Construction and Traversal of Hash Chain with Public Links

Vipul Goyal
OSP Global
Mumbai, India
vgoyal@ospglobal.com

Abstract

Current hash chain traversal techniques require that the intermediate links of the hash chain be stored secretly on a trusted storage. This requirement is undesirable in several applications. We propose a new construction of hash chains based on inserting a 'breakpoint' after fixed number of links in the chain. We also propose a method with which the current hash chain traversal techniques can be applied to our construction without any significant changes in the storage and computation requirements and with the added advantage that the intermediate links may be stored on a public and non-trusted storage. We are also able to prove the security of our construction by replacing the hash function with a MAC function.

1 Introduction

The idea of hash chains was first proposed by Lamport [1] to design an authentication system called one time passwords (OTPs), which would be safe from both eavesdropping as well as server database compromise. Since then it has been employed extensively in a wide range of cryptographic systems, many of them proposed only recently. Hash chains have interesting public key cryptography like properties while employing nothing more than a fast one way hash function (OWHF).

Examples of system build upon the concept of hash chains include password based authentication [1, 3], micropayments [19, 38, 39], server-supported signatures [35, 36], certificate revocation [10, 37], secure address resolution [21], online auctions [14], digital cash [40], web click hit shaving [22], efficient multicasting [11-13], spam fighting protocols [41], one time signature schemes [42], sensor network security protocols [48-49] and securing routing information[43-47].

Hash chains are generated by applying a one way hash function recursively to a seed value. The chain is used by exposing its links one by one in the reverse order. For this, unless all the links are stored, the next link to be exposed should be computed when required. This is done efficiently using hash chain traversal techniques. A number of traversal techniques exist [50-53] which trade space for time and are efficient and practical for most applications. In particular, the solution presented in [52] has been proved to be an almost optimal solution to the traversal problem.

All these traversal technique require some intermediate hash chain link values to be stored secretly. This may be an unrealistic assumption in some applications. Consider the original application of hash chains, i.e., one time passwords (OTPs). Devised in 1981 [1], OTPs are implemented, standardized [2, 3, 6, 7, 8] and widely used. In OTPs, the hash chain is generated by the user using the password as the seed. The user authenticates to the server by revealing links of the hash chain one by one. This system is secure against server compromise (or server fraud) in the sense that no security sensitive quantities are stored on the server. Now, for using the current hash chain traversal techniques in order to efficiently compute the successive links, the user must

secretly store a number of intermediate hash chain links. However, this is not possible since in all 'password based authentication systems', the user is assumed to be stateless and cannot be assumed to store anything. The only thing a user is required for logging in is the password. Hence, the hash chain traversal techniques are not applicable to this system. Consequently, for authentication, the user must compute the successive link values starting with the seed, i.e., the password itself, which is computationally intensive. Clearly, the system could be significantly more efficient if the secrecy of the intermediate links was not required. In that case, the server would store those links on behalf of the user and would supply them to the user at the time of authentication.

In general, these traversal techniques cannot be applied when the hash chain generator does not have a secure and trusted non-volatile memory for secretly storing the intermediate links. Further, when the hash chain generator is a mobile device having limited storage [35, 36, 48, 49], it may be desirable to use alternate storage, like hash chain verifier's database, whenever possible.

In applications like micropayments [19, 38, 39], a user typically establishes payment chains simultaneously with many vendors, e.g., to use web services provided by several different vendors or to repeatedly view web pages of many paid websites. This means that a hash chain generator has to deal with and hence store intermediate links of several hash chains at the same time. The system could have new possibilities if each vendor server had stored the intermediate links of the hash chain meant for itself on behalf of the user. In that case, the user could generate all the hash chain seeds pseudorandomly with a master secret such as with the key embedded in his smartcard and could utilize any (possibly untrusted) terminal in the world to continue payments to any vendor. This kind of setting is also preferable in cases where the user is a mobile device.

We introduce a new construction of hash chains. The basic idea is insert a number of breakpoints after fixed distances in the hash chain. A breakpoint is inserted at a link by re-seeding the hash chain at that link with a derivative of the seed value. The links at which a breakpoint is inserted may be made public and are no longer considered to be security sensitive. Further, it turns out that there is a method with which any existing hash train traversal technique may be applied to our construction also, without any significant changes in the storage and computation complexity. Thus, with our construction, effectively, without compromising efficiency, we remove the restriction that the intermediate links be secretly stored.

It should be stated here that our construction is not meant to replace regular hash chain construction in all applications. In many of the system like certificate revocation [10, 37], secure address resolution [21] and multicasting [11-13] etc, the present construction and traversal techniques work well since the hash chain generator is not expected to have trusted storage constraints. Further, there is no fixed verifier (or obvious third party) whose memory may be used for storing the intermediate links of the hash chain.

Rest of the paper is organized as follows- Section 2 gives a background on hash chains and hash chain traversal techniques, section 3 describes the proposed construction of hash chains, section 4 describes the application of present traversal techniques to our construction, section 5 computes the optimal number of links between two breakpoints for minimizing the computation complexity, section 6 provides the security proof for the optimal construction by replacing the hash function with a MAC function, section 7 concludes the paper.

2 Hash Chain Construction and Traversal Techniques

Hash chains are based a public function H that is easy to compute but computationally infeasible to invert, for suitable definitions of "easy" and "infeasible". Such functions are called one-way functions (OWF) and were first employed for use in login procedures by Needham [27]. If the

output of a one-way function is of fixed length, it is called a one-way hash function (OWHF). More precisely, the definition of a OWHF is given as [26]-

Definition A function H that maps bit strings, either of an arbitrary length or a predetermined length, to strings of a fixed length is a OWHF if it satisfies two additional properties-
→ Given *x*, it is easy to compute $H(x)$
→ Given $H(x)$, it is hard to compute *x*

A hash chain of length *N* is constructed by applying a one-way hash function $H(.)$ recursively to an initial seed value *s*.

$$H^N(s) = \underbrace{H(H(...H(s)))}_{\text{N times}}$$

The last element $H^N(s)$ also called the tip *T* of the hash chain resembles the public key in public key cryptography i.e., by knowing $H^N(s)$ but not *s*, $H^{N-1}(s)$ can not be computed but its correctness can be verified. This property of hash chains has directly evolved from the property of one-way hash functions.

In the hash-chain applications, first the tip *T* or $H^N(s)$ is securely distributed to the verifying party (or parties) and then the links of the hash chain are spent (or used) one by one in the reverse order starting from $H^{N-1}(s)$ and continuing until the value of *s* is reached. In general, the i^{th} link of the hash chain is $H^i(s)$ and can be verified for correctness by the verifying party by hashing it once and comparing with the $(i+1)^{th}$ link.

Straightforward methods of computing the hash chain links require either too much storage or too much computation. A trivial solution to this problem could be to re-compute the next link whenever required directly from the seed *s*. Clearly, this requires $O(N)$ hash function evaluations per link computation. Another trivial solution would store all the links of the chain, and would plainly perform a lookup for each link to be output. Such a solution would have a memory complexity of $O(N)$. One could easily trade memory and storage against each other in these trivial solutions by storing some fraction of the values, and computing the next required link from the nearest stored link. It can be seen that all such variations of these trivial approaches will have a memory-times computational complexity of $O(N)$.

Jakobsson initiated the study of this problem and proposed a traversal technique [50, 51] to reduce the computation-times-storage complexity to $O\left(\log_2{}^2(N)\right)$. This was accomplished by the introduction of a technique in which a constantly changing set of intermediary hash chain links are stored. The computation as well as storage complexity is this technique is $\lceil \log_2(N) \rceil$. This technique was improved by Coppersmith and Jakobsson [52] who proposed a new idea which allows for a reduction of the computational requirements to slightly less than half of Jakobsson's, while slightly reducing the storage demands for most practical parameter choices. The computational costs of this solution are approximately $\frac{1}{2}\log_2(N)$ hash function evaluations, using only a little more than $\log_2(N)$ storage cells. An interesting result in this work was the lower bound provided for the optimal solution of this problem and the fact that their technique is very close to the optimal solution. Recently, Sella [53] studied the traversal problem with a constant bound, m, on the number of hash function evaluations allowed per link usage of the hash chain. The result is a generalized technique having storage requirements $k\sqrt[k]{N}$ where $k = m + 1$.

All these techniques require the storage of some intermediate hash chain links. These links should be stored secretly as the exposure of the i^{th} link in the chain would enable an adversary to compute all the links from $(i+1)$ to N just by recursive hashing.

3 The Proposed Hash Chain Construction

In this section, we describe the proposed construction of hash chains. As discussed before, the basic idea is to insert a 'breakpoint' after fixed number of links in the chain. This is done by reseeding the hash chain with a reseeding factor (RF) which is a derivative of the seed value s. Before going further, we introduce a few basic notations to be used in this paper.

$H(X)$	Output of a hash function applied to X. We discover the exact propoerties required from *H* in section 6. For now, it suffices to assume that *H* is a one way hash function.
$< X,Y >$	Concatenation of *X* and *Y*
$H(X,Y)$	$H(< X,Y >)$
D	The distance between two breakpoints in the hash chain. The minimum value of *D* is 1 (when a breakpoint is inserted at every link in the chain). We compute the optimal value of *D* in section 5.
N	Total number of links in the hash chain

The hash chain generator first chooses a seed *s* with which the chain is generated. *s* should be kept secret by the generator, e.g., by memorizing or writing it down as a password or by storing it if a limited trusted storage is available. *s* may also be generated pseudorandomly from a master secret which is used for multiple purposes. In that case, there is no need of storing the seed *s* and multiple hash chain seeds may be generated from the same master secret. This kind of setting is preferable in applications like micropayments [19, 38, 39] as discussed before.

The length *N* of the chain is selected by choosing a parameter *n*. We have $N = n \times D$. Define a function ω as follows-

$$\omega_D^{i+1}(s) = H\left(\omega_D^i(s), s_i\right) \qquad 0 \le i \le (N-1)$$

$$\omega_D^0(s) = s$$

$$s_i = \begin{cases} H(s,j) & \text{if there exists an integer } j, \text{ s.t., } j \times D = i, 0 \le j \le (N-1) \\ null & \text{otherwise} \end{cases}$$

Where $null$ is a string of length zero, i.e., $null \in \{0,1\}^0$. The above function ω describes the proposed hash chain construction, $\omega_D^N(s)$ being the tip of the hash chain. Observe that the chain is being reseeded for $i = 0, D, 2D, ..., (N-1)D$ with the reseeding factor s_i.

Note that an important security property of our construction is that the links $\omega_D^i(s)$, with *i* being a multiple of D, are not security sensitive and may be made public. This is because with the knowledge of $\omega_D^i(s)$, an adversary can neither compute $\omega_D^{i-1}(s)$ (one way-ness of the hash function) nor $\omega_D^{i+1}(s)$ (knowledge of s_i, and hence that of the seed s required).

The proposed chain is used by exposing its links $\omega_D^i(s)$ one by one in the reverse order as in ordinary hash chains, the only difference being for links where i is a multiple of D. Such links require the exposure of the reseeding factor s_i along with themselves to enable verification.

This completes our description of the proposed hash chain construction. Though we will call our chain to be a hash chain in a general sense, we will denote it by ω-chain and an ordinary hash chain by *H*-chain wherever we need to distinguish between the two.

4 Applying Present Hash Chain Traversal Techniques to Our Construction

In this section, we discuss the application of present hash chain traversal techniques to our construction, i.e., the application of *H*-chain traversal techniques (HCTT) to ω-chain, and discuss the computational and storage complexity changes if any. Note that HCTT cannot be directly applied to ω-chain since we require all the stored intermediate links to be non-security sensitive.

We model a *H*-chain traversal technique by the following-

$S_i(s,N)$	Set of stored links after the i^{th} link of the *H*-chain with seed s and length N has been exposed or used
$C_{i-1}(s,N,S_i(s,N))$	Denotes the algorithm for the computation of the $(i-1)^{th}$ link of the *H*-chain with seed s and length N using $S_i(s,N)$. The algorithm returns the $(i-1)^{th}$ link and also outputs the new set of links to be stored $S_{i-1}(s,N)$
$SC(N)$	Storage complexity (i.e., the number of intermediate links stored) of the HCTT for a *H*-chain of length N. Equal to the average number of elements in $S_i(s,N)$ for all i.
$CC(N)$	Computational complexity (i.e., the number of hash function evaluations required per successive link computation) of the HCTT for a *H*-chain of length N. Equal to the average number of hash function evaluations made by $C_i(s,N,S_{i+1}(s,N))$ for all i.

We stress that the above parameters can be defined for any HCTT. Consequently, our method for applying HCTT to ω-chains is generic and is independent of the actual working of the HCTT.

To apply a HCTT to a ω-chain with seed s and having length $N = n \times D$, we define a γ-chain of length n as follows-

$$\gamma_D^{i+1}(s) = H^D\left(\gamma_D^i(s), H(s,i)\right) \qquad 0 \le i \le (n-1)$$

$$\gamma_D^0(s) = s$$

Observe that the links of the γ-chain are actually the same as those links of the ω-chain which are non-security sensitive. In particular, we have $\gamma_D^i(s) = \omega_D^{iD}(s)$. The non security sensitivity of the γ-chain links is also suggested by the fact that a γ-function evaluation can only be done by the γ-chain generator since the evaluation will require the knowledge of seed *s* to compute the reseeding factor $H(s,i)$. Hence, it follows that neither the link $\gamma_D^{i-1}(s)$, nor $\gamma_D^{i+1}(s)$ can be computed from a given link $\gamma_D^i(s)$.

To apply the HCTT to a ω-chain of length N, we actually apply it to the corresponding γ-chain of length $n = N/D$. All the hash function evaluation made by algorithm C are replaced by γ function evaluations. This results in a traversal technique in which the i^{th} link of the γ-chain is computed as follows-

$$\gamma_D^i(s) = C_i'\left(s, n, S_{i+1}'(s,n)\right)$$

Where C_i' denotes the algorithm obtained by replacing all the hash function evaluations in algorithm C_i by γ function evaluations. S_i' denotes the set of intermediate links output by C_i'.

The security property we achieve through this technique is that the intermediate links in the set S_i' are non-security sensitive since all of them will be the links of the γ-chain. Hence, the

chain generator may store these links at a public and non-trusted storage. In order to compute the next ω-chain link, the generator will read the link set stored at the public storage, will compute the appropriate γ-chain link from it by running $C_i^{'}$ and will finally compute the required ω-chain link from the obtained γ-chain link. More precisely, the computation of the link $i = j.D + k, k < D$ of the ω-chain, i.e. $\omega_D^i(s)$, is done as follows-

$$\gamma_D^j(s) = C_j^{'}\left(s, n, S_{j+1}^{'}(s,n)\right)$$

Since $\gamma_D^i(s) = \omega_D^{iD}(s)$, we have

$$\omega_D^i(s) = H^k\left(\gamma_D^j(s), H(s,j)\right)$$

Note that it is possible to do the above computation on a non-trusted terminal with the aid of a smartcard holding the seed value (or the master key with which the seed is generated). The terminal passes all the γ function evaluation calls to the smartcard which, without any checks, does the required evaluations and passes the result back to the terminal. However, any reseeding factor is not revealed by the card at this stage. For the second part of the computation, the smartcard prompts the user for the pin number and only then computes $\omega_D^i(s)$ from $\gamma_D^j(s)$. If $k = 0$, the smartcard also reveals $H(s,j)$.

For verification, the chain generator exposes $\omega_D^i(s)$. If $k = 0$, i.e., i is a multiple of D, the reseeding factor s_i (or $H(s,j)$) is also exposed. Further, if $k = 0$, the stored set of links $S_{j+1}^{'}(s,n)$ is replaced by the new set $S_j^{'}(s,n)$ output by $C_j^{'}\left(s, n, S_{j+1}^{'}(s,n)\right)$. The complexities of the resulting traversal technique may be computed as follows-

STORAGE COMPLEXITY

Since the traversal technique is actually applied on a chain of length n, we have

$$SC^{'}(N) = SC(n) = SC\left(N/D\right)$$

COMPUTATIONAL COMPLEXITY

$CC^{'}(N)$ = Computing $\gamma_D^j(s)$ by running $C_j^{'}\left(s, n, S_{j-1}^{'}(s,n)\right)$ + computing reseeding factor $H(s,j)$ + computing $\omega_D^i(s)$ using $\gamma_D^j(s)$ and $H(s,j)$

$$CC^{'}(N) = (D+1).CC\left(N/D\right) + 1 + k$$

The factor $(D+1)$ with $CC\left(N/D\right)$ is because of the fact that each call to hash function *H* is replaced by a γ-function whose evaluation requires $(D+1)$ *H* functions evaluations.

5 The Optimal Construction

In this section, we discover the optimal distance D between two breakpoints in the ω-chain for the HCTT by coppersmith et al [52] shown to be almost optimal by them. Given the ω-chain length N, our goal will be to minimize the average computation complexity as found in the previous section

$$CC^{'}(N) = (D+1).CC\left(N/D\right) + 1 + k$$

Clearly, k varies uniformly from 0 to $(D-1)$, the average value being $(D-1)/2$. Further, $CC(.) = \left(1/2\right)\log(.)$ for HCTT in [52]. Hence, we have

$$CC'(N) = \frac{D+1}{2}.\log\left(N/D\right) + \frac{D-1}{2} + 1$$
$$= \frac{D+1}{2}.\log\left(N/D\right) + \frac{D+1}{2}$$

There is no minimum for the above expression due to the factor $(D+1)/2$ .The expression may potentially vary from $-\infty$ to $+\infty$. However, we are only interested in the +ve integral values of D . The minima can be found out with a simple analysis with various values of D . We rewrite the above equation as

$$CC'(N) = \underbrace{(\log(N)+1)}_{\text{factor 1}} + \underbrace{\left(\frac{D}{2}.\log\left(\frac{N}{D}\right) + \frac{D-1}{2} - \frac{1}{2}.\log(ND)\right)}_{\text{factor 2}}$$

It can be shown that the second factor is always +ve for $D > 2$ as well as for $D = 2$ and $N \geq 4$. Further, this factor is zero for $D = 1$. This means for $CC'(N)$ to be minimum, we have

$$D = \begin{cases} 1 & \text{for } N \geq 4 \\ 2 & \text{otherwise} \end{cases}$$

The above values clearly depict that for the interesting values of N , the optimal distance D between two breakpoints is one[1]. Hence, a value of D may be set as the constant for ω -chain with [52] as the HCTT.

$$D = 1$$

Thus, for the optimal construction, the ω -chain coincides with the γ -chain.

6 Security Analysis

In this section, we analyze the security of our optimal ω -chain construction, i.e., with $D = 1$. The chain can be defined as follows

$$\omega_1^{i+1}(s) = H\left(\omega_1^i(s), s_i\right) \qquad 0 \leq i \leq (N-1)$$
$$\omega_1^0(s) = s\,,\; s_i = H(s,i)$$

The links $\omega_1^i(s)$ of the ω -chain are not considered to be security sensitive. The reseeding factor (RF) s_i should be revealed along with the link $\omega_1^i(s)$ to enable verification and is the basis of the security of our construction. Clearly, our security is based on the assumption that with the knowledge of $H(X,Y)$ as well as X , an adversary is not able to compute Y with a non-negligible probability. However, the definition and the properties of hash functions are usually analyzed with a single argument $H(X)$ and the above assumption is not proved or widely supported in the literature. To overcome this fact, we replace *H* with a *MAC* function and redefine the chain construction as follows-

$$\omega_1^{i+1}(s) = MAC_{s_i}\left(\omega_1^i(s)\right) \qquad 0 \leq i \leq (N-1)$$
$$\omega_1^0(s) = s$$

We will call the above chain to be a modified ω -chain ($M\omega C$). To introduce provable security, we assume that the reseeding factors s_i 's are random. Later, we instantiate s_i with a keyed pseudorandom number generator like a hash function.

[1] Traversal techniques are applied to efficiently compute the successive hash chain links. For a hash chain of length 3 or less, talking about optimization of the traversal technique used makes little sense since the successive hash chain links may even be computed directly from the seed itself. Hence, for *N<4*, *D=1* is equally acceptable.

The security of a MAC function has been defined as follows-

Definition 1 *We say that a MAC function MAC is secure if any probabilistic polynomial time algorithm $\mathcal{F}$, which is given as input the MACs $MAC_k(m_1), MAC_k(m_2), ..., MAC_k(m_q)$ of the adaptively chosen message $m_1, m_2, ..., m_q$ with key k, outputs a valid MAC $MAC_k(m)$ of a new previously unseen message m only with negligible probability.*

Observe that an adversary who recovers the key certainly breaks the *MAC* function. The *MAC* functions satisfying the above definition do exist. In particular, Bellare et al [54] designed NMAC and HMAC using a hash function as the underlying primitive. Both of them are provably secure if the hash function used satisfies some reasonable properties like weak collision resistance [54]. Further, their performance is essentially as good as that of the underlying hash function.

Now, we define and prove the security of our modified ω-chain construction.

Definition 2 *We say that a modified ω-chain is secure if any probabilistic polynomial time algorithm $\mathcal{F}$, which is given access to the $M\omega C$ links $\omega_1^i(s)$, $1 \le i \le N$ and is allowed to query a RF (reseeding factor) oracle for getting the reseeding factors s_i's one by one in the reverse order, outputs a valid and previously unseen reseeding factor only with a negligible probability.*

Theorem 1 If *MAC* is a secure MAC function, the resulting modified ω-chain $M\omega C$ is secure.

Proof

Assume that the thesis is false, i.e., that there is an algorithm $\mathcal{F}$, given as input $M\omega C$ links $\omega_1^i(s)$, $1 \le i \le N$, that succeds in computing a reseeding factor s_i by querying the RF oracle to get $s_j, s_{j+1}, ..., s_{N-1}$ with $i < j$, with a non-negligible probability ε. We show how $\mathcal{F}$ can be used to build an algorithm $\mathcal{F}_M$ which forges the MAC function.

Before going further, we state two results which we can prove in this theorem.

Result 1 $\mathcal{F}_M$ takes as input one MAC $MAC_k(m)$ of the message m of its choice and recovers the key k with probability ε/N.

Result 2 $\mathcal{F}_M$ takes as input one MAC each $MAC_{k_1}(m_1), MAC_{k_2}(m_2), ..., MAC_{k_N}(m_N)$ of the adaptively chosen messages $m_1, m_2, ..., m_N$ with secret keys $k_1, k_2, ..., k_N$, asks for a set of keys $k_j, k_{j+1}, ..., k_N$ to be revealed and then recovers a key k_i, $i < j$ with probability ε.

Result 2 seems to be a stronger result. While Result 1 implies a security loss of a factor N, Result 2 essentially implies that a MAC forgery can be done with the same probability as $M\omega C$ forgery. Result 2 means that a key recovery is done for at least one of the (message, MAC) pairs in a set of N such pairs. Further, it can easily be shown that Result 1 follows from Result 2. Hence, we choose to prove Result 2 in this theorem. The construction of $\mathcal{F}_M$ follows.

$\mathcal{F}_M$ randomly chooses the seed s of the $M\omega C$ chain and constructs the chain with the unknown keys $k_1, k_2, ..., k_N$ as the reseeding factors $s_0, s_1, ..., s_{N-1}$. This is done by asking for the MAC of one message of its choice with each of the keys. More precisely, the $M\omega C$ is

constructed by setting $\omega_1^0(s) = s$ and asking for the MAC of $\omega_1^i(s)$ with s_i (i.e., with k_{i+1}) to get $\omega_1^{i+1}(s)$.

Now, $\mathcal{F}_M$ runs $\mathcal{F}$ giving all the computed links $\omega_1^i(s)$'s for $i = 1$ to N as input. When $\mathcal{F}$ first queries the RF oracle, $\mathcal{F}_M$ asks for key k_N to be revealed and answers the query by supplying k_N as s_{N-1}. $\mathcal{F}_M$ does so for $s_{N-2}, s_{N-3}, ..., s_j$ for subsequent queries by $\mathcal{F}$ by asking for $k_{N-1}, k_{N-2}, ..., k_{j+1}$ to be revealed. Now, $\mathcal{F}$ outputs a reseeding factor s_i, $i < j$ with probability ε. This in turn means that $\mathcal{F}_M$ gets a key k_{i+1} which it did not ask to be revealed. Thus, $\mathcal{F}_M$ halts outputting k_{i+1} to demonstrate a successful key recovery attack.

Since the existence of $\mathcal{F}_M$ contradicts our hypothesis, the thesis must be true.

■

Remark 1 *As shown by $\mathcal{F}_M$, forging our $M\omega C$ construction requires the adversary to be capable of recovering the key just by seeing a single MAC. This attack is much stronger than forging MAC of a message by seeing MAC of several messages of its choice. Thus, even if the adversary is able to forge a MAC in the sense of definition 1, it is still possible that he may not be able to forge $M\omega C$ construction.*

Remark 2 *In the proof, we assume that the reseeding factors are random and independent of each other or the seed s. We may replace them with a keyed pseudorandom number generator like a hash function with s as the key. Although this obviously is secure with random oracle assumption, we stress that we require a weaker assumption than random oracle for security. Our security is in-tact even if a reseeding factor s_i is distinguishable from a pure random number but cannot be computed given the knowledge of other reseeding factors and the MAC of one chosen message with s_i as the key.*

Remark 3 *We have proved the security of the modified ω-chain obtained by replacing a H function with a MAC function. It is commonly believed that a H function with two arguments $H(X,Y)$ possesses a MAC function like properties. With this assumption, our ω-chain construction also seems to be secure. However, if the above assumption is undesirable, a $M\omega C$ may actually be used in practice instead of ω-chains. Our traversal technique and complexity analysis is also applicable to $M\omega C$. The results of Bellare et al [54] implies that this can be done with essentially negligible performance degradation.*

7 Conclusion

We remove the requirement that for using the hash chain traversal techniques, the intermediate hash chain links be stored secretly. This is done by slightly modifying the hash chain construction. Our technique expands the scope of hash chain traversal techniques to the applications where a secure and trusted non-volatile memory is not available to the hash chain generator. A possible example is one time passwords. Further, storage reduction is possible in cases where the generator is a mobile device. The generator may store the intermediate links with the hash chain verifier or on a third party storage. This is especially useful in systems like micropayments where a (possibly constrained) user may have to deal with several hash chains simultaneously. Our complexity analysis shows that there are no significant changes in storage and computational requirements. Further, we also prove the security of our optimal construction by replacing a hash function with a MAC function.

References

[1] L. Lamport, “Password Authentication with Insecure Communication”, *Communications of the ACM* 24.11 (November 1981), pp 770-772.

[2] D.L. McDonald, R.J. Atkinson, C. Metz "One-Time Passwords in Everything (OPIE): Experiences with Building and Using Strong Authentication," In *Proc. of the 5th USENIX UNIX Security Symposium*, June 1995.

[3] N Haller, "The S/KEY One-Time Password System", *Proceedings of the ISOC Symposium on Network and Distributed System Security*, pp 151-157, February 1994.

[4] G Tsudik, “Message authentication with one-way hash functions”, *ACM Computer Communications Review*, 22(5), 1992, pp 29-38.

[5] A. D. Rubin, “Independent One-Time Passwords”, *USENIX Journal of Computer Systems*, February 1996.

[6] N Haller, “The S/KEY One-Time Password System”, *RFC 1760*, February 1995. Available from http://www.ietf.org.

[7] N Haller, “A One-Time Password System”, *RFC 1938*, May 1996. Available from http://www.ietf.org.

[8] N Haller, C. Metz, P. Nesser and M. Straw, “A One-Time Password System”, *RFC 2289*, Feb 1998. Available from http://www.ietf.org.

[9] National Institute of Standards and Technology (NIST), "Announcing the Secure Hash Standard", *FIPS 180-1*, U.S. Department of Commerce, April 1995.

[10] S. Micali, "Eficient Certificate Revocation," *Proceedings of RSA '97*, and U.S. Patent No. 5,666,416.

[11] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Eficient and Secure Source Authentication for Multicast," *Proceedings of Network and Distributed System Security Symposium NDSS 2001*, February 2001.

[12] A. Perrig, R. Canetti, D. Song, and D. Tygar, "Eficient Authentication and Signing of Multicast Streams over Lossy Channels," *Proc. of IEEE Security and Privacy Symposium S & P 2000*, May 2000.

[13] A. Perrig, R. Canetti, D. Song, and D. Tygar, "TESLA: Multicast Source Authentication Transform", Proposed *IRTF draft*, http://paris.cs.berkeley.edu/ ~perrig/

[14] S. Stubblebine and P. Syverson, "Fair On-line Auctions Without Special Trusted Parties," *Financial Cryptography '01*.

[15] R Impagliazzo and S Rudich, "Limites on the Provable Consequances of one way permutations," *Proceedings of the 21st Annual Symposium on Theory of Computing*, ACM, 1989.

[16] S. Halevi and H. Krawczyk, “Public-Key Cryptography and Password Protocols,” *ACM Transactions on Information and System Security*, Vol. 2, No. 3, August 1999, Pages 230–268.

[17] W. Diffie, and M. E. Hellman, “New directions in cryptography”. *IEEE Transactions on Information Theory IT-11* (November 1976), 644–654.

[18] R. H. Morris, and K. Thompson, “Unix password security”, *Communications of the ACM* 22, 11 (November 1979), 594.

[19] R. L. Rivest and A. Shamir. PayWord and MicroMint-two simple micropayment schemes. In Mark Lomas, editor, *Proceedings of 1996 International Workshop on Security Protocols*, volume 1189, Lecture Notes in Computer Science, pages 69-87. Springer, 1997.

[20] R. L. Rivest, A. Shamir, and L. M. Adleman, “A method for obtaining digital signatures and public-key cryptosystems”, *Communications of the ACM*, 21(2):120–126, 1978.

[21] V. Goyal, “A Solution to the ARP Cache Poisoning Problem”, *manuscript*, June 2004.
[22] Michael K. Reiter, Vinod Anupam and Alain Mayer, "Detecting Hit Shaving in Click-Through Payment Schemes", *Proceedings of the 3rd USENIX Workshop on Electronic Commerce*, 1998, Pp. 155–166.
[23] P. T. Devanbu and S. Stubblebine. Stack and queue integrity on hostile platforms. In *Proceedings 1998 IEEE Symposium on Research in Security and Privacy*, May 1998. (Oakland).
[24] S. Even, O. Goldreich, and S. Micali, “On-line/off-line digital signatures”, *Crypto ‘89*, LNCS Vol No. 435, pp 263–277, Springer-Verlag, 1990.
[25] D.L. McDonald, R.J. Atkinson, C. Metz "One-Time Passwords in Everything (OPIE): Experiences with Building and Using Strong Authentication," In *Proc. of the 5th USENIX UNIX Security Symposium*, June 1995.
[26] T. Berson, L. Gong and M. Lomas, "Secure, Keyed and Collisionful Hash Function", *Technical Report, SRI International*, September 1994.
[27] M. V. Wilkes, “Time-Sharing Computer Systems”, *New York: Elsevier*, 1972.
[28] R.C. Merkle, A Digital Signature Based on a Conventional Encryption Function, *Proc. CRYPTO’87*, LNCS 293, Springer Verlag, 1987, pp 369-378.
[29] R.C. Merkle, A Certified Digital Signature, *Proc. CRYPTO’89*, LNCS 435, Springer Verlag, 1990, pp 218-238.
[30] D. Bleichenbacher and U.M. Maurer, Directed Acyclic Graphs, One-way Functions and Digital Signatures, *Proc. CRYPTO’94*, LNCS 839, Springer Verlag, 1994, pp 75-82.
[31] D. Bleichenbacher, U.M. Maurer, Optimal Tree-Based One-time Digital Signature Schemes, *Proc. STACS’96*, LNCS 1046, Springer-Verlag, pages: 363-374, 1996.
[32] D. Bleichenbacher, U.M. Maurer, On the efficiency of one-time digital signatures, *Proc. ASIACRYPT’96*, LNCS 1163. Springer-Verlag, pages: 145-158, 1996.
[33] K. Bicakci, G. Tsudik, B. Tung, How to construct optimal one-time signatures, *Computer Networks (Elsevier),* Vol.43(3), pp. 339-349, October 2003.
[34] M.Jakobsson and S.Wetzel, Secure Server-Aided Signature Generation. In *Proc.of the Workshop on Practice and Theory in Public Key Cryptography (PKC 2001),* LNCS No.1992, Springer, 2001.
[35] N.Asokan, G.Tsudik and M.Waidners, “Server-supported signatures”, *Journal of Computer Security*, November 1997.
[36] X.Ding, D.Mazzocchi and G.Tsudik. Experimenting with Server-Aided Signatures, *Network and Distributed Systems Security Symposium (NDSS ’02),* February 2002.
[37] William Aiello, Sachin Lodha, and Rafail Ostrovsky. Fast digital identity revocation. In Hugo Krawczyk, editor, *Advances in Cryptology – Crypto ’98*, volume 1462 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, 1998.
[38] Ralf Hauser, Michael Steiner, and Michael Waidner. Micro-payments based on iKP. In *14th Worldwide Congress on Computer and Communications Security Protection*, pages 67–82, C.N.I.T Paris-La Defense, France, June 1996.
[39] Torben Pryds Pedersen. Electronic payments of small amounts. Mark Lomas, editor. *Security Protocols—International Workshop,* volume 1189 of Lecture Notes in Computer Science, Cambridge, UK, April 1997. Springer-Verlag, Berlin Germany.
[40] Khanh Quoc Nguyen, Yi Mu, Vijay Varadharajan, “Digital Coins based on Hash Chain”.
[41] C. Dwork, A. Goldberg, and M. Naor. On Memory-Bound Funtions for Fighting Spam. *Advances in Cryptology - Crypto 2003*, Volume 2729 of Lecture Notes in Computer Science. Springer-Verlag, 2003.
[42] A. Perrig. The BiBa One-Time Signature and Broadcast Authentication Protocol. *Proceedings of the Annual Conference on Computer and Communications Security (CCS),* pages 28-37. ACM Press, 2001.

[43] Steven Cheung. An efficient message authentication scheme for link state routing. In *13th Annual Computer Security Applications Conference,* pages 90–98, 1997.

[44] Ralf Hauser, Antoni Przygienda, and Gene Tsudik. Reducing the cost of security in link state routing. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '97),* pages 93–99, San Diego, California, February 1997. Internet Society.

[45] Kan Zhang. Efficient protocols for signing routing messages. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS '98),* San Diego, California, March 1998. Internet Society.

[46] Y.-C. Hu, D. Johnson, and A. Perrig. SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks. *Workshop on Mobile Computing Systems and Applications (WMCSA)* 2002. IEEE Computer Society Press, 2002.

[47] Y.-C. Hu, A. Perrig, and D. Johnson. Efficient Security Mechanisms for Routing Protocols. *Annual Symposium on Network and Distributed System Security (NDSS)* 2003. Internet Society, 2003.

[48] Donggang Liu and Peng Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In *Network and Distributed System Security Symposium, NDSS '03*, pages 263– 276, February 2003.

[49] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. SPINS: Security protocols for sensor networks. *Wireless Networks,* 8(5):521–534, September 2002.

[50] M. Jakobsson. Fractal hash sequence representation and traversal. In *Proceedings of the 2002 IEEE International Symposium on Information Theory (ISIT '02),* pages 437–444, July 2002.

[51] M. Jakobsson, "Method and Apparatus for Efficient Computation of One-Way Chains in Cryptographic Applications," U. S. Patent Application 09/969,833.

[52] D. Coppersmith and M. Jakobsson. Almost optimal hash sequence traversal. In *Proceedings of the Fourth Conference on Financial Cryptography* (FC '02), Lecture Notes in Computer Science, 2002.

[53] Yaron Sella. On the computation-storage trade-offs of hash chain traversal. In *Proceedings of Financial Cryptography 2003 (*FC 2003), 2003.

[54] Mihir Bellare, Ran Canettiy and Hugo Krawczykz, "Keying Hash Functions for Message Authentication", *Advances in Cryptology - Crypto 96 Proceedings*, Lecture Notes in Computer Science Vol. 1109, N. Koblitz ed., Springer-Verlag, 1996.