An Improved Elegant Method to Re-initialize Hash Chains

Yuanchao Zhao, Daoben Li
Department of Information Engineering
Beijing University of Posts and Telecommunications
Beijing 100876, P. R. China

Abstract: Hash chains are widely used in various cryptographic systems such as electronic micropayments and one-time passwords etc. However, hash chains suffer from the limitation that they have a finite number of links which when used up requires the system to re-initialize new hash chains. So system design has to reduce the overhead when hash chains are re-initialized. Recently, Vipul Goyal proposed an elegant one-time-signature-based method to re-initialize hash chains, in this efficient method an infinite number of finite length hash chains can be tied together so that hash chains can be securely re-initialized in a non-repudiable manner. Vipul Goyal's method is improved in this paper to reach a little more efficient method, which, more importantly, is a natural extension of the concept of conventional hash chains.

Key words: Hash chain, One-time signature, Non-repudiability

1 Introduction

Hash chains are first proposed by Lamport [1]. Although originally they are used to protect one-time password against eavesdrop and replay, hash chains quickly are employed extensively in various cryptographic systems because they have public key cryptography like properties while they are efficient computationally.

For example, in electronic micropayment schemes which are dedicated to business transaction only involving tiny amount of money, the non-repudiability of payment information is required, which, however, can result in the computation overhead. Micropayments can't use conventional public key signature as macropayment because signing and verifying cost too much on computing, otherwise, the cost will be higher or equal to the value of transaction itself. So efficiency is very important for the design of micropayment system, and hash chains can satisfy these requirements. The micropayment schemes using hash chains, such as PayWord [2], NetCard [3] and Pederson [4], typically look each link on the hash chains as a piece of non-repudiable payment information for one payment unit. Also multiple hash chains can be used together to implement multi-denomination payment, as in Netpay [5].

In addition to password based authentication and micropayments, server-supported signatures [6, 7], efficient multicasting [8, 9], certificate revocation [10] and so on are also based on the concept of hash chains.

However, most of these systems suffer from a common limitation that the hash chains have a finite length. And the length of hash chain can't be selected too long because the computational and storage capacity of sender and receiver is limited in many cases. When exhausted, hash chains should be re-initialized. At this time, the need for a good method for the efficient and secure re-initialization of hash chains was clear. Vipul Goyal [11] introduces the notion of 'tying' multiple finite length hash chains. This tying is achieved by using one time signature. This paper improves the idea of Goyal's to lead to a little more efficient method. Furthermore, it is in a very natural way that our method extends the concept of hash chains.

The rest of the paper is organized as follows: Section 2 briefly introduces the idea of hash chain and one time signature. Section 3 discusses the related works, especially the work of Vipul

Goyal. Section 4 gives a detailed description of our method to re-initailze the hash chains. Section 5 compares our methods with Vipul Goyal's. Section 6 concludes this paper.

2 The concept of hash chain and one time signature

2.1 Conventional hash chain (CHC)

A conventional hash chain is constructed by applying a public one-way hash function (OWHF) *h*() iteratively to an random seed value. Function *h* maps a bit string of an arbitrary length or a predetermined maximal length to a string of a fixed length. And function *h* satisfies 3 properties: (1) Given *x*, it is easy to evaluate *h*(*x*); (2) Given *h*(*x*), it is infeasible to compute *x*; (3) It is infeasible to find two values *x* and *y* ($x \neq y$) such that *h*(*x*)=*h*(*y*).

When constructing a CHC of length *N*, one should first select a random seed value *s*, then he repeatedly applies the one-wary hash function *h* to *s* totally *N* times. The following sequence is resulted:

$$s, \quad h(s), \quad h^2(s), \quad \ldots \quad h^i(s), \quad \ldots, h^{(N-1)}(s), \quad h^N(s)$$

where *s* can be written as $h^0(s)$ and $h^N(s)$ is called the tip of the CHC. The tip resembles the public key in public key cryptography, because knowing $h^N(s)$ and not knowing *s*, one can't generate $h^{(N-1)}(s)$, but given $h^{(N-1)}(s)$, one can easily verify its correctness by $h^N(s)$. The rest may be deduced by analogy, i.e., knowing $h^i(s)$ and not knowing *s*, one can't generate $h^{(i-1)}(s)$, but one can easily verify the correctness of $h^{(i-1)}(s)$ by $h^i(s)$, where $i = N, N-1, \ldots, 1$.

The CHC is typically used as follows: first $h^N(s)$ is securely distributed and then the elements of the hash chain are released in turn starting from $h^{(N-1)}(s)$ and continuing until the seed value is used. At this time the hash chain is said to be exhausted and the whole process should be re-initialized again with a different seed value.

2.2 One time signature (OTS)

First of all, it is import to note that OTS employs nothing more than OWHFs same as in hash chain, so OTS adapt to the applications requiring high efficiency.

OTS is also first proposed by Lamport, whose idea is retailed in [12]: signing a single 1 bit message. The signer selects as the secret key components two random value *x*1and *x*2, respectively representing '0' and '1'; Then he evaluate *y*0=*h*(*x*0) and *y*1=*h*(*x*1) using OWHF *h* and publishes *y*0 and *y*1 as the public key components; To sign a one bit message, he exposes *x*0 if the message is '0', otherwise, he exposes *x*1; The verifier of the message evaluate the hash value using the exposed value as input and compare the output to the corresponding public key component. To sign a message of *n* bit length, 2*n* *x*'s and 2*n* *y*'s are needed.

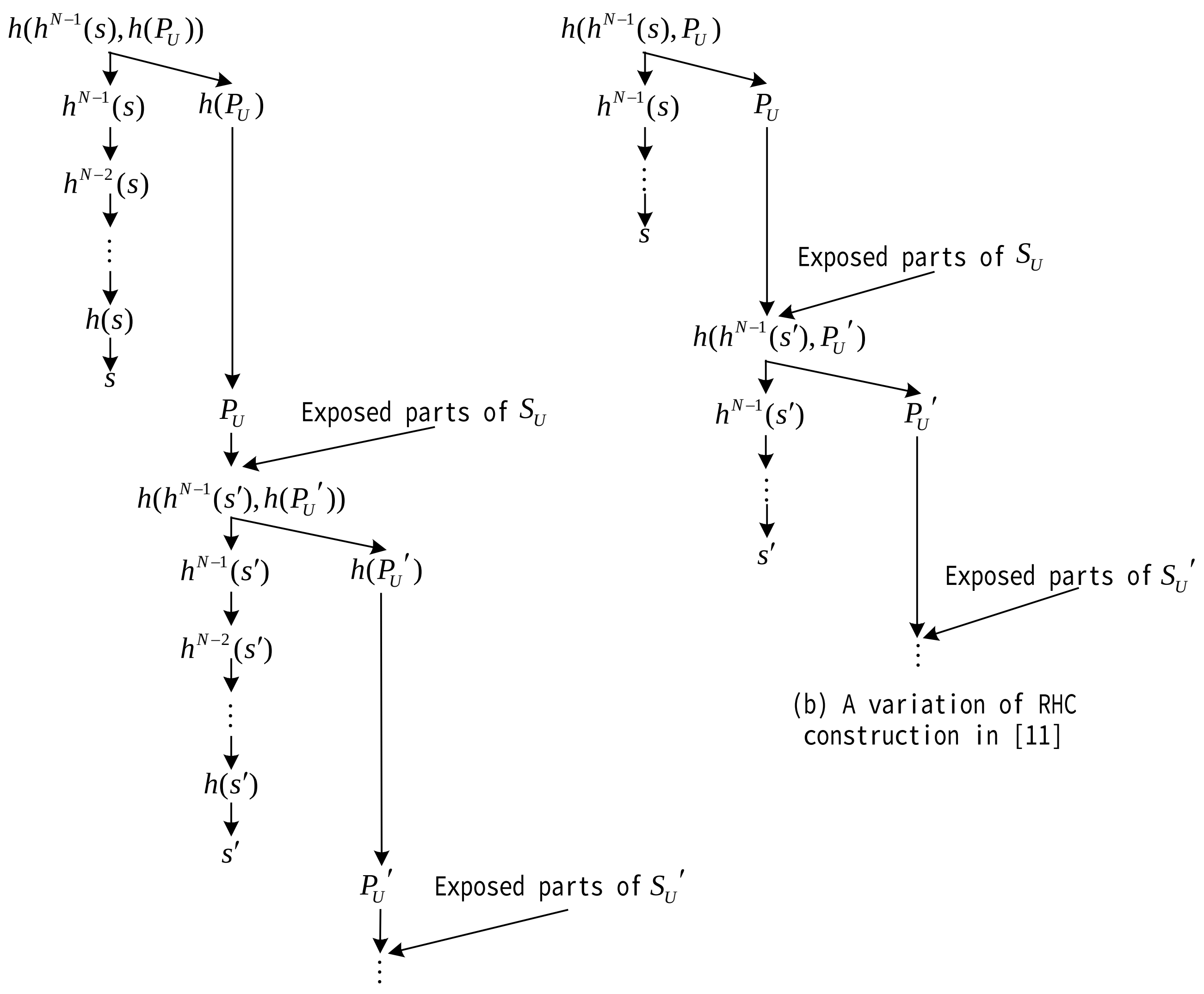
Among the various improvements to this original OTS scheme, for our purpose, we exploit the scheme proposed by Merkle [13]: the signer generates only one random value

$x_i(i=1,2,...,n)$ for each bit of the message of *n* bit length, computes $y_i = h(x_i)(i=1,2,...,n)$ and publishes all y_i's; When the *i*th bit of the message is '1', the signer releases x_i, otherwise, he releases nothing. Because this gives the chances to receiver to pretend that he did not receive some of the x_i's and therefore to pretend that some the '1's in the signed message were '0's, the signer must also sign the count of '0's in the message, and the signing method for the count field is the same as that for the message body. Now, if the receiver wants to deny that he received a '1', he must also increase the value of the count field, therefore, he must give some x_i's that he doesn't know for the count field, which, however, is infeasible for him. So, in Merkle's scheme, to sign a *n* bit message, one should prepare $\left(n+\lceil \log_2(n) \rceil\right)$ x_i's and the corresponding $\left(n+\lceil \log_2(n) \rceil\right)$ y_i's.

3 Related works countering finite length limitation of CHC

Bicaki and Baykal [14] propose signature chains of infinite length based on conventional public key cryptography. But this method can't replace CHC because of its compromising on efficiency.

Recently, Vipul Goyal [11] presents a new idea of Re-initializable Hash Chain (RHC). When a RHC is exhausted, it can be efficiently and securely re-initialized in a non-repudiable manner to result in another RHC. His construction can be briefly illustrated in Figure 1(a) where arrows show the non-repudiation relations, i.e., the side from which an arrow comes provides non-repudiable evidence for the side to which the arrow goes. This process can be continued indefinitely to give rise to an infinite number of finite length hash chains tied together using OTS. In Figure 1(a), P_U and S_U are a pair of OTS key instances, also are $P_U{}'$ and $S_U{}'$ $\cdots$. For detailed description of Goyal's construction, see [11].



(a) RHC construction in [11]

(b) A variation of RHC construction in [11]

Figure 1 RHC construction in [11] and a vatiation of it

4 The proposed construction

Similar to [11], our construction also can be called RHC. Assume that the output length of OWHF *h*() is *L*bits. For instance, the output of MD5 [15] algorithm is *L*=128bits long.

The sender first generates an pair of instances of OTS for a message of *L*bits length, including generating $\left(L + \lceil \log_2(L) \rceil\right)$ random values, the concatenation of which is written as S_U, and evaluating the corresponding image of *h*() with these random values as inputs to result in $\left(L + \lceil \log_2(L) \rceil\right)$ hash values, the concatenation of which is written as P_U. S_U and P_U are respectively the secret key components and the public key components of OTS. In fact the OTS key instances will be used to sign for the next RHC. Then sender uses the P_U as seed value to compute a CHC of length *N*, which tip is $h^N(P_U) = h(h(h(...h(P_U)...)))$ (N times). Note that this seed P_U is somewhat different with usual seed of CHC, i.e., the length of former is $\left(L+\lceil\log_2(L)\rceil\right)\bullet L$ bits and the length of latter usually can be Lbits. $h^N(P_U)$ is securely distributed to the appropriate receivers depending on the application settings. Now the links in the chain can be spent as in the usual manner till $h^0(P_U)$ or P_U is spent. At this point, the RHC is exhausted and a new RHC should be generated and tied to the foregoing RHC. In order to do this, we use P_U and S_U.

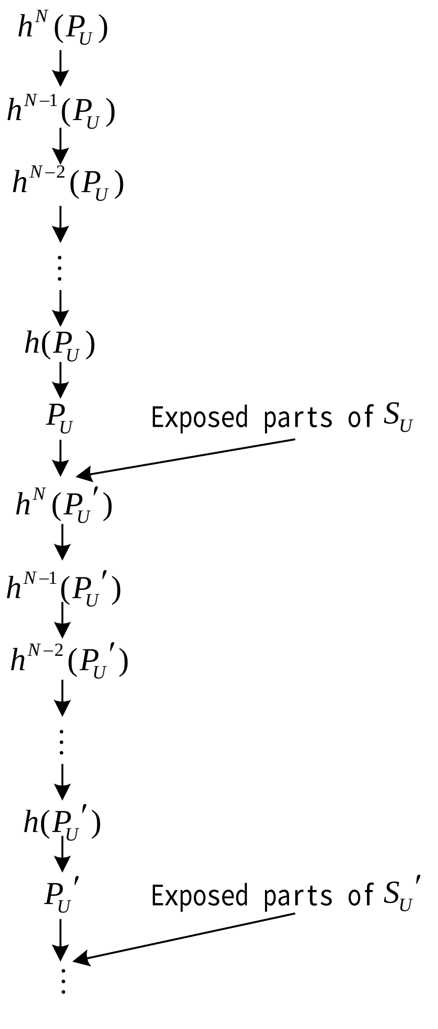


Figure 2 Our RHC construction

The sender generates another pair of instances of OTS, P_U' and S_U', in the same way as before and computing $h^N(P_U')$ as the tip of the new RHC. Now, the sender sends $h^N(P_U')$ and the appropriate parts of S_U required to sign $h^N(P_U')$ to the receiver. The receiver checks the signature on $h^N(P_U')$ using P_U and the exposed parts of S_U. Once the checking is passed, the receiver accepts the tip $h^N(P_U')$ of the new RHC, which is ready to be spent as CHC. This process can continue indefinitely in the similar way so that an infinite number of finite length hash chains tied together. Obviously, the non-repudiability is maintained during the process. Figure 2 illustrates our construction.

5 Comparison between our construction and Goyal's

Our construction also has the advantage of improving efficiency further similar to Goyal's if the to-be-exposed parts of OTS secret key are also spent in the same manner as the normal links of the RHC. The operation is as following:

a) The *N* links of RHC are released in the usual manner till P_U is reached.

b) Then, the sender computes $h^N(P_U')$ and sends it to the receiver who stores it without verification. Now the sender sends the to-be-exposed parts of S_U one by one (in different messages) as links.

c) After the receiver gets all the required parts of S_U, he can verify $h^N(P_U')$. Now the RHC can be used as in a) and b).

On average, the number of the to-be-exposed parts of S_U is $\left(L+\lceil \log_2(L)\rceil\right)/2$, so a RHC of length *N* can be used $N+\left(L+\lceil \log_2(L)\rceil\right)/2$ times.

In addition, our construction has the following advantages over Goyal's:

a) We note that in [11] the usage of $h(P_U)$ is somewhat special. In our construction, $h(P_U)$ is a link on chain naturally so that the generation of tip need not any special operation and the receiver need not to pre-store $h(P_U)$. Of course, [11] can also directly use P_U instead of $h(P_U)$, correspondingly the tip should be computed as $h(h^{N-1}(s), P_U)$ to result in a variation of its construction, referring to Figure 1(b), which can be thought out of Figure 1(a) intuitively. However, when current RHC is used, no one can determine whether a new RHC should be re-initialized. Under this condition, transferring P_U is of no significance. Especially, the size of P_U is not small, if unnecessary, P_U should not exist in traffic. Although not illuminated in [11], it is reasonable to use $h(P_U)$ to avoid unnecessary traffic. In our construction, it is natural to transfer P_U only when the current RHC is exhausted, so that there can't be any unnecessary traffic.

b) In our construction, there is no need to generate extra random seed values for hash chain. Because S_U are random, the images under *h*(), P_U, are also random. It is natural and reasonable to make use of P_U as seed value.

From above analysis, our construction is a little more efficient than Goyal's. More importantly, our construction is a very natural extension to the concept of CHC.

6 Conclusions

We present an improved method to re-initialize hash chains. It can efficiently and securely re-initialize hash chains in a non-repudiable manner and extends the concept of conventional hash chains in a very natural way. Our method is pragmatic for most systems using hash chains.

References

[1] L. Lamport. Password authentication with insecure communication. Communications of the ACM, 1981, 24(11): 770-772.
[2] R. Rivest and A. Shamir. Payword and MicroMint: two simple micropayment schemes. Proceedings of the 4th Security Protocols International Workshop (Security Protocols), Lecture Notes in Computer Science, Cambridge, UK, 1996, 1189:69-87.
[3] R. Anderson, H. Manifavas and C. Sutherland. NetCard – a practical electronic cash system. Proceedings of the 4th Security Protocols International Workshop (Security Protocols), Lecture Notes in Computer Science, Cambridge, UK, 1996, 1189: 49-57.
[4] T. Pedersen. Electronic payments of small amounts. Proceedings of the 4th Security Protocols International Workshop (Security Protocols), Lecture Notes in Computer Science, Cambridge, UK, 1996, 1189: 59-68.
[5] X. Dai and B. Lo. Netpay-An efficient protocol for micropayments on the WWW. URL: http://ausweb.scu.edu.au/aw99/papers/dai/paper.html
[6] N. Asokan, G. Tsudik and M. Waidners. Server-supported signatures. Journal of Computer Security, Nov. 1997.
[7] X. Ding, D. Mazzocchi and G. Tsudik. Experimenting with server-aided signatures. Network and Distributed Systems Security Symposium (NDSS '02), Feb. 2002.
[8] A. Perrig, R. Canetti, D. Song and D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. Proceedings of IEEE Security and Privacy Symposium S&P 2000, May 2000.
[9] A. Perrig, R. Canetti, D. Song and D. Tygar. Efficient and secure source authentication for multicast. Proceedings of Network and Distributed System Security Symposium NDSS 2001, Feb. 2001.
[10] W. Aiello, S. Lodha and R. Ostrovsky. Fast digital identity revocation. Proceedings of Advances in Cryptography – Crypto '98, Lecture Notes in Computer Science, Berlin, Germany, 1998, vol. 1462: 137-152.
[11] V. Goyal. How to re-initialize a hash chain. URL: http://eprint.iacr.org/ 2004/097.pdf
[12] W. Diffie and M. Hellman. New directions in cryptography. IEEE Trans. on Information Theory, 1976, IT-22(11): 644-654.
[13] R. Merkle. A digital signature based on a conventional encryption function. Proceedings of Advances in Cryptology – CRYPTO '87, Lecture Notes in Computer Science, Santa Barbara, CA, USA, 1988, vol. 293: 369-378.
[14] K. Bicakci and N. Baykal. Infinite length hash chains and their applications. Proceedings of the 11th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative

Enterprises (WETICE’02), Pittsburgh, USA, June 2002: 57-61.
[15] R. Rivest. The MD5 message digest algorithm. RFC 1321, April 1992.