A Novel Hash Chain Construction for Simple and Efficient Authentication

Yuta Kurihara* and Masakazu Soshi*
*Graduate School of
Information Sciences
Hiroshima City University
Hiroshima, 731-3194, Japan

***Abstract*—These years Internet of Things (IoT) has been paid much attention to and the importance of lightweight and efficient authentication protocols has been increasing.**

In this paper, we propose a novel and flexible hash chain construction, *hash chain aggregation* (HCA), and a scheme to establish a common key for two users using HCA. Our proposed scheme has the following significant advantages: (1) cryptographic primitives for our scheme are hash functions only and the resultant scheme is efficient, (2) our scheme is based on a totally new hash chain construction (HCA), and (3) what two users generating a common secret key must know is only the identity (ID) of the other. No communication is required except for in the initial setting. That is, our scheme is actually an ID-based authentication protocol.

I. Introduction

These years Internet of Things (IoT) [1] has been paid much attention to. We can consider IoT to be a network environment that connects various resource-constrained devices. Such devices have limited computational and storage capabilities and often suffer from the buttery consumption problem. However, because they are interconnected through a network, security is of critical concern. Therefore the importance of lightweight and efficient authentication protocols has been increasing.

One of the most promising lightweight cryptographic primitive is a hash function. A *hash function* is a *one-way* and *collision-resistant* function [2] and can efficiently be computed. Furthermore, a *hash chain* is a sequence of hash values, each of which is generated by successively applying a hash function to an input value, which is called a *seed*. Hash chains are important applications of a hash function and are building blocks of many security protocols [3]–[6]. However, unfortunately, previous protocols use hash chains in a rather straightforward manner and thus can not fully exploit the potential of hash chains.

Therefore, in this paper, first we shall propose a novel and flexible hash chain construction, *hash chain aggregation* (HCA). Second, we propose a scheme to establish a common secret key for two users using HCA. Our proposed scheme has the following significant advantages: (1) cryptographic primitives for our scheme are hash functions only and the resultant scheme is efficient, (2) our scheme is based on a totally new hash chain construction (HCA), and (3) what two users generating a common key must know is only the identity (ID) of the other. No communication is required except for in the initial setting. That is, our scheme is actually an ID-based authentication protocol.

A hash chain construction proposed in this paper is interesting in itself. We believe that it opens up a new vista of research in hash chain constructions.

Finally, we discuss security and performance of our scheme in detail in this paper.

The remainder of this paper is organized as follows. Section II discusses related work. We propose a new hash chain construction and a key generation scheme with it in Section III. In Section IV we evaluate security and performance of our scheme in detail. Finally Section V concludes the paper.

II. Related Work

This section discusses some important security protocols with hash chains.

Lamport proposed a simple authentication protocol using hash chains, which is also known as a one time password protocol [4]. The advantage of Lamport's protocol is that each password (a hash value) is used only once and there is no problem even if it is exposed. However, the protocol is mainly utilized for authentication between a server and users. It is not appropriate for mutual authentication between a pair of ordinary users.

TESLA is one of the most famous protocols with hash chains [5]. TESLA employs hash chains to develop an efficient broadcast authentication protocol and is advantageous in terms of computational cost and resistance to a packet loss. However, TESLA is too complicated when a simple mutual authentication protocol is needed.

Joye and Yen generalize the idea of hash chains to propose one-way cross trees (OWCT) [3]. OWCT generates independent secret values by applying hash functions to each element of a sequence of hash values in a multidimensional manner. Although OWCT has various applications such as key escrow, it is not known how to establish a common secret of two users using OWCT.

III. Our Scheme

In this section we shall propose an authentication scheme (a common key generation scheme for two users) using a new hash chain construction.

To begin with, let us define a hash chain. With respect to a *hash function* h and an input value (*seed*) s, we define $h^1(s) := h(s)$ and $h^i(s) := h(h^{i-1}(s))$ $(i \geq 2)$. For an integer n, suppose that $(i_1, i_2, ..., i_n)$ is a permutation of a set $\{1, 2, ..., n\}$. Now a (general) *hash chain* of *length* n with seed s is defined to be:

$$(v_1, v_2, ..., v_n) \quad \text{where } v_j = h^{i_j}(s), 1 \leq j \leq n \ . \qquad (1)$$

Note that n hash values are first computed as $h(s)$, $h^2(s)$, ..., $h^n(s)$ in this order. After that, the order of the hash values is rearranged to compose the sequence Eq. (1).

A. Basic Idea

This section demonstrates the basic idea behind our proposed scheme.

In our model there are two kinds of players involved: one *Key Distribution Center* (KDC) and N *users*. KDC is supposed to be trusted and there is a secure channel between KDC and every user.

Now in order to understand the basic idea, for an example, consider a situation where two users are about to generate their common secret key using hash chains.

It would be impossible to invent a scheme that establishes a common key of arbitrary two users using only one hash chain. Hence the first step toward the simplest solution would be that KDC prepares "two way hash chains" C_1 and C_2. For example, suppose that N is five, s_1 and s_2 are seeds, and h is a hash function. Then, we have $C_1 = (h(s_1), h^2(s_1), \ldots, h^5(s_1))$ and $C_2 = (h^5(s_2), h^4(s_2), \ldots, h(s_2))$. Next, after KDC computes C_1 and C_2, it sends $(h^i(s_1), h^{6-i}(s_2))$ to user i via her secure channel $(i = 1, 2, ..., 5)$. As a result, both user i and j (without loss of generality, here we assume $i < j$) either have, or can calculate, $h^j(s_1)$ and $h^{6-i}(s_2)$, from their own hash values received from KDC. Therefore when user i and j want to make a common secret key for secure communication, they only have to compute $F(h^j(s_1), h^{6-i}(s_2))$ as the common key with a one-way function F. Notice that to obtain $F(h^j(s_1), h^{6-i}(s_2))$, the two users need no communication and what they have to know is only the ID of the other.

Now consider the case where user 2 and 4 want to authenticate each other for an example. Because KDC sends $(h^2(s_1), h^4(s_2))$ and $(h^4(s_1), h^2(s_2))$ to user 2 and 4, respectively, user 2 and 4 can generate

$$K_{2,4} := F(h^4(s_1), h^4(s_2)) \qquad (2)$$

as their common key.

B. Improvement on the Basic Idea

In the basic scheme presented in Section III-A, if an attacker does not have any hash values that are distributed by KDC, then he cannot derive the common key of any two users. Therefore in order to consider security of the basic scheme, let us assume that an attacker is an insider, i.e., that he is one of N users. In what follows in this section, let an attacker be user a $(1 \leq a \leq N)$.

When $a < i$ or $j < a$, user a cannot obtain the common key $K_{i,j}$ of user i and j due to one-wayness of of hash function h. To see this, consider again the example scenario in Section III-A, where $N = 5$, $i = 2$, and $j = 4$. Suppose that attacker (user) $a = 1$. Because user 1, who received $(h(s_1), h^5(s_2))$ from KDC, cannot compute $h^4(s_2)$ due to one-wayness of h, he cannot compute the common key $K_{2,4}$ (Eq.(2)) of user 2 and 4. Similarly, if attacker (user) $a = 5$, who has $(h^5(s_1), h(s_2))$, then user 5 cannot generate $K_{2,4}$ because he cannot generate $h^4(s_1)$.

However, if $i < a < j$, then user a can compute $K_{i,j}$ indeed. In the example above, if attacker (user) $a = 3$, then user 3 has hash values $(h^3(s_1), h^3(s_2))$ and from them he can compute both $h^4(s_1)$ and $h^4(s_2)$ and then $K_{2,4}$. This is definitely undesirable.

Therefore we consider new hash chains to exclude user a $(i < a < j)$ when user i and j compute their common key. For that purpose, we shall introduce "non-consecutive" hash chains C_3 and C_4. Although we give the formal definitions of C_3 and C_4 in Section III-C, here in order to intuitively grasp the idea of our chains, let us consider the case $N = 5$ as in Section III-A.

Regarding some seeds s_3 and s_4, KDC computes $h(s_3)$, $h^2(s_3)$, ..., $h^5(s_3)$ and $h(s_4)$, $h^2(s_4)$, ..., $h^5(s_4)$ in order. Then KDC rearranges the hash values and forms the following hash chains C_3, C_4:

$$C_3 = (h^3(s_3), h^4(s_3), h^5(s_3), h(s_3), h^2(s_3)), \qquad (3)$$
$$C_4 = (h^3(s_4), h^2(s_4), h(s_4), h^5(s_4), h^4(s_4)) \ . \qquad (4)$$

For $1 \leq i \leq N$ $(= 5)$, we can express each hash value of C_3, C_4 as

$$h^{((i-\lceil N/2 \rceil - 1) \bmod N)+1}(s_3), \text{and}$$
$$h^{((\lceil N/2 \rceil - i) \bmod N)+1}(s_4),$$

respectively.

Recall that in previous work, exponents of a hash function in a hash chain strictly increase (or decrease). On the other hand, as we can see from Eq. (3), we cut an exponent-increasing hash chain (say C_1) in halves and exchange the order of the halves. We can consider the case of Eq. (4) similarly. It should be emphasized that this way of constructing hash chains is completely new and makes it possible to devise lightweight and efficient key generation algorithms as shown later.

Putting it altogether, our improved basic scheme is as follows. KDC forms hash chains C_1, C_2, C_3, and C_4 and securely distributes to user i $(1 \leq i \leq N)$ the four hash values given below:

$$h^i(s_1), h^{N-i+1}(s_2), h^{((i-\lceil N/2 \rceil - 1) \bmod N)+1}(s_3),$$
$$h^{((\lceil N/2 \rceil - i) \bmod N)+1}(s_4) \ .$$

See also Fig. 1.

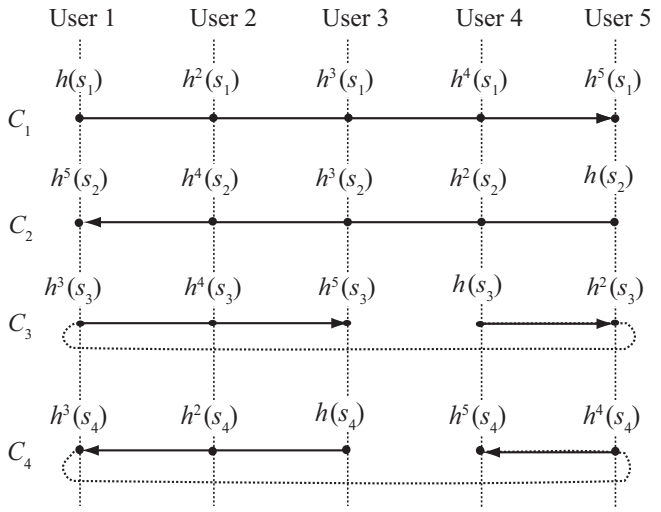

Fig. 1. Hash Chains C_1, C_2, C_3, C_4 ($N = 5$)

When user i and j ($i < j$) want to have their common secret key $K_{i,j}$, they compute:

$$h^j(s_1), h^{N-i+1}(s_2), h^{((i-\lceil N/2\rceil-1) \bmod N)+1}(s_3), \\ h^{((\lceil N/2\rceil-j) \bmod N)+1}(s_4) \tag{5}$$

from their own hash values and get $K_{i,j}$ by feeding these values into a one way function F.

Now consider the case $N = 5$ again. In the case, user 2, 3, and 4 receive from KDC $\{h^2(s_1), h^4(s_2), h^4(s_3), h^2(s_4)\}$, $\{h^3(s_1), h^3(s_2), h^5(s_3), h(s_4)\}$, and $\{h^4(s_1), h^2(s_2), h(s_3), h^5(s_4)\}$, respectively. Then user 2 and 4 can compute their common key as

$$F(h^4(s_1), h^4(s_2), h^4(s_3), h^5(s_4)) \ . \tag{6}$$

User 3 cannot obtain Eq. (6) because he cannot compute $h^4(s_3)$ due to one-wayness of hash function h.

It should be now easy to verify that in our improved basic scheme, with respect to almost all combinations of i and j ($1 \le i \le 3 < j \le 5 = N$) [1], no user other than user i and j cannot generate the common key of user i and j.

C. Our Hash Chain Construction

On the basis of the discussion given in Sections III-A and III-B, we shall propose a new way of hash chain constructions in this section.

Hereinafter for brevity, we assume that $N = 2^m$ for some integer m (≥ 2). Moreover for a sequence $Q = (q_1, q_2, \ldots, q_j)$, i-th element q_i of Q is denoted by $Q[i]$ ($1 \le i \le j$).

[1]Key generation by Eq. (5) fails in the four cases of total 10 ($= \binom{5}{2}$): (i, j) = (i) (1, 2), (ii) (1, 3), (iii) (2, 3), and (iv) (4, 5), respectively. However, with a little thought we see that the actual failure is the case (ii) (of total ten cases) only. Of course our final scheme presented in Section III-E has no such a flaw.

1) Basic Hash Chain: First, we introduce four *basic hash chains* (Type I, II, III, IV hash chains) below to define our proposed hash chain construction. For some integer b ($2 \le b \le m$), let the length ℓ of the basic hash chains be 2^b. Furthermore in the definitions below we suppose that $1 \le i \le \ell$ and s is a seed. Hereafter throughout the paper, h denotes a hash function.

- *Type I hash chain* $C_\ell^I(s)$

$$C_\ell^I(s) := (v_1, v_2, \ldots, v_\ell) \quad \text{where } v_i = h^i(s)$$

- *Type II hash chain* $C_\ell^{II}(s)$

$$C_\ell^{II}(s) := (v_1, v_2, \ldots, v_\ell) \quad \text{where } v_i = h^{\ell-i+1}(s)$$

- *Type III hash chain* $C_\ell^{III}(s)$

$$C_\ell^{III}(s) := (v_1, v_2, \ldots, v_\ell)$$
$$\text{where } v_i = h^{((i-\ell/2-1) \bmod \ell)+1}(s)$$

- *Type IV hash chain* $C_\ell^{IV}(s)$

$$C_\ell^{IV}(s) := (v_1, v_2, \ldots, v_\ell)$$
$$\text{where } v_i = h^{((\ell/2-i) \bmod \ell)+1}(s)$$

2) Hash Chain List: By the definitions in Section III-C1, we can now define a *hash chain list* (HCL) $\mathcal{L}(\tau, k, s_1, ..., s_k)$ as follows:

$$\mathcal{L}(\tau, k, s_1, ..., s_k) \\ := \quad (C_{N/k}^\tau(s_1), C_{N/k}^\tau(s_2), ..., C_{N/k}^\tau(s_k)) \tag{7}$$

where $k = 2^f$ for some integer f ($0 \le f \le m-2$), $\tau \in \{I, II, III, IV\}$, and $s_1, ..., s_k$ are distinct seeds.

Hereinafter for simplicity we sometimes omit the input variables $\tau, k, s_1, ..., s_k$ of $\mathcal{L}(\tau, k, s_1, ..., s_k)$ and merely represent it as $\mathcal{L}$. Moreover, with regard to hash chain list $\mathcal{L}$, the number of hash chains in $\mathcal{L}$ is denoted by $k_\mathcal{L}$ ($= k$ in Eq. (7)).

3) Hash Chain Aggregation: Now it is the time that we shall propose a hash chain aggregation. A *hash chain aggregation* (HCA) is simply a list of hash chain lists:

$$\mathcal{A} := (\mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_r) \tag{8}$$

where $\mathcal{L}_i$ is defined by Eq. (7) ($1 \le i \le r$).

4) Hash Values for Each User: In our scheme, KDC uses a hash chain aggregation (HCA) to distribute a set of hash values to user i ($1 \le i \le N$). Let us discuss what hash values are sent to users by KDC.

To begin with, consider an HCA $\mathcal{A}$ and arbitrary HCL $\mathcal{L}$ ($\in \mathcal{A}$) (see also Eq. (8)). It is assumed that $k_\mathcal{L} = 2^f$. Then, $\mathcal{L}$ has $k_\mathcal{L}$ basic hash chains of the same type and the length of each chain in $\mathcal{L}$ is equally $N/k_\mathcal{L}$ ($= 2^{m-f}$). It follows that $\mathcal{L}$ has N distinct hash values in total and can assign one hash value to user i ($1 \le i \le N$) from one basic hash chain in $\mathcal{L}$. Given $\mathcal{L}$, we can determine the hash value and the basic hash chain for user i in the following manner.

Assume that one hash value of d-th basic hash chain in $\mathcal{L}$ is assigned to user i. d must satisfy the following inequality:

$$\frac{N}{k_{\mathcal{L}}}(d-1)+1 \leq i \leq \frac{N}{k_{\mathcal{L}}}d \quad (1 \leq d \leq k_{\mathcal{L}}) \ . \tag{9}$$

Note that given $\mathcal{L}$ and i, the value of d that satisfies Eq. (9) is uniquely determined. We refer to the value as $\alpha(\mathcal{L}, i)$. Now we know that the basic hash chain that contains the hash value for user i in $\mathcal{L}$ is represented by $\mathcal{L}[\alpha(\mathcal{L}, i)]$.

From the discussion above, in $\mathcal{L}$, we find that the hash value assigned to user i is obtained by:

$$(\mathcal{L}[\alpha(\mathcal{L}, i)])\left[i - \frac{N}{k_{\mathcal{L}}} \cdot (\alpha(\mathcal{L}, i) - 1)\right] \ . \tag{10}$$

Furthermore, the set of hash values assigned to user i are given by $V(\mathcal{A}, i)$, which is defined as follows:

$$V(\mathcal{A}, i) := \left\{(\mathcal{L}[\alpha(\mathcal{L}, i)])\left[i - \frac{N}{k_{\mathcal{L}}} \cdot (\alpha(\mathcal{L}, i) - 1)\right] \;\middle|\; \mathcal{L} \in \mathcal{A}\right\} \ . \tag{11}$$

KDC sends $V(\mathcal{A}, i)$ to user i via her secure channel. She stores them secretly.

Finally, with respect to $\mathcal{A}$, we introduce function W to express the set of basic hash chains that contain the hash values of user i. W is defined as:

$$W(\mathcal{A}, i) := \{\mathcal{L}[\alpha(\mathcal{L}, i)] \mid \mathcal{L} \in \mathcal{A}\} \ . \tag{12}$$

D. Proposed Hash Chain Aggregation

In this section in order to realize lightweight and efficient mutual authentication, we shall propose our HCA, $\mathcal{HCA}$, as follows. Recall that $N = 2^m$.

$$\mathcal{HCA} := (\mathcal{L}_{(1)}, \mathcal{L}_{(2)}, \ldots, \mathcal{L}_{(2m-1)}, \mathcal{L}_{(2m)}) \tag{13}$$

where $\mathcal{L}_{(1)} = (C_N^I(s_1))$ and $\mathcal{L}_{(2)} = (C_N^{II}(s_2))$. When $3 \leq i \leq 2m$, $\mathcal{L}_{(i)}$ is defined as follows. We assume that $2 \leq j \leq m$ and $s_x, s_{y,z}$ denote distinct seeds for integers x, y, z.

1) If $i = 2j - 1$, then

$$\mathcal{L}_{(2j-1)} := (C^{III}_{2^{m-j+2}}(s_{2j-1,1}), C^{III}_{2^{m-j+2}}(s_{2j-1,2}), \cdots, C^{III}_{2^{m-j+2}}(s_{2j-1,2^{j-2}})) \ .$$

2) If $i = 2j$, then

$$\mathcal{L}_{(2j)} := (C^{IV}_{2^{m-j+2}}(s_{2j,1}), C^{IV}_{2^{m-j+2}}(s_{2j,2}), \cdots, C^{IV}_{2^{m-j+2}}(s_{2j,2^{j-2}})) \ .$$

In particular, we have $\mathcal{L}_{(3)} = (C_N^{III}(s_{3,1}))$ and $\mathcal{L}_{(4)} = (C_N^{IV}(s_{4,1}))$. In the rest of this paper, for simplicity we write $s_{3,1}$ and $s_{4,1}$ as s_3 and s_4, respectively.

Now in order to see how $\mathcal{HCA}$ is constructed, let us consider the case $N = 8$ ($m = 3$). In the case we obtain $\mathcal{HCA} = (\mathcal{L}_{(1)}, \mathcal{L}_{(2)}, \ldots, \mathcal{L}_{(5)}, \mathcal{L}_{(6)})$, which is depicted in Fig. 2. It is easy to see $\mathcal{L}_{(5)} = (C_4^{III}(s_{5,1}), C_4^{III}(s_{5,2}))$ and $k_{\mathcal{L}_{(5)}} = 2$. Similarly, $\mathcal{L}_{(6)} = (C_4^{IV}(s_{6,1}), C_4^{IV}(s_{6,2}))$ and $k_{\mathcal{L}_{(6)}} = 2$. Hence, for example, regarding user 6, we have $\alpha(\mathcal{L}_{(5)}, 6) = \alpha(\mathcal{L}_{(6)}, 6) = 2$ and $\mathcal{L}_{(5)}[2] = C_4^{III}(s_{5,2})$ and $\mathcal{L}_{(6)}[2] = C_4^{IV}(s_{6,2})$. It follows that $W(\mathcal{HCA}, 6) = \{C_8^I(s_1), C_8^{II}(s_2), C_8^{III}(s_3), C_8^{IV}(s_4), C_4^{III}(s_{5,2}), C_4^{IV}(s_{6,2})\}$. $V(\mathcal{HCA}, 6)$ can be computed from Eq. (11). For example, $(\mathcal{L}_{(5)}[2])[6-(8/2)\cdot(2-1)] = C_4^{III}(s_{5,2})[2] = h^4(s_{5,2})$. Finally we have $V(\mathcal{HCA}, 6) = \{h^6(s_1), h^3(s_2), h^2(s_3), h^7(s_4), h^4(s_{5,2}), h(s_{6,2})\}$.

E. Key Generation using $\mathcal{HCA}$

Our proposed common key generation scheme for user i, j ($1 \leq i < j \leq N$) is defined in this section.

1) If $i + 1 = j$, then $F(C_N^I[j], C_N^{II}[i])$ is the common key.
2) Otherwise, first note that there exist a set of integers $(2 \leq) \ w_1 < w_2 < \cdots < w_u \ (\leq m)$ such that we can uniquely write $W(\mathcal{HCA}, i) \cap W(\mathcal{HCA}, j)$ in the following way:

$$\begin{aligned} &W(\mathcal{HCA}, i) \cap W(\mathcal{HCA}, j) = \\ &\quad \{C_N^I(s_1), C_N^{II}(s_2), \\ &\quad \mathcal{L}_{(2w_1-1)}[\alpha(\mathcal{L}_{(2w_1-1)}, i)], \mathcal{L}_{(2w_1)}[\alpha(\mathcal{L}_{(2w_1)}, i)], \\ &\quad \vdots \\ &\quad \mathcal{L}_{(2w_u-1)}[\alpha(\mathcal{L}_{(2w_u-1)}, i)], \mathcal{L}_{(2w_u)}[\alpha(\mathcal{L}_{(2w_u)}, i)]\} \end{aligned}$$

where $\alpha(\mathcal{L}_{(2w_1-1)}, i) = \alpha(\mathcal{L}_{(2w_1-1)}, j) = \alpha(\mathcal{L}_{(2w_1)}, i) = \alpha(\mathcal{L}_{(2w_1)}, j)$, $\cdots$, $\alpha(\mathcal{L}_{(2w_u-1)}, i) = \alpha(\mathcal{L}_{(2w_u-1)}, j) = \alpha(\mathcal{L}_{(2w_u)}, i) = \alpha(\mathcal{L}_{(2w_u)}, j)$. It should be noted that $\mathcal{L}_{(2w_u-1)}[\alpha(\mathcal{L}_{(2w_u-1)}, i)]$ and $\mathcal{L}_{(2w_u)}[\alpha(\mathcal{L}_{(2w_u)}, i)]$ are the shortest basic hash chains that contain both i and j. Now both user i and j can compute the following four hash values:

$$\begin{aligned} &\nu_1 := C_N^I(s_1)[j], \\ &\nu_2 := C_N^{II}(s_2)[i], \\ &\nu_3 := (\mathcal{L}_{(2w_u-1)}[\alpha(\mathcal{L}_{(2w_u-1)}, i)])[i - \beta_{i,2w_u-1}], \\ &\nu_4 := (\mathcal{L}_{(2w_u)}[\alpha(\mathcal{L}_{(2w_u)}, i)])[j - \beta_{j,2w_u}] \end{aligned} \tag{14}$$

where $\beta_{x,y} = (N \cdot (\alpha(\mathcal{L}_{(y)}, x) - 1))/k_{\mathcal{L}_{(y)}}$.

The common key $K_{i,j}$ can be computed as:

$$K_{i,j} := F(\nu_1, \nu_2, \nu_3, \nu_4) \ . \tag{15}$$

F. Example

To demonstrate how our scheme works using $\mathcal{HCA}$, we consider the case where user 5 and 7 are going to generate their common secret key when $N = 8$. See Fig. 2.

We find that $W(\mathcal{HCA}, 5) = \{C_8^I(s_1), C_8^{II}(s_2), C_8^{III}(s_3), C_8^{IV}(s_4), C_4^{III}(s_{5,2}), C_4^{IV}(s_{6,2})\}$ and $W(\mathcal{HCA}, 7) = \{C_8^I(s_1), C_8^{II}(s_2), C_8^{III}(s_3), C_8^{IV}(s_4), C_4^{III}(s_{5,2}), C_4^{IV}(s_{6,2})\}$. Therefore $W(\mathcal{HCA}, 5) \cap W(\mathcal{HCA}, 7) = \{C_8^I(s_1), C_8^{II}(s_2), C_8^{III}(s_3), C_8^{IV}(s_4), C_4^{III}(s_{5,2}), C_4^{IV}(s_{6,2})\}$ and $w_1 = 2$, $w_u = 3$ (i.e, $u = 2$), $\alpha(\mathcal{L}_{(3)}, 5) = \alpha(\mathcal{L}_{(3)}, 7) = \alpha(\mathcal{L}_{(4)}, 5) = \alpha(\mathcal{L}_{(4)}, 7) = 1$, $\alpha(\mathcal{L}_{(5)}, 5) = \alpha(\mathcal{L}_{(5)}, 7) = \alpha(\mathcal{L}_{(6)}, 5) = \alpha(\mathcal{L}_{(6)}, 7) = 2$, $\beta_{5,5} = \beta_{7,6} = 4$. Hence

$$\begin{aligned} &\nu_1 = C_8^I(s_1)[7] = h^7(s_1), \\ &\nu_2 = C_8^{II}(s_2)[5] = h^4(s_2), \\ &\nu_3 = (\mathcal{L}_{(5)}[2])[5-4] = C_4^{III}(s_{5,2})[1] = h^3(s_{5,2}), \\ &\nu_4 = (\mathcal{L}_{(6)}[2])[7-4] = C_4^{IV}(s_{6,2})[3] = h^4(s_{6,2}) \ . \end{aligned}$$

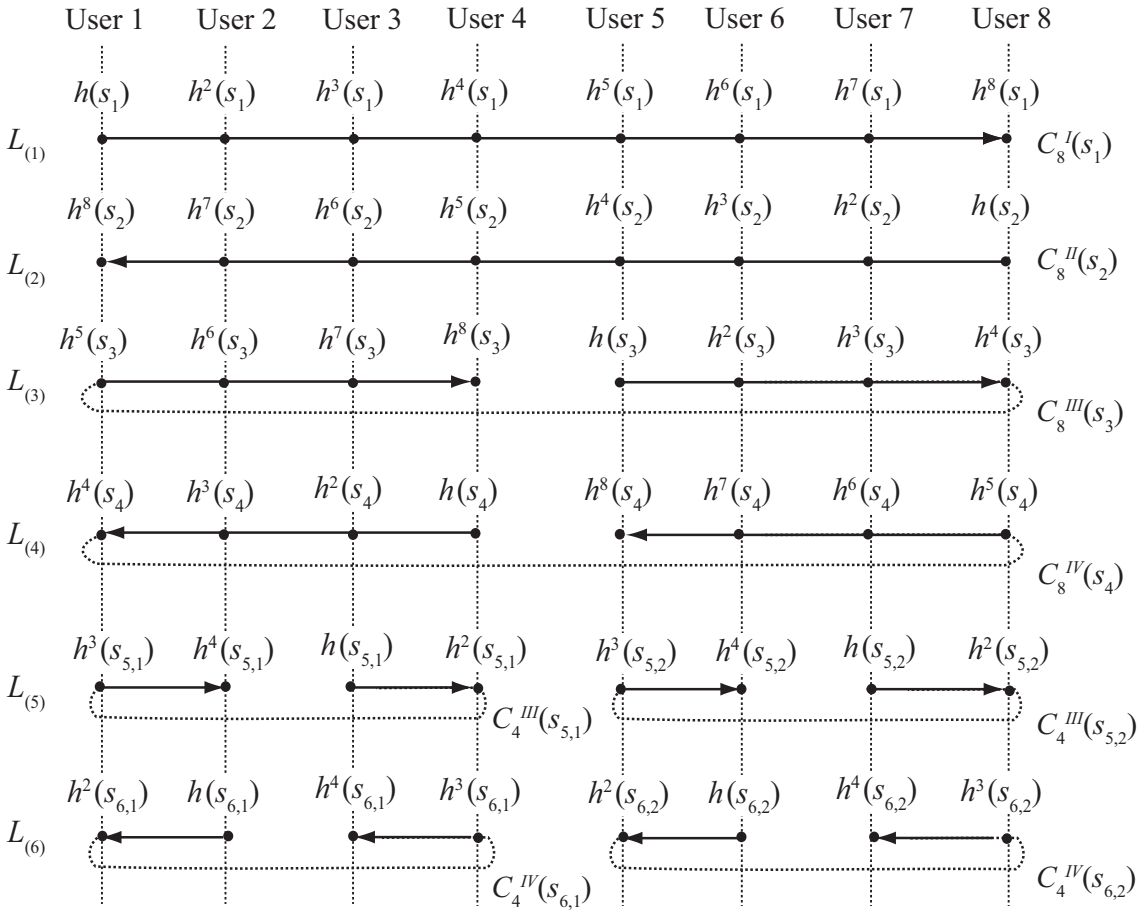

Fig. 2. Our $\mathcal{HCA}$ ($N = 8$)

Notice that user 5 and 7 can calculate $K_{5,7}$ from their own hash values without any communication. Furthermore, no user other than user 5 and 7 can obtain $K_{5,7}$. Security of our scheme is evaluated in detail in Section IV-A.

IV. Evaluation

This section evaluates security and performance of our scheme in detail.

A. Security Evaluation

In our scheme, KDC distributes a set of hash values to each user via her secure channel and she computes her keys from her own hash values with hash function h and one way function F. Due to one-wayness of h and F, an attacker other than N users cannot derive such keys. Therefore, if an attacker succeeds in computing some keys in our scheme, then he must have obtained some hash values in some way or he must be among N users. In either case, we can suppose that some user is actually an attacker when we evaluate security of our scheme.

1) Attack by a Single Attacker: If there exists only one attacker, then we can prove Theorem 1.

Theorem 1. *Suppose that there exists only one attacker. For arbitrary i and j ($1 \leq i < j \leq N$), if the attacker is neither user i nor j, then he cannot compute the common key $K_{i,j}$ of user i and j.*

Proof. From the discussion above, when we evaluate security of our scheme, we can suppose that user a ($1 \leq a \leq N$) is the attacker. Now we know that in order to compute $K_{i,j}$, from Eq. (14), we must have at least $C_N^I(s_1)[j]$ and $C_N^{II}(s_2)[i]$. However, if $a < i$, then user a (the attacker) cannot obtain $C_N^{II}(s_2)[i]$ due to one-wayness of h. Similarly, if $j < a$, then user a cannot obtain $C_N^I(s_1)[j]$.

Next we assume that $i < a < j$, which implies that $i + 1 \neq j$. Then user a can compute $C_N^I(s_1)[j]$ and $C_N^{II}(s_2)[i]$. However, user a cannot generate just one of ν_3 and ν_4 in Eq. (14) (however, he can actually generate the other). We prove this fact below.

First from Eq. (14), with respect to $\mathcal{L}_{(2w_u-1)}$ and $\mathcal{L}_{(2w_u)}$, we can write

$$\mathcal{L}_{(2w_u-1)} = \mathcal{L}(III, k, s_{2w_u-1,1}, ..., s_{2w_u-1,k})$$
$$\mathcal{L}_{(2w_u)} = \mathcal{L}(IV, k, s_{2w_u,1}, ..., s_{2w_u,k})$$

where $k = k_{\mathcal{L}_{(2w_u-1)}} = k_{\mathcal{L}_{(2w_u)}} = 2^{w_u-2}$. Furthermore, as we mentioned in Section III-E, we know $\alpha(\mathcal{L}_{(2w_u-1)}, i) = \alpha(\mathcal{L}_{(2w_u-1)}, j) = \alpha(\mathcal{L}_{(2w_u)}, i) = \alpha(\mathcal{L}_{(2w_u)}, j)$, the value of which is simply denoted by α.

Now we split the range from i to j (not including i and j) into two smaller ones and consider which range a is in. Note that from the definition of w_u and $i+1 \neq j$, we see that $i \leq N(\alpha-1)/k + N/2k$ and $N(\alpha-1)/k + N/2k + 1 \leq j$.

1) If $i < a \leq N(\alpha-1)/k + N/2k$, then user a cannot compute ν_3 and in turn cannot compute $K_{i,j}$.
2) If $N(\alpha-1)/k + N/2k + 1 \leq a < j$, then user a cannot compute ν_4.

Thus the theorem is proved. □

2) Collusion Attack: If two attackers collude to attack our scheme, then we can prove Theorem 2.

Theorem 2. *Let α and k be defined in the same way as in Theorem 1. Furthermore, for some integers i, j, a_1, a_2, let us suppose that $1 \leq i < a_1 \leq N(\alpha-1)/k + N/2k$ and $N(\alpha-1)/k + N/2k + 1 \leq a_2 < j \leq N$. Then user a_1 and a_2 in collusion can compute the common key $K_{i,j}$ of user i and j.*

Proof. First, see Eq. (14). Both user a_1 and a_2 can compute ν_1 and ν_2. Furthermore, user a_1 can compute ν_4 (but not ν_3). User a_2 can compute ν_3 (but not ν_4). Therefore, if user a_1 and a_2 cooperate, then they can compute $K_{i,j}$. □

As a consequence of Theorem 2, our scheme should be used in moderately trusted environments.

B. Performance Evaluation

We discuss performance of our scheme in detail in this section. Performance of our scheme is discussed in terms of the number of hash values that each user should have (Section IV-B1) and complexity (SectionIV-B2). Furthermore, in Section IV-B3 we discuss the situations where the number of each user's hash values is less than four.

1) The Number of Hash Values Stored By Each User: If the number of users is N, then in our scheme the number of hash values that each user must store securely is

$$2\lceil \log_2 N \rceil \ .$$

On the other hand, in a naive mutual authentication protocol using a symmetric cipher, each user would possess $N-1$ secret keys. Thus our scheme is highly efficient in terms of the number of each user's hash values.

2) Complexity: We evaluate the number of hash function computations when user i and j ($1 \leq i < j \leq N$) generate their common key $K_{i,j}$. See Eq. (14). Let us consider the number of hash computations that user i does for the key generation (almost the same discussion applies to the case of user j).

User i has ν_2 and ν_3 and no computation is necessary to derive them. In order to obtain ν_1, user i computes hash function h $(j-i)$ times. Similarly, user i computes h $((i-j) \bmod (N/k))$ times to have ν_4, where $k = 2^{w_u - 2}$. Therefore, except for the computation of F, the number of user i's hash computations when she generates $K_{i,j}$ can be approximately estimated as

$$\frac{N}{3} + \frac{N}{k} \ . \tag{16}$$

Hence it is expected that the larger N becomes, the larger the complexity of our scheme becomes.

However, modern hash functions, say, SHA-2 or SHA-3, are known to be remarkably fast, especially in hardware implementations [7]. Furthermore, we can compute hash chains efficiently by using the well-known techniques developed by Coppersmith and Jakobsson [8].

Therefore we can expect our scheme to work well in practical environments.

3) On the Number of Hash Values for Key Generation: In order not for users other than user i and j ($1 \leq i < j \leq N$) to be able to compute the common key $K_{i,j}$ of user i and j, we need only four hash values $(\nu_1, \nu_2, \nu_3, \nu_4)$ in Eq. (14). However, depending on i and j, we have the cases where less number of hash values are enough for the key generation.

Below α and k are defined in the same way as in Theorem 1. Moreover in the examples below, $N = 8$ (see Fig. 2).

1) $i + 1 = j$ or adjacent users in Type III and IV chains:
   When $i + 1 = j$, two hash values ν_1 and ν_2 suffice.
   Moreover, because user $i = N(\alpha-1)k + 1$ is adjacent to user $j = N\alpha/k$ in Type III and IV hash chains, in the case, two hash values ν_3 and ν_4 suffice.
   Example: if $i = 1$ and $j = 8$, then $\nu_3 = C_8^{III}(s_3)[1] = h^5(s_3)$, $\nu_4 = C_8^{IV}(s_4)[8] = h^5(s_4)$.
2) $i = N(\alpha-1)/k + N/2k$, $j = N\alpha/k$:
   Two hash values ν_2 and ν_4 suffice.
   Example: if $i = 4$ and $j = 8$, then $\nu_2 = C_8^{II}(s_2)[4] = h^5(s_2)$, $\nu_4 = C_8^{IV}(s_4)[8] = h^5(s_4)$.
3) $i = N(\alpha-1)/k + 1$, $j = N(\alpha-1)/k + N/2k + 1$:
   Two hash values ν_1 and ν_3 suffice.
   Example: if $i = 5$ and $j = 7$, then $\nu_1 = C_8^{I}(s_1)[7] = h^7(s_1)$, $\nu_3 = C_4^{III}(s_{5,2})[1] = h^3(s_{5,2})$.
4) $i = N(\alpha-1)/k + N/2k$, $N(\alpha-1)/k + N/2k + 1 < j < N\alpha/k$:
   Three hash values ν_1, ν_2, and ν_4 suffice.
   Example: if $i = 4$, $j = 6$, then $\nu_1 = C_8^{I}(s_1)[6] = h^6(s_1)$, $\nu_2 = C_8^{II}(s_2)[4] = h^5(s_2)$, $\nu_4 = C_8^{IV}(s_4)[6] = h^7(s_4)$.
5) $1 < i < N(\alpha-1)/k + N/2k$, $j = N(\alpha-1)/k + N/2k + 1$:
   Three hash values ν_1, ν_2, and ν_3 suffice.
   Example: if $i = 3$, $j = 5$, then $\nu_1 = C_8^{I}(s_1)[5] = h^5(s_1)$, $\nu_2 = C_8^{II}(s_2)[3] = h^6(s_2)$, $\nu_3 = C_8^{III}(s_3)[3] = h^7(s_3)$.

V. Conclusion

In this paper, we proposed a novel and flexible hash chain construction, *hash chain aggregation* (HCA), and a scheme to establish a common key for two users using HCA. Our proposed scheme has the following significant advantages: (1) cryptographic primitives for our scheme are hash functions only and the resultant scheme is efficient, (2) our scheme is based on a totally new hash chain construction (HCA), and (3) what two users generating a common secret key must know is only the identity (ID) of the other. No communication is

required except for in the initial setting. That is, our scheme is actually an ID-based authentication protocol.

A hash chain construction proposed in this paper is interesting in itself. We believe that it opens up a new vista of research on hash chain constructions.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 15K00189 and Japan Science and Technology Agency (JST), Infrastructure Development for Promoting International S&T Cooperation.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[2] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*. Chapman and Hall/CRC, 2007.

[3] M. Joye and S. Yen, "One-way cross-trees and their applications," in *Public Key Cryptography (PKC)*, ser. Lecture Notes in Computer Science, vol. 2274. Springer-Verlag, 2002, pp. 346–356.

[4] L. Lamport, "Password authentication with insecure communication," vol. 24, no. 11, pp. 770–772, Nov. 1981.

[5] A. Perrig, R. Canetti, J. D. Tygar, and D. Song, "Efficient authentication and signing of multicast streams over lossy channels," in *Proceedings of the IEEE Symposium on Security and Privacy*, May 2000, pp. 56–73.

[6] R. Rivest and A. Shamir, "PayWord and MicroMint: Two simple micropayment schemes," in *Security Protocols*, ser. Lecture Notes in Computer Science, vol. 1189. Springer-Verlag, 1997, pp. 69–87.

[7] X. Guo, S. Huang, L. Nazhandali, and P. Schaumont, "Fair and comprehensive performance evaluation of 14 second round SHA-3 ASIC implementations," NIST 2nd SHA-3 Candidate Conference, Aug. 2010.

[8] D. Coppersmith and M. Jakobsson, "Almost optimal hash sequence traversal," in *Proceesings of 6th International Conference on Financial Cryptography (FC 2002)*, ser. Lecture Notes in Computer Science, vol. 2357. Springer-Verlag, 2002, pp. 102–119.