i > 0$ and $(N_2)_i < 0$ for some i, then we cannot have $(D''_1)_i = (D''_2)_i$. The reason is that, to satisfy this latter equality, we would need to have:

$$\begin{aligned} (t_1)_i 2^{z-1} + \frac{(N_1)_i - 1}{2} &= (t_2)_i 2^{z-1} + \frac{2^z + (N_2)_i - 1}{2} \\ ((t_1)_i - (t_2)_i)2^{z-1} &= \frac{2^z + (N_2)_i - (N_1)_i}{2} \\ ((t_1)_i - (t_2)_i)2^z &= 2^z + (N_2)_i - (N_1)_i \end{aligned}$$

Since $(N_1)_i > 0$ and $(N_2)_i < 0$, $2^z + (N_2)_i - (N_1)_i < 2^z$, so

$$\begin{aligned} ((t_1)_i - (t_2)_i)2^z < 2^z &\Rightarrow (t_1)_i = (t_2)_i \\ 0 = 2^z + (N_2)_i - (N_1)_i &\Rightarrow (N_1)_i - (N_2)_i = 2^z \end{aligned}$$

This case is impossible, though, because, $|(N_1)_i - (N_2)_i| < 2^z$, since, by definition, $|N_i| < 2^{z-1}$.
Finally, the case where $(N_1)_i < 0$ and $(N_2)_i > 0$ can be proven analogously to the case where $(N_1)_i > 0$ and $(N_2)_i < 0$: once again, the fact that $D'_1 \neq D'_2$ in this scenario leads to $(D''_1)_i \neq (D''_2)_i$.

Therefore, we have shown that, for $i < l$, if $(D_1^{00})_i = (D_2^{00})_i$ then $(D_1^0)_i = (D_2^0)_i$.

2) $l_1 \neq l_2$: assume, without loss of generality, that $l_1 > l_2$. Then, when $i \geq l_1$, we also have $(D_1^{00})_i = (D_2^{00})_i = 0$; in addition, when $i < l_2$, the same arguments given for the case where $l_1 = l_2$ apply, so $(D_1^{00})_i = (D_2^{00})_i$ implies $(D_1^0)_i = (D_2^0)_i$. Finally, in the region where $l_2 \leq i < l_1$, we have $(D_2^{00})_i = 0$ and, from the map, $(D_1^{00})_i = (D_2^{00})_i = 0$ only when $(D_1^0)_i = 1$. At first sight, thus, $(m_1^0)_{grouped}$ and $(m_2^0)_{grouped}$ would be identical when $l_2 \leq i < l_1$ if $(D_1^0)_i = 1$ for all i in this range, violating the injectivity property. This is not the case, however, because $(D_1^0)_i = 1$ for all $l_2 \leq i < l_1$ implies $(D_1^0)_j \neq (D_2^0)_j$ for some $j < l_2$. To see that, assume $(D_1^0)_i = (D_2^0)_i$ for all $i < l_2$. Note that the length of m_{NAF}^0 (and, consequently, the length of m) is encoded in $m_{grouped}^0$; specifically, the zero-runs on the left of each N_i (after removing the mandatory zeros) are encoded in the t_i portion of D_i^0. Now, suppose that $(m_1^0)_{grouped}$ was generated from a valid $(m_1^0)_{NAF}$ with $n+1$ signed digits, and that we try build $(m_2^0)_{grouped}$ from it by discarding its digits in the $l_2 \leq i < l_1$ range; in that case, the hypothetical $(m_2^0)_{NAF}$ would have less than $n+1$ signed digits. Conversely, suppose $(m_2^0)_{grouped}$ was generated from a valid $(m_2^0)_{NAF}$ with $n+1$ signed digits, and that we try to construct $(m_1^0)_{grouped}$ by appending $0^{t_i}N_i = 1$ to the left of $(m_2^0)_{grouped}$; then the hypothetical $(m_1^0)_{NAF}$ would have more than $n+1$ signed digits. In either case, given that a message m of length n is represented in the z-NAF form by exactly $n+1$ signed digits, those hypothetical $m_{grouped}^0$ can never occur. This means that, to make $D_1^{00} = D_2^{00}$ in the $l_2 \leq i < l_1$ range while keeping $D_1^0 \neq D_2^0$ in that range, we would also need to change some digits in the $i < l_2$ range to compensate for the different message length. However, this modification makes $(m_1^0)_{grouped} \neq (m_2^0)_{grouped}$. Therefore, we can conclude that, if $l_1 \neq l_2$, then $(m_1^0)_{grouped} \neq (m_2^0)_{grouped}$.

From these observations, we can conclude that it is computationally infeasible to find distinct messages m and m_2 resulting in the same m^{00}: unless an attacker can find a collision for the hash function h employed in Step 1, the fact that all subsequent steps are injective prevent collisions from occurring as a result of their application. Hence, given a collision-resistant h, signatures generated with z-OTS display the same security properties observed for W-OTS+.

VI. Performance Analysis

In this section, we evaluate the signature size and processing time of the z-OTS, both in absolute terms and in comparison with W-OTS+.

First, we note that W-OTS+ supports an exponential trade-off between signature size and processing time, which is regulated by the Winternitz parameter w. More precisely, the signature size is given by $|\sigma|_{W\text{-}OTS+} = \frac{n}{w}$, whereas the total number of chaining function operations performed during key generation is given by $\#(c)_{W\text{-}OTS+} = \frac{n}{w}(2^w - 1)$. These equations are valid when ignoring the checksum, which normally represents less than 5% of the total signature size and processing overhead according to our benchmarks. However, because w can only be positive integers, both the signature size and processing time change in large steps when w changes, leaving large gaps in between. Even though this ability to fine-tune processing time and signature size can be seen as an advantage of z-OTS when compared with W-OTS+ and similar hash-based schemes, it hinders direct comparisons between them. Therefore, aiming to avoid unfairness, we use an interpolated model for W-OTS+, filling in the signature size and processing time gaps as if they were possible in practice with non-integer values for w. For that purpose, we note that the equations for $|\sigma|_{W\text{-}OTS+}$ and $\#(c)_{W\text{-}OTS+}$ can be combined and $\#(c)_{W\text{-}OTS+}$ can be written as a function of $|\sigma|_{W\text{-}OTS+}$ as follows:

$$\#(c)_{W\text{-}OTS} = |\sigma|_{W\text{-}OTS+}\left(2^{n/|\sigma|_{W\text{-}OTS+}} - 1\right) \quad (1)$$

Equation 1 can then be used to compare the trade-off between signature size and processing costs of W-OTS+, as well as with other OTS schemes whose settings do not necessarily match the integer points defined by the Winternitz parameter. Figure 3 illustrates the interpolation, with the integer values of w highlighted (up to $w = 8$).

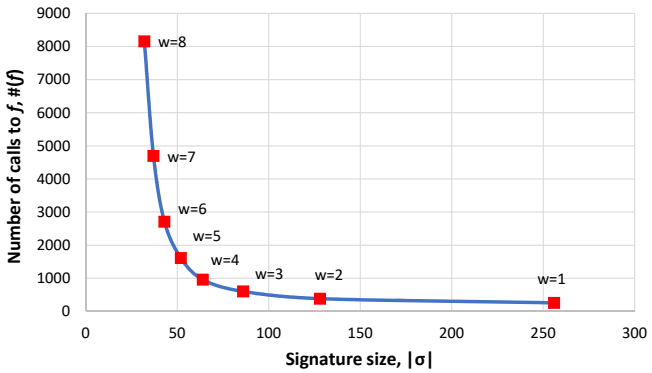


Fig. 3. Performance of W-OTS+: number of one-way function evaluations as a function of signature size. The small squares are the curve points for $w = 1; 2; \cdots; 8$, while the line is the interpolation of those points over $\mathbb{R}$ (see Equation 1).

Another important aspect of z-OTS's performance analysis is the choice of z, since it controls signature size and performance trade-offs in z-OTS analogously to w in W-OTS. Indeed, z affects key generation, signature generation, and signature verification, besides determining the choice of L_1, t_{max}, and w. Specifically, a large value of z results in a small signature size, but the key generation, signature generation and signature verification algorithms should take longer to execute. Conversely, choosing a small z speeds up the signature algorithms, but the resulting signatures grow larger.

A. Signature Size

The signature size of z-OTS, in terms of the number of b-bit values in the signature, is given by

$$|\sigma_{z\text{-}OTS}| = L_1 + L_2 + |p|$$

where L_1 is the number of hash chains used to sign m'', L_2 the number of hash chains used to sign the checksum, and $|p|$ the length of the nonce p. Since $|p|$ and L_2 are, like in W-OTS+, much smaller than L_1 (generally less than 5% of the total signature size), we can assume the approximation

$$|\sigma_{z\text{-}OTS}| \approx L_1$$

Note that, from Properties 2 and 3 of z-NAF, encoding a n-bit integer results $(n+1)/(z+1)$ non-zero z-NAF digits on average. Therefore, when the goal is to reduce the signature by reducing L_1, one should do so without over-limiting the number of non-zero digits allowed in the z-NAF form. Otherwise, this could result in too many rejections of p from the Restriction 2. Even considering this limitation, a suitable choice of L_1 together with t_{max} allows for many different trade-offs between signature size and processing overhead, as shown in the examples given in Section VII.

B. Processing Time

We analyze the processing time for the three algorithms comprised in z-OTS: key generation, signature generation and signature verification. Each of these processing times can be expressed in terms of the number of evaluations of c_k^i, g and h, the number of PRF calls and the number of z-NAF conversions.

Normally, the key generation and signature verification time is dominated by the number of evaluations of c_k^i, while the signature generation is dominated by the number of evaluations of c_k^i, the number of evaluations of h (when calculating $m' = h(m||p)$, and the number of z-NAF conversions. Note that all three algorithms are sensitive to the value of z: a larger z makes all three algorithms slower, whereas a small z has the opposite effect. Each algorithm is analyzed separately in what follows, whereas suggested parameters are given in Section VII.

Key generation: For a given private key x, there are $L_1 + L_2$ calls to the PRF function to generate all the $(x_m)_i$ and $(x_c)_i$. Then, the function c_k^i is applied $len - 1 = (1 + t_{max})2^{z-1} - 1$ times for each $(x_m)_i$ to generate the corresponding $(y_m)_i$, and $2^w - 1$ times for each $(x_c)_i$ to generate the corresponding $(y_c)_i$; the total is, thus, $L_1((1 + t_{max})2^{z-1} - 1) + L_2(2^w - 1)$ evaluations of c_k^i. Finally, g is called once on $(y_m)_{L_1-1}||\ldots||(y_m)_0||(y_c)_{L_2-1}||\ldots||(y_c)_0$ to generate y. Hence, the key generation time increases exponentially with z and linearly with L_1 and t_{max}.

Signature generation: For randomly generated nonces p_i, the signer needs to calculate $m' = h(m||p_i)$ and m'_{NAF}. Consequently, one needs to evaluate h and convert m' to the z-NAF form $1/Prob$ times on average, where $Prob$ is the probability of finding p_i whose corresponding m'_{NAF} satisfies the three restrictions from Step 3 in the signature generation algorithm. Then, after finding a suitable p_i and calculating m'', the hash chains are calculated by evaluating c_k^i accordingly.

Like in key generation, signature generation time increases exponentially with z. The parameters L_1 and t_{max}, however, do not have a monotonic effect in signature generation. For example, large values of L_1 and t_{max} increase the probability of finding a suitable p_i and reduce the number of evaluations of h, as well as the number of z-NAF conversions required. At the same time, however, the number and length of hash chains that need to be calculated are increased. Hence, the larger L_1 and t_{max}, the higher the number of evaluations of c_k^i.

Signature verification: For this algorithm, the verifier needs to perform one evaluation of h and one z-NAF conversion to calculate m'' from the message and p provided in the signature. Then, the verifier "completes" the hash chains by evaluating c_k^i repeatedly, and recovers the end-of-chain values y_i'. Finally, a single evaluation of g is performed to recover y', which is compared against the public key y to validate the signature.

Similarly to the key and signature generation algorithms, signature verification time increases exponentially with z, but the behavior with different values of L_1 and t_{max} is once again not monotonic. Without applying any restriction from Step 3 of signature generation, the average m'_{NAF} would have non-zero digit density of $1/(z+1)$ and a zero digit density of $z/(z+1)$. So, after removing the mandatory $z - 1$ zeros on the left of each non-zero digit, the resulting zero density is roughly $z/(z+1) \cdot (z-1)/(z+1) = 1/(z+1)$, which is the same density as the number of non-zero digits. This means that the number of zero and non-zero digits in $m'_{grouped}$ is the same on average. However, the restrictions enforced by the choice of L_1 and t_{max} may affect this distribution on m'_{NAF}.

Since L_1 limits the number of non-zero digits in m_{NAF}, a value of L_1 too small would select only those m_{NAF} with more zeros than non-zero digits. Due to the map defined in the signature generation algorithm, though, each non-zero digit increases the total number of evaluations of c_k^i in the verification algorithm by some value in $[0; 2^{z-1})$, while each zero digit increases it by exactly 2^{z-1}. Therefore, making L_1 too small reduces the number of hash chains that need to be calculated, but also increases the individual digits in m''. This potentially increases the overall number of evaluations of c_k^i, making verification slower. Conversely, choosing L_1 too large removes this digit imbalance, but increases the number of hash chains and slows down the verification process.

The parameter t_{max} has yet other effects: although choosing a small t_{max} decreases the number of evaluations of c_k^i during signature verification, picking a large t_{max} does not significantly increase it. This happens because, even when very long runs of zeros are allowed in m'_{NAF} (after discarding the $z - 1$ zeros), the probability of occurrence of such large zero runs decreases exponentially: for each additional zero digit, it decreases by a rate of $P_Z \approx 0.5$, where P_Z is the (empirically calculated) probability of a z-NAF digit not being zero At the same time, the total number of hashes only increases linearly, at a step of 2^{z-1}. This means that the number of calls to c_k^i during signature verification is bounded for any choice of t_{max}. For that reason, aiming to optimize the signature size and verification time, one could use smaller values of L_1 and

compensate the increased signature generation time with larger values of t_{max}, which does not increase the verification time by much.

Besides reducing the total chain length len, there is another reason why a small t_{max} can make verification faster: while L_1 limits the number of non-zero digits in m'_{NAF}, t_{max} tends to limit the number of zero digits. Hence, a small t_{max} decreases the number of zero digits relative to the amount of non-zero digits in m'_{NAF}. Given the aforementioned effects of zero and non-zero digits balance, this reduces the digits in m'' and requires fewer evaluations of c_k^i during signature verification.

Finally, differently from W-OTS+, note that the digits in m'' are skewed towards zero, since longer zero runs on the left of non-zero digits are less probable. The result is that the number of evaluations of c_k^i during signature verification is significantly lower than half of the total evaluations of c_k^i in the key generation algorithm. This is a core reason why z-OTS provides faster verification at the cost of a slower signature generation when compared with W-OTS+.

C. Signature size vs. processing time trade-offs

In z-OTS, the trade-off between signature size and processing time depends on the choice of z, L_1, and t_{max}. The underlying Winternitz parameter w can then be picked such that the checksum hash chains have roughly the same length as the message hash chains.

The parameter with the strongest influence over z-OTS performance is the z. Recall that the signature size is determined basically by L_1, which should be proportional to z: according to the analysis in Section VI-B, in general the value of L_1 should be somewhere around the average number of non-zero digits in m'_{NAF}, given by $(n+1)/(z+1)$. The total chain length len, however, is given by $len = (1+t_{max})2^{z-1} - 1$ and, thus, grows exponentially with z. In this way, the total number of evaluations of c_k^i grows exponentially with z, but the signature size only decreases linearly when z increases. This behavior is similar to the effect of the w parameter in W-OTS+, where the signature size is given by $\lceil n/w \rceil e$ and the total chain length is given by $2^w - 1$. From these observations, we also note that it is possible to obtain similar signature sizes in z-OTS and W-OTS+ by picking $z = w - 1$.

After picking a suitable z, the parameters L_1 and t_{max} can be used to fine-tune the trade-off between signature size and processing time. In general, one should pick small values of L_1 and t_{max}, reducing signature size and key generation costs, while still making signature verification significantly faster than signature generation. Nevertheless, as discussed in Section VI-B, choosing a smaller value of L_1 for a slightly larger value of t_{max} may be useful when the goal is to reduce the signature size without increasing verification costs by much.

With the formulas for signature size and key generation time for both z-OTS and W-OTS+, we can compare the trade-offs offered in both algorithms. For this purpose, assume $L_1 = \lceil (n+1)/(z+1) \rceil$ and the usual hash length of $n = 256$. In this scenario, and as discussed in Section VI-C, we can obtain signatures of similar sizes in z-OTS and in W-OTS+ by picking an arbitrary w in the latter and $z = w - 1$ in the former.

Therefore, substituting $w = z + 1$ into the formulas for W-OTS+, the total number of evaluations of c_k^i during key generation can be computed as:

$$\begin{cases} \#(c)_{z\text{-OTS}} = \left\lfloor \frac{n+1}{z+1} \right\rfloor ((1+t_{max})2^{z-1} - 1) \\ \#(c)_{\text{W-OTS}} = \frac{n}{z+1}(2^{z+1} - 1) \end{cases}$$

As z tends to infinity, the ratio $\#(c_k^i)_{z\text{-OTS}} / \#(c_k^i)_{\text{W-OTS+}}$ would be evaluated as

$$\lim_{z \to 1} \frac{\#(c_k^i)_{z\text{-OTS}}}{\#(c_k^i)_{\text{W-OTS+}}} = \lim_{z \to 1} \frac{\left\lfloor \frac{n+1}{z+1} \right\rfloor ((1+t_{max})2^{z-1} - 1)}{\frac{n}{z+1}(2^{z+1} - 1)} = \frac{1+t_{max}}{4}$$

This shows that, by choosing $t_{max} < 3$ for a large z, z-OTS should have better asymptotic behavior than W-OTS+ in terms of c_k^i evaluations required for key generation (or equivalently, for signature generation combined with signature verification). Indeed, for a large z, the fixed W-OTS+ can take four times more calls to c_k^i than z-OTS.

D. Storage Requirements

The storage requirements of the z-OTS are similar to that of W-OTS+. Namely, what is stored are the public and private keys for the signer, and the public key for the verifier. In the z-OTS scheme, the private key is b-bit long, while the public key has $(\max(len - 1; 2^w - 1) + 1)b$ bits. Hence, the signer needs to store $(\max(len - 1; 2^w - 1) + 2)b$ bits while the verifier needs to store $(\max(len - 1; 2^w - 1) + 1)b$ bits.

VII. Experimental results

To test the z-OTS, a simulator was written in Python language. The source code, together with reproducible executions of our experiments, can be found at [30]. The security parameter b and the message length n were both chosen as 256 bits. For simplicity, the simulations do not employ any parallelism. Nevertheless, we note that an actual implementation could perform in parallel (1) the computation of individual hash chains and, (2) for z-OTS, the calls to h and z-NAF conversions when searching for a suitable p_i during the signature generation process.

The simulations were run for many different values of z and involved two sets of parameters: a basic set and a set intended to further improve signature verification speed. The results, in terms of the number of hash function evaluations taken by each algorithm, were recorded and compared to those of W-OTS+, in order to estimate the practical benefits and limitations of z-OTS.

For the experiments, we used the W-OTS+ interpolated model (see Figure 3) to compare the number of calls to c_k^i between W-OTS+ and z-OTS when the latter's signature size did not match the specific sizes supported by W-OTS+ for any integer w.

Since W-OTS+ with some arbitrary w leads to a signature size similar to the one obtained with z-OTS when $z = w - 1$, we restricted our simulations to values of z such that $2 \leq z \leq 8$. This choice is compatible with the range of w commonly suggested in the literature for W-OTS+ and similar schemes [31], and includes the most commonly found $w = 4$ parameter [32]. For the other two parameters, L_1 and t_{max}, we start by picking values for which the signature size of z-OTS matches those of W-OTS+, and also the hash chain lengths would be as close as possible without too much processing overhead. These choices led to the basic parameter set shown in Table IIIa. In addition, we built an optimized parameter set aimed at lower verification costs, following the recommendations discussed in Section VI-B and choosing smaller values of L_1 while increasing the values of t_{max}. This second set of verification-optimized parameters is presented in Table IIIb.

TABLE II
z-OTS PARAMETERS (z; L_1; t_{max}; w) LEADING TO SIGNATURE SIZES SIMILAR TO W-OTS+ WITH WINTERNITZ PARAMETER w.

z	L_1	t_{max}	w
2	86	5	3
3	64	5	4
4	52	4	5
5	43	4	6
6	37	4	7
7	32	4	8
8	29	3	9

(a) Basic

z	L_1	t_{max}	w
2	74	8	3
3	56	8	4
4	45	8	5
5	38	7	6
6	32	10	7
7	28	11	8
8	25	9	9

(b) Verification-optimized

The results for key generation as well as the average results for signing and verifying 1000 messages are summarized in Tables III and IV for, respectively, the basic and verification-optimized parameter sets from Table II. In addition, Figure 4 depicts both results for easy comparison among them.

TABLE III
PERFORMANCE OF z-OTS FOR BASIC PARAMETER SET (TABLE IIIA).

	Calls	Calls to $c_k^{i\ z}$		
z	to h^y	KG	Sign	Ver
2	27	946	726 ± 8	220 ± 8
3	22	1472	1102 ± 17	370 ± 17
4	36	2028	1431 ± 31	597 ± 31
5	32	3397	2364 ± 55	1033 ± 55
6	21	5883	4104 ± 112	1779 ± 112
7	27	10208	6972 ± 195	3236 ± 195
8	45	14819	9566 ± 364	5253 ± 364

[y] Average calls to h performed until a suitable value of p_i is found; standard deviation was close to 100%. [z] Number of calls to c_k^i does not include calculation of the hash chains for checksums.

As expected from our theoretical analysis, the number of calls to c_k^i and the total machine cycles of all three algorithms grow exponentially with z, similarly to the behavior observed in W-OTS+ for the w parameter. However, the cost of signature verification is significantly lower than the cost of signature generation, besides grows more slowly, showing the asymmetric effect of z-OTS over those operations.

We note that the actual optimal parameters for the scheme depend on the implementation and machine cycles taken by the functions h, c_k^i, g and the z-NAF conversion algorithm.

TABLE IV
PERFORMANCE OF z-OTS FOR VERIFICATION-OPTIMIZED PARAMETER SET (TABLE IIIB).

	Calls	Calls to $c_k^{i\ z}$		
z	to h^y	KG	Sign	Ver
2	978	1258	1000 ± 6	258 ± 6
3	1509	1960	1515 ± 12	445 ± 12
4	2656	3195	2413 ± 22	782 ± 22
5	3608	4826	3453 ± 43	1373 ± 43
6	8740	11232	8630 ± 83	2602 ± 83
7	13460	21476	16637 ± 157	4839 ± 157
8	28140	31975	23013 ± 283	8962 ± 283

[y] Average calls to h performed until a suitable value of p_i is found; standard deviation was close to 100%. [z] Number of calls to c_k^i does not include calculation of the hash chains for checksums.

For example, if the cost of z-NAF conversion is competitive with the function c_k^i, one may choose more restrictive values for L_1 and t_{max} and, thus, get smaller signatures and faster verification.

We also observe that, compared to the initial parameter set, the optimized parameter set had smaller values of L_1, which caused the calls to h (and, thus, the number of z-NAF conversions) to increase significantly during signature generation. Besides, with the larger values of t_{max}, we also obtain a significant increase in the number of calls to c_k^i during the key and signature generation procedures. However, the growth in the signature verification time was only modest; this fact, combined with the smaller signature sizes obtained, shows the potential of z-OTS in scenarios where signature verification efficiency are desired.

A. Comparison with W-OTS+

Finally, we compare z-OTS with W-OTS+ in terms of average number of calls to c_k^i, using both parameter sets from Table II. The goal is to show the gains in signature verification speed, as well as the corresponding trade-offs for the key and signature generation procedures.

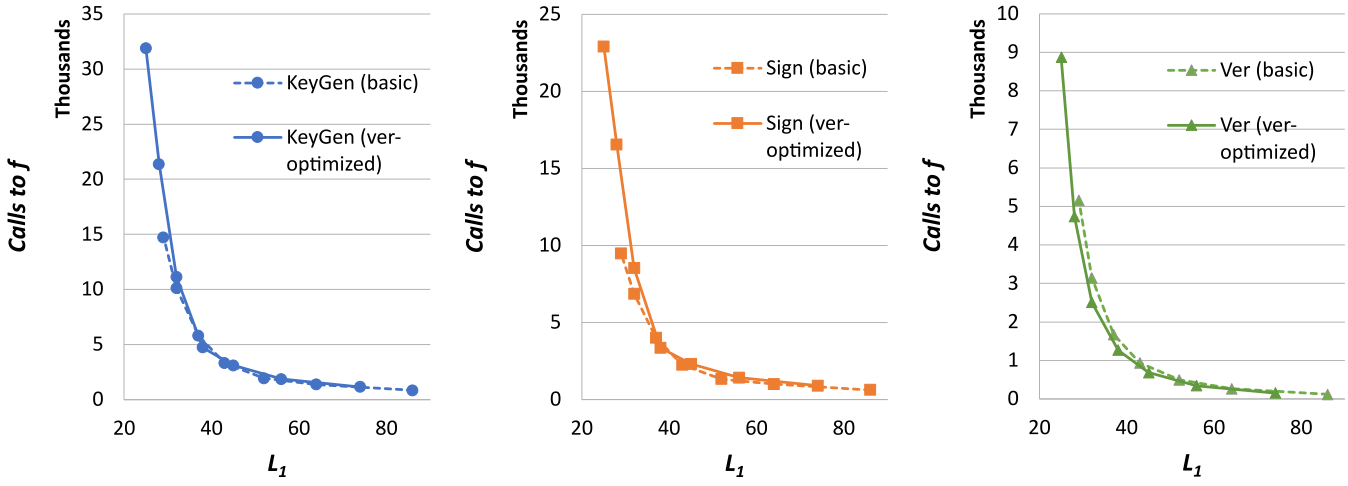
For this purpose, we consider the relative difference between the costs of z-OTS and W-OTS+ for each evaluated procedure, i.e., the following formula is used:

$$\text{Diff\_}\#(c_k^i) = \frac{\#(c_k^i)_{z\ \ OTS} \quad \#(c_k^i)_{w\ \ OTS+}}{\#(c_k^i)_{w\ \ OTS+}}$$

1) z-OTS vs. W-OTS: basic parameter set: For the parameter set given in Table IIIa, no interpolation over W-OTS+ results were needed because the signature sizes chosen in the z-OTS scheme match those in W-OTS+. Table V summarizes the comparison results.

As observed in Table V, z-OTS requires 12.7% to 60.1% more processing than W-OTS+ during the key generation procedure, whereas the overhead for signature generation can be up to 145.3% of W-OTS+. The signature verification time, on the other hand, is reduced somewhere from 20.1% to 25.7% depending on the average signature size.

2) z-OTS vs. W-OTS: verification-optimized parameters: In the case of the verification-optimized parameter set given in Table IIIb, we needed to interpolate the numbers for W-OTS+ because the corresponding signature sizes do not match those



(a) KeyGen (b) Signature (c) Verification

Fig. 4. Graphical representations of Tables III and IV, for easier observation of asymptotic behavior and trade-offs for basic and verification-optimized parameters. Signature size is roughly proportional to L_1 when $L_1 \gg L_2$, which is the case for usual values of n.

TABLE V
COMPARATIVE PERFORMANCE OF z-OTS AND W-OTS+ FOR z-OTS'S BASIC PARAMETER SET (TABLE IIIA).

		Diff_#(c_k^i) [z]		
z	Sig. Size[y]	KeyGen	Sign	Ver
2	86	60.1%	145.3%	-25.7%
3	64	53.3%	129.6%	-22.9%
4	52	32.9%	87.5%	-21.8%
5	43	29.6%	80.3%	-21.2%
6	37	32.5%	84.9%	-19.9%
7	32	25.1%	70.9%	-20.7%
8	29	12.7%	45.5%	-20.1%

[y]Signature size is given as a multiple of 256-bits and does not include the checksum part. [z] Number of calls to c_k^i does not include calculation of the hash chains for checksums.

obtained with z-OTS. Using this approach, Table VI shows the comparison between z-OTS and W-OTS+ processing time in this scenario.

TABLE VI
PROCESSING TIME (IN TERMS OF THE NUMBER OF CALLS TO c_k^i) OF z-OTS AND W-OTS+, USING z-OTS'S VERIFICATION-OPTIMIZED PARAMETER SET (TABLE IIIB).

		Diff_#(c_k^i) [z]		
z	Sig. Size[y]	KeyGen	Sign	Ver
2	74	70.0%	170.3%	-30.3%
3	56	53.7%	137.5%	-30.3%
4	45	40.4%	112.0%	-31.3%
5	38	20.2%	72.0%	-31.6%
6	32	37.6%	111.5%	-36.2%
7	28	35.9%	110.6%	-38.7%
8	25	5.8%	52.4%	-40.7%

[y]Signature size is given as a multiple of 256 bits and does not include the checksum part. [z] Number of calls to c_k^i does not include calculation of the hash chains for checksums.

When compared to W-OTS+, the overheads of z-OTS configured with such a verification-optimized parameter set are once again somewhat high for the key and signature generation procedures. Namely, key generation in z-OTS is 5.8% to 70.0% more costly than in W-OTS+, whereas the overhead for signature generation was observed to reach up to 170:3%. As expected, though, the costs for the signature verification procedure are considerably lower, with gains ranging from 30.3% to 40.6% when compared to W-OTS+. These experiments show the potential of z-OTS in trading key and signature generation time for a faster signature verification procedure, assuming a careful choice of the L_1 and t_{max} parameters.

We also point out that the verification-optimized parameters were chosen so that the key generation, signing and verification operations would take around a second or less to finish without employing any kind of parallelism. With parallelism and abundant computing resources for signature generation, one could use smaller values of L_1 (while increasing t_{max} as needed) to achieve even smaller verification times than those presented in this article. For example, our experiments show that more than 50% verification gains can be achieved with $L_1 = 24$ for $z = 8$.

VIII. CONCLUSION

With the growing interest and investments in quantum technology by governments and private corporations, large scale quantum computers are growing closer to reality every year. This context motivates the research for post-quantum cryptographic algorithms that can replace traditional schemes based on the (elliptic) discrete logarithm or number factorization, which would be vulnerable to attacks in a quantum scenario. Among the many existing proposals, hash-based digital signature schemes are quite promising. In special, this is due to their small public/private key pairs and to the fact that their security relies on well-known properties of hash functions.

In this work, we propose a novel OTS that uses the z-NAF encoding for pre-processing the message to be signed. The main benefit of the proposed z-OTS scheme is that, by skewing the frequency of zero and non-zero digits in the to-be-signed message, it can be used to speed up verification by slowing down key and signature generation. Compared to

W-OTS+, and for comparable signature sizes, the verification time in z-OTS can be up to 40.6% lower according to our experimental results. This is accomplished while retaining the main advantages of W-OTS+, in particular its short public keys and the ability to recover those keys from the corresponding signatures. Consequently, the proposed z-OTS scheme can be used as a drop-in replacement for W-OTS+ where signature verifications are done much more frequently than signature generations, such as in software distribution applications and other common use cases.

Acknowledgment. This work was supported by Intel and FAPESP (grants 2015/50520-6 and 13/25977-7), and in part by CNPq (research productivity grant 304643/2020-3).

REFERENCES

[1] D. J. Bernstein, J. Buchmann, and E. Dahmen, *Post-Quantum Cryptography*. Heidelberg, Deutschland: Springer, 2008.

[2] S. Anthony, "IBM will sell 50-qubit universal quantum computer 'in the next few years'," 2017, accessed: 2017-09-28. [Online]. Available: https://arstechnica.co.uk/gadgets/2017/03/ibm-q-50-qubit-quantum-computer

[3] R. L. Rivest, A. Shamir, and L. Adelman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, 1977.

[4] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Berlin, Heidelberg: Springer, 2003.

[5] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Rev.*, vol. 41(2), p. 303–332, 1999.

[6] A. Hülsing, J. Rijneveld, and P. Schwabe, "ARMed SPHINCS," in *Public-Key Cryptography–PKC 2016*. Berlin, Heidelberg: Springer, 2016, pp. 446–470.

[7] A. Hülsing, "W-OTS+ – shorter signatures for hash-based signature schemes," *Progress in Cryptology – AFRICACRYPT 2013*, vol. 7918 of Lecture Notes in Computer Science, p. 173–188, 2013.

[8] R. C. Merkle, "A certified digital signature," *Advances in Cryptology — CRYPTO' 89 Proceedings*, pp. 218–238, 1990.

[9] A. Hülsing, L. Rausch, and J. Buchmann, "Optimal parameters for XMSS-MT," in *International Conference on Availability, Reliability, and Security*. Berlin, Heidelberg: Springer, 2013, pp. 194–208.

[10] D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, and P. Schwabe, *SPHINCS+ - Submission to the NIST post-quantum project*, 2017 (accessed March 14, 2019). [Online]. Available: https://sphincs.org/data/sphincs+-specification.pdf

[11] NIST, "Post-quantum cryptography (PQC): Round 3 submissions." https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions, National Institute of Standards and Technology, Gaithersburg, MD, July 2020.

[12] R. Steinwandt and V. I. Villányi, "A one-time signature using run-length encoding," *Information Processing Letters*, vol. 108, no. 4, pp. 179–185, 2008.

[13] D. Roh, S. Jung, and D. Kwon, "Winternitz signature scheme using nonadjacent forms," *Security and Communication Networks*, vol. 2018, 2018.

[14] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *Proc. of the 21st Annual ACM Symposium on Theory of Computing (STOC'89)*. New York, NY, USA: Association for Computing Machinery, 1989, pp. 33—-43.

[15] R. M. Avanzi, "A note on the signed sliding window integer recoding and a left-to-right analogue," in *International Workshop on Selected Areas in Cryptography*. Berlin, Heidelberg: Springer, 2004, pp. 130–143.

[16] L. Lamport, "Constructing digital signatures from a one way function," SRI International Palo Alto, Menlo Park, CA, Tech. Rep., 1979.

[17] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS – A practical forward secure signature scheme based on minimal security assumptions," *Post-Quantum Cryptography*, vol. 7071, LNCS, p. 117–129, 2011.

[18] A. Perrig, "The BiBa one-time signature and broadcast authentication protocol," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. New York, NY: ACM, 2001, pp. 28–37.

[19] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *Australasian Conf. on Information Security and Privacy*. Berlin, Heidelberg: Springer, 2002, pp. 144–153.

[20] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, P. Schwabe, and Z. WilcoxO'Hearn, "SPHINCS: practical stateless hash-based signatures," *Advances in Cryptology – EUROCRYPT 2015*, vol. 9056, LNCS, p. 368–397, 2015.

[21] A. Miyaji, T. Ono, and H. Cohen, "Efficient elliptic curve exponentiation," in *International Conference on Information and Communications Security*. Berlin, Heidelberg: Springer, 1997, pp. 282–290.

[22] J. A. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," in *Annual International Cryptology Conference*. Berlin, Heidelberg: Springer, 1997, pp. 357–371.

[23] M. Joye and C. Tymen, "Compact encoding of non-adjacent forms with applications to elliptic curve cryptography," in *Public Key Cryptography*. Berlin, Heidelberg: Springer, 2001, pp. 353–364.

[24] C. Heuberger and D. Krenn, "Optimality of the width-w non-adjacent form: General characterisation and the case of imaginary quadratic bases," *J. Théor. Nombres Bordeaux*, vol. 25, no. 2, pp. 353–386, 2013.

[25] J. Muir and D. Stinson, "Minimality and other properties of the width-w nonadjacent form," *Mathematics of Computation*, vol. 75, no. 253, pp. 369–384, 2006.

[26] H. Cohen, "Analysis of the flexible window powering algorithm," https://www.math.u-bordeaux.fr/ hecohen/, 2001.

[27] NIST, *FIPS 180-4 – Secure Hash Standard (SHS)*, National Institute of Standards and Technology, Gaithersburg, MD, August 2015.

[28] ——, *FIPS 202 – SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, National Institute of Standards and Technology, Gaithersburg, MD, August 2015, DOI:10.6028/NIST.FIPS.202.

[29] ——, *SP 800-106 – Randomized Hashing for Digital Signatures*, National Institute of Standards and Technology, Gaithersburg, MD, February 2009.

[30] A. Zheng and M. Simplicio, "Simulator for the zOTS hash-based one-time signature scheme," Code Ocean: https://doi.org/10.24433/CO.3627530.v1, 2021.

[31] D. McGrew, M. Curcio, and S. Fluhrer, *RFC 8554: Leighton-Micali Hash-Based Signatures*, Internet Research Task Force (IRTF), Apr 2019.

[32] NIST, *Special Publication 800-208 – Recommendation for Stateful Hash-Based Signature Schemes*, National Institute of Standards and Technology, Gaithersburg, MD, Oct 2020.

Amos Zheng has a Master degree in Computer Engineering conferred by Universidade de Sao Paulo (USP) in 2019 and has special interest in topics related to computer architecture and cybersecurity.

Marcos A. Simplicio Jr. is an Associate Professor at Escola Politecnica, Universidade de Sao Paulo (USP). He has a Master degree (2006) in Engineering conferred by the Ecole Centrale des Arts et Manufactures (Ecole Centrale Paris), France, and received his Computer Engineering PhD from USP in 2010. His main research interests are (applied) cryptography and network security.