WOTS-S

Shahid, Furqan; Khan, Abid; Malik, Saif Ur Rehman; Choo, Kim Kwang Raymond

Published in:
Information Sciences

DOI:
10.1016/j.ins.2020.05.024

Publication date:
2020

Citation for published version (APA):
Shahid, F., Khan, A., Malik, S. U. R., & Choo, K. K. R. (2020). WOTS-S: A Quantum Secure Compact Signature Scheme for Distributed Ledger. *Information Sciences*, *539*, 229-249. https://doi.org/10.1016/j.ins.2020.05.024

WOTS-S: A Quantum Secure Compact Signature Scheme for Distributed Ledger

Furqan Shahid[1], Abid Khan[2*], Saif Ur Rehman Malik[3], Kim-Kwang Raymond Choo[4]

[1]Department of Computer Science, COMSATS University, Islamabad (CUI), Pakistan.

[2]Department of Computer Science, Aberystwyth University, Aberystwyth, SY23 3DB, United Kingdom.

[3]Cybernetica AS, Estonia

[4]Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, TX 78249, USA

Abstract

The digital signature scheme, which underpins most of the existing distributed ledgers, is generally based on non-quantum-resilient algorithms (e.g. elliptic curve digital signature algorithm). This highlights the need for quantum-secure signature schemes in future distributed ledgers (and other products). Therefore, in this paper, we propose a novel quantum-secure digital signature scheme designed specifically for cryptocurrencies. Our proposed scheme is a hash-based signature scheme, which is a variant of Winternitz-one-time signature scheme. A comparison of the proposed scheme and two other competing quantum-secure cryptocurrencies (IoTA and QRL) reveals that our scheme respectively achieves 59% and 24% reductions in signature lengths without compromising the level of security. A salient feature of the proposed approach is that, unlike the previously proposed variants of Winternitz scheme, we avoid the need for any expensive computation. In addition, we formally model the classical cryptocurrency and the proposed quantum-secure cryptocurrency using high-level Petri-nets, which allows the implementer to understand their workings in the presence of a quantum attacker. Furthermore, we also provide formal security proof in the random oracle model.

Key words: Distributed ledger, Cryptocurrency, Digital signature scheme, Quantum secure

1. Introduction

Digital currencies at their core are about creating, representing, storing, and exchanging coins digitally rather physically [1]. The roots of digital currencies can be traced back to 80's in the work of Chaum, which was later followed by a series of innovations and improvements [2, 3]. However, none of the digital currencies proposed before Bitcoin have been successful in practice [4]. Bitcoin is the first widely used cryptocurrency. Despite the many concerns raised by governmental financial institutions and some economists, Bitcoin has captured a significant market and has started to compete against real-world currencies like Dollars and Euros.

---

*Abid Khan [Email: abk15@aber.ac.uk]

[2]Department of Computer Science, Aberystwyth University, Aberystwyth, SY23 3DB, United Kingdom

Due to its increasing popularity, governmental revenue institutions have started planning to tax financial activities involving the use of Bitcoins. Decentralization is a distinguishing feature, which plays an important role in the popularity of Bitcoin. Specifically, the role of a central controlling authority (such as a bank) is eliminated from such a system. Users can create and maintain their accounts at their computers, the balance information of a user is stored in a distributed public ledger called as the *blockchain*, and transactions are verified and accepted by the mutual consensus of the community. Bitcoin network is a peer-to-peer network, in which every peer can access complete history of the transactions which prevents users from double spending attacks. The distributed ledgers are publicly available and tamper-free databases of the financial transactions ever made by the system. The sizes of the distributed ledgers of the popular cryptocurrencies are usually extremely large, for example Bitcoin is reportedly approaching several hundred GBs. Because of the rapidly increasing and ever-growing size of the distributed ledger, the cryptocurrencies are concerned with the size of their financial transactions. The major data elements of a transaction are *ID*, *time*, *inputs*, and *outputs*. An *output* binds some coin(s) to a user's public key, whereas, an *input* redeems an *output* with the help of digital signatures. The size of transaction mainly depends on the size of public key and the size of signatures. Currently, the digital signature scheme used by most of the distributed ledgers is *Elliptic Curve Digital Signature Algorithm (ECDSA)*, which is not quantum-safe [5]. ECDSA is based on the hard mathematical problem *Elliptic Curve Discrete Logarithm Problem (ECDLP),* which is breakable using a sufficiently powerful quantum computer as demonstrated by Shor [6]. Therefore, the present signature scheme of the distributed ledgers must be replaced by a quantum-secure signature scheme to assure survival of this technology in the quantum era. A number of quantum-secure signature schemes have been proposed including, *hash-based schemes, lattice-based schemes, code-based schemes, isogeny-based schemes,* and *multi-variate based schemes* [7]. However, existing quantum-secure signature schemes may not be practical for deployment in distributed ledgers because of certain limitations such as execution times, security level provided, and/or key and signature sizes [8]. The *isogeny-based schemes* and *multi-variate based schemes*, for example, both have extremely large key/signature sizes or execution times. The *code-based schemes* provide just marginal level of post-quantum security. The *hash-based schemes* and *lattice-based schemes* offer appropriate security with an acceptable key and signature size and therefore may be considered as suitable alternates of ECDSA in distributed ledgers [9, 10]. A number of recent post quantum cryptographic schemes leverage the hardness of hash functions or lattice based hard problems (e.g. learning with error - LWE). The latest post quantum cryptographic schemes proposed using lattice-based or hash-based assumptions include, proxy oriented identity based encryption with keyword search (PO-IBEKS) [11], forward secure public key

encryption keywords search (FS-PEKS) [12], identity based key exposure resilient auditing (IB-KERA) scheme [13], and PQ-Chain [9].

Hash-based schemes are quite efficient and offer an appropriate level of security. However, a common problem of hash based digital signature schemes is larger key and signature sizes [14], which must be addressed very carefully before adapting these schemes for distributed ledgers. A hash-based scheme is constructed using a one-time signature (OTS) scheme, which determines key and signature sizes of the corresponding hash based scheme. The key size in a hash-based scheme can be compressed to an acceptable level with the help of hash tress (like XMSS L-tree); however, compressing the signature size remains an open challenge. The current signature sizes of the popular post-quantum cryptocurrencies which use hash based schemes are *3.9 KB* (for *IoTA* [15]) and *2.1 KB* (for *QRL* [16]) respectively, which are significantly larger than Bitcoin (which is just about *1 KB*). The increment in signature size is justified as being a trade-off for post-quantum security. Both IoTA and QRL, for example, offer *128-bit* post-quantum security. IoTA uses the *Winternitz-one time signatures (WOTS)* [17], whereas QRL uses a variant of WOTS (i.e. *WOTS*$^{+}$) [18]. In order to reduce signature size, WOTS^{+} replaces a collision resistant (CR) hash function with a simple undetectable one-way function. For this purpose, WOTS^{+} involves bitmasks and randomizations. However, research shows that CR is actually cheaper to achieve on quantum processors as compared to bitmasks [19].

Therefore, in this paper we propose a new variant of WOTS, which achieves a reduction in the signature size without involving bitmasks or randomizations. Our proposed variant is based on CR hash functions, which avoid any type of quantum-expensive processing. The proposed variant is hereafter referred to as *WOTS-S*, where *S* stands for *short signatures*.

Problem statement: The signature scheme used by most of the distributed ledgers is ECDSA, which is not quantum-safe [5]; therefore the cryptocurrencies would be at significant risk once quantum computers are available at large scale. Existing quantum-secure cryptocurrencies [15, 16], which are using quantum-secure signature schemes, suffer from larger signature sizes. The signature sizes of the two state of the art quantum-secure cryptocurrencies IoTA and QRL are *3.9 KB* and *2.1 KB* respectively, which are significantly greater than non-quantum-resilient cryptocurrencies e.g. Bitcoin. The signature size of the QRL is relatively smaller, however, it uses *bitmasks* and *randomizations*, which are expensive for quantum processors [19].

Our contributions in this paper can be summarized as follows:

1. We propose a new quantum-secure digital signature scheme (*WOTS-S*), which is a variant of the well-known hash-based OTS scheme *WOTS.* The salient features of *WOTS-S* are as follows:

- *WOTS-S* reduces length of signatures by *59%* and *24%* in comparison to *IoTA (WOTS)* and *QRL (WOTS$^+$)*, respectively.
- *WOTS-S* is based on CR hash functions. Therefore, unlike *WOTS$^+$* [18] the design of WOTS-S does not involve any expensive computations in order to reduce the signature size. The existing compact variants of WOTS replace a collision resistant (CR) hash function with a simple undetectable one-way function in order to reduce the signature size. Thus, the existing variants of WOTS, like WOTS$^+$, use bitmasks and randomization operations, which are more expensive for quantum processors than CR hash functions. On the other hand, WOTS-S just uses CR hash functions and adopts inexpensive approaches to achieve a reduction in the signature size.

2. We formally model the classical cryptocurrency and the proposed quantum-secure cryptocurrency using high-level Petri-nets. This helps us understand their workings in the presence of a quantum attacker.

The next two sections briefly review the related literature and the attacker model used in this paper. In Section 4, we describe the proposed scheme and the algorithms. In Section 5, we introduce a new cryptocurrency created using our proposed signature scheme, with the help of High Level Petri-Nets (HLPNs). Finally, in Section 6, we provide security and performance analysis. In the last section, we conclude this paper.

2. Related Work

Chaum proposed the very first digital currency, namely, *blind signatures for untraceable transactions* [20]. Later schemes include universal electronic cash (UEC) [21], revocable versatile electronic money (RVEM) [22], auditable anonymous electronic cash (AAEC) [23], and recoverable untraceable electronic cash (RUEC) [2], etc. The centralized structure of these digital currencies was a major barrier in their adoption, since users rely on a central authority to generate and even facilitate the exchange of the coins. In other words, users are not able to conduct financial transaction (e.g. paying for a cup of coffee) without involving the central authority. The first truly successful digital currency is Bitcoin, which is fully decentralized and is maintained as a public distributed ledger (also known as Blockchain) [4]. The popular decentralized digital currencies proposed after Bitcoin include, Litecoin [24], Ethereum [25], Zerocash [26], and Ripple [27]. These digital currencies are commonly referred to as altcoins. Bitcoin and many of the altcoins use the signature scheme "ECDSA". The existing digital currencies can be classified into three categories i.e. centralized, decentralized, and post-quantum, as shown in Table-1.

2.1. Post-quantum (PQ) cryptocurrencies

The quantum threats to ECDSA invited researchers to replace ECDSA by a quantum-secure signature scheme. Such cryptocurrencies are commonly known as PQ cryptocurrencies. Some of the popular PQ cryptocurrencies include, IoTA [15], QRL [16], Quantum-secured (QS) blockchain [28], qBitcoin [29], PQ-blockchain [10], and Quantum-Bitcoin [30]. Quantum secured blockchain and Quantum-Bitcoin, both are based on quantum technologies and therefore, can be implemented only when quantum computers are available at a large scale. IoTA and QRL are the most popular PQ cryptocurrencies, which are being used at a large scale. Both IoTA and QRL use hash-based signature schemes. PQ-blockchain [10] uses a *Short Integer Solution (SIS)*-based signature scheme. IoTA uses Tangle to maintain history of the transactions. Tangle is a prune-able ledger which offers a mechanism for removal of the un-necessary transactions. Furthermore, Tangle avoids classifying the users into two types, *simple users* and *miners*. Every user is equal in status and *simple users* are not dependent on *miners* for posting of their transactions into the ledger. PQChain [9] is a post quantum blockchain which building blocks include an HBS scheme, hash combiners, and a public key infrastructure (PKI). The overall security of the PQChain depends upon the underlying hash functions, which must be pre-image resistant, pseudorandom, and collision resistant hash functions. Hash combiners allow PQChain to combine two hash functions in a way such that the security is preserved even if one of the two hash functions is compromised. Quantum Bitcoin [30] is a cryptocurrency which uses *no-cloning* theorem as its foundation. The no-cloning quantum mechanics allows to generate a non-forgeable non-copy-able item. A Quantum-Bitcoin is basically a quantum state. During transaction, payer transfers a quantum state to the payee over a quantum channel. The payee can receive a Quantum-Bitcoin without a fear of double-spending from payer side. Quantum-Blockchain is not supposed to store financial transactions, however, just descriptors of the Quantum-Bitcoins are stored in the Quantum-Blockchain.

Table-2 provides a comparative analysis of the existing post-quantum cryptocurrencies.

Table 1: Overview of the existing digital currencies

Reference	Characteristics	Strengths	Weaknesses
		Centralized digital currencies	
[20]	Digital coins	Fully anonymous	No transaction without central authority No coin-divide-ability No coin-recoverability No coin-transferability between accounts
[21]	Un-traceability, Off-line payments, Transferability, and	Fully anonymous	Transactions must finally be verified

	Divide-ability	A coin can be spent in parts Coins can be transferred between accounts	by the central authority No coin-recoverability No blockage of the fake money
[22]	Revocable anonymity	Lost or theft coins can be recovered	Transactions must finally be verified by the central authority Two central authorities (bank and an ombudsman) Partial anonymity
[23]	Auditability, Non-rigidness	Bank is auditable for the generated coins Fake money can be invalidated	No anonymity at all
	Decentralized digital currencies (Cryptocurrencies)		
[4]	Distributed ledger (blockchain), Signature scheme (ECDSA), Hash algorithm (SHA256)	No central authority at all Full user's anonymity	Signature scheme is not quantum-secure Ever-growing size of the blockchain Unusual computational efforts for mining of the new blocks Transaction confirmation delay (10 minutes on average)
[24]	Distributed ledger (blockchain), Signature scheme (ECDSA), Hash algorithm (script)	No central authority Full user's anonymity Faster transaction confirmation (2.5 minutes average)	Signature scheme is not quantum-secure Ever-growing size of the blockchain Memory-intensive hash function used
[25]	Distributed ledger (blockchain), Signature scheme (ECDSA), Hash algorithm (ETHash)	Transaction verification delay reduced to few seconds	Limited scope (not developed primarily as a cryptocurrency but as a distributed applications platform)
[26]	Blockchain-based, ECDSA, SHA256, RSA, and ZKP	Eliminated the known attacks on user's anonymity in the Bitcoin	Increased computational cost due to involvement of additional security algorithms
	Post-quantum Cryptocurrencies		
[15]	Distributed ledger (Tangle), Signature scheme (WOTS), Hash algorithm (keccak-384)	Quantum-secure signature scheme No expensive mining computation Ledger scalability	Involves role of a central coordinator to resist ledger tampering and allow for its pruning No key-reusability Increased signature length
[16]	Blockchain-based, Signature scheme (WOTS+ with XMSS), PRF	Quantum-secure signature scheme Relatively shorter	Ever-growing size of the distributed ledger Heavy mining cost

		signature size Key-reusability for a limited no. of times	
[28]	Blockchain-based, QKD	Quantum-secure No expensive mining	Can not be implemented without a quantum computer Every peer must store and maintain its own copy of the ledger
[30]	No-cloning quantum phenomenon	Quantum-secure No public ledger, no mining at all	Can not be implemented without a quantum computer
[10]	Blockchain-based, Signature scheme (Lattice SIS-based)	Quantum-secure	Extremely large public key length Have never been practically used

Table 2: PQ Cryptocurrencies

Cryptocurrency	**Signature Scheme**	**Ledger Type**	**Characteristics**			
			LP[1]	**RCC[2]**	**IQT[3]**	**RM[4]**
IoTA [15]	W-OTS	Tangle	✓	✓	×	×
QRL [16]	"W-OTS +" with XMSS	Blockchain	×	×	×	✓
QS-blockchain [28]	QKD with Symmetric Key Encryption	Blockchain	×	×	✓	×
qBitcoin [29]	Quantum digital signature	Nil	N/A	×	✓	×
PQ-blockchain [10]	Lattice SIS-based scheme	Blockchain	×	×	×	✓

2.2. Hash-based digital signature schemes

Hash based digital signature (HBS) schemes are computationally-efficient and provably secure, however, suffer from larger key and signature sizes [8, 31].The major building block of a HBS scheme is a one-time signature (OTS) or a few-time signature (FTS) scheme. The signature size of a HBS scheme depends on its underlying OTS/FTS scheme(s). An OTS/FTS scheme is not necessarily to always be a part of a

[1] *Ledger pruning (LP): Removing un-necessary transactions from the distributed ledger*
[2] *Requires central coordinator (RCC): Involvement of a central controlling authority for ledger pruning etc.*
[3] *Involves quantum technologies (IQT): Involvement of the quantum technologies for implementation of the currency*
[4] *Requires miners (RM): Involvement of the extra-ordinary powerful peers to mine transactions into the ledger*

HBS scheme, rather, an OTS scheme can exist independently. IoTA adopted a popular OTS scheme "WOTS", which is an example of the independent existence of WOTS scheme. The popular OTS/FTS schemes proposed to-date include, Lamport-Diffie OTS [32], WOTS [17], WOTS-Buchmann [33], WOTS^{+} [18], HORS FTS [34], HORS-T FTS [35], and PORS FTS [14] schemes. Table-3 provides a comparison of key and signature sizes of popular OTS/FTS schemes. The popular HBS schemes proposed to-date include, MSS [17], XMSS [36], XMSS^{MT} [37], SPHINCS [35], Gravity-SPHINCS [14], SPHINCS-Simpira [38], and blockchained PQ signatures [39]. MSS and XMSS both allow reusing a single public key for just a specific number of times. XMSS^{MT}, SPHINCS and variants of SHPINCS, all allow reusing a single public key for virtually an unlimited number of times. XMSS^{MT} is a state-based HBS scheme, whereas, SPHINCS and its variants are stateless HBS schemes. Gravity-SPHINCS is a compact, whereas, SPHINCS-Simpira is an efficient variant of SHPINCS. SPHINCS-Simpira replaces the simple hash functions by AES-based hash permutations. Blaauwendraad [40] proposed a compact stateful hash based signature scheme. The proposed scheme allows verification of signatures in a chorological order. In order to verify signatures corresponding to a given public key, the verifier needs all of the preceding signatures corresponding to the same public key.

2.3. Blockchain beyond cryptocurrencies:

Although blockchain technology was born with cryptocurrencies, however, recently it is accepted as an independent technology, with a wide range of applications. Beyond cryptocurrencies, the blockchain technology has been used in the construction of smart contracts [24], smart grids [41] and other diverse-natured applications. Very recently, use of blockchain in IoT is very popular [42]. Merging of two technologies is believed to solve security and privacy issues of the cloud-based IoT.

Table 3: Hash-based OTS/FTS schemes

Scheme	Sizes		Security level		Key usage	Parameters used
	Key size	Sig. size	Classical	Quantum		
Lamport [32]	16 KB	8 KB	128-bit	85-bit	One time	SHA-256 hash
W-OTS [17]	2 KB	2 KB	128-bit	85-bit	One time	SHA-256 hash W = 4-bit
W-OTS Buchmann [33]	2.2 KB	2.1 KB	256-bit	128-bit	One time	Salsa-20 PRF M = 256-bit W = 4-bit
W-OTS^{+} (Hulsing) [18]	2.7 KB	2.1 KB	256-bit	128-bit	One time	Salsa-20 PRF

						M = 256-bit W = 4-bit
HORS [34]	20.5 KB	0.3 KB	80-bit	53-bit	Few time	RIPEMD-160 M = 160-bit K = 16
HORS-T [35]	0.3 KB	6.3 KB	128-bit	85-bit	Few time	SHA-256 hash M = 256-bit K = 32 X = 6

3. Attacker Model

The user account in the cryptocurrency is just a private-public key pair, where, funds are received in the public key and spent through the private key. More accurately, funds are transferred to the hash of the public key, not the plain public key, therefore, the plain public key is not revealed to the transferor during transfer of the funds. In order to spend the funds, the owner digitally signs his transaction, which reveals his plain public key to the other users. In a decentralized environment, a new transaction is not submitted to a central authority, rather transactions are verified and accepted by the mutual consensus of the peers. Once a peer initiates a new transaction, the other honest peers verify that transaction and add it to the pool of unconfirmed transactions, which, after consensus of the sufficient *number of peers*, is posted into the blockchain. The time between transaction initiation and its posting into the blockchain is referred to as *transaction processing delay*, which can invite a quantum adversary to steal the funds. Upon receiving a newly initiated transaction, a quantum adversary would be able to compute the private key of the corresponding owner and to initiate another fraudulent transaction to steal his assets. In that case, it will be equally likely that the attacker successfully tricks the peers to accept his fraudulent transaction as genuine, and reject the genuine transaction submitted by the real owner. The attacker can further mine into the blockchain to search for more funds allocated to the same public key and finally to steal all those coins by initiating more fake transactions. We make the following assumptions about the capabilities possessed by an attacker:

1. An attacker may possess a quantum computer, which is sufficiently powerful enough to break the signature scheme “ECDSA”. Furthermore, the attacker has the knowledge of Shor’s algorithm that can be used to break ECDSA and compromise the whole blockchain.

2. The attacker has access to the blockchain with enough expertise to mine useful information such as, list down the unspent transaction outputs (UTXO) for a specific address (*scriptpubkey*) and initiate new fraudulent transactions.
3. The attacker has access to a Bitcoin client. Furthermore, he can perform account creation and initiate new transactions for an arbitrary number of users. This means that the attacker has the knowledge of the hash functions (SHA-256 and RIPEMD-160), number systems, and the error correcting codes (like checksums).
4. The attacker may try to steal the funds of others in order to gain financial benefit at the cost of innocent users. That's why an attacker is always motivated to steal funds of others.

4. Proposed Quantum-secure Signature Scheme

This section covers *WOTS-S*, our proposed variant of the *WOTS* scheme. *WOTS-S* offers the minimum signature size as compared to *WOTS* and existing compact variants of WOTS, including WOTS-Buchmann and WOTS$^+$.

4.1. WOTS-S

WOTS-S, the proposed variant of WOTS, reduces signature size without compromising security of the original scheme. To achieve an appropriate post-quantum security, we use the hash function "*SHA384"* (details are given in the *security analysis* section). We use two sets of values as private key, namely *forward private key* and *backward private key*. While computing public key, we use a *substring* operation with each *hash-iteration*.. First we give an overview of WOTS-S, then we will provide complete pseudocodes for key-generation and signature creation/verification processes.

4.2. Overview of WOTS-S

WOTS-S and other variants of WOTS including WOTS-Buchmann and WOTS$^+$, sign each of the individual hexadecimal character in the message-hash separately. Finally, the concatenations of all those individual signatures represent the overall signatures on the corresponding message. In order to sign an individual hexadecimal character (m) in the message-hash, WOTS computes iterative hash of the corresponding private key (sk) item for m number of times. It is important to note that the number of key-items are exactly equal to the number of hexadecimal characters in the message-hash. The m^{th} hash of the corresponding sk-item represents signature on m. WOTS$^+$ follows the same approach with just one notable difference, i.e. WOTS$^+$ randomizes each of the hash output before using it as an input for the next hash iteration. The randomization operation exempts WOTS$^+$ from using a collision resistant (CR) hash function. Due to randomization operation, it is safe to instantiate WOTS$^+$ with a simple second pre-image resistant, undetectable one-way function. Because a second pre-image resistant, undetectable one-way

function offers *n/2*-bit post-quantum security, whereas, a CR hash function offers just *n/3*-bit post-quantum security [39], therefore, the size (*n*) of an individual signature item decreases; and as a result, the overall signature size decreases.

WOTS-S also generates hash-chains just like WOTS; however, it decreases size of the hash output with each hash iteration in the hash-chain. Therefore, the hash outputs of the later hash iterations in a hash-chain are smaller in sizes as compared to the former hash outputs. The gradually decreasing size of the hash outputs helps WOTS-S in reducing the signature size. Because of the gradually decreasing size of the hash output, the signature size for an individual hexadecimal character (*m*) depends on the number of hash iterations followed to sign the corresponding *m*. And because, the number of hash iterations used to sign an *m* is roughly equal to *m*, therefore, the signature size will be inversely proportional to the corresponding *m*. The size of the corresponding signature-item will be largest if *m* is "*zero*" and smallest if *m* is "*f*". WOTS-S adopts a second approach to allow generating large sized hash-chains even for smaller values of *m*. For this purpose, WOTS-S alters the number of hash iterations required to sign the smaller hexadecimal characters (i.e. *zero* to *seven*). For example, WOTS-S uses maximum number of hash iterations to sign a "*zero*" character (while in WOTS and WOTS$^+$, the *zero* character needs minimum number of hash iterations). WOTS-S uses minimum *nine* (and maximum *sixteen*) hash iterations while signing an individual *m*. WOTS-S binds duplicate values of *m* with a same size of the hash-chain, e.g. in order to sign both of the characters "*zero*" and "*f*", WOTS-S builds a hash-chain of *sixteen* length. This would be a security compromise, however, in order to preserve security, WOTS-S uses two different key chains *forward key* and *backward key*. While signing a "*zero"* character, WOTS-S uses forward private key (*fsk*), whereas, while signing an "*f"* character, WOTS-S uses backward private key (*bsk*). WOTS-S uses *fsk* to sign an *m* in the range *zero* to *seven*; and *bsk* to sign an *m* in the range *eight* to *f*. Finally, while signing an *m* in the range *zero* to *seven*, WOTS-S simply appends the 16th hash of the corresponding *bsk* to the signatures; whereas, while signing an *m* in the range *eight* to *f*, WOTS-S simply appends the 16th hash of the corresponding *fsk* to the signatures. Table-4 differentiates WOTS-S from WOTS and WOTS$^+$.

Table 4: Signature creation: WOTS-S vs. WOTS and WOTS$^+$

$\mathbf{m} \rightarrow$	**0**	**7**	**8**	**f**
σ^{WOTS}	$H^1(sk_i)$	$H^8(sk_i)$	$H^9(sk_i)$	$H^{16}(sk_i)$
σ^{WOTS^+}	$f^1(sk_i \oplus r_i)$	$f^8(sk_i \oplus r_i)$	$f^9(sk_i \oplus r_i)$	$f^{16}(sk_i \oplus r_i)$
σ^{WOTS-S}	$H^{16}(fsk_i)+$ $H^{16}(bsk_i)$	$H^9(fsk_i)+$ $H^{16}(bsk_i)$	$H^9(bsk_i)+$ $H^{16}(fsk_i)$	$H^{16}(bsk_i)+$ $H^{16}(fsk_i)$

4.2.1. Key Generation

The private key consists of two sets *forward private key* and *backward private key*, each containing *99* values (i.e. *198* values in total). All these values are generated from a *seed*, which is randomly selected. The size of *seed* should be *48-bytes* (*384-bits*). The public key is generated by Computing *SHA384* hash of each of the elements in the private key for *17* times. The hash output in each step is truncated in a way such that the size of output gradually decreases in each of the iteration (details are given in Algorithm-1). However, we exempt the last hash output from truncation. This way, we get the public key consisting of *198* values in total, each value being *48*-bytes long. Finally, these values are cryptographically compressed to get a single *48*-bytes long value, which is treated as the ledger address of the user.

4.2.2. Signature creation

To start with, we will compute *SHA384* hash of the transaction to be signed and just like WOTS scheme, append a checksum into it. We process hash of the transaction in its hexadecimal form. *SHA384* generates the transaction-hash consisting of *96* hexadecimal symbols in total, whereas, checksum further appends three more hexadecimal symbols into it. In this way, we get *99* hexadecimal symbols in total. We already have two sets *fsk* and *bsk* each containing *99* values in it. We sign each of the hexadecimal symbols in hash of the transaction using the corresponding element of *fsk* and the corresponding element of *bsk*. Let x_i be the i^{th} hexadecimal symbol in hash of the transaction; If x_i is less than *8* then, we will compute hash of fsk_i for $(16\text{-}x_i)$ number of times, and, we will compute hash of bsk_i for *16* times. However, if x_i is not less than *8* then, the criteria will be different. In that case, we will compute hash of bsk_i for $(x_i + 1)$ number of times, and, we will compute hash of fsk_i for *16* times. While computing hashes of the *sk* elements, the hash-output will be truncated each time, following the same pattern given in Algorithm 1. Thus, at the end, we will get two signature-values against each of the hexadecimal symbols in the hash of transaction, i.e. we get *198* signature-values altogether. We denote these two sets of values as forward

signatures ($f\sigma$) and backward signatures ($b\sigma$). All these values jointly represent signatures on the given transaction.

4.2.3. Signature verification

To begin with, the verifier computes *SHA384* hash of the transaction and appends checksum to it. This way, verifier gets *99* hexadecimal symbols. Verifier already knows signatures, which consists of *198* values in total classified into two sets $f\sigma$ and $b\sigma$. Each of the hexadecimal symbol in the transaction-hash has two signature values associated to it. Verifier computes a verification key *vk*, which, once again, consists of *198* values in total classified into two sets *fvk* and *bvk*. Each of the hexadecimal symbol in the transaction-hash allows verifier to compute the corresponding *fvk* and *bvk* values. Let x_i be the i^{th} hexadecimal symbol in hash of the transaction; If x_i is less than *8* then, verifier will compute hash of $f\sigma_i$ for $(x_i + 1)$ number of times, and, he will compute hash of $b\sigma_i$ for just one time. However, if x_i is not less than *8* then, the criteria will be different. In that case, verifier will compute hash of $b\sigma_i$ for $(16\text{-}x_i)$ number of times, and, he will compute hash of $f\sigma_i$ for just one time. While hashing the σ elements, the hash-output will be truncated each time, following the same pattern given in Algorithm 1. However, the very last hash-output will be exempted from truncation. Therefore, each of the *vk*-value will be *384*-bit long in size. Now, both the sets, i.e. *fvk* and *bvk*, will be cryptographically compressed to finally generate a single *48*-byte value. This value must be already stored in the ledger (with sufficient funds allocated to it), otherwise the transaction will be rejected.

4.3. Incorporating WOTS-S into distributed ledger

This sub-section provides rules and pseudocodes for incorporating WOTS-S into a distributed ledger. Figure-1 explains structure of the transaction. Algorithms *4* - *6* provide pseudocodes for creating, crediting and debiting a wallet account, respectively. Algorithms *7* – *9* provide pseudocodes for transaction initiation, transaction verification, and updating wallet once transaction is accepted for mining. Figure-2 explains life-cycle of a transaction from initiation to its mining into the ledger. In the proposed distributed ledger, a new transaction will be processed in seven different steps. Here is a brief explanation of each of the phases (marked in Figure-2):

1. Owner generates a new transaction (to spend his coins) and transmits to the network.
2. Another peer (acting as verifier) compute hash of the transaction and append checksum into it.
3. The hash of the transaction is manipulated as an array of hexadecimal symbols.
4. Verifier computes verification-key (following Algorithm-3) using hash of the transaction and signatures of the transaction (submitted by the owner as a part of the transaction). Verifier also cryptographically compress the verification-key to get a 48-byte long value.

5. Verifier accesses the blockchain to read the corresponding unspent transaction outputs (XTXO) and compares the compressed verification key with the public-key already stored in the blockchain. Furthermore, the corresponding public-key must be having sufficient funds allocated to it.
6. On a successful verification, the verifier stores the transaction into the pool of unconfirmed transactions.
7. Finally, miner mines the unconfirmed transaction into the blockchain.

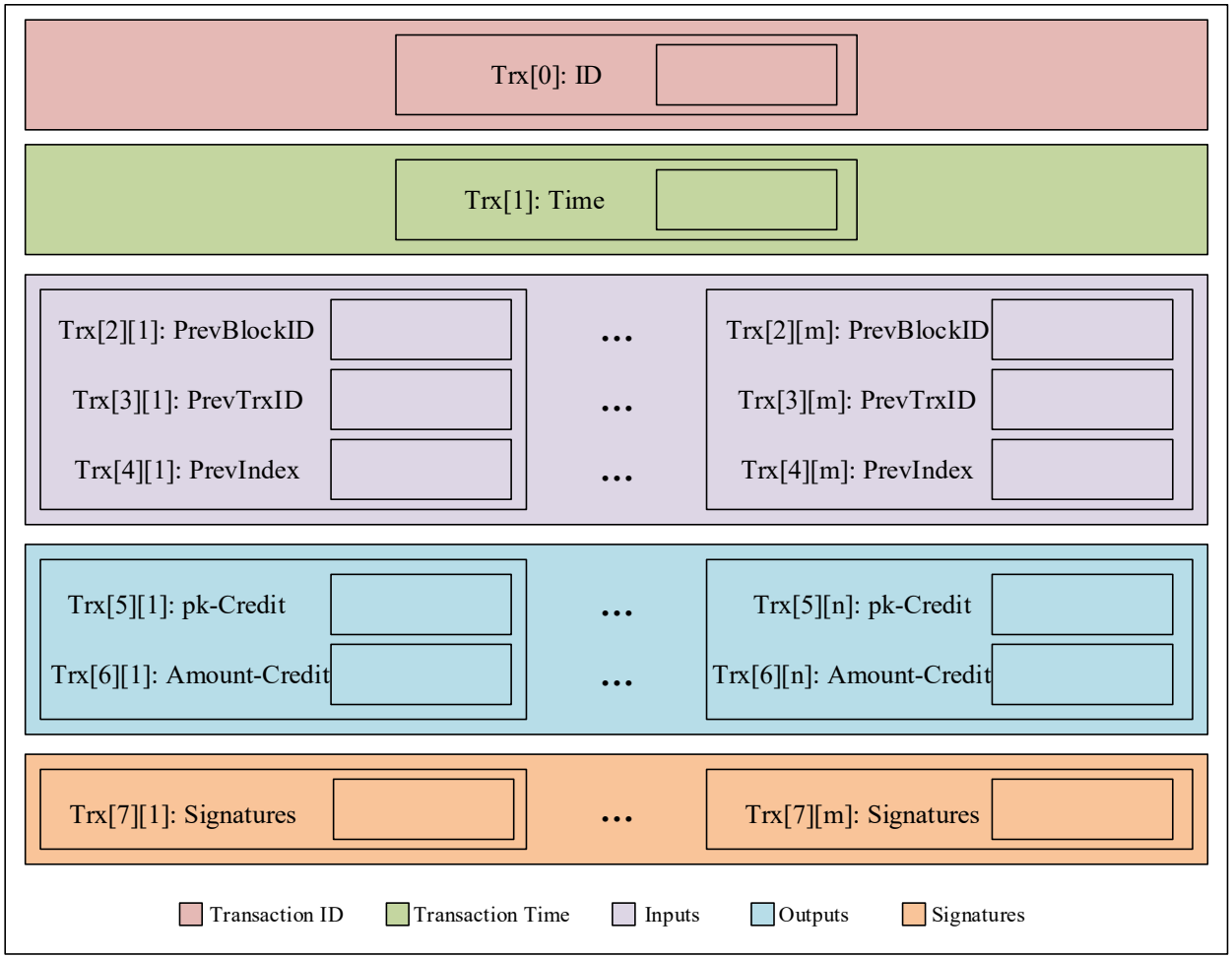

Figure-1: Structure of transaction in Proposed DL

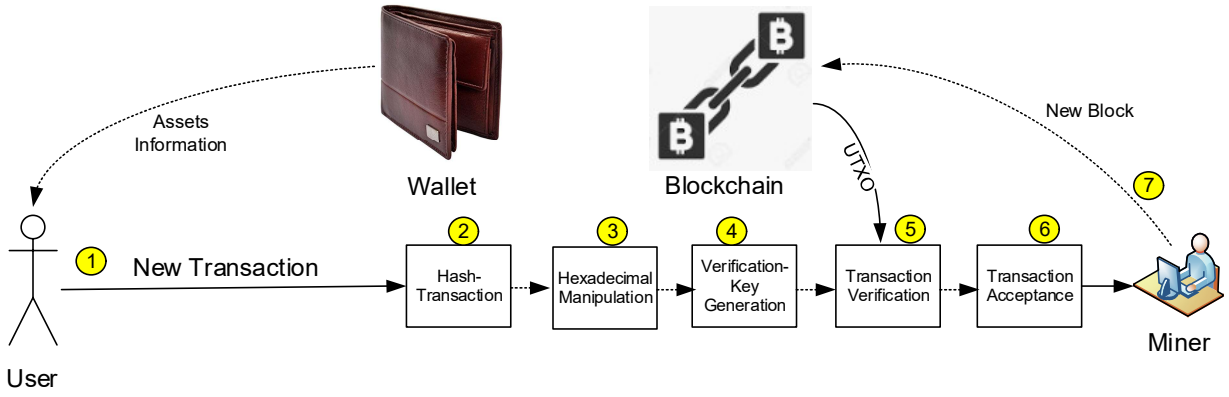

Figure 2: Transaction processing using WOTS-S

Algorithm 1 Key Generation

Input: *Security Parameter* (1^{384})
Output: *Private_Key, Public_Key*

1: $Seed \leftarrow \{0,1\}^{384}$
2: $X \leftarrow Seed$
3: ***for*** $a = 1 \rightarrow 99$ ***do***
4: $X' \leftarrow SHA384(X)$
5: $Forward\_Private\_Key[a] \leftarrow X'$
6: $X \leftarrow X'$
7: ***end for***
8: ***for*** $a = 1 \rightarrow 99$ ***do***
9: $X' \leftarrow SHA384(X)$
10: $Backward\_Private\_Key[a] \leftarrow X'$
11: $X \leftarrow X'$
12: ***end for***
13: ***for*** $a = 1 \rightarrow 99$ ***do***
14: $X \leftarrow SHA384(Forward\_Private\_Key[a])$
15: $j \leftarrow 1$
16: ***for*** $b = 1 \rightarrow 15$ ***do***
17: $X' \leftarrow SHA384(X)$
18: $X'' \leftarrow X'.substring(1, X'.length - j*24)$
19: $X \leftarrow X''$
20: $j \leftarrow j + 1$
21: ***end for***
22: $Forward\_Public\_Key[a] \leftarrow SHA384(X)$
23: ***end for***
24: ***for*** $a = 1 \rightarrow 99$ ***do***
25: $X \leftarrow SHA384(Backward\_Private\_Key[a])$
26: $j \leftarrow 1$
27: ***for*** $b = 1 \rightarrow 15$ ***do***
28: $X' \leftarrow SHA384(X)$
29: $X'' \leftarrow X'.substring(1, X'.length - j*24)$
30: $X \leftarrow X''$
31: $j \leftarrow j + 1$
32: ***end for***
33: $Backward\_Public\_Key[a] \leftarrow SHA384(X)$
34: ***end for***
35: ***for*** $a = 1 \rightarrow 99$ ***do***
36: $All\_Public\_Key[a] \leftarrow SHA384(Forward\_Public\_Key[a] + Backward\_Public\_Key[a])$
37: ***end for***
38: ***for*** $a = 1 \rightarrow 99$ ***do***
39: $PK \leftarrow PK + All\_Public\_Key[a])$
40: ***end for***
41: $Public\_Key \leftarrow SHA384(PK)$
42: $Private\_Key \leftarrow Seed$

Algorithm 2 Signature Creation

Input: *Transaction, Forward_Private_Key[1..96], Backward_Private_Key[1..96]*
Output: *Sig_Part_One, Sig_Part_Two*

```
1:  Trx_Hash[48] ← SHA384(Transaction)
2:  Trx_Hex_Hash[96] ← Trx_Hash[48]
3:  Trx_Hex_Hash[99] ← Trx_Hex_Hash[96] + checksum
4:  for a = 1 → 99 do
5:      m ← Trx_Hex_Hash[a]
6:      if 0 ≤ m ≤ 7 then
7:          X ← Forward_Private_Key[a]
8:          Y ← Backward_Private_Key[a]
9:          j ← 0
10:         for b = 1 → (16 – m) do
11:             X′ ← SHA384(X)
12:             X″ ← X′.substring(1, X′.length – j*24)
13:             X ← X″
14:             j ← j + 1
15:         end for
16:     else
17:         X ← Backward_Private_Key[a]
18:         Y ← Forward_Private_Key[a]
19:         j ← 0
20:         for b = 1 → (m + 1) do
21:             X′ ← SHA384(X)
22:             X″ ← X′.substring(1, X′.length – j*24)
23:             X ← X″
24:             j ← j + 1
25:         end for
26:     end if
27:     Sig_Part_One ← Sig_Part_One + X
28:     j ← 0
29:     for b = 1 → 16 do
30:             Y′ ← SHA384(Y)
31:             Y″ ← Y′.substring(1, Y′.length – j*24)
32:             Y ← Y″
33:             j ← j + 1
34:     end for
35:     Sig_Part_Two ← Sig_Part_Two + Y
36: end for
```

Algorithm 3 Signature Verification

Input: *Transaction, Sig_Part_One, Sig_Part_Two, Public_Key*
Output: *Verified/Failed*

```
Trx_Hash[48] ← SHA384(Transaction)
Trx_Hex_Hash[96] ← Trx_Hash[48]
Trx_Hex_Hash[99] ← Trx_Hex_Hash[96] + checksum
pointer1← pointer2 ← 1
for a = 1 → 99 do
    m ← Trx_Hex_Hash[a]
    if 0 ≤ m ≤ 7 then
        S ← Sig_Part_One.substring(pointer1, (m + 1) * 24)
        j ← 0
        for b = 1 → m do
            S′ ← SHA384(S)
            S″ ← S′.substring(1, S′.length – (16 – m + j) * 24)
            S ← S″
            j ← j + 1
        end for
        Forward_Public_Key[a] ← SHA384(S)
        S ← Sig_Part_Two.substring(pointer2, 24)
        Backward_Public_Key[a] ← SHA384(S)
        pointer1 ← pointer1 + (m + 1) * 24
        pointer2 ← pointer2 + 24
    else
        S ← Sig_Part_One.substring(pointer1, (16 - m) * 24)
        j ← 0
        for b = 1 → 15 - m do
            S′ ← SHA384(S)
            S″ ← S′.substring(1, S′.length – (m + 1 + j) * 24)
            S ← S″
            j ← j + 1
        end for
        Backward_Public_Key[a] ← SHA384(S)
        S ← Sig_Part_Two.substring(pointer2, 24)
        Forward_Public_Key[a] ← SHA384(S)
        pointer1 ← pointer1 + (16 - m) * 24
        pointer2 ← pointer2 + 24
    end if
end for
for a = 1 → 99 do
    All_Public_Key[a] ← SHA384(Forward_Public_Key[a] + Backward_Public_Key[a])
end for
for a = 1 → 99 do
    PK ← PK + All_Public_Key[a])
end for
if SHA384(PK) == Public_Key then output: Verified else output: Failed
```

Algorithm 4 *Create-wallet-account*

Inputs: *Current-wallet*
Outputs: *Updated-wallet*

```
(sk, pk) ← WOTS-S.keyGeneration ( )
balance ← 0
Create new entry into wallet with (sk, pk, balance)
Print (pk)
Output ("Account created") and return
Show-Error ("Account creation failed")
```

Algorithm 5 *Credit-account*

Inputs: *Credit-sk, Amount*
Outputs: *Updated-Wallet*

```
If Credit-sk ϵ Wallet then
    balance = balance + Amount where sk = Credit-sk
Else
    Show-Error ("Account not found")
End if
```

Algorithm 6 *Debit-account*

Inputs: *sk, Amount*
Outputs: *Updated-Wallet*

```
If sk ϵ Wallet then
    sk.balance = sk.balance - Amount
Else
    Show-Error ("Account not found")
End if
```

Algorithm 7 *Initiate-transaction*

Inputs: *Input-information, Output-information, Private-key (sk)*
Outputs: *New-transaction*

```
Trx[1]← Current-time
For i = 1 → number-of-inputs do:
    Trx[2][i].← previous-blockID
    Trx[3][i].← previous-trxID
    Trx[4][i].← previous-index
End For
For i = 1 → number-of-outputs do:
    Trx[5][i].← pk-to-be-credited
    Trx[6][i].← Amount
End For
For i = 1 → number-of-inputs do:
    Trx[7][i].← WOTS-S.sign(Trx, sk)
End For
Trx[0] ← SHA384 (Trx)
```

Algorithm 8 *Verify-transaction*

Inputs: *Transaction-to-be-verified (Trx)*
Outputs: *Valid/Invalid*

```
Trx[1]=Current-time
available-amount ← 0
For i = 1 → number-of-inputs do:
    Read amount from ledger where:
        blockID == Trx[2][i] and trxID == Trx[3][i] and index == Trx[4][i]
    available-amount ← available-amount + amount
End For
amount-to-spend ← 0
For i = 1 → number-of-outputs do:
    amount-to-spend ← amount-to-spend + Trx[6][i].
End For
If amount-to-spend < available-amount then:
    Print ("In-sufficient funds") and return
End If
For i = 1 → number-of-inputs do:
    If WOTS-S.verify(Trx, Trx[7]) == "Failed" then:
        Print ("Invalid") and return
    End If
End For
Print ("Valid")
```

Algorithm 9 *Update-wallet*

Inputs: *Confirmed-transaction (Trx)*
Outputs: *Updated-wallet*

```
For i = 1 → number-of-inputs do:
    Read pk into ledger-pk from ledger where:
        blockID == Trx[2][i] and trxID == Trx[3][i] and index == Trx[4][i]
    If ledger-pk ϵ wallet then:
        balance = 0 where pk = ledger-pk
    End If
End For
For i = 1 → number-of-outputs do:
    If Trx[5][i] ϵ wallet then:
        Set balance = balance + Trx[6][i] where pk = Trx[5][i]
    End If
End For
```

5. Formal definition of WOTS-S based Cryptocurrency

This section formally defines a new cryptocurrency, created using our proposed signature scheme (WOTS-S). Two different HLPNs have been created, one for Bitcoin (Figure 3), and second for the proposed cryptocurrency (Figure 4). Both of the HLPNs formally define the rules of transaction

processing into the corresponding cryptocurrency. The HLPNs also differentiate role of a quantum adversary in Bitcoin and the proposed cryptocurrency. In case of Bitcoin, the adversary can successfully sign his tampered transaction and therefore, can steal coins of others. However, WOTS-S based cryptocurrency prevents adversary from signing the tampered transaction.

5.1. Transaction processing and possible quantum-attack in Bitcoin

In decentralized cryptocurrencies, the peers verify a new transaction and after sufficient number of verifications, the transaction is posted into the ledger by the miners. The HLPN in Figure-3 explains different steps involved in the verification of a new transaction. First, each new transaction must be referred by an old *unspent transaction*. The place $RefVerTrx$ contains transactions, which are referred by a valid past transaction; and token $T1$ denotes these transactions. The rules for both successful and unsuccessful *reference verification* are given in Eq. (1) and (2), respectively.

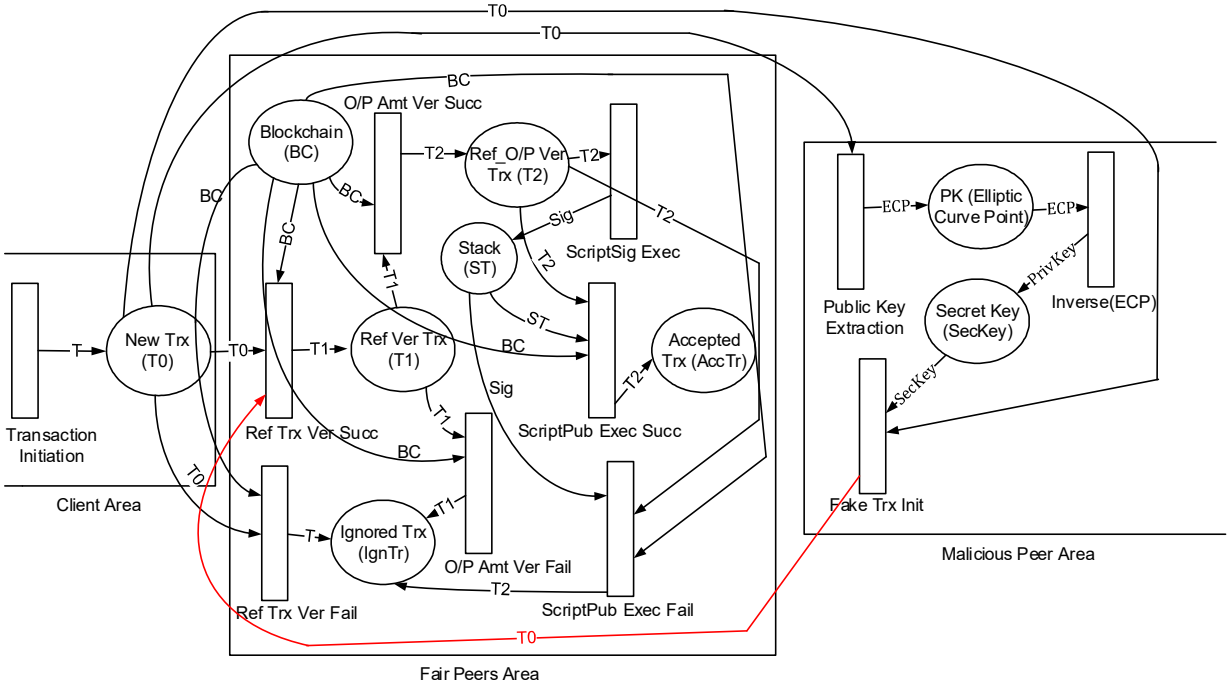

Figure 3: HLPN for classical cryptocurrencies (Bitcoin)

Table 5: Data-types mapping to places for Bitcoin's HLPN

Place	Description
$\emptyset(NewTrx)$	$\{TrxTime, RefTrxID, ScriptSig, OutAmt, OutAdress\}$
$\emptyset(Blockchain)$	$\mathbb{P}\{TrxID, TrxTime, UnspentAmt, ScriptPubKey, Status\}$

$\emptyset(RefVerTrx)$	$\{TrxTime, RefTrxID, ScriptSig, OutAmt, OutAdress\}$
$\emptyset(RefOPVerTrx)$	$\{TrxTime, RefTrxID, ScriptSig, OutAmt, OutAdress\}$
$\emptyset(IgnoredTrx)$	$\mathbb{P}\{TrxTime, RefTrxID, ScriptSig, OutAmt, OutAdress\}$
$\emptyset(AcceptedTrx)$	$\mathbb{P}\{TrxTime, RefTrxID, ScriptSig, OutAmt, OutAdress\}$
$\emptyset(Stack)$	$(STACK)$
$\emptyset(PK\_ECP)$	$(Elliptic\ Curve\ Point)$
$\emptyset(SecretKey)$	$(Elliptic\ Curve\ Point)$

$$\boldsymbol{R}\ (\boldsymbol{RefTrxVerSucc}) = (\exists\ b \in BC \bullet (b[1] = T0\ [2] \wedge b\ [5] = \text{`}Unspent\text{'})) \wedge (T1' = T1 \cup \{T0[1], T0[2], T0[3], T0[4], T0[5]\}) \quad \textbf{(1)}$$

$$\boldsymbol{R}(\boldsymbol{RefTrxVerFail}) = (\forall\ b \notin BC \bullet (b[1] = T0\ [2]) \vee (\exists\ b \in BC \bullet (b[1] = T0\ [2] \wedge b[5] \neq \text{`}Unspent\text{'})) \wedge (IgnTr' = IgnTr \cup \{T0[1], T0[2], T0[3], T0[4], T0[5]\}) \quad \textbf{(2)}$$

Secondly, the total amount to be spent must not exceed the available amount. The place $Ref\_OPVerTrx$ stores the transactions which qualify both of the checks, *valid reference* and *sufficient available funds* and token *T2* denotes such transactions. Eq. 3 and 4 define rules for availability of the funds.

$$\boldsymbol{R}(\boldsymbol{OutAmtVerSucc}) = (\exists\ b \in BC \bullet (b[1] = T1\ [2] \wedge b[3] >= T1[4]) \wedge (T2' = T2 \cup \{T1[1], T1[2], T1[3], T1[4], T1[5]\}) \quad \textbf{(3)}$$

$$\boldsymbol{R}(\boldsymbol{OutAmtVerFail}) = (\exists\ b \in BC \bullet (b[1] = T1\ [2] \wedge b\ [3] < T1[4])) \wedge (IgnTr' = IgnTr \cup \{T1[1], T1[2], T1[3], T1[4], T1[5]\}) \quad \textbf{(4)}$$

The owner proves his ownership over the coins with the help of a script known as '*scriptsig*'. Another script namely, '*scriptpubkey*' is already stored in the blockchain. During verification process, the verifier executes both of the scripts. First, verifier executes *scriptsig* and stores results into a stack (Eq. 5). The second script (i.e. *scriptpubkey*) uses output of *scriptsig* as input. The output of the second script must be equal to one, otherwise transaction verification will be failed. (See Eq. 6 and 7).

$$\boldsymbol{R}(\boldsymbol{ScriptSigExec}) = (ST' = ST \cup Execute(T2[3])) \quad \textbf{(5)}$$

$$\boldsymbol{R}(\boldsymbol{ScriptPubExecSucc}) = (\exists\ b \in BC \bullet (b[1] = T2[2] \wedge Execute(b[4], ST) = 1) \wedge (AccTr' = AccTrx \cup \{T2[1], T2[2], T2[3], T2[4], T2[5]\}) \quad \textbf{(6)}$$

$$\boldsymbol{R}(\boldsymbol{ScriptPubExecFail}) = (\exists\ b \in BC \bullet (b[1] = T2[2] \wedge Execute(b[4], ST) \neq 1)) \wedge (IgnTr' = IgnTr \cup \{T2[1], T2[2], T2[3], T2[4], T2[5]\}) \quad \textbf{(7)}$$

The HLPN also highlights the quantum-attack scenario on Bitcoin. A quantum adversary being able to deduce private key of the owner can initiate a fraudulent transaction. The adversary exploits *scriptsig* to deduce the private key (Eq. 8 – 9). The fraudulent transaction can deprive the real owner from his/her legitimate assets (Eq. 10)

$$\boldsymbol{R}(\boldsymbol{PublicKeyExtraction}) = (ECP' = ECP \cup PKExtract(T0[3])) \quad (8)$$

$$\boldsymbol{R}(\boldsymbol{InverseECP}) = (SecKey' = SecKey \cup ShorAlgo(ECP)) \quad (9)$$

$$\boldsymbol{R}(\boldsymbol{FakeTrxInit}) = (T0' = T0 \cup \{T0[1], T0[2], ScriptSigGen(SecKey), T0[4], T0[5]'\}) \quad (10)$$

5.2. Transaction processing in the proposed cryptocurrency

The proposed WOTS-S based ledger adopts hash-based rules for signature verification. Verifier computes hash of the transaction (Eq. 11), and follows the signatures (provided by the owner) to generate the corresponding public key (Eq. 12). Then verifier compresses the public key to generate a verification key (Eq. 13) The verification key must already be stored in the ledger, otherwise transaction verification will be failed (Eq. 14 – 15).

$$\boldsymbol{R}(\boldsymbol{TrxHashing}) = (TrHash' = TrHash \cup SHA384(T2[1], T2[2], T2[3], T2[4], T2[5])) \quad (11)$$

$$\boldsymbol{R}(\boldsymbol{PublicKeyGen}) = (PK\_Set' = PK\_Set \cup GeneratePublicKey(T2[6])) \quad (12)$$

$$\boldsymbol{R}(\boldsymbol{Vf\_PK\_Gen}) = (PK\_Vf' = PK\_Vf' \cup GenerateVerificationPK(PK\_Set)) \quad (13)$$

$$\boldsymbol{R}(\boldsymbol{SigVerSucc}) = (PK\_Vf = T2[3]) \wedge AccTrx' = AccTrx \cup \{T2[1], T2[2], T2[3], T2[4], T2[5], T2[6]\} \quad (14)$$

$$\boldsymbol{R}(\boldsymbol{SigVerFail}) = (PK\_Vf \neq T2[3]) \wedge IgnTrx' = IgnTrx \cup \{T2[1], T2[2], T2[3], T2[4], T2[5], T2[6]\} \quad (15)$$

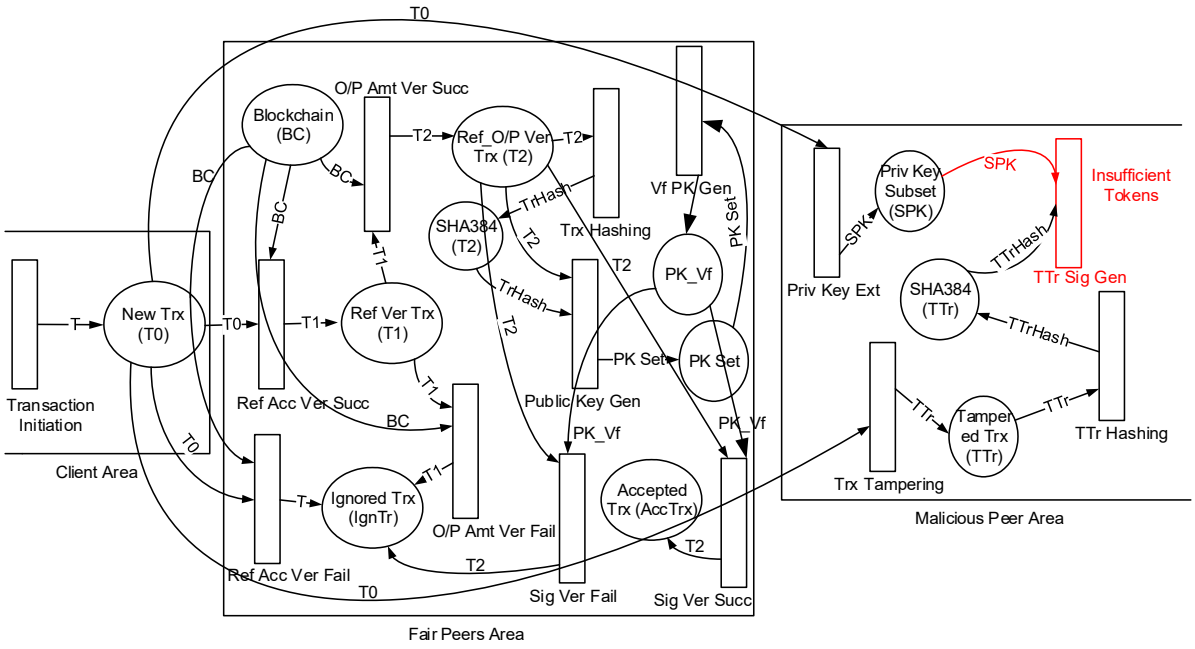

Figure 4: HLPN for our proposed cryptocurrency (with attack scenario)

Table 6: Data-types mapping to places for our proposed scheme's HLPN

Place	Description
$\emptyset(NewTrx)$	$\{TrxTime, RefTrxID, DbAccPubKey, OutAmt, CrAccPubKey, TrxSig\}$
$\emptyset(Blockchain)$	$\mathbb{P}\{TrxID, TrxTime, UnspentAmt, AccPubKey, Status\}$
$\emptyset(RefVerTrx)$	$\{TrxTime, RefTrxID, DbAccPubKey, OutAmt, CrAccPubKey, TrxSig\}$
$\emptyset(RefOPVerTrx)$	$\{TrxTime, RefTrxID, DbAccPubKey, OutAmt, CrAccPubKey, TrxSig\}$
$\emptyset(IgnoredTrx)$	$\mathbb{P}\{TrxTime, RefTrxID, DbAccPubKey, OutAmt, CrAccPubKey, TrxSig\}$
$\emptyset(AcceptedTrx)$	$\mathbb{P}\{TrxTime, RefTrxID, DbAccPubKey, OutAmt, CrAccPubKey, TrxSig\}$
$\emptyset(SHA384)$	$(384 - bit\ length\ string)$
$\emptyset(PK\_Set)$	$(48x64x2\ byte\ string)$
$\emptyset(PK\_Vf)$	$(48\ byte\ string)$
$\emptyset(TamperedTrx)$	$\{TrxTime, RefTrxID, DbAccPubKey, OutAmt, CrAccPubKey\}$
$\emptyset(SecKeySubset)$	$(A\ proper\ subset\ of\ 48x64x2\ byte\ string)$

If adversary tampers the transaction, the hash of the transaction will change and therefore, signature will sustain no more to be valid (Eq. 16, 17). WOTS-S based signatures do not allow adversary to deduce enough knowledge about the private key, required to sign the tampered transaction (Eqs. 18, 19).

$$\boldsymbol{R(TrxTampering)} = (TTr' = TTr \cup \{T0[1], T0[2], T0[3], T0[4], T0[5]', T0[6]\}) \quad \textbf{(16)}$$

$$\boldsymbol{R(TTrHashing)} = (TTrHash' = TTrHash \cup SHA384(TTr[1], TTr[2], TTr[3], TTr[4], TTr[5])) \quad \textbf{(17)}$$

$$\boldsymbol{R(SecKeyExt)} = (SSK' = SSK \cup ExtractSecKey(T0[6])) \quad \textbf{(18)}$$

$$\boldsymbol{R(TTrSigGen)} = (TTr[6]' = TTr[6] \cup GenerateSig(TTrHash, SK)) \text{ // SK is not available} \quad \textbf{(19)}$$

6. Security and performance analysis

This section compares WOTS-S with existing state of the art distributed ledgers IoTA and QRL. Subsection 6.1 provides a formal security proof of WOTS-S. Subsection 6.2 compares signature size of WOTS-S with IoTA and QRL. Finally, Subsection 6.3 compares computational efficiency of WOTS-S with WOTS (IoTA) and existing variants of WOTS, like WOTS$^+$ (QRL).

6.1. Security analysis of WOTS-S

First we define security preliminaries, and then we will evaluate and prove security of WOTS-S.

6.1.1. Preliminaries:

WOTS-S is a hash-based scheme, which uses a hash function as its building block. The underlying hash function of WOTS-S must be committing an appropriate level of security. The three fundamental characteristics of a secure hash function include, *preimage resistance*, *second pre-image resistance*, and *collision resistance*. Equations 20 – 22 explains the three characteristics, respectively. A *pre-image resistant* hash function does not allow an adversary (A) to deduce an input 'x' which corresponding output 'y' is known to him. A *second pre-image resistant* hash function does not allow the adversary to deduce an input x' corresponding to an output 'y' when adversary already knows another input *'x'* corresponding to the same output i.e. *'y'*. A *collision-resistant* hash function does not allow the adversary to find any two inputs *'x'* and x' which are not equal but their corresponding output is equal.

$$Pr\,[y \leftarrow h(x), x' \leftarrow A(y): x = x'] < \in \quad \textbf{(20)}$$

$$Pr\,[y \leftarrow h(x), x' \leftarrow A(x, y): x \neq x' \wedge y = h(x')] < \in \quad \textbf{(21)}$$

$$Pr\,[x, x' \leftarrow A: x \neq x' \wedge h(x) = h(x')] < \in \quad \textbf{(22)}$$

The classical and quantum security levels offered by a hash function depend upon output length (n) of that function [31]. The quantum-based Grover's search algorithm [43] reduces post-quantum security

level of the hash functions. An *n*-sized hash function is capable of providing *n*-bit classical and *n/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, an *n*-sized hash function provides *n/2*-bit classical and *n/3*-bit post-quantum security against collision-based attacks [39]. Table-7 lists down the classical and post-quantum security levels of the common hash functions.

Table7: Hash functions security levels [31, 39]

Output length	Classical security level (bit)		Quantum security level (bit)	
	Pre-image/ 2nd Pre-image	Collision	Pre-image/2nd Pre-image	Collision
160-bit	160	80	80	53
256-bit	256	128	128	85
384-bit	384	192	192	128
512-bit	512	256	256	171

6.1.2. Security properties of WOTS-S

WOTS-S is a quantum-secure one time signature scheme, which achieves *integrity*, *authentication*, and *non-repudiation*. The public key is a set of post-images, which corresponding pre-images are known solely to the owner of the public key. Any assets allocated to a *public key* are owned by the person possessing knowledge of the corresponding *private key*. An effort to spend the assets without the appropriate *private key*, will be noticeable and rejected by the peers (*integrity* and *authentication*). Similarly, the assets once spent with the help of the appropriate *private key* will no more be claimable by the spender (*non-repudiation*).

We will prove that WOTS-S is an existentially unforgeable digital signature scheme under adaptive chosen plaintext attack (CPA) model. Our proof reduces security of WOTS-S to the *onewayness* of its underlying hash function.

6.1.3. Formal security proof of WOTS-S

This section formally proves that WOTS-S is existentially unforgeable under CPA) model as long as the underlying hash function is a *one-way* hash function. WOTS-S is a triple (*KEYGEN*, *SIGN*, *VERIFY*); *KEYGEN* is a key generation algorithm which takes a security parameter (*n*) as input and outputs a key pair (*SK*, *PK*), where, both *SK* and *PK* are sets with $|SK| = |PK| = (n/4)*2$. *KEYGEN* generates a hash-chain consisting of *seventeen* hash iterations in order to transform an *sk*-value into the corresponding *pk*-

value. *SIGN* takes a message (M) and a private key $SK = \{(sk_1^f, sk_1^b), (sk_2^f, sk_2^b) \dots (sk_{n/4}^f, sk_{n/4}^b)\}$ as input and outputs signatures of M, i.e. $\boldsymbol{\sigma^M} = \{(\sigma_1^f, \sigma_1^b), (\sigma_2^f, \sigma_2^b) \dots (\sigma_{n/4}^f, \sigma_{n/4}^b)\}$. *SIGN* distributes M as $(m_1, m_2, m_3 \dots m_{n/4})$ and signs each m_i individually. Each of the σ_i is some of the middle value of the corresponding hash-chain such that:

$\sigma_i^f = H^{16-mi}(sk_i^f)$; $\sigma_i^b = H^{16}(sk_i^b)$ *for* $0 \le m_i \le 7$, *and*

$\sigma_i^f = H^{16}(sk_i^f)$; $\sigma_i^b = H^{m+1}(sk_i^b)$ *for* $8 \le m_i \le 15$

VERIFY takes M, $\boldsymbol{\sigma^M}$, and *PK* as input and outputs either *TRUE* or *FALSE*. *VERIFY* uses (m_i, σ_i) pairs to complete each of the corresponding hash-chain. *VERIFY* compares final value of each of the hash-chain to the corresponding pk_i and outputs *TRUE* if and only if each of the final hash-chain result matches the corresponding *pk*-value.

Existential unforgeability of WOTS-S:

KEYGEN generates a new key pair (*SK*, *PK*). A signing oracle O having knowledge of the private key (*SK*), responds the forger's queries. Forger *FOR* can submit at most one query to O. *FOR* has knowledge of *PK*. Upon receiving a query from *FOR*, O must return valid signatures of the queried message M^q (i.e. σ^{Mq}). The challenge for the *FOR* is to return a message/signature pair (M^o, σ^{Mo}) such that σ^{Mo} are valid signatures of M^o whereas, $M^o \neq M^q$. WOTS-S is existentially unforgeable under CPA model if the probability that the *FOR* wins the above game in a time t, is at most ε. We formally write it as, WOTS-S is a $(t, \varepsilon, 1)$-existentially unforgeable signature scheme.

The security proof:

This section formally proves that the security of WOTS-S is a security reduction of its underlying hash function. Let $H = \{h: (0,1)^* \rightarrow (0,1)^n\}$ be a family of secure hash functions, and *ADV* be an adversary that breaks onewayness of the functions in H. The *ADV* takes an image y as input such that, $y = h(x)$ and $x = (0, 1)^n$. The challenge for *ADV* is to return a pre-image x^{ADV} such that, $x^{ADV} = x$. Algorithm-10 explains that how $ADV_{\text{onewayness}}$ can exploit $FOR_{\text{WOTS-S}}$ to win the game. $ADV_{\text{onewayness}}$ calls the *KEYGEN* algorithm to generate a key pair (*SK, PK*). The *PK* is of the form $PK = \{(pk_1^f, pk_1^b), (pk_2^f, pk_2^b) \dots (pk_{n/4}^f, pk_{n/4}^b)\}$. $ADV_{\text{onewayness}}$ alters a randomly chosen element (pk_β^f) of the public key (Lines 2, 3). Then $ADV_{\text{onewayness}}$ runs the forger $FOR_{\text{WOTS-S}}$. When $FOR_{\text{WOTS-S}}$ queries a message M then, $ADV_{\text{onewayness}}$ either returns valid signature (σ) of M or aborts the algorithm (Line 5). When $FOR_{\text{WOTS-S}}$ returns a message/signature pair $(M', \sigma^{M'})$ to the $ADV_{\text{onewayness}}$ then, depending upon the value of m'_β, either $ADV_{\text{onewayness}}$ returns the challenged pre-image or aborts the algorithm (Line 6).

Algorithm 10: $ADV_{onewayness}$

Input: Security parameter n, a one-way hash-function h, a post-image y such that $y=h(?)$, an adversary ADV to break onewayness of h, a forger FOR to break WOTS-S

Output: an x such that $h(x) = y$

1. Generate a new WOTS-S key pair (SK, PK)
2. Randomly choose a $\beta \in \{1,\ldots,n/4\}$
3. Generate a random x and alter pk_{β}^{f} as $h(x[0,n/128))$
4. Run forger FOR
5. **When** FOR asks for a signature-query on a message M *i.e.* $(m_1,\ldots,m_{n/4})$ **then:**
    a. **If** $1 \leq m_{\beta} \leq 7$ **then return *fail***
    b. Generate σ on M and respond to FOR as (M,σ^{M})
6. **When** FOR return an $(M',\sigma^{M'})$ pair **then**
    a. **If** $\sigma^{M'}$ is a valid signature of M' **then**
        i. **If** $m'_{\beta} = 0$ or $8 \leq m'_{\beta} \leq 15$ **then return *fail***
        ii. **Return** $SIGN_{WOTS\text{-}S}^{(m'_{\beta}-1)}(\sigma^{M'}{}_{\beta})$
7. In any other cases **return *fail***

For the adversary to be successful, he must 1) be able to respond forger's query [step 5b], 2) receive a valid message/signature pair from the forger, *and* 3) be able to generate the challenged pre-image from message-signature pair retuned by the forger [step 6a]. Because the success probabilities of both the steps (5b and 6a) are non-zero, therefore, the success probability of the forger must be negligible. A non-negligible success probability of forger means a non-negligible success probability of adversary, which is impossible because '*h*' is a one-way hash function.

6.2. Signature size in WOTS-S

The signature size in WOTS-S, may be as less as *0.58 KB* or as greater as *2.6 KB*. Thus the average signature size in WOTS-S is *1.6 KB*, which is less than both IoTA and the QRL. IoTA uses the signature scheme "*WOTS"*, which signature size (for *128-bit* post-quantum security), is *4.6 KB*. However, because IoTA has a tryte-based implementation, therefore IoTA's signature size is *3.9 KB*, which is somehow less than the normal WOTS size. QRL uses WOTS^{+}, which signature size is *2.1 KB*. Thus WOTS-S achieves *59%* and *24%* reductions in signature sizes than IoTA and QRL respectively. Figure-5 provides a comparison of signature sizes of WOTS (IoTA) and its popular variants including WOTS^{+} (QRL).

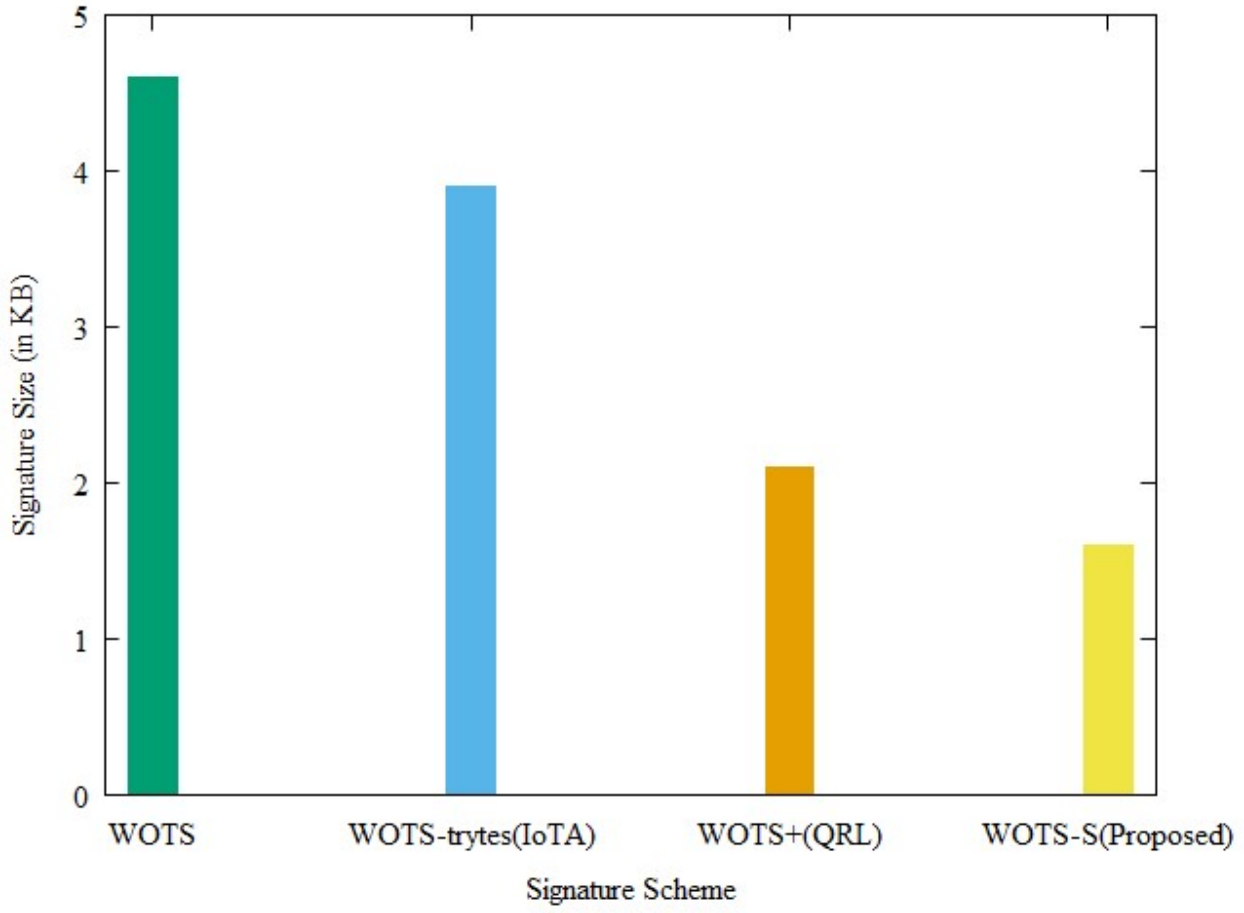

Figure 5: Signature sizes for 128-bit post-quantum security

6.3. Executional efficiency evaluation of WOTS-S

WOTS-S in not only a compact but also an efficient variant of WOTS scheme. The implementation results reveal that WOTS-S offers *70%* and *51%* reductions in key-generation time as compared to WOTS+ and WOTS_Buchmann, respectively. Furthermore, WOTS-S offers *63%* and *29%* reduction in signature creation time as compared to WOTS+ and WOTS_Buchmann, respectively. For our experiments, we used a test bed consisting of an Intel core i5 CPU (2.4 GHz) with 4GB RAM, running Windows 8.1 32-bit release. We used Python language with PyCharm IDE for implementing WOTS-S and the benchmarked schemes. Figure-6 provides a comparison of execution times for key generation, signature creation, and signature verification algorithms. Figures 7 – 9 reveal execution times for the three algorithms for a number of function calls (up to *500*).

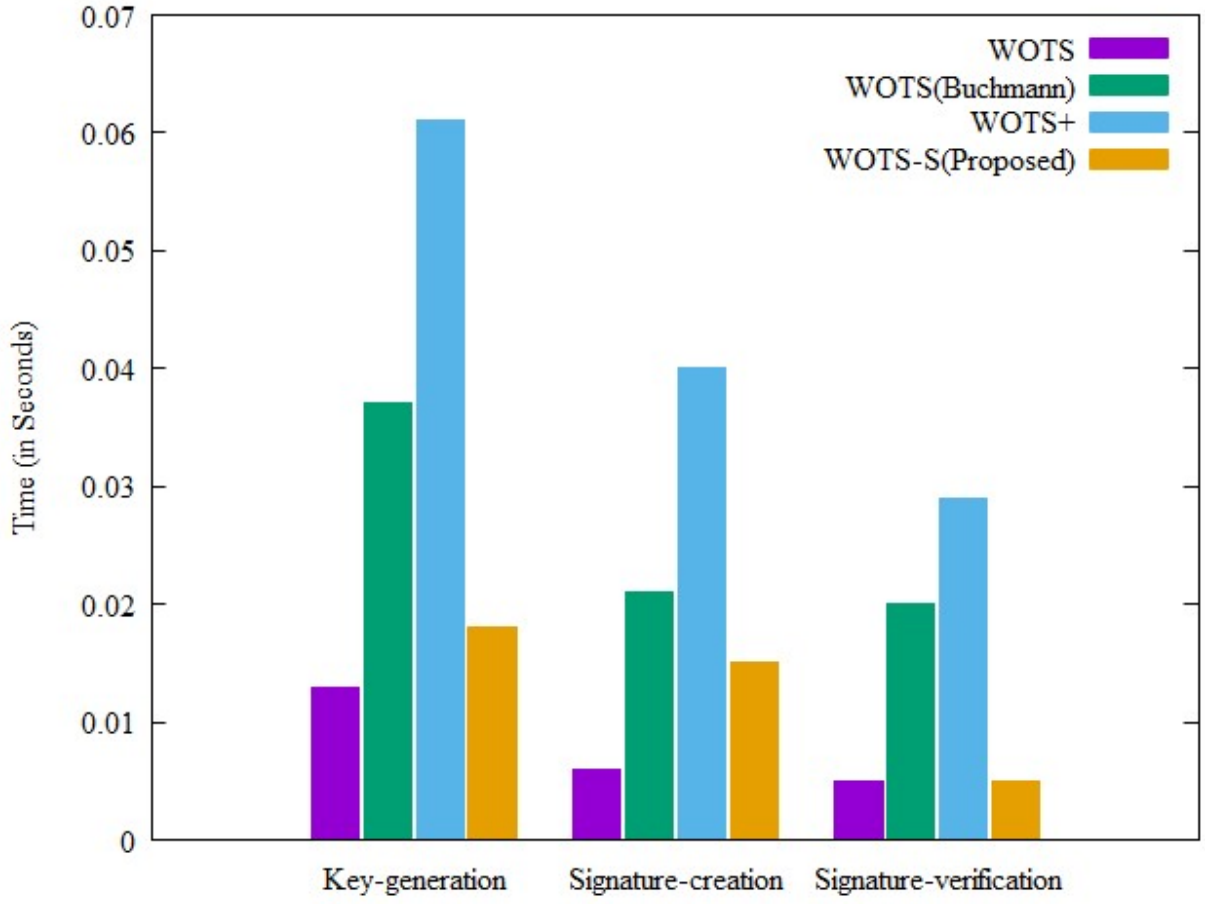

Figure 6: Time for key generation, signature creation and verification (single function call)

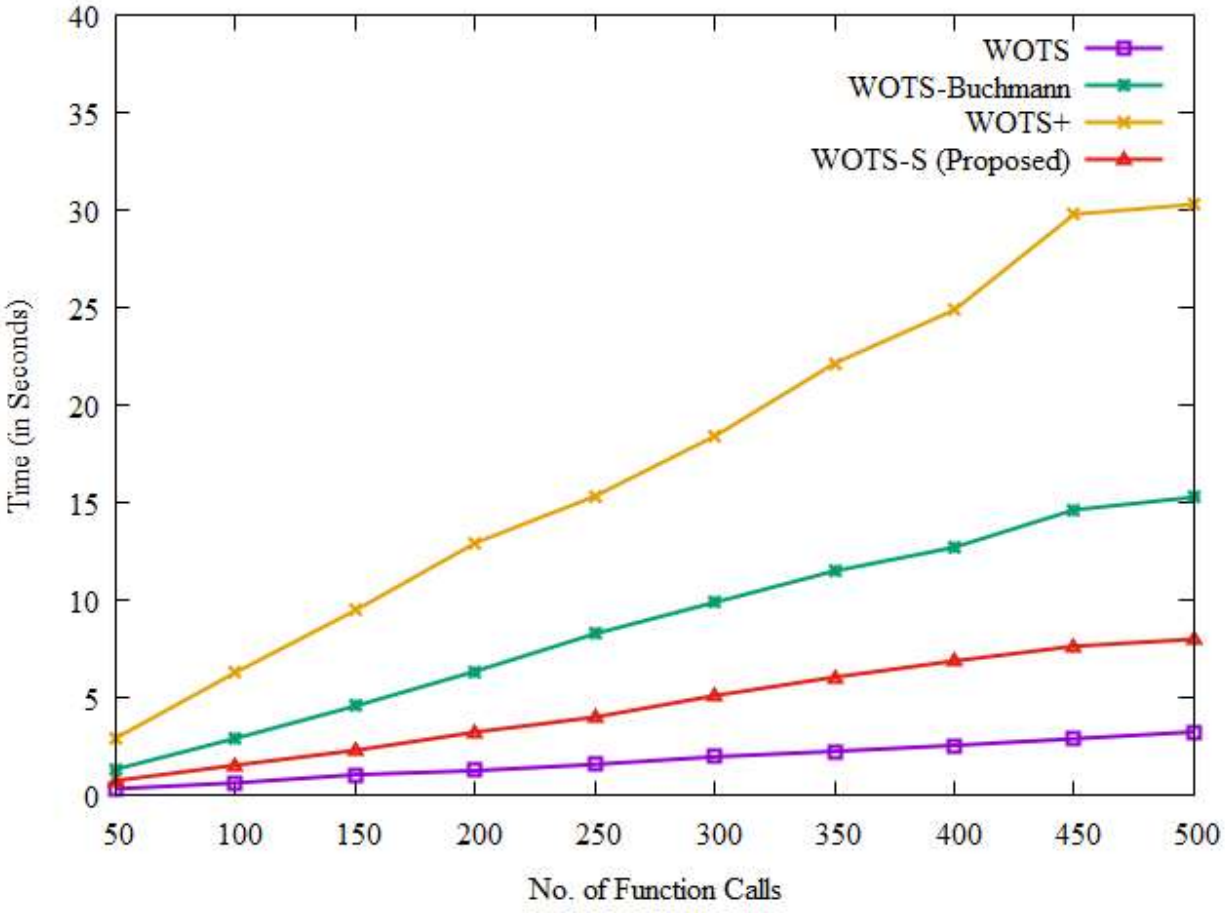

Figure 7: Key generation time (multiple function calls)

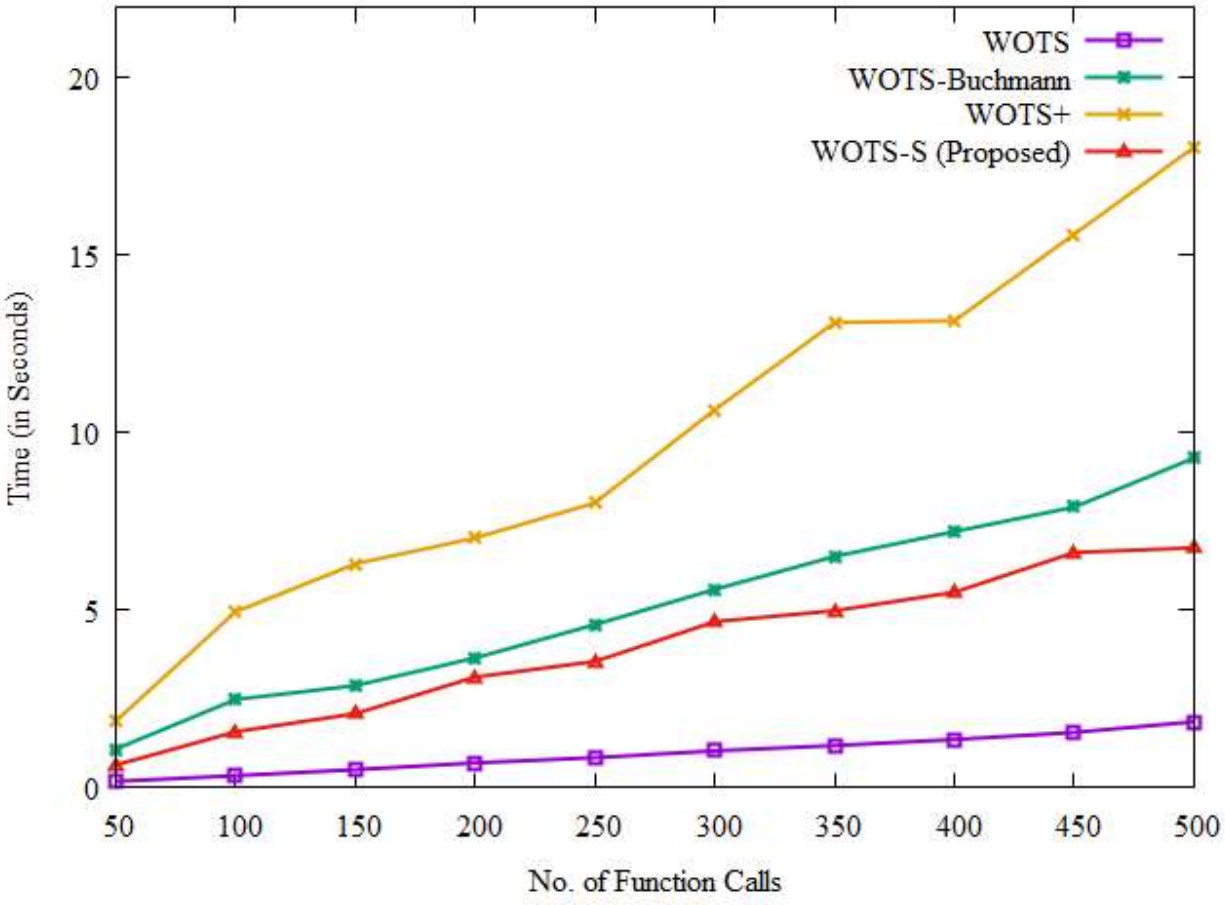

Figure 8: Signature creation time (multiple function calls)

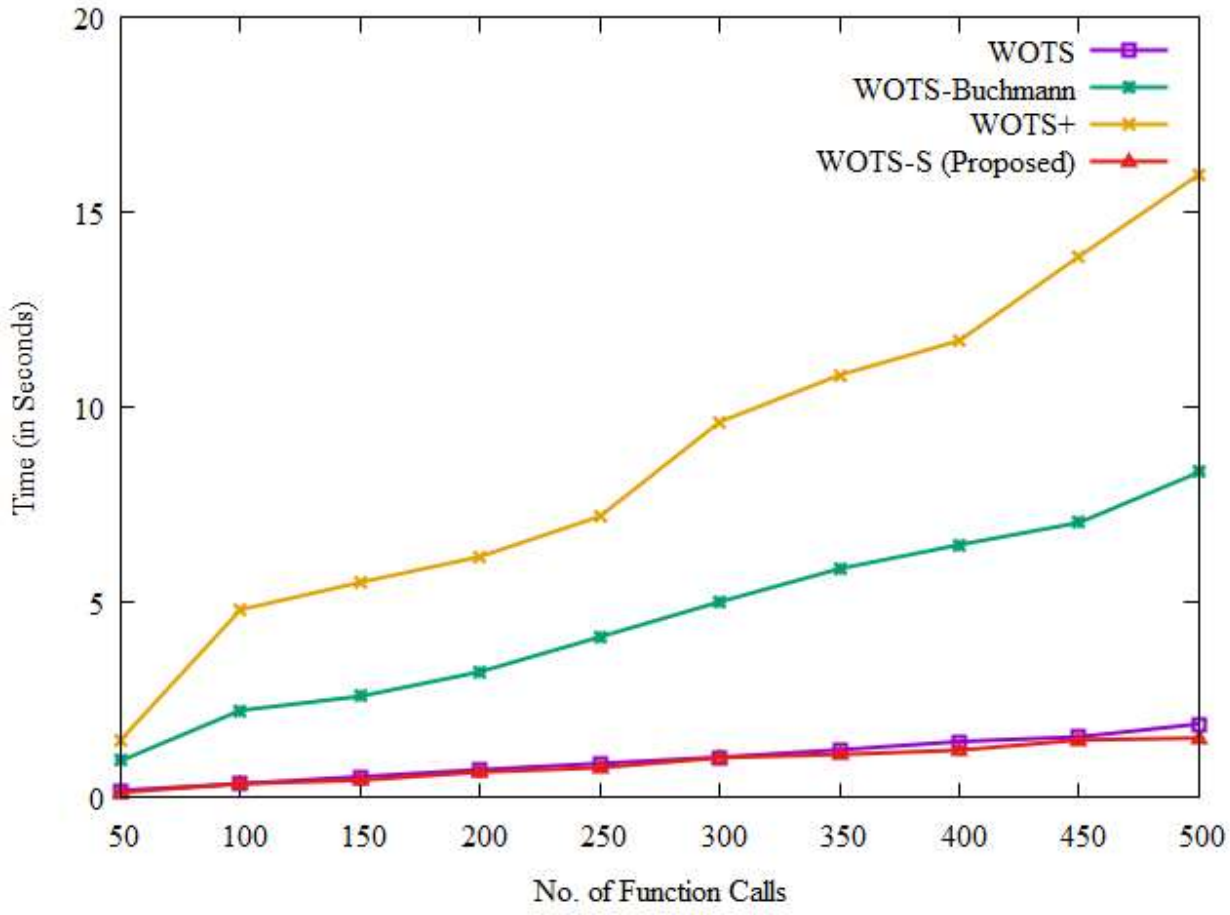

Figure 9: Signature verification time (multiple function calls)

7. Conclusions and Future Work

In this paper, we have proposed WOTS-S, which is a new variant of the popular hash-based OTS scheme WOTS. The proposed signature scheme is customized particularly for post-quantum cryptocurrencies. Since almost all existing cryptocurrencies use ECDSA, which will no longer be secure once quantum computers are available at large scale. Therefore, it is imperative to design quantum resistant ledger, which can resist attacks from a quantum computer. Hence, the very recently proposed popular cryptocurrencies, IoTA and QRL, have switched from ECDSA to WOTS or some of its variants. Although hash based signature schemes are a good alternate of ECDSA, a key limitation is the relatively larger signature sizes. WOTS-S offers reduced signature sizes as compared to WOTS and its previously proposed variant, including WOTS (Buchmann) and WOTS^{+}. Our implementation reveals that WOTS-S is also computationally-efficient than both variants of WOTS. Furthermore, the previously proposed variants of WOTS reduce signature sizes with the help of

randomizations and bitmasks, which are expensive. On the other hand, WOTS-S is based on collision resistance hash functions and avoids bitmasks. In future, we will use our proposed compact signature scheme into the blockchain technology beyond cryptocurrencies. Internet of Energy (IoE) is one of the potential research areas for year 2020 and onwards. We will use WOTS-S to design blockchain-based solutions for IoE scenarios. WOTS-S being an efficient OTS scheme, is suitable for the distributed ledgers deployed into IoE environments.

Acknowledgement:

The authors would like to thank the anonymous reviewers for their comments and suggestions.

REFERENCES:

[1]. Wu, Y., Fan, H., Wang, X. et al. A regulated digital currency. Sci. China Inf. Sci. 62, 32109 (2019). https://doi.org/10.1007/s11432-018-9611-3

[2]. Liu J.K., Tsang P.P., Wong D.S. (2005) Recoverable and Untraceable E-Cash. In: Chadwick D., Zhao G. (eds) Public Key Infrastructure. EuroPKI 2005. Lecture Notes in Computer Science, vol 3545. Springer, Berlin, Heidelberg (DOI: 10.1007/11533733_14)

[3]. Camenisch J., Hohenberger S., Lysyanskaya A. (2005) Compact E-Cash. In: Cramer R. (eds) Advances in Cryptology – EUROCRYPT 2005. EUROCRYPT 2005. Lecture Notes in Computer Science, vol 3494. Springer, Berlin, Heidelberg (DOI: 10.1007/11426639_18)

[4]. Nakamoto, S., Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf, 2009

[5]. Aggarwal, D.; Brennen, G.; Lee, T.; Santha, M. & Tomamichel, M., Quantum Attacks on Bitcoin, and How to Protect Against Them, Ledger, 2017 (DOI: 10.5195/LEDGER.2018.127)

[6]. Shor, P.W. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Comput. 26, 1484–1509, 1997 (DOI: 10.1137/S0097539795293172).

[7]. Buchanan, W. & Woodward, A., Will quantum computers be the end of public key encryption? Journal of Cyber Security Technology, 2016, 1 – 22, (DOI: 10.1080/23742917.2016.1226650)

[8]. C. Dods, N. P. Smart, M. Stam, Hash based digital signature schemes, in: N. P. Smart (Ed.), Cryptography and Coding, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 96-115. doi:10.1007/11586821_8.

[9]. R. El Bansarkhani, M. Geihs, J. Buchmann, Pqchain: Strategic design decisions for distributed ledger technologies against future threats, IEEE Security & Privacy 16 (2018) 57-65. doi:10.1109/MSP.2018.3111246

[10]. Gao, Y.; Chen, X.; Chen, Y.; Sun, Y.; Niu, X. & Yang, Y., A Secure Cryptocurrency Scheme Based on Post-Quantum Blockchain, IEEE Access, 2018 (DOI: 10.1109/ACCESS.2018.2827203)

[11]. X. Zhang, Y. Tang, H. Wang, C. Xu, Y. Miao, H. Cheng, Lattice based proxy-oriented identity-based encryption with keyword search for cloud storage, Information Sciences 494 (2019) 193-207. doi:https://doi.org/10.1016/j.ins.2019.04.051. URL:http://www.sciencedirect.com/science/article/pii/S0020025519303706

[12]. X. Zhang, C. Xu, H. Wang, Y. Zhang, S. Wang, Fs-peks: Lattice-based forward secure public-key encryption with keyword search for cloud assisted industrial internet of things, IEEE Transactions on Dependable and Secure Computing (2019) 1-1 doi:10.1109/TDSC.2019.2914117.

[13]. X. Zhang, H. Wang, C. Xu, Identity-based key-exposure resilient cloud storage public auditing scheme from lattices, Information Sciences 472 (2019) 223-234. doi:https://doi.org/10.1016/j.ins.2018.09.013. URL:http://www.sciencedirect.com/science/article/pii/S0020025518307138

[14]. J.-P. Aumasson, G. Endignoux, Improving stateless hash-based signatures, in: N. P. Smart (Ed.), Topics in Cryptology - CT-RSA 2018, Springer International Publishing, Cham, 2018, pp. 219-242. doi:10.1007/978-3-319-76953-0_12.

[15]. S. Popov, Academic papers IoTA, The Tangle, Version 1.4.3, 2018, Online Available: https://www.iota.org/research/academic-papers

[16]. theQRL, The quantum resistant ledger, in: QRL white paper, 2019. URL https://github.com/theQRL/Whitepaper/blob/master/QRL_whitepaper.pdf

[17]. Merkle, R. C., "A certified digital signature," in Proceedings on Advances in Cryptology, ser. CRYPTO '89. Springer-Verlag New York, Inc., 1989 (DOI: 10.1007/0-387-34805-0_21)

[18]. Hülsing, A. Youssef, A.; Nitaj, A. & Hassanien, A. E. (Eds.) W-OTS+ -- Shorter Signatures for Hash-Based Signature Schemes Progress in Cryptology - Africacrypt 2013, Springer Berlin Heidelberg, 2013, 173-188 (DOI: 10.1007/978-3-642-38553-7_10)

[19]. D. J. Bernstein, "Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete?", SHARCS 2009, 2009.

[20]. Chaum D. (1983) Blind Signatures for Untraceable Payments. In: Chaum D., Rivest R.L., Sherman A.T. (eds) Advances in Cryptology. Springer, Boston, MA (DOI: 10.1007/978-1-4757-0602-4_18)

[21]. Okamoto T., Ohta K. (1992) Universal Electronic Cash. In: Feigenbaum J. (eds) Advances in Cryptology — CRYPTO '91. CRYPTO 1991. Lecture Notes in Computer Science, vol 576. Springer, Berlin, Heidelberg (DOI: 10.1007/3-540-46766-1_27)

[22]. Jakobsson, M. & Yung, M., Revokable and Versatile Electronic Money (Extended Abstract), Proceedings of the 3rd ACM Conference on Computer and Communications Security, ACM, 1996, 76-87 (DOI: 10.1145/238168.238191)

[23]. Sander T., Ta-Shma A. (1999) Auditable, Anonymous Electronic Cash. In: Wiener M. (eds) Advances in Cryptology — CRYPTO' 99. CRYPTO 1999. Lecture Notes in Computer Science, vol 1666. Springer, Berlin, Heidelberg (DOI: 10.1007/3-540-48405-1_35)

[24]. Litecoin – open source P2P didgital currency, https://litecoin.org/, accessed 03 October, 2017

[25]. V. Buterin, A next generation smart contract & decentralized application platform, ethereum White Paper (2014). URL https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.p

[26]. Ben-Sasson, E. Chiesa, A. Garman, C. Green, M. Miers, I. Tromer, E. and Virza, M., Zerocash: Decentralized anonymous payments from Bitcoin. In IEEE Symposium on Security and Privacy, 2014 (DOI: 10.1109/SP.2014.36)

[27]. Ripple – One frictionless experience to send money globally, https://ripple.com/, accessed 03 October, 2017

[28]. Kiktenko, E.O. Pozhar, N.O. Anufriev, M.N. Trushechkin, A.S. Yunusov, R.R. Kurochkin, Y.V. Lvovsky, A.I. and Fedorov, A.K., Quantum-secured blockchain, Quantum Science and Technology, 2017 (DOI: 10.1088/2058-9565/aabc6b)

[29]. Ikeda K. (2019) qBitcoin: A Peer-to-Peer Quantum Cash System. In: Arai K., Kapoor S., Bhatia R. (eds) Intelligent Computing. SAI 2018. Advances in Intelligent Systems and Computing, vol 858. Springer, Cham (DOI: 10.1007/978-3-030-01174-1_58)

[30]. Jogenfors, J. (2019, May). Quantum Bitcoin: An Anonymous, Distributed, and Secure Currency Secured by the No-Cloning Theorem of Quantum Mechanics. In 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC) (pp. 245-252). IEEE. (DOI: 10.1109/BLOC.2019.8751473)

[31]. E. Dahmen, K. Okeya, T. Takagi, C. Vuillaume, Digital signatures out of second-preimage resistant hash functions, in: J. Buchmann, J. Ding (Eds.), Post-Quantum Cryptography, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 109-123. Doi:10.1007/978-3-540-88403-3_8.

[32]. Lamport, L., "Constructing digital signatures from a one-way function," Tech. Rep., 1979.

[33]. Buchmann, J.; Dahmen, E.; Ereth, S.; Hülsing, A. Rückert, M. (Eds.) On the Security of the Winternitz One-Time Signature Scheme, Progress in Cryptology -- AFRICACRYPT 2011, Springer Berlin Heidelberg, 2011, 363-378 (DOI: 10.1007/978-3-642-21969-6_23)

[34]. Reyzin L., Reyzin N. (2002) Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In: Batten L., Seberry J. (eds) Information Security and Privacy. ACISP 2002. Lecture Notes in Computer Science, vol 2384. Springer, Berlin, Heidelberg (DOI: 10.1007/3-540-45450-0_11)

[35]. Bernstein D.J. et al. (2015) SPHINCS: Practical Stateless Hash-Based Signatures. In: Oswald E., Fischlin M. (eds) Advances in Cryptology -- EUROCRYPT 2015. EUROCRYPT 2015. Lecture Notes in Computer Science, vol 9056. Springer, Berlin, Heidelberg (DOI: 10.1007/978-3-662-46800-5_15)

[36]. Buchmann J., Dahmen E., Hülsing A. (2011) XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In: Yang BY. (eds) Post-Quantum Cryptography. PQCrypto 2011. Lecture Notes in Computer Science, vol 7071. Springer, Berlin, Heidelberg (DOI: 10.1007/978-3-642-25405-5_8)

[37]. A. Hulsing, L. Rausch, J. Buchmann, Optimal parameters for xmssmt, in: A. Cuzzocrea, C. Kittl, D. E. Simos, E. Weippl, L. Xu (Eds.), Security Engineering and Intelligence Informatics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 194-208. doi:10.1007/978-3-642-40588-4_14.

[38]. S. Gueron, N. Mouha, Sphincs-simpira: Fast stateless hash-based signatures with post-quantum security, IACR Cryptology ePrint Archive 2017 (2017) 645.

[39]. K. Chalkias, J. Brown, M. Hearn, T. Lillehagen, I. Nitto, T. Schroeter, Blockchained post-quantum signatures, 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) (2018) 1196–1203 doi: 10.1109/Cybermatics_2018.2018.00213.

[40]. Blaauwendraad, Breus, Zekeriya Erkin, Peter Schwabe, Oguzhan Ersoy, and Bart de Jong. "Post-quantum Hash-based Signatures for Multi-chain Blockchain Technologies." Master Thesis (2019)

[41]. Suciu, George, Mari-Anais Sachian, Marius Dobrea, Cristiana-Ioana Istrate, Ana Lavinia Petrache, Alexandru Vulpe, and Marius Vochin. "Securing the Smart Grid: A Blockchain-based Secure Smart Energy System." In 2019 54th International Universities Power Engineering Conference (UPEC), pp. 1-5. IEEE, 2019. (DOI: 10.1109/UPEC.2019.8893484)

[42]. Zhu, Qingyi, Seng W. Loke, Rolando Trujillo-Rasua, Frank Jiang, and Yong Xiang. "Applications of Distributed Ledger Technologies to the Internet of Things: A Survey." ACM Computing Surveys (CSUR) 52, no. 6 (2019): 120. (DOI: 10.1145/3359982)

[43]. Grover, L. K., A Fast Quantum Mechanical Algorithm for Database Search, Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, ACM, 1996, 212-219 (DOI: 10.1145/237814.237866)

Furqan Shahid is pursuing his Ph.D. degree in Computer Science at COMSATS University Islamabad, Pakistan under supervision of Dr. Abid Khan. Previously, he completed his Masters from the International Islamic University, Islamabad in 2012. His research interests include Blockchain and post-quantum cryptography.

Abid Khan is with the department of computer science, Aberystwyth University, United Kingdom. Previously, he was with the department of computer science, COMSATS University Islamabad (CUI), Pakistan. His research interests include applied cryptography, security/privacy issues in distributed systems (including blockchain, IoT, smart grids and cloud computing) and quantum cryptography. He was awarded the prestigious "Fellowship for Young Researcher" by Politecnico De Torino for his postdoctoral research (2009-2011). Abid did his PhD from Harbin Institute of Technology, P.R.China. Before, that he did his MSC in computer science from Quaid I Azam University, Islamabad, Pakistan.

Saif Ur Rehman Malik did his Ph.D. in 2014 from Department of Electrical and Computer Engineering, North Dakota State University, USA. Currently, he is a Senior Researcher at Cybernetica, AS Estonia.

His areas of expertise include the application of Formal Methods in Large Scale Computing Systems, Distributed Computing, Data Centers, Security and Routing Protocols, and IOT. His research work appears in several reputable journals and transactions. He is also serving as a reviewer and TPC of many well reputed Journals and transactions.

Kim-Kwang Raymond Choo received the Ph.D. degree in information security from the Queensland University of Technology, Australia, in 2006. He currently holds the Cloud Technology Endowed professorship with The University of Texas. In 2016, he was named the Cybersecurity Educator of the Year - APAC (Cybersecurity Excellence Awards are produced in cooperation with the Information Security Community on LinkedIn), and in 2015 he and his team won the Digital Forensics Research Challenge organized by Germany's University of Erlangen-Nuremberg. He is the recipient of the 2019 IEEE Technical Committee on Scalable Computing (TCSC) Award for Excellence in Scalable Computing (Middle Career Researcher), 2018 UTSA College of Business Col. Jean Piccione and Lt. Col. Philip Piccione Endowed Research Award for Tenured Faculty, Outstanding Associate Editor of 2018 for IEEE Access, British Computer Society's 2019 Wilkes Award Runner-up, 2019 EURASIP Journal on Wireless Communications and Networking (JWCN) Best Paper Award, Korea Information Processing Society's Journal of Information Processing Systems (JIPS) Survey Paper Award (Gold) 2019, IEEE Blockchain 2019 Outstanding Paper Award, IEEE TrustCom 2018 Best Paper Award, ESORICS 2015 Best Research Paper Award, 2014 Highly Commended Award by the Australia New Zealand Policing Advisory Agency, Fulbright Scholarship in 2009, 2008 Australia Day Achievement Medallion, and British Computer Society's Wilkes Award in 2008. He is also a Fellow of the Australian Computer Society, an IEEE Senior Member, and Co-Chair of IEEE Multimedia Communications Technical Committee's Digital Rights Management for Multimedia Interest Group.