计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20210619
58(10): 2300-2309, 2021

SOTS:一个基于哈希函数更短的后量子数字签名方案

卫宏儒　　黄靖怡
(北京科技大学数理学院　北京　100083)
(weihr@ustb.edu.cn)

SOTS: A Hash Function-Based Shorter Post-Quantum Digital Signature Scheme

Wei Hongru and Huang Jingyi
(*School of Mathematics and Physics*, *University of Science and Technology Beijing*, *Beijing* 100083)

Abstract　In the post-quantum digital signature schemes, the Hash-based signature schemes are efficient and provably secure. However, one major drawback of Hash-based signature schemes is the large size of the key and the signature. In this study, based on existing digital signature schemes, a new One-Time Signature (OTS) scheme, which reduces both the number of the signatures and the size of each signature, has been proposed. Under the same post-quantum security level, the proposed scheme reduces the key and the signature sizes by 77% and 82.0% respectively as compared with the Winternitz OTS scheme. And it also reduces the key and the signature sizes by 60.7% and 60.5% respectively as compared with WOTS+. In terms of the signature size, compared with the NOTS, SDS-OTS and WOTS-S schemes proposed in the past two years, this proposed novel scheme has reduced by 17%, 24.5% and 48.1% respectively. Furthermore, this novel scheme is existentially unforgeable under the Chosen-Plaintext Attack (CPA) model. The security of this scheme is a security reduction of the onewayness of the underlying Hash function. Moreover, compared with WOTS+, the proposed signature scheme reduces the time of generating keys, creating signatures and verifying signatures by 71.4%, 47.7%, and 60.9% respectively.

Key words　Hash function-based digital signature; one-time signature; post-quantum cryptography; information security; distributed ledger

摘　要　在后量子数字签名方案中,基于哈希函数的签名方案是高效和可证明安全的.然而,过长的密钥和签名是基于哈希函数的签名方案最主要的问题.在已有签名方案的基础上,提出一个新的一次签名方案,该方案不仅减少了签名的数量,同时减少了每个签名的长度.和 Winternitz OTS 方案相比,新的方案在密钥和签名尺寸上分别减少了 77%和 82%,和 WOTS+方案相比,在密钥和签名尺寸上分别减少了 60.7%和 60.5%.在签名长度上,新方案与近 2 年提出的 NOTS,SDS-OTS 和 WOTS-S 方案相比,分别减少了 17%,24.5%和 48.1%.另外,证明了新的方案在选择明文攻击(Chosen-Plaintext Attack, CPA)下是存在不可伪造的,安全性可规约为底层哈希函数的单向性.除此之外,实验证实了与 WOTS+方案相比,在密钥生成、签名生成和签名验证所需时间上,新的方案分别减少了 71.4%,47.7%和 60.9%.

关键词　基于哈希函数的数字签名方案;一次签名;后量子密码学;信息安全;分布式账本

中图法分类号　TP309

收稿日期:2021-06-10;**修回日期**:2021-07-30

基金项目:国家自然科学基金项目(61873026);广东省重点领域研发计划项目(2020B0909020001)
This work was supported by the National Natural Science Foundation of China (61873026) and the Key-Area Research and Development Program of Guangdong Province (2020B0909020001).

随着量子计算机的高速发展,Shor 与 Grover 算法的出现,基于离散对数问题(discrete logarithm problem, DLP)、大整数分解问题(inter factorization problem, IFP)和椭圆曲线离散对数问题(elliptic curve discrete logarithm problem, ECDLP)的传统公钥签名算法将不安全.因此,可以抵抗量子计算机的后量子签名方案备受关注.基于哈希的签名方案是高效和可证明安全的.除此之外,与其他的基于格、基于编码、基于多变量的后量子签名方案相比,基于哈希函数的签名方案执行时间最少.因此,基于哈希函数的签名方案备受关注.然而,过大的密钥和签名尺寸限制了它在分布式账本和加密货币中的应用[1].

一个基于哈希函数的签名方案是由一个一次签名(one-time-signature, OTS)或一个多次签名(few-time-signature, FTS)方案,结合一个用于压缩和管理公钥的哈希树构成.其中,用于生成密钥和签名的 OTS 或 FTS 方案是基于哈希签名方案的核心.最初的 OTS 方案是由 Lamport 提出的 LD-OTS 方案.该方案逐位对消息进行签名,并对不同的待签名消息换用不同的密钥对.因此,对于一个长度为 m 位的消息进行签名,就需要 m 个签名和 $2m$ 个密钥,如果一个签名长度为 n 位,则生成的签名和密钥长度分别为 mn 和 $2mn$ 位,会消耗大量空间.WOTS 方案[2]和它的变体方案 WOTS^{PRF} 和 WOTS+[3]都是在 LD-OTS 上进行了改进,将逐位签名替换为分组签名.其中 WOTS^{PRF} 方案引入伪随机函数来替代抗碰撞(CR)哈希函数.WOTS+方案引入位掩码,将 CR 哈希函数替换为不可测单向函数(可以为一个哈希消息身份认证码,也可以为一个分组密码)来减少签名的数量和尺寸.但这些方案的签名和密钥的尺寸对于区块链等分布式账本来说还是过大.除此之外,运行效率低是 WOTS+的问题.经过评估,WOTS+的密钥生成、签名生成和签名验证所需时间远高于 WOTS.使用位掩码和随机操作,用不可测单向函数替代 CR 哈希函数比 CR 函数更加耗时.因此,使用 CR 哈希函数,通过其他方式降低空间的占用更佳.

椭圆曲线数字签名方案 ECDSA 是分布式账本最常用的数字签名方案.然而,ECDSA 在后量子时代不是一个安全的签名方案.近几年,如 IOTA,QRL,PQChain[4]和 DL-for-IOT[5]等后量子分布式账本已经采用了基于哈希函数的数字签名方案.在这些分布式账本中应用的 OTS 方案主要是 WOTS,WOTS+和 WOTS^{PRF}.因此,这些后量子分布式账本的不足之处包括了哈希签名方案密钥和签名尺寸过大的问题.

在近期文献[6-7]中提出的 NOTS 和 SDS 方案在 WOTS 的基础上,将对消息的签名转化为对 16 进制表示字符的签名,缩小了签名的个数.另一个新的方案 WOTS-S[8]引入在哈希迭代中取子串的操作,减少了每个签名的长度.在这些 WOTS 变体方案的基础上,本文提出一个新的一次签名方案 SOTS,在减少了密钥和签名长度的同时也提高了运行效率.

本文的主要贡献包括 3 个方面:

1) 提出了一种基于哈希函数的一次签名方案 SOTS,通过对 16 进制字符进行签名和引入迭代取子串的过程,在减少签名个数的同时也缩短了每个签名的长度.相比于 WOTS,SOTS 在密钥和签名尺寸上分别缩小了 77%和 82%,相比于 WOTS+,在密钥和签名尺寸上分别缩小了 60.7%和 60.5%;

2) 证明了在 CPA 模型和伪造者在一定时间内只能询问一条消息的情况下,SOTS 方案是存在不可伪造的,它的安全性可规约为底层的单向哈希函数的安全性;

3) 通过实验,评估了新的签名方案,结果表明 SOTS 是一个高效的方案.与 WOTS+相比,在密钥生成、签名生成和验证的时间上分别减少了 71.4%,47.4%和 60.9%.相比于 WOTS-S,SOTS 在运行效率上有明显提升.

1 基础知识

本节将介绍常用哈希函数的安全性以及攻击模型,这有利于后文对提出的签名方案的安全性分析.本文主要用到的符号及其说明在表 1 中给出.

Table 1 Symbols and Descriptions

表 1 符号说明

符号	说明
M	待签名消息
$checksum$	检查数
$seed$	一个长度为 n 位的随机种子
n	安全参数
H	M 的哈希值
H_hex	H 的 16 进制表示
$sk/pk/vk$	私钥/公钥/验证密钥

续表 1

符号	说明
$count\_sb[i]$	每个 16 进制字符在 H_hex 中的个数
$sum\_index[i]$	每个字母在 H_hex 中索引地址的和
$sub\_iteration[i]$	每个字母从 $count_sb[i]$中提取的特征
$iteration[i]$	每个字母从 $sum_index[i]$中提取出的特征
σ	M 生成的签名
sk_size/pk_size	私钥/公钥的大小/b
σ_size	签名的大小/b
$sk[i]/pk[i]/vk[i]$	私钥/公钥/验证密钥中的元素
$fsk[i]/fpk[i]/fvk[i]$	私钥/公钥/验证密钥的第 1 部分
$bsk[i]/bpk[i]/bvk[i]$	私钥/公钥/验证密钥的第 2 部分
$f\sigma[i]/b\sigma[i]$	σ 的第 1/第 2 部分
ADV	试图破坏哈希函数的攻击者
O	一个签名预言机
FOR	一个试图破坏签名方案的伪造者
M^Q	FOR 询问的消息
σ^{M^Q}	O 返回的合理的 M^Q 的签名
$(M',\sigma^{M'})$	FOR 构造的合理的消息-签名对
h_{ow}	一个单向哈希函数
$ADV_{onewayness}$	一个试图破坏 h_{ow}的单向性的攻击者

1.1 哈希函数的安全性

底层哈希函数的安全性是基于哈希函数数字签名方案最基本的安全性要求.一个安全的哈希函数可以抵抗原像攻击、第二原像攻击和碰撞攻击:

$$Pr(y\leftarrow h(x);x'\leftarrow \mathrm{ADV}(y);x=x')\leqslant\varepsilon. \quad (1)$$

对于抗原像攻击,攻击者(ADV)不能或者以很小的概率能在已知像 y 的情况下推测出原像 x:

$$Pr(y\leftarrow h(x);x'\leftarrow \mathrm{ADV}(x,y):x\neq x'\cap y=h(x'))\leqslant\varepsilon. \quad (2)$$

根据一个输入输出对(x,y),攻击者很难找到另一个输入 x',使得输出也为 y,这就是抗第二原像性.通过

$$Pr(x,x'\leftarrow \mathrm{ADV}:x\neq x'\cap h(x)=h(x'))\leqslant\varepsilon \quad (3)$$

可知,抗碰撞攻击是指攻击者不能找到任意 2 个不同的输入 x 和 x',使得它们所成像相同.

一个哈希函数的安全水平是由它的输出长度 n 决定的.Grover 算法和 Shor 算法降低了哈希函数的量子安全水平.表 2 给出了常用哈希函数的经典和量子安全水平.可见,一个 n 位的哈希函数可以提供 n 位和 $n/2$ 位的经典安全水平来抵抗原像和碰撞攻击[9].然而,同样的哈希函数只能提供 $n/2$ 位和 $n/3$ 位的量子安全性来抵抗原像和碰撞攻击.

Table 2 Security of Hash Functions

表 2 哈希函数的安全性 b

哈希函数	输出长度	经典安全性		量子安全性	
		第二原像攻击/原像攻击	碰撞攻击	第二原像攻击/原像攻击	碰撞攻击
$SHA160$	160	160	80	80	53
$SHA256$	256	256	128	128	85
$SHA384$	384	384	192	192	128
$SHA512$	512	512	256	256	171

1.2 选择明文攻击(CPA)模型

现存的很多研究,例如 PQ-chain,compact PQ-chain 等利用 CPA 模型来证明提出的数字签名方案的安全性.

CPA 是一个攻击模型.在一定时间内,伪造者 F 选择消息,询问签名预言机 O 这些消息的签名,通过反馈得到的签名来构造一个合理的消息-签名$(M',\sigma^{M'})$.完整的过程如图 1 所示.一个伪造者发送选择的消息集合$(M^1,M^2,\cdots,M^n)$给 O,O 将对应的签名集合$(\sigma^1,\sigma^2,\cdots,\sigma^n)$反馈给 F.通过这些签名,F 可以尝试构造一个新的消息-签名对$(M',\sigma^{M'})$.如果该消息-签名对是合理的,即 $\sigma^{M'}$ 是合理的并且 M'没有在询问的消息集合中,则 F 成功.CPA-Secure-Model 指 F 成功的概率很小可以忽略.

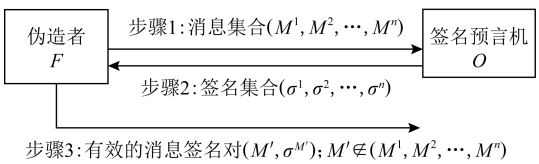


Fig.1 CPA model

图 1 CPA 模型示意图

2 更短的一次签名 SOTS

SOTS 方案结合并提升了 NOTS[5] 和 WOTS-S[7] 方案,不仅减少了签名的个数,还缩小了每个签名的尺寸.本节将从密钥生成、签名生成和签名验证 3 个方面来详细阐述提出的新方案.

2.1 密钥生成

通过哈希函数 SHA512,将随机生成的 64 B 的随机种子 $seed$ 哈希迭代 17 次,每次生成一个私钥元素 $sk[i]$.私钥集合 sk 是由 17 个长 512 b 的私钥元素构成.

每个公钥元素由对应的私钥元素生成.在生成公钥之前,本文将每个私钥元素 $sk[i]$均分为长 256 b 的

前后 2 部分,分别记为 $fsk[i]$和 $bsk[i]$.每个公钥元素的生成过程不同.详细的算法如算法 1 所示.

算法 1.密钥生成算法.

输入:安全参数 $n=512$;

输出:私钥集合 $sk[]$、公钥集合 $pk[]$.

① $seed \leftarrow os.urandom(64)$; /* 随机种子 */

② $x \leftarrow \mathrm{SHA512}(seed)$;

③ for $i=0$ to 16 do /* 生成私钥集合 */

④ $sk.append(x)$;

⑤ $x \leftarrow \mathrm{SHA512}(x)$;

⑥ end for

⑦ for $i=0$ to 15 do /* 生成前 16 个公钥 */

⑧ $fsk \leftarrow SHA256(sk[i][0:31])$;

⑨ $bsk \leftarrow SHA256(sk[i][32:63])$;

⑩ for $j=1$ to 15 do

⑪ $fsk \leftarrow SHA256(fsk)$;

⑫ $fsk \leftarrow fsk.Substrings(1, fsk.length - j*16)$; /* 取字串 */

⑬ end for

⑭ $fpk \leftarrow SHA256(fsk)$;

⑮ for $j=1$ to 128 do

⑯ $bsk \leftarrow SHA256(bsk)$;

⑰ end for

⑱ $bpk \leftarrow bsk$;

⑲ $pk[i] \leftarrow fpk \| bpk$;

⑳ end for

㉑ $fsk \leftarrow sk[16][0:31]$; /* 第 17 个 fsk */

㉒ $bsk \leftarrow sk[16][32:63]$; /* 第 17 个 bsk */

㉓ for $k=1$ to 1921 do /* 生成第 17 个公钥 */

㉔ $fsk \leftarrow SHA256(fsk)$;

㉕ $bsk \leftarrow SHA256(bsk)$;

㉖ end for

㉗ $pk[16]=fsk \| bsk$.

如算法 1 中行⑦~⑭所示,本文用 $SHA256$ 函数将前 16 个私钥元素 $fsk[i]$分别哈希迭代 17 次,其中,第 2 次到第 16 次迭代过程加入取子串的操作,即取上 1 个哈希结果的子串作为当前哈希运算的输入值,生成对应的前 16 个长 256 b 的公钥元素 $fpk[i]$.如算法 1 的行⑮~⑰所示,本文将前 16 个 $bsk[i]$元素分别哈希迭代 129 次,生成对应的前 16 个 $bpk[i]$.第 17 个 $fpk[16]$和 $bpk[16]$是通过 $fsk[16]$和 $bsk[16]$分别哈希迭代 1 921 次生成.每个公钥元素 $pk[i]$由对应的 $fpk[i]$和 $bpk[i]$合并生成:

$$pk[i]=fpk[i] \| bpk[i], i=0 \rightarrow 16.$$

公钥用于验证签名,因此公钥的设计要与签名的设计匹配.公钥元素生成所需的迭代次数也与签名过程有关.前 16 个签名元素所需哈希迭代次数的取值范围为[1,16]和[0,128],因此前 16 个 $fpk[i]$和 $bpk[i]$元素分别由相应的私钥元素哈希迭代 17 和 129 次生成.第 17 个公钥元素用于验证检查数的签名,检查数对字符的个数进行检查,取值范围为[0,1 920],因此 $fpk[16]$和 $bpk[16]$由相应的私钥通过哈希迭代 1 921 次生成.签名过程中决定哈希迭代次数函数的详细设计参见 2.2 节.

2.2 签名生成

在生成签名之前,本文先用 $SHA512$ 函数计算待签名 M 消息的哈希值,并用 16 进制表示为 H_hex.本文提取表示 16 进制字符('0'至'f')的每个字符在 H_hex 中的个数和所在的位置索引为特征.

本文用参数 $count_sb[i]$表示 16 个字符在 H_hex 中的个数,因此该参数的取值一定在[0,128]范围内.显然,如果每个字符出现次数均等,则该值都应为 8.参数 $sum_index[i]$记录每个字符所在的索引地址之和.

$$sub\_iteration[i]=\begin{cases} count\_sb[i]+1, \\ \quad count\_sb[i]\leqslant 15; \\ count\_sb[i]//8, \\ \quad count\_sb[i]>15. \end{cases} \tag{4}$$

$$iteration[i]=\begin{cases} sum\_index[i]//count\_sb[i], \\ \quad count\_sb[i]\neq 0; \\ 1,\ count\_sb[i]=0. \end{cases} \tag{5}$$

在这 2 个参数的基础之上,计算出对应的 $sub_iteration[i]$和 $iteration[i]$来决定签名过程中哈希的迭代次数.

本文计算相应的检查数 $checksum$ 为

$$checksum=\sum_{i=0}^{15}|count\_sb[i]-8|. \tag{6}$$

构造签名的过程如算法 2 所示.

算法 2.签名构造算法.

输入:待签名消息 M、私钥集合 $sk[]$;

输出:2 部分签名 $Signature\_one$, $Signature\_two$.

① $H\_hex \leftarrow hexlify(SHA512(M))$;

/* 十六进制 */

② $hex\_symbols \leftarrow$ '0123456789abcdef'

③ for $i=0$ to 15 do

/* 计算每个字母的个数和索引信息 */

④ $sum\_sb \leftarrow 0$;
⑤ $hsb \leftarrow hex\_symbols[i]$;
⑥ $index\_sb \leftarrow 0$;
⑦ for $j=0$ to 127 do
⑧ if $H\_\_hex[j]==hsb$ then
⑨ $sum\_sb \leftarrow sum\_sb+1$;
⑩ $index\_sb \leftarrow index\_sb+j$;
⑪ end if
⑫ end for
⑬ $count\_sb[i] \leftarrow sum\_sb$;
/* 计算每个字母个数 */
⑭ $sum\_index[i] \leftarrow index\_sb$;
/* 计算每个字母的索引地址 */
⑮ if $count\_sb[i] <= 15$ then
/* 计算每个 $sub\_iteration[i]$ */
⑯ $sub\_iteration[i] \leftarrow count\_sb[i]+1$;
⑰ else
⑱ $sub\_iteration[i] \leftarrow count\_sb[i]//8$;
⑲ end if
⑳ if $count\_sb[i]==0$ then
/* 计算每 $iteration[i]$ */
㉑ $iteration[i] \leftarrow 1$;
㉒ else
㉓ $iteration[i] \leftarrow sum\_index[i]//cont\_sb[i]$;
㉔ end if
㉕ end for
㉖ for $k=0$ to 15 do
㉗ $x \leftarrow SHA256(sk[k][0:31])$;
㉘ $y \leftarrow SHA256(sk[k][32:63])$;
㉙ for $j=1$ to $sub\_iteration[k]-1$ do
/* 计算前 16 个 $Signature\_one$ 中元素 */
㉚ $x \leftarrow SHA256(x)$;
㉛ $x \leftarrow x.Substrings(1, x.length-16*j)$;
㉜ end for
㉝ $f\sigma[k] \leftarrow x$;
㉞ $Signature\_one \leftarrow Signature\_one+x$;
㉟ for $j=1$ to $itertaion[k]-1$ do
/* 计算前 16 个 $Signature\_two$ 元素 */
㊱ $y \leftarrow SHA256(y)$;
㊲ end for
㊳ $b\sigma[k] \leftarrow y$;
㊴ $Signature\_two \leftarrow Signature\_two+y$;
㊵ end for
㊶ $checksum \leftarrow 0$;
㊷ for $j=0$ to 15 do /* 计算 $checksum$ */
㊸ $checksum \leftarrow checksum+|count\_sb[j]-8|$;
㊹ end for
㊺ $x \leftarrow sk[16][0:31]$;
㊻ $y \leftarrow sk[16][32:63]$;
㊼ for $j=1$ to $checksum$ do
/* 计算第 17 个签名元素 */
㊽ $x \leftarrow SHA256(x)$;
㊾ end for
㊿ $f\sigma[16] \leftarrow x$;
(51) $Signature\_one \leftarrow Signature\_one+f\sigma[16]$;
(52) for $j=1$ to $1920-checksum$ do
(53) $y \leftarrow SHA256(y)$;
(54) end for
(55) $b\sigma[16] \leftarrow y$;
(56) $Signature\_two \leftarrow Signature\_two+b\sigma[16]$.

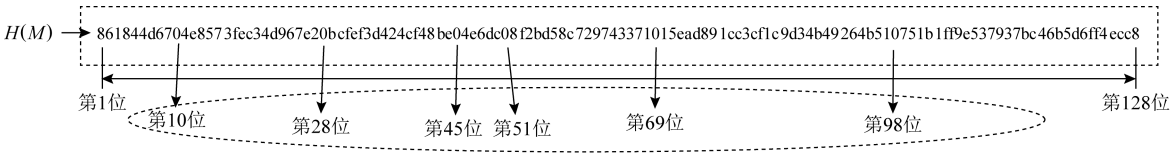
前 16 个签名元素和第 17 个签名元素生成不同.前 16 个 $f\sigma[i]$和 $b\sigma[i]$元素分别是由对应的 $fsk[i]$和 $bsk[i]$元素通过哈希迭代 $sub_iteration[i]$和 $iteration[i]$次生成.但是这 2 种迭代过程不同.$f\sigma[i]$的生成过程与 $fpk[i]$类似,都加入了迭代取子串的操作.其中,第 2 次到最后一次的哈希过程中,都进行了取子串操作.可见,因为 $sub_iteration[i]$值的不同,生成的 $f\sigma[i]$不是定长的 256 b,而是变长的.每个 $f\sigma[i]$的长度为

$$f\sigma[i].length=256-(sub\_iteration[i]-1)\times 16,(i=0\rightarrow 15). \quad (7)$$

第 17 个签名元素 $f\sigma[16]$和 $b\sigma[16]$是通过 $fsk[16]$和 $bsk[16]$分别哈希迭代 $checksum$ 和 $1920-checksum$ 次生成.生成的 17 个 $f\sigma[i]$元素合并生成 $Signature\_one$,17 个 $b\sigma[i]$合并生成 $Signature\_two$.图 2 展示了消息“Hello World!”的签名构造过程.其中函数 $f_{\mathrm{H}}^{i}()$表示从第 2 次到第 i 次的哈希运算加入了取子串操作.算法 2 展示了详细的签名构造过程.

2.3 签名验证

与 WOTS 方案相同,验证密钥的生成只需要消息 M 和签名的信息即可.通过已知 M 的信息,验证者按照签名构造过程相同的方式计算相应的 $sub\_iteration[i]$,$iteration[i]$和 $checksum$ 的值.接着,将 $f\sigma[i]$和 $b\sigma[i]$元素从签名信息 $Signature\_one$ 和 $Signature\_two$ 中提取出来,具体的提取过程如算法 3 中行④~⑧所示.



(a) 'Hello World!' 对应哈希值的十六进制表示

symbol(*sb*)	*count_sb*[*i*]	*sub_iteration*[*i*]	*sum_index*[*i*]	*iteration*[*i*]	*fσ*[*i*]	*bσ*[*i*]
0	6	7	301	50	$f_H'^{7}(sk[0][0:31])$	$f_H^{50}(sk[0][32:63])$
1	8	9	602	75	$f_H'^{9}(sk[1][0:31])$	$f_H^{75}(sk[1][32:63])$
2	5	6	270	54	$f_H'^{6}(sk[2][0:31])$	$f_H^{54}(sk[2][32:63])$
3	9	10	589	65	$f_H'^{10}(sk[3][0:31])$	$f_H^{65}(sk[3][32:63])$
4	14	15	780	55	$f_H'^{15}(sk[4][0:31])$	$f_H^{55}(sk[4][32:63])$
5	7	8	565	80	$f_H'^{8}(sk[5][0:31])$	$f_H^{80}(sk[5][32:63])$
6	7	8	413	59	$f_H'^{8}(sk[6][0:31])$	$f_H^{59}(sk[6][32:63])$
7	9	10	561	62	$f_H'^{10}(sk[7][0:31])$	$f_H^{62}(sk[7][32:63])$
8	8	9	373	46	$f_H'^{9}(sk[8][0:31])$	$f_H^{46}(sk[8][32:63])$
9	7	8	554	79	$f_H'^{8}(sk[9][0:31])$	$f_H^{79}(sk[9][32:63])$
a	1	2	73	73	$f_H'^{2}(sk[10][0:31])$	$f_H^{73}(sk[10][32:63])$
b	8	9	645	80	$f_H'^{9}(sk[11][0:31])$	$f_H^{80}(sk[11][32:63])$
c	12	13	887	73	$f_H'^{13}(sk[12][0:31])$	$f_H^{73}(sk[12][32:63])$
d	8	9	448	56	$f_H'^{9}(sk[13][0:31])$	$f_H^{56}(sk[13][32:63])$
e	9	10	483	53	$f_H'^{10}(sk[14][0:31])$	$f_H^{53}(sk[14][32:63])$
f	10	11	710	71	$f_H'^{11}(sk[15][0:31])$	$f_H^{71}(sk[15][32:63])$
checksum				$\sum_{i=0}^{15}\lvert count\_sb[i]-8\rvert=30$	$f_H'^{30}(sk[16][0:31])$	$f_H^{1890}(sk[16][32:63])$
	Signature_one=*fσ*[0]‖*fσ*[1]‖…‖*fσ*[16]					
	Signature_two=*bσ*[0]‖*bσ*[1]‖…‖*bσ*[16]					

(b) 过程参数与签名

Fig. 2 Signature Creation of 'Hello World!'

图 2 'Hello World!'签名生成过程

算法 3. 签名验证算法.

输入:待签名消息 M,签名 *Signature_one*, *Signature_two*,公钥集合 $pk[]$;

输出:成功 Succeed/失败 Failed.

① 根据算法 2 中的步骤计算 *iteration*[],*sub_iteration*[]和 *checksum*.

② $lf \leftarrow 1$;

③ $lb \leftarrow 1$;

④ for $i=0$ to 15 do

/* 前 16 个验证公钥元素 */

⑤ $x \leftarrow Signature\_one.Substrings(lf, lf+256-(sub\_iteration[i]-1)\times 16)$;

/* 提取每个 $f\sigma[i]$元素 */

⑥ $lf \leftarrow lf+256-(sub\_iteration[i]-1)\times 16$;

⑦ $y \leftarrow Signature\_two.Substrings(lb, lb+256)$;

⑧ $lb \leftarrow lb+256$;

⑨ for $j=1$ to $16-sub\_iteration[i]$ do

/* 计算 $fvk[i]$元素 */

⑩ $x' \leftarrow SHA256(x)$;

⑪ $x \leftarrow x'.Substrings(1, x'.length-(sub\_iteratio\ n[i]-1+j)\times 16)$;

/* 取子串操作 */

⑫ end for

⑬ for $k=1$ to $129-iteration[i]$ do

/* 计算 $bvk[i]$元素 */

⑭ $y \leftarrow SHA256(y)$;

⑮ end for

⑯ $fvk \leftarrow SHA256(x)$;

⑰ $bvk \leftarrow y$;

⑱ $vk[i] \leftarrow fvk \parallel bvk$;

⑲ end for

⑳ $x \leftarrow Signature\_one.Substrings(lf, -1)$;

```
㉑ y←Signature_two.Substrings(lb,-1);
㉒ for i=1 to 1921-checksum do
㉓   x←SHA256(x);
㉔ end for
㉕ for i=1 to checksum+1 do
㉖   y←SHA256(y);
㉗ end for
㉘ vk[16]←x ‖ y;
㉙ for i=0 to 16 then
㉚   if vk[i]==pk[i]
㉛     output:Succeed;
㉜   else
㉝     output:Failed;
㉞   end if
㉟ end for
```

以下生成验证密钥.首先,验证者用 $SHA256$ 函数分别对前 16 个 $f\sigma[i]$和 $b\sigma[i]$元素迭代 $16-sub_iteration[i]$和 $129-iteration[i]$次,生成对应的前 16 个 $fvk[i]$和$bvk[i]$值.其中,如算法 3 中行⑨~⑫所示,在生成 $fvk[i]$的过程中,采用了取子串的操作,每个哈希输入值由 $sub_iteration[i]$决定.式(8)展示了所选取子串的长度.如算法 3 中行⑳~㉗所示,第 17 个验证密钥元素 $fvk[16]$和 $bvk[16]$是由对应的 $f\sigma[16]$和 $b\sigma[16]$哈希迭代 $1921-checksum$ 和 $checksum+1$ 次生成.每个验证密钥元素 $vk[i]$由对应的 $fvk[i]$与 $bvk[i]$合并构成.

$$x=x'.Substrings(1,x'.length-(sub\_iteration[i]-1+j)\times 16),\quad j=1\rightarrow 16-sub\_iteration[i]. \tag{8}$$

如果每个公钥元素 $pk[i]$与对应的验证密钥元素 $vk[i]$相等,则验证成功.算法 3 展示了详细的签名验证过程.

3 安全性分析

本节将证明 SOTS 的存在不可伪造性.

3.1 存在不可伪造性

在第 1.2 节中介绍的 CPA 模型的基础上,本节将阐述 SOTS 的存在不可伪造性定义.本文先假设伪造者 F 的能力.假设 F 只知道公钥 pk,并且只能询问一个消息的签名.

定义 1[6]. SOTS 的存在不可伪造性定义为:在签名预言机(O)知道新的密钥对(sk,pk),伪造者 F 只知道公钥 pk 的前提下,伪造者 F 向 O 询问一个消息 M^Q 的合理签名.当接收到来自 O 返回的签名 σ^{M^Q},F 试图返回一个新的合理的消息-签名对(M',$\sigma^{M'}$),即需要满足 $\sigma^{M'}$ 是合理的,并且$M^Q\neq M'$的条件.如果在时间 t 内,F 成功返回消息-签名对的概率不大于ε,则就说 SOTS 在 CPA 模型下是存在不可伪造的,记为$(t,\varepsilon,1)$-EU.

3.2 SOTS 的安全性证明

本节将证明只要底层哈希函数是一个单向哈希函数,SOTS 在 CPA 模型下就是存在不可伪造的,即证明 SOTS 的安全性可规约为底层哈希函数的安全性.算法 4 阐述了一个攻击者 $ADV_{\text{onewayness}}$ 利用 F 去攻击所用的单向哈希函数 h_{ow}.在这个过程中,$ADV_{\text{onewayness}}$扮演签名预言机 O 的角色.

算法 4. 攻击函数的单向性.

输入:单向哈希函数 h_{ow},SOTS 签名算法、安全参数 n、伪造者 F、像 y;

输出:y 的前像 x,使得 $y=h_{ow}(x)$.

```
① 生成一个新的 SOTS 密钥对(sk,pk);
② 随机选取 α∈{0,1,…,16};
     /* 选择 y 信息放入的公钥 */
③ if 0≤α≤15 then
④   随机选取 β∈{1,2,…,128};
⑤   bpk[α]←h_ow^(129-β)(y);
⑥   伪造者 F 询问消息 M^Q 的签名;
⑦   if iteration^(M^Q)[α]<β then
⑧     Return fail;
⑨   else
⑩     构造消息 M^Q 的签名;
⑪     When i≠α then
        构造签名元素 σ^(M^Q)[i];
        /* 方法同 SOTS */
⑫     When i=α then {
⑬       fσ^(M^Q)[α]←h_ow^(sub_iteration^(M^Q)[α])(fsk[α]);
⑭       bσ^(M^Q)[α]←h_ow^(iteration^(M^Q)[α]-β)(y); }
⑮   end if
⑯   发送 σ^(M^Q) 给伪造者 F;
⑰   When F 返回一个消息/签名对 (M',σ') then {
⑱   if M'≠M^Q 且σ'是有效签名 then
⑲     if iteration^(M')[α]>β then
⑳       Return fail;
㉑     else
㉒       计算 x←h_ow^(β-iteration^(M')[α]-1)(bσ^(M')[α]);
㉓       Return x; /* ADV_onewayness 成功 */
```

㉔ end if
㉕ end if }
㉖ 其他所有情况 Return fail;
㉗ end if
㉘ if $\alpha=16$ then /* 变换第 17 个公钥 */
㉙ 随机选取 $\beta\in\{1,2,\cdots,1920\}$;
㉚ $fpk[\alpha]\leftarrow h_{ow}^{1921-\beta}(y)$;
㉛ F 询问消息 M^Q 的签名;
㉜ if $checksum^{M^Q}<\beta$ then
㉝ Return fail;
㉞ else
㉟ 构造消息 M^Q 的签名;
㊱ When $0\leqslant i\leqslant 15$ then
构造签名元素 $\sigma^{M^Q}[i]$;
/* 方法同 SOTS */
㊲ When $i=\alpha$ then {
㊳ $f\sigma^{M^Q}[\alpha]\leftarrow h_{ow}^{checksum-\beta}(y)$;
㊴ $b\sigma^{M^Q}[\alpha]\leftarrow h_{ow}^{1920-checksum}(bsk[\alpha])$; }
㊵ end if
㊶ 发送 σ^{M^Q} 给伪造者 F;
㊷ When F 返回一个消息/签名对
(M',σ') then {
㊸ if $M'\neq M^Q$ 且 σ' 是有效签名 then
㊹ if $checksum^{M'}>\beta$ then
㊺ Return fail;
㊻ else
㊼ 计算 $x\leftarrow h_{ow}^{\beta-checksum^{M'}-1}(f\sigma^{M'}[\alpha])$;
㊽ Return x; /* $ADV_{\text{onewayness}}$ 成功 */
㊾ end if
㊿ end if }
51 其他所有情况 Return fail;
52 end if

如算法 4 中行①~②可见,攻击者构造一个新的密钥对(sk,pk),随机从 0~16 中选择一个数 α 来决定放入 y 的信息的公钥元素.根据第 2 节中 SOTS 的构造方法可见,第 17 个公钥元素的构造与前 16 个公钥元素的构造不同,第 17 个签名的生成方法也与前 16 个不同,因此,这节将分为 2 种情况讨论.

第 1 种情况是 $0\leqslant\alpha\leqslant 15$.本文参考文献[6]中的方法来证明提出的方案的安全性.$ADV_{\text{onewayness}}$ 随机生成一个取值在[0,128]范围内的数 β,将 y 用哈希函数 h_{ow} 迭代 $129-\beta$ 次,将结果作为第 $\alpha+1$ 个公钥元素 $bpk[\alpha]$.随后,运行 F.当 F 询问一个消息 M^Q,如果该消息的 $iteration[\alpha]$ 值满足算法 4 中⑦的条件,则攻击失败并退出.否则,$ADV_{\text{onewayness}}$ 将按照算法 4 中行⑩~⑮的方式构造合理的签名 σ^{M^Q},反馈给 F.如果 F 构造的消息-签名对$(M',\sigma^{M'})$没有满足算法 4 中行⑲的条件,则 y 的前像 x 可以从 $\sigma^{M'}$ 中推测出来,使得 $h_{ow}(x)=y$.x 具体的计算方式如算法 4 的行㉒所示.

可见,只有 $ADV_{\text{onewayness}}$ 成功的构造消息 M^Q 的签名,F 成功返回一个消息-签名对$(M',\sigma^{M'})$,并且 x 可以从$(M',\sigma^{M'})$中推测出来,攻击才是成功的.式(9)说明了满足 $0\leqslant\alpha\leqslant 15$ 并且攻击成功的概率.因此,最大的成功概率为 $0.0074\varepsilon_F$,即 $Pr(ADV_{\text{onewayness}})\leqslant 0.0074\varepsilon_F$.式(10)计算了攻击过程需要的总时间.式(11)和(12)展示了参数 $t_{\text{FOR\_SOTS}}$ 和 $\varepsilon_{\text{SOTS}}$ 的取值,因此,在 CPA 模型和$(t_{\text{FOR_SOTS}},\varepsilon_{\text{SOTS}},1)$参数下,SOTS 是存在不可伪造的,记为:$(t_{\text{FOR_SOTS}},\varepsilon_{\text{SOTS}},1)$-EU-CPA.即,$ADV_{\text{onewayness}}$ 在时间 $t_{\text{FOR\_SOTS}}$ 内成功返回一个合理消息-签名对$(M',\sigma^{M'})$的概率不超过 $\varepsilon_{\text{SOTS}}$.这也能看出该方案的安全性可规约为底层哈希函数的单向性.

$$Pr(ADV_{\text{onewayness}})=\frac{16}{17}\times\frac{(128-\beta)}{128}\varepsilon_F\ \frac{\beta}{128},\quad(9)$$

$$t_{ADV_{\text{onewayness}}}=t_{\text{Key\_Generation}}+t_{Sign\sigma M}+t_{\text{FOR\_SOTS}},\quad(10)$$

$$t_{\text{FOR\_SOTS}}=t_{ADV_{\text{onewayness}}}-t_{\text{Key\_Generation}}-t_{sign\sigma M},\quad(11)$$

$$\varepsilon_{\text{SOTS}}\leqslant\frac{5\,000}{37}pr(ADV_{\text{onewayness}}).\quad(12)$$

第 2 种情况是:$\alpha=16$,即变换第 17 个公钥值.所用的初始密钥对同第 1 种情况.首先,$ADV_{\text{onewayness}}$ 随机选取一个范围为[1,1920]的整数 β,将 y 哈希迭代 $1921-\beta$ 次,生成新的公钥元素 $fpk[\alpha]$.随后,攻击的步骤与第一种情况相同.算法 4 中行㉘~㊿详细阐述了该情况下的攻击过程和需要满足的条件.式(10)、式(13)计算了攻击所用的时间和成功的概率.可见,成功的概率不超过 $0.000\,03\varepsilon_F$.同理,在这种情况下,SOTS 是存在不可伪造的,即$(t_{\text{FOR_SOTS}},\varepsilon_{\text{SOTS}},1)$-EU-CPA.

$$Pr(ADV_{\text{onewayness}})=\frac{1}{17}\times\frac{1920-\beta}{1920}\varepsilon_F\ \frac{\beta}{1920}.\quad(13)$$

因此,SOTS 在 CPA 模型和伪造者的假设条件下是存在不可伪造的.它的安全性可规约为底层所用哈希函数的单向性.

4 密钥和签名尺寸

本节将计算 SOTS 的密钥和签名大小,并与其他的 WOTS 变体方案比较.

4.1 SOTS 密钥和签名尺寸

通过第 2 节对签名方案的阐述,密钥和签名尺寸是可以计算的.17 个私钥元素是一个随机种子 *seed* 通过哈希函数 SHA512 迭代生成,17 个公钥元素是通过对应的私钥生成,则它们的尺寸为

$$pk\_size=sk\_size=|sk|\times 512=|pk|\times 512=17\times 512=1.06\ \text{KB}.$$

由算法 2 可见,每个签名元素都是由对应私钥元素通过 SHA256 哈希函数迭代生成.在 $f\sigma[i]$生成过程中,进行了取子串操作,而迭代的次数和取子串的长度由十六进制字符在消息哈希中的特征决定.因此,由于每个字符的数量不同,哈希函数迭代的次数和生成的签名长度是不固定的.从式(7)可以看出,最长和最短的 $f\sigma[i]$元素的长度分别是 16 b 和 256 b.因此,由 17 个 $f\sigma[i]$元素合并生成的 *Signature_one* 的长度也是变化的.每个 $b\sigma[i]$的长度为 256 b,因此,*Signature_two* 的长度是固定的.因此,SOTS 的签名尺寸最小为 0.59 KB,最大为 1.06 KB,平均长度为 0.83 KB.

4.2 评 估

SOTS 是一个基于 WOTS 改进的签名方案,可以提供更小的密钥和签名尺寸.表 3 将 SOTS 与其他 WOTS 变体方案进行对比.

Table 3 Sizes and Security: OTS Schemes[6-8]

表 3 OTS 方案[6-8]的尺寸和安全性

签名方案	安全参数/b	消息哈希长度/b	哈希函数	签名尺寸/KB	密钥尺寸/KB	后量子安全水平/b
LD-OTS	512	512	SHA512	32.80	65.50	171
WOTS	512	512	SHA512	8.40	8.40	171
WOTS+	384	512	SHA384	6.30	7.10	185
NOTS	512	512	SHA256	1.00	1.00	128
SDS-OTS	512	512	SHA512	1.10	1.10	164
WOTS-S	384	384	SHA384	1.60	9.28	128
SOTS	512	512	SHA256	0.83	1.06	128

可见,在保证 128 b 的后量子安全水平下,在签名尺寸上,相比于 WOTS 和 WOTS+分别减小了 90.1%和 86.8%.与最近提出的 NOTS,SDS-OTS 和 WOTS-S 方案相比,在签名尺寸上,分别减少了 17%,24.5% 和 48.1%.SOTS 相比于 WOTS 和 WOTS+,在密钥尺寸上,分别减少了 87.4%和 85.1%.

在保证相同的 128 b 的后量子安全级别下,WOTS 的密钥和签名尺寸都为 4.6 KB,WOTS+的密钥和签名尺寸分别为 2.7 KB 和 2.1 KB.因此,在 128 b 后量子安全水平下,相比于 WOTS,在签名和密钥尺寸上分别减少了 82%和 77%.相比于 WOTS+,在签名和密钥尺寸上分别减少了 60.5%和 60.7%.

5 运行效率评估

为了评估 SOTS 运行的效率,即时间,本文将 SOTS 与 LD-OTS,WOTS,WOTS+,NOTS,SDS-OTS,WOTS-S 方案进行对比.图 3～5 是在 Intel Core i7-8650U CPU(2.1 GHz)处理器,16 GB RAM 的"JetBrains PyCharm Community Edition 2019.2 EAP"环境中运行 Windows 10 x64 系统,并使用 python 语言编译得到的结果.图 3～5 分别展示了这 7 个签名方案在表 3 的参数下,生成密钥、构造签名和验证签名所需要的时间.

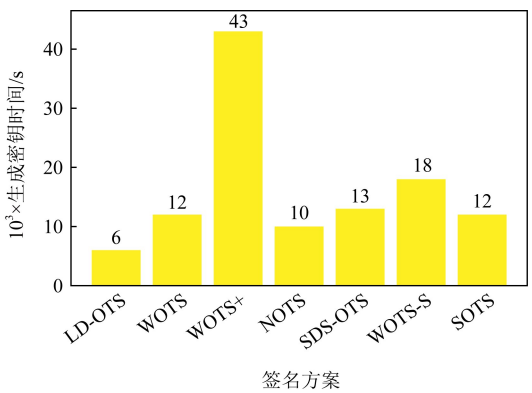


Fig. 3 Key generation time of the OTS schemes

图 3 OTS 方案生成密钥的时间

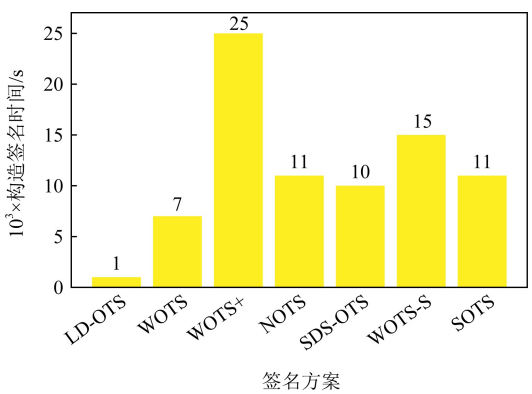


Fig. 4 Signature creation time of the OTS schemes

图 4 OTS 方案构造签名的时间

由结果可见,WOTS+方案运行效率很低.与 WOTS+相比,SOTS 在生成密钥、构造签名和验证签名的时间上,分别减少了 71.4%,47.7%和 60.9%.与 WOTS 方案相比,SOTS 以增加很短的运行时间

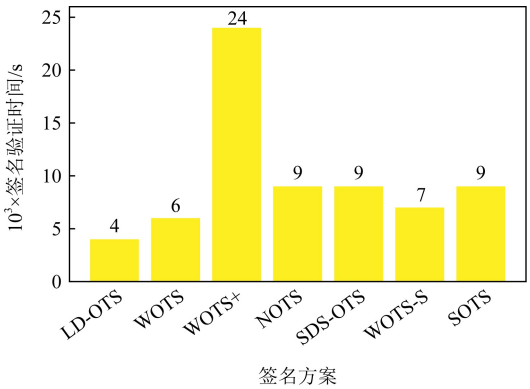


Fig. 5 Signature verification time of the OTS schemes

图 5 OTS 方案验证签名的时间

为代价,大量减少了空间上的占用.SOTS 与 NOTS,SDS-OTS 方案的运行效率相差甚小.相比于 WOTS-S,SOTS 在签名和验证时间上有明显减少.

6 总结和展望

在 CPA 模型下,SOTS 数字签名方案是存在不可伪造地高效的 WOTS 变体方案.在相同 128 b 后量子安全级别下,与 WOTS 相比,SOTS 在密钥和签名尺寸上分别减少了 77%和 82%.与 WOTS+相比,SOTS 在密钥和签名上分别减少了 60.7%和 60.5%.除此之外,SOTS 运行效率高.与 WOTS+相比,在生成密钥、构造签名和验证签名的时间上分别减少了 71.4%,47.7%和 60.9%.在未来的研究中,我们将应用 SOTS 于分布式账本中,基于有向无环图设计一个抗量子的共识机制来提高分布式账本的安全性和性能.

参 考 文 献

[1] Jogenfors J. Quantum bitcoin: An anonymous, distributed, and secure currency secured by the no-cloning theorem of quantum mechanics [C] //Proc of 2019 IEEE Int Conf on Blockchain and Cryptocurrency (ICBC). Piscataway, NJ: IEEE, 2019: 245-252

[2] Merkle R C. A certified digital signature [C] //Proc of the Conf on the Theory and Application of Cryptology. Berlin: Springer, 1989: 218-238

[3] Hülsing A. W-OTS+—shorter signatures for hash-based signature schemes [C] //Proc of the Int Conf on Cryptology in Africa. Berlin: Springer, 2013: 173-188

[4] El Bansarkhani R, Geihs M, Buchmann J. Pqchain: Strategic design decisions for distributed ledger technologies against future threats [J]. IEEE Security & Privacy, 2018, 16(4): 57-65

[5] Shahid F, Khan A, Jeon G. Post-quantum distributed ledger for Internet of things [J]. Computers & Electrical Engineering, 2020, 83: No.106581

[6] Shahid F, Ahmad I, Imran M, et al. Novel one time signatures (NOTS): A compact post-quantum digital signature scheme [J]. IEEE Access, 2020, 8: 15895-15906

[7] Shahid F, Khan A. Smart digital signatures(SDS): A post-quantum digital signature scheme for distributed ledgers [J]. Future Generation Computer Systems, 2020, 111: 241-253

[8] Shahid F, Khan A, Malik S U R, et al. WOTS-S: A quantum secure compact signature scheme for distributed ledger [J]. Information Sciences, 2020, 539: 229-249

[9] Chalkias K, Brown J, Hearn M, et al. Blockchained post-quantum signatures [C] //Proc of 2018 IEEE Int Conf on Internet of Things(iThings) and IEEE Green Computing and Communications(GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). Piscataway, NJ: IEEE, 2018: 1196-1203

Wei Hongru, born in 1963. Master, associate professor. His main research interests include mathematics, information security and cryptology, and Internet of things technology.

卫宏儒,1963 年生.硕士,副教授.主要研究方向为数学、信息安全与密码学和物联网技术.

Huang Jingyi, born in 1997. Master. Her main research interests include post-quantum cryptography, distributed ledgers, and the blockchain.

黄靖怡,1997 年生.硕士.主要研究方向为后量子密码学、分布式账本和区块链.