IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 27, 2019, accepted December 26, 2019, date of publication January 14, 2020, date of current version January 27, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966259

Novel One Time Signatures (NOTS): A Compact Post-Quantum Digital Signature Scheme

FURQAN SHAHID[1], IFTIKHAR AHMAD[2], MUHAMMAD IMRAN[3], AND MUHAMMAD SHOAIB[4]

[1]Department of Computer Science, COMSATS University Islamabad (CUI), Islamabad 45550, Pakistan.
[2]Faculty of Computer and Information Technology, King Abdulaziz University, Jeddah 11451, Saudi Arabia
[3]College of Applied Computer Science, King Saud University, Riyadh 11451, Saudi Arabia
[4]College of Computer and Information Sciences, King Saud University, Riyadh 11451, Saudi Arabia

Corresponding author: Muhammad Imran (dr.m.imran@ieee.org)

This work is partially supported by the Deanship of Scientific at King Saud University through research group project number RG-1439-036.

ABSTRACT The future of the hash based digital signature schemes appears to be very bright in the upcoming quantum era because of the quantum threats to the number theory based digital signature schemes. The Shor's algorithm is available to allow a sufficiently powerful quantum computer to break the building blocks of the number theory based signature schemes in a polynomial time. The hash based signature schemes being quite efficient and provably secure can fill in the gap effectively. However, a draw back of the hash based signature schemes is the larger key and signature sizes which can prove a barrier in their adoption by the space critical applications, like the blockchain. A hash based signature scheme is constructed using a one time signature (OTS) scheme. The underlying OTS scheme plays an important role in determining key and signature sizes of a hash based signature scheme. In this article, we have proposed a novel OTS scheme with minimized key and signature sizes as compared to all of the existing OTS schemes. Our proposed OTS scheme offers an 88% reduction in both key and signature sizes as compared to the popular Winternitz OTS scheme. Furthermore, our proposed OTS scheme offers an 84% and an 86% reductions in the signature and the key sizes respectively as compared to an existing compact variant of the WOTS scheme, i.e. WOTS$^+$.

INDEX TERMS Hash-based digital signatures, post-quantum cryptography, Blockchain, one-time signatures.

I. INTRODUCTION

The one way mathematical functions [1] act as the building blocks of the todays most popular digital signature schemes. These functions emerge as hard mathematical problems which provide a base for digital signatures and other cryptographic protocols. The three core hard mathematical problems currently being used by a wide range of cryptographic protocols include Integer Factorization (IF) problem, Discrete Logarithm Problem (DLP), and Elliptic Curve Discrete Logarithm Problem (ECDLP). The digital signature schemes constructed using these hard mathematical problems are commonly referred to as number theory based digital signature schemes which include, Rivest-Shamir-Adleman (RSA) signature scheme [2], El-Gamal signature scheme [3], and Elliptic Curve Digital Signature Algorithm (ECDSA) [4]. However, a sufficiently powerful quantum computer will be able to break these hard mathematical problems with the help of the Shor's algorithm [5]. The advancement trends of technology allow us to expect that a quantum computer being able to break these hard mathematical problems will be available after just a decade [6]. So what will be the future of the cryptographic protocols constructed over these hard mathematical problems? We are particularly concerned with the future of the digital signature schemes in the quantum era. Thankfully, quantum computers will not erase the digital signatures technology at all because of the availability of the other types of digital signature schemes which can defeat quantum attacks [7]. We refer those digital signature schemes to as post-quantum digital signature schemes. There are total five types of post-quantum digital signature schemes available to-date, including the lattice-based signature schemes, the hash-based signature schemes, the elliptic curve isogeny based signature schemes, the multivariate signature schemes,

The associate editor coordinating the review of this manuscript and approving it for publication was Jixiang Yang.

and the code-based signature schemes. Although all these types of digital signature schemes are not newer, rather some of them bear a fairly old history (like, the hash-based signature schemes), however, none of them could attract the practitioners at a large scale. The possible resistors to their wide-range adoption include, the low efficiency, the breakable security, and the difficult key management [7], [8].

The hash-based digital signature (HBS) schemes, being quite-efficient and provably-secure, appear as a dominant type of post quantum digital signature schemes [9]. The security of the HBS schemes has strongly been established against both classical and quantum attacks. Furthermore, HBS schemes are the most efficient type of schemes with key and signature creation times minimum among all type of digital signature schemes [10]. However, the major drawback of the HBS schemes is the larger signature and key sizes [11], [12].

An HBS scheme is a combination of two schemes; one is a core *One-Time-Signature (OTS)* or a *Few-Time-Signature (FTS)* scheme and second is a *hash tree* which maps a no. of OTS/FTS public keys to another single public key. Without covering an OTS/FTS scheme by a hash trees, the key management is a challenging task in an HBS scheme. The signature size depends purely on the core OTS/FTS used, whereas, the key size depends upon both of the core OTS/FTS scheme as well as the nature and size of the hash tree used by the scheme. The signature size in the very first OTS scheme, i.e. Lamport-Diffie (LD) OTS scheme [13], was impractically larger. However, the later OTS schemes, like Winternitz OTS scheme [14], reduced the signature size to a practical level. Even after the improvement, the signature sizes of the OTS schemes are larger than the classical schemes, which make them unfavorite for highly space sensitive applications, like the distributed financial ledgers (cryptocurrencies). In this article, we have proposed a novel OTS scheme "*NOTS*" with key and signature sizes minimum among all of the existing OTS schemes. *NOTS* offers an 88% reduction in both key and signature sizes as compared to the popular Winternitz OTS scheme.

Among the existing OTS/FTS schemes, WOTS and its variants [14]–[16] emerge as the most efficient type of OTS schemes, which offer minimum key and signature sizes. Furthermore, WOTS and its variants allow for computation of the OTS public key purely from the corresponding signatures, which is a valued characteristic of WOTS and its variants. Other type of OTS/FTS schemes (except WOTS and its variants) are not capable for allowing computation of the public key from the signatures unless a huge additional set of information is provided to the verifier. This additional set of information may either be as large as the original signatures (like in case of LD-OTS scheme [13]) or it may be exponentially larger than the original signatures (like, in case of HORS [17], HORST [18], and PORS [19]). Our proposed schemes (*NOTS*) is a WOTS like scheme in which the signatures are intelligent enough to allow the verifier for computation of the corresponding public key without any additional set of information. The intelligent signatures not only reduce the signature size but also make the scheme more convenient for the hash trees.

Our contribution:

1) We have proposed a novel OTS scheme *NOTS* with following valued features:
   a) *NOTS* offers an 88% reduction in both key and signature sizes as compared to the popular WOTS scheme
   b) *NOTS* offers an 84% and an 86% reductions in the signature and the key sizes respectively as compared to an existing compact variant of WOTS, i.e. WOTS$^+$.
   c) *NOTS* signatures are intelligent enough to allow the verifier for computation of the corresponding public key without any additional set of information.
2) We have formally proved that our proposed scheme (*NOTS*) is existentially unforgeable under adaptive chosen message attack model.

The rest of the paper is organized as: Section-2 will provide a preliminary knowledge about HBS schemes and post-quantum cryptocurrencies proposed to-date. In Section-3, we will discuss our proposed OTS scheme (*NOTS*) in detail. In Sections - 4, 5, and 6, we will respectively evaluate security, space requirements, and execution time of *NOTS*. Finally, in Section-7, we will conclude our discussion.

II. LITERATURE REVIEW

The popular OTS/FTS schemes proposed to-date include, Lamport-Diffie OTS (LD-OTS) [13], Winternitz OTS (WOTS) [14], WOTS^{PRF} [15], WOTS$^+$ [16], HORS [17], HORS with Tree (HORST) [18], and PRNG to obtain a random subset (PORS) [19].

The pioneer HBS scheme is the Merkle signature scheme (MSS) [14] which uses WOTS as its base OTS scheme. An improved version of MSS is eXtended Merkle Signature Scheme (XMSS) [12] which uses WOTS^{PRF} [15] as its base OTS scheme. An MSS tree or an XMSS tree can map a finite no. of OTS public keys to a single public key. An enhanced version of XMSS is Multi-tree XMSS (XMSS^{MT}) [20] which is capable of mapping *virtually* an unlimited no. of OTS key pairs to single public key. XMSS^{MT} also uses WOTS^{PRF} as the base OTS scheme. XMSS^{MT} is a state-based scheme which maintains a *state* to guarantee that a distinct *seed* is selected each time the scheme is instantiated to sign a new message. SPHINCS [18] is a stateless HBS scheme which guarantees a distinct seed in each of its instantiation without preserving a state. SPHINCS uses HORST FTS and WOTS$^+$ as its base schemes. Gravity-SPHINCS [19] is a compact version of SPHINCS which uses PORS and WOTS as its core schemes. SPHINCS-Simpira [21] is an efficient version of SPHINCS which have replaced simple hash functions (SHA256 and SHA512) by AES-based hash permutations

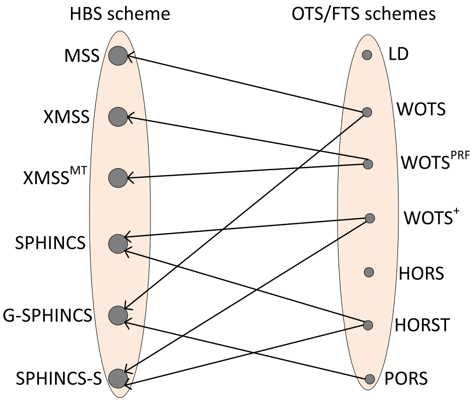


FIGURE 1. OTS/FTS schemes mapping to the HBS schemes.

Simpira [22]. Figure 1 shows a mapping between OTS/FTS and HBS schemes.

A. POST-QUANTUM CRYPTOCURRENCIES

The popular post-quantum cryptocurrencies proposed to-date include, IoTA [23], QRL [24], quantum-secured blockchain [25], qBitcoin [26], PQChain [27], and post-quantum blockchains incorporating lattice-based signature schemes [28], [29]. Among the existing post-quantum cryptocurrencies, three are using hash-based digital signature schemes. IoTA uses WOTS, QRL uses WOTS^{+} with XMSS, and PQChain recommends using WOTS^{PRF} with XMSS. The post-quantum blockchain proposed in [28] uses a short integer solutions (SIS) based signature scheme. Quantum-secured blockchain [25] allows a couple of peers to connect over a quantum channel to generate a symmetric key. Then those peers would be able to securely communicate over a classical channel with the help of their symmetric key. qBitcoin [26] proposes to represent the coins as quantum states. Because it is impossible to generate duplicate copies of a quantum state (i.e. no-cloning theorem), therefore, the proposed cryptocurrency is safe against double-spending attacks. Both quantum-secured blockchain and qBitcoin involve quantum-based technologies and hence, would only be practical when quantum computers will be available at a large scale.

III. PRELIMINARY KNOWLEDGE

In this section, we provide a preliminary knowledge about hash based digital signature schemes and OTS/FTS schemes. The discussion in this section helps reader to understand and compare, key and signature sizes of the existing OTS/FTS schemes.

A. HASH-BASED SIGNATURE (HBS) SCHEMES

The building block of an HBS scheme can either be an un-keyed hash functions, a keyed hash functions, or a block cipher (like AES). Because all these cryptographic protocols are very efficient (especially the un-keyed/keyed hash functions), therefore HBS schemes are the most efficient type of digital signature schemes [10]. An HBS scheme is a two-fold; first, there is a base OTS/FTS scheme and second is a hash tree that encapsulates a (finite or virtually infinite) no. of OTS/FTS public keys into another single public key. Although the hash trees are very important because otherwise key management is hard in HBS schemes, however there are real life applications which use an OTS or an FTS independently. For example, the popular post-quantum digital currency IoTA [23] uses WOTS signature scheme independently without a hash tree.

1) HASH-BASED OTS/FTS SCHEMES

Lamport proposed the very first hash based OTS scheme in late seventies [13]. The security of Lamport-Diffie (LD) OTS was proved later in the studies [30] and [31]. The LD OTS scheme suffers from impractically large key and signature sizes. In this scheme, we sign hash of the message, *bit-by-bit*; i.e. we create a separate signature-item for each of the individual bits. For a 512-bit long message-hash (we use l to denote bit-length of the message-hash to be signed), there will be a total of 512 signature-items. If each item itself is 512-bit long then, the total signature size will be 32.8KB. The key-size will be even double than the signature size because each of the bits has two key-items associated to it. The bit-length of an individual key/signature item depends upon the desired level of security therefore we refer it as the security parameter (n). The formulas for computing signature (σ) and the key (PK) sizes for the LD OTS scheme are given in equations (1) and (2).

$$\sigma_{(LD)} = (l)(n) \tag{1}$$
$$PK_{(LD)} = 2(l)(n) \tag{2}$$

Winternitz [14] made first major improvement in the initial work of Lamport. In Wintenitz OTS (WOTS) scheme, the bits are signed in groups/patches. We create a single signature-item for a group or patch of bits. The patch-size (let we denote it as p) is customizable, i.e. user can select the patch-size, he wants. The patch size is inversely proportional to the key and signature length, however, it is directly proportional to the processing cost. Therefore, a balance must be established. A typical patch-size is 4-bits. For WOTS, we can write the message-hash (H) like given in eq. (3). In WOTS, both key and the signatures consist of total $\frac{l}{p}$ items. Each of the private key item is transformed to its corresponding public key item by passing it through a hash chain. There are total 2^p hash iterations in a single hash chain. The signature elements are basically some of the middle stages of the hash chains. The hash of the message (to be signed) allows the signer to decide which of the middle stage of an individual chain should be declared as signatures. Finally, the hash of the corresponding message also allows the verifier to complete all of the hash chains to produce public key of the signer.

$$H = h_1||h_2||h_3||......||h_{\frac{l}{p}} \tag{3}$$

TABLE 1. Parameters description.

Parameter	Description
M	Message to be signed
H	Hash of the message (M) to be signed
H_{hex}	Hexadecimal representation of the hash of the message (H)
h	An individual character in the hash of the message (H)
p	The bit-length of an individual character (h) in the hash of the message
l	The bit-length of the hash of the message (H) to be signed
SK/PK	An OTS private key/public key
sk/pk	An individual value/element in SK/PK
σ^M	Signatures created on message M
σ_i	An individual value/element in σ
n	The bit-length of an individual key/signature element ($sk/pk/\sigma_i$)
$fsk/fpk/f\sigma$	First half part of $sk/pk/\sigma_i$
$bsk/bpk/b\sigma$	Last half part of $sk/pk/\sigma_i$
$(M^{\mathcal{F}}, \sigma^{\mathcal{F}})$	Message with corresponding signatures sent by $\mathcal{F}$
f_H	A common hash function
f_{ow}	A one-way (pre-image resistant) hash function
f_{cr}	A collision resistant hash function
c	The checksum appended to M by WOTS and its variants
$\mathcal{ADV}$	An adversary trying to break security of a hash function
$\mathcal{FOR}$	A forger trying to break security of the proposed scheme *NOTS*
$index(chr)$	A function which returns index of a given character chr in the hash (H) of the message
$sumDigits(str)$	A function which computes sum of the digits in a given string str

An individual patch in the message-hash can produce a value in the range *zero* to $2^p - 1$. WOTS also appends a checksum c to the message-hash which is computed using the formula given in equation (4). Finally, the signature size of WOTS can be computed using the formula given in equation (5). The key-size in WOTS is exactly same as the signature size. The WOTS scheme is provably secure under Existentially Unforgeable - Chosen Message Attack (EU-CMA) model [15].

$$c = \sum_{i=1}^{\frac{l}{p}} (2^p - 1) - h_i \tag{4}$$

$$\sigma_{(WOTS)} = \left(\frac{l}{p} \,||\, c\right)(n) \tag{5}$$

WOTS^{PRF} [15] and WOTS^{+} [16] are the two compact variants of WOTS which offer reduced key and signature sizes as compared to WOTS. WOTS^{PRF} has reduced key and signature sizes by replacing a collision resistant (CR) hash function by a pseudo-random function (PRF). For a CR hash function the length of an individual key/signature item must be at least *three* times the desired level of post-quantum security however, for a PRF the length of an individual key/signature item must be at least *two* times the desired level of post-quantum security. WOTS^{+} uses bit-masks to replace a CR hash function by an undetectable one-way function (which may either by a keyed hash function or a block cipher). The signature sizes of both WOTS^{PRF} and WOTS^{+} can be computed using the same formula given in eq. (5) by adjusting value of the security parameter (n) accordingly. The key size of WOTS^{PRF} is approximately same as its signature size however, the key size of WOTS^{+} is somehow larger because of an additional set of randomization elements. The formula to compute key size of WOTS^{+} is given in eq. (6)

$$PK_{(WOTS^{+})} = \left[\left(\frac{l}{p} \,||\, c\right) + 2^p\right](n) \tag{6}$$

The HORS FTS scheme yet provides another different approach for creating hash-based few time signatures [17]. Like WOTS scheme, HORS also creates signatures on the patches of bits; means there is a single signature-item for a patch of bits. However, the patch size in HORS must be significantly larger because for small sized patches this scheme will not be secure. The large sized patches reduce the signature size significantly as compared to WOTS. Another difference is that HORS does not append any checksum to the message-hash which also reduces the signature size. However the key size of HORS is extremely larger. The formulas to compute key and signature sizes of HORS scheme are given in eqs. (7) and (8) respectively. HORS scheme is nearly impractical because of its extremely large key size.

$$PK_{(HORS)} = (2^p)(n) \tag{7}$$

$$\sigma_{(HORS)} = \left(\frac{l}{p}\right)(n) \tag{8}$$

The PORS [19] scheme is very close to the HORS scheme however, offers stronger security than HORS at a very marginal computational overhead. The key and signature sizes of PORS are exactly same as HORS. The only difference is that in case of HORS, multiple bit-patches may correspond to a same signature-item whereas in PORS there is always a distinct signature-item against each of the bit-patches.

Our proposed scheme (*NOTS*) recommends using 4-bit long patches. The key and signature sizes of *NOTS* both are computed using the formula given in equation (9). Because

p is fairly smaller (just 4-bit) therefore both key and the signature sizes of *NOTS* are significantly smaller. *NOTS* offers just 1KB key and signature sizes for a 512-bit long security parameter (i.e. $n = 512$). The different parameters referred in this section and throughout the article have been explained in Table 1.

$$\sigma_{(NOTS)} = (2^p)(n) \tag{9}$$

IV. NOVEL ONE TIME SIGNATURES (NOTS): THE PROPOSED SCHEME

This section explains our proposed scheme (*NOTS*) in detail. Our proposed scheme works as follows:

A. KEY GENERATION

The private key (sk) is simply a set of sixteen values each being 512 bits long (Eq. (10)). We recommend generating all of the values in the private key from a single *seed*. We can apply a simple hash chain to the seed to generate the sk values. Because our scheme never disclose any of the sk values during signature verification, therefore it is safe to use just a simple chain of values generated using a common hash function like SHA512. The complete pseudo code for key generation is given in Algorithm 1.

$$SK = \sum_{i=0}^{15}[sk_i \cdot bitLength(sk_i) == 512] \tag{10}$$

The public key (pk) is computed from the sk. There is a corresponding pk value against each of the sk values. In order to compute a pk value, we divide the corresponding sk value into two equal halves and we compute hash of each of the halves for 129 times (Eq. (11)). The length of the hash function must be the same as the length of an individual half. Like, if size of a single half is 256 bit, then we may use the hash function SHA256.

$$PK = \sum_{i=0}^{15}\Bigg[pk_i = sha256^{129}\Big(sk_i\big[0, \frac{|sk_i|}{2}\big)\Big) + sha256^{129}\Big(sk_i\big[\frac{|sk_i|}{2}, |sk_i|\big)\Big)\Bigg] \tag{11}$$

B. SIGNATURE CREATION

We initiate the signature creation process by computing hash of the message (H) to be signed. We recommend using a 512-bit hash function (like SHA512). In this way, H will be consisting of a total 128 hexadecimal symbols. We use H_{hex} to denote hexadecimal representation of the message-hash. H_{hex} guides the signer for generating *index_strings* which is a list consisting of 16 different strings. The list *index_strings* basically classifies the indexes of the hexadecimal characters in H_{hex} into 16 different strings (Eqs. (12), (13)). There is a separate string of indexes for each type of alphabet in H_{hex}. In next step, the signer will compute sum of the digits in each of the *index_strings*; we name this new list as *sum_index_string*. Signer also ensure that all the values in the list *sum_index_string* must be in the range $\{1 \rightarrow 128\}$ (Eq. (14)).

$$\sum_{i=0}^{15} index\_string_i = <> \tag{12}$$

$$\forall h \in H_{hex} \cdot index\_string'_h = index\_string_h + index(h) \tag{13}$$

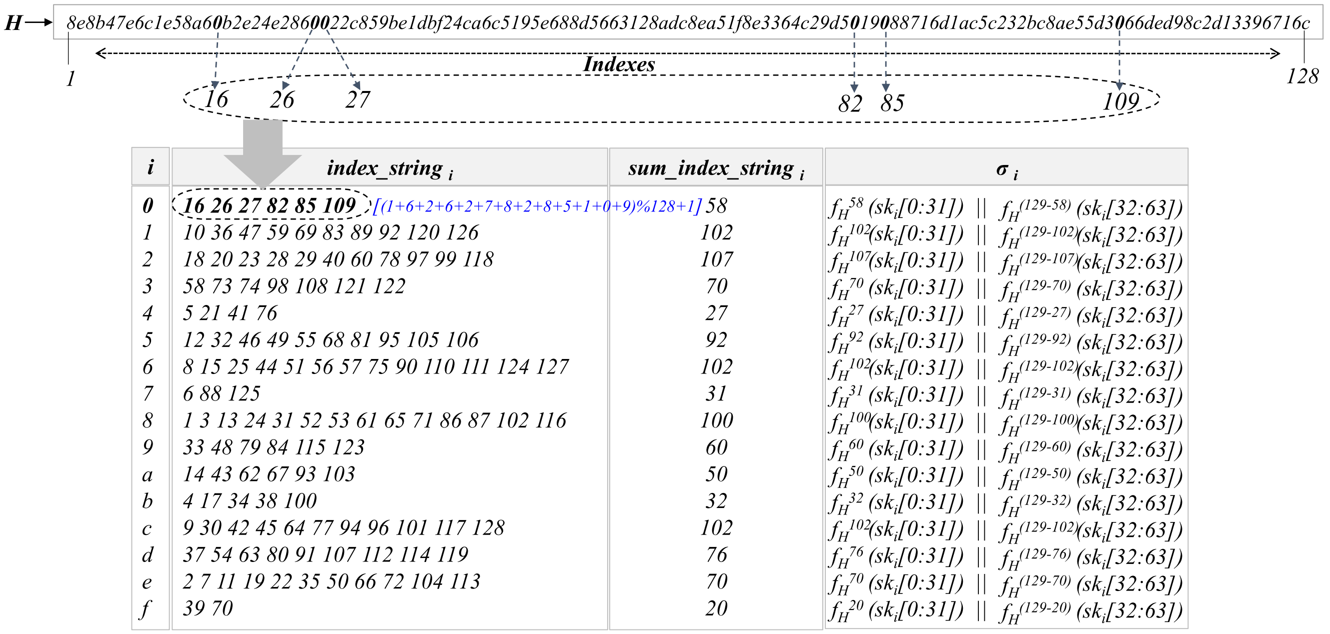
$$\sum_{i=0}^{15} sum\_index\_string_i = sumofdigits(index\_string_i)\ \%\ 128 + 1 \tag{14}$$

The *sum_index_string* will finally let the signer to produce signatures (σ) on the corresponding message (M). The signer will compute hash of each of the sk value for number of times, equal to the corresponding value in the list *sum_index_string*. While computing hash of an individual sk value, the signer will divide it into two halves (we say them *forward sk (fsk)* and *backward sk (bsk)*). Signer will compute hash of *fsk* for number of times, exactly equal to the corresponding value in the list *sum_index_string*, however, signer will compute hash of *bsk* for number of times equal to 129 *minus* the corresponding values in *sum_index_string*. Finally, signer will concatenate both of the final hash outputs to generate an individual signature value (Eq. (15)). Signer will adopt the same procedure for each of the 16 sk values to generate the 16 signature values. Figure 2 explains the signature creation process for an example message; and the pseudo code for signature creation is given in Algorithm 2.

$$\sum_{i=0}^{15}\Bigg[\sigma_i = sha256^{sum\_index\_string_i}\Big(sk_i\big[0, \frac{|sk_i|}{2}\big)\Big) + sha256^{129\ -\ sum\_index\_string_i}\Big(sk_i\big[\frac{|sk_i|}{2}, |sk_i|\big)\Big)\Bigg] \tag{15}$$

C. SIGNATURE VERIFICATION

The verifier will compute the *sum_index_string* following the same steps as followed by the signer during signature creation. The list *sum_index_string* will allow the verifier to produce the verification key (VK) from the signatures (σ). In order to compute an individual vk value from the corresponding σ value, the verifier will divide the σ value into two halves (we say them *forward signature* ($f\sigma$) and *backward signature* ($b\sigma$)). Verifier will compute hash of $f\sigma$ for number of times, equal to the 129 *minus* the corresponding value in the list *sum_index_string*, however, verifier will compute hash of $b\sigma$ for number of times, exactly equal to the corresponding value in *sum_index_string*. Verifier will concatenate both of the final hash outputs to generate an individual vk value (Eq. (16)). Verifier will adopt the same procedure for each of the 16 "σ values" to generate the 16 vk values. Finally, verifier will compare his own computed verification key VK with the signers previously announced public key PK. If both of the keys will be equal then verifier will accept the signatures. The pseudo code for signature verification is given


H
8e8b47e6c1e58a60b2e24e2860022c859be1dbf24ca6c5195e688d5663128adc8ea51f8e3364c29d5019088716d1ac5c232bc8ae55d3066ded98c2d13396716c
1
Indexes
128
16
26
27
82
85
109
[(1+6+2+6+2+7+8+2+8+5+1+0+9)%128+1]

i	*index_string $_i$*	*sum_index_string $_i$*	*σ $_i$*
0	16 26 27 82 85 109	58	$f_H^{58}(sk_i[0:31]) \mid\mid f_H^{(129-58)}(sk_i[32:63])$
1	10 36 47 59 69 83 89 92 120 126	102	$f_H^{102}(sk_i[0:31]) \mid\mid f_H^{(129-102)}(sk_i[32:63])$
2	18 20 23 28 29 40 60 78 97 99 118	107	$f_H^{107}(sk_i[0:31]) \mid\mid f_H^{(129-107)}(sk_i[32:63])$
3	58 73 74 98 108 121 122	70	$f_H^{70}(sk_i[0:31]) \mid\mid f_H^{(129-70)}(sk_i[32:63])$
4	5 21 41 76	27	$f_H^{27}(sk_i[0:31]) \mid\mid f_H^{(129-27)}(sk_i[32:63])$
5	12 32 46 49 55 68 81 95 105 106	92	$f_H^{92}(sk_i[0:31]) \mid\mid f_H^{(129-92)}(sk_i[32:63])$
6	8 15 25 44 51 56 57 75 90 110 111 124 127	102	$f_H^{102}(sk_i[0:31]) \mid\mid f_H^{(129-102)}(sk_i[32:63])$
7	6 88 125	31	$f_H^{31}(sk_i[0:31]) \mid\mid f_H^{(129-31)}(sk_i[32:63])$
8	1 3 13 24 31 52 53 61 65 71 86 87 102 116	100	$f_H^{100}(sk_i[0:31]) \mid\mid f_H^{(129-100)}(sk_i[32:63])$
9	33 48 79 84 115 123	60	$f_H^{60}(sk_i[0:31]) \mid\mid f_H^{(129-60)}(sk_i[32:63])$
a	14 43 62 67 93 103	50	$f_H^{50}(sk_i[0:31]) \mid\mid f_H^{(129-50)}(sk_i[32:63])$
b	4 17 34 38 100	32	$f_H^{32}(sk_i[0:31]) \mid\mid f_H^{(129-32)}(sk_i[32:63])$
c	9 30 42 45 64 77 94 96 101 117 128	102	$f_H^{102}(sk_i[0:31]) \mid\mid f_H^{(129-102)}(sk_i[32:63])$
d	37 54 63 80 91 107 112 114 119	76	$f_H^{76}(sk_i[0:31]) \mid\mid f_H^{(129-76)}(sk_i[32:63])$
e	2 7 11 19 22 35 50 66 72 104 113	70	$f_H^{70}(sk_i[0:31]) \mid\mid f_H^{(129-70)}(sk_i[32:63])$
f	39 70	20	$f_H^{20}(sk_i[0:31]) \mid\mid f_H^{(129-20)}(sk_i[32:63])$

FIGURE 2. NOTS: Signature creation.

TABLE 2. Hash functions security levels [30], [32], [33].

Hash function	Classical security		Quantum security	
	Pre-image	**Collision**	**Pre-image**	**Collision**
SHA160	160-bit	80-bit	80-bit	53-bit
SHA256	256-bit	128-bit	128-bit	85-bit
SHA384	384-bit	192-bit	192-bit	128-bit
SHA512	512-bit	256-bit	256-bit	171-bit

in Algorithm 3.

$$\sum_{i=0}^{15}\Big[vk_i = sha256^{129\,-\,sum\_index\_string_i}\Big(\sigma_i\big[0, \frac{|\sigma_i|}{2}\big)\Big) + sha256^{sum\_index\_string_i}\Big(\sigma_i\big[\frac{|\sigma_i|}{2}, |\sigma_i|\big)\Big)\Big] \quad (16)$$

V. *NOTS* Security Analysis

The foremost security requirement of *NOTS* is that it must be populated with a secure hash function which can resist three types of attacks, pre-image attacks, second pre-image attacks, and collision attacks. In the case of the pre-image attack, the challenge for the adversary ($\mathcal{ADV}$) is to find such an *input* which corresponding *output* is known to him (Eq. 17). In case of second pre-image attack, the adversary knows an input-output pair (x,y), whereas the challenge for him is to find another input which must be different from x, however its output should be the same (i.e. y) [Eq. 18]. Finally, in collision-based challenge, the adversary has to find any two different inputs which must map to the same output (Eq. 19).

$$Pr[y = f_h(x); x' \leftarrow \mathcal{ADV}(y) : x = x'] \leq \epsilon \quad (17)$$

$$Pr[y = f_h(x); x' \leftarrow \mathcal{ADV}(x, y) : x' \neq x \wedge y = f_h(x')] \leq \epsilon \quad (18)$$

$$Pr[x, x' \leftarrow \mathcal{ADV} : x \neq x' \wedge f_h(x) = f_h(x')] \leq \epsilon \quad (19)$$

The resistance power of a cryptographic protocol against different types of attacks is generally known as the security-level offered by that protocol. The classical and quantum security levels offered by a hash function (f_h) depend upon digest-size (d) of that function [32]. The post-quantum security level of a family of hash functions is relatively smaller than the classical security level because of the popular Grover's search algorithm [34]. A d-sized hash function is capable of providing d-bit classical and $\frac{d}{2}$-bit post-quantum security against pre-image and second pre-image based attacks. However, collision resistant is relatively a complex security requirement and hence, relatively harder to achieve. Therefore, a d-sized hash function provides $\frac{d}{2}$-bit classical and $\frac{d}{3}$-bit post-quantum security against collision based attacks [30], [33]. Table- 2 lists down the classical and post-quantum security levels of the common hash functions.

A. FORMAL SECURITY PROOF OF NOTS

In this subsection, we formally proof that *NOTS* is an existentially unforgeable signature scheme under adaptive chosen message attack (CMA). We prove that NOTS is unforgeable until the underlying hash function used by NOTS is a *oneway* hash function. Formally stating, we prove that the security of NOTS is a *security reduction* of the *onewayness* of the underlying hash function used to instantiate NOTS.

Algorithm 1 Key Generation

```
Input: security parameter(n)                    ▷ we use n = 512
Output: sk[ ], pk[ ]
$seed \leftarrow os.urandom(64)$                ▷ “seed” is 512-bit (64-byte) cryptographic random value
$s \leftarrow sha512(seed)$                     ▷ hash of “seed” is computed and stored in “s”
$sk \leftarrow [\,]$                            ▷ “sk” initialization
for $a = 0 \rightarrow 15$ do                   ▷ generates the private key “sk”
    $sk.append(s)$
    $s \leftarrow sha512(s)$                    ▷ “sk” is simply the hash-chain of the “seed”
end for
$pk \leftarrow [\,]$                            ▷ public key “pk” initialization
for $a = 0 \rightarrow 15$ do
    $k \leftarrow sk[a]$
    $kf \leftarrow k[0:31]$                     ▷ each sk element is divided into two halves “kf” and “kb”
    $kb \leftarrow k[32:63]$
    for $b = 1 \rightarrow 129$ do              ▷ a pk-element is the 129$^{th}$ post-image of the corresponding sk-element
        $kf \leftarrow sha256(kf)$              ▷ hash chains are applied to “kf” and “kb” separately
        $kb \leftarrow sha256(kb)$
    end for
    $pk.append(kf + kb)$                        ▷ concatenation of the final chain results produce the corresponding pk-element
end for
```

Algorithm 2 Signature Creation

```
Input: message(M), private key(sk[ ])
Output: signatures(σ[ ])
$H \leftarrow sha512(M)$                        ▷ hash of message is computed
$H_{hex} \leftarrow hexlify(H)$                 ▷ hash of message in its hexadecimal representation
$index\_strings \leftarrow [\,]$                ▷ defines an empty list “index_strings”
$hex\_symbols \leftarrow$ “0123456789abcdef″    ▷ the hexadecimal alphabet-set stored as an array
for $(hs)$ in $(hex\_symbols)$ do               ▷ a loop iterating for each of the hexadecimal alphabet
    $str \leftarrow$ “ ″                        ▷ an empty string declaration
    for $a = 1 \rightarrow 128$ do              ▷ a loop parsing whole of the hexadecimal message hash
        if $H_{hex}[a] == hs$ then              ▷ filters the message-hash indexes containing the corresponding hash-alphabet
            $str \leftarrow str + a$
        end if
    end for
    $index\_strings.append(str)$                ▷ appends “str” from the inner loop into the list “index_strings”
end for
$sum\_index\_strings \leftarrow [\,]$           ▷ defines an empty list “sum_index_strings”
for $(indstr)$ in $(index\_strings)$ do         ▷ a loop parsing whole of the list “index_strings”
    $sum \leftarrow 0$
    for $(a)$ in $(indstr)$ do                  ▷ to access each of the digit in the string “indstr”
        $sum \leftarrow sum + int(a)$           ▷ summing-up the digits of “indstr”
    end for
    $sum \leftarrow (sum\%128) + 1$             ▷ to ensure that “sum” always lies in the range (1 → 128)
    $sum\_index\_strings.append(sum)$           ▷ appending “sum” from the inner loop into the list “sum_index_string”
end for
$\sigma \leftarrow$ “ ″                         ▷ initializes signatures as an empty list
for $a = 0 \rightarrow 15$ do
    $k \leftarrow sk[a]$
    $kf \leftarrow k[0:31]$                     ▷ each sk-element is divided into two halves “kf” and “kb”
    $kb \leftarrow k[32:63]$
    for $f = 1 \rightarrow sum\_index\_strings[a]$ do              ▷ the “kf” will be hashed for “sum_index_strings” no. of times
        $kf \leftarrow sha256(kf)$
    end for
    for $b = 1 \rightarrow (129 - sum\_index\_strings[a])$ do      ▷ the “kb” will be hashed for 29 minus the “sum_index_strings” no. of times
        $kb \leftarrow sha256(kb)$
    end for
    $\sigma.append(kf + kb)$                    ▷ concatenation of the above chain results produce the corresponding signature-element
end for
```

1) AN OVERVIEW OF *NOTS*

NOTS is a triple *(GEN, SIGN, VERIFY)*. *GEN* takes a security parameter (n) as input and returns a key pair (*SK*, *PK*), such that: $SK = \sum_{i=0}^{15} sk_i$ and $PK = \sum_{i=0}^{15} pk_i$. *GEN* follows Eq. (11) to compute an individual pk_i from the corresponding sk_i. *SIGN* takes a message (M) as input and returns signatures of M, i.e. σ^M. In order to compute the signatures, *SIGN* computes a list/array of sixteen values *sum_index_strings* (following Equations. (12) to (14)) and transforms each of the sk_i into the corresponding σ_i following

Algorithm 3 Signature Verification

```
Input: message(M), signatures(σ[ ]), public key(pk[ ])
Output: Succeeded/Failed
Follow steps 1 to 22 of algorithm 2 (Signature Creation) to generate the list "sum_index_strings"
$vk \leftarrow$ " "                                   ▷ the public key computed by the verifier
for $a = 0 \rightarrow 15$ do
    $s \leftarrow \sigma[a]$
    $sf \leftarrow s[0:31]$                           ▷ each signature element is divided into two halves "sf" and "sb"
    $sb \leftarrow s[32:63]$
    for $f = 1 \rightarrow (129 - sum\_index\_strings[a])$ do   ▷ "sf" will be hashed for 129 − "sum_index_strings" no. of times
        $sf \leftarrow sha256(sf)$
    end for
    for $b = 1 \rightarrow sum\_index\_strings[a]$ do   ▷ "sb" will be hashed for "sum_index_strings" no. of times
        $sb \leftarrow sha256(sb)$
    end for
    $vk.append(sf + sb)$                              ▷ concatenation of the above chain results produce the corresponding ver_pk-element
end for
if $\sum_{i=0}^{15} vk[i] == pk[i]$ then             ▷ the "ver_pk" computed by the verifier must be equal to the "pk"
    output : verification succeeded
else
    output : verification failed
end if
```

Eq. (15). *VERIFY* takes message (*M*), signatures (σ^M), and public key (*PK*) as input and returns either *TRUE* (if σ^M is valid signature of *M*) or *FALSE* otherwise. *VERIFY* follows Eq. (16) to compute a verification key (*VK*) and compares this verification key with the public key (*PK*) to reach a consensus about validity of the signatures.

2) EXISTENTIAL UNFORGEABILITY OF *NOTS*

GEN generates a new key pair (*SK*, *PK*). A signing oracle $\mathcal{O}$ having knowledge of *SK* is able to sign an arbitrary number of messages. A forger $\mathcal{FOR}$ accepts challenge of breaking security of the scheme (NOTS). $\mathcal{FOR}$ has knowledge of *PK* and the underlying algorithm of *NOTS*, however, $\mathcal{FOR}$ does not know *SK*. $\mathcal{FOR}$ can query a message ($M^{\mathcal{Q}}$) to $\mathcal{O}$, where $\mathcal{O}$ must return valid signature of $M^{\mathcal{Q}}$ to $\mathcal{FOR}$. In the end, $\mathcal{FOR}$ returns a message-signature pair ($M^{\mathcal{F}}$, $\sigma^{\mathcal{F}}$) to $\mathcal{O}$. $\mathcal{FOR}$ wins the game if, $\sigma^{\mathcal{F}}$ are valid signatures of $M^{\mathcal{F}}$ *and* $M^{\mathcal{F}} \neq M^{\mathcal{Q}}$. *NOTS* is an existentially unforgeable scheme if, $\mathcal{FOR}$ queries at most one message to $\mathcal{O}$ and the success probability of $\mathcal{FOR}$ in a time *t* is at most ϵ. We formally write it as, NOTS is a *(t, ϵ, 1)-existentially unforgeable signature scheme*.

3) SECURITY REDUCTION TO PRE-IMAGE RESISTANCE

The background knowledge provided in Subsections V-A1 and V-A2 allows us to finally present our security reduction proof. In this subsection, we prove that the security of NOTS is a *security reduction* of the *onewayness* of the underlying hash function used by NOTS. An adversary $\mathcal{ADV}_{onewayness}$ acting as a signing oracle $\mathcal{O}$ initiates the experiment. Algorithm 4 explains how an adversary ($\mathcal{ADV}_{onewayness}$) can use the forger ($\mathcal{FOR}_{NOTS}$) to break onewayness of the hash function (f_{ow}) used by NOTS. $\mathcal{ADV}_{onewayness}$ initiates by generating a new key pair (SK, PK). Then he alters forward part of a randomly chosen pk-item (i.e. fpk_α). He computes hash of the challenged post-image *y* (for which he has to deduce the pre-image) for $129 - \beta$ times and sets this value as the new value of fpk_α, where β is also a randomly chosen value. Then he runs $\mathcal{FOR}_{NOTS}$. When $\mathcal{FOR}_{NOTS}$ queries a message ($M^{\mathcal{Q}}$) for signatures then either $\mathcal{ADV}_{onewayness}$ will respond the $\mathcal{FOR}_{NOTS}$ with the valid signature ($\sigma^{\mathcal{Q}}$) of $M^{\mathcal{Q}}$ or will quit the algorithm (Lines $6 - 11$). Finally, when $\mathcal{FOR}_{NOTS}$ will return a message-signature pair (i.e. $M^{\mathcal{F}}, \sigma^{\mathcal{F}}$) to $\mathcal{ADV}_{onewayness}$ then $\mathcal{ADV}_{onewayness}$ will be able to return the challenged pre-image *x*, if and only if, the conditions in Line-13 are satisfied and the value of $sum\_index\_strings_\alpha$ for $M^{\mathcal{F}}$ is sufficiently larger.

Now we compute success probability of the $\mathcal{ADV}_{onewayness}$. Since, $\mathcal{ADV}_{onewayness}$ choses both α and β purely at random therefore, the success probability of $\mathcal{ADV}_{onewayness}$ in Lines $6 - 11$ is $(128 - \beta)(2048)^{-1}$. The $\mathcal{FOR}_{NOTS}$'s success probability in Line-13 is ϵ_{NOTS}. Finally, the success probability of $\mathcal{ADV}_{onewayness}$ in Line-14 is $(\beta)(2048)^{-1}$. This allows us to conclude that the overall maximum success probability of $\mathcal{ADV}_{onewayness}$ is $\epsilon_{NOTS}(250)^{-1}$. The total time taken by $\mathcal{ADV}_{onewayness}$ ($t_{\mathcal{ADV}_{onewayness}}$) includes, the key generation time t_{GEN} (Line-1), the message signing time t_{SIGN} (Lines $8 - 10$), and the $\mathcal{FOR}_{NOTS}$'s time t_{NOTS} (Line-12). Hence it is proved that NOTS is an existentially unforgeable scheme under CMA model with $\epsilon_{NOTS} \leq (250)(\epsilon_{onewayness})$ and $t_{NOTS} = t_{onewayness} - t_{GEN} - t_{SIGN}$. The security reduction of *NOTS* to the *onewayness* of the underlying hash function allows us to infer that *NOTS* is capable of providing 128−bit post-quantum security.

4) SECURITY REDUCTION TO COLLISION RESISTANCE

This subsection formally reduces security of our proposed scheme *NOTS* to collision-resistance of the hash function (f_{cr}) used by the scheme. Algorithm 5 explains that how an adversary $\mathcal{ADV}_{collision}$ can exploit a forger $\mathcal{FOR}_{NOTS}$ to find hash-collisions in f_{cr}. $\mathcal{ADV}$ generates a new key-pair (*SK*, *PK*) [step-1]. Then $\mathcal{ADV}$ initiates the forger $\mathcal{FOR}$, providing him knowledge of the corresponding public

Algorithm 4 $\mathcal{ADV}_{onewayness}$

```
Input: NOTS scheme (GEN, SIGN, VERIFY), one-way function $f_{ow}$, security parameter $n$, forger $\mathcal{FOR}$, a post-image $y$
Output: A pre-image $x$ computed from $y$, such that: $y = f_{ow}(x)$ or fail
 1: Generate a new NOTS key pair (SK, PK)
 2: Randomly choose an $\alpha \in \{0, \cdots, 15\}$
 3: Randomly choose a $\beta \in \{1, \cdots, 128\}$
 4: $fpk_\alpha \leftarrow f_{ow}^{129-\beta}(y)$        ▷ $\mathcal{ADV}$ tampers first half of the corresponding PK-item
 5: Run the forger $\mathcal{FOR}(., PK)$
 6: When $\mathcal{FOR}$ queries signature on a message $(M^{\mathcal{Q}})$ then
 7:     If $sum\_index\_strings_\alpha^{\mathcal{Q}} < \beta$ then return fail
 8:     Generate signatures on message $M^{\mathcal{Q}}$ like:
 9:         $\sum_{i=0}^{15} \left(\sigma_i^{\mathcal{Q}} \leftarrow f_{ow}^{sum\_index\_strings_i^{\mathcal{Q}}}(fsk_i) + f_{ow}^{129-sum\_index\_strings_i^{\mathcal{Q}}}(bsk_i)\right)$ for $i \neq \alpha$        ▷ $sk_i = fsk_i + bsk_i$
10:         $\sigma_\alpha^{\mathcal{Q}} \leftarrow f_{ow}^{sum\_index\_strings_\alpha^{\mathcal{Q}}-\beta}(y) + f_{ow}^{129-sum\_index\_strings_\alpha^{\mathcal{Q}}}(bsk_\alpha)$
11:     Send $\sigma^{\mathcal{Q}}$ back to $\mathcal{FOR}$
12: When $\mathcal{FOR}$ returns a message/signature pair $(M^{\mathcal{F}}, \sigma^{\mathcal{F}})$ then
13:     If $\sigma^{\mathcal{F}}$ are valid signatures of $M^{\mathcal{F}}$ and $M^{\mathcal{F}} \neq M^{\mathcal{Q}}$ then
14:         If $sum\_index\_strings_\alpha^{\mathcal{F}} > \beta$ then return fail
15:         Compute $x \leftarrow f_{ow}^{\beta-sum\_index\_strings_\alpha^{\mathcal{F}}-1}(f\sigma_\alpha^{\mathcal{F}})$        ▷ "$f\sigma$" refers to the first half part of the corresponding signature-item
16:         return $x$
17: In all other cases return fail
```

key (PK) [step-2]. $\mathcal{FOR}$ can ask $\mathcal{ADV}$ to generate signatures on a message $M^{\mathcal{Q}}$, and $\mathcal{ADV}$ returns valid signatures of $M^{\mathcal{Q}}$ to $\mathcal{FOR}$ [steps 3, 4]. Finally, $\mathcal{ADV}$ exploits the message-signature pair returned by $\mathcal{FOR}$ $(M^{\mathcal{F}}, \sigma^{\mathcal{F}})$ to find collisions in f_{cr} [steps 5 - 11].

5) QUANTUM-RESILIENCY IN *NOTS*

NOTS is purely based on hash functions which are provably quantum-resistant [16]. Even a powerful quantum will be able to slightly affect security of the hash functions [30], with the help of Grover's quantum-based search algorithm [34]. This affect can easily be subsided by adjusting digest-size of the corresponding hash function. For example, because of a quantum computer, the security of the common hash function SHA256 will decrease from 128-bit to 85-bit. One can easily manage the affect by replacing SHA256 by SHA384, which (i.e. SHA384) offers 128-bit post-quantum security [33].

VI. *NOTS* KEY AND SIGNATURE SIZES

NOTS offers a significant reduction in both key and signature sizes as compared to all of the existing OTS/FTS schemes. The popular OTS/FTS schemes proposed before *NOTS* include Lamport OTS [13], WOTS and its variants [14]–[16], HORS [17], and PORS [19]. *NOTS* is basically a WOTS-like scheme which allows computation of the public key purely from signatures without any additional set of information. The key and signature lengths of WOTS-like schemes are already smaller than the other type of OTS/FTS schemes, however, *NOTS* offers a further significant reduction in both key and signature lengths as compared to WOTS and its existing variants. A comparison of the "key and signature" sizes of *NOTS* with other OTS/FTS schemes has been given in Table 3. The formulas for computing key and signature sizes for the different OTS/FTS schemes have already been explained in Subsection III-A (see Equations (1) -(9)). The key and signature sizes vary with values of certain parameters, therefore, Table 3 also specifies the values, we set for different parameters in order to make a comparison. Here, we provide a brief explanation of those parameters. n is the bit-length of an individual key/signature element. The size of an individual key/signature element is directly related to the security strength of the corresponding scheme. The digest-size of the hash function used by the scheme (f_H) to transform an sk_i to the corresponding pk_i also affects key/signature sizes and security strength of the scheme. l is bit-length of the hash of the message (H) to be signed. $|key|$ represents the number of values/elements in SK/PK. w represents the number of hash iterations used to transform an sk_i to the corresponding pk_i. $|\sigma|$ represents the number of elements in the signatures. σ-size is the size of signatures (in KB) computed against the parameters-values specified in the corresponding row. $|key|$-size is size of SK/PK (in KB) computed against parameters-values specified in the corresponding row. Finally, Post-Quantum Security Level (PQ-SL) allows reader to realize and compare security strengths of the different schemes against quantum-computer based attacks.

The comparison shows that *NOTS* offers an 88% reduction in both key and signature sizes as compared to WOTS and an 84% reduction in both key and signature sizes as compared to WOTS^{PRF}. Furthermore, *NOTS* offers an 84% and an 86% reductions in the signature and key sizes respectively as compared to the compact variant of WOTS, i.e. WOTS$^+$. Finally, *NOTS* has achieved all these reductions in key and signature sizes without compromising the security level. *NOTS* still offers an appropriate post-quantum security.

VII. *NOTS* EXECUTION TIME

NOTS offers fairly smaller execution time. The exact execution time for the three algorithms, *key generation*, *signature*

Algorithm 5 $\mathcal{ADV}_{collision}$

```
Input: NOTS scheme (GEN, SIGN, VERIFY), collision-resistant function $f_{cr}$, security parameter $n$, forger $\mathcal{FOR}$
Output: $x, x'$, such that, $x \neq x' \wedge f_{cr}(x) = f_{cr}(x')$ or fail
Generate a new NOTS key pair (SK, PK)
Run the forger $\mathcal{FOR}(., PK)$
When $\mathcal{FOR}$ queries signature on a message $(M^{\mathcal{Q}})$ then
    Respond $\mathcal{FOR}$ with $\sigma^{\mathcal{Q}}$
When $\mathcal{FOR}$ returns a message/signature pair $(M^{\mathcal{F}}, \sigma^{\mathcal{F}})$ then
    If $\sigma^{\mathcal{F}}$ are valid signatures of $M^{\mathcal{F}}$ and $M^{\mathcal{F}} \neq M^{\mathcal{Q}}$ then
        Compute sum_index_strings for $M^{\mathcal{F}}$
        If there exists an $i$ such that, $f_{cr}^{sum\_index\_strings_i^{\mathcal{F}}}(fsk_i) \neq f\sigma_i^{\mathcal{F}}$ then
            Compute the smallest $j$ such that, $f_{cr}^{sum\_index\_strings_i^{\mathcal{F}}+j}(fsk_i) = f_{cr}^{j}(f\sigma_i^{\mathcal{F}})$ and $f_{cr}^{sum\_index\_strings_i^{\mathcal{F}}+(j-1)}(fsk_i) \neq f_{cr}^{j-1}(f\sigma_i^{\mathcal{F}})$
            $x \leftarrow f_{cr}^{sum\_index\_strings_i^{\mathcal{F}}+(j-1)}(fsk_i), x' \leftarrow f_{cr}^{j-1}(f\sigma_i^{\mathcal{F}})$
            return $(x, x')$
In any other case return fail
```

TABLE 3. Key and signature sizes: Hash-based OTS/FTS schemes.

Scheme	n [1]	Hash function	l [2]	|Key| [3]	w [4]	$\|\sigma\|$ [5]	σ-size [6]	Key-size [7]	PQ-SL [8]
[13]	512	SHA512	512	1024	1	512	32.8	65.5	171
[14]	512	SHA512	512	131	16	131	8.4	8.4	171
[15]	384	SHA384	512	131	16	131	6.3	6.3	173
[16]	384	SHA384	512	147	16	131	6.3	7.1	185
[19]	512	SHA512	512	65536	1	32	2.0	4194	171
NOTS	512	SHA256	512	16	129	16	1.0	1.0	128

[1] *Security parameter (n)*: The length of an individual key/signature element
[2] The bit-length of the hash of the message to be signed
[3] The total no. of elements/values in the set of private/public key
[4] The no. of hash iterations to transform an sk-element to the corresponding pk-element
[5] The total no. of elements/values in the set of signatures
[6] The size of signatures in KB
[7] The size of the private/public key in KB
[8] The post-quantum security level offered by the scheme

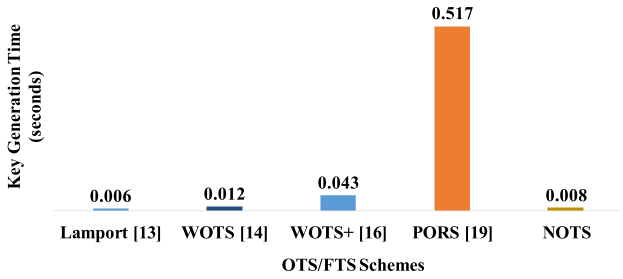


FIGURE 3. Key generation time of the OTS/FTS schemes.

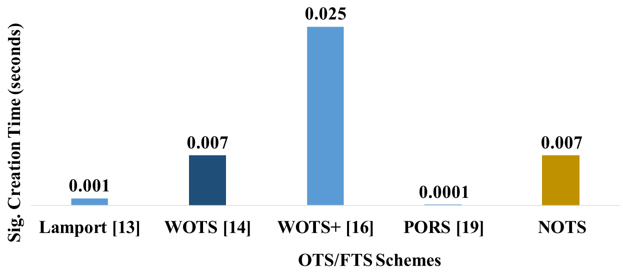


FIGURE 4. Sig. Creation time of the OTS/FTS schemes.

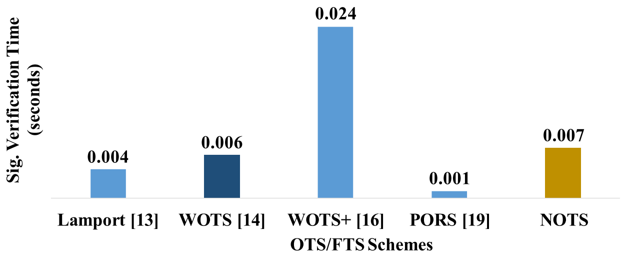


FIGURE 5. Sig. Verification time of the OTS/FTS schemes.

creation, and *signature verification* can be seen in the graphs in Figures 3, 4, and 5 respectively. The graphs also help to compare the execution time of *NOTS* with other OTS/FTS schemes. All these results have been taken on Intel Core i5 CPU (2.4 GHz) with 4GB RAM, running Windows 8.1 32-bit release. The schemes have been implemented in Python language using the environment "JetBrains PyCharm Community Edition 2018.3.3". The values of parameters set for these implementations are the same as given in table 3. The results show that the execution time of *NOTS* is comparable to other OTS/FTS schemes. Because *NOTS* is a WOTS-like scheme therefore we were specially concerned to compare its execution time with WOTS [14] and WOTS$^+$ [16]. The results show that execution time of *NOTS* is equal to WOTS, whereas, *NOTS* is clearly faster than WOTS$^+$. Furthermore, WOTS$^+$ uses bit-masking and randomization to replace a collision resistant (CR) hash functions by an undetectable one-way function, however, research reports that bit-masking is more expensive to achieve on quantum processors as compared to collision resistance [35]. Therefore, NOTS is based on pure CR hash functions and avoids bit-masking and/or randomization operations.

VIII. CONCLUSION

In this article, we have proposed a novel one-time signature scheme *NOTS*, which offers minimum key and signature sizes from all of the existing OTS/FTS schemes. *NOTS* has achieved an 88% reduction in both key and signature sizes as compared to the popular WOTS scheme. Furthermore, *NOTS* has achieved an 84% and an 86% reductions in the signature and the key sizes respectively as compared to the existing compact variant of WOTS, i.e. WOTS$^+$. The execution time of *NOTS* is fairly smaller for all three algorithms, key generation, signature creation, and signature verification. *NOTS* offers an appropriate level of post-quantum security. *NOTS* can be used as a base OTS of any of the popular hash based digital signature scheme like XMSS or XMSS^{MT} etc. to achieve a magical reduction in both key and signature sizes. The minimal key and signature sizes of *NOTS* allow us to hope that *NOTS*-based digital signature schemes will prove a best alternate of ECDSA in cryptocurrencies, in the quantum era.

REFERENCES

[1] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976, doi: 10.1109/tit.1976.1055638.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, Jan. 1983, doi: 10.1145/357980.358017.

[3] T. Elgamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. 31, no. 4, pp. 469–472, Jul. 1985.

[4] K. Koyama, U. M. Maurer, T. Okamoto, and S. A. Vanstone, "New public-key schemes based on elliptic curves over the ring Z_n," in *Proc. 11th Annu. Int. Cryptol. Conf. Adv. Cryptol.* London, U.K.: Springer-Verlag, 1992, pp. 252–266. [Online]. Available: http://dl.acm.org/citation.cfm?id=646756.705363

[5] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, Oct. 1997, doi: 10.1137/S0097539795293172.

[6] D. Aggarwal, G. Brennen, T. Lee, M. Santha, and M. Tomamichel, "Quantum attacks on bitcoin, and how to protect against them," *Ledger*, vol. 3, no. 1, p. 68–90, 2018, doi: 10.5195/LEDGER.2018.12.

[7] W. Buchanan and A. Woodward, "Will quantum computers be the end of public key encryption?" *J. Cyber Secur. Technol.*, vol. 1, no. 1, pp. 1–22, Jan. 2017.

[8] J. Buchmann, C. Coronado, M. Döring, D. Engelbert, C. Ludwig, R.Overbeck, A. Schmidt, A. Vollmer, and R.-P. Weinmann, "Post-quantum signatures," IACR Cryptol. ePrint, Berlin, Germany, Tech. Rep. 297, 2004.

[9] J. Buchmann, E. Dahmen, and M. Szydlo, *Hash-based Digital Signature Schemes*. Berlin, Germany: Springer, 2009, pp. 35–93, doi: 10.1007/978-3-540-88702-7_3.

[10] C. Dods, N. P. Smart, and M. Stam, "Hash based digital signature schemes," in *Cryptography Coding*, N. P. Smart, Ed. Berlin, Germany: Springer, 2005, pp. 96–115.

[11] D. Naor, A. Shenhav, and A. Wool, "One-time signatures revisited: Practical fast signatures using fractal Merkle tree traversal," in *Proc. IEEE 24th Conv. Electr. Electron. Eng. Israel*, Nov. 2006, pp. 255–259.

[12] J. Buchmann, E. Dahmen, and A. Hülsing, "XMSS—A practical forward secure signature scheme based on minimal security assumptions," in *Post-Quantum Cryptography*, B.-Y. Yang, Ed. Berlin, Germany: Springer, 2011, pp. 117–129.

[13] L. Lamport, "Constructing digital signatures from a one-way function," SRI Int., Menlo Park, CA, USA, Tech. Rep. CSL-98, 1979.

[14] R. C. Merkle, "A certified digital signature," in *Advces Cryptology*, G. Brassard, Ed. New York, NY, USA: Springer, 1990, pp. 218–238.

[15] J. Buchmann, E. Dahmen, S. Ereth, A. Hülsing, and M. Rückert, "On the security of the winternitz one-time signature scheme," in *Progress in Cryptology*, A. Nitaj and D. Pointcheval, Eds. Berlin, Germany: Springer, 2011, pp. 363–378.

[16] A. Hülsing, "W-OTS+—Shorter signatures for hash-based signature schemes," in *Progress in Cryptology*, A. Youssef, A. Nitaj, and A. E. Hassanien, Eds. Berlin, Germany: Springer, 2013, pp. 173–188.

[17] L. Reyzin and N. Reyzin, "Better than BIBA: Short one-time signatures with fast signing and verifying," in *Information Security Privacy*, L. Batten and J. Seberry, Eds. Berlin, Germany: Springer, 2002, pp. 144–153.

[18] D. J. Bernstein, D. Hopwood, A. Hülsing, T. Lange, R. Niederhagen, L. Papachristodoulou, M. Schneider, P. Schwabe, and Z. Wilcox-O'Hearn, "Sphincs: Practical stateless hash-based signatures," in *Advances in Cryptology*, E. Oswald and M. Fischlin, Eds. Berlin, Germany: Springer, 2015, pp. 368–397.

[19] J.-P. Aumasson and G. Endignoux, "Improving stateless hash-based signatures," in *Topics in Cryptology*, N. P. Smart, Ed. Cham, Switzerland: Springer, 2018, pp. 219–242.

[20] A. Hülsing, L. Rausch, and J. Buchmann, "Optimal parameters for xmssmt," in *Security Engineering and Intelligence Informatics*, A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, and L. Xu, Eds. Berlin, Germany: Springer, 2013, pp. 194–208.

[21] S. Gueron and N. Mouha, "Sphincs-simpira: Fast stateless hash-based signatures with post-quantum security," IACR Cryptol, Bellevue, WA, USA, Tech. Rep. 645, 2017.

[22] S. Gueron and N. Mouha, "Simpira v2: A family of efficient permutations using the AES round function," in *Advances in Cryptology*, J. H. Cheon and T. Takagi, Eds. Berlin, Heidelberg: Springer, 2016, pp. 95–125.

[23] S. Popov, "Academic papers IoTA, the tangle, Version 1.4.3," 2018. [Online]. Available: https://www.iota.org/research/academic-papers

[24] theQRL. (2016). *The Quantum Resistant Ledger. QRL White Paper*. [Online]. Available: https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf

[25] E. Kiktenko, N. Pozhar, M. Anufriev, A. Trushechkin, R. Yunusov, Y. Kurochkin, A. Lvovsky, and A. Fedorov, "Quantum-secured blockchain," *Quantum Sci. Technol.*, vol. 3, no. 3, May 2017, Art. no. 035004.

[26] K. Ikeda, "Qbitcoin: A peer-to-peer quantum cash system," in *Intelligent Computing*, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham, Switzerland: Springer, 2019, pp. 763–771.

[27] R. El Bansarkhani, M. Geihs, and J. Buchmann, "Pqchain: Strategic design decisions for distributed ledger technologies against future threats," *IEEE Secur. Privacy*, vol. 16, no. 4, pp. 57–65, Jul./Aug. 2018.

[28] Y. Gao, X. Chen, Y. Sun, X. Niu, and Y. Yang, "A secure cryptocurrency scheme based on post-quantum blockchain," *IEEE Access*, vol. 6, pp. 27205–27213, 2018.

[29] C.-Y. Li, X.-B. Chen, Y.-L. Chen, Y.-Y. Hou, and J. Li, "A new lattice-based signature scheme in post-quantum blockchain network," *IEEE Access*, vol. 4, pp. 2026–2033, 2018.

[30] E. Dahmen, K. Okeya, T. Takagi, and C. Vuillaume, "Digital signatures out of second-preimage resistant hash functions," in *Post-Quantum Cryptography*, J. Buchmann and J. Ding, Eds. Berlin, Germany: Springer, 2008, pp. 109–123.

[31] L. C. Garcia, "On the security and the efficiency of the merkle signature scheme," IACR Cryptol. ePrint Archive, Berlin, Germany, Tech. Rep. 192, Jan. 2005.

[32] A. K. Lenstra, "Key length. Contribution to the handbook of information security," Tech. Rep., 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=91FA82192CFC802B6C6A91896F82AD5C?doi=10.1.1.694.8206&rep=rep1&type=pdf

[33] K. Chalkias, J. Brown, M. Hearn, T. Lillehagen, I. Nitto, and T. Schroeter, "Blockchained post-quantum signatures," in *Proc. IEEE Int. Conf. Internet Things (iThings)*, Jul./Aug. 2018, pp. 1196–1203.

[34] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proc. 28th Annu. ACM Symp. Theory Comput.*, Jul. 1996, pp. 212–219, doi: 10.1145/237814.237866.

[35] D. Bernstein, "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete," in *Proc. SHARCS*, Lausanne, Switzerland, Jan. 2009, pp. 105–116.

FURQAN SHAHID received the bachelor's degree in computer science from the Petroman Training Institute, Faisalabad, in affiliation with Allama Iqbal Open University (AIOU) Islamabad, the master's degree in computer science from the University of Agriculture, Faisalabad (UAF), and the M.S. and M.Phil. degrees in software engineering from International Islamic University at Islamabad (IIUI), Islamabad, Pakistan, in 2012. He is currently pursuing the Ph.D. degree in computer science from COMSATS University Islamabad (CUI), Islamabad. His research interests include distributed ledgers, post-quantum cryptography, the IoT, hash-based digital signature schemes, cryptocurrencies, and garbled computing.

IFTIKHAR AHMAD received the B.Sc. degree from Islamia University, Bahawalpur, Pakistan, in 1999, the M.Sc. degree in computer science from the University of Agriculture, Faisalabad, Pakistan, in 2001, the M.S./M.Phil. degree in computer science from the COMSATS Institute of Information Technology, Abbottabad, Pakistan, in 2007, and the Ph.D. degree in information technology from Universiti Teknologi PETRONAS, Malaysia, in 2011. He is currently a Faculty Member with the Information Technology Department, Faculty of Computing and Information Technology. He has been serving as a Faculty Member and a Research Supervisor for various universities, since 2001. He has been involved in several funded projects as PI and Co-PI. He has published several articles in reputed journals and conferences papers. He is also a member of several scientific and professional bodies.

MUHAMMAD IMRAN received the Ph.D. degree in information technology from University Teknologi PETRONAS, Malaysia, in 2011. He is currently an Associate Professor with the College of Applied Computer Science, King Saud University, Saudi Arabia. He has published more than 150 research articles in top conferences and journals. European Alliance for Innovation (EAI) has appointed him as an Editor-in-Chief for *EAI Transactions on Pervasive Health and Technology*. His research interests include the Internet of Things, mobile and wireless networks, big data analytics, cloud computing, and information security. His research is financially supported by several grants. He has completed a number of international collaborative research projects with reputable universities. He has been involved in more than seventy conferences and workshops in various capacities such as a Chair, a Co-Chair, and a Technical Program Committee Member. He also serves as an Associate Editor for reputable international journals such as the *IEEE Communications Magazine*, *Future Generation Computer Systems*, IEEE Access, *Ad Hoc & Sensor Wireless Networks Journal* (SCIE), *IET Wireless Sensor Systems*, *International Journal of Autonomous and Adaptive Communication Systems* (Inderscience). He also served as a Guest Editor for more than a dozen special issues in journals such as the *IEEE Communications Magazine*, *Computer Networks* (Elsevier), *Future Generation Computer Systems* (Elsevier), *MDPI Sensors*, *International Journal of Distributed Sensor Networks* (Hindawi), the *Journal of Internet Technology*, and the *International Journal of Autonomous and Adaptive Communications Systems*.

MUHAMMAD SHOAIB received the B.Eng. and M.Eng. degrees from the NED University of Engineering and Technology, Karachi, in 1995 and 2005, respectively, and the Ph.D. degree in communication and information system from the Beijing University of Posts and Telecommunications, China, in 2010. He was a Senior Manager (IP Operations, South) in Pakistan Telecommunication Company Limited, Pakistan. He was also a Maintenance Engineer with R. M. International. He is currently an Assistant Professor with the Information Systems Department, College of Computer and Information Sciences, King Saud University. His areas of research include video compression techniques, multilayer video coding, commercial Data Center facilities and IP packet-based networks, infrastructure, and security.