

Post Quantum Cryptography (PQC) Signatures Without Trapdoors

William J. Buchanan ¹

Blockpass ID Lab, Edinburgh Napier University, Edinburgh.

Abstract. Some of our current public key methods use a trap door to implement digital signature methods. This includes the RSA method, which uses Fermat’s little theorem to support the creation and verification of a digital signature. The problem with a back-door is that the actual trap-door method could, in the end, be discovered. With the rise of PQC (Post Quantum Cryptography), we will see a range of methods that will not use trap doors and provide stronger proof of security. In this case, we use hash-based signatures (as used with SPHINCS+) and Fiat Shamir signatures using Zero Knowledge Proofs (as used with Dilithium).

1 Introduction

Some of our existing public key methods, such as RSA, use a trapdoor method to produce a digital signature. The problem with a trap door is that eventually the target will find out where the catch is and not fall down the trap door. For many of our existing public key methods, we have used a trap door — a magical way to reverse an operation. The strength of the method is then the strength of the trap door method. This is the case for the RSA method, and which uses Fermat’s Little Theorem to perform the trap door. Our new post-quantum cryptography methods avoid the trapdoor technique and focus on more difficult problems. For quantum robust signatures which do not use trap doors, there are two main methods: lattice approaches with the Fiat-Shamir method [1] for a non-interactive approach. For PQC (Post Quantum Cryptography, NIST has standardised the hash-based methods as SLH-DSA (FIPS 205, aka SPHINCS+ [2]), and the Fiat-Shamir approach with ML-DSA (FIPS 204, aka CRYSTALS Dilithium [3].

2 A trap door method

First, let’s look at a trap door method. We generate the RSA key pair with two prime numbers (p and q), and compute the modulus (N) [4]:

$$N = pq \tag{1}$$

We then compute:

$$\phi = (p - 1).(q - 1) \quad (2)$$

We then pick an encryption key value ($e = 0x010001$) and compute:

$$d = e^{-1} \pmod{\phi} \quad (3)$$

The public key is then (e, N) and the private key is (d, N) . To sign the hash of a message (H), we create a signature with:

$$S = H^d \pmod{N} \quad (4)$$

and then verify with:

$$H = S^d \pmod{N} \quad (5)$$

This all works because of Little Fermat's Theorem, and which is defined as:

$$a^{p-1} \equiv 1 \pmod{p} \quad (6)$$

3 Hash-based signatures

We can create a one-time signature using the Lamport approach [5]. With this, we generate 256 private keys for A and 256 private keys for B [6]. The public key is then the hash of all of these keys (512 in total). When we sign, we take a hash of the message (giving us 256 bits of a hash). We then go through each bit, and if we have a 0, we select the associated private key from A. Otherwise, we select it from B. We then go through the bits and end up with 256 keys, either from A or B. This gives us a 256x256 signature size (8 kB). When verifying, we check that each of the hashed values is contained in the public key. This is, of course, a one-time signature, and we would have to create new keys for the next signature and republish our public key (Figure 1).

To improve this, we can define the WOTS+ (Winternitz one-time signature scheme) approach [7,8]. The method is (Figure 2):

- Initially, we create 32 256-bit random numbers. These 32 values will be our private key.
- We then hash each of these values 256 times. These 32 values will be our public key.
- We now take the message and hash it using SHA-256. This produces 32 8-bit values ($N_1, N_2 \dots N_{32}$).
- For the signature, we take each 8-bit value in the hash of the message, and then hash 256- N times (where N is the value of the 8-bit value).

- To prove the signature, we take the message and hash it with SHA-256, and then take each 8-bit value. We then hash the 8-bit signature value by the number of times defined by the message hash value ($N_1, N_2..$). The result for each operation should equal the public key value.
- For the signature, we take each 8-bit value in the hash of the message, and then hash 256- N times (where N is the value of the 8-bit value).

In this way, we cut down the size of the signature and of the public key. We can then produce a Merkle Root of our public key and private key, and reduce them down to a single 256-bit value. We can then create a tree of these keys, so that we can sign more than once and have different private key, but still a single 256-bit private key and a 256-bit public key. This is the method that SPHINCS+ uses, and which has small public and private keys, but has a larger signature (around 17KB for 128-bit security):

Table 1: SPHINCS+ key and cipher sizes

Method	Public key size	Private key size	Signature size
Security level			
SPHINCS+ 256 128-bit	SHA- 32	64	17,088
SPHINCS+ 256 192-bit	SHA- 48	96	35,664
SPHINCS+ 256 256-bit	SHA- 64	128	49,856

4 Fiat Shamir signatures using Zero Knowledge Proofs

The method for creating the lattice-based signatures is actually based on the Schnorr signature method [9] for proof of identity (a Zero Knowledge Proof) and then applies the Fiat-Shamir method [1] to make it non-interactive. Basically, it is a NI-ZKP (Non-interactive Zero Knowledge Proof) of a secret (a person's private key).

If Bob wants to prove that he knows a secret, he creates a short random secret vector (x). He then creates a random matrix array of \mathbf{A} , and an error matrix of small errors (e_1). His public key (*textbfu*) is then:

$$u = \mathbf{A}.x + e_1 \quad (7)$$

Let's ignore the error, as it will eventually be removed when we perform the calculations. So let's just use:

$$u = \mathbf{A}.x \quad (8)$$

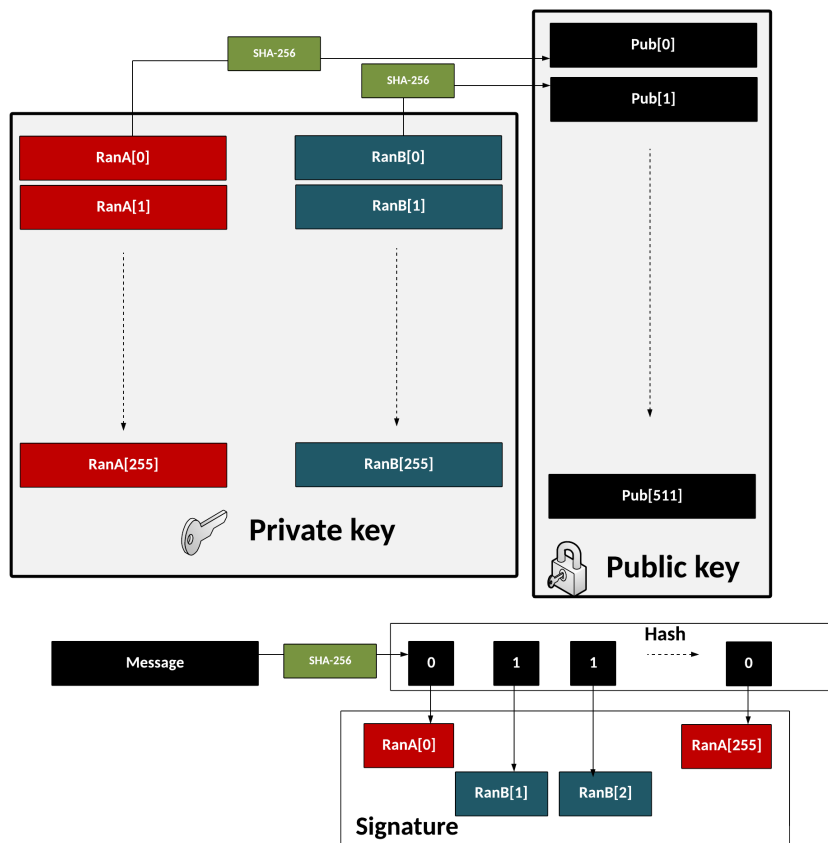


Fig. 1. Lamport one-time signatures

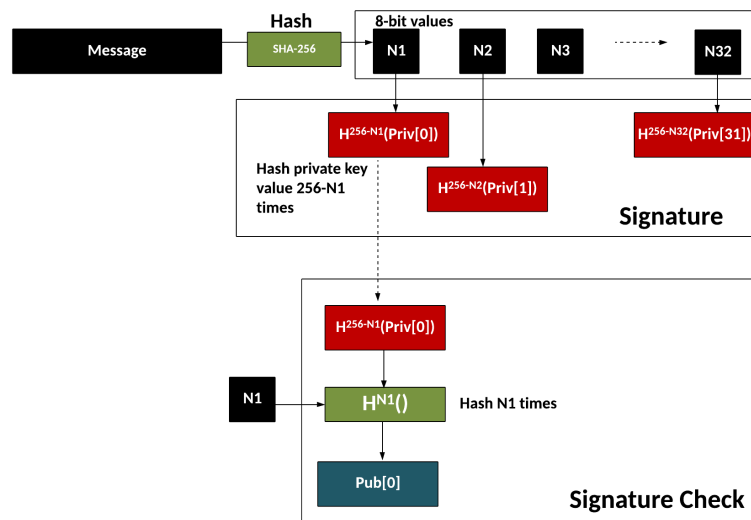
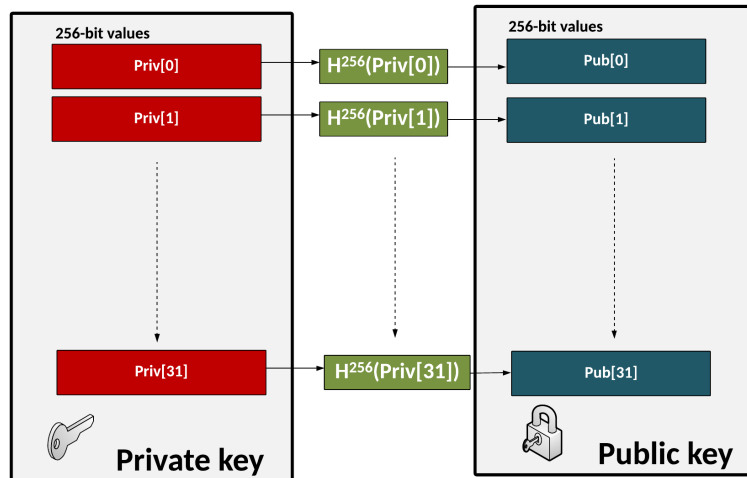


Fig. 2. WOTS+ one-time signatures

This value will be sent to Alice. She then can send a challenge (c) to Bob, and which is a random integer value. Bob then creates a random short vector of y , and computes:

$$v = \mathbf{A}.y + e_2 \quad (9)$$

Let's ignore e_2 again, as it will eventually be removed. Bob then computes using the Schnorr identity proof:

$$z = c.x + y \quad (10)$$

Bob then sends x and v to Alice, and she checks if these are equal:

$$\mathbf{A}.z = c.\mathbf{u} + \mathbf{v} \quad (11)$$

If they are, then Bob has proven that he knows the value of x (his private key). This works because:

$$\mathbf{A}.z = \mathbf{A}(c.x + y) = \mathbf{A}.c.x + \mathbf{A}.y \quad (12)$$

$$c.\mathbf{u} + \mathbf{v} = c.\mathbf{A}.x + \mathbf{A}.y \quad (13)$$

Now, we can turn a zero-knowledge proof into a signature by using the Fiat-Shamir heuristic. With this, there is no need for Alice to send a challenge, as Bob can compute:

$$c = H(v||M) \quad (14)$$

and where M is the message, “ $||$ ” is a concatenation of the values, and $H()$ is a hash function (such as with SHA-256). We can then just perform the same operations, and Alice will be able to regenerate c from the values of the message and v , and check that:

$$\mathbf{A}.x = c.\mathbf{u} + \mathbf{v} \quad (15)$$

5 Conclusions

Many of our existing public key methods are based on trapdoor methods. Improved proofs of security can often be created using methods which do not have a trap door. In this case, we have outlined two core methods for creating non-trapdoor-based signatures: hash-based signatures and converting the Schnorr identity method [9] into a signature using the Fiat-Shamir heuristic [1] that converts a Zero Knowledge Proof into a digital signature method.

References

1. A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.
2. D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, “The sphincs+ signature framework,” in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2129–2146.
3. L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, “Crystals-dilithium: A lattice-based digital signature scheme,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.
4. W. J. Buchanan, “Hazmat rsa signatures (with key values),” <https://asecuritysite.com/signatures/rsasig>, Asecuritysite.com, 2025, accessed: April 18, 2025. [Online]. Available: <https://asecuritysite.com/signatures/rsasig>
5. L. Lamport, “Constructing digital signatures from a one way function,” 1979.
6. W. J. Buchanan, “Quantum robust: Lamport signature,” <https://asecuritysite.com/hashsig/lamport>, Asecuritysite.com, 2025, accessed: April 18, 2025. [Online]. Available: <https://asecuritysite.com/hashsig/lamport>
7. —, “Quantum robust: Winternitz one time signature scheme (w-ots),” <https://asecuritysite.com/hashsig/wint>, Asecuritysite.com, 2025, accessed: April 18, 2025. [Online]. Available: <https://asecuritysite.com/hashsig/wint>
8. A. Hülsing, “W-ots+—shorter signatures for hash-based signature schemes,” in *Progress in Cryptology—AFRICACRYPT 2013: 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22–24, 2013. Proceedings 6*. Springer, 2013, pp. 173–188.
9. C.-P. Schnorr, “Efficient identification and signatures for smart cards,” in *Advances in Cryptology—CRYPTO’89 Proceedings 9*. Springer, 1990, pp. 239–252.