Post-quantum Signature Schemes in Hcash

Lab of Cryptology and Computer Security (LoCCS)
Shanghai Jiao Tong University
China

1 Introduction

Bitcoin is the first decentralized cryptocurrency based on the blockchain technology [Nak08]. The blockchain has proven to be a remarkable way to implement secure decentralized systems. The consensus scheme of Bitcoin is provably secure, and transactions in blockchain are protected by well-developed cryptographic signatures.

The upcoming quantum computers threatens almost all the cryptography which are foundation of the internet security. The quantum computing is also a huge threat to the blockchain based cryptocurrency. In particular, the quantum algorithm Shor [Sho99] for computing discrete logarithms breaks the ECDSA signature scheme used by Bitcoin. When that happens, one can easily derive the secret key from the public key presented in the Bitcoin transactions. Therefore, when a transaction is broadcast to the network, it is at risk before being placed on the blockchain. In particular, attackers may intercept the transaction, obtain the public key published with it, and compute the corresponding secret key. Then, the attacker can modify the transaction content and generate a valid signature of the modified transaction. If the new transaction is placed on the blockchain before the original one, the attacker can steal the bitcoin from the original output address.

The Hcash project aims to build a secure, efficient, robust and reliable decentralized system. Among all the features Hcash is designed to present, the security against quantum computing is one of the most attractive. Many signature schemes have been proposed and are presumed to be quantum-safe. Hcash incorporates two most popular post-quantum signature schemes, BLISS [DDLL13] and MSS/LMS [BDH11,

LM95]. BLISS is the currently most efficient post-quantum signature scheme with the smallest public key and signature sizes, while MSS/LMS is a most efficient hash-based signature scheme, which relies on weak security assumptions, i.e. no more than the security of the underlying hash function. [ABL+17]. Both schemes have been thoroughly analyzed and evaluated by the academic community with respect to their security.

Although BLISS and MSS/LMS signatures are comparably small among all the post-quantum signature schemes, they are still larger than the traditional ECDSA signatures. Large signatures will increase the size of the transactions and reduce the amount of payments the system is able to provide. To mitigate the negative effects exerted by the large signatures, we design a new block transmission protocol based on the idea of segregated witness.

In Section 2, we will give a brief review of the post-quantum signature schemes in the literature, and explain the reason for choosing BLISS and MSS/LMS for Hcash. Section 3 demonstrates the BLISS scheme, and presents our countermeasures against the side-channel attacks on BLISS. Section 4 describes the MSS/LMS scheme. We brief the block transmission protocol in Section 5. Finally, Section 6 concludes this report.

2 Post-quantum Signature Schemes

Many public-key signature schemes have been proposed in the literature, and are presumed to be quantum-safe. Generally speaking, the existing schemes can be classified into four categories:

- Hash-based signature schemes, including MSS [Mer89, DOTV08], LMS [LM95, MCF17], XMSS [BDH11], SPHINCS [BHH+15] and NSW [NSW05];
- Lattice-based signature schemes, including GVP [GPV08], LYU [Lyu12], GLP [GLP12], BLISS [DDLL13], DILITHIUM [DLL+17] and NTRU [MBDG14];
- Code-based signature schemes, including CFS [CFS01] and QUARTZ [PCG01];
- Multivariate-polynomial-based signature schemes, including RAINBOW [DS05].

Table 1 presents the signature sizes and public-key sizes of the schemes [ABL+17,

BDH11, dOL15] [1].

Table 1: Signature and Public-key Sizes of Post-Quantum Signature Schemes

Type	Name	Security (bits)	PK Size (kb)	Sig. Size (kb)
I	LMS	128	0.448	22.624
	MSS	128	0.256	30.752
	XMSS	100	13.568	15.384
	SPHINCS	128	8	328
	NSW	128	0.256	36
II	GPV	100	300	240
	LYU	100	65	103
	GLP	100	12	9
	BLISS	128	7	5
	DILITHIUM	138	11.8	21.6
III	CFS	83	9216	0.1
	QUARTZ	80	568	0.128
IV	RAINBOW	160	305	0.244

With respect to the public-key and signature sizes, the only practical options are lattice and hash based signature schemes. The most significant advantage of hash-based schemes is having provable security, at least in the random oracle model. Currently, the most effective quantum-attack against the hash-based schemes is Grover's searching algorithm [Gro96]. As a result of the Grover's algorithm, the quantum security level of a hash-based scheme is only half of the classical security level. Therefore, at the same quantum security level, lattice-based schemes have advantage in signature and public key sizes.

MSS was developed by Ralph Merkle in the late 1970s, based on one-time-signatures (OTS) such as Winternitz (WOTS). The security of MSS depends only on the security of hash functions, i.e. collision resistance and second pre-image resistance. LMS is one of the many variations of MSS which improves MSS with respect to its security and efficiency. LMS was proposed by Leighton and Mechali in 1995, and has gone through a number of revisions based on the analysis by Katz [Kat16, Kat] and Fluhrer [Flu17]. We decide to use the most updated LMS algorithm [MCF17] as the hash-based signature in Hcash. We use SHA3 as the underlying hash function,

[1]Most of the data presented in Table 1 are referred to [ABL+17] Table II. However, we believe that the public key size of XMSS presented in [ABL+17] is unreliable, and we use the data from the original XMSS paper [BDH11] instead.

which is strong in collision resistance and second pre-image resistance.

The BLISS algorithm has the shortest signature and public key sizes of all existing post-quantum signature schemes. The security of BLISS relies on hardness of the NTRU problem, and the assumption that solving this problem is equivalent to finding a short vector in the NTRU lattice. The disadvantage of BLISS is its difficulty to implement in a secure way, as it is susceptible to side channel attacks [PBY17].

Note that DILITHIUM is another lattice-based signature scheme which has higher level of security, though at the cost of larger public keys and signatures, in comparison with BLISS. Currently there is no side-channel attacks on DILITHIUM, which makes it seemingly more secure than BLISS. However, DILITHIUM is proposed only recently, therefore its security in implementation is not as thoroughly analyzed as that of BLISS. Considering our countermeasures against the revealed side-channel attacks on BLISS, and the advantage in efficiency of BLISS, together with the significant impact of the public key and signature size on the performance of blockchain based cryptocurrencies, we think BLISS is a better choice compared to DILITHIUM.

Hcash implements both MSS/LMS and BLISS schemes, taking the advantage of the efficiency of BLISS and the security of MSS/LMS. To address the side-channel attacks on BLISS, we incorporates many side-channel-proof techniques in our implementation of BLISS.

3 BLISS Algorithm

The BLISS signature scheme is an improvement to the LYU [Lyu12] scheme by replacing the distribution of the signature, i.e. *discrete Gaussian distribution*, with a *bimodal* Gaussian distribution. This modification significantly reduces the *reject sampling rate*, which is used to force the signature distribution into a fixed Gaussian distribution, so as to eliminate any information leakage by the distribution of signature. The improvement is explained by Fig. 1. Here we give a brief explanation of the basic idea of BLISS, the full description will be presented in the complete Hcash paper.

BLISS proposes an efficient discrete Gaussian sampling algorithm. Before that, all known algorithms to sample according to a distribution statistically close to a discrete Gaussian distribution on a lattice require either long-integer arithmetic at some point or large memory storage. The Gaussian sampling algorithm proposed in BLISS

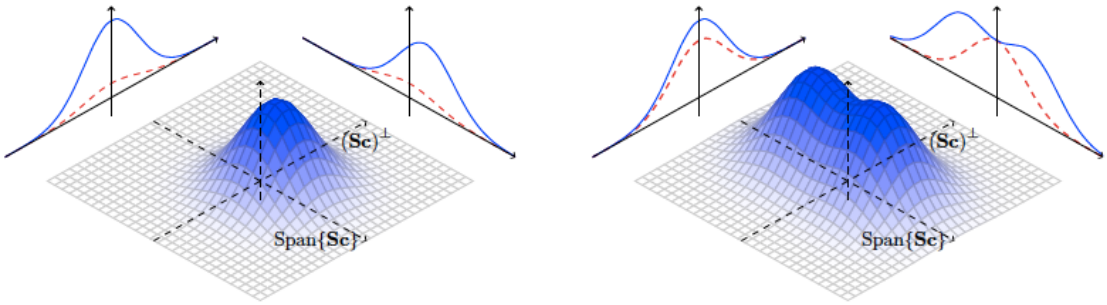
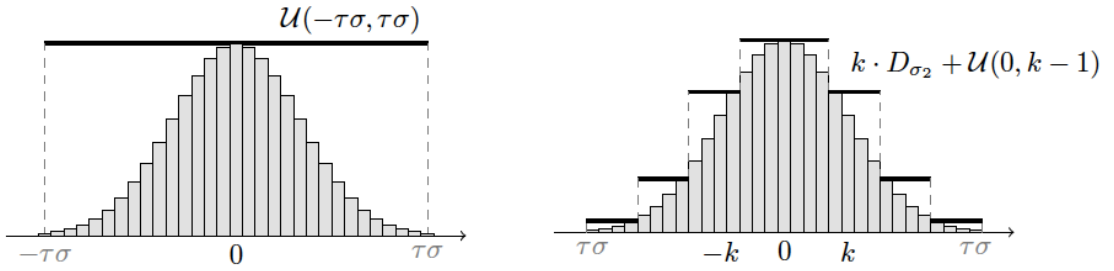


Figure 1: Improvement of Rejection Sampling with Bimodal Gaussian Distributions

can efficiently sample discrete Gaussian without resorting to large precomputed tables, nor evaluations of transcendental function. First, algorithms are proposed to sample according to a Bernoulli distribution with bias of the form $\exp(x/f)$ and $1/\cosh(x/f))$ without actually computing transcendental functions. Then, BLISS builds an efficient distribution based on the *binary discrete Gaussian distribution*, and apply rejection sampling to this distribution to obtain the target discrete Gaussian distribution. This procedure is explained by Fig. 2.



(a) from uniform distribution (repetition rate ≈ 10)

(b) from our adapted distribution (repetition rate ≈ 1.47)

Figure 2: Sample from Discrete Gaussian by Rejection Sampling

The algorithm for sampling Bernoulli distribution with bias of the form $\exp(x/f)$ is presented in Algorithm 1. Bernoulli distribution with bias of the form $1/\cosh(x/f))$ is sampled based on the fact that $\mathcal{B}_{1/\cosh(x)} = \mathcal{B}_{\exp(-|x|)} \oslash (\mathcal{B}_{1/2} \vee \mathcal{B}_{\exp(-|x|)})$ where $\mathcal{B}_a \oslash \mathcal{B}_b$ is defined as $\mathcal{B}_{a/(1-(1-a)b)}$. The binary discrete Gaussian distribution is sampled by Algorithm 2. The final discrete Gaussian distribution sampling algorithm is presented in Algorithm 3.

BLISS also replaces the ring $\mathcal{R} = \mathbb{Z}$ from which the matrix elements are taken, using

Algorithm 1 Sampling Bernoulli Distribution with Bias $\exp(x/f)$

Input: $x \in [0, 2^\ell)$ an integer in binary form $x = x_{\ell-1} \cdots x_0$

Compute $c_i = \exp(-2^i/f)$ for $0 \le i \le \ell - 1$
for $i = \ell - 1$ **to** 0 **do**
  if $x_i = 1$ **then**
    Sample A_i according to Bernoulli distribution of probability c_i
    if $A_i = 0$ **then return** 0
  end if
end for

Algorithm 2 Sampling Binary Discrete Gaussian Distribution

Output: An integer $x \in \mathbb{Z}^+$ according to binary discrete Gaussian distribution

Generate a bit $b \leftarrow \mathcal{B}_{1/2}$
if $b = 0$ **then return** 0
for $i = 1$ **to** ∞ **do**
  Draw random bits $b_1, \cdots, b_k$ for $k = 2i - 1$
  if $b_1 \cdots b_k \neq 0 \cdots 0$ **restart**
  if $b_k = 0$ **then return** i
end for

Algorithm 3 Sampling Discrete Gaussian Distribution

Input: An integer $k \in \mathbb{Z}$

Output: An integer $z \in \mathbb{Z}^+$ according to discrete Gaussian distribution of deviation $k\sigma_2$

Sample x by Algorithm 2
Sample y uniformly in $\{0, \cdots, k-1\}$
Compute $z \leftarrow kx + y$
Sample $b \leftarrow \mathcal{B}_{\exp(-y(y+2kx)/(2\sigma^2))}$
if not b **then restart**
if $z = 0$ **then restart** with probability $1/2$
Generate a bit $b \leftarrow \mathcal{B}_{1/2}$ and return $(-1)^b z$

the NTRU ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(x^n + 1)$ instead. The size of the public key matrix $\boldsymbol{A}$ and private key matrix $\boldsymbol{S}$ are reduced to 1×2 and 2×1 respectively. The private key of BLISS consists of two polynomials $\boldsymbol{s}_1, \boldsymbol{s}_2$ from $\mathcal{R}_q$, and the public key consists of one polynomial $\boldsymbol{a}_1$ (the other element of the matrix $\boldsymbol{A}$ is a constant $q - 2$). The signature consists of a pair of polynomials $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$, together with a challenge $\boldsymbol{c}$, which is a sparse polynomial with coefficients from $\{0, 1\}$.

BLISS reduces the signature size by compressing the polynomial $\boldsymbol{z}_2$ without compromising the security. In particular, BLISS replaces $\boldsymbol{z}_2$ by $\boldsymbol{z}_2^\dagger$ which equals $\lfloor \boldsymbol{u} \rceil_d - \lfloor \boldsymbol{u} - \boldsymbol{z}_2 \rceil_d$, where $\lfloor \cdot \rceil_d$ means dropping d bits from each coefficients of the polynomial. Compared to $\boldsymbol{z}_2$, the coefficients of $\boldsymbol{z}_2^\dagger$ are much smaller, thus easier to compress by Huffman codes.

The BLISS scheme has been further optimized in efficiency, and the improved version is called BLISS-B [Duc14]. BLISS-B replaces the product of polynomials $\boldsymbol{S} \cdot \boldsymbol{c}$ by a procedure $\mathsf{GreedySC}(\boldsymbol{S}, \boldsymbol{c})$, which effectively computes $\boldsymbol{S} \cdot \boldsymbol{c}'$ where $\boldsymbol{c}'$ differs from $\boldsymbol{c}$ in the sign of a subset of the nonzero coefficients. $\mathsf{GreedySC}(\boldsymbol{S}, \boldsymbol{c})$ efficiently derives a $\boldsymbol{c}'$ such that $\|\boldsymbol{S} \cdot \boldsymbol{c}'\|_2$ is minimized, reducing the reject sampling rate by a considerable factor. The algorithm for computing $\mathsf{GreedySC}(\boldsymbol{S}, \boldsymbol{c})$ is presented in Algorithm 4.

Algorithm 4 GreedySC

Input: a matrix $\boldsymbol{S} = [\boldsymbol{s}_1, \cdots, \boldsymbol{s}_n]$ and a binary vector $\boldsymbol{c}$
Output: $\boldsymbol{v} = \boldsymbol{S}\boldsymbol{c}'$ for some $\boldsymbol{c}' \equiv \boldsymbol{c} \bmod 2$

```
v ← 0 ∈ Z^m
for i ∈ I_c do
    ζ_i ← sgn(⟨v, s_i⟩)
    v ← v − ζ_i · s_i
end for
```

The final algorithms of the BLISS scheme are presented in Alg. 5, Alg. 6 and Alg. 7.

3.1 BLISS Against Side-channel Attacks

BLISS has been demonstrated to be vulnerable to side-channel attacks [EFGT17, PBY17]. A major part of the vulnerability comes from the discrete Gaussian sampling which plays an important role in lattice cryptography [MW17]. Much efforts

Algorithm 5 BLISS Key Generation

Output: Key pair $(\boldsymbol{A}, \boldsymbol{S})$ such that $\boldsymbol{AS} = q \bmod 2q$

1: Choose $\boldsymbol{f}, \boldsymbol{g}$ as uniform polynomials with exactly d_1 entries in $\{\pm 1\}$ and d_2 entries in $\{\pm 2\}$
2: $\boldsymbol{S} = (\boldsymbol{s}_1, \boldsymbol{s}_2)^T \leftarrow (\boldsymbol{f}, 2\boldsymbol{g} + 1)^T$
3: **if** $N_\kappa(\boldsymbol{S}) \geq C^2 \cdot 5 \cdot (\lceil \delta_1 n \rceil + 4\lceil \delta_2 n \rceil) \cdot \kappa$ **then**
4: **restart**
5: **end if**
6: $\boldsymbol{a}_q = (2\boldsymbol{g} + 1)/\boldsymbol{f} \bmod q$ (**restart** if $\boldsymbol{f}$ is not invertible)
7: **Output**$(\boldsymbol{A}, \boldsymbol{S})$ where $\boldsymbol{A} = (2\boldsymbol{a}_q, q - 2) \bmod 2q$

Algorithm 6 BLISS Signature Algorithm

Input: Message μ, public key $\boldsymbol{A} = (\boldsymbol{a}_1, q - 2) \in \mathcal{R}_{2q}^{1\times 2}$, secret key $\boldsymbol{S} = (\boldsymbol{s}_1, \boldsymbol{s}_2)^T \in \mathcal{R}_{2q}^{2\times 1}$

Output: A signature $(\boldsymbol{z}_1, \boldsymbol{z}_2^\dagger, \boldsymbol{c})$ of the message μ

1: $\boldsymbol{y}_1, \boldsymbol{y}_2 \leftarrow D_{\mathbb{Z}^n, \sigma}$
2: $\boldsymbol{u} = \zeta \cdot \boldsymbol{a}_1 \cdot \boldsymbol{y}_1 + \boldsymbol{y}_2 \bmod 2q$
3: $\boldsymbol{c} \leftarrow H(\lfloor \boldsymbol{u} \rceil_d \bmod p, \mu)$
4: $(\boldsymbol{v}_1, \boldsymbol{v}_2) \leftarrow \mathsf{GreedySC}(\boldsymbol{s}, \boldsymbol{c})$
5: Choose a random bit b
6: $(\boldsymbol{z}_1, \boldsymbol{z}_2) \leftarrow (\boldsymbol{y}_1, \boldsymbol{y}_2) + (-1)^b \cdot (\boldsymbol{v}_1, \boldsymbol{v}_2)$
7: **Continue** with probability $1/\left(M \exp\left(-\frac{\|v\|^2}{2\sigma^2}\right) \cosh\left(\frac{\langle \boldsymbol{z}, \boldsymbol{v} \rangle}{\sigma^2}\right)\right)$ otherwise **restart**
8: $\boldsymbol{z}_2^\dagger \leftarrow (\lfloor \boldsymbol{u} \rceil_d - \lfloor \boldsymbol{u} - \boldsymbol{z}_2 \rceil_d) \bmod p$
9: **Output** $(\boldsymbol{z}_1, \boldsymbol{z}_2, \boldsymbol{c})$

Algorithm 7 BLISS Verification Algorithm

Input: Message μ, public key $\boldsymbol{A} = (\boldsymbol{a}_1, q - 2) \in \mathcal{R}_{2q}^{1\times 2}$, signature $(\boldsymbol{z}_1, \boldsymbol{z}_2^\dagger, \boldsymbol{c})$

Output: Accept or Reject the signature

1: **if** $\|(\boldsymbol{z}_1 \mid 2^d \cdot \boldsymbol{z}_2^\dagger)\|_2 > B_2$ **then** Reject
2: **if** $\|(\boldsymbol{z}_1 \mid 2^d \cdot \boldsymbol{z}_2^\dagger)\|_\infty > B_\infty$ **then** Reject
3: Accept iff $\boldsymbol{c} = H(\lfloor \zeta \cdot \boldsymbol{a}_1 \cdot \boldsymbol{z}_1 + \zeta \cdot q \cdot \boldsymbol{c} \rceil_d + \boldsymbol{z}_2^\dagger \bmod p, \mu)$

have been devoted in designing a discrete Gaussian sampling algorithm with high efficiency and security. Another source of information leakage comes from the rejection sampling, which causes the program execution and memory access to depend on the random data.

We take the following countermeasures to strengthen the BLISS implementation in Hcash against the side-channel attacks. First, we implement the Bernoulli sampler with probability in the form of e^x in constant-time way. The Bernoulli sampler is used in the discrete Gaussian sampling algorithm utilized in BLISS [DDLL13], and the execution procedure depends on the bits of x. Specifically, the sampler invokes a table lookup for each bit of value 1 in x. We eliminate this potential source of leakage by forcing the program to execute the lookup regardless of the bit value.

Second, we prevent the attackers from perceiving the sampled $\boldsymbol{y}$ by splitting $\boldsymbol{y}$ into the sum of two independently sampled $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$. Then we compute $\boldsymbol{Ay}$ by first computing $\boldsymbol{Ay}_1$ and $\boldsymbol{Ay}_2$ and take the addition. We carefully select the standard deviation and other parameters of $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ such that the statistical distance $\Delta(\boldsymbol{y}, \boldsymbol{y}_1+\boldsymbol{y}_2)$ is negligible according to Theorem 3.1 in [Pei10]. Due to the overhead brought by the protection techniques, the signature generation procedure is three times slower than the unprotected BLISS, which is still fast enough.

4 The MSS and LMS Algorithms

The hash-based signature schemes start from a one-time signature scheme (OTS), that is, a signature scheme where each key pair cannot be used to sign more than one message. Currently the most efficient OTS scheme is the Winternitz scheme (WOTS) [BBD09]. The core idea of WOTS is to iteratively apply a function on a secret input, whereas the number of iterations depends on the message to be signed. The functions are members of the function family

$$F(n) = \{f_k := \{0,1\}^n \to \{0,1\}^n\} \mid k \in \{0,1\}^n$$

Let $f_k^0(x) := x, f_k^1(x) := f_k(x)$ and $f_k^i(x) := f_{f_{k-1}(x)}(x)$.

The WOTS is parameterized by the hash size n, and the compression level $w \in \mathbb{Z}^+, w > 1$.

WOTS key generation. To generate a pair of key for WOTS, choose a random value $x \in \{0,1\}^n$. Let ℓ be computed as follows

$$\ell_1 = \left\lceil \frac{n}{\log(w)} \right\rceil, \quad \ell_2 = \left\lfloor \frac{\log(\ell_1(w-1))}{\log(w)} \right\rfloor + 1, \quad \ell = \ell_1 + \ell_2$$

The secret key sk consists of ℓ uniformly randomly chosen bit-strings of length n. The public key pk is computed from sk as follows: $\mathsf{pk}_0 = x$, $\mathsf{pk}_i = f_{\mathsf{sk}_i}^{w-1}(x)$ for $1 \leq i \leq \ell$.

WOTS signature generation. To sign an n-bit message $M = (M_1, \cdots, M_{\ell_1})$ given in base-w representation, i.e. $0 \leq M_i \leq w-1$. First compute the checksum

$$C = \sum_{i=1}^{\ell_1} (w - 1 - M_i)$$

and represents C in base w as $(C_1, \cdots, C_{\ell_2})$. Then we set $B = (b_1, \cdots, b_\ell) = M \| C$. The signature is computed as $\sigma = (\sigma_1, \cdots, \sigma_\ell) = (f_{\mathsf{sk}_1}^{b_1}(x), \cdots, f_{\mathsf{sk}_\ell}^{b_\ell}(x))$.

WOTS signature verification. To verify the signature, first computes the base-w string B as described above, then check that

$$(f_{\sigma_1}^{w-1-b_1}(x), \cdots, f_{\sigma_\ell}^{w-1-b_\ell}(x)) = (\mathsf{pk}_1, \cdots, \mathsf{pk}_\ell)$$

The Merkle signature scheme (MSS) works with any cryptographic hash function and any one-time signature scheme. MSS uses a Merkle tree to link $N = 2^H$ OTS public keys together, where H is the height of the tree. The Merkle tree root is then used as the public key of MSS. The algorithm for computing the Merkle root is called Treehash, see Algorithm 8.

To generate a signature of a message M, take the secret key of the next unused OTS public key, and generate a signature of OTS. The signature of MSS consists of the OTS signature, the OTS public key, and an authentication path to prove the existence of the OTS public key in the Merkle tree. To verify the signature, first check the validity of the OTS signature against the OTS public key, then verify that the OTS public key is a valid leaf of the Merkle tree with the authentication path.

Algorithm 8 Threehash

Input: Height $H > 2$
Output: Root of the Merkle tree

```
for j = 0, ··· , 2^H − 1 do
    Compute the jth leaf: NODE_1 ← LEAFCALC(j)
    while NODE_1 has the same height as the top node on STACK do
        Pop the top node from the stack: NODE_2 ← STACK.pop()
        Compute their parent node: NODE_1 ← g(NODE_2‖NODE_1)
    end while
    Push the parent node on the stack: STACK.push(NODE_1)
end for
Let R be the single node stored on the stack: R ← STACK.pop()
return R
```

The MSS private key consists of 2^H OTS secret keys. Storing such a huge amount of data is not feasible for most practical applications. Space can be saved by using a deterministic pseudo random number generator (PRNG) and storing only the seed of that PRNG. Then each one-time signature key must be generated twice, once for the MSS public key generation and once during the signing phase. In addition to the private key size, using a PRNG for the one-time signature key generation has another benefit. It makes MSS forward secure as long as PRNG is forward secure.

Computing the Merkle root and an authentication path from a PRNG seed is a computation consuming work. Meanwhile, saving the entire Merkle tree in the memory is efficient but causes large memory usage. To leverage the storage and computation cost, various Merkle tree traversing algorithms are proposed. The main purpose of Merkle tree traversal is to sequentially output the leaf values of all nodes, together with the associated authentication path. Currently the most efficient Merkle tree traversal algorithm is one proposed by Szydlo in [Szy04]. See Algorithm 9 for details.

LMS proposed by Leighton Micali is a modification to the MSS with respect to the security and efficiency. Here we briefly introduce the modification to the secret key, the public key and the signature respectively.

As for the one-time **secret key**, LMS prepend security fields including `typecode`, I and q as a prefix to the plain secret key $\mathbf{x} = (x_0, \ldots, x_{\ell-1})$ in MSS, and the final secret key is encoded as

$$\text{typecode} \| I \| q \| x_0 \| \ldots \| x_{\ell-1}$$

Algorithm 9 Logarithmic Merkle Tree Traversal

1: Set $s = 0$
2: **for each** $h \in [0, H-1]$ **output** AUTH_h
3: **for all** h such that 2^h divides $s+1$ **do**
4: Set AUTH_h be the sole node value in STACK_h
5: Set startnode $= (s+1+2^h) \oplus 2^h$
6: $\text{STACK}_h.initialize(\text{startnode}, h)$
7: **end for**
8: **for** i **from** 1 **to** $2H-1$ **do**
9: Let $\ell_{\min}$ be the minimum of $\text{STACK}_h.low$
10: Let $focus$ be the least h such that $\text{STACK}_h.low = \ell_{\min}$
11: $\text{STACK}_h.update$
12: **end for**
13: Set $s = s+1$
14: **if** $s < 2^H$ **goto** line 2

Hence, the secret key in LMS comsumes 24 bytes more than that of MSS.

When it comes to the **public key** (i.e., the Merkle root), a similar pre-padding is employed, The public key of one LMS is encoded as

$$\text{sigtype}\|\text{otstype}\|I\|\nu$$

where `sigtype` and `otstype` are both `typecode`, and ν is the origin public key for MSS.

A LMS **signature** σ is formatted as

$$q\|\sigma_{ots}\|\text{sigtype}\|A_s$$

where $\sigma_{ots} = \text{otstype}\|C\|\sigma_0\|\ldots\|\sigma_\ell$ is the one-time signature, and $A_s = \{a_i\}_{i=0}^{\ell-1}$.

In MSS, the one-time public key is $\mathbf{y} = (y_0, y_1, \ldots, y_\ell)$, and it can be derived from σ_{ots} and the message M if σ_{ots} on M is signed by the paired secret key of $\mathbf{y}$. In LMS, the one-time public key $\mathbf{y} = (y_0, y_1, \ldots, y_\ell)$ to verify the one-time signature σ_{ots} part of σ is compressed as

$$\mathbf{y} = \text{typecode}\|I\|q\|H(I\|q\|\text{D\_PBLC}\|y_0\|y_1\|\ldots\|y_\ell)$$

while the counterpart in MSS stores each y_i publicly. Every time σ_{ots} needs verifying, the one-time public key candidate $\mathbf{z} = (z_0, z_1, \ldots, z_\ell)$ can be derived from $\sigma_{ots} =$

$(\sigma_0, \sigma_1 \ldots, \sigma_\ell)$, and then check the extended digest of $\mathbf{z}$ as $H(I\|q\|\text{D\_PBLC}\|z_0\|z_1\|\ldots\|z_\ell)$ against the corresponding component $H(I\|q\|\text{D\_PBLC}\|y_0\|y_1\|\ldots\|y_\ell)$ in the public key.

Both LMS and MSS are based on the Merkle tree. Other intermediate routines such as updating the authentication path $A_s = (a_0, a_1, \ldots, a_\ell)$ through traversing the tree, are the same. The major difference is the compression of the one-time public $\mathbf{y}$ in the σ_{ots} component.

5 Block Transmission Protocol

Although BLISS and MSS/LMS signatures are comparably small among all the post-quantum signature schemes, they are still larger than the traditional ECDSA signatures. Large signatures will increase the size of the transactions and reduce the amount of payments the system is able to provide. To mitigate the negative effects exerted by the large signatures, we design a new block transmission protocol based on the idea of segregated witness. Here we give a brief explanation of the basic idea of our block transmission protocol, the full description will be presented in the complete Hcash paper.

Simply speaking, the idea of segregated witness is to separate the signatures from the transaction body. Specifically, the transaction signature is not taken into calculation of the transaction ID. We further improve this idea by allowing users to transmit only the transaction IDs when passing a block. In this way, the communication cost of transmitting blocks decreases significantly. On receiving a block, a node checks if he/she has already obtained (and saved into the memory pool) all the transactions whose IDs are in the block. If he/she misses some transaction, he/she will ask the peer who gave him/her the block for the missing transaction. Fig. 3 is an example of the transmission procedure of a newly generated block.

6 Conclusion

Hcash is the first blockchain-based currency system that supports multiple post-quantum signature schemes, and is compatible with the traditional ECDSA signature. We are the first to propose a side-channel-safe implementation of the post-quantum signature schemes. The block transmission protocol based on segregated

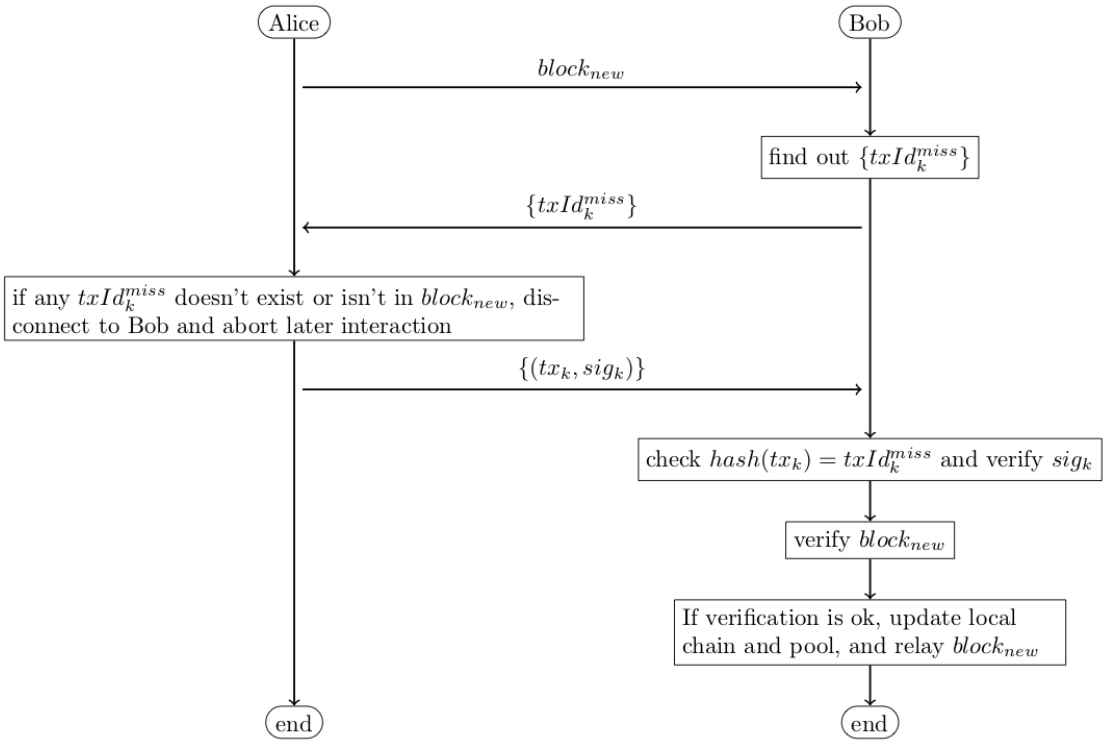


Figure 3: Broadcast of a new block

witness will significantly cut down the communication cost brought by large post-quantum signatures, and this protocol can be generalized and transplanted into other blockchain and cryptocurrency systems.

References

[ABL+17] Divesh Aggarwal, Gavin K Brennen, Troy Lee, Miklos Santha, and Marco Tomamichel. Quantum attacks on bitcoin, and how to protect against them. *arXiv preprint arXiv:1710.10377*, 2017.

[BBD09] Daniel J Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-quantum cryptography*. Springer Science & Business Media, 2009.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss: a practical forward secure signature scheme based on minimal security assumptions. *Post-Quantum Cryptography*, pages 117–129, 2011.

[BHH+15] Daniel J Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. Sphincs: practical stateless hash-based signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 368–397. Springer, 2015.

[CFS01] Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a mceliece-based digital signature scheme. In *Asiacrypt*, volume 2248, pages 157–174. Springer, 2001.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013.

[DLL+17] Léo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-dilithium: Digital signatures from module lattices. Technical report, Cryptology ePrint Archive, Report 2017/633, 2017.

[dOL15] Ana Karina DS de Oliveira and Julio López. An efficient software implementation of the hash-based signature scheme mss and its variants.

In *International Conference on Cryptology and Information Security in Latin America*, pages 366–383. Springer, 2015.

[DOTV08] Erik Dahmen, Katsuyuki Okeya, Tsuyoshi Takagi, and Camille Vuillaume. Digital signatures out of second-preimage resistant hash functions. *PQCrypto*, 5299:109–123, 2008.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *ACNS*, volume 5, pages 164–175. Springer, 2005.

[Duc14] Léo Ducas. Accelerating bliss: the geometry of ternary polynomials. *IACR Cryptology ePrint Archive*, 2014:874, 2014.

[EFGT17] Thomas Espitau, Pierre-Alain Fouque, Benoit Gerard, and Mehdi Tibouchi. Side-channel attacks on bliss lattice-based signatures – exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. Cryptology ePrint Archive, Report 2017/505, 2017. `https://eprint.iacr.org/2017/505`.

[Flu17] Scott Fluhrer. Further analysis of a proposed hash-based signature standard. Technical report, Cryptology ePrint Archive, Report 2017/553, 2017.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In *CHES*, volume 7428, pages 530–547. Springer, 2012.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 197–206. ACM, 2008.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[Kat] Jonathan Katz. Analysis of a proposed hash-based signature standard, rev. 4.

[Kat16] Jonathan Katz. Analysis of a proposed hash-based signature standard. In *Security Standardisation Research*, pages 261–273. Springer, 2016.

[LM95] Frank T Leighton and Silvio Micali. Large provably fast and secure digital signature schemes based on secure hash functions, July 11 1995. US Patent 5,432,852.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–755. Springer, 2012.

[MBDG14] Carlos Aguilar Melchor, Xavier Boyen, Jean-Christophe Deneuville, and Philippe Gaborit. Sealing the leak on classical ntru signatures. In *International Workshop on Post-Quantum Cryptography*, pages 1–21. Springer, 2014.

[MCF17] Dr. David A. McGrew, Michael Curcio, and Scott Fluhrer. Hash-Based Signatures. Internet-Draft draft-mcgrew-hash-sigs-08, Internet Engineering Task Force, October 2017. Work in Progress.

[Mer89] Ralph C Merkle. A certified digital signature. In *Conference on the Theory and Application of Cryptology*, pages 218–238. Springer, 1989.

[MW17] Daniele Micciancio and Michael Walter. Gaussian sampling over the integers: Efficient, generic, constant-time. In *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, pages 455–485, 2017.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[NSW05] Dalit Naor, Amir Shenhav, and Avishai Wool. One-time signatures revisited: Have they become practical? *IACR Cryptology ePrint Archive*, 2005:442, 2005.

[PBY17] Peter Pessl, Leon Groot Bruinderink, and Yuval Yarom. To bliss-b or not to be - attacking strongswan's implementation of post-quantum signatures. Cryptology ePrint Archive, Report 2017/490, 2017. `https://eprint.iacr.org/2017/490`.

[PCG01] Jacques Patarin, Nicolas Courtois, and Louis Goubin. Quartz, 128-bit long digital signatures. In *Cryptographers' Track at the RSA Conference*, pages 282–297. Springer, 2001.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *Annual Cryptology Conference*, pages 80–97. Springer, 2010.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In *Eurocrypt*, volume 3027, pages 541–554. Springer, 2004.