UDC 004.056

A.V. POTII, A.S. KARPENKO

REALIZATION OF THE MECHANISM OF CONTROL SOFTWARE INTEGRITY IN POST QUANTUM PERIOD

1. Introduction

Digital signature is important primitive of modern cryptography. Most security protocol such as SSH, TLS, SSL are using digital signature for verify the integrity and authenticity of the information. The resistance of the cryptographic algorithms with the public key is based on the computational complexity of the problems of factorization of large integers, discrete logarithms and transformation of the points on the elliptic curve. The known algorithms are RSA, DSA, and ECDSA (Table 1) [1].

Investigations in the sphere of quantum calculations form up new challenges in the given sector of the cryptography. With using of the quantum computer and the Shor algorithm the known at present crypto algorithms with the public key would be compromise. Today, regional organizations such as NIST and ETSI are already research in this field. The workgroups of ETSI and NIST determined the promising trends, within the framework of which there could be obtained acceptable solutions – the supersingular elliptic curves, the multi-variative cryptography, the cryptography on the basis of the noise immunity encoding, and the cryptography based on the hash functions. Recently, NIST open a competition for the standardization of the digital signature algorithm in the post quantum stage. This publication focuses on algorithms based on the use a hash function. Their main advantage is that they rely on simple assumptions on hash functions, such as collision or second-preimage resistance, instead of a specific algebraic structure. In particular, if the attack is detect in a hash function, one can replace it by another function without modifying the overall structure of the scheme. Most hash-based schemes also come with relatively simple proofs of security reductions to the hash function's properties. Their main drawback is signature size, which typically grows with the number of messages signed by a key pair [2-5].

A significant part of the research is focused on increasing of their efficiency. Besides, the simplest hash-based schemes are stateful, which means that a signer must maintain a state that is modify every time a signature is issued. This requirement can be a burden because trivial forgeries become possible if it is violated once, e.g. if two signatures are issued in the same state. Stateful schemes must therefore guarantee that this kind of misuse will not happen, which can be non-trivial for practical systems. In theory, somebody can rolling back the state of a machine after a crash, cloning virtual machines, or maintaining a pool of signing machines working in parallel. Hence, it is advisable to use a stateless scheme GRAVITY [6].

Table 1

Security level of the applied algorithms

Algorithm	**Key length**	**Security level**	
		Classical computer	**Quantum computer**
RSA-1024	1024 b	80 b	0 b
RSA-2048	2048 b	112 b	0 b
ECC-256	256 b	128 b	0 b
ECC-384	384 b	256 b	0 b

2. Stateless algoruthm

GRAVITY scheme have three stage: key generation, signature, verification and batch signature and verification.

An instance of the GRAVITY signature scheme requires the following parameters: the hash output bit length n, a positive integer; the Winternitz [7] depth w, a power of two such that w>=2; the PORS [6] set size t, a positive power of two; the PORS subset size k, a positive integer such that k<=t; the internal Merkle [7] tree height h, a positive integer; the number of internal Merkle [7] trees d, a non-negative integer; the cache height c, a non-negative integer; the batching height b, a non-negative integer; the message space M, usually a subset of bit string {0,1}*. From this parameters are derive the follow values [5]:

– The Winternitz width l

$$l=\mu+\lfloor \log_2(\mu(w-1))/\log_2 w \rfloor+1 \text{ where } \mu=n/\log_2 w \quad (1)$$

– The PORS set $T=\{0,..., t-1\}$.

– The address space A

$$A=\{0,\ldots,d\}\times\{0,...,2^{c+dh}-1\}\times\{0,...,\max(l,t)-1\} \quad (2)$$

– the public key space $PK=B_n$.

– The secret key space $SK=B_n^2$.

– The signature space SG

$$SG=B\times B^k\times B^{k(\log_2 t - |\log_2 k|)}\times (B^l\times B^l)^d\times B^c \quad (3)$$

– The batched signature space SG_B

$$SG_B=B_n^b\times\{0,\ldots,2^b-1\}\times SG \quad (4)$$

– The public key size, of n bits.

– The secret key size, of 2n bits.

– The maximal signature size, of
sigsz = $(1 + k + k(\log_2 t - \lfloor \log_2 k \rfloor) + d(l + h) + +c)n$ bits.

– The maximal batched signature size, of
sigsz + bn + b bits.

2.1 Primitives

GRAVITY signature scheme based on next primitives that depend on scheme parameters [6]:

– a length-preserving hash function $F : B_n \rightarrow B_n$

– a length-halving hash function $H : B_n^2 \rightarrow B_n$

– a pseudo-random function $G : B_n \times A \rightarrow B_n$

– a general-purpose hash function $H^* : M \rightarrow B_n$

2.2 Key generation

Key generation takes as input 2n bits of random numbers and outputs the secret key and public key.

– Generate the secret key from 2n bits of random numbers and put to by address (seed, salt) in the Merle tree

$$sk = (\text{seed}, \text{salt}) \leftarrow B_n^2 \quad (5)$$

– For each i that $0 \le i < 2^{c+h}$ generated a Winternitz public key(WOTS-genpk) and generate next to new address for keys pair (make-addr(0, i))

$$p_i \leftarrow \text{WOTS-genpk}(\text{seed}, \text{make-addr}(0, i)) \quad (6)$$

– Generate the public key (root of Merkle-tree) where x is array of hashes leaf

$$pk \leftarrow \text{Merkle-root}_{c+h}(x_0, \ldots, x_{2^{c+h}-1}) \quad (7)$$

2.3 Signature

Message takes as input a *m* hash and secret key
sk = (seed, salt) and outputs a signature computed as follow [6]:

– Compute the public salt s ← H(salt, m).

– Compute the hyper-tree index and random subset as j, $(x_1, \ldots, x_k) \leftarrow$ PORS(s, m).

– Compute the PORST [6] signature and public key where oct is a parameter of authentication path[6]

$$(\sigma_d, \text{oct}, p) \leftarrow \text{PORST-sign}(\text{seed}, \text{make-addr}(d, j), x_1, \ldots, x_k) \quad (8)$$

For $i \in \{d-1, \ldots, 0\}$ do the following:

– Compute the WOTS (Winternitz one time signature) [7] signature

$$\sigma_i \leftarrow \text{WOTS-sign}(\text{seed}, \text{make-addr}(i, j), p) \quad (9)$$

– compute p ← WOTS-extractpk(p, σ_i).

– Set $j' \leftarrow \lfloor j/2^h \rfloor$.

– for $u \in \{0, \ldots, 2^h-1\}$compute the WOTS public key (WOTS-genpk)

$$p_u \leftarrow \text{WOTS-genpk}(\text{seed}, \text{make-addr}(i, 2^h j + u)) \quad (10)$$

– Compute the Merkle authentication path (Merkle-auth_h)

$$A_i \leftarrow \text{Merkle-auth}_h(p_0, \ldots, p_{2^h-1}, j - 2^h j) \quad (11)$$

– set $j \leftarrow j'$.

– For $0 \le u < 2^{c+h}$ compute the WOTS public key

$$p_u \leftarrow \text{WOTS-genpk}(\text{seed}, \text{make-addr}(0, u)) \quad (12)$$

– Compute the Merkle authentication

$$(a_1, \ldots, a_{h+c}) \leftarrow \text{Merkle-auth}_{h+c}(p_0, \ldots, p_{2^{h+c}-1}, 2^h j) \quad (13)$$

– Set $A \leftarrow (a_{h+1}, \ldots, a_{h+c})$.

– The signature is

$$(s, \sigma_d, \text{oct}, \sigma_{d-1}, A_{d-1}, \ldots, \sigma_0, A_0, A) \quad (14)$$

2.4 Verification

Verification function takes a hash message, public key and a signature $(s,\sigma_d,\text{oct},\sigma_{d-1},A_{d-1},\ldots,\sigma_0,A_0,A)$ and verifies it as follows[6]:

– Compute the hyper-tree index and random subset as j, $(x_1 \ldots x_k) \leftarrow$ PORS(s, m).

– Compute the PORST public key

$$p \leftarrow \text{PORST-extract pk}(x_1 \ldots x_k, \sigma_d, \text{oct}) \quad (15)$$

– If p = ⊥, then abort and return 0.

For $i \in \{d-1, \ldots, 0\}$ do the following:

– Compute the WOTS public key p ← WOTS-extract pk(p, σ_i).

– Set $j' \leftarrow \lfloor j/2^h \rfloor$.

– Compute the Merkle root

$$p \leftarrow \text{Merkle-extract}_h(p, j - 2^{h_j'}, A_i) \quad (16)$$

– set $j \leftarrow j'$.
– Compute the Merkle root

$$p \leftarrow \text{Merkle-extract}_c(p, j, A) \quad (17)$$

– The result is 1 if p = pk, and 0 otherwise.

2.5 Batch signature

Batch signature takes as input a sequence of messages $(M_1,...,M_i) \in M^i$ with $0 < i \leq 2^b$ and a secret key
sk = (seed,salt) along with its secret cache, and outputs *i* signatures σ_j, computed as follows[5]:

– For $j \in \{1,...,i\}$ compute the message digest $m_j \leftarrow H^*(M_j)$.
– For $j \in \{i+1,\ldots,2^b\}$ set $m_j \leftarrow m_1$.
– Compute $m \leftarrow \text{Merkle-root}_b(m_1, \ldots, m_{2^b})$.
– Compute $\sigma \leftarrow S(sk, m)$.
– For $j \in \{1,...,i\}$ the j-th signature is $\sigma_j \leftarrow (j,\text{Merkle-auth}_b(m_1,...,m_{2^b},j),\sigma)$.

For b = 0, we simplify $S_B(sk,M)$ to $S(sk,H^*(M))$.

2.6 Batch verification

Batch verification function V takes as input a public key pk, a message $M \in M$ and a signature (j, A, σ), and verifies it as follows [6]:

– Compute the message digest $m \leftarrow H^*(M)$.
– Compute the Merkle root.
$m \leftarrow \text{Merkle-extract}_b(m, j, A)$.
– The result is V (pk, m, σ).

For b = 0, we simplify $V_B(pk,M,\sigma)$ to $V(pk,H^*(M),\sigma)$.

3. Security proofs

The security of hash based signature depend on using in this scheme hash function. In this article suggests using DSTU 7564:2014 [7].

Ukrainian national hashing function standard is capable of operation with the hash value lengths of 256/384/512 bits. The above hash function base on the Rijndael structure. The DSTU 7564:2014 cryptographic strength is provide in Table II [7].

Table 2

Strength against the cryptographic attacks

Type of attack	**Kupyna – 256**
Collision	2^{128}
Preimage	2^{256}
Second preimage	2^{256}
Fixed point	2^{256}
Increase of the length	2^{256}

4. Development cycle

In this project, propose to use Continuous integration/Continuous delivery (CI/CD) [9] software development practice. It methodology consist of a continuous cycle of plan, code, build, test, release, deploy, operate, monitor. Continuous integration focuses on blending the work products of individual developers together into a repository. Often, this is done several times each day, and the primary purpose is to enable early detection of integration bugs, which should eventually result in tighter cohesion and more development collaboration. The aim of continuous delivery is to minimize the friction points that are inherent in the deployment or release processes. Typically, the implementation involves automation of each of the steps for build deployments such that a safe code release can be done—ideally—at any moment in time.

For development applications propose is use CI/CD tools, in particular, Jenkins, Gitlab on Amazon web service (Fig.1) [9].

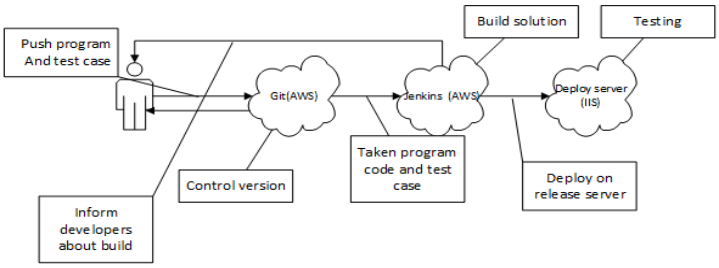

Figure 1. Develop application cycle

Public-key infrastructure consist of three level: web application, desktop application and hardware application (Fig. 2).

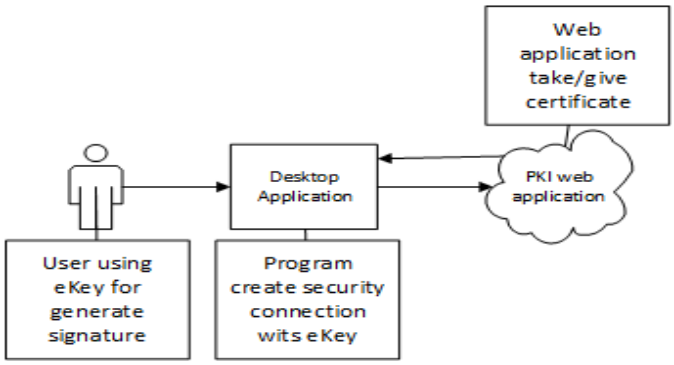

Figure 2. Architecture applications

The web application is being built using C# languages and ASP .NET MVC technology, in turn, desktop application is being built using C# as well and Windows Form technology, hardware application in eKey is being built using C language. Desktop application is create security channel with eKey and transfer data for generation key pair, singing and verification documents.

5. Conclusion

In this article, describe stateless algorithm GRAVITY [6]. These schemes rely on a limited number of assumptions that form collision-resistant, one-way, undetectable and pseudo-random function families. This means that their security is relatively well-understood, even against hypothetical adversaries with a quantum computer. Besides, the stateless property gives some “misuse-resistance” guarantees for signers that cannot reliably maintain a state over the lifetime of a key

pair. Another advantage of hash-based signatures is speed and simplicity of verification. On the signer side, we have seen that several trade-offs are available of signature size, computational resources, memory resources, as well as the planned number of signatures issued by a key pair. Development of group application show how to work public-key infrastructure with using stateless quantum-resistant algorithm.

References:

1. ETSI GR QSC 001 V.1.1.1 (2016-07). Quantum-Safe Cryptography (QSC); Quantum-safe algorithmic framework.
2. NIST PQC workshop: SAFEcrypto Project, M. O'Niell. 2015.
3. NIST Workshop on Cyber Security in a Post-Quantum World (2015). PQCrypto project, T Lange.
4. PQCrypto. Initial recommendation of Long-term secure post-quantum systems.
5. Endignoux G. Design and implementation of a post-quantum hash-based cryptographic signature scheme: mater's thesis:july 2017 /Guillaume Endignoux – Switzerland,2017. 42 p.
6. Gravity-Sphincs : Kudelski Security : 25 p./Jean-Philippe Aumasson, Guillaume Endignoux.
7. Becjer G. Merkle signature schemes, Merkle Trees and Their Cryptanalusis: 2013.
8. Gorbenko Yu.I., (2015), Construction and analysis of systems, protocols, and methods of cryptographic information protection. Pt. 1, Methods of construction and analysis, standardization and application of cryptographic systems, Kharkiv, Ukraine: Fort, (in Ukrainian).
9. Ellingwood. J. CI/CD tools comparison: Jenkins,Gitlab CI etc.[Internet resource]/ Justin Ellingwood. [Access mode]: https://www.digitalocean.com/community/tutorials/ci-cd-tools-comparison-jenkins-gitlab-ci-buildbot-drone-and-concourse. 02.02.2017.

Национальный аэрокосмический
университет имени Н.Е. Жуковского «ХАИ»

Поступила в редколлегию 27.03.2018