LETTER

Generalized Hash Chain Traversal with Selective Output

Dae Hyun YUM[†a)], **Jae Woo SEO**[†], **Kookrae CHO**[††], *Nonmembers*, *and* **Pil Joong LEE**[†], *Member*

SUMMARY A hash chain H for a one-way hash function $h(\cdot)$ is a sequence of hash values $\langle v_0, v_1, \ldots, v_n \rangle$, where v_0 is a public value, v_n a secret value, and $v_i = h(v_{i+1})$. A hash chain traversal algorithm $\mathcal{T}$ computes and outputs the hash chain H, returning v_i in time period (called round) i for $1 \le i \le n$. While previous hash chain traversal algorithms were designed to output all hash values v_i $(1 \le i \le n)$ in order, there are applications where every m-th hash value (i.e., $v_m, v_{2m}, v_{3m}, \ldots$) is required to be output. We introduce a hash chain traversal algorithm that selectively outputs every m-th hash value efficiently. The main technique is a transformation from a hash chain traversal algorithm outputting every hash value into that outputting every m-th hash value. Compared with the direct use of previous hash chain traversal algorithms, our proposed method requires less memory storages and computational costs.
key words: *hash chain, fractal traversal, amortization*

1. Introduction

A one-way hash function $h : \{0,1\}^* \to \{0,1\}^l$ converts an arbitrary block of input message into a fixed-size bit string. It is easy to compute the hash value for a given message but infeasible to find a message that has a given hash value. A hash chain H for the one-way hash function $h(\cdot)$ is a sequence of hash values $\langle v_0, v_1, \ldots, v_n \rangle$, where v_n is a randomly chosen secret value, v_i is iteratively generated by $v_i = h(v_{i+1})$ (from $i = n-1$ to $i = 0$), and v_0 is a public value. For $i < i'$, v_i can be easily calculated from $v_{i'}$ by $v_i = h^{i'-i}(v_{i'})$ but computing $v_{i'}$ from v_i is infeasible because of the one-wayness of $h(\cdot)$. A hash chain traversal algorithm $\mathcal{T}$ computes and outputs the hash chain H, starting from v_1 and ending with v_n, by using dynamic memory storages (called pebbles). All previous hash chain traversal algorithms (e.g., [1]–[5]) have the output interval $m = 1$, which means that $\mathcal{T}$ outputs all hash values $v_1, v_2, v_3, \ldots, v_n$.

Hash chains have been used for a variety of security applications such as one-time password [6], secure routing [7], multicast authentication [8], payment system [9], and online auction [10]. To utilize hash chains efficiently, applications need appropriate hash chain traversal algorithms. As each application requires a different hash output interval, hash chain traversal algorithms with various m are required. For example, a distance vector update originated from a node in SEAD (Secure Efficient Ad hoc Distance vector) routing protocol contains a sequence number and a metric for each destination [7]. The sequence number is used to indicate the freshness of each route update and is limited by n (the length of hash chain). The metric is the distance, measured in number of hops, from the originating node to the destination and is limited by m (the output interval). To date, no hash chain traversal algorithm with $m \ge 2$ is known. Therefore, one has no choice but to use a hash chain traversal algorithm $\mathcal{T}$ with $m = 1$ and extract $v_m, v_{2m}, v_{3m}, \ldots$ from the outputs of $\mathcal{T}$ (i.e., $v_1, v_2, v_3, \ldots, v_n$).

In this work, we introduce a generalized hash chain traversal algorithm that selectively outputs every m-th hash value efficiently where $m \ge 1$, to which previous algorithms belong as a special case of $m = 1$. The main technique is a transformation from a hash chain traversal algorithm $\mathcal{T}$ with $m = 1$ into a generalized traversal algorithm $\mathcal{GT}$ with $m \ge 1$. Since we treat the underlying traversal algorithm $\mathcal{T}$ as a black-box, any previous hash chain traversal algorithm can be used as an input to the proposed transformation. The generalized hash chain traversal algorithm obtained from the transformation reduces memory storages and computational costs.

2. Hash Chain Traversal for *m* = 1

Influenced by amortization techniques of [11], Jakobsson [1] first introduced a single-layer fractal hash chain traversal algorithm that can traverse a hash chain of length n with $\lceil \log n \rceil$ budget and $\lceil \log n \rceil$ pebbles, where budget means the worst case computational cost to output each hash value and $\log \doteq \log_2$. To reduce the budget, Coppersmith and Jakobsson [2] adopted a double-layer fractal structure with extra pebbles and constructed a hash chain traversal algorithm that can traverse a hash chain of length n with $\lfloor \frac{1}{2} \log n \rfloor$ budget and $\lceil \log n \rceil + \lceil \log(\log n + 1) \rceil$ pebbles, which is almost optimal in terms of budget-times-storage complexity.

At CT–RSA 2009, we introduced a single-layer fractal hash traversal algorithm that is also almost optimal [5]*. While our algorithm of [5] is based on the simple single-layer fractal structure of [1], it reduces budget by half without using extra pebbles; total $\lceil \log n \rceil$ pebbles and $\lceil \frac{1}{2} \log n \rceil$ budget are needed. In terms of the budget-times-storage complexity, our algorithm of [5] is the best hash chain traversal algorithm.

Manuscript received November 11, 2009.
[†]The authors are with the Department of Electronic and Electrical Engineering, POSTECH, Pohang, Kyungbuk, 790–784, Republic of Korea.

[††]The author is with the Division of Advanced Industrial Science & Technology, DGIST, Daegu, 704–230, Republic of Korea.

a) E-mail: dhyum@postech.ac.kr
DOI: 10.1587/transinf.E93.D.1303

*Note that Sungwook Eom was a co-author of the CT-RSA paper, while Kookrae Cho joins in this work.

Algorithm 1 Traversal algorithm $\mathcal{T}$

Input: $(n, l, h(\cdot), \vec{p})$
$\mathcal{T}_{\mathsf{setup}}\,(n, l, h(\cdot), \vec{p})$;
$v = \perp$;
for $c = 1$ to $c = n$ **do**
  $\mathcal{T}_{\mathsf{traversal}}\,(n, h(\cdot), \vec{p}, c, v)$;
  output v;
end for
halt;

Algorithm 2 Subroutine $\mathcal{T}_{\mathsf{setup}}$

Input: $(n, l, h(\cdot), \vec{p})$
$\kappa \leftarrow \log n$;
$v_n \stackrel{R}{\leftarrow} \{0, 1\}^l$;
$v_0 \leftarrow h^n(v_n)$;
$i \leftarrow 1$;
while $i \leq \kappa$ **do**
  $p_i.\mathsf{position} \leftarrow 2^i$;
  $p_i.\mathsf{destination} \leftarrow p_i.\mathsf{position}$;
  $p_i.\mathsf{value} \leftarrow h^{n-2^i}(v_n)$;
  $p_i.\mathsf{status} \leftarrow \mathit{arrived}$;
  $i \leftarrow i + 1$;
end while
halt;

As all previous hash chain traversal algorithms have the output interval $m = 1$, we can think of a traversal algorithm $\mathcal{T}$ as a combination of two subroutines $\mathcal{T}_{\mathsf{setup}}$ and $\mathcal{T}_{\mathsf{traversal}}$. As an example, we present the traversal algorithm of [5] in Algorithm 1 with subroutines in Algorithm 2 and Algorithm 3, where n is the length of the hash chain, l is a security parameter, $h(\cdot)$ is a hash function, and $\vec{p}$ is an array of pebbles. $\mathcal{T}_{\mathsf{setup}}$ stores in pebbles some carefully chosen hash values of the chain including the secret value v_n. Each time $\mathcal{T}_{\mathsf{traversal}}$ is executed, it outputs the next hash value and rearranges pebbles to facilitate future computations. The performance of Algorithm 1 is as follows.

Theorem 1 ([5]): Algorithm 1 can traverse a hash chain of length $n = 2^\gamma$ for an integer γ, with $\lceil \frac{1}{2} \log n \rceil$ budget and $\log n$ pebbles.

For details on the algorithm and its analysis, please refer to [5].

3. Hash Chain Traversal for $m \geq 1$

In this section, we build a generalized hash chain traversal algorithm $\mathcal{GT}$ with an output interval $m \geq 1$ based on a traversal algorithm $\mathcal{T}$ with $m = 1$. For simplicity, we let $n = 2^\gamma = km$ for some integers γ, k and use Algorithm 1 as $\mathcal{T}$.

Firstly, to construct a generalized hash chain traversal algorithm $\mathcal{GT}_1$, we can use the traversal algorithm $\mathcal{T}$ directly by extracting every m-th hash value (i.e., $v_m, v_{2m}, v_{3m}, \ldots$) from the outputs of $\mathcal{T}$; algorithm 4 formally describes the idea. An example run of $\mathcal{GT}_1$ for $n = 16$ and $m = 4$ is given in Fig. 1, which is based on the corresponding example of $\mathcal{T}$ in [5]; we only change C to V from the

Algorithm 3 Subroutine $\mathcal{T}_{\mathsf{traversal}}$

Input: $(n, h(\cdot), \vec{p}, c, v)$
1: $\kappa \leftarrow \log n$;
2: **if** $c > n$ **then**
3:   halt;
4: **else**
5:   $\mathit{available} \leftarrow \lceil \frac{\kappa}{2} \rceil$;
6: **end if**
7: $i \leftarrow \mathcal{P}(\vec{p}, \mathit{arrived}, 0)$;
8: **if** $c \bmod 2 = 1$ **then**
9:   $v \leftarrow h(p_i.\mathsf{value})$;
10:   $\mathit{available} \leftarrow \mathit{available} - 1$;
11: **else**
12:   $v \leftarrow p_i.\mathsf{value}$;
13:   $p_i.\mathsf{position} \leftarrow p_i.\mathsf{position} + 3 \cdot 2^i$;
14:   $p_i.\mathsf{destination} \leftarrow p_i.\mathsf{destination} + 2 \cdot 2^i$;
15:   $p_i.\mathsf{value} \leftarrow \perp$;
16:   $p_i.\mathsf{status} \leftarrow \mathit{ready}$;
17:   $j \leftarrow \mathcal{P}(\vec{p}, \mathit{arrived}, p_i.\mathsf{position})$;
18:   **if** $j \neq \perp$ **then**
19:     $p_i.\mathsf{value} \leftarrow p_j.\mathsf{value}$;
20:     $p_i.\mathsf{status} \leftarrow \mathit{active}$;
21:   **end if**
22: **end if**
23: $i \leftarrow \mathcal{P}(\mathit{active}, 0)$;
24: **if** $i = \perp$ **then**
25:   halt;
26: **end if**
27: **while** $\mathit{available} > 0$ **do**
28:   $p_i.\mathsf{position} \leftarrow p_i.\mathsf{position} - 1$;
29:   $p_i.\mathsf{value} \leftarrow h(p_i.\mathsf{value})$;
30:   $\mathit{available} \leftarrow \mathit{available} - 1$;
31:   **if** $p_i.\mathsf{position} = p_i.\mathsf{destination}$ **then**
32:     $p_i.\mathsf{status} \leftarrow \mathit{arrived}$;
33:     $j \leftarrow \mathcal{P}(\vec{p}, \mathit{ready}, p_i.\mathsf{destination})$;
34:     **if** $j \neq \perp$ **then**
35:       $p_j.\mathsf{value} \leftarrow p_i.\mathsf{value}$;
36:       $p_j.\mathsf{status} \leftarrow \mathit{active}$;
37:     **end if**
38:     go to line 23;
39:   **end if**
40: **end while**
41: halt;

example of [5] if the round is a multiple of m (= 4).

The storage requirement of $\mathcal{GT}_1$ is the same as that of $\mathcal{T}$, i.e., $\log n$ pebbles. To compute each output hash value v_{jm} where $1 \leq j \leq k$, $\mathcal{GT}_1$ needs m executions of $\mathcal{T}_{\mathsf{traversal}}$. Therefore, the budget of $\mathcal{GT}_1$ is $m \cdot \lceil \frac{1}{2} \log n \rceil$ evaluations of $h(\cdot)$. Theorem 2 follows easily.

Theorem 2: $\mathcal{GT}_1$ can traverse a hash chain of length $n = 2^\gamma = km$ for integers γ, k, and the output interval m, with $m \cdot \lceil \frac{1}{2} \log n \rceil$ budget and $\log n$ pebbles.

To build a more efficient traversal algorithm, we should remove unnecessary computations from $\mathcal{GT}_1$. Let us explain how to simplify the traversal algorithm with the previous example of Fig. 1. Basically, we introduce four improvements. First, the pebble P_1, which is marked by ◯ in Fig. 2, is not necessary in the setup stage, because we do not output v_1 or v_2. Second, we may compute v_4 from v_8 instead of storing v_4 in the setup stage. This technique, which does not increase budget while reducing a pebble, was also used

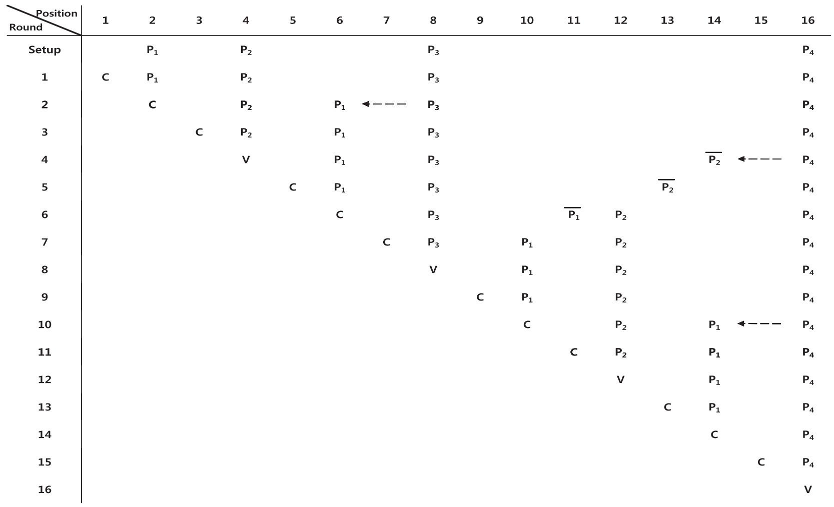

Fig. 1 Generalized traversal algorithm $\mathcal{GT}_1$.

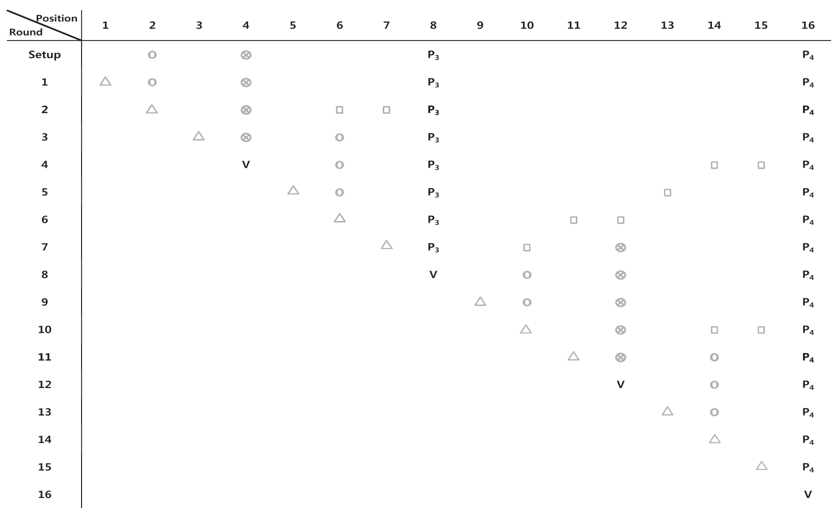

Fig. 2 Simplification of $\mathcal{GT}_1$.

Algorithm 4 $\mathcal{GT}_1$

Input: $(n, l, h(\cdot), \vec{p}, m)$

$\mathcal{T}_{\text{setup}}\,(n, l, h(\cdot), \vec{p})$;
$v = \perp$;
for $c = 1$ to $c = n$ **do**
  $\mathcal{T}_{\text{traversal}}\,(n, h(\cdot), \vec{p}, c, v)$;
  if $c \equiv 0 \pmod{m}$ **then**
    output v;
  end if
end for
halt;

in previous works [1], [2], [5]. Therefore, P_2 marked by $\otimes$ in Fig. 2 can be removed. Third, as we do not output hash values v_i for $i \not\equiv 0 \pmod 4$, the computations marked by $\triangle$ in Fig. 2 can be omitted. Finally, rearrangement of pebbles can be simplified by eliminating computations related to v_i for $i \not\equiv 0 \pmod 4$. That is, the computations marked by $\square$ in Fig. 2 can also be omitted.

Fortunately, these four simplifications can be applied without modifying the inner codes of $\mathcal{T}_{\text{setup}}$ or $\mathcal{T}_{\text{traversal}}$. Let us denote $h'(\cdot) \doteq h^m(\cdot)$ and define an auxiliary hash chain $H_{\text{aux}} = \langle u_1, u_2, u_3 \ldots, u_k \rangle$ of length $\frac{n}{m} (= k)$ with respect to $h'(\cdot)$. Note that $h'(\cdot)$ is only for notational convenience. If we set $u_k = v_n (= v_{km})$, we have $H_{\text{aux}} = \langle u_1, u_2, u_3, \ldots, u_k \rangle = \langle v_m, v_{2m}, v_{3m}, \ldots, v_{km} \rangle$ because each element u_i of H_{aux} satisfies $u_i = h'(u_{i+1}) = h^m(u_{i+1})$. Therefore, we can design an improved generalized traversal algorithm $\mathcal{GT}_2$ as Algorithm 5.

The storage requirement of $\mathcal{GT}_2$ is that of $\mathcal{T}$ with respect to the auxiliary hash chain H' of length $\frac{n}{m}$, i.e., $\lceil \log \frac{n}{m} \rceil$ pebbles. To compute each output hash value v_{jm} where $1 \le j \le k$, $\mathcal{GT}_2$ executes the underlying traversal algorithm $\mathcal{T}_{\text{traversal}}$ only one time and thus the budget is $\lceil \frac{1}{2} \log \frac{n}{m} \rceil$ evaluations of $h'(\cdot)$ or equivalently $m \cdot \lceil \frac{1}{2} \log \frac{n}{m} \rceil$ evaluations of $h(\cdot)$. Consequently, Theorem 3 follows easily.

Theorem 3: $\mathcal{GT}_2$ of Algorithm 5 can traverse a hash chain

Algorithm 5 $\mathcal{GT}_2$

Input: $(n, l, h(\cdot), \vec{p}, m)$

1: Define $h'(\cdot)$ as $h^m(\cdot)$;
2: $\mathcal{T}_{\mathsf{setup}}\left(\frac{n}{m}, l, h'(\cdot), \vec{p}\right)$;
3: $v = \perp$;
4: **for** $c = 1$ to $c = \frac{n}{m}$ **do**
5: $\quad\mathcal{T}_{\mathsf{traversal}}\left(\frac{n}{m}, h'(\cdot), \vec{p}, c, v\right)$;
6: $\quad$output v;
7: **end for**
8: halt;

of length $n = 2^\gamma = km$ for integers γ, k, and the output interval m, with $m \cdot \lceil \frac{1}{2} \log \frac{n}{m} \rceil$ budget and $\lceil \log \frac{n}{m} \rceil$ pebbles.

For example, if the SEAD routing protocol [7] with a 16-bit sequence number and maximum distance 16 hops between routers implements $\mathcal{GT}_1$ and $\mathcal{GT}_2$, then $\mathcal{GT}_1$ requires $\lceil \log 2^{16} \rceil = 16$ pebbles and $16 \cdot \lceil \frac{1}{2} \log 2^{16} \rceil = 128$ budget and $\mathcal{GT}_2$ requires $\lceil \log \frac{2^{16}}{16} \rceil = 12$ pebbles and $16 \cdot \lceil \frac{1}{2} \log \frac{2^{16}}{16} \rceil = 96$ budget.

Generally, if $\mathcal{T}$ requires $p(n)$ pebbles and $b(n)$ budget with respect to a hash chain of length n, then $\mathcal{GT}_1$ needs $p(n)$ pebbles and $m \cdot b(n)$ budget and $\mathcal{GT}_2$ needs $p(\frac{n}{m})$ pebbles and $m \cdot b(\frac{n}{m})$ budget.

Remark. Sella [4] proposed a scalable hash chain traversal algorithm that traverses a hash chain of length n with budget b, where b is a constant unrelated to n. Kim [3] reduced the storage requirement of Sella's algorithm slightly by $\frac{n^{1/(b+1)}-1}{n^{1/(b+1)}}$. If the Sella algorithm requires $p(n, b)$ pebbles for a given b, the transformed traversal algorithm by $\mathcal{GT}_2$ uses $p(\frac{n}{m}, b)$ pebbles.

4. Conclusion

We, for the first time, studied the hash chain traversal algorithm that selectively outputs hash values. We constructed an efficient hash chain traversal algorithm with the output interval $m \geq 1$ based on previous traversal algorithms with $m = 1$. As the proposed transformation is generic, it can be directly applied to known traversal algorithms. We leave a non-black-box approach to design more efficient generalized traversal algorithm as an open problem.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2009-0075147), the Deagu Gyeongbuk institute of science and technology (DGIST) Basic Research Program of the MEST, and BK21.

References

[1] M. Jakobsson, "Fractal hash sequence representation and traversal," IEEE International Symposium on Information Theory, pp.437–444, 2002. Available at IACR ePrint Archive, Report 2002/001, http://eprint.iacr.org/
[2] D. Coppersmith and M. Jakobsson, "Almost optimal hash sequence traversal," Financial Cryptography 2002, Lect. Notes Comput. Sci., vol.2357, pp.102–119, Springer, 2002.
[3] S.R. Kim, "Improved scalable hash chain traversal," ACNS 2003, Lect. Notes Comput. Sci., vol.2846, pp.86–95, Springer, 2003.
[4] Y. Sella, "On the computation-storage trade-offs of hash chain traversal," Financial Cryptography 2003, Lect. Notes Comput. Sci., vol.2742, pp.270–285, Springer, 2003.
[5] D.H. Yum, J.W. Seo, S. Eom, and P.J. Lee, "Single-layer fractal hash chain traversal with almost optimal complexity," CT-RSA 2009, Lect. Notes Comput. Sci., vol.5473, pp.325–339, Springer, 2009.
[6] N. Haller, "The s/key one-time password system," RFC 1760, Internet Engineering Task Force, 1995.
[7] Y.C. Hu, D.B. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," Ad Hoc Networks, vol.1, no.1, pp.175–192, 2003.
[8] A. Perrig, R. Canetti, J.D. Tygar, and D.X. Song, "Efficient authentication and signing of multicast streams over lossy channels," IEEE Symposium on Security and Privacy, pp.56–73, IEEE Computer Society, 2000.
[9] R.L. Rivest and A. Shamir, "Payword and micromint: Two simple micropayment schemes," Security Protocols Workshop, Lect. Notes Comput. Sci., vol.1189, pp.69–87, Springer, 1996.
[10] S.G. Stubblebine and P.F. Syverson, "Fair on-line auctions without special trusted parties," Financial Cryptography 1999, Lect. Notes Comput. Sci., vol.1648, pp.230–240, Springer, 1999.
[11] G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying," CRYPTO 2001, Lect. Notes Comput. Sci., vol.2139, pp.332–354, Springer, 2001.