Li *et al. Cybersecurity* (2022) 5:13
https://doi.org/10.1186/s42400-022-00117-w

Cybersecurity

REVIEW Open Access

Hash-based signature revisited

Lingyun Li[1,2,3*], Xianhui Lu[1,2] and Kunpeng Wang[1,2]

Abstract
The current development toward quantum attack has shocked our confidence on classical digital signature schemes. As one of the mainstreams of post quantum cryptography primitives, hash-based signature has attracted more and more concern in both cryptographic research and application in recent years. The goal of this paper is to present, classify and discuss different solutions for hash-based signature. Firstly, this paper discusses the research progress in the component of hash-based signature, i.e., one-time signature and few-time signature; then classifies the tree-based public key authentication schemes of hash-based signature into limited number and stateful schemes, unlimited number and stateful schemes and unlimited number and stateless schemes. The above discussion aims to analyze the overall design idea of different categories of hash-based signatures, as well as the construction, security reduction and performance efficiency of specific schemes. Finally, the perspectives and possible development directions of hash-based signature are briefly discussed.

Keywords: Hash-based signature, One-time signature, Few-time signature, Hash tree

Introduction

Digital signature based on public-key cryptosystems has been widely used in the modern online information transmission, for example, electronic elections, digital cash, etc., to provide entity authentication and non-repudiation, message integrity and confidentiality. The original research of public-key digital signature is focused on the schemes upon the computationally hard number theoretic problems, such as RSA designed on the difficulty assumption of factoring large integers (Rivest et al. 1978), ElGamal designed on the difficulty assumption of discrete logarithms, etc. (ElGamal 1985) But the development of quantum computer has shocked our confidence on this kind of classical digital signature schemes (Gisin et al. 2002; Bennett et al. 1992; Bernstein 2009; Brassard et al. 2000; Ekert 1991; Bennett 1992; Gröblacher et al. 2006), by applying "Shor's algorithm" (Shor 1999), theoretic problems will no longer be hard and classical cryptosystems can be broken easily. We can only achieve computational security by improving the related parameters in scale, it is dynamically affected by the development of number theory and computing performance, as a result, it lacks sustainedly durable security.

Another kind of public-key based signature scheme, hash-based signature, which is believed to resist to both classical and quantum computers, is attracting more and more attention in cryptography research. Hash-based signature is commonly designed in combining one-time signature (OTS) (Merkle 1979b; Naor et al. 2005) or few-time signature (FTS) (Perrig 2001; Reyzin and Reyzin 2002; Aumasson and Endignoux 2017, 2018; Bernstein et al. 2017, 2019) with hash tree, and its security is only based on the security assumptions of the underlying hash function, such as collision resistance, second preimage resistance, onewayness (Coron et al. 2005; Rogaway and Shrimpton 2004), etc., avoiding the dependence on the hard number theoretic problems of other digital signature schemes. Hash function is one of the foundational topics in cryptography, and many different rapid design approaches and relevant researches have been presented to achieve different security properties. Opposite to the digital signature designed on hard number theoretic problem, attacks against a specific hash function will not affect the overall security of hash-based signature; it

*Correspondence: lilingyun@iie.ac.cn
[1] State Key Laboratory of Information Security, Institute of Information Engineering, CAS, No. 89 Minzhuang Road, Haidian District, Beijing 100093, China
Full list of author information is available at the end of the article

could be solved by replacing the attacked hash function by others remain secure easily.

Due to the concision and efficiency in the design, hash function has been implemented maturely after decades of improvement, hash-based signature which treats hash function as the central subroutine can achieve high efficiency as well. In addition, through choosing distinct underlying hash functions and their parameters, as well as one-time/FTS schemes, tree traversal algorithms, etc., it is sufficiently flexible to make a trade-off between signature size, time and storage, etc., to meet distinct needs in applications.

However, hash-based signature still has some drawbacks which need to be solved to make it more practical. First, the signature key can be used only once to sign one message in the underlying OTS, even for FTS in which a single key pair could be used to sign a few messages, the security of the signature decreases as the number of signatures increases, it raises a crucial issue that how to manage large-scale public keys consequently. The tree-based public key authentication is a classical way to solve the public key management issue. In the original path authentication schemes the signature size grows logarithmically with the number of signatures, and more path authentication schemes have been presented in the recent research to achieve more efficiency. Second, as the key pair has to be changed into a fresh one after each signature generation, a natural way is to utilize the state management to keep record of the state and synchronization between signer and verifier, otherwise, there may exists reduplicate use of the signature key, such that the adversary can forge a valid signature easily. Therefore, state management plays an essential role in many scenarios of the application of hash-based signature, but it also limits it less practical. In recent years, some stateless hash-based signatures have been presented which attract researcher's concerns, but the stateless hash-based signature has significantly higher signatures size compared with the stateful ones in the analogous security level, as a result, it is difficult to be widely applied in practice.

The goal of this survey is to present, classify and discuss different schemes that provide solutions for OTS/FTS, many-time and full-time hash-based signatures, focus on different security analysis and implementation strategies about stateful and stateless hash-based signatures. The rest of the survey is organized as follows. In "Related work" section, we briefly introduce the related work and progress in the research of hash-based signature; in "Security notions" section, we give the security notions of hash function, as well as the definition and the corresponding security notions of signature scheme; in "Hash-based Signature" section, we discuss the progress in the component of hash-based signature in detail, i.e., one-time signature and few-time signature, and classify the tree-based public key authentication schemes of hash-based signature into limited number and stateful schemes, unlimited number and stateful schemes and unlimited number and stateless schemes, introduce their constructions, analyze and compare the security assumptions of underlying function required in these specific hash-based signature schemes; in "Conclusion" section, the perspectives and possible development directions of hash-based signature are briefly discussed.

Related work

Research into hash-based signature has a long history. In 1979, Merkle presented the first hash-based signature scheme constructed out of hash function only (Merkle 1979b; Rogaway and Shrimpton 2004), therefore, it is also named as Merkle signature scheme, which is a combination of an OTS to sign a single message per key pair and a path authentication scheme to provide an authentication path to verify the public key related to the signature of this single message. During the past 40 years, much more schemes with security analysis have become available on hash-based signature. For OTS, the following schemes have been wildly studied: the Lamport OTS (Merkle 1989), the Merkle OTS (Merkle 1989), the Winternitz OTS (Merkle 1989; Buchmann et al. 2011a), the Bleichenbacher-Maurer (Bleichenbacher and Maurer 1994) OTS and W-OTS+ (Hülsing 2013), etc. Merkle OTS is designed based on the Lamport OTS, Winternitz OTS can be regarded and as a generalization of Merkle OTS, Merkle OTS and Winternitz OTS, it iterates an underlying function repeatedly while iteration times relate to the message to be signed. Bleichenbacher and Maurer gave a generalization of OTS schemes on acyclic graphs. W-OTS+ provides shorter signatures than previous schemes with the analogous security level, and offers a tight reduction that W-OTS+ is existentially unforgeable under an adaptive chosen message attack (EUCMA) in the standard model, if the underlying hash function family is second preimage resistant and undetectable one-way.

FTS, as the name suggests, can be used to sign more than one message with the same key pair. Typical schemes for FTS schemes are Biba FTS (Perrig 2001), Hors FTS (Reyzin and Reyzin 2002) with its variants Pors FTS (Aumasson and Endignoux 2017, 2018) and Fors FTS (Bernstein et al. 2017, 2019), etc. Biba FTS can achieve high verification efficiency at the cost of heavy pre-computation on key generation and signature; meanwhile, it requires time synchronization between the sender and receiver. Hors FTS is mainly focus on selecting some elements which is determined by the message to be signed from a far larger set. Pors and Fors FTS improves Hors FTS in a way avoiding weak messages, which are mapped to a small subset and vulnerable to subset resilience attack.

Although all FTS schemes suggest using the single key pair to sign more than one message in contrast to one message in OTS scheme, the security of the signature decreases as the number of signatures increases.

Since key size increases linearly with the number of the signed messages in hash-based signature, it is essential to use key management to realize using fewer key pairs to authenticate more messages, thus path authentication scheme comes into being. Among them, tree-based authentication using the hash tree to authenticate public keys efficiently is wildly adopted to solve the problem of key management. To sign a message, after the OTS signature is generated, an authentication path generated by the tree-based authentication scheme has to be applied to verifier. Verifier authenticates the one-time public key along with the authentication path to construct the tree from leaf nodes to root, or from the root to bottom, which comes into two main different approaches to authenticate public keys in large scale, depending on stateful or stateless of the hash-based signature scheme, and signing limited number or cryptographic unlimited number of messages. In terms of stateful hash-based signature, its signature key needs to be renewed when exceeding its service time, namely, the signing times it can be used to sign the messages; whereas the stateless hash-based signature selects signature key pseudorandomly, and doesn't require key management. Typical schemes of limited number and stateful hash-based signature are Merkle hash-based signature (Merkle 1989), eXtended Merkle Signature Scheme (XMSS) (Buchmann et al. 2011b; Hülsing et al. 2018), Leighton-Micali signature (LMS) (Leighton and Micali 1995; Katz 2016; Failed 2017; McGrew et al. 2019; Buchmann et al. 2006), etc.; typical Schemes of unlimited number and stateful hash-based signature are generalized Merkle signature scheme (GMSS) (Buchmann et al. 2006, 2007), XMSS with multi-tree (XMSS^{MT}) (Hülsing et al. 2018, 2013), Hierarchical Signature System (HSS) (McGrew et al. 2019), etc.; typical Schemes of unlimited number and stateless hash-based signature are SPHINCS(Bernstein et al. 2015), SPHINCS+(Bernstein et al. 2017, 2019), Gravity-SPHINCS (Aumasson and Endignoux 2017, 2018), etc.

There has been an increasing amount of literatures on traversal algorithm of hash-based signature, such as tree traversal algorithm and hash chain traversal algorithm, which allows for optimal trade-off between signature time and storage cost. In terms of tree traversal algorithm, two different approaches are mainly discussed to compute authentication paths, depending on considering the node or the subtree of a Merkle tree as the basic computational element. In terms of fractal hash chains traversal algorithm (Jakobsson 2002; Jakobsson et al. 2003; Naor et al. 2006; Coppersmith and Jakobsson 2002; Sella 2003; Berman et al. 2007; Knecht et al. 1409; Buchmann et al. 2008), all elements in a hash chain are determined by the initial input and the combined output is a single element. As the main purpose of this paper is to introduce typical hash-based signature schemes which have been discussed and adopted widely, readers can refer to the literature in detail to learn more about traversal algorithm of hash-based signature.

In the application, Even and Goldreich proposed an on-line/off-line signature scheme (Even et al. 1996), which uses the efficient OTS scheme for the on-line signing, along with an ordinary signature scheme used for the off-line phase. Other applications such as TESLA (Perrig et al. 2002), wireless security, etc., are also well researched (Perrig et al. 2001; Bergadano et al. 2002; Buldas et al. 2017, 2018).

Security notions

A hash function family H is a map $H := \left\{h_k : \ (0,\ 1)^* \to (0,\ 1)^n | k \in (0,\ 1)^n \right\}$, such that n is polynomial in security parameter λ, here, $\{0,\ 1\}^*$ denotes the binary string of arbitrary length.

In this section, firstly we give four security assumptions, one-way, second preimage resistant, collision resistant, enhanced target collision resistant hash function related to hash function, as the basic component of hash-based signature, the security of hash function affects the performance of corresponding hash-based signature seriously. Secondly, we give the definition and the corresponding security notion of signature scheme.

Definition 1 *One-way hash function* (Merkle 1979a).

Hash function family H, security parameter λ are defined as above. We give the definition that function family H is *one way* if for any probabilistic polynomial-time adversary A there is a negligible function *negl* such that

$$\begin{aligned} Adv_H^{OW} &= \Pr\left\{ \begin{array}{l} k \overset{\$}{\leftarrow} \{0,1\}^n, y \overset{\$}{\leftarrow} \{0,1\}^n, \ \ m \leftarrow A(1^\lambda) : \\ H_k(m) = y \end{array} \right\} \\ &= negl(\lambda) \end{aligned}$$

Here $x \overset{\$}{\leftarrow} X$ denotes that x is chosen from X uniformly at random.

Definition 2 *Second preimage resistant hash function* (Menezes et al. 2018).

Hash function H, security parameter λ are defined as above. We give the definition of the function family H is *second preimage resistant* if for any probabilistic polynomial-time adversary A there is a negligible function *negl* such that

$$Adv_H^{SPR} = \Pr\left\{\begin{array}{l} k \xleftarrow{\$} \{0,1\}^n,\ m \xleftarrow{\$} \{0,1\}^*,\ m\prime \leftarrow A\left(1^\lambda, m\right) : \\ H_k(m) = H_k(m\prime) \end{array}\right\} = negl(\lambda)$$

Definition 3 *Collision-resistant hash function* (Damgård 1989; Preneel et al. 1993; Black et al. 2002; Goldwasser et al. 1988).

Hash function H, security parameter λ are defined as above. We give the definition of the function family H is *collision resistant* if for any probabilistic polynomial-time adversary A there is a negligible function *negl* such that

$$Adv_H^{CR} = \Pr\left\{\begin{array}{l} k \xleftarrow{\$} \{0,1\}^n, (m, m\prime) \leftarrow A\left(1^\lambda\right) : \\ H_k(m) = H_k(m\prime)] \end{array}\right\} = negl(\lambda)$$

Definition 4 *Enhanced target collision resistant hash function* (*eTCR*) (Halevi and Krawczyk 2006*)*.

Hash function H, security parameter λ are defined as above. We give the definition of the function family H is *eTCR* if for any probabilistic polynomial-time adversary A there is a negligible function *negl* such that

$$Adv_H^{eTCR} = \Pr\left\{\begin{array}{l} (m, m\prime) \leftarrow A\left(1^\lambda\right);\ k \xleftarrow{\$} \{0,1\}^n;\ (k\prime, m\prime) \leftarrow A\left(1^\lambda\right) : \\ H_k(m) = H_{k'}(m\prime)] \end{array}\right\} = negl(\lambda)$$

In *eTCR* game, first A commits a message m, then receives a random key K, A wins the game if he can output (m′, k′) such that

$$(m\prime, k\prime) \neq (m, k) \text{ and } H_k(m) = H_{k\prime}(m\prime).$$

Definition 5 *Signature scheme* (Rivest et al. 1978; ElGamal 1985).

A digital signature scheme ∏ is defined as a triple of probabilistic polynomial-time algorithms ∏ = (*Gen, Sign, Ver*):

Gen On input security parameter 1^λ, output a private signature key *sk* and public verification key *pk*;

Sig On input a signature key *sk*, message *m*, output a signature σ;

Ver Ver is a deterministic algorithm, on input a public key, a message and its signature triple *(pk, m, σ)*, output 1 iff σ is a valid signature on *m*.

Definition 6 *EUCMA security* (Halevi and Krawczyk 2006).

Let $\prod = (Gen, Sig, Ver)$ be a signature scheme with private key *sk* and public key *pk*. We define *EUCMA* as follows. The forger A has access to the public key and a signing oracle O (*sk*, ·). On input the query of a message *m*, O returns A the signature *Sig*(*m*) on *m*. A has the chance to adaptively query O at most q times. The parameter q is bounded up with different kinds of signature schemes. For one time signature, $q = 1$. The property adaptive means a message may depend on answers of O replied previously. On input security parameter λ, ∏ is unforgeability under an adaptive chosen message attack if for any probabilistic polynomial-time A there is a negligible function *negl* such that

$$Adv_\Pi^{EU-CMA} = \Pr\left\{\begin{array}{l} (sk_0, pk) \leftarrow Gen\left(1^\lambda\right) \\ \{(M_i, \sigma_i)\}|_{i=1}^q \textit{be the q times answer} \\ \textit{of } \ O\left(sk_j, \cdot\right); \\ (M*, \sigma*) \leftarrow A\left(pk, (M_i, \sigma_i)\}|_{i=1}^q\right) : \\ M* \neq M_i|_{i=1}^q,\ Ver(M*, \sigma*, pk) = 1 \end{array}\right\} = negl(\lambda)$$

Definition 7 *Forward-secure property* (Krawczyk 2000; Bellare and Miner 1999).

Forward-secure property makes sense for key-evolving signature scheme, in which the whole signature process is divided into T periods, the public key pk i.e., the overall public key, is fixed for all these T periods, but the private key sk_i is evolved in each period, where $i = 1, ..., T$. Compared with the conventional signature scheme, forward-secure signature scheme only generates pk and sk_0 in key generation algorithm, and an additional secret key update algorithm Sku is needed, which generates sk_i on input sk_{i-1}, for $i = 1, ..., T$; Sig in each period i takes as input the signature key sk_i and message m, output a signature σ along with the period index i.

Let $\Pi = (Gen, Sku, Sig, Ver)$ be a key-evolving signature scheme with private key sk_0 of the first period and public key pk. We define *forward-secure EUCMA* as follows.

The forger A has access to the total number of time periods, the current time period, the public key pk and a signing oracle $O\ (sk, \cdot)$. In the chosen-message attack phase, on input the query of a message m, O returns A the signature $Sig(m)$ on m under the private key sk_1, ..., sk_T in order. A has the chance to query O at most q times. The parameter q is bounded up with different kinds of signature schemes. For one time signature, $q = 1$.

A stops the chosen-message attack at a period of his choice, and then goes into the break-in phase, where A obtains sk_j of the current period j, his goal is to forge a signature $(i, \sigma*)$ on m^* of his choice such that $(i,M*) \neq (i,M_i|_{i=1}^{q})$ and $Ver(m*, i, \sigma*, pk) = 1$, where $i < j$. In the following, we use break-in to denote both the break-in phase and the index of it for simplicity. On input security parameter λ, Π is *forward-secure EUCMA* if for any probabilistic polynomial-time A there is a negligible function *negl* such that

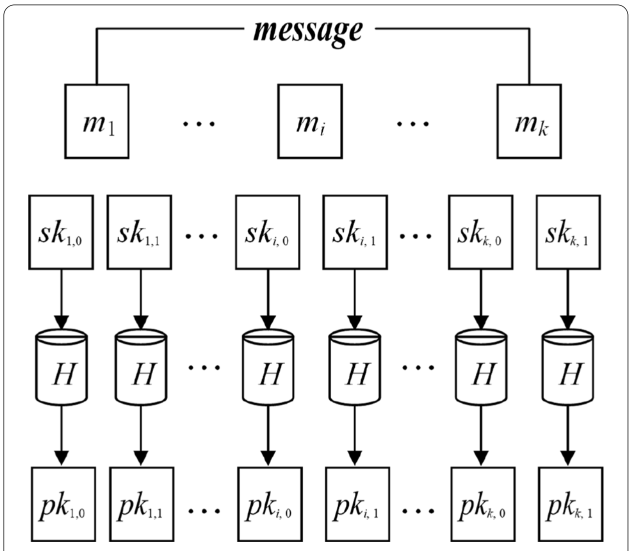

Fig. 1 Lamport OTS scheme

Hash-based signature

OTS

In this section, we introduce several OTS schemes, including Lamport OTS and its improved version Merkle OTS, Winternitz OTS and W-OTS + OTS. Compared with other kinds of OTS scheme, such as graph-based OTS, the above schemes following Merkle and Winternitz's approach are still more widely accepted and employed in the application.

Idea

One signature key of OTS can only be used to sign one message.

$$Adv_{\Pi}^{FW-EUCMA} = \Pr\left\{\begin{array}{l} (sk,pk) \leftarrow Gen\left(1^{\lambda}\right) \\ repeat \\ j = j+1;\ sk_j{=}Sku\left(sk_{j-1}\right) \\ until\ break-in\ phase\ or\ j=T. \\ if\ not\ break-in\ phase\ and\ j=T,\ then\ j=T+1 \\ \{(M_i,\sigma_i,p)\}|_{i=1}^{q}\ be\ the\ q\ times\ answer \\ of\ O\ \left(sk_p,\cdot\right); \\ //\ the\ values\ of\ p\ are\ in\ ascending\ order \\ (i,\ M*,\sigma*) \leftarrow A\left(pk,(M_i,\sigma_i,p)\}|_{i=1}^{q}, sk_{break-in}\right): \\ (i,M*) \neq \left(i,M_i|_{i=1}^{q}\right),\ Ver(i,M*,\sigma*,pk)=1 \\ 0 \leq i < break-in \end{array}\right\} = negl(\lambda)$$

Typical Schemes

(1) *Lamport OTS*

The first OTS scheme was presented by Lamport in 1979(Merkle 1989; Lamport 1979), called Lamport OTS or Lamport-Diffie OTS.

A. *Idea*

Lamport OTS adopts the construction with maximum storage cost and the minimum calculation cost. In order to sign a k-bit message, it utilizes $2k$ private keys and $2k$ public keys. As shown in Fig. 1, each message bit corresponds to two private keys. When the message bit is 0, it corresponds to the first private key; otherwise, it corresponds to the second one. Only one hash evaluation of a one-way hash function is used to construct the public key on input the relevant private key.

B. *Scheme*

For Lamport OTS scheme Sig = (Gen, Sig, Ver), a one-way hash function H is used which is a map: $\{0, 1\}^n \to \{0, 1\}^n$, where n is polynomial in security parameter λ. The message m is presented as $m = (m_0, m_1, \ldots, m_k)$.

The signature scheme is described as follows.

Gen On input security parameter 1^λ, choose private key sequence $sk = (sk_{1,0}, sk_{1,1}, sk_{2,0}, sk_{2,1}, \ldots, sk_k, sk_{k,0,1}) \xleftarrow{\$} (0,1)^{n*2k}$, n is polynomial in security parameter λ, then output the public key sequence pk as follows:

$$\begin{aligned} pk &= \left(pk_{1,0}, pk_{1,1}, pk_{2,0}, pk_{2,1}, \ldots, pk_{k,0}, pk_{k,1}\right) \\ &= \left(H\left(sk_{1,0}\right), H\left(sk_{1,1}\right), H\left(sk_{2,0}\right), H\left(sk_{2,1}\right), \right. \\ &\quad \left. \ldots, H\left(sk_{k,0}\right), H\left(sk_{k,1}\right)\right) \end{aligned}$$

Sig On input the message $m = (m_0, m_1, \ldots, m_k)$ and private key sequence sk, output the signature as follows:

$$\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_k) = \left(sk_{1,m_1}, sk_{2,m_2}, \ldots, sk_{k,m_k}\right)$$

Ver On input the message m, public key sequence pk, signature σ, the following compution and comparison are done in order to verify the signature.

$$(H(\sigma_1), H(\sigma_2), \ldots, H(\sigma_k)) \overset{?}{=} \left(pk_{1,m_1}, pk_{2,m_2}, \ldots, pk_{k,m_k}\right)$$

To sign a message of arbitrary length, compression hash function should be used to compute message digest, which is applied to the signature algorithm as input.

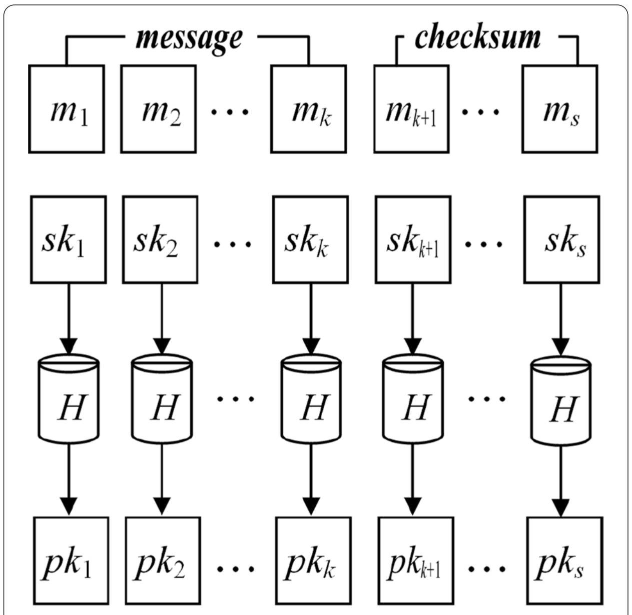

Fig. 2 Merkle OTS scheme

C. *Security*

Unforgeability of Lamport OTS depends on the one-wayness of hash function H.

(2) *Improved Lamport OTS: Merkle OTS*

A. *Idea*

Merkle improves Lamport OTS by appending an extra checksum value to each message before signing the message, which records the quantity of 0 bit in the message, as shown in Fig. 2. The length needs to be signed becomes $\lfloor \log k \rfloor + 1$ bits longer than that of Lamport OTS, set $s = \lfloor \log k \rfloor + 1$.

B. *Scheme*

The public key sequence pk is generated as $pk = (pk_1, pk_2, \ldots, pk_{k+s})$
$= (H(sk_1), H(sk_2), \ldots, H(sk_{k+s}))$.

To sign a message m, $m||checksum = (m_1, \ldots, m_{k+s})$, the signer only needs to reveal the sk_i which is related to $m_i = 1$, $i = 1, \ldots, k+s$. The signature σ turns to be:

$$\sigma = \left(\sigma_{j_1}, \sigma_2, \ldots, \sigma_{j_p}\right) = \left(sk_{j_1}, sk_2, \ldots, sk_{j_p}\right),$$

where $m_{j_p} = 1, \;\; 0 < j_p \leq k + s$.

Consequently, the verification becomes into check:

$$\left(H\left(\sigma_{j_1}\right), H\left(\sigma_{j_2}\right), \ldots, H\left(\sigma_{j_p}\right)\right) \overset{?}{=} \left(pk_{j_1}, pk_2, \ldots, pk_{j_p}\right)$$

Adversary who what to alter the bit value of the message to be signed has to reveal at least one preimage concerning either the original message or its checksum.

C. *Security and Efficiency*

Lamport OTS scheme and Merkle OTS is EUCMA as long as the used hash function is one-way.

Although the key generation, signature and verification of Lampot OTS becomes more efficient, the scale of the signature and corresponding key pair is still quite large; compared with Lamport OTS, Merkle OTS only achieves limited improvement in performance.

(3) Winternitz OTS

In this section we describe Winternitz OTS mentioned in Merkle (1989); Buchmann et al. 2011a), which is an improvement of Lamport OTS and Merkle's scheme, achieves shorter signature and key pair size. An iterative function is applied in the Winternitz OTS to compute the public keys from the private keys.

A. *Idea*

The main drawback of Lamport OTS signatures is its long signature size as well as key pair size, the signature size of Lamport OTS increases proportionally with the bit length of message to be signed by n times, where n is the bit length of each component of key sequence. Moreover, both private key and private key size as 2n times as bit length of message. Although by appending a checksum to each message, only limited decrease has been made in Merkle OTS of signature and key pair size. The idea of Winternitz OTS is to reduce the signature size and key pair size at the expense of some extra hash evaluations as shown in Fig. 3. Concretely, Winternitz OTS processes message m to the new form in base w representation firstly, then decomposes m into blocks of length log w. for each block, iterates a one-way hash function at most w–1 times, and the output of the hash function is considered as the signature for each block, which will be concatenated sequentially to form the whole signature of m. Winternitz OTS provides a trade-off between the signature time and the signature size using the parameter w. The larger w is, the smaller signature size will be. For example, on input a 256 bits message, the hash functions used in Lamport OTS and Winternitz OTS are both maps of $\{0, 1\}^n$ to $\{0, 1\}^n$, when w is chosen as 16, the signature size of Winternitz OTS is 67n bits, signature generation requires 960 and 483 evaluations of the underlying hash function in the worst case and average case, separately; when w is chosen as 256, the signature size of Winternitz OTS is 34n bits, signature generation requires 8160 and 4082 evaluations of the underlying hash function in the worst case and average case, separately; compared with signature size of 256n bits and signature generation of 512 hash evaluations in Lamport OTS, signature size of 264n bits and 136n bits in worst case and average case, signature generation of 264n hash evaluations in Merkle OTS.

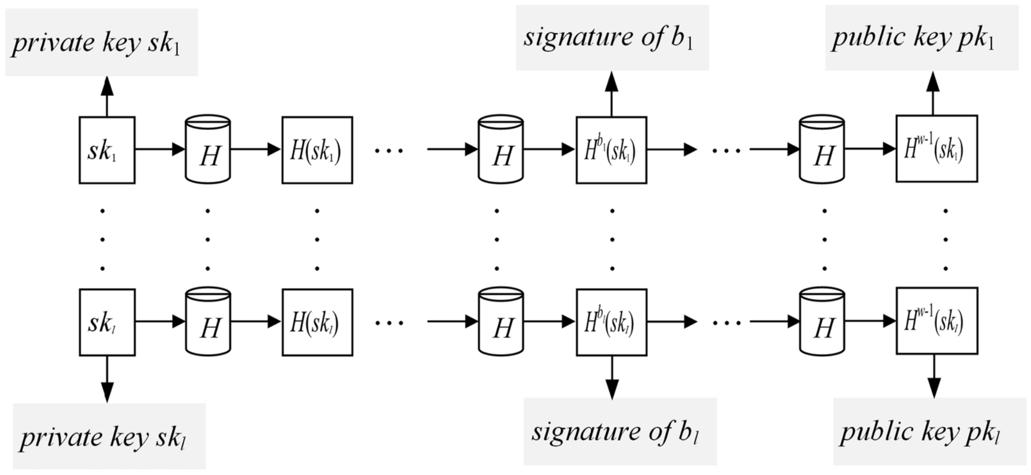

Fig. 3 Winternitz OTS scheme

B. *Scheme*

For Winternitz OTS scheme Sig = (Gen, Sign, Ver), a one-way keyed hash function h is used: $(0,1)^n \times k \to (0,1)^n$ where k is chosen from the key space K uniformly at random. Two parameter w (to be power of two) and l are used, which are related to the bit length of message block and the number of key components in one signature respectively.

The iteration function $h^i(x)$ is constructed by repeating the function $h(x)$ i times, where i ∈ {0,..., $w-1$}, that is, $h^2(x) = h(h(x))$ and $h^0(x) = x$. On input a m bits message, process it to the new form (m_1, m_2, ..., m_p) in base w representation, then attach checksum $C = \sum_{i=1}^{p}(w-1-m_i)$ in base w representation is to m, denote the whole string of $m||C$ as (b_1, b_2, ..., b_l), where $l = \left\lceil \frac{m}{\log w} \right\rceil + \left\lceil \frac{\log\left(\left\lceil \frac{m}{\log w}\right\rceil \cdot (w-1)\right)}{\log w} \right\rceil$.The signature scheme is described as follows.

Gen On input security parameter 1^λ, choose private key sequence $sk = (sk_1, sk_2, \ldots, sk_l) \xleftarrow{\$} (0,1)^{n*l}$, n is polynomial in security parameter λ, then compute public key sequence pk as follows:

$$pk = (pk_1, \ldots, pk_l) = \left(h^{w-1}(sk_1), \ldots, h^{w-1}(sk_l)\right)$$

Sig On input the message m and private key sequence sk, compute the signature sequenceσis:

$$\sigma = (\sigma_1, \ldots, \sigma_l) = \left(h^{b_1}(sk_1), \ldots, h^{b_l}(sk_l)\right)$$

Ver On input the message m, public key sequence pk, signature sequence σ, the following compution and comparison are done verify the signature:

$$\left(h^{w-1-b_1}(\sigma_1), \ldots, h^{w-1-b_l}(\sigma_l)\right) \overset{?}{=} (pk_1, \ldots, pk_l)$$

A collision resistant hash function should be utilized here to sign message of arbitrary length.

C. *Security*

It has been proven that the Winternitz OTS is EUCMA if using either a collision resistant, undectable hash function or a PRF (Dods et al. 2005).

In Dods et al. (2005), the Authors provide security reductions for graph based OTS schemes, and achieves the conclusion that any compressed graph based OTS scheme with strongly compatible signature sets, including Winternitz OTS, is secure if the underlying functions are collision-resistant, one-way and undetectable.

In Buchmann et al. (2011a), it is proven that a variant of W-OTS is EUCMA when instantiated with PRF. Furthermore, it has been proven that a variant of one-way function named key onewayness function exists if PRF exists. Actually, this variant of W-OTS is proven to be EUCMA directly if the underlying function is key-one-way in their reduction. The following conclusions about original Winternitz OTS can be obtained spontaneously that the

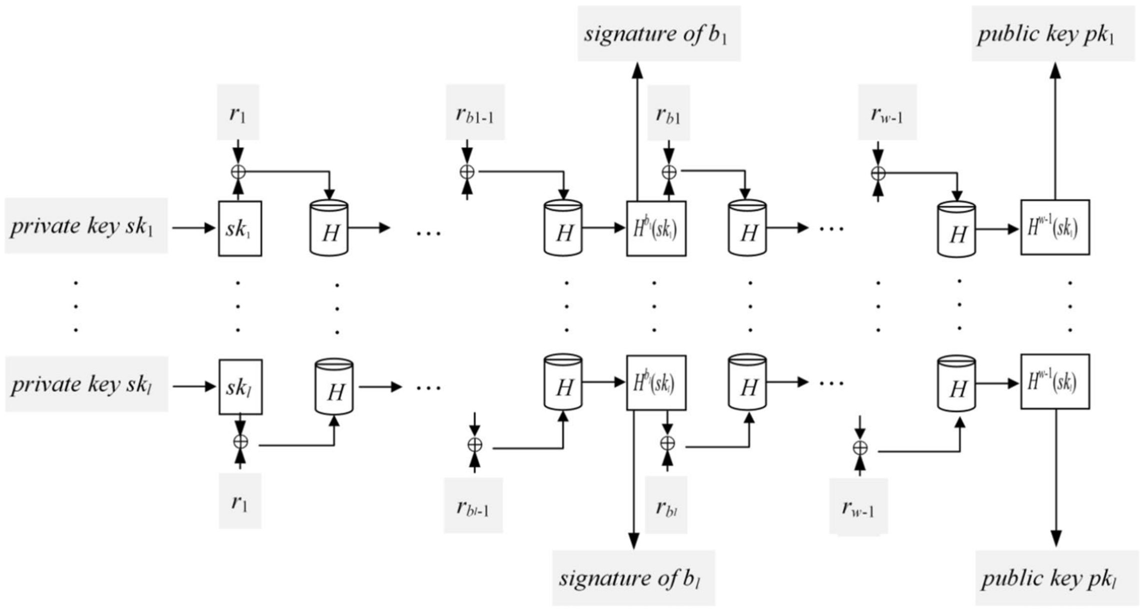

Fig. 4 W-OTS + scheme

Winternitz OTS is EUCMA when the used hash function is one-way. Here, we omit the reduction for concision.

It is not discussed that whether or not the message to be signed needs to be compressed into a digest in Winternitz OTS in Merkle (1989), we come to the above conclusion in the assumption that the message will not be compressed by a hash function before being signed, and if on the contrary, no better conclusion than collision resistance can be drawn.

(4) W-OTS+

A. *Idea and Scheme*

W-OTS+ presented in Hülsing (2013) still follows the construction of WOTS as shown in Fig. 4, the main difference between them is that W-OTS+ uses a new iteration function h_i (sk, x) constructed as follows.

For Winternitz OTS scheme Sig = (Gen, Sign, Ver), a one-way function family F is used: $F := \{f_k : k \times (0,1)^n \rightarrow (0,1)^n | k \in (0,1)^n\}$, k is chosen from $\{0, 1\}^n$ uniformly at random. Definitions and assignment of two parameters w, l are same as that of WOTS. On input the iteration counter $i \in \{0, \ldots, w-1\}$, randomized element $r = (r_1, \ldots, r_j) \in \{0, 1\}^{n*j}$ where $j \geq i$. the iteration function $h^i(x, r)$ is constructed by repeating the function $h(x)$ i times, $h^i\,(x, r) = h^{i-1}\,(x, r) \oplus r_i, h^{\,0}(x) = x$.

W-OTS+ is only designed to sign the message of fixed length, similarly a collision resistant hash function should be utilized for message of arbitrary length.

B. *Security*

It has been proven that the W-OTS+ is EUCMA as long as the used hash function family is second preimage resistant and undetectable one-way function family.

The bit security of Winternitz OTS is n-w-1-2log(w), compared with n-log($w^2l + w$) of that of W-OTS+. It can be concluded that as a negative impact on the security level, w has a linear effect on Winternitz OTS whereas logarithmic effect on W-OTS+. In consequence, it is more autonomous in setting security level by using W-OTS+ in implementations.

We describe a variant of W-OTS+ in RFC8391 (Hülsing et al. 2018) in "Typical Schemes" section. In order to randomize each call of the iteration function, a unique pair of key and bitmask is used, which is generated by a PRF taking a seed key and a unique address as input at the cost of two additional PRF calls. For more details see "Typical Schemes" section.

Performance and security comparison

Here, we compare the performance and security level of Lampot OTS, Merkle OTS, Winternitz OTS and OTS+ in the average size of signature key, verification key, signature, the average time cost of key generation scheme Gen, signature scheme Sig, verification scheme Ver, and the security level, separately, as shown in Table 1, where k denotes the bit length of message, n denotes the input and output bit length of hash/one-way function, w and l follow the definition in WOTS and W-OTS+. We measure the average timing in terms of evaluations of underlying hash/ one-way function, we just concern the time cost related to the computations of hash/one-way function, and omit the time cost of initial elements generation such as generating the private key and randomized elements, we also ignore the computation cost of XOR operation, as it can be neglected compared with the operation cost of hash function. As shown above, in terms of former two OTS schemes, although the key generation, signature and verification of are very efficient, the scale of the signature and corresponding key pair is quite large, and only limited decrease has been made in Merkle OTS compared with Lamport OTS in performance. The latter two OTS schemes reduced the key pair size and signature size significantly at the cost of additional computation, they perform similar in size of key pair and signature and the time cost when ignoring the producing cost of randomized elements and computation cost of XOR operation. The main difference between Winternitz OTS and W-OTS+ is the influence of parameter w on the security levels. That is, w impacts the security level of Winternitz

Table 1 Performance and security comparison of OTS schemes

Average size				**Average timing (evaluation times of underlying hash function)**			**Security level**
Scheme	**Signature key**	**Verification key**	**Signature**	**Gen**	**Sig**	**Ver**	
Lampot OTS	$2nk$	$2nk$	nk	$2k$	0	k	n
Merkle OTS	$(k + \lfloor \log k \rfloor + 1)n$	$(k + \lfloor \log k \rfloor + 1)n$	$\left(\frac{k}{2} + \lfloor \log k \rfloor + 1\right)n$	$k + \lfloor \log k \rfloor + 1$	0	$\frac{k}{2} + \lfloor \log k \rfloor + 1$	n
Winternitz OTS	ln	ln	ln	$l(w-1)$	$\frac{l(w-1)}{2}$	$\frac{l(w-1)}{2}$	$n-w-1-2log(lw)$
WOTS+	ln	$(l+w-1)n$	ln	$l(w-1)$	$\frac{l(w-1)}{2}$	$\frac{l(w-1)}{2}$	$n-log(w^2l+w)$

OTS negative linearly while impacts W-OTS+negative logarithmically in sense of security level, which leads that w will be limited if we target a specific security level in WOTS compared with W-OTS+.

FTS

In this section, we introduce several FTS schemes, including Biba, Hors, Pors.

Idea

FTS, as the name suggests, can be used to sign more than one message with the same key pair. Although all FTS schemes uses the single key pair to sign more than one message in contrast to one of OTS scheme, the security of the signature decreases with the number of messages to be signed. Examples for FTS schemes are Biba, Hors, Pors, Fors, etc. Biba FTS can achieve high verification efficiency at the cost of signature generation time as well as the public key scale, it is suggested to use Biba in the secure broadcast communication protocol to achieve low communication overload and high robustness to packet loss. On the downside, this scheme requires time synchronization between the sender and receiver, and heavy pre-computation before signature generation. Hors, Pors, Fors is mainly focus on selecting k elements which is determined by the message to be signed from a large set of t, where $t>>k$. The security of Hors is based on the subset-resilient assumption. But actually, Hors maps some messages to small subsets which are easier to be covered in a subset resilience attack. Pors solves this problem in such a way that a pseudorandom number generator (PRG) is utilized to obtain a pseudorandom subset, Fors is designed that k elements are selected from k sets of t elements each, i,e, one in each.

Typical schemes

(1) Biba

In this section we describe BiBa signature scheme mentioned in Perrig (2001), which achieves fast verification and small signature size, at the cost of signature generation time as well as the public key scale. We also describe the main idea of a secure broadcast communication protocol based on Biba mentioned in Perrig (2001).

A. *Idea*

BiBa exploits k-way ($k \geq 2$) collision of a hash function family to generate the signature. Precisely, the signer precomputes a large scale of random numbers which are authenticated by the public keys. In particular, the calculation from random numbers to public keys is one-way. Then, use the random numbers to generate BiBa signature in a way that the signature is the k random elements among them who collides in the relatively function family. The verification can easily and efficient be achieved by verify the output of the relatively function family on input the k elements of the signature separately. The verifiable random number possessed by the signer is significantly larger than that of the adversary, as a result, signer can find a signature with overwhelming probability than adversary.

B. *Scheme*

(a) *Initial Biba FTS*

i. *Scheme*

For Biba FTS scheme Sig=(Gen, Sign,Ver), the following hash function h, one-way function family G are used:

$$F:(0,1)^n\times(0,1)^{n_1}\rightarrow(0,1)^n \; h:(0,1)^*\rightarrow(0,1)^{n_2} \; G(n_2)=\left\{G_h:(0,1)^n\rightarrow[0,n-1]|h\in(0,1)^{n_2}\right\}$$

On input a message to be signed, the digest of the message $h(m)$ needs to be computed first. The signature scheme is described as follows.

Gen On input security parameter 1^λ, firstly generate t random numbers SEAL$_i$ from $(0,1)^n$, for $i=1,\dots,t$, t is polynomial in security parameter λ. Then on input SEAL$_i$ for $i=1,\dots,t$, output the public keys sequence as follows:

$$(pk_1,\ldots,pk_t)=\left(F_{SEAL_1}(0),\ldots,F_{SEAL_t}(0)\right)$$

Sig On input the message m, SEAL$_i$ $i=1,\dots,t$, choose k $SEAL_{j_p}$ by checking

$$G_{h(m)}\left(SEAL_{j_1}\right)=G_{h(m)}\left(SEAL_{j_2}\right)=\cdots=G_{h(m)}\left(SEAL_{j_k}\right),$$

where $p=1,\dots,k$, $i\leq j_p\leq t$.
Output the signature as follows:

$$\begin{aligned}\sigma &= (\sigma_1,\dots,\sigma_k)\\ &=\left(SEAL_{j_1},SEAL_{j_2},\ldots,SEAL_{j_k}\right)\end{aligned}$$

Ver On input the message m, public key sequence $(pk_1,\ldots,pk_t)$, signature sequence $(\sigma_1,\ldots,\sigma_k)$, the following compution and comparison are done in order to verify the signature:

$$\left(pk_{j_1},\ldots,pk_{j_k}\right) \overset{?}{=} \left(F_{\sigma_1}(0),\ldots,F_{\sigma_k}(0)\right)$$

$$G_{h(m)}(\sigma_1) \overset{?}{=} G_{h(m)}(\sigma_1) \overset{?}{=} \ldots \overset{?}{=} G_{h(m)}(\sigma_k)$$

ii. *Why is Biba FTS?*

In Biba FTS, for any message digest, the signer all can find k-way collisions with high probability to form the signature of the messages as long as the signer has sufficient SEALs, which are verifiable as the preimage of the corresponding public key of function F; meanwhile, we have pointed that the SEALs possessed by the signer is significantly larger than that of the adversary, signer can find a signature with overwhelming probability than adversary. Therefore, the more signatures have been generated, the more SEALs can be obtained by the adversary simultaneously, and the higher probability the adversary is with to forge a signature successfully. For a k-way collision, the probability that adversary forges a signature is as follows:

$$P = \frac{C_t^k \cdot (n-1)^{t-k}}{n^{t-1}},$$

where t is the number of SEALs revealed by previous signature.

(b) Broadcast Authentication Protocol Based on Biba

The main idea of the broadcast authentication protocol based on Biba is to use one-way hash chains to achieve the fast authentication and replenishment of SEALs. Two pseudorandom functions (PRF) F, $F\prime$ are used here to generate the SEALs and the corresponding dedicated-keys.

$$F : (0,1)^n \times (0,1)^{n_1} \rightarrow (0,1)^n,$$

$$F\prime : (0,1)^{n_1} \times (0,1)^{n_1} \rightarrow (0,1)^{n_1}$$

For the dedicated-key sequence $\{K_i\}$ where$1 \leq i \leq l$,

$$K_l \overset{\$}{\leftarrow} \{0,1\}^{n_1},\ K_i = F'_{K_{i+1}}(0)(1 \leq i < l).$$

For the dedicated-key sequence $\{\text{SEAL}_{i,j}\}$ where $i=1, \ldots, t, 1 \leq j \leq l$,

$$SEAL_{i,l} \overset{\$}{\leftarrow} \{0,1\}^{n_2},\ (1 \leq i < t)$$
$$SEAL_{i,j} = F_{SEAL_{i,j+1}}\left(K_{j+1}\right),\ \left(i = 1,\ldots,t,\ 1 \leq j < l\right)$$

The label i in the above is considered as the signature period, which is used from 1 to l in accordance with signature order. In each time period p, the $\text{SEAL}_{1,p}$, ..., $\text{SEAL}_{t,p}$ and dedicated-key K_p are active. These k colliding elements to generate the signature in Biba schemes are chosen form $\text{SEAL}_{1,p}$, ..., $\text{SEAL}_{t,p}$. In order to verify the signature, the signer publishes K_p, and sends $\text{SEAL}_{i,p-1}$ ($i=1, \ldots, t$) and K_{p-1} over an authenticated channel. The following compution and comparison are done in order to achieve the signing verification: firstly, check whether the SEALs corresponding to the k elements in the signature are generated correctly by function F on input the related SEALs of previous period and dedicated-key of current period; secondly, verify whether k elements in the signature collide with each other of function G. This broadcast authentication protocol requires time synchronization between the signer and verifier.

(2) *Hors*

A. *Idea*

Hors can be considered as a generalization of Lamport OTS proposed in 2002 (Reyzin and Reyzin 2002), in which a large number of public keys are precomputed on input the related private keys of a one-way function. The essence of Hors is to solve the problem of selecting a small subset containing k elements from a large set of t elements pseudo-randomly, $t >> k$. The message digest is interpreted as a label sequence which indicates a specific subset of the key pairs, and the private keys in this subset form the signature. In order to sign a few messages using these key pairs, the underlying hash function H used to digest the message should be r-subset-resilient, i.e., for any probabilistic polynomial-time adversary A who can output $(m_1, m_2, \ldots, m_{r+1})$ such that $H(m_{r+1}) \subseteq_{i=1}^{r} H(m_i)$ successfully with negligible probability.

B. *Scheme*

For Hors FTS scheme Sig = (Gen, Sign, Ver), the following one-way function f, hash function h are used:

$$f : (0,1)^n \rightarrow (0,1)^n$$

$$h : (0,1)^* \rightarrow (0,1)^l.$$

On input a message to be signed, digest the message m to be $h(m)$ first, then sign $h(m)$ as follows.

Gen On input security parameter 1^λ, firstly generate t n-bit private keys sk_i uniformly at random, for $i=1, \ldots, t$, t is polynomial in security parameter λ. Then, output the public keys sequence as follows:

$$(pk_1,\ldots,pk_t) = \left(f(sk_1),\ldots,f(sk_t)\right)$$

Sig On input the message m, private keys sk_i $i=1, \ldots, t$, denote $h(m)$ as $(b_1, b_2, \ldots, b_k)$ in base t representation, where $l = k * \log t$. Output the signature as follows:

$$\begin{aligned}\sigma &= (\sigma_1, \ldots, \sigma_k) \\ &= \left(sk_{b_1}, sk_{b_2}, \ldots, sk_{b_k}\right)\end{aligned}$$

Ver On input the message m, public key sequence $(pk_1, \ldots, pk_t)$, signature sequence $(\sigma_1, \ldots, \sigma_k)$, firstly digest the message m to be $h(m)$ and denote h(m) as $(b_1, b_2, \ldots, b_k)$ in base t representation, the following compution and comparison are done in order to check the signature:

$$\left(pk_{b_1}, \ldots, pk_{b_k}\right) \overset{?}{=} \left(f(\sigma_1), \ldots, f(\sigma_k)\right)$$

C. *Security*

It has been proven that the signatures are r-time EUCMA if the hash function used to digest the message is r-subset-resilient and function used to generate the public keys is one-way.

(3) *Pors*

The security of Hors is based on the subset-resilient assumption (Aumasson and Endignoux 2018). But actually, Hors maps some messages to small subsets which are easier to be covered in a subset collision attack. Pors solves this problem in such a way that a PRG is utilized to obtain a pseudorandom subset. Concretely, a hash function family H: $\{h_k : (0,1)^* \rightarrow (0,1)^n | k \in K\}$ and a PRF G: $(0,1)^n \rightarrow (0,1)^*$ is used in Pors. Firstly, on input a message m into G, choose the first k distinct output $(b_1, b_2, \ldots, b_k)$ as label sequence which indicates the specific subset of the key pairs, where each block is in base t representation, i.e., $G(m) = (b_1, b_2, \ldots, b_k)$.

The key generation is the same as that of Hors, and the above k indices are used to select k parameters as the signature value, from the private key which contains t values.

(4) *Fors*

A. *Idea*

Fors is presented in the SPHINCS + hash-based signature in 2017 (Bernstein et al. 2017), which is also a successor of Hors. As Hors selects k elements all from one set of t elements, the same indices of the message digests yield the same elements in the signature; in order to solve this issue, Fors is designed in such a way that k elements are selected from k sets of t elements each, i,e, one for each; t is set to be power of 2, t > > k. In this case, only the same indices with the same position in the message digest blocks sequence yield the same elements in the signature. a PRF F: $(0, 1)^n \times (0, 1)^{32} \rightarrow (0, 1)^n$ and a tweakable hash function H is used in Pors.

Tweakable hash utilizes a public parameter and a tweakable parameter in addition to the message as input to a hash function to generate different outputs, in order to be multi-function multi-target resistant. Several specific tweakable hash functions are introduced in Bernstein et al. (2019), one can choose one among them on demand, for more details of tweakable hash see (Bernstein et al. 2019; Hülsing et al. 2016).

B. *Scheme*

Fors FTS scheme Sig = (Gen, Sig, Ver) is described as follows.

Gen On input security parameter 1^λ, firstly choose n-bit private keys sk uniformly at random, n is polynomial in security parameter λ. Then, on input sk, the address of each private key $ADRS_i$, $i=1, \ldots, kt$, output the private keys sequence as follows:

$$(sk_1, \ldots, sk_{kt}) = (H(sk, ADRS_1), \ldots, H(sk, ADRS_{kt}))$$

For each set $(sk_{it+1}, \ldots, sk_{(i+1)t})$, $i=0, \ldots, k\text{-}1$, use the tweakable hash function H to build a binary tree with height of $\log t$, and the roots of k trees consist of the set of $(pk_1, \ldots, pk_k)$, which is then input into a tweakable hash function, generating the final public key pk.

Sig On input the message m, private keys sk_i $i=1, \ldots, kt$, denote $h(m)$ as $(b_1, b_2, \ldots, b_k)$ in base t representation, where $l = k * \log t$. Output the signature as follows:

$$\begin{aligned}\sigma &= (\sigma_1, \ldots, \sigma_k) \\ &= \begin{pmatrix} sk_{b_1},\ path_{auth,1},\ sk_{b_2+t},\ path_{auth,2}, \\ \ldots, sk_{b_k+(k-1)t},\ path_{auth,k} \end{pmatrix}\end{aligned}$$

where $path_{auth,i}$ is the authentication path in the ith tree, authentication path of the hash tree will be thoroughly discussed in section.

Ver On input the message m, public key pk, signature σ, firstly digest the message m to be $h(m)$ and denote $h(m)$ as $(b_1, b_2, \ldots, b_k)$ in base t representation, the following compution and comparison are done in order to check the signature:

Firstly, invoke the tweakable hash function on input $\left(sk_{b_i+(i-1)t}, path_{auth,i}\right)$ to compute PK_i, $i=1, \ldots, k$. Then, $(pk_1, \ldots, pk_k)$ is input into the tweakable hash function, generating the final public key pk'.

Finally, compare $pk' \overset{?}{=} pk$, if yes, it is a valid signature.

Table 2 Performance and security comparison of FTS schemes

Size				Timing (evaluations of hash function)			Security level
Scheme	Signature key	Verification key	Signature	Gen	Sig	Ver	
Biba	tn	tn	kn	t	$(t+1)/P_s$	$2k+1$	$(kr-1)\log n - \log C_{kr}k - (kr-k)\log(n-1)$
Hors	tn	tn	kn	t	$1/P_s$	$k+1$	$k(\log t - \log k - \log r)$
Pors	tn	tn	kn	t	$1/P_s$	$k+1$	$k(\log t - \log k - \log r)$
Fors	ktn	n	$(k+logt)n$	kt	$1/P_s$	$klogt+k-1$	$-\log\left[(1+q)\sum_{\gamma}((1-(1-\frac{1}{t})^{\gamma})^{k}\binom{q'}{\gamma}(1-\frac{1}{2^h})^{q'-\gamma}\frac{1}{2^{h\gamma}})\right]$

Performance and security comparison

Here, we compare the performance and security level of Biba, Hors and Pors in the size of signature key, verification key and signature, the time cost of key generation scheme Gen, signature scheme Sig, verification scheme Ver and the security level, separately, as shown in Table 2. Without loss of generality, in all schemes t denotes the number of key pairs (also the number of private keys and public keys separately), k denotes the component number in the signature, n denotes the input and output of hash/one-way function, r denotes the signature times, l and n_1 follow the definition in broadcast authentication protocol based on Biba, q and q' denote the number of queries to oracle and hash function separately, Ps denotes the probability that the signer can find a signature in one trial. We measure the timing in terms of evaluations of underlying hash function/one-way function/PRF. We compare the security level of the above scheme in r-non-adaptive-chosen-message attack model, in which the adversary queries signatures r times on r messages of his choice, and then he tries to forge a signature on a new message m of his choice.

As shown above, all these schemes can be used to sign t messages with the same key pair and achieve high verification efficiency at the cost of signature generation time as well as the public key scale. As an improvement of Hors, Pors can achieve a higher security level.

Tree-based public key authentication

We have pointed out the main drawback of both OTS and FTS schemes is that one key pair can only be used to sign one or a few messages. In a signature scheme, we authenticate the message based on the integrity of public key. As a result, to authenticate public key on a large scale has become a vital issue in the research of hash-based signature.

The tree-based public key authentication is a classical way to solve the public key management issue. Furthermore, as the key pair has to be changed into a fresh one after each signature generation, a natural way is to utilize the state management to keep record of the state and synchronization between signer and verifier, otherwise there may exists reduplicate use of the signature key, such that the adversary can forge a valid signature easily. Therefore, state management plays an essential role in many scenarios of the application of hash-based signature, but it also limits it less practical. Stateless hash-based signature settles state management at the cost of significantly higher signature size compared with the stateful ones in the analogous security level.

In this section, we focus on different constructions and security analysis about hash-tree based public key authentication of hash-based signature, classify hash-based signature into limited number and stateful schemes, unlimited number and stateful schemes and unlimited number and stateless schemes, the goal is to present, classify and discuss different solutions for tree-based public key authentication.

Different ideas

(1) Limited number or (cryptographic) unlimited number

A. *Down-top or top-down*

Tree-based authentication scheme efficiently using the hash tree to authenticate public keys is wildly adopted to solve the problem of key management. To sign a message, after the OTS is generated, the following parameters, including the message, the OTS and its index, the one-time public key, along with an authentication path generated by the tree-based authentication scheme, have to be applied to verifier. Verifier authenticates the signature in two steps. Firstly, verify the OTS by the first four parameters of the signature; Secondly, authenticate the one-time public key along with the authentication path to construct the tree from leaf nodes to root, or from the root to bottom, which comes into two main different approaches to authenticate public keys in large scale, depending on signing limited number or cryptographic unlimited number of messages.

In terms of authentication down-top, the root, i.e., the overall public key, which can be used to verify all of the public key as leaf nodes, should be constructed form the leaf nodes in the bottom layer to the top. In terms of authentication top-down, along this direction one can establish the trust relationship, as well as the tree with unlimited number leaf nodes, each node in the tree can be constructed independent in contrast to the down-to the tree with unlimited number leaf nodes, each node in the tree can be constructed independent in contrast to the down-top approach that the interior nodes need to be constructed by its children nodes, the private key corresponding to the node on the higher layer which has already been authenticated is used to sign the node on one layer lower, the building-up of all these signatures in the authentication path is based on the authenticated root, along top-down direction one can establish the trust relationship.

Specifically, in the case of down-top authentication, we set all the public keys to be leaf nodes of a binary hash tree, these nodes above the lowest layer are calculated from the output of hash function on input the two closest nodes, called children nodes one level lower. For a binary tree whose height we denote as h, actually, it has 2^h leaf nodes and $h+1$ layers, which are ranged from 0 to h from bottom up. All of the nodes are denoted as $N_{i,j}$, where $i=0, \ldots, h$, denotes the height of the node counted from bottom to top in the tree, $j=0, \ldots, 2^i-1$, denotes the horizontal index counted from left to right. To authenticate the ith public key, an inner node sequence is used as the authentication path which consist of the sibling nodes on the path from $\mathrm{N}_{0,i}$ to root $\mathrm{N}_{h,0}$. As shown in Fig. 5, to authenticate $N_{0,4}$, the corresponding authentication path is $N_{0,5} \rightarrow N_{1,3} \rightarrow N_{2,0} \rightarrow N_{3,0}$ in order, along with the message, the OTS and its index have to be applied to verify the signature in the verification phase. In many cases the one-time public key can be calculated from the message and the OTS, and needs not to be transmitted to the verifier as a consequence. The nodes in the authentication path should be used in order, with $\mathrm{N}_{0,i}$ as the initial one, to generate the root node, the signature is valid as long as the root node generated by the verifier equals the authenticated one.

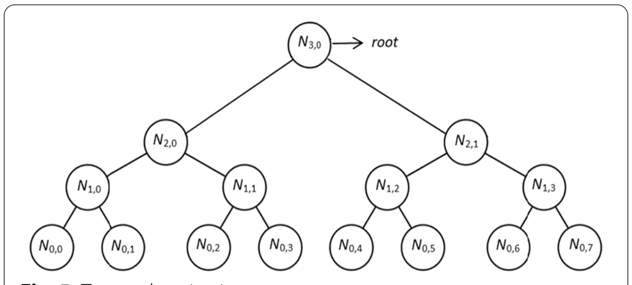

Fig. 5 Tree authentication

In the case of top-down authentication, firstly, one generates OTS key pairs randomly or pseudorandomly, here we will not discuss how to generate these key pairs, which depends on the specific application scenario; then, construct the authentication path in the opposite directions compared with the above approach, i.e., from root node to leaf node, the private key related to the parent node is used to sign its child node in the top-down order. As shown in Fig. 5, to authenticate $N_{0,4}$, the corresponding authentication path is $N_{3,0} \rightarrow N_{2,1} \rightarrow N_{1,2} \rightarrow N_{0,4}$ in order. Specifically, $N_{3,0}$ is the overall public key of the whole system whose integrity has already been authenticated, i.e., the private key corresponding to $N_{3,0}$ is used to sign the node $N_{2,1}$, and so on, till $N_{0,4}$, the building-up of all these signatures in the authentication path is based on the authenticated $N_{3,0}$. This approach is similar to the case of hierarchical public key infrastructure (PKI), where public key of root CA is signed by a trusted authority, and corresponding public key certificate of root CA has been distributed to all other entities in this PKI system, and can be verified when necessary. Take three layers PKI for example, the root CA is on the highest layer will the end entities in on the lowest layer. The intermediate CA certificate is issued by Root CA, i.e., the public key of the intermediate certificate is signed by the Root CA's private key. The intermediate certificate then issues certificates to end entities. Authentication is done from top to bottom.

For the two tree authentication approaches mentioned above, it is a natural way to construct the authentication tree by the first approach as long as the number the signatures is acceptable, and there no other computing expense except signing the original message and generating the related authentication path in the tree. Nevertheless, the key generation cost goes up exponentially with the number of layers in the tree in this approach. Compared with the first approach, the number of signatures generated by the tree is no longer fixed in the second one, and it exponentially reduces the computing and storing cost to generate the authentication path at the expense of linearly increasing signatures with the quantity of the layers, which are generated by the private key corresponding to the node on the higher layer. In consequence, the top-down authentication is a more flexible way to sign unlimited number of messages.

B. *Hyper-tree*

For a larger scale hash-based signature signing enough messages, it is common to use the hyper-tree trees which combines both of the two approaches above, such as GMSS, XMSS^{MT} and HSS which are the hyper-tree variants of Merkle hash-based signature, LMS and XMSS separately; SPHINCS with many of its variants such as SPHINCS+, Gravity SPHINCS, etc., allowing to make a tradeoff between the signature size and time by the number of layers.

A whole tree of height h is constructed by d layers subtrees of height $(h+1)/d$-1, where h, $d \in N$, $d|h+1$. The whole tree can be used to sign 2^h messages while single sub-tree to sign a group of $2^{(h+1)/d}$ public keys or messages in the hash-based signature, these sub-trees will be used consecutively and only one sub-tree on each layer needs to be computed.

By using the multi-layer sub-trees, firstly, the key generation time reduces from $O(2^h)$ to $O(2^{h+1/d})$ at the cost of additional $(d–1)$ signatures. The computation scale reduces approximately exponentially with increase of the number of layers, as a consequence, can meet the requirement of signing enough message in the resource-limited environment; secondly, using a single tree to generate a large scale of signature keys will inevitably lead to a variety of problems in key storing and distributing. When the signature key is copied for distributing to other application module, there is risk of reuse. By using the multi-layer sub-trees, one can using a single sub-tree to generate the one-time key pairs for one specific usage, by doing this, the above problems are easily solved.

(2) Stateful or stateless

The hash-based signature schemes commonly used in practice are stateful due to the underlying OTS/FTS, the signature key needs to be renewed when exceeding its service time, namely, the signing times it can be used to sign the messages, thus it is necessary to update the internal key state over time, otherwise, it is feasible for the forger to generate a new signature without recovering the signature key. Although it is a natural way to use this stateful approach to construct a hash-based signature scheme, it is not in line with the standard definition of digital definition, in which there is no requirement about the key state management; moreover, state management may be difficult to implement in a cryptosystem with limited functions. In consequence, how to design stateless hash-based signature has become a raised concern.

In the design of the stateless hash-based schemes, to select a signature key pseudorandomly when signing a message, to sign as many messages as possible on the same system scale, to design a tree in a much larger scale and to efficiently generate each sub-tree are all the key issues needed to be fulfilled in the stateless schemes. See “Typical schemes” section for detailed analysis.

Typical schemes

(1) Limited number and stateful schemes

As analyzed above, to build authentication path from bottom top and update the signature key after exceeding its service time, is the most fundamental way to construct hash-based signature.

A. *Merkle tree*

Merkle tree is proposed by Merkle in 1979 (Merkle 1989), which is the original tree authentication scheme following the approach of signing limited number of messages mentioned above. A hash function H:$\{0, 1\}^{2n} \to \{0, 1\}^n$ is utilized to construct the nodes in it. A Merkle tree is a binary tree with 2^{h} leaves on the bottom layer, whose height we denote as h. for interior nodes we denote the two nearest nodes on the first lower layer as its children nodes, the interior nodes values are set in such a way that it is the output of the hash function H on input their children node values, i.e.,

$$N_{i,j} = H\left(N_{i-1,2j}||N_{i-1,2j+1}\right)$$

where $i = 1, \ldots, h, j = 0, \ldots, 2^i - 1$.

The interior nodes are computed from left to right, from bottom to top sustainedly, and we denote the node on $h+1$th layer as Root.

A hash-based signature consisting of OTS and Merkle tree works in the following way. Firstly, a Merkle tree is constructed whose leaf on the lowest layer is the one-time public key (or its digest). The Root is authenticated by public key authentication technology such as digital certificate. The Root is the overall public key which will be used to authenticate all the signatures constructed by the one-time key related to the leaf in the Merkle tree.

To sign a message, the one-time private key is used to generate the one-time signature. Then, the following data related to the signature, including the message, the one-time signature and its index, the public key, along with authentication path by the tree authentication scheme, have to be applied to verifier. Before the verification, the authentication of the Root, such as the root's digital certificate, has to been sent to verifier in advance. The verifier authenticates the Root first, then verifies the one-time signature, finally validates the one-time public key using the authentication path sequentially from bottom up to calculate the tree root, and compare the calculated root with Root, if they are equal, the signature is valid.

Treehash algorithm has been presented to compute the root of the Merkle tree with lower storage requirement. The basic idea is to use a stack along with the push and pop operations to store the value of nodes when necessary. The treehash algorithm is also used in the BDS algorithm as a subroutine. For more details see (Buchmann et al. 2008).

B. *XMSS*

XMSS, short for eXtended Merkle Signature Scheme proposed in 2011 (Buchmann et al. 2011b), can be considered as an improved version of SPR-MSS and achieves forward secure and EUCMA if the underlying hash function family is second preimage resistant.

a) *Idea*

The original version of XMSS uses WOTS as its underlying OTS scheme (which is changed to W-OTS+ in the later version), and an XOR tree as its authentication scheme. First, in order to achieve forward secure, XMSS makes the following changes to the underlying OTS scheme: using a forward-secure PRG to generate the signature key sequence instead of generating them randomly. It has been proven by Krawczyk (2000) that an unforgeable forward-secure signature scheme can be constructed by combining a forward-secure PRG to generate the signature keys and an unforgeable signature scheme to generate the signature. In this way, XMSS achieves forward secure analogously. Second, in the public key authentication phrase, XMSS uses the XOR tree authentication proposed by Bellare and Rogaway, to construct unforgeable hash-based signature scheme, in order to relax the security assumption of the underlying hash function to second preimage resistance instead of collision resistance. In this way, the signature size is reduced in the analogical security level. Compared with SPR-MSS which use Lamport OTS as its underlying OTS in its reduction to compute its security level, the usage of WOTS in XMSS reduces the signature size by more than 75% in the analogical security level.

(b) *Scheme*

As we have discussed about WOTS scheme in "Typical schemes' section, here, we just introduce the one-time key generation scheme and tree authentication scheme of XMSS.

i. *One-Time Key Generation*

A PRF $F(n)=\{f_k : (0,1)^n \to (0,1)^n | k \in (0,1)^n\}$ is used in XMSS to iteratively generate the state in each Winternitz OTS started from a random initial state *SEED*. Precisely, to generate each Winternitz OTS signature key, beginning with the initial state *SEED*, XMSS uses the state $State_{i-1}$ of the i-1th Winternitz OTS to produce a new state $State_i$ and a new $SEED_{OTS_i}$ of the ith Winternitz OTS, and then use $SEED_{OTS_i}$ to generate the one-time private key sequence sk_i, i.e.,

$$State_0 = SEED$$
$$State_i || SEED_{OTS_i} = f_{State_{i-1}}(0) || f_{State_{i-1}}(1)$$
$$sk_{OTS_i} = f_{\mathrm{SEED}_{OTS_i}}(0) || \cdots || f_{\mathrm{SEED}_{OTS_i}}(l-1).$$

where $i = 1, \ldots, l$.

The public key pk_i are computed in the same way as that of WOTS on input the corresponding sk_i.

The one-time key pair generation above using the iterative PRG to achieve the key-evolving has already been proven to be the forward-secure PRG (Krawczyk 2000).

ii. *Tree authentication*

Two type of authentication tree, L-tree and XMSS-tree, are used in the XMSS tree authentication scheme. The former is to construct each leaf node of the XMSS tree on input the one-time public key, whereas the latter is to reduce the authentication of all these one-time public keys to a single root of XMSS-tree.

Both the two kinds of trees use the following algorithm to construct the whole tree: A keyed hash function $H_{ht}(n) = \{h_k : (0,1)^{4n} \to (0,1)^n | k \in (0,1)^n\}$ is utilized to construct the nodes on the tree. Similar to that in the Merkle tree, the two nearest nodes on the first lower layer in the tree are defined as the children nodes of the interior nodes. For a tree whose height we denote as h, the bitmasks $b_{l,j} || b_{r,j}$ are chosen uniformly at random from $\{0,1\}^{2n*h}$, k is chosen uniformly at random from $\{0,1\}^n$. All of the nodes are denoted by $N_{i,j}$, where i denotes the height of the node counted from bottom to top in the tree, j denotes the horizontal serial index counted from left to right, both i and j are counted from 0. The interior nodes values are calculated as:

$N_{i,j} = H_{ht}(N_{i-1,2j}||N_{i-1,2j+1} \oplus b_{l,j}||b_{r,j})$,where $i = 1, \ldots, h, j = 0, \ldots, 2^i - 1$.

The interior nodes are computed from left to right, from bottom to top sustainedly.

The main differences between L-tree and XMSS-tree are as follows. Firstly, the number of leaf nodes of XMSS-tree must be set to the power of 2, but it is not necessary for L-tree, which leads that the last node in some layers of L-tree may have no sliding node. In this case, this last node will be raised to the lowest higher layer in which it has the sibling node in the L-tree. Secondly, the leaf nodes of each L-tree are the public keys from the underlying OTS scheme, whereas the leaf nodes of each XMSS-tree are the root nodes of each L-tree.

The verification is done similarly as in the Merle-tree hash-based signature. The message, OTS, the state, the authentication path along with the public key of the XMSS-tree has to been sent from the signer to verifier. The verifier authenticates the message by verifying OTS first, then validates the one-time public key by using the nodes in the authentication path sequentially from bottom up to calculate the tree root, and compare it with the received one, the signature is valid if they are equal.

iii. *Security*

By using a forward-secure PRF $F(n)$, a second preimage resistant hash function family H_{ht} as well as W-OTS+, XMSS achieves forward secure and EUCMA.

B' *a Variant of XMSS in RFC 8391*

In RFC 8391 (Hülsing et al. 2018), a variant of XMSS was proposed using variants of W-OTS+and tree authentication scheme, the EUCMA of this variant can be deduced from multi-function multi-target second preimage resistance of the underlying function.

(a) *Idea*

Multi-function multi-target notions of preimage, second preimage resistance and eTCR, which have been proven to have identical or similar security as the standard notions of that in single-function single-target notions (Hülsing et al. 2016). However, initial XMSS along with its tree and multi-tree variant, like XMSS^{MT} and SPHINCS, are referred to single-function multi-target notions, where an attacker obtains d target images or preimages together with a random function from the hash function family, he attacks successfully if he can just find one preimage or second preimage for d targets with non-negligible probability. It has been proven that for single-function multi-target attack, the complexities are reduced by d and $\sqrt{d}$ in classical and quantum attacks respectively. By using independent keys and bitmasks to randomize each call of the PRF and the hash function in the OTS and the hash tree, the EUCMA of XMSS can be deduced from multi-function multi-target second preimage resistance of the underlying function.

(b) *Scheme*

i. *A Variant of W-OTS+*

In this section, we introduce the address generation scheme, iteration function, Key generation, signature and verification schemes used in this variant of W-OTS+.

- *An Address Generation Scheme*

In order to randomize each call of either the PRF or the hash function, a unique pair of key and bitmask is used, which is generated by a PRF taking a seed key and a unique address as input. We describe the address generation scheme in detail as follows.

For different usages in OTS, L-tree, and multi-tree, the addresses are generated in different way. Generally, the address begins with the layer address and tree address, which separately describe the height of a tree and the position in a layer, to indicate the position of a specific tree in multi-tree and are set to zero in the case of a single tree as in OTS and L-tree. The third parameter called type is used to distinguish the different usages of this address, which is set to different values to indicate the OTS address, L-tree address, sub-tree address respectively. For a OTS address, the next three words are OTS address, chain address and hash address which describe the position of the OTS in the tree, the chain address and the index of the hash call in the chain. For an L-tree address, the next three words are the L-tree address, tree height and the tree index which describe the index of the corresponding leaf calculated with this L-tree, the height of the node for the next computation and the position of the node at this height in the L-tree. For a multi-tree address, the next three words are zero padding, tree height and the tree index, the latter two describe the height of the node for the next computation and the position of the node at this height in the multi-tree. For the three types of addresses, the last parameter called keyAndMask indicates the different usage of the address, which is set to different values for generating the key, the most significant bytes and the least significant bytes of bitmask.

- *Iteration Function*

Compared with the initial W-OTS+, this variant randomizes each call of the iteration function family $H(n) = \{h_k : (0,1)^n \times (0,1)^n \rightarrow (0,1)^n | k \in (0,1)^n\}$ by a PRF $G(n) = \{g_{SEED} : (0,1)^n \to (0,1)^n | SEED \in (0,1)^n\}$ on input a seed key and the corresponding address to generate a key of H and a bitmask with whom the intermediate value will XOR with. Definition of parameters w is same as in W-OTS+. On input a value $x \in \{0, 1\}^n$, an iteration counter $i \in \{0, \ldots, w-1\}$, a public seed key $SEED$, an address $ADRS_{key,i}$ or $ADRS_{mask,i}$ with index indicating its usage in the ith iteration, the new iteration function $h^i(x)$ with G works as follows.

$$\begin{aligned} &G(SEED, ADRS_{key,i}) = k_i \\ &G(SEED, ADRS_{mask,i}) = Mask_i \\ &h^0(x,\ seed) = x \\ &h^i(x,\ seed) = h\left(k_i, h^{i-1}(x,\ seed) \oplus Mask_i\right) \end{aligned}$$

The iteration function iterated as needed times returns the output of any length of this chain, i.e., iterating from the initial state or any intermediate state any rounds as needed.

The above is an instance of tweakable hash. For more discuss of tweakable hash see "Typical schemes" section.

- *Key Generation, Signature and Verification*

The private keys are either chosen uniformly at random or generated pseudorandomly. Similar as in the initial W-OTS+, the public keys are generated by iterating the private key w–1 times, while signature and verification are generated by iterating either the private key or the signature certain times determined by the message to be signed, with only difference of the above new iteration function used.

ii. *A Variant of XMSS Tree Authentication*

The variant of XMSS in 8391 uses the above W-OTS+variant as its OTS scheme, before inputting into the W-OTS+variant, the message of arbitrary length is compressed by a hash function $H_{dgt} : (0,1)^n \times (0,1)^n \times (0,1)^n \times (0,1)^* \to (0,1)^n$ to n-bit message digest; after OTS scheme, a specific tree authentication scheme with the $PRF\ G(n)$ required in the above W-OTS+and a hash function $H'_{ht}(n) = \left\{h'_{ht,k} : (0,1)^n \times (0,1)^{2n} \to (0,1)^n | k \in (0,1)^n\right\}$ will be used to compute a authentication path, as shown in Fig. 6. Finally, the private key is renewed to prepare for the next signature. We describe the message digest scheme and the tree authentication scheme in detail.

- *Message Digest*

Choose private key sk_{dgt} uniformly at random from $\{0, 1\}^n$; on input sk_{dgt}, public $SEED$, public key PK_{root}, message m, index of the signature IDX,

Calculate

$$\begin{aligned} &Rand = H_{rdm}(sk_{dgt}, IDX); \\ &M_{dgt} = H_{dgt}(Rand || PK_{root} || IDX, m). \end{aligned}$$

The generation of PK_{root} is described in the following tree authentication scheme.

- *Tree Authentication*

In the variant of XMSS in 8391 (Hülsing et al. 2018), as shown in Fig. 6, each subroutine H_{ht} used to the

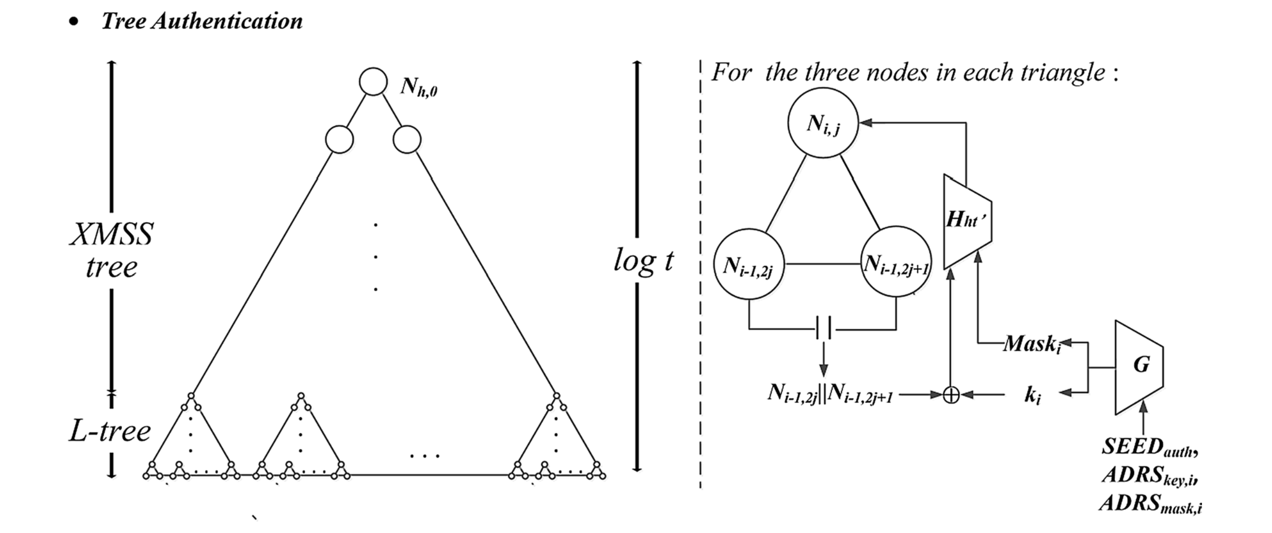

Fig. 6 variant of XMSS in 8391

XMSS tree is randomized by using a PRG on input a seed key and the corresponding address to generate a key of H_{ht}' and two bitmasks with whom the intermediate values will XOR with in H_{ht}'.

The variant of XMSS still uses L-tree and XMSS-tree of similar structure in constructing authentication tree, but with the distinct tree function H_{ht}' as follows.

The bitmasks $Mask_i$ and function key k_i of H'_{ht} are generated by function $G(n)$ of public *SEED* and corresponding address. The interior node value $N_{i,\,j}$ of the hash tree is calculated by function H_{ht}' as:

$$G(SEED, ADRS_{key,i}) = k_i$$
$$G(SEED, ADRS_{mask,i}) = Mask_i$$
$$N_{i,j} = h'_{ht}(k_i, N_{i-1,2j}||N_{i-1,2j+1} \oplus Mask_i)$$

where $N_{i\text{-}1,2j}$ and $N_{i\text{-}1,2j+1}$ are two nearest nodes on the first lower layer in the tree, $i=0, \ldots, h, j=0, \ldots, 2^i\text{-}1$.

The nodes on the bottom layer of the L-tree are each component of one public key, whereas those of XMSS-tree are the root nodes of each L-tree. The interior nodes are computed from left to right, from bottom to top sustainedly, and the root of XMSS-tree is PK_{root}.

The authentication path of a public key node $N_{0,i}$ consists of the sibling nodes on the path from $N_{0,i}$ to root $N_{h,0}$.

- *Security*

In contrast to the original XMSS, in which once the key is randomly chosen, the function will be fixed until the whole tree is constructed, and all the bitmasks in the tree are chosen uniformly at random as well, tree authentication scheme in this variant by using independent keys and bitmasks to randomize each call in the OTS and the hash tree, can achieve multi-function multi-target resistance in the related security assumption of hash function.

It is obvious to see that if the signature is EUCMA, the adversary cannot find a second preimage (*Rand*', *m*') of (*Rand*, *m*) under H_{dgt} in message digest phase, otherwise, the adversary can output a forge a signature by simply replace (*Rand*, *m*) with (*Rand*', *m*') in a valid signature, it is a more stringent security assumption than *eTCR*.

The literature (Aumasson and Endignoux 2018) gives the security reduction of XMSS-T. The difference between XMSS-T and the variant of XMSS are: first, two secret keys must be generated pseudorandomly by function H_{prf}; second, PK_{root} is not part of the input parameters in the message digest function H_{dgt}'; therefore, H_{dgt}'only needs to be (post-quantum) eTCR secure in XMSS-T. XMSS-T is post-quantum EUCMA under the following assumptions: the hash function family $H(n)$ in the variant of W-OTS+and H'_{ht} (n) in tree authentication scheme are post-quantum multi-function multi-target second pre-image resistant, where F has at least two preimages; H_{rdm} and H_{prf} are PRF; $G(n)$ needs to be programmed as quantum random oracle, the message digest function H_{dgt}' to be post-quantum eTCR secure.

C. LMS

(a) *Scheme*

The LMS scheme was proposed in 1995 as a USA patent by Leighton and Micali (1995), and recently was published as RFC8554 by IETF (McGrew et al. 2019). The LMS scheme also can be considered as an improved version Merkle scheme, and can achieve stronger security by using different prefix in its iteration function. The main difference between LMS and Merkle scheme lies in the iteration function from the perspective of designing structure. Here, we mainly introduce the iteration function of the version in RFC8554.

LMS randomizes each call of the iteration function $h : (0, 1)^n * (0, 1)^n \rightarrow (0, 1)^n$ by an additional prefix as part of the input. Precisely, when using the iteration function h in the OTS of LMS, on input a value $x \in \{0, 1\}^n$, iteration counter $i \in \{0, \ldots, w-1\}$, $PRIFIX_i$ which is related to sub-tree address, the position of the OTS in the sub-tree, the chain address and the index of the hash call in the chain, etc. The iteration function $h^i(x)$ works as follows.

$$h^0(x) = x$$
$$h^i(x) = h\left(PRIFIX_i||h^{i-1}(x)\right)$$

When using it in the tree authentication generation, the parameter $PRIFIX_i$ is related to node number and some fixed values, $N_{i\text{-}1,2j}$ and $N_{i\text{-}1,2j+1}$ denote the values of two children nodes. The iteration function $H'_{ht}(x)$ is changed into:

$$h_{ht}(x) = h'_{ht}(PRIFIX_i||N_{i-1,2j}||\,N_{i-1,2j+1})$$

where $i=0, \ldots, h, j=0, \ldots, 2^i-1$.

For more recommended parameters see (Cooper et al. 2019).

(b)*Security*

As we can see from above, different prefixes are used in each iteration function, as a result, the following attack will be prevented: the adversary tries some random inputs of H or or H'_{ht} or the hash function used to generate the one-time public key by digesting the concatenation of each single public key component, to check if the related outputs match the outputs of these functions, in some sense this kind of attack is essentially equivalent

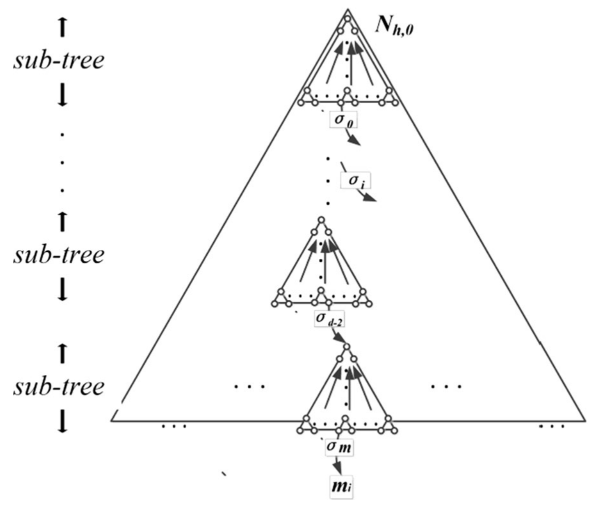

Fig. 7 structure of unlimited number and stateful hash-based signature

to multi-function multi-target attack proposed in Hülsing et al. (2016). The iteration functions of OTS and hash tree both need to be second preimage resistant. The literature (Eaton 2017) gives the security reduction of LMS in quantum random oracle model.

Security analysis of LMS is given in the random oracle model; in addition, the hash compression function is also needed to be a random oracle, which is less convincing compared with that of initial XMSS analyzed in the standard model (in the variant of XMSS in RFC8391, PRF $G(n)$ used to generate the bitmask and key for each call of hash function, needs to be programmed as random oracle).

In terms of implementation performance, the prefix values in LMS are just related to position and some fixed values that can be prepared in the initialization phase, compared with some parameters of the iteration in the XMSS dependent on the output of the previous round, LMS is faster than XMSS in the signature generation. Meanwhile, LMS is more flexible and adaptable, one can implement initialization and signature generation on separated devices.

(2) *Unlimited Number and Stateful*

As the name suggests, unlimited number and stateful hash-based signature can sign cryptographic unlimited number of messages, meanwhile, the signature key must be renewed after exceeding its service time. One can use the limited number and stateful hash-based signature schemes as the sub-tree to construct the corresponding hyper-tree schemes to sign enough messages. Most of the limited number and stateful hash-based signature schemes have their own unlimited number and stateful versions, such as the GMSS, XMSS$^{\mathrm{MT}}$ and HSS which are the multi-tree variants of Merkle tree scheme, XMSS and LMS separately, which we will introduce in this section.

Generally, by using its limited number and stateful version as the sub-tree, the unlimited number and stateful scheme builds a whole tree of height h by d layers sub-trees of height $(h+1)/d$–1, where h, $d \in N$, $d|h+1$, as shown in Fig. 7. As a stateful scheme, one can make full use of all the one-time/few-time signature keys with the key state management algorithm, as a consequence, $k{*}2^{h}$ signatures can be made by the whole tree while only a single sub-tree of height $(h+1)/d$–1 needs to be computed per layer signing one message, where k is the times one key pair used in signature.

To sign a message, authentication path inner a sub-tree follows *down-top* approach, and the root node of the sub-tree is calculated from the bottom node to the top; while authentications between sub-trees follows *top-down* approach, the leaf nodes of the higher sub-tree is used to authenticate the root of the sub-tree one layer lower. Concretely, each sub-tree is constructed independently; the leaf nodes of each sub-tree are the public keys, nodes of the subtree above the lowest layer are calculated by the tree hash on input the two children nodes. In this way, we have established all the nodes of the whole tree. The root of the top sub-tree is the overall public key; it is based on its integrity that the leaf nodes of sub-tree on the higher layer can be used to authenticate the signature of the root of the sub-tree one layer below consecutively. Key state management chooses the signature key to sign the message directly, whose related public key is one leaf node of the bottom sub-tree. The signature can be presented as follows:

$$\sigma = \left(i,\ \sigma_m,\ Auth_{d-1}, \sigma_{d-2}, Auth_{d-2}, \ldots,\ \sigma_0, Auth_0\right)$$

The following data related to the signature σ have to be applied to verifier, including the message m, the OTS/FTS σ_m and its index i, along with the following data per layer containing signature σ_j of the root of the sub-tree on layer j generated using a signature key of the sub-tree on layer j–1, and the authentication path $Auth_j$ used to construct the root of sub-tree on layer j from the bottom up to verify or construct the one-time/few-time public key on the bottom of this sub-tree traversed by the whole authentication path, where $j=1,\ldots d$–1.

What need to be mentioned here is, in many cases, the one-time/few-time public key traversed by the authentication path on the bottom of each sub-tree can be deduced by the corresponding one-time/FTS of the root of the sub-tree below, as a consequence, it need not to be contained in the signature σ, we just use each OTS/FTS to construct the corresponding public key along the authentication path until the root of the sub-tree on the top layer, and compare it with the overall public key. The signature is valid if the comparison returns equal; otherwise, it is rejected. If in some exceptional cases the interior one-time/few-time public keys mentioned above cannot be deduced in this way, we still need to contain them in the signature σ, which turns to be:

$\sigma = (i, \sigma_m, Auth_{d-1}, Pk_{d-1}, \sigma_{d-2}, Pk_{d-2}, Auth_{d-2}, ..., \sigma_0, Pk_0, Auth_0)$,
where Pk_i is the interior public keys per layer, $i = 1, \ldots, d$.

The verification can be done similarly as the above schemes.

A. *GMSS and XMSS^{MT}*

Structural comparison of GMSS and XMSS^{MT} is shown in Table 3.

(a) *GMSS*

GMSS is a variant of the Merkle signature presented in 2007 (Buchmann et al. 2007), which allows signing cryptographically unlimited number of messages by using hyper-tree. It can be seen as an improved version of CMSS proposed in 2006 which just consists of two layers of trees (Buchmann et al. 2006). To sign a message, GMSS uses a hash function to generate its digest, Winternitz OTS as its OTS scheme to sign the message digest and the root of each single sub-tree expect the top one, Merkle tree to build the interior authentication path in a single tree. For a sub-tree, the leaf nodes of each Merkle tree are the hash values of concatenation of public keys from the Winternitz OTS one-time signature scheme, whereas the leaf nodes of each XMSS-tree are the root nodes of each L-tree.

Similar to the construction of XMSS, a *PRF F* is used in GMSS to iteratively generate the state and key pairs in each Winternitz OTS started from a random initial state *SEED*. Precisely, GMSS uses the state $State_{i-1}$ of the i–1th Winternitz OTS as the index to choose a fixed function $f_{State_{i-1}}$ from the *PRF*, then uses $f_{State_{i-1}}$ to produce a new state $State_i$ and a new $SEED_{OTS_i}$ of the ith Winternitz OTS, and at last uses $SEED_{OTS_i}$ to generate the one-time private key sequence sk_i of the ith Winternitz OTS. The public key pk_i are computed in the same way as that of WOTS on input the corresponding sk_i. It has been mentioned above that, the one-time key pair generation in the above way has already been proven to be forward-secure PRG if the underlying F is a secure *PRF*.

Table 3 Structural comparison of GMSS and XMSS^{MT}

Scheme	Component	
	Underlying single tree	Signatures of the root of each single tree
GMSS	*WOTS + Merkle tree*	*WOTS*
XMSS^{MT}	*XMSS*	*WOTS(W-OTS + in RFC8391)*

Remark

1. Sub-tree in different layers may have different heights.
2. Different Sub-tree may use different Winternitz parameters.

- *Security Comments*

No exact security reduction has been given in Buchmann et al. (2007). As GMSS uses hash function to generate message digest and the leaf nodes of each Merkle tree, security obviously cannot be better than the collision resistance of the underlying hash function. The argument is similar to the above schemes and it is straight forward to obtain the following conclusion that GMSS achieves EUCMA if F is a PRF, the underlying message digest function is collision resistant, and the hash tree function is one-way. The proof is omitted for the sake of brevity.

Furthermore, the GMSS key generation approach fits the construction to generate the forward-secure signature proposed in Cooper et al. (2019). In consequence, we can extend the security of GMSS as follows: if GMSS is an unforgeable signature scheme and the function F used to generate the one-time keys is a secure PRF, then the GMSS is an unforgeable forward-secure signature scheme.

(b) *XMSS^{MT}*

i. *Construction*

- *Basic Construction*

XMSS^{MT}, an abbreviation for XMSS with Multi-Tree, is the hyper-tree vision of XMSS which allows signing cryptographically unlimited number of messages purposed in Bernstein et al. (2015). It uses XMSS to build the interior authentication path in a sub-tree, and Winternitz OTS to sign the root of the sub-tree by the signature key corresponding to the leaf node on the one layer higher.

- *Using BDS Algorithm*

Particularly, $\mathrm{XMSS}^{\mathrm{MT}}$ applies the BDS algorithm to make a trade-off between the signature generation time and the storage cost, which can decrease runtime in the worst case by incomplete calculation of the authentication path at the cost of some additional storage. By doing this, the signature generation time in the worst case reduces from $2^{h} - 1$ to $(h - k)/2$ computations of leaf and TreeHash, where h denotes the tree height and k is the BDS parameter.

- *Optimal Parameters Selection*

Moreover, optimal parameters for $\mathrm{XMSS}^{\mathrm{MT}}$, including the BDS parameters, Winternitz parameters, the number of layers, the tree height and the security level, have been provided by linear optimization, to meet different application requirements towards the key size, key generation time, signature time, signature size, etc. For more details about parameter selection see (Hülsing et al. 2013).

ii. Security

The security of $\mathrm{XMSS}^{\mathrm{MT}}$ can be deduced from that of Winternitz OTS and XMSS directly. we omit it for concision.

B. HSS
(a) *Construction*

HSS, an abbreviation for Hierarchical Signature System, is the hyper-tree version of LMS which allows signing cryptographically unlimited number of messages. In its construction, it uses LMS both to build the interior authentication path in a sub-tree, and the signature between the sub-tree. For more details about parameter selection see (McGrew et al. 2019).

(b)*Security*

The EUCMA of HSS is based on the EUCMA of LMS, i.e., if the adversary can forgery a valid signature for HSS with negligible probability, implies he can also forgery a valid signature for the LMS with negligible probability. The iteration functions of OTS and hash tree both need to be second preimage resistant as in the LMS.

(3) Unlimited Number and Stateless

Stateless hash-based signature schemes don't need key state management to update the signature key any more, i.e., the signature key is chosen pseudorandomly when signing a message. In the consequence, it collides with the probability of $k^{-1/2}$ based on the birthday attack if the OTS schemes are used to sign messages, where k is the signature keys space, i.e., p^2 signature keys are needed if signing p messages. Therefore, the scale of the whole tree is much larger than that of the same number of messages to be signed in the stateful schemes. Several key issues should be concerned in the design unlimited number and stateless hash-based signature schemes.

Firstly, an efficient algorithm should be used to select the key from the space of all the signature keys randomly or pseudorandomly instead of selecting sequentially by the key state management; in practice, most stateless hash-based signature schemes take the message to be signed as one input of the PRG algorithm to select the key pseudorandomly.

Secondly, hyper-tree is still the most common approach to construct unlimited number and stateless schemes. A whole tree in a much larger scale needs to be constructed to avoid the key collision in which one key is used twice to sign two messages. As we analyzed in previous section, by using the hyper-tree in hash-based signatures, only one sub-tree needs to be computed on each layer, the key generation time reduces exponentially just at the cost of linear increase of the number of signatures between subtrees. Therefore, hyper-tree is still an extraordinary applicable way in designing unlimited number and stateless schemes, which is similar to the unlimited number and stateful schemes.

Finally, the specific implementation should be efficient enough to make this scheme practical. Several approaches could be used to reduce the storage and computation cost. For instance, at the bottom of the whole tree, FTS schemes are used to sign the messages instead of signing them by OTSs; in terms of generating the subtree in each layer, PRG is used to generate its leaf nodes efficiently with a short seed, instead of producing them randomly, etc.

Similarly, the signature consists of the index of the private key being used, FTS of the message, as well as the signatures and corresponding authentication path of the root public key on each layer.

A. *SPHINCS*

SPHINCS schemes purposed in 2015 is the first practical stateless hash-based signature with practical performance (Bernstein et al. 2015); moreover, its security is given in the standard model instead of the random oracle.

(a) *Idea*

SPHINCS can be viewed as the combination of pseudorandom signature key chosen algorithm, Horst FTS, and XMSS^{MT}. The whole structure can be viewed as a hyper-tree of height h with d layers of sub-tree of height $(h+1)/d$-1, PRG and PRF are used to generate the private keys and select them pseudorandomly to sign the message. XMSS^{MT} is used to sign the root nodes of the sub-tree, Horst is used on the bottom layer to sign the message directly.

(b) *Construction*

In this section, we introduce the construction of SPHINCS by explaining how it solves the several key issues in the stateless hash-based signature.

Firstly, to solve the matter of selecting the signature key (pseudo-)randomly, SPHINCS uses a hash function H: $\{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ and a PRF F:$\{0, 1\}^n \times \{0, 1\}^* \rightarrow \{0, 1\}^{2n}$ to generate the index IDX of Horst signature key on input the message m as follows.

Choose two private key sk_{rand}, sk_{SEED} uniformly at randomly from $\{0, 1\}^{\text{n}}$;

Calculate

$$Rand = F(sk_{rand}, m);$$
$$M_{dgt} = H_{dgt}\left(prefix_{1,n}(Rand), m\right);$$
$$IDX = prefix_{n+1,n+h}(Rand).$$

where $prefix_{a,\ b}$ (x) denotes the bit sub-string of x from the ath bit to bth bit counted from left to right Starting with 1; IDX and $prefix_{1,\ n}$ $(Rand)$ are parts of the final signature.

Secondly, SPHINCS uses $XMSS^{MT}$ as its hyper-tree scheme, i.e., the root of each sub-tree is signed by the W-OTS+key pairs of the trees one layer higher, each sub-tree is constructed using *XMSS* excepting the sub-tree on the bottom layer whose leaf nodes used to authenticate the signature of message are generated by Horst FTS.

Finally, two main approaches are used in SPHINCS to achieve more efficient and practical. On one hand, the FTS scheme Hors is used instead of OTSs to sign the messages. Moreover, in order to reduce the public key size of Hors and the combined signature size, an improved scheme Horst, which is the abbreviation for Hors with tree, is used in SPHINCS for further compressing public key. Concretely, a single primary tree is used to compress t components in one Horst public key to one Horst root. As a result, a Horst signature consists of k signature keys and the related authentication paths, one for each. Horst reduces the public key size by increasing authentication paths in the signature. Particularly, an optimization technique is used in generating Horst authentication path to make the combined size of signature and public key minimum. On the other hand, SPHINCS uses an address algorithm, to allocate a unique address *ADRS* for each node in the whole tree which is set to different values to indicate OTS address, Horst address, L-tree address, and sub-tree address respectively. Then the PRF F is used to generate the seed of private key on input of the *ADRS* address in Hors or W-OTS+and private key sk_{SEED}, then PRG G will be used to generate the Horst or W-OTS+private key sequence $K_{Horst\ or\ W\text{-}OTS+,1}$, $K_{Horst\ or\ W\text{-}OTS+,2}$, …, $K_{Horst\ or\ W\text{-}OTS+,\ t}$ where n is polynomial in security parameter, i.e.,

$$\begin{aligned} Seed_{Horst\ or\ W-OTS+} &= F(ADRS, sk_{SEED}); \\ & G(Seed_{Horst\ or\ OTS}) \\ & \quad = \big(K_{Horst\ or\ W-OTS+,1}, \\ & \quad K_{Horst\ or\ W-OTS+,2}, \ldots, \\ & \quad K_{Horst\ or\ W-OTS+,t}\big) \end{aligned}$$

(c) *Security*

SPHINCS is EUCMA under the following condition: the iteration function used in W-OTS+is second preimage resistant and undetectable one-way, the hash function used to generate authentication path is second preimage resistant, PRG and PRF used are secure pseudorandom and the underlying function to generate the Horst signature is subset-resilient. Furthermore, SPHINCS-256 achieves 128-bit security against quantum attack.

(d) *Performance*

SPHINCS-256 instantiated with BLAKE-256, BLAKE-512, CHACHA etc., generating a signature of a short message takes around 51 M cycles on a single core while hashing throughput is 1.6 cycles per byte, and can generate 200 signatures per second on a 4-core 3.5 GHz Intel CPU, with size of 41 KB, public keys size of 1 KB and private key size of 1 KB, which makes it practical in application.

B. SPHINCS+

SPHINCS+is an improved version of SHINCS and purposed in 2017 (Aumasson and Endignoux 2017, 2018; Bernstein et al. 2017, 2019), the following improvements have been made compared with SHINCS: firstly, Hors is replaced by Fors, which has been introduced in "Typical schemes" section. Fors is designed in such a way that k elements are selected from k sets instead one, to solve the weak message problem in which the same indices of the message digests yield the same elements in the signature; secondly, tweakable hash is utilized to achieve multi-fucntion multi-target attack resistant, different bitmask and function key are applied to each evaluation of the underlying hash function; finally, all the components in one W-OTS + public key are compressed by one call of tweakable hash instead of L-tree. Though these improvements, SPHINCS + reduces the signature size dramatically, and becomes one of the eight candidate algorithms in the third round PQC standardization process.

Alike SPHINCS, SPHINCS + can be considered as the combination of pseudorandom signature key chosen algorithm, XMSS^{MT} of the version in RFC8391, along with Fors FTS. Similarly, the whole structure also can be viewed as a hyper-tree with d layers of sub-tree, XMSS^{MT} is used to sign the immediate nodes in the hyper-tree, excepting that Fors is used on the bottom layer to sign the messages directly. PRG and PRF are used to generate the private keys and select them pseudorandomly to sign the message. Specially, tweakable hash functions mentioned in "Typical schemes" section are used in many components of SPHINCS +, such as the iteration functions, the public keys compression, etc., to make each call of the hash function is independent with each other, so as to prevent the multi-function multi-target attack.

We also introduce the construction of SPHINCS + by explaining how it solves the key issues in the stateless hash-based signature; moreover, the main improvement of SPHINCS + compared with SPHINCS will be included in this section as well.

Firstly, in order to achieve stateless, SPHINCS + uses a hash function H and a PRF F to generate the message digest and the index Idx of the Fors signature key on input the message m and a pair of public parameters *PK.seed* and *PK.root* which separately denote the public seed and overall root:

Choose private key sk_{rand} (pseudo-)randomly;

Calculate Rand = F (sk_{rand}, R, m) and make it public;

where R is set to zero by default, and also can be generated (pseudo-)randomly.

$$Digest||Idx = H(Rand,\ PK.seed,\ PK.root,\ m).$$

Compared with the index selection algorithm of Horst in SPHINCS, which uses private key as part of input and cannot be verified, the adversary can attack SPHINCS by evaluating index randomly, computing the related message digest, and then checking if the private keys of Horst used to sign this message have already been revealed in previous signatures; and if yes, the adversary forges the signature successfully. It will not happen in SPHINCS + as all the input parameters used to generate the index are public and can be verified. Therefore, it is not needed to contain the index in signature of SPHINCS + anymore.

Secondly, for the hyper-tree, SPHINCS + also uses XMSS^{MT} as its hyper-tree scheme, excepting the sub-tree on the bottom layer are generated using Fors FTS.

Finally, several approaches are adopted by SPHINCS + to reduce the signature size, and to make the whole scheme more efficient and practical.

On the first place, the Fors is used in SPHINCS + instead of Hors. For Hors which is the most principal component of Horst, its underlying subset-resilience problem has been proven to be insecure against adaptive attacks; furthermore, Hors pseudorandomly selects k elements from the set of t elements, and the distribution of the selected k elements should be as uniform as possible, but in fact, some weak messages will map the digest into a smaller space, which makes it easier to be attacked. Fors, the abbreviation of forest of random subsets, selects k elements from t sets rather than one, and compress each set using a subtree; finally, uses a tweakle hash function to compress the k toot nodes to construct an overall public key for one Fors instance.

Table 4 Required security required security of underlying function in hash tree and pseudorandom key selection

Scheme	Merkle tree	XMSS	XMSS in RFC 8391	LMS	GMSS	XMSS^{MT}	HSS	SPHINCS	SPHINCS +
Required security of underlying function in hash tree and pseudorandom key selection	CR	SPR	PRF of $G(n)$; multi-function multi-target SPR of H	SPR	CR	SPR	SPR	PRF of F; Subset-resilience of H_{dgt}; SPR of H	PQ-MM-SPR of H and *tweakable hash;* PQ-PRF of F; PQ-target subset resilience of H_{dgt}
Model	Standard	Standard	RO	RO	Standard	Standard	RO	Standard	RO

Security comparison

In this section, we analyze and compare the security assumptions of underlying function of hash tree and reduction model required in these specific hash-based signature schemes, the relevant conclusions are shown in Table 4.

For all the hash-based signature schemes mentioned above, we do not consider the security assumptions of underlying function required in key generation. In terms of stateful signature schemes, we only discuss the security assumptions of underlying functions used in constructing hash tree. In terms of the stateless signature scheme, we consider not only that in constructing hash tree, but security assumptions of the underlying function related to pseudorandom key selection. H in the table above denotes the underlying function used to generate parent node in hash tree, other representations are defined in the specific schemes. Among the above conclusions, CR, SPR, PQ, MM, RO denote collision resistant, second preimage resistant, post-quantum, multi-function and multi-target, random oracle, separately. The security of CR, SPR, PRF decreases in turn, MM-SPR has the same security level with standard SPR against classical and quantum attacks.

In practice, we expect to design the hash-based signature scheme with the security assumption of underling function as weak as possible. Only consider the required security of underlying function in hash tree, under the same security level, the output length of the above SPR-based signature algorithm is halved compared with CR-based one. In addition, the proof given in standard model is more convincing than that of RO.

Conclusion

In this paper, firstly, we discuss the research progress in the component hash-based signature, i.e., OTS and FTS, then classify the tree-based public key authentication schemes of hash-based signature into limited number and stateful schemes, unlimited number and stateful schemes and unlimited number and stateless schemes, to analyze the overall design idea of different categories of hash-based signatures, as well as the construction, security reduction and performance of specific schemes. Due to the better performance of stateful hash-based schemes, they are more widely accepted and become standards in practice, such as the XMSS and XMSS^{MT} are specified by IETF in RFC8391, whereas LMS in RFC8554. However, in the standardization of post standard quantum cryptography algorithms, stateless hash-based signature, which is more in line with the core of digital signature primitive, has attracted more interest in both research and application. The research of the stateful and stateless hash-based signature parallel needs to be carried out parallelly and cannot be replaced with each other. The specification of stateful schemes is more conducive to its application in industry; however, in scenarios that do not support key management, stateless schemes is the necessary choice.

Acknowledgements
Not applicable.

Authors' contributions
The first author completed the main work of the paper and drafted the manuscript. The second and third authors participated in problem discussions and improvements of the manuscript. All authors read and approved the final manuscript.

Funding
This work is supported by National Natural Science Foundation of China (No. 61972391).

Availability of data and materials
Not applicable.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]State Key Laboratory of Information Security, Institute of Information Engineering, CAS, No. 89 Minzhuang Road, Haidian District, Beijing 100093, China. [2]School of Cyber Security, University of Chinese Academy of Sciences, No. 19 Yuquan Road, Shijingshan District, Beijing 100049, China. [3]School of Computer Science, Liaocheng University, No. 1, Hunan Road, Dongchangfu District, Liaocheng City 252000, China.

Received: 19 November 2021 Accepted: 15 February 2022
Published online: 01 July 2022

References

Aumasson JP, Endignoux G (2017) Gravity SPHINCS, A submission to the NIST standardization project on post-quantum cryptography. https://github.com/gravity-postquantum/gravity-sphincs. Accessed 09 Jan 2018

Aumasson JP, Endignoux G (2018) Improving stateless hash-based signatures. In: Cryptographers' track at the RSA conference. Springer, Cham, pp 219–242

Bellare M, Miner SK (1999) A forward-secure digital signature scheme. In: Annual international cryptology conference. Springer, Berlin, pp 431–448

Bennett CH (1992) Quantum cryptography using any two nonorthogonal states. Phys Rev Lett 68(21):3121

Bennett CH, Bessette F, Brassard G et al (1992) Experimental quantum cryptography. J Cryptol 5(1):3–28

Bergadano F, Cavagnino D, Crispo B (2002) Individual authentication in multiparty communications. Comput Secur 21(8):719–735

Berman P, Karpinski M, Nekrich Y (2007) Optimal trade-off for Merkle tree traversal. Theoret Comput Sci 372(1):26–36

Bernstein DJ (2009) Introduction to post-quantum cryptography//Post-quantum cryptography. Springer, Berlin, pp 1–14

Bernstein DJ, Hopwood D, Hülsing A et al (2015) Sphincs: practical stateless hash-based signatures. In: Annual international conference on the theory and applications of cryptographic techniques. Springer, pp 368–397

Bernstein DJ, Dobraunig C, Eichlseder M et al (2017) SPHINCS+, A submission to the NIST standardization project on post-quantum cryptography. https://sphincs.org/. Accessed 09 Jan 2018

Bernstein DJ, Hülsing A, Kölbl S et al (2019) The SPHINCS+ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security, pp 2129–2146

Black J, Rogaway P, Shrimpton T (2002) Black-box analysis of the block-cipher-based hash-function constructions from PGV. In: Advances in cryptology—CRYPTO 2002, California, USA

Bleichenbacher D, Maurer UM (1994) Directed acyclic graphs, one-way functions and digital signatures. In: Advances in cryptology—CRYPTO '94, volume 839 of lecture notes in computer science, pp 75–82
Brassard G, Lütkenhaus N, Mor T et al (2000) Limitations on practical quantum cryptography. Phys Rev Lett 85(6):1330
Buchmann J, García LCC, Dahmen E et al (2006) CMSS—an improved Merkle signature scheme. In: International conference on cryptology in India. Springer, pp 349–363
Buchmann J, Dahmen E, Klintsevich E et al (2007) Merkle signatures with virtually unlimited signature capacity. In: International conference on applied cryptography and network security. Springer, Berlin, pp 31–45
Buchmann J, Dahmen E, Schneider M (2008) Merkle tree traversal revisited. In: International workshop on post-quantum cryptography. Springer, Berlin, pp 63–78
Buchmann J, Dahmen E, Ereth S et al (2011a) On the security of the Winternitz one-time signature scheme. In: International conference on cryptology in Africa. Springer, Berlin, pp 363–378
Buchmann J, Dahmen E, Hülsing A (2011b) Xmss-a practical forward secure signature scheme based on minimal security assumptions. In: International workshop on post-quantum cryptography. Springer, pp 117–129
Buldas A, Laanoja R, Truu A (2017) A server-assisted hash-based signature scheme. In: Nordic conference on secure IT systems. Springer, Cham, pp 3–17
Buldas A, Laanoja R, Truu A (2018) A blockchain-assisted hash-based signature scheme. In: Nordic conference on secure IT systems. Springer, Cham, pp. 138–153
Cooper D, Apon D, Dang Q, et al (2019) Recommendation for stateful hash-based signature schemes. draft NIST Special Publication 800–208. NIST. SP, pp 800–208
Coppersmith D, Jakobsson M (2002) Almost optimal hash sequence traversal. In: International conference on financial cryptography. Springer, Berlin, pp 102–119
Coron JS, Dodis Y, Malinaud C et al (2005) Merkle-Damgård revisited: how to construct a hash function. In: Annual international cryptology conference. Springer, Berlin, pp 430–448
Damgård I (1989) A design principle for hash functions. In: Crypto '89. LNCS No. 435, pp 416–427
Dods C, Smart NP, Stam M (2005) Hash based digital signature schemes. In: IMA international conference on cryptography and coding. Springer, Berlin, pp 96–115
Eaton E (2017) Leighton-Micali hash-based signatures in the quantum random-oracle model. In: International conference on selected areas in cryptography. Springer, Cham, pp 263–280
Ekert AK (1991) Quantum cryptography based on Bell's theorem. Phys Rev Lett 67(6):661
ElGamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31(4):469–472
Even S, Goldreich O, Micali S (1996) On-line/off-line digital signatures. In: 1996 International association for cryptologic research
Gisin N, Ribordy G, Tittel W et al (2002) Quantum cryptography. Rev Mod Phys 74(1):145
Goldwasser S, Micali S, Rivest RL (1988) A digital signature scheme secure against adaptive chosen-message attacks. SIAM J Comput 17(2):281–308
Gröblacher S, Jennewein T, Vaziri A et al (2006) Experimental quantum cryptography with qutrits. New J Phys 8(5):75
Halevi S, Krawczyk H (2006) Strengthening digital signatures via randomized hashing. In: CRYPTO 06
Hülsing A (2013) W-OTS+–shorter signatures for hash-based signature schemes. In: International conference on cryptology in Africa. Springer, Berlin, pp 173–188
Hülsing A, Rausch L, Buchmann J (2013) Optimal parameters for XMSS^{MT}. In: International conference on availability, reliability, and security. Springer, pp 194–208
Hülsing A, Rijneveld J, Song F (2016) Mitigating multi-target attacks in hash-based signatures. In: Public-key cryptography–PKC 2016. Springer, Berlin, pp 387–416
Hülsing A, Butin D, Gazdag S, Rijneveld J, Mohaisen A (2018) RFC8391-XMSS: eXtended hash-based signatures. RFC 8391, RFC Editor
Jakobsson M (2002) Fractal hash sequence representation and traversal. In: Proceedings IEEE international symposium on information theory. IEEE, p 437
Jakobsson M, Leighton T, Micali S et al (2003) Fractal Merkle tree representation and traversal. In: Cryptographers' track at the RSA conference. Springer, Berlin, pp 314–326
Kampanakis P, Fluhrer SR (2017) Lms vs xmss: a comparison of the stateful hash-based signature proposed standards. IACR Cryptology ePrint Archive
Katz J (2016) Analysis of a proposed hash-based signature standard. In: International conference on research in security standardisation. Springer, pp 261–273
Knecht M, Meier W, Nicola CU (2014) A space-and time-efficient Implementation of the Merkle Tree Traversal Algorithm. arXiv:1409.4081
Krawczyk H (2000) Simple forward-secure signatures from any signature scheme. In: Proceedings of the 7th ACM conference on computer and communications security, pp 108–115
Lamport L (1979) Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International Palo Alto
Leighton FT, Micali A (1995) Large provably fast and secure digital signature schemes based on secure hash functions. US Patent 5,432,852
McGrew D, Curcio M, Fluhrer S (2019) Leighton-Micali hash-based signatures. RFC 8554, IRTF
Menezes AJ, Van Oorschot PC, Vanstone SA (2018) Handbook of applied cryptography. CRC Press, Boca Raton
Merkle RC (1979a) Secrecy, authentication, and public key systems. Stanford University, Stanford
Merkle RC (1979b) Secrecy, "Authentication and public key systems". Ph.D. Thesis, Stanford
Merkle RC (1989) A certified digital signature. In: Conference on the theory and application of cryptology. Springer, New York, pp 218–238
Naor D, Shenhav A, Wool A (2005) One-time signatures revisited: Have they become practical? IACR Cryptol. ePrint Arch
Naor D, Shenhav A, Wool A (2006) One-time signatures revisited: practical fast signatures using fractal merkle tree traversal. In: 2006 IEEE 24th convention of electrical and electronics engineers in Israel. IEEE, pp 255–259
Perrig A (2001) The BiBa one-time signature and broadcast authentication protocol. In: Proceedings of the 8th ACM conference on computer and communications security, pp 28–37
Perrig A, Canetti R, Song D et al (2001) Efficient and secure source authentication for multicast. In: Network and distributed system security symposium, NDSS, vol 1, pp 35–46
Perrig A, Canetti R, Tygar JD et al (2002) The TESLA broadcast authentication protocol. RSA Cryptobytes 5(2):2–13
Preneel B, Govaerts R, Vandewalle J (1993) Hash functions based on block ciphers: a synthetic approach. In: Advances in cryptology—CRYPTO '93. Santa Barbara, California, USA
Reyzin L, Reyzin N (2002) Better than BiBa: short one-time signatures with fast signing and verifying. In: Australasian conference on information security and privacy. Springer, pp 144–153
Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126
Rogaway P, Shrimpton T (2004) Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: International workshop on fast software encryption. Springer, Berlin, pp 371–388
Sella Y (2003) On the computation-storage trade-offs of hash chain traversal. In: International conference on financial cryptography. Springer, Berlin, pp 270–285
Shor PW (1999) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev 41(2):303–332

Publisher's Note