Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8888679, 9 pages
https://doi.org/10.1155/2020/8888679

Research Article

An Authentication Scheme Based on Novel Construction of Hash Chains for Smart Mobile Devices

Qinglong Huang[1], **Haiping Huang**[1], **Wenming Wang**[1,2], **Qi Li**[1] and **Yuhan Wu**[1]

[1]*School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210003, China*
[2]*University Key Laboratory of Intelligent Perception and Computing of Anhui Province, Anqing Normal University, Anqing 246011, China*

Correspondence should be addressed to Haiping Huang; hhp@njupt.edu.cn

Received 26 June 2020; Revised 3 October 2020; Accepted 14 October 2020; Published 18 December 2020

Academic Editor: Ding Wang

With the increasing number of smart mobile devices, applications based on mobile network take an indispensable role in the Internet of Things. Due to the limited computing power and restricted storage capacity of mobile devices, it is very necessary to design a secure and lightweight authentication scheme for mobile devices. As a lightweight cryptographic primitive, the hash chain is widely used in various cryptographic protocols and one-time password systems. However, most of the existing research work focuses on solving its inherent limitations and deficiencies, while ignoring its security issues. We propose a novel construction of hash chain that consists of multiple different hash functions of different output lengths and employ it in a time-based one-time password (TOTP) system for mobile device authentication. The security foundation of our construction is that the order of the hash functions is confidential and the security analysis demonstrates that it is more secure than other constructions. Moreover, we discuss the degeneration of our construction and implement the scheme in a mobile device. The simulation experiments show that the attacker cannot increase the probability of guessing the order by eavesdropping on the invalid passwords.

1. Introduction

The Internet of Things (IoT) is becoming more and more closely connected with people's daily lives, due to the popularity of mobile devices which takes a central role in IoT. Users can use applications installed on mobile devices to obtain and transfer sensitive data, so the user authentication is a necessary process to ensure the security and privacy. For instance, SMS-based (Short Message Service) authentication is widely employed in many applications, such as Gmail. However, in the latest draft of the Digital Authentication Guideline, NIST (National Institute of Standards and Technology) has announced that the SMS-based authentication is deprecated and may no longer be allowed in the future releases of the guideline [1]. Furthermore, unlike traditional personal computers and laptops, mobile devices have limited energy, computing power, and storage capacity. Thus, it is not practical for the authentication schemes to employ expensive cryptographic primitives.

Hash chain used in Lamport's one-time password (OTP) [2] has become an important lightweight cryptographic primitive since it was proposed, which greatly improves the security of the simple password system. It also has been widely adopted to design key management and authentication mechanisms. The time-based one-time password (TOTP) system based on the hash chain is the most widely applied authentication system in which each password is valid only for a fixed time interval. TOTP system is more secure than SMS-based authentication, and the user can be authenticated implicitly. For example, some multifactor key management and authentication schemes [3–5] adopt the hash chain to ensure forward security, because the security foundation of the hash chain is guaranteed by its unidirectional nature. Hash chain is also employed by some broadcast authentication protocols [6] as one of the building blocks. And in many countries, the TOTP system has been chosen as a major security component of Internet banking to authenticate account holders. Thus, the TOTP system can be an alternative to replace SMS-based authentication [7, 8].

Hash chain is a very important tool to design the TOTP system. However, the hash chain still has some limitations and disadvantages that are issued by many literatures. First, one hash chain can only perform one-way authentication for two parties. The mutual authentication implemented by the hash chain requires both parties to store a secret value that is known as "seed," which cannot be implemented or may cause huge security issues in many scenarios. Second, due to the limited length of the hash chain, a new hash chain must be generated and both parties have to reregister when the last hash chain has been consumed. Park [9] proposed an endless hash chain composed of many short chains in which each authentication message contains the commitment of the next short hash chain. The studies of [8, 9] designed a multilevel hash chain to avoid its exhaustion. And [10–15] presented a self-renewal hash chain in which a new hash chain will be established if the old one is consumed. Third, the computational burden is much higher when the sender generates a new one-time password in which multiple hash operations are required. [16–18] proposed construction methods to store and recover an intermediate value of a hash chain. [18, 19] further discussed the optimal time-memory tradeoff for the traversal of sequence hash chain. Hu et al. [10] gave two construction methods enabling faster verification.

Although most of the existing researches have addressed the issues caused by the above limitations and disadvantages, there is little related literature on the security research of the hash chain itself. Given the complex mobile environment and the open nature of channels (wireless channels) connected to mobile devices, the Lamport's hash chain is easier to be inverted than a single hash function if the attacker can eavesdrop on lots of values of hash chains. Thus, Kogan et al. [20] improves the safety of Lamport's hash chain by domain separation.

1.1. Our Work. In this paper, we introduce a novel construction of the hash chain and design a TOTP scheme for authentication of mobile devices. In our scheme, the hash chain is composed of k different hash functions with different output length. Compared to Lamport's hash chain, our construction can address the issue that the invalid password may help the attacker to invert the hash chain. Therefore, keeping the order of these hash functions confidential can effectively prevent the attacker from inverting the hash chain. And meanwhile, this design presents a challenge whether an attacker can easily find the order, which will come through a simulation. Besides, our scheme can be easily adopted by multifactor authentication scheme because of the simplified structure, especially adaptive to mobile devices.

The remainder of this paper is organized as follows. Section 2 illustrates the overview of hash chains. Our construction of hash chain and a TOTP system is given in Section 3. In Section 4, we analyse the security of our construction. We discuss the degeneration of our construction and run simulation experiments in Section 5. Section 6 concludes this paper.

2. Overview of Hash Chains

In this section, we briefly review the Lanport's and Kogan et al.'s hash chains and their security.

2.1. Lamport's Hash Chain. The hash chain $\{x_i = h(x_{i-1}), i = 1, \cdots, N\}$ proposed by Lamport is the continuous iteration of a secret value x_0 with the same hash function, as shown in Figure 1.

The secret value is generated by the user and the root of the hash chain x_N, which is called commitment, should be registered on the server in advance. Whenever the server receives a password x_i from the user, the server verifies whether it equals the commitment x_{i+1} by a hash operation, namely, $h(x_i)? = x_{i+1}$. If the verification is passed, the server stores x_i as a new commitment for the next authentication.

A classic attack on the hash function is the birthday attack. The Lamport's hash chain is generated by iterating the same hash function, so birthday attack is still an effective attack to the hash chain. Hu et al. [10] has discussed the susceptibility of iterating the hash function to birthday attack. Furthermore, Håstad and Näslund [21] got a conclusion that inverting the k-th iteration is k times easier than a single hash function if the same hash function is used in each step of the hash chain. Nevertheless, this research still works on the hash chain based on the same hash function.

2.2. Kogan et al.'s Construction. Kogan et al. [20] proposed a TOTP system called T/Key that is designed as a second-factor authentication scheme, which can be used in mobile devices. The hash chain used in T/Key is composed of independent hash functions h_k in every step, as shown in Figure 2.

These hash functions $h_i : \{0,1\}^n \longrightarrow \{0,1\}^n$, $(i = 1, 2, \cdots, N)$ are obtained from a certain hash function $H : \{0,1\}^m \longrightarrow \{0,1\}^m$ by domain separation.

$$h_i(x) = H(<t_{\text{init}} + n - i>_c \mid |id| \mid x)|_n, \tag{1}$$

where t_{init} is the initialization time and $id \in \{0,1\}^s$ is a random salt. For a numeral t, $<t>_c$ denotes the c-bit binary representation of t, and $t|_n$ denotes the n-bit prefix of t. And these functions have the same domain so that it can withstand length extension attack.

In T/Key, each password is valid for a specific time interval I. The user computes

$$x_N = h_N(h_{N-1}(\cdots h_1(x_0) \cdots)), \tag{2}$$

at first and sends it along with random salt id to the server as a commitment. At a later time t, the user sends a password x_i to the server. When the server receives it, the server verifies whether it equals the stored commitment x_{i+j} by j hash operations, namely,

$$h_{i+j}(\cdots h_{i-1}(x_i) \cdots)? = x_{i+j}. \tag{3}$$

If the verification passes, the server stores the x_i as a new commitment for the next authentication.

Kogan et al. also demonstrate that the difficulty of inverting this construction is almost the same as inverting a single hash function. However, as the length of the hash chain increases, a larger domain of H is integrant.

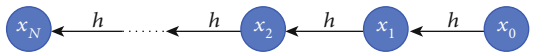

Figure 1: Lamport's hash chain.

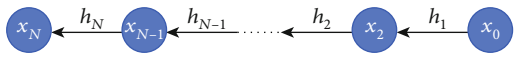

Figure 2: The structure of T/Key's hash chain. $x_0 \in \{0,1\}^n$ is a uniformly random secret key.

3. Our Construction

The hash chain used in T/Key has a limitation that the domain should be larger as the length of the hash chain increases, and the independent hash functions are actually generated by the same hash function. Therefore, the basic ideal of our construction is that multiple hash functions, e.g., SHA-2 and MD5, are selected to build the hash chain, and these hash functions have different output length. The order of these hash functions ord is confidential. We denote these hash functions as $h_0, \cdots, h_{k-1}$, and their output lengths are $n_i\,(i = 0, \cdots, k-1)$. For example, if two hash functions $(k = 2)$, SHA-2 and MD5, have been chosen, there are two orders, ord $= \{h_0 = \text{SHA} - 2, h_1 = \text{MD5}\}$ or ord $= \{h_0 = \text{MD } 5, h_1 = \text{SHA} - 2\}$. These hash functions are used cyclically in the hash chain, namely,

$$H(x) = h_{k-1}^{m}\left(\cdots h_{k-1}^{2k}\left(\cdots h_0^{k+1}\left(h_{k-1}^{k}\left(\cdots h_1^{2}\left(h_0^{1}(x)\right)\cdots\right)\right)\cdots\right)\cdots\right), \tag{4}$$

where h_i^j is one of the k hash functions, and $i = (j-1) \mod k$ represents the index of hash function $h_i\,(i = 0, \cdots, k-1)$ used in j-th iteration and m is the hash chain length. Since the hash function used in j-th iteration can be inferred from its index, h_i^j will be simplified as h^j, and $h^{[a,b]}\,(b > a)$ is equivalent to $h^b \circ h^{b-1} \circ \cdots \circ h^a$. The secret information $x = \text{sk}\|\text{ord}$ consists of two parts: the secret key sk and the order of these hash functions ord.

The hash chain we constructed is used as a time-based one-time password scheme to authenticate the mobile device where each password is valid for a short time interval. The scheme consists of only two phases so that it can be easily integrated into many multifactor authentication schemes or used as a separate auxiliary authentication method for mobile devices. Since that, a hash chain can only be used for authentication by two parties; the roles in our scheme are simplified to two. The party requesting authentication is a mobile device M (or a mobile application), and the verifier who verifies the password sent by a mobile device or application is a server S. Figure 3 shows the design of our proposal.

3.1. Setup. The mobile device M selects k hash functions $h_0, \cdots, h_{k-1}$ with different output length n_i and then chooses and stores the order ord, the hash chain length m, and a random secret key sk. The mobile device M (or the sever S) notes the time interval I (in seconds), which represents the valid time of each password and the initial time of the hash chain t_{init} (measured in I). The public parameters $(h_0, \cdots, h_{k-1}, n_0, \cdots, n_{k-1}, I, t_{\text{init}}, m)$ can be sent to both two parties through an insecure channel. The order of hash functions ord should be sent to the server in a secure channel.

Moreover, M computes

$$\text{mes}_{\text{root}} = h^{[1,m]}(x), \tag{5}$$

and sends it to the server S. Then, S stores it as mes_v for the next verification and records t_{init} as t_v.

3.2. Authentication. At some time $t > t_{\text{init}}$, M wants to access the server. The mobile device M and the server S proceed as follows:

(i) M generates the password mes_t using the secret key sk and the order ord. And M sends it to the server

$$\text{mes}_t = h^{[1,\,m-t]}(x) \tag{6}$$

Particularly, if $t = m$, then $\text{mes}_t = x$.

(ii) Upon receiving the mes_t, S firstly checks whether the password $\text{len}(\text{mes}_t)$ is valid. If $\text{len}(\text{mes}_t) = n_{(m-t-1) \mod k}$, S accepts it, otherwise, refuses it

(iii) S then verifies the password according to ord and computes

$$\text{mes}_t' = h^{[m-t+1,\,m-t_v]}(\text{mes}_t) \tag{7}$$

(iv) If $\text{mes}_t' = \text{mes}_v$, then S sets $\text{mes}_v = \text{mes}_t$ and $t_v = t$, and the authentication successes, otherwise, the authentication fails

3.3. Clock Synchronization. In the TOTP system, a synchronized clock is necessary to ensure the authentication process. However, time skew or network natural delay is unavoidable, which may cause authentication failure. If time skew can be quickly repaired, then no additional mechanism is needed, as this only causes one authentication failure when it happens. Otherwise, when the time skew or network natural delay continues, a solution is that each password is valid for serval time intervals (related to I), instead of only valid for a time interval. In this case, the server S needs to verify the password in each valid time, and t_v should be updated to the time of successful verification.

4. Security Analysis

4.1. Forward Security. Our protocol can provide forward security, which means that if one key of the hash chain mes_t is leaked at a certain time, the previous authentication process will not be affected. In our construction, the order of hash functions is securely stored in both mobile devices and servers. The adversary has no way to compute the key

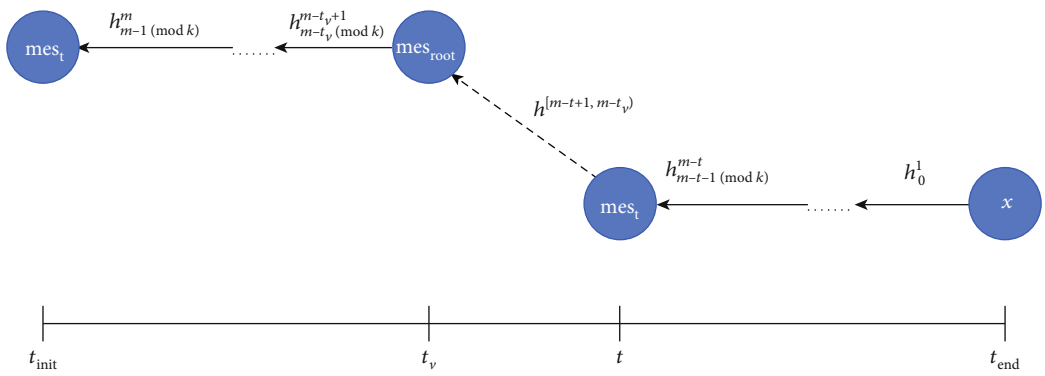

FIGURE 3: The design of our proposal. $x = \mathrm{sk}\|\mathrm{ord}$ is the secret information and t_{end} is the moment that the hash chain is consumed.

mes_i $(i < t)$. In the worse situation where ord is no longer a secret, the mes_i computed by the adversary is expired, so the adversary cannot pass the verification. Thus, our protocol achieves the forward security.

4.2. A Lower Bound of Inverting Hash Chain. Our construction is hard to invert because it is composed of multiple different hash functions that have different lengths. However, this is vulnerable to length extension attacks, especially when some hash functions are MD-based hash functions. To prevent this kind of attack, the server has to verify the length of the password first after receiving it.

The next crucial question is how difficult is it to invert our construction in our TOTP system. For clarity and completeness, we review some key theories before analysing the security of our scheme.

4.2.1. Lamport's Construction. In the Lamport's hash function, the attacker can utilize invalid passwords which have been authenticated in the past to invert the hash chain. As more and more passwords are collected, attackers will have a greater chance of obtaining a legitimate preimage by oracle query.

Furthermore, Håstad and Näslund show that, in a hash chain composed of the same hash function, the adversary inverts the k-th iteration is actually k times easier than inverting a single hash function. The representation is rearranged here for completeness and better understanding.

Theorem 1 (Inverting the original hash chain) (see [21]). *Let A be an algorithm that tries to invert a hash function $h : \{0,1\}^n \longrightarrow \{0,1\}^n$ which makes T oracle queries at most. Given $y = f^{(k)}(x)$ for a randomly chosen x, then*

$$\Pr\left[h(A(y)) = y\right] = \Omega\left(\frac{Tk}{2^n}\right). \tag{8}$$

Moreover, A succeeds with the probability at most $O(Tk/2^n)$

.

4.2.2. T/Key's Construction. Kogan et al. give a further analysis of the inverting hash chain which uses k independent hash functions defined by a certain hash function H through *domain separation*, which shows as Theorem 2.

Theorem 2 (Inverting the T/Key's hash chain) (see [20]). *Let functions $h_1, \cdots, h_k : \{0,1\}^n \longrightarrow \{0,1\}^n$ be chosen independently and uniformly at random. Let A be an algorithm that can get oracle queries to all of the functions $h_1, \cdots, h_k$ which makes T oracle queries at most overall. Thus,*

$$\Pr\left[h_k\left(A\left(h_{[1,k]}(x_0)\right)\right) = h_{[1,k]}(x_0)\right] \le \frac{2T+3}{2^n}, \tag{9}$$

where $h_{[1,k]} = h_k \circ h_{k-1} \circ \cdots \circ h_1$.

Proof. We briefly prove this theorem which will prepare for the following proof of other theorems. Let $P = (p_0, p_1, \cdots, p_k)$ be the values of the hash chain, i.e., $p_0 = x$, $p_i = h_i(p_{i-1})$, and $i = 1, \cdots, k$. The algorithm A also needs to maintain a list $L = \{(i, x, y), h_i(x) = y\}$, which records the oracle queries x and their answers y. For any query (i, x), if $x = p_{i-1}$, A responds with $y = p_i$ and adds (i, p_{i-1}, p_i) to the list. Else if $(i, x) \in L$, A replies with y. Otherwise, A chooses $y = \{0,1\}^n$ randomly. Thus, to invert the hash chain, at least one query result should collide with P. It follows that

$$\begin{aligned}\Pr_{H,x_0}[A \text{ loses}] &= \Pr_{H,x_0}\left[\bigcap_{j=1}^{T+1} y_j \neq p_{i_j}\right] = \prod_{j=1}^{T+1} \Pr_{H,x_0} \\ &\quad \cdot \left[y_j \neq p_{i_j} \mid \bigcap_{l=1}^{j-1} y_l \neq p_{i_l}\right] \\ &\ge \prod_{j=1}^{T+1}\left(1 - \frac{2}{2^n - j + 1}\right) \\ &\ge \frac{2^{2n} - (2T+3)2^n}{2^{2n}}.\end{aligned} \tag{10}$$

Therefore,

$$\Pr_{H,x_0}[A \text{ wins}] \le \frac{2T+3}{2^n}. \tag{11}$$

Theorem 2 demonstrates that inverting a hash chain using independent hash function results in a loss of security by a factor of 2, but by a factor of $O(k)$ if the same hash function is used in the entire chain. Inverting this construction is as hard as a single hash function. Moreover, in the proof, algorithm A is designed to get oracle access to all the functions $h_1, \cdots, h_k$. In fact, these functions have the same domain as well as the function H, which means that algorithm A actually only gets oracle access to one hash function H.

4.2.3. Our Construction. In our proposal, the hash chain is composed of k different hash functions, and the attacker is hard to invert the chain due to the secrecy of the order of these functions. For an attacker who does not know ord, the entire hash chain can be regarded as consisting of multiple independent hash functions.

When the attacker wants to invert the chain, he has two choices. The attacker either "guesses" a hash function that may be the right one and queries (see Theorem 3), or averages T oracle queries to all hash functions (see Theorem 4). These two choices are both analysed.

Theorem 3. *Let functions $h_0, \cdots, h_{k-1} : \{0,1\}^* \longrightarrow \{0,1\}^{n_i}$ be different hash functions with distinct output length n_i. Let A be an algorithm that can get oracle queries to a certain hash function which can make T oracle queries at most overall. Thus,*

$$\Pr\left[h^m\left(A\left(h^{[1,m]}(x)\right)\right) = h^{[1,m]}(x)\right] \leq \frac{2T+3}{k2^{n_{(m-1) \bmod k}}}. \quad (12)$$

Proof. We say the attacker successfully inverts the hash chain when he finds a preimage that meets the length requirement. Let A be an algorithm as in the statement of Theorem 2, and we used it to construct another algorithm A_1 for finding the preimage of the last iteration. The first step of A_1 is to select a hash function H which is used in h^{m-1} and

$$\Pr\left[H = h^{m-1}\right] = \frac{1}{k}, \quad (13)$$

$$\Pr\left[H \neq h^{m-1}\right] = \frac{k-1}{k}. \quad (14)$$

Then, algorithm A_1 runs algorithm A to get oracle access to hash function H and query T times at most and inputs the result to h^m. It follows that

$$\begin{aligned}\Pr_{H,\ x}[A_1 \text{ loses}] &= \Pr_{H,\ x}\left[H \neq h^{m-1}\right] + \Pr_{H,\ x}\left[\bigcap_{j=1}^{T+1} y_j \neq p_{i_j}, H = h^{m-1}\right] \\ &= \frac{k-1}{k} + \Pr_{H,\ x}\left[H \neq h^{m-1}\right] \bullet \Pr_{H,\ x} \\ &\quad \cdot \left[\bigcap_{j=1}^{T+1} y_j \neq p_{i_j} \mid H = h^{m-1}\right].\end{aligned} \quad (15)$$

According to Theorem 2, we know

$$\Pr_{H,\ x}\left[\bigcap_{j=1}^{T+1} y_j \neq p_{i_j} \mid H = h^{m-1}\right] \geq \frac{2^{2n_{(m-1) \bmod k}} - (2T+3) \bullet 2^{2n_{(m-1) \bmod k}}}{2^{2n_{(m-1) \bmod k}}}. \quad (16)$$

Overall,

$$\Pr_{H,\ x}[A_1 \text{ loses}] \geq \frac{n_{(m-1) \bmod k}{}^2 - (2T+3) \bullet 2^{n_{(m-1) \bmod k}}}{2^{2n_{(m-1) \bmod k}}}. \quad (17)$$

Therefore,

$$\Pr_{H,\ x}[A_1 \text{ wins}] \leq \frac{2T+3}{k2^{n_{(m-1) \bmod k}}}. \quad (18)$$

Theorem 4. *Let functions $h_0, \cdots, h_{k-1} : \{0,1\}^* \longrightarrow \{0,1\}^{n_i}$ be different hash functions with distinct output length n_i. Let A be an algorithm that can get oracle queries to all of the functions $h_0, \cdots, h_{k-1}$ which makes T oracle queries at most overall. Thus,*

$$\Pr\left[h^m\left(A\left(h^{[1,m]}(x)\right)\right) = h^{[1,m]}(x)\right] \leq \frac{2T+3k}{k^2 2^{n_{(m-1) \bmod k}}}. \quad (19)$$

Proof. Let A be an algorithm as in the statement of Theorem 2. We use it to construct another algorithm A_2 which finds the preimage of the last iteration of the hash chain. The algorithm A_2 runs algorithm A first, which queries T times at most to all k hash functions, scilicet T/k oracle queries for each function. The legal preimage could be got only from the query results which returned by the oracle access to h^{m-1}, and the algorithm A makes T/k oracle queries to it.

Therefore, according to Theorem 3,

$$\Pr_{H,\ x}[A_2 \text{wins}] \leq \frac{(2T/k)+3}{k2^{n_{(m-1) \bmod k}}} = \frac{2T+3k}{k^2 2^{n_{(m-1) \bmod k}}}. \quad (20)$$

The above two theorems establish the difficulty of finding a preimage of the last iteration of the hash chain in the authentication scheme. And the difficulty of inverting the hash chain which is composed of k different hash functions with different output length is further reduced to $O(T/k2^n)$, where n is the average length of these functions.

4.3. Formal Security Analysis. In the following, we show our protocol is provably secure in the random oracle model since the hash function behaves closely like a random oracle [22–24]. We first present a formal description of the proposed protocol before defining the security game, which is a tuple $\mathcal{P} = (\mathcal{G}, \mathcal{K}, \mathcal{P}, \mathcal{V})$.

In the setup phase, the polynomial time algorithm $\mathcal{G}(k, m) \longrightarrow (n_i)$ takes as input the number of hash functions k and the length of hash chain m and outputs the password length n_i. Next, the algorithm $\mathcal{K}(n_i, m) \longrightarrow (sm, vs)$ takes as input the password length n_i and hash chain length m and outputs the secret message sm and the verifier state vs. In the authentication phase, the prover $\mathcal{P}(sm, t) \longrightarrow pwd$

outputs a one-time password pwd by taking as input the secret message sm and a time t. While the verifier $v(vs, pwd, t) \longrightarrow$ (ACCEPT/REJECT, vs') takes as input the verifier state vs, one-time password pwd and a time t outputs a state that the password is accepted or rejected and a new verifier state vs'. Afterwards, we are ready to define the attack game.

Attack Game 5. Let $\mathscr{P}$ be a one-time password protocol and let $\mathscr{R}$ be a random oracle. Given a challenger $\mathscr{C}$ and an adversary $\mathscr{A}$, the attack game acts as follows:

(i) *Setup*. The challenger generates password lengths $(n_i) \longleftarrow \mathcal{G}_{\mathscr{C}}(k, m)$

(ii) *KeyGen*. The challenger generates $(sm, vs) \longleftarrow \mathcal{K}_{\mathscr{C}}(n_i, m)$ by random oracle

(iii) *Password Query*. The adversary sends a time t to the challenger. The challenger generates a password $pwd \longleftarrow \mathcal{P}_{\mathscr{C}}(sm, t)$, which is fed to the verifier (ACCEPT/REJECT, vs') $\longleftarrow v_{\mathscr{C}}(vs, pwd, t)$

(iv) *Test*. The adversary submits a password $(t_{\mathscr{A}}, pwd_{\mathscr{A}})$

The attacker will win the game if the verifier output ACCEPT by $v_{\mathscr{C}}(vs, pwd, t)$. We denote $\mathrm{Adv}_{\mathscr{P}}^{\mathrm{Test}}(\mathscr{A})$ as the probability that $\mathscr{A}$ successfully impersonates as a legal user in the execution of protocol $\mathscr{P}$.

Theorem 6. *Let $\mathscr{P}$ be the proposed protocol in the Section 3. Let A be a probabilistic polynomial-time adversary attacking the protocol that makes at most T oracle queries with the length m and the number of hash function k.*

$$Adv_{\mathscr{P}}^{Test} \leq \frac{2T + 2m + 3}{k2^{n_{(m-1) \bmod k}}}. \tag{21}$$

Proof. Let A be an algorithm stated before. We, using algorithm A, construct algorithm A_3 that makes the oracle query of the last iteration of the hash chain. The adversary wins the game if and only if the verifier outputs ACCEPT by taking as input the password $(t_{\mathscr{A}}, pwd_{\mathscr{A}})$, i.e., $h^{[m-t_{\mathscr{A}}+1, k]}(pwd_{\mathscr{A}}) = vs$. Without loss of generality, we assume that the verifier state $vs = h^{[1,m]}(sm)$. With the input $(n_i) \longleftarrow \mathcal{G}_{\mathscr{C}}(k, m)$, the algorithm A_3 runs A to get a point y and computes $y' = h^{[1,m-1]}(y)$. If $h^m(y') = h^{[1,m]}(sm)$, then $h^{[1,m]}(y) = h^{[1,m]}(sm)$. Thus, the algorithm makes at most $T + m - 1$ oracle queries. According to Theorem 3,

$$\mathrm{Adv}_{\mathscr{P}}^{\mathrm{Test}} \leq \frac{2(T + m - 1) + 3}{k2^{n_{(m-1) \bmod k}}} \leq \frac{2T + 2m + 1}{k2^{n_{(m-1) \bmod k}}}. \tag{22}$$

5. Discussions and Experiment

5.1. Degeneration. The hash chain constructed by us consists of k different hash functions with different output length that can be freely selected by the mobile device or the server. This freedom of choice has caused more changes in our construction that needs to be discussed.

First and foremost, as shown in security analysis, the length of the hash chain is no longer a key factor affecting the difficulty of inverting the hash chain, but the quantity of hash function selected is a key factor. Obviously, the larger the value of k, the more difficult to invert the hash chain. Furthermore, when there is only one hash function $(k = 1)$ in the hash chain, our construction will degenerate to Lamport's hash chain.

Second, the quantity of hash function with different output length is not unlimited. So, what happens if there are several functions that have the same output length, or the same hash function is used twice or more in the hash chain? At a macro level, it would be easier for an attacker to forge a password with this length because there is a greater probability of "guessing" the preimage correctly. If all the hash functions used in the hash chain have the same output length but the hash functions are different, it will degenerate into a special instance of T/Key's construction. The chain is equivalent to connecting T/Key's construction multiple times, and they have a similar difficulty for inverting the hash chain.

Last but most importantly, the security of our construction is guaranteed by the confidentiality of the order of these hash functions ord. If the ord is leaked, the difficulty of breaking our solution will be reduced to $O(T/n)$ which is the same as the difficulty of inverting a single hash function, because the invalid password only provides a limited effect to the attacker. In theory, the probability that an attacker finds the order without any prior knowledge is $1/k!$. However, in practice, the mobile device (or application) authenticates to the server as a Poisson process. We will show how difficult it is for an attacker to guess ord if the attacker can eavesdrop on the authentication message through simulation in the next subsection.

5.2. Experimental Evaluation. As we analysed above, in our scheme, the order of hash functions ord is one part of the secret information x. Once it is leaked, the security of our mechanism will be greatly affected. In this section, we run a simulation to compute the difficulty of the attacker finding ord and then implement our construction in a real smart mobile device.

Before starting the simulation, two crucial issues, the login pattern of the mobile device and the attacker' behaviour, have to be discussed.

In the real world, it is reasonable that the login behaviour of a mobile device (or application) follows the Poisson Process, which means that the time interval between consecutive logins can be modelled using the exponential distribution. Thus, the probability density function that the next authentication of mobile device at time t is

$$p(t) = \lambda e^{-\lambda t}, \tag{23}$$

where λ is the average login rate.

Then, we need to know how the attacker guesses the ord which actually has $k!$ possible permutations ord_i, $i = 1, 2, \cdots, k!$. The authentication message intercepted by the attacker can help him/her to guess ord. Since the lengths of these authentication messages are totally random, it is

hard for an attacker to guess ord based on the context between them. But, for $\text{ord} = h_0, \cdots, h_{k-1}$, in the sequence of attacker owned, h_j only appear after $h_i\,(j < i)$ before h_0 appears. Thus, the attacker can count the number that ord_i appears through the sequence. We define that the attacker finds the real ord by the probability

$$\Pr\,[\text{attacker wins}] = \frac{\text{num}(ord)}{\sum_i ord_i}. \tag{24}$$

Figure 4 shows how the probability of an attacker's guesses the ord changes over time (per month). If the attacker can eavesdrop on every authentication message, as the number of chain value held by the attacker increases, the probability of the attacker's success decreases. This is because when the attacker has fewer chain values, the higher probability of correct ord appearing makes the attacker more likely to succeed.

The conclusion still holds when we change the eavesdropping probability of the attacker. Figure 5 compares the probability that the attacker finds ord in the following eavesdropping probability: 0.2, 0.5, and 0.8, respectively. When $p \leq 0.5$, the probability peaks at third and fifth months, respectively. Meanwhile, due to the increasing number of hash chain values captured by attackers, the probability of attackers' success is still decreasing. According to the law of large numbers, the probability that the attacker observes the correct ord approaches the theoretical value when the attacker has a large amount of hash chain values.

Theoretically, the probability that an attacker finds the correct ord is $1/4! = 1/24 \approx 0.0417$. While in our simulation, even if the attacker can intercept every authentication message, the probability does not exceed 0.0400. Therefore, an attacker eavesdropping on a channel does not increase the probability of finding the correct ord.

The experiment environment is set up as follows: The smart device is a HUAWEI Mate 30 smartphone with 2.86 GHz CPU and 6 GB RAM running Android 10, and the server is executed on a 2-core CPU and 1024 MB memory running Ubuntu 18.04.

We use 4 hash functions to instantiate our scheme discussed in Section 3, which are MD5, SHA-1, SHA-2, and BLACK2b, and the output length is 128 bits, 160 bits, 256 bits, and 512 bits, respectively. The time interval I when each password is valid uses time slots of 30 seconds. We generate a hash chain with the length 1.05×10^6 that would be valid in one year. We assume that mobile device login once a day on average ($\lambda = 1/86400$). Furthermore, an attacker eavesdrops on authentication message with a certain probability p. These parameters used in simulation experiments are shown in Table 1.

As a comparison, we also implement Lamport's hash chain with 4 kinds of hash functions, respectively. We evaluate the following time: mobile device setup time, average password generation time (mobile device), and average verification time (server). Table 2 shows the results.

As shown in Table 2, the MD5 and SHA-1 hash functions have better performance, while it is not a good choice to construct the Lamport's hash chain because both of them are not

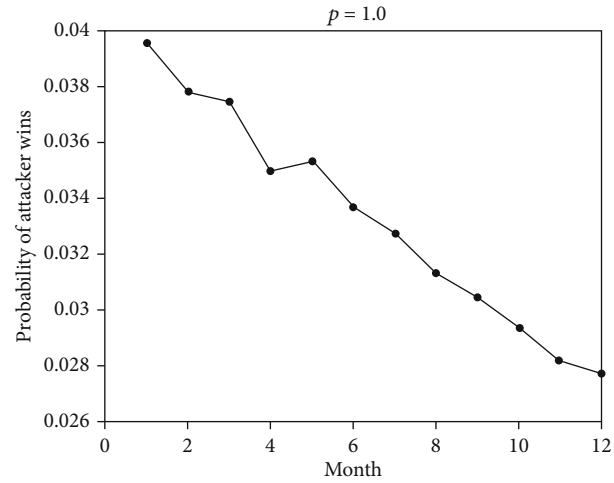

Figure 4: Probability of attacker wins when $p = 1$.

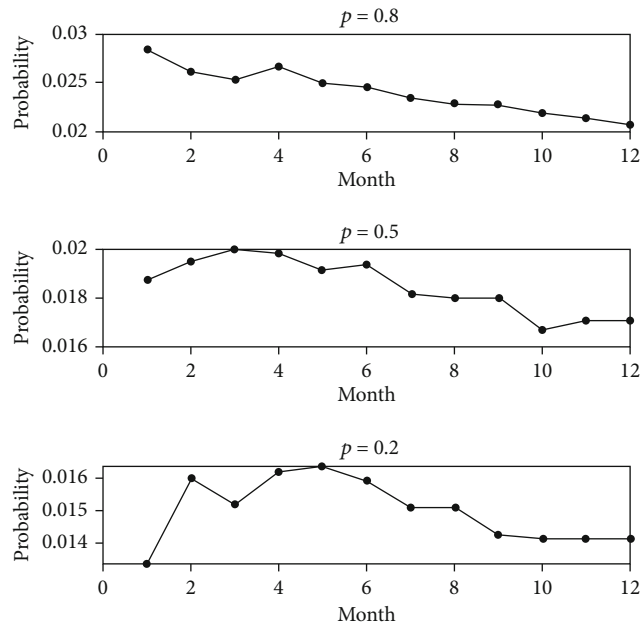

Figure 5: Probability of attacker wins when $p = 0.2, 0.5, 0.8$.

secure enough [20, 21]. Our solution has the best computational performance on a mobile device while ensuring security. And the server only takes several milliseconds to verify the password which is acceptable in general. More experimental results can be obtained through online resources (https://github.com/qinglong-huang/hash_chains_experiments).

Both theoretical analyses in Section 4 and simulation experiments that we performed demonstrate that the hash chain scheme proposed in this paper is still harder to invert. Therefore, our scheme has better performance on computation and security.

6. Conclusion

In this paper, a novel construction of hash chain was presented, and a TOTP system based on this construction was

Table 1: Experiment parameters.

Parameter	Value	Description
λ	1/86400	Average authentication rate
m	1.05×10^6	Length of hash chain
I	30 sec	Interval time
k	4	Number of hash functions
p	[0, 1]	The probability of an attacker eavesdrops on passwords

Table 2: Experiment result.

	Setup time (seconds)	Average password generation time (seconds)	Average verification time (milliseconds)
MD5	0.96	0.51	1.30
SHA-1	2.21	1.08	2.40
SHA-2	8.34	3.36	9.02
BLAKE2b	8.72	3.59	4.80
Ours	7.74	2.99	5.1

designed for mobile device authentication. This system could be easily employed by some lightweight authentication schemes to ensure forward security or be applied as a second-factor authentication method replacing SMS-based authentication for mobile devices. We gave a formal security analysis regarding the difficulty of inverting the hash chain and demonstrate that the attacker inverting the hash chain in T queries is $O(T/kn)$ at most. Besides, we discussed several situations that may reduce its security when the selection of hash function changes. Finally, we implemented the scheme on a smartphone, and the simulation result showed that even if an attacker can eavesdrop on every password; the probability that she/he uses these invalid passwords to guess ord successfully is not higher than the theoretical value. Therefore, our scheme met the higher security requirement in mobile device authentication.

Data Availability

Data available is on request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is sponsored by the National Natural Science Foundations of China (Grant Nos. 61672297 and 62072252), the Key Research and Development Program of Jiangsu Province (Social Development Program, No. BE2017742), the Postgraduate Research & Practice Innovation Program of Jiangsu Province (Grant No. KYCX19_0908), and the Key Project on Anhui Provincial Natural Science Study by Colleges and Universities (Grant Nos. KJ2019A0579 and KJ2019A0554).

References

[1] https://pages.nist.gov/800-63-3/sp800-63b.html#out-of-band.

[2] L. Lamport, "Password authentication with insecure communication," *Communications of the ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[3] M. Shuai, B. Liu, N. Yu, and L. Xiong, "Lightweight and secure three-factor authentication scheme for remote patient monitoring using on-body wireless networks," *Security and Communication Networks*, vol. 2019, Article ID 8145087, 14 pages, 2019.

[4] M. Alshahrani and I. Traore, "Secure mutual authentication and automated access control for IoT smart home using cumulative keyed-hash chain," *Journal of Information Security and Applications*, vol. 45, pp. 156–175, 2019.

[5] L. Xiong, N. Xiong, C. Wang, X. Yu, and M. Shuai, "An efficient lightweight authentication scheme with adaptive resilience of asynchronization attacks for wireless sensor networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–13, 2020.

[6] A. Perrig, R. Szewczyk, J. D. Tygar, V. Wen, and D. E. Culler, "SPINS: security protocols for sensor networks," *Wireless Networks*, vol. 8, no. 5, pp. 521–534, 2002.

[7] H. Sun, K. Sun, Y. Wang, and J. Jing, "TrustOTP: transforming smartphones into secure one-time password tokens," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, pp. 976–988, Denver, CO, USA, 2015.

[8] E. Erdem and M. T. Sandikkaya, "OTPaaS-one time password as a service," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, pp. 743–756, 2019.

[9] C.-S. Park, "One-time password based on hash chain without shared secret and re-registration," *Computers & Security*, vol. 75, pp. 138–146, 2018.

[10] Y. C. Hu, M. Jakobsson, and A. Perrig, "Efficient constructions for one-way hash chains," in *Applied Cryptography and Network Security. ACNS 2005*, pp. 423–441, Springer, 2005.

[11] D. Liu and P. Ning, "Multilevel μTESLA," *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 4, pp. 800–836, 2004.

[12] T. Dai, H. P. Huang, R. C. Wang, and X. X. Pan, "Novel self-renewal hash chain based on Ito-Saito-Nishizeki secret sharing scheme," *The Journal of China Universities of Posts and Telecommunications*, vol. 19, pp. 122–127, 2012.

[13] Z. Wei, "Self-updating hash chains based on erasure coding," in *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, pp. 173–175, Changchun, China, 2010.

[14] X.-Y. Yang, J.-J. Wang, J.-Y. Chen, and X.-Z. Pan, "A self-renewal hash chain scheme based on fair exchange idea(SRHC-FEI)," in *2010 3rd International Conference on Computer Science and Information Technology*, pp. 152–156, Chengdu, China, 2010.

[15] H. Zhang and Y. Zhu, "Self-updating hash chains and their implementations," in *Web Information Systems – WISE 2006. WISE 2006*, pp. 387–397, Springer, 2006.

[16] H. Zhang, X. Li, and R. Ren, "A novel self-renewal hash chain and its implementation," *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, 2008, pp. 144–149, Shanghai, China, 2008.

[17] M. Q. Zhang, B. Dong, and X. Y. Yang, "A new self-updating hash chain scheme," in *2009 International Conference on Computational Intelligence and Security*, pp. 315–318, Beijing, China, 2009.

[18] D. Coppersmith and M. Jakobsson, "Almost optimal hash sequence traversal," in *Financial Cryptography. FC 2002*, pp. 102–119, Springer, 2003.

[19] M. Jakobsson, "Fractal hash sequence representation and traversal," in *Proceedings IEEE International Symposium on Information Theory*, Lausanne, Switzerland, 2002.

[20] D. Kogan, N. Manohar, and D. Boneh, "T/Key: second factor authentication from secure hash chains," in *Proceedings Article published 30 Oct 2017 in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 983–999, Dallas, TX, USA, 2017.

[21] J. Håstad and M. Näslund, "Practical construction and analysis of pseudo-randomness primitives," in *Advances in Cryptology — ASIACRYPT 2001. ASIACRYPT 2001*, pp. 442–459, Springer, 2001.

[22] C. Wang, D. Wang, Y. Tu, G. Xu, and H. Wang, "Understanding node capture attacks in user authentication schemes for wireless sensor networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 5971, 2020.

[23] D. Wang, W. Li, and P. Wang, "Measuring two-factor authentication schemes for real-time data access in industrial wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4081–4092, 2018.

[24] D. Wang and P. Wang, "Two birds with one stone: two-factor authentication with security beyond conventional bound," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 708–722, 2018.