Digital signatures over hash-entangled chains

Luis A. Lizama-Perez¹

Received: 5 September 2019 / Accepted: 1 November 2019 / Published online: 7 November 2019
© Springer Nature Switzerland AG 2019

Abstract

In this work we introduce a new approach for digital signature based on hash chains. The method is supported on the well known security principles of hash functions. Some main advantages of this public key system include security and estimated efficiency through the so called signature chains. In addition, the algorithm is a promissory alternative for digital signatures for the quantum era.

Keywords Hash · Chain · Digital signature

1 Introduction

Generally speaking, a digital signature covers the methods useful to demonstrate irrevocably that a user has written a digital message. To achieve the goal, such algorithms have exploited some computational problems based on modular arithmetic or the security properties of hash functions. Unfortunately, methods based on the integer factorization problem or discrete logarithm fail under Shor's algorithm for the quantum era [1]. By contrast, hash functions are considered a good alternative to construct post-quantum algorithms [2]. In such scenario, we have developed a new method based on what we call hash-entangled chains. The term hash-entangled denotes that a hash-chain is linked to a previous chain. Later, we will give detail about the algorithm. First, in the next section, we will describe briefly some of the most relevant systems for digital signatures. Next, we will introduce the hash-entangled method to discuss, right away, the security issues and performance of the algorithm.

1.1 Digital signatures

Some methods have been developed to perform digital signatures. We can mention two main algorithms emerged

among modular arithmetic which are based on the difficulty of the integer factorization problem: the RSA cryptosystem [3, 4] and the Rabin cryptosystem [5]. Also, the Digital Signature Algorithm (DSA) [4, 6, 7] is based on the difficulty of the discrete logarithm problem.

By contrast, other methods have been conceived over the basis of hash functions. For this case, we can mention the one time signature method (OTS) [8], OTS with Merkle Trees [9–11] and Lizama Certification Authority [12]. We identify some major advantages of hash functions which we summarize as follows:

- Well known security foundations
- A variety of algorithms is available
- A promissory alternative to be post-quantum
- Easy and fast to compute hash chains
- Specific hardware has been developed to compute hash chains

In the following section we will introduce a new method to achieve digital signatures based on hash functions. But before to introduce it, let us write the security assumptions of hash functions:

- Arbitrary data size input.

✉ Luis A. Lizama-Perez, luislizama@upp.edu.mx | ¹Sección de Posgrado de la Universidad Politécnica de Pachuca, Ex-Hacienda de Santa Bárbara, 43830 Pachuca, Mexico.

- Fixed data size output (e.g. 128, 160, 256 bits, etc).
- Feasible to compute. Is easy to compute $f(x)$ for all x .
- One way function. For any given hash value h it is computationally infeasible to find x given $f(x) = h$.
- Resistant to collision simple. For any given input x , it is computationally infeasible to find $y \neq x$ so that $f(y) = f(x)$.
- Resistant to strong collision. It is computationally infeasible to find any pair (x, y) in such a way that $f(x) = f(y)$.

Before concluding this section we will mention that hash cryptography is resistant to quantum-crypto-analysis [13]. By contrast, methods based on modular arithmetic and the integer factoring problem or the discrete logarithm problem [14–16] are vulnerable to quantum computers capable to execute Peter Shor's algorithm [1]. However, a new approach based on quantum physics has emerged [17] which has been enhanced to resist quantum attacks [18, 19].

2 Digital signatures using hash-chains

For the present discussion, a hash chain is computed from a secret seed s then performing a hash chain whose length is fixed to $2^n - 1$ (n could be chosen between 128, 160, 256, etc.). After applying the hash function over the seed it yields the head of the chain $f^{2^n-1}(s)$. If some middle point of the chain is shared or revealed, say $f^{2^n-1-x}(s)$, this value conducts to $f^{2^n-1}(s)$ by applying successively x times the hash function, that is $f^x(f^{2^n-1-x}(s)) = f^{2^n-1}(s)$.

From the previous discussion, a message can be signed in the following manner: Alice publishes $f^{2^n-1}(s)$ where s is a secret number and n is publicly shared and it is fixed. Now, taking the hash of the message to be signed say $f(m) = h$ Alice publishes m and $f^{2^n-1-h}(s)$. Any user who wants to verify the signature just compute h and from here $f^h(f^{2^n-1-h}(s)) = f^{2^n-1}(s)$.

Unfortunately, the scheme described before is not completely functional at a practical level. First, consider that in the quantum era a secure hash function must be at least 256 bit length. Therefore, the length of the hash chain grows to $2^{256} - 1$ which implies an enormous computational cost. Moreover, if we want to sign several messages with the same hash chain it must be enhanced properly. To overcome such limitations we will use multiple hash chains. In Fig. 1 we show the approach to reduce the length of the hash chain: the hash of the message $h_0 = f(m_0)$ is divided in two portions h_{00} and h_{01} which correspond to the most and least significant bits of h_0 . This reduces the length of the original hash chain to the half. As it can be inferred, increasing the number of hash chains also increases the number of the public head-chains to

Fig. 1 The length of the hash chain could be reduced projecting the hash of the message over two hash chains

be stored (which constitutes the public key). Let k be the number of the hash chains to be used. The effect of increasing k will be discussed later.

2.1 Digital signatures using multiple hash-chains

Two hash chains could not be enough to minimize the computational effort required to sign a message. Now, we will describe the method using $k = 4$ and a hash function of 128 bits, so any user (say Alice) must perform the following steps to sign messages (see the overall signature process in Fig. 2):

1. Alice chooses randomly four seeds s_1, s_2, s_3 and s_4 , then she computes the hash-chains $f^{2^n-1}(s_1), f^{2^n-1}(s_2), f^{2^n-1}(s_3)$ and $f^{2^n-1}(s_4)$. For this example, the size of each seed is $|s_i| = 128$ bits. The number n is fixed to $128 / k$ thus the length of the hash chain is $2^{32} - 1$. Now, Alice publishes her public head-chains: $P_{u_A} = \{f^{2^{32}-1}(s_1), f^{2^{32}-1}(s_2), f^{2^{32}-1}(s_3), f^{2^{32}-1}(s_4)\}$.
2. To sign the first message m_1 , Alice computes the hash of the message which is denoted as $h_1 = f(m_1)$. Then, she separates the 128 bits in four portions of 32 bits. Let us denote such portions as $h_1^{00}, h_1^{01}, h_1^{10}$ and h_1^{11} . Alice takes the seeds and computes $f^{2^{32}-1-h_1^{00}}(s_1), f^{2^{32}-1-h_1^{01}}(s_2), f^{2^{32}-1-h_1^{10}}(s_3)$ and $f^{2^{32}-1-h_1^{11}}(s_4)$.
3. Alice publishes the message m_1 , so any user who wants to verify the signature must check that: $f^{h_1^{00}}(f^{2^{32}-1-h_1^{00}}(s_1)) = f^{2^{32}-1}(s_1), f^{h_1^{01}}(f^{2^{32}-1-h_1^{01}}(s_2)) = f^{2^{32}-1}(s_2), f^{h_1^{10}}(f^{2^{32}-1-h_1^{10}}(s_3)) = f^{2^{32}-1}(s_3), f^{h_1^{11}}(f^{2^{32}-1-h_1^{11}}(s_4)) = f^{2^{32}-1}(s_4)$.
4. Finally, Alice generates and publishes the next head chains $f^{2^{32}-1}(s_5)$ and $f^{2^{32}-1}(s_6), f^{2^{32}-1}(s_7), f^{2^{32}-1}(s_8)$ as they can be seen in Fig. 2. The process continues to sign an undefined number of incoming messages.

Fig. 2 The overall signature publication process with $k = 4$. Alice sends to the receiver the corresponding messages m_i . The super-indices 00, 01, 10 and 11 denote the portion bits of the hash code. Alice continues signing an undefined number of messages

Fig. 3 Eve cannot compute $f^{2^n-1-h_0-\delta x_1'}(s_0)$ because this operation requires to reverse the hash function from $f^{2^n-1-h_0-\delta x_1}(s_0)$ to $f^{2^n-1-h_0-\delta x_1'}(s_0)$

3 The entanglement algorithm

The security of the method is based in what we call the entanglement algorithm. To describe it, let us consider firstly the case $k = 1$. At the end of this section, we will generalize the algorithm.

Suppose Alice has signed a message m_0 using $f^{2^n-1}(s_0)$, it means she has revealed $f^{2^n-1-h_0}(s_0)$ where $h_0 = f(m_0)$. If Alice signs a new message m_1 so $h_1 = f(m_1)$ an intruder could fake the signature of m_1' provided $h_1' < h_1$ where $h_1' = f(m_1')$ because it is easy to compute the hash chain in such direction (see bottom of Fig. 3).

To surpass such weakness we propose the entanglement algorithm which is based on the fact that in a hash

chain after signing a message always remains unused some points at the left of the computed values of the chain. This is so because $2^n - 1$ is the maximum number that could be allocated in the chain so $2^n - 1 > h_i$ (see Fig. 3).

The first step is compute the relation between the complement of h_1 that is $2^n - 1 - h_1$ over the length of the hash chain $2^n - 1$ then projecting it over $2^n - 1 - h_0$ as indicated in Eq. 1. We must demonstrate that in the presence of an attacker who is trying to fake h_1 with h_1' , the point that defines $\frac{2^n-1-h_1'}{2^n-1}(2^n-1-h_0)$ is always at the left of $\frac{2^n-1-h_1}{2^n-1}(2^n-1-h_0)$ in the hash chain of s_0 (see Fig. 3). Such condition is represented in Eq. 2. We must remark that δx_1 defines a unique point inside the hash chain according to hash principles (see Fig. 3). The steps indicated after Eq. 2 allow us to conclude that $h_1' < h_1$ which is always true as it was commented previously.

We introduce the chain ratio δx_1 in Eq. 3 that is computed as $h_1 - \frac{h_0 h_1}{2^n-1}$ (a rounding function must be applied to δx_1 if it were necessary), so any user who wants to verify the signature must apply Eq. 4. To be clearer, let us enumerate again the steps of the signature process:

1. Assume Alice has previously published $f^{2^n-1}(s_1)$. To sign the message m_1 Alice computes the chain ratio δx_1 to obtain $f^{2^n-1-h_0-\delta x_1}(s_0)$, then she publishes it along m_1 .
2. Using m_0 and m_1 Bob computes δx_1 and verifies that:

- $f^{\delta x_1}(f^{2^n-1-h_0-\delta x_1}(s_0)) = f^{2^n-1-h_0}(s_0)$ (remember that $f^{2^n-1-h_0}(s_0)$ has been previously published by Alice.
- $f^{h_1}(f^{2^n-1-h_1}(s_1)) = f^{2^n-1}(s_1)$.

$$\frac{2^n - 1 - h_1}{2^n - 1} (2^n - 1 - h_0) \quad (1)$$

$$\left(1 - \frac{h_1}{2^n - 1}\right) (2^n - 1 - h_0) = 2^n - 1 - h_0 - h_1 + \frac{h_0 h_1}{2^n - 1}$$

$$2^n - 1 - h_0 - h'_1 + \frac{h_0 h'_1}{2^n - 1} > 2^n - 1 - h_0 - h_1 + \frac{h_0 h_1}{2^n - 1} \quad (2)$$

$$-h'_1 + \frac{h_0 h'_1}{2^n - 1} > -h_1 + \frac{h_0 h_1}{2^n - 1}$$

$$h'_1 - \frac{h_0 h'_1}{2^n - 1} < h_1 - \frac{h_0 h_1}{2^n - 1}$$

$$h'_1 \left(1 - \frac{h_0}{2^n - 1}\right) < h_1 \left(1 - \frac{h_0}{2^n - 1}\right) \quad (3)$$

$$h'_1 < h_1$$

$$\delta x_1 = h_1 - \frac{h_0 h_1}{2^n - 1}$$

$$f^{\delta x_1}(f^{2^n-1-h_0-\delta x_1}(s_0)) = f^{2^n-1-h_0}(s_0) \quad (4)$$

An intruder who intercepts T_1 cannot forward to Bob another message say T'_1 signed as T_1 , because:

- As stated before, if $h'_1 > h_1$, Eve would try to forge the message m'_1 but this condition violates Eq. 2. This requires Eve to compute the hash function from $f^{2^n-1-h_0-\delta x_1}(s_0)$ to $f^{2^n-1-h_0-\delta x'_1}(s_0)$ (see Fig. 3). This operation implies reversing (several times) the hash function which it is assumed to be computationally infeasible.

The strategy described before can be generalized to multiple hash chains. For example, when $k = 4$ (Alice signs the first message h_0 using s_k , $k = 0, \dots, 3$). To sign the next message h_1 , Alice publishes $f^{2^n-1-h_1-\delta x_k}(s_k)$, $k = 4, \dots, 7$. Bob must compute δx_k and verify that $f^{\delta x_k}(f^{2^n-1-h_1-\delta x_k}(s_k)) = f^{2^n-1-h_1}(s_k)$, $k = 4, \dots, 7$ and so on. It must be clear that the computation process of $f^{\delta x_k}(f^{2^n-1-h_1-\delta x_k}(s_k))$ is much faster than $f^{h_1}(f^{2^n-1-h_1}(s_k)) = f^{2^n-1}(s_k)$ because $\delta x_1 = h_1 - \frac{h_0 h_1}{2^n - 1}$ is always small compared with h_1 (the maximum value for h_0 and h_1 is $2^n - 1$) (Fig. 4).

Timeline of Alice's signature process

Fig. 4 The signature publication process with δx security. Alice sends to the receiver the corresponding message m_i

3.1 Single-chain

Suppose Alice and Bob have recorded its signature history in a web service where both of them have been registered. In this context, we call single-chain to this user-chain history. Now, Bob needs to verify Alice's single-chain who wants to perform a digital transaction with him. To achieve this goal Bob must perform the next steps:

- Firstly, Bob must verify the hash of the first two messages signed by Alice $h_0 = f(m_0)$ and $h_1 = f(m_1)$.
- Then he computes $\delta x_1 = h_1 - \frac{h_0 h_1}{2^n - 1}$.
- Using $f^{2^n-1-h_1-\delta x_1}(s_0)$ Bob verifies that $f^{\delta x_1}(f^{2^n-1-h_1-\delta x_1}(s_0)) = f^{2^n-1-h_1}(s_0)$.
- Bob obtains the hash of the next message $h_2 = f(m_2)$.
- Using $f^{2^n-1-h_2-\delta x_2}(s_1)$ Bob verifies that $f^{\delta x_2}(f^{2^n-1-h_2-\delta x_2}(s_1)) = f^{2^n-1-h_2}(s_1)$.
- The verification process continues until Bob reaches the current signature executed by Alice.

The method described before prevents Bob to verify each Alice's digital signature of each chain through the computation $f^{h_i}(f^{2^n-1-h_i}(s_j)) = f^{2^n-1}(s_j)$ which requires a larger verification process. It is supported by the fact that in the

Fig. 5 Alice verifies $f^{\delta x_1}(f^{2^n-1-h_1-\delta x_1}(s_{j-1}))$ instead of $f^{h_1}(f^{2^n-1-h_1}(s_j))$ which requires a larger hash chain computation

Fig. 6 The order at which the information is published in the web service. In step 2, the web service substitutes $f^{2^n-1}(s_0)$ by $f^{2^n-1-h_0}(s_0)$

relation $\delta x_j = h_j - \frac{h_{j-1}h_j}{2^{n-1}}$, δx_j is derived from two values h_j and h_{j-1} . Thus, given h_j and h_{j-1} they produce a unique hash-sequence of the messages that have been previously signed by the user. We write it symbolically in Eq. 5 and it is represented in Fig. 5:

$$\delta x_1 \rightarrow h_0 h_1, \delta x_2 \rightarrow h_1 h_2 \dots \delta x_k \rightarrow h_{k-1} h_k \dots \delta x_n \rightarrow h_{n-1} h_n \quad (5)$$

Before analyzing the performance analysis in detail, we can introduce the following optimization of the storing space. The web service would store the hash of the message, the public-head hash $f^{2^n-1}(s_j)$, the signing hash $f^{2^n-1-h_i}(s_j)$ and the security hash $f^{2^n-1-h_i-\delta x_{i+1}}(s_j)$:

1. h_i
2. $f^{2^n-1}(s_j)$

3. $f^{2^n-1-h_i}(s_j)$
4. $f^{2^n-1-h_i-\delta x_{i+1}}(s_j)$

However, after a new message h_i is verified, then $f^{2^n-1-h_i}(s_j)$ could take the place of $f^{2^n-1}(s_j)$ to be stored in the web service, so we can keep just three hashes: the hash of the message, the signing hash and the security hash (see Fig. 6).

1. h_i
2. $f^{2^n-1-h_i}(s_j)$
3. $f^{2^n-1-h_i-\delta x_{i+1}}(s_j)$

To avoid an attack over $f^{2^n-1-h_i}(s_j)$ the hash of the signing message h_i is computed applying the hash function to the concatenation of the hash of the current message m_i and the hash of the previous one h_{i-1} . Symbolically, it is written in Eq. 6.

$$\begin{aligned} h_0 &= f(m_0), h_1 = f(h_0 || f(m_1)), \\ h_2 &= f(h_1 || f(m_2)), \dots h_i = f(h_{i-1} || f(m_i)) \end{aligned} \quad (6)$$

4 Performance analysis

As discussed before, using a 256-bit hash requires to compute the hash chain as $f^{2^{256}-1}(s)$ which could be and extremely heavy process. However, as commented earlier the hash of the message can be divided into k chains.

We will analyze the performance of the signature method for some k values at a hash rate of 15,840,454 hash/s according to the data published on the site bitcointalk.org. Let us call N to the size of the hash function used in the signature process. So we have $N = \sum_{i=1}^k \cdot n_i$. For example, if $N = 128$ and $k = 4$ they are generated four chains of 64 bits.

Tables 1, 2 and 3 show the size of the keys and the estimated time required to compute the hash chains for $k = \{2, 4, 8, 16, 32\}$ and $N = \{128, 160, 256\}$ at a hash speed of 10^7 hash/s. Finally, Table 4 summarize the best results in the current state of computer technology.

Table 1 The hash length is $N = 256$

$k : f^{2^n-1}(s_i)$ where $i = 1, \dots, k$	$k \cdot 256$ (bits)	Number of hashes	Required time (s)
2 : $f^{128}(s_1), f^{128}(s_2)$	512	$2^{128}-10^{38}$	$\sim 10^{31}$
4 : $f^{64}(s_1), f^{64}(s_2), \dots f^{64}(s_4)$	1024	$2^{64}-10^{18}$	$\sim 10^{11}$
8 : $f^{32}(s_1), f^{32}(s_2), \dots f^{32}(s_8)$	2048	$2^{32}-10^8$	$\sim 10^1$
16 : $f^{16}(s_1), f^{16}(s_2), \dots f^{16}(s_{16})$	4096	$2^{16}-10^4$	$\sim 10^{-3}$
32 : $f^8(s_1), f^8(s_2), \dots f^8(s_{32})$	8192	2^8-10^1	$\sim 10^{-6}$

It has been assumed a hash speed of 10^7 hash/s. The notation $f^{|e|}(s_i)$ means that the exponent $|e|$ is computed as 2^e . The size of the key is obtained as $k \cdot N$. The time has been derived according to the site bitcointalk.org with a processor AMD Phenom(tm) II X6 1055T, 6 cores, 2819.765 MHz, 15,840,454 hash/s

Table 2 $N = 160$ bits at a hash speed of 10^7 hash/s

$k : f^{2^n-1}(s_i)$ where $i = 1, \dots, k$	$k \cdot 160$ (bits)	Number of hashes	Required time (s)
2 : $f^{180}(s_1), f^{180}(s_2)$	320	$2^{80}-10^{23}$	$\sim 10^{16}$
4 : $f^{140}(s_1), f^{140}(s_2), \dots, f^{140}(s_4)$	640	$2^{40}-10^{11}$	$\sim 10^4$
8 : $f^{120}(s_1), f^{120}(s_2), \dots, f^{120}(s_8)$	1280	$2^{20}-10^6$	$\sim 10^{-1}$
16 : $f^{101}(s_1), f^{101}(s_2), \dots, f^{101}(s_{16})$	2560	$2^{10}-10^2$	$\sim 10^{-5}$

Table 3 $N = 128$ bits at a hash speed of 10^7 hash/s

$k : f^{2^n-1}(s_i)$ where $i = 1, \dots, k$ (bits)	$k \cdot 128$	Number of hashes	Required time (s)
2 : $f^{64}(s_1), f^{64}(s_2)$	256	$2^{64}-10^{18}$	$\sim 10^{11}$
4 : $f^{32}(s_1), f^{32}(s_2), \dots, f^{32}(s_4)$	512	$2^{32}-10^8$	$\sim 10^1$
8 : $f^{16}(s_1), f^{16}(s_2), \dots, f^{16}(s_8)$	1024	$2^{16}-10^4$	$\sim 10^{-3}$
16 : $f^{8}(s_1), f^{8}(s_2), \dots, f^{8}(s_{16})$	2048	2^8-10^1	$\sim 10^{-6}$
32 : $f^{4}(s_1), f^{4}(s_2), \dots, f^{4}(s_{32})$	4096	2^4-10^0	$\sim 10^{-7}$

Table 4 Some promising results are 128 bit-hash and 1024 bits key-size, 160 bit-hash with 1280 key-bit and 256 bit-hash with 4096 key-bit

$k : f^{2^n-1}(s_i)$ where $i = 1, \dots, k$	Hash function (bits)	Key size (bits)	Number of hashes	Required time (s)
8 : $f^{16}(s_1), f^{16}(s_2), \dots, f^{16}(s_8)$	128	1024	$2^{16}-10^4$	$\sim 10^{-3}$
8 : $f^{20}(s_1), f^{20}(s_2), \dots, f^{20}(s_8)$	160	1280	$2^{20}-10^6$	$\sim 10^{-1}$
8 : $f^{32}(s_1), f^{32}(s_2), \dots, f^{32}(s_8)$	256	2048	$2^{32}-10^8$	$\sim 10^1$
16 : $f^{16}(s_1), f^{16}(s_2), \dots, f^{16}(s_{16})$	256	4096	$2^{16}-10^4$	$\sim 10^{-3}$

The private key is formed by the seeds Alice uses to compute the chains. For example, for $k = 16$ we have the following seeds: $\{s_0, s_1, \dots, s_{15}\}$. A convenient method to reduce the size of the private key is represented in Eq. 7.

$$s_0, s_1, s_2 = f(s_0 || s_1), s_3 = f(s_1 || s_2), \dots \quad (7)$$

The size of the private key is reduced to the length of two hashes. If we are using SHA-256, then the size of the private key (to be stored in a mobile device) is fixed to 512 bits.

4.1 System implementation

Figure 7 shows the system architecture of the signature system. For this implementation we used SHA-256 and $k = 6$. The signature file occupies 32 hashes (8192 bits), which can be viewed according to the points in Fig. 6:

- 16 points marked with the number 3 in the chain starting with s_0 .
- 16 verification points marked with the number 3 in the chain of s_1 .

The Web file contains 32 hashes (8192 bits) plus the hash of the message:

Fig. 7 Digital system architecture for signing a file (upper) and the signature verification (low)

- 16 new head-chains, points marked with the number 3 in the chain of s_2 .
- 16 hashes, points marked with the number 2 in the chain of s_0 used by the web system to replace the head-chains (marked with the number 1 in the chain of s_0 in Fig. 6).

Table 5 Comparative analysis between some of the main blockchain-based protocols

Protocol	Data base	Signature (s)	Verification of the chain	Block size	Total size	Q
Bitcoin	Blockchain	10^{-3}	Minutes to hours	~ MB	242 GB	No
Ethereum		10^{-3}	10–20 s	~ kB	600 GB	No
Single-chain	Distributed	1.6–2.6	1–2 min	~ kB	–	Yes

The distributed data base of the single-chain protocol means that exists a data base by each user. The column Q, points out if the protocol survives in the quantum era

- The hash of the message written as hash File_1 in Fig. 7.

The user stores the seeds s_0 and s_1 (possibly in his mobile device) which are used by the signature system to generate the next seeds and the head-chains (public key). In this implementation, the time to generate the public head-chains goes from 1.1 to 1.3 s. Alice sends the file of the message (File_1) to Bob accompanied by its signature file. After Bob downloads the Web file, the signature verification is achieved successfully provided (see Fig. 6):

- He can go from the (16) points marked with the number 3 to the (16) points marked with the number 2 in the chain of s_1 .
- Bob can move from the (16) points marked with the number 3 to the (16) points marked with the number 2 in the chain of s_0 .

The verification of the signature file takes around 0.3 s in our implementation. However, verification of the single-chain varies according to its length, occupying few minutes to complete the verification.

4.2 Applications

Digital signatures based on hash chains, as the ones compared in Table 5, have a wide range of applications because transactions are highly traceable and verified before they are stored in an encrypted data base. Although exist some interesting applications for this kind of service one of the most relevant is the electronic payment service. Other potential uses include smart contracts, tracing physical or digital assets in the scope of the Internet of Things (IoT). Moreover, it could be properly enhanced other services as healthcare, music, government and identity.

To reduce the time to verify the single-chain of each user, it could be defined middle certificated points. It could be explored in a future implementation work.

5 Conclusions

Signature methods developed so far using hash functions are either ephemeral, as Lamport signature or they are not completely scalable as Merkle trees or Lizama hash signature. In this work, we have introduced a new approach based on hash functions and hash chains, which is capable to work over a public key infrastructure so that users can initialize remotely their hash-chain in its own computational device.

The signature method, which can exploited as a single block chain is based on the security principles of hash functions and our analyses suggests that the signature chains introduced here are scalable with today's hash technology. The method does not require other math function apart of hash function, therefore the algorithm is a promissory alternative for digital signatures in the quantum era.

Compliance with ethical standards

Conflict of interest The author declares that they have no competing interests.

References

1. Shor PW (1994) Algorithms for quantum computation: discrete logarithms and factoring. In: 35th Annual Symposium on Foundations of Computer Science, 1994 proceedings. IEEE, pp 124–134
2. Buchmann JA, Butin D, Göpfert F, Petzoldt A (2016) Post-quantum cryptography: state of the art. In: Ryan P, Naccache D, Quisquater JJ (eds) The new codebreakers. Springer, Berlin, pp 88–108
3. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21(2):120–126
4. FIPS P (2009) 186-3. Digital signature standard (DSS)

5. Rabin MO (1979) Digitalized signatures and public-key functions as intractable as factorization. Tech. rep., Massachusetts Inst of Tech Cambridge Lab for Computer Science
6. FIPS N (1998) 186-1. Digital Signature Standard
7. FIPS P (2000) 186-2. Digital signature standard (dss). National Institute of Standards and Technology (NIST)
8. Lamport L (1979) Constructing digital signatures from a one-way function. Tech. rep., Technical Report CSL-98, SRI International Palo Alto
9. Merkle RC (1982) Method of providing digital signatures. US Patent 4,309,569
10. Merkle RC (1987) A digital signature based on a conventional encryption function. In: *Advances in cryptology, CRYPTO 87*. Springer, pp 369–378
11. Merkle RC (1989) A certified digital signature. In: *Advances in cryptology, CRYPTO 89 proceedings*. Springer, pp 218–238
12. Adrián LPL, Javier MAL, Seleyda HMF, Adrián LSL, Eric S-A (2019) Public hash signature for mobile network devices. *Ing Investig Tecnol* 20(2):1–10
13. Perlner RA, Cooper DA (2009) Quantum resistant public key cryptography: a survey. In: *Proceedings of the 8th symposium on identity and trust on the internet*. ACM, pp 85–93
14. Nielsen MA, Chuang IL (2010) *Quantum computation and quantum information*. Cambridge University Press, Cambridge
15. Dattani NS, Bryans N (2014) Quantum factorization of 56153 with only 4 qubits. arXiv preprint [arXiv:1411.6758](https://arxiv.org/abs/1411.6758)
16. Dridi R, Alghassi H (2016) Prime factorization using quantum annealing and computational algebraic geometry. arXiv preprint [arXiv:1604.05796](https://arxiv.org/abs/1604.05796)
17. Bennett CH, Brassard G (1984) Quantum cryptography: public key distribution and coin tossing. In: *Int. conf. on computers, systems and signal processing, Bangalore*, pp 175–179
18. Lizama-Pérez LA, López JM, De Carlos-López E, Venegas-Andraca SE (2014) Quantum flows for secret key distribution in the presence of the photon number splitting attack. *Entropy* 16(6):3121–3135
19. Lizama-Pérez LA, López JM, De Carlos López E (2016) Quantum key distribution in the presence of the intercept-resend with faked states attack. *Entropy* 19(1):4

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.