Addressing the Shortcomings of One-Way Chains

Abstract

One-way hash chains have been the preferred choice, over the symmetric and asymmetric key cryptography, in security setups where efficiency mattered; despite the ephemeral confidentiality and authentication they assure. Known constructions of one-way chains (for example, SHA-1 based), only ensure the forward secrecy and have limitations over their length i.e., a priori knowledge of chain's length is necessary before constructing it. In this paper, we will see how our approach, based on chameleon functions, leads to the generation of practically unbounded one-way chains with constant storage and computational requirements. We provide the construction and advantages of our proposal with the help of a secure group communication setup. We also provide the implementation details of our construction and argue its suitability for security setups, where one cannot a priori determine the longevity of the setup.

Keywords: Authentication, one-way chain, secure multicast, secure group management.

1 Introduction

Entity authentication is one of the core primitives that is required to build dependable and secure systems [11]. Several cryptographic protocols have been proposed to reach this goal. These protocols use cryptographic primitives based on symmetric or asymmetric key crypto-system. The appropriateness of a cryptographic protocol for an application also depends on the storage and computational costs it incurs. In general, entity authentication is achieved using asymmetric cryptography leading to data confidentiality using computationally less expensive symmetric cryptography. But, to realize such a mechanism, an underlying security framework is required, called PKI (Public Key Infrastructure). Integration of PKI in computationally constrained environment or in dynamically changing setups is not practical due to the cost and complexity of the framework [7, 12].

A widely accepted approach is the use of SHA-1 or MD5 based one-way hash chains [16, 14] (alternatively referred as Lamport chain, in this paper), that provides verifiable data authentication in an efficient way. Much recent work [20], even tries to achieve the characteristics provided by asymmetric cryptography with the help

of one-way hash chains and loose time synchronization between the communicating principals [19], where ephemeral data confidentiality (forward) is ensured but backward secrecy (i.e., confidentiality of previously communicated data) is absent. Though these approaches try to address a particular domain of applications (e.g., [19]), the philosophy can be extended to much complex scenarios (e.g., group membership, secure multicast communication and key management, etc.,), in distributed environment. Lamport chains achieve these objectives efficiently, but with few limitations, such as: bounded chain length and no support for backward secrecy.

In this paper, we propose a construction of one-way chain that provides all the properties supported under traditional SHA-1 based one-way chain, apart from unbounded length and support for backward secrecy. Our construction is based on chameleon functions [15, 4, 9, 21], that were originally proposed for undeniable signatures. In comparison with the traditional one-way chain schemes, our solution: allows to generate practically un-bounded one-way chain, whose length is only limited by the finite-ness of the field over which the values are generated; has constant storage and computation requirement; provides forward as well as backward secrecy of communications; and does not require generation of complete chain before starting its use, since generation and usage of the chain proceed in same direction, where the generation process is at least one step ahead of the later.

The paper is organized as follows: In the next section, we briefly provide a background and related work, followed by the properties of the basic building block of our construction, chameleon functions, and their use in constructing one-way chains of any length in Section 3. In Section 4, we argue the importance of the additional properties (unbounded length and backward secrecy) provided by our construction in a secure group communication setup. The implementation details of three different ways to construct the chameleon chains are organized in Section 5, followed by conclusions in Section 6.

2 Background and Related Work

The chameleon hash functions stem from a non-interactive chameleon-commitment scheme [8]. An implementation of the chameleon hash, based on the discrete logarithm, is provided in [6]. The first implementation of a chameleon hash function designed with the goal of computational efficiency is given in [15], which is a specific and efficient implementation of a general claw-free permutation introduced in [13]. These algorithms have been employed with digital signature to build *not transferable* signature. The non-transferability property is convenient in many scenarios in which the signer has a legitimate interest in controlling subsequent disclosures of the signed information. One application suggested in [3] is private auctions. However, that first effective proposal

of chameleon hash functions [15] suffered from a key exposure problem: revealing two colliding chameleon hash values would reveal the trapdoor, the chameleon hash function is built upon. To address this problem an identity-based scheme was proposed in [3], while a key-exposure free construction, based on the elliptic curves with pairings, appeared later in [9]. The much recent work in [4] provides several constructions of exposure-free chameleon hash functions based on different cryptographic assumptions, such as RSA and the discrete logarithm assumptions. These algorithms also show an improvement in their computational efficiency. The rising interest in chameleon functions will probably bring forward more efficient constructions.

As for authentication in multicast communication, the proposals [19, 20] are efficient; however they work under the assumption of loose time synchronization between the sender and the receiver. Relaxing these assumptions results in possible violation of the packet authenticity. Whereas in our proposal, violating the loose time synchronization assumption results in the violation of fresh-ness, without compromising on the key's authenticity. Related work addressing these problems of authentication over a lossy channel appeared in [18, 17]. These proposals are mainly based over the amortization of a signature over several packets. Similar analogy can be exploited in our approach. However, addressing issues related with the transmission over unreliable channels is not our main concern in this paper.

3 Construction of Chameleon Chain

In this section, we provide a brief overview of chameleon functions [15, 3, 9, 4] and their properties followed by our construction of chameleon one-way chain. We shall also explain functioning of our construction in contrast to the traditional SHA-1 based one-way chains.

Assume that a principal chooses an asymmetric key-pair, where HK_R denotes the public-key and CK_R denotes its corresponding private-key (also called trap-door). A chameleon hash function is associated with a unique public-key. Let $CH_R(.,.)$ be the chameleon hash function derived from public-key HK_R.

Given a message m_i and a random seed r_i, $CH_R(m_i, r_i)$ provides an image (hash value) satisfying the following properties:

- **Collision resistance:** There is no efficient algorithm that on input the public key HK_R can find pairs m_1, r_1 and m_2, r_2 where $m_1 \neq m_2$, such that $CH_R(m_1, r_1) = CH_R(m_2, r_2)$, except with negligible probability.
- **Trapdoor Collision:** There is an efficient algorithm that on input the secret key CK_R, any pair m_1, r_1, and any additional message m_2, finds a value r_2 such that $CH_R(m_1, r_1) = CH_R(m_2, r_2)$.

- **Uniformity:** All messages m induce the same probability distribution on $CH_R(m,r)$ for a given r chosen uniformly at random [15].

Therefore,

1. the knowledge of public key HK_R allows a user to derive the corresponding chameleon hash function $CH_R(.,.)$
2. only the owner of HK_R's corresponding private key/trap-door, i.e., CK_R, can efficiently find a collision for any given output, and
3. for others, the function $CH_R(.,.)$ offers strong collision-resistance, i.e., it is computationally infeasible to find two inputs with the same image.

Traditional constructions of one-way hash chains using SHA-1 like cryptographic primitives involves randomly choosing a seed value and successively applying the one-way hash function (SHA-1) on the seed value until a desired length of one-way chain is achieved [16]. Thus, a chain of length n is:

$$H^n(k),\ ...,\ H(H(H(k))),\ H(H(k)),\ H(k),\ k. \tag{1}$$

Therefore, given $H^{n-i}(k)$, it is easy to compute $H^{n-i+1}(k)$, where $i = 1,\ \ldots,\ n-1$; but not vice versa. To use this chain values as encryption keys for secure communication, the owner of the chain at first communicates $H^n(k)$, called *anchor*, securely (generally, using asymmetric cryptographic mechanism) to the recipient of the intended communication. Let us refer to the owner of the chain as "Sender" and as "Receiver" to the recipient of key and data. Once after providing the *anchor* to the "Receiver", "Sender" uses subsequent values in the chain as encryption keys for data communication. The authenticity of new encryption key can be checked by "Receiver" by performing one hash application on this new encryption key and checking it against the old encryption key [16, 19]. As mentioned in previous sections, this approach suffers from the bounded length of the chain and requires generation of new chains and communicating the *anchor* to the "Receiver." The drawback of this approach is more starking if one uses it in group communication where backward secrecy is necessary to restrain newly added members from knowing past communications among the group members.

Generation of chameleon chain - Generating each value that constitutes chameleon chain involves, computing a chameleon hash (cf. Appendix A for discrete-log based approach), such that the new value collides with the old value in the chain. Unlike SHA-1 based one-way chain, we do not need to compute the whole

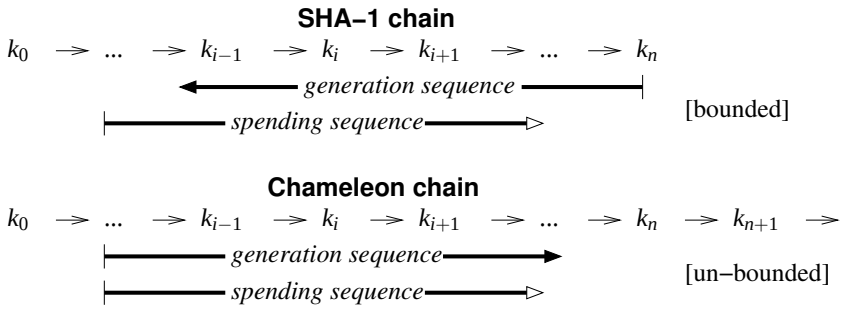

Figure 1: Life spans of SHA-1 and chameleon one-way chains

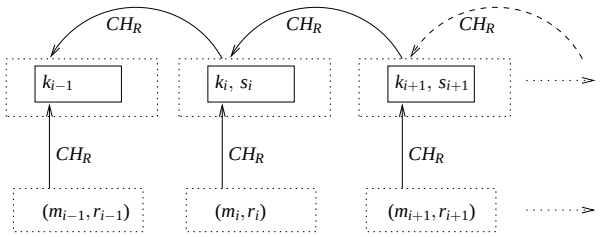

Figure 2: Generation of chameleon one-way chain by the sender

chain a priori to use it. As shown in Figure 1, in our construction, the chain generation and spending sequence proceed in same direction. The crux of our chain construction and its use is enumerated below, with the help of Figure 2, in step-wise fashion:

- **Sender:**

  1. randomly chooses m_{i+1} and r_{i+1}, and creates new key $k_{i+1} = CH_R(m_{i+1}, r_{i+1})$,
  2. chooses s_{i+1} such that; $k_i = CH_R(m_i, r_i) = CH_R(k_{i+1}, s_{i+1})$, and
  3. sends the message $p = \{k_{i+1}, s_{i+1}\}$ to the user.

- **Receiver:**

  1. receives the message packet p,
  2. authenticates k_{i+1} by checking if $k_i = CH_R(k_{i+1}, s_{i+1})$, otherwise discards k_{i+1}.

Complexity - Our construction, based on schemes proposed in [15, 21], achieves key authenticity in *(storage*computation)* complexity $O(1)$, since;

- **on sender side:**

– it only needs to *store* the values of current key k_i, next generated key k_{i+1}, the public key HK_R and corresponding secret key CK_R, and

– needs to *compute* $O(max\{|m_{i+1}|,|r_{i+1}|\})$ modular multiplications to perform $k_{i+1} = CH_R(m_{i+1}, r_{i+1})$, and $O(max\{|m_i|,|r_i|\})$ modular multiplications to find a collision $k_i = CH_R(m_i, r_i) = CH_R(k_{i+1}, s_{i+1})$. Note that, $|m_{i+1}|$ and $|r_{i+1}|$ are constants irrespective of the chain length.

• **on receiver side:**

– needs to *store* the values of public key HK_R, current key k_i, and k_{i+1} sent by the server;

– needs to *compute* $O(max\{|k_{i+1}|,|s_{i+1}|\})$ modular multiplications to verify $k_i = CH_R(k_{i+1}, s_{i+1})$. Note that, $|k_{i+1}|$ and $|s_{i+1}|$ are also constants.

Use of SHA-1 based one-way hash chains as a tool for improving the efficiency of a variety of practical and valuable security applications is well rooted. The fact that SHA-1 based chains have to be a priori computed before starting their use forces its user (the sender) to settle for a storage versus computation trade-off. For example, given the *anchor* of the chain and an intermediate value k_i, the task to compute next value k_{i+1} in the chain either takes $(i+1)$ SHA-1 computations, or has a storage requirement of $O(n)$ of a chain of length n. All straightforward combinations of these two techniques can be shown to have a *(storage*computation)* complexity of $O(n)$, which can be a substantial computational burden for many resource-constrained devices, such as wireless sensor networks or hand-held devices [11]. We have provided a detailed performance analysis of our implementation in Section 5. Let us bring forward other advantages of our construction, i.e., unbounded chain length and backward secrecy with the help of a typical application setup. In next section we provide an application of chameleon chain in key management for LKH (Logical Key Hierarchy) setups.

4 Application to Multicast Communication

One-way hash chains have been exploited in multicast communications (e.g., in TESLA [19]) to achieve efficient source authentication and data confidentiality. However, the nature of multicast communication is a group communication where the data transmitted by a member (broadcaster) should be available to the members of that group only while ensuring source authentication and data confidentiality. For example, in TESLA, these properties are achieved with the help of one-way hash chain under the assumption that group members are loosely time-synchronized with the broadcaster. Another more practical approach for secure group communication is

using “Key Graphs” [22] that provides a mechanism to dynamically add (join group) and delete (leave group) users from a group. This approach, also called LKH (Logical Key Hierarchy), works under the assumption that users are honest and there exists one or more trusted key management servers that act (update keys in the LKH) on each join or leave operation in the group. In [10], LKH is shown to be prone to session disruption, session hijack, replay attacks and had been shown robust against these attacks using values of one-way hash chain as values for the auxiliary key nodes in LKH setup. In this section, we briefly review this proposal [10], its drawbacks, and the necessity of chameleon chains in lieu of traditional SHA-1 based one-way chains. Note that, application to LKH setup is just one of the applications of our construction and it can stand a great enabler in a variety of other applications that demand one-way (unbounded) chains with backward secrecy.

In the framework proposed for secure group communication (key-oriented setup, i.e., LKH) [22], a secure group is denoted by a triple $(U,\ K,\ R)$ where U denotes a set of users, K a set of keys held by the users, and $R \subset U \times K$ a user-key relation which specifies keys held by each user in U. Each user is given a subset of keys which includes user's individual key, a group key, and sub-group keys based on the configuration of the LKH. The set K is maintained (updated upon each join/leave) by the group administrator i.e., root node of LKH. However, this framework is susceptible to attacks brought forward in [10] and can be made resistant against these attacks by making auxiliary keys self verifiable with the help of one-way hash chain values as keys.

Herein, we briefly present the re-keying mechanism in LKH setup that uses one-way hash chain values as keys for auxiliary nodes. Therefore, the auxiliary key for node i is k_{i,j_i} where j_i keeps track of index of one-way chain assigned to node i. On g^{th} re-keying, the key of the i^{th} auxiliary node will change to $k_{i,j_i'}$, where $j_i' = g$. The *anchor*s of the chains used for auxiliary nodes, are provided to appropriate users by the group administrator through a secure channel as in original LKH protocol [22]. With the help of Figure 3, and Table 1, we show a leave operation under the LKH setup described in [10]. Figure 3, shows a LKH setup for $U = \{u_1,\ ..,\ u_8\}$, controlled by group administrator c. The leaf nodes of the tree are the users belonging to U. Each user has to store the keys that are on the path from itself (leaf-node) to the root. Let k_{u_z} be the a key shared by the user u_z with the center. A communication message p flowing from c to u is shown as; $c \rightarrow u : p$. Table 1, gives sequence of messages broadcasted by c to evict user u_1 from the secure group. We employ following notations; $\{string\}_{key}$ indicates that, the message *string* is encrypted using symmetric-key encryption and key *key*.

*Message*1 is broadcasted for the users of the right sub-tree and contains the new session key k_{0,j_0+1} encrypted with k_{2,j_2}. *Message*2 and *Message*3 are used to change the old keys $k_{1,j_1},\ k_{3,j_3}$ that the user u_1 shared with the users u_2, u_3, u_4.

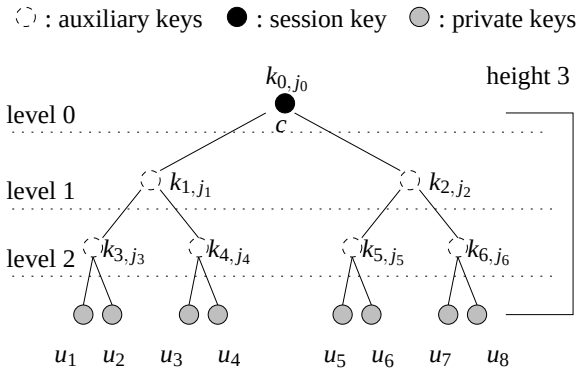

Figure 3: LKH setup and notations

Message 1 - $c \rightarrow u_5, u_6, u_7, u_8 \ : \ \{k_{0,j_0+1}\}_{k_{2,j_2}}$
Message 2 - $c \rightarrow u_3, u_4 \ : \ \{k_{1,j_1+1}\}_{k_{4,j_4}}, \ \{k_{0,j_0+1}\}_{k_{1,j_1+1}}$
Message 3 - $c \rightarrow u_2 \ : \ \{k_{3,j_3+1}\}_{k_{u_2}}, \ \{k_{1,j_1+1}\}_{k_{3,j_3+1}}, \ \{k_{0,j_0+1}\}_{k_{1,j_1+1}}$

Table 1: Eviction of u_1 under key-oriented protocol

As already mentioned, the classical re-keying protocol is subject to attacks that are pointed out in [10], and these highlighted attacks are addressed using self-verifiable keys (i.e., one-way hash chain values as auxiliary keys). However, two issues still stand: 1) overcoming the finite-ness of the traditional one-way hash chains, and 2) enforcing backward secrecy. Indeed, in [10], using the Lamport's scheme, if a new user receives key k_i, then the user can locally compute all the previously generated values of the hash chain and can decrypt the previously communicated data within the secure group, thus demanding backward confidentiality to the communication. This justifies the need for our one-way chain construction. We show the above re-keying protocol using chameleon one-way chains.

To set up the secure group using chameleon one-way chains, the center c computes the keys $k_{i,j_i} = CH_R(m_{j_i}, r_{j_i})$ for all auxiliary nodes in the setup. Let k_{0,j_0} be the session key of the group. Then, the group administrator, c, distributes these keys to appropriate users (that is, to the users that include the auxiliary node on their path to the root of the tree). This initial key distribution is done in a secure fashion, as done in [22, 23]. Let us analyze the re-keying under this setup. Assume user u_1 needs to be removed from the secure group. The center needs to renew the entire set of keys that are on the path between u_1 and the root. The internal details of chameleon chain while performing renewal of auxiliary key k_{i,j_i} to k_{i,j_i+1} are enumerated below.

- **Sender (Center):**

  1. generates a new key k_{i,j_i+1} such that $k_{i,j_i+1} = CH_R(m_{j_i+1}, r_{j_i+1})$ and $m_{j_i} \neq m_{j_i+1}, r_{j_i} \neq r_{j_i+1}$

Message 1 -	$c \rightarrow u_5, u_6, u_7, u_8 \ : \ \{k_{0,j_0+1}, s_{j_0+1}\}_{k_{2,j_2}}$
Message 2 -	$c \rightarrow u_3, u_4 \ : \ \{k_{1,j_1+1}, s_{j_1+1}\}_{k_{4,j_4}}, \ \{k_{0,j_0+1}, s_{j_0+1}\}_{k_{1,j_1+1}}$
Message 3 -	$c \rightarrow u_2 \ : \ \{k_{3,j_3+1}, s_{j_3+1}\}_{k_{u_2}}, \ \{k_{1,j_1+1}, s_{j_1+1}\}_{k_{3,j_3+1}}, \ \{k_{0,j_0+1}, s_{j_0+1}\}_{k_{1,j_1+1}}$

Table 2: Re-keying steps for eliminating u_1 in LKH with chameleon one-way chains

2. chooses s_{j_i+1}, using the trapdoor collision property, such that $k_{i,j_i} = CH_R(k_{i,j_i+1}, s_{j_i+1})$,
3. sends the re-keying messages (e.g., as enumerated in Table 2.)

- **Receiver:**

1. receives the re-keying message,
2. decrypts the message containing the new keys using the appropriate keys, according to the LKH setup.
3. checks for each newly received key k_{i,j_i+1}, if the new key verifies $k_{i,j_i} = CH_R(k_{i,j_i+1}, s_{j_i+1})$. If the match succeeds, the new key k_{i,j_i+1} is authentic.

Our solution for secure group communications overcomes the limitations arising in [10]. In general, our construction provides the following additional properties, to exploit, for the applications that use traditional one-way chains: backward secrecy, key authenticity, unbounded chain with constant storage and computation requirement, and no need of even loose synchronization between the sender and the receivers. Thus, relieving the group manager from tracking one-way chains for each auxiliary key in the setup. One should also note that, re-keying is carried out even when there are no join/leave operations in the setup, to desist cryptanalytic attacks [1].

5 Experimental Analysis

In this section, we argue that the proposed scheme is viable and offers good trade-off of computation between sender and receiver of the chain. Before that, we shall discuss the environment under which we carried our experiments, provide the results and discuss the suitability of our approach to a class of applications.

Experimental Setup: The implementation is carried out on GNU/Linux (i486) platform with gcc-3.3.5, OpenSSL 0.9.7e library for cryptographic primitives (without any external cryptographic acceleration) and numerical analysis. To get a fair computational estimation, we did not use any code optimization of gcc while building our executables.

5.1 Approach to Compute Execution Time

Various approaches are possible to audit the process execution time. We employed the method of tracking CPU cycles consumed during execution of a function of our interest. The experiments are carried out on an AMD 750MHz machine, that complies IA32 architecture (which provides cycle counter; a 64-bit, unsigned number). The IA32 counter is accessed with the `rdtsc` (read time stamp counter) instruction. This instruction takes no arguments. It sets register `%edx` to the high-order 32 bits of the counter and register `%eax` to the low-order 32 bits. Based on this methodology, a pair of functions are integrated with our code that allows us to measure the total number of cycles that elapse between any two time points:

```
#include "clock.h"
void start_counter(); /* Starts the counter */
double get_counter(); /* Returns: Number of cycles since last call to start_counter */
```

To verify the precision of this approach we marked the counter before and after `sleep(sleeptime);` function call (where sleeptime equals to one). We obtained 756,154,624.0 as return value (i.e., 756.2 MHz). We run each function of our interest for 101 times and discarded the first value of execution time in favor of cache warming process. Furthermore, results are gathered in run-level 1; to minimize interference from other processes. To plot all the results into graphs with common scale, we introduced dummy 101^{st} and 102^{nd} entries in our results with values equal to -1 and 200,000,000 respectively.

5.2 Comparative Analysis

We implemented Chameleon scheme with three different methods, namely: Simple Factorization (SF), Discrete Logarithm (DL) (both from [15]), and Advanced Factorization (AF) [21]. Implementation of these schemes can be categorized into two phases: Hash Generation (HG) and Finding Collision (FC). They produce hash of length 160-bits. Lamport scheme (SHA-1) is realized using OpenSSL EVP library, and gives 160-bit digest as value of intermediate unit of the one-way chain. Our results are summarized in Figure 5, and Table 3. The values in Table 3, for chameleon implementations, are the average taken over 100 runs.

It would be inappropriate to compare the computational cost for generating one unit value of one-way chain in our construction and SHA-1 based construction, because our construction has constant storage and computational costs due to the unidirectional propagation of its generation and spending (cf. figure 1). While, SHA-1 approach wins over our approach using storage versus computation trade-offs. To normalize the comparison,

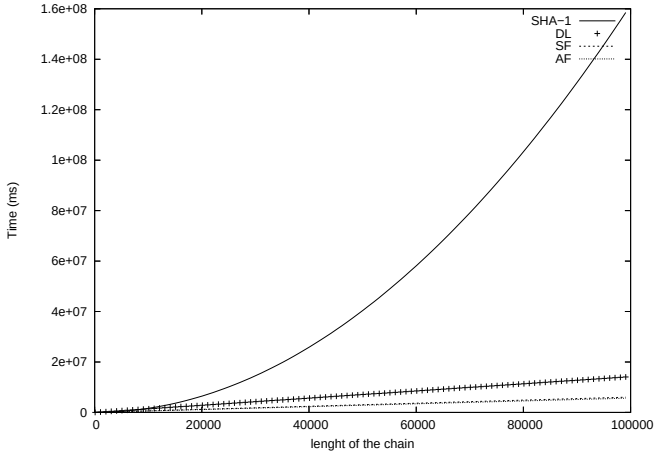

Figure 4: SHA-1 versus chameleon SF, DL, and AF

we assess both schemes in Table 3 with constant storage requirement and comparing them based on their pure computational costs. That means for a SHA-1 chain of length 1000, the sender of the chain can store only the seed value of the chain and its associated counter and compute 999^{th} value by performing 999 hash operations and so on. Since initial key distribution (i.e., securely communicating the *anchor*s of one-way chains to intended users) costs are the same for both, SHA-1 based chains and chameleon chains, we do not provide the actual cost for this phase. Here, our one-way chain has advantage over SHA-1 based one-way chain as SHA-1 based approach may need as many such operations as new chains of fixed length are generated. Note that, in the LKH application shown in this paper, it is impossible to a priori determine, either the longevity of the setup or the number of possible join/leave operations for any auxiliary node.

In the following, we show how one-way hash chain can place a heavy computational burden on the center, where it stores only the seed and the corresponding counters of chains. In a one way chain of length n, to compute the j^{th} value on-the-fly, the center needs to perform $n-j$ hash operations. Hence, to exhaust the one-way chain (i.e., after n authentications), the number of hash operations (NH) performed by the center is given by the following equation;

$$NH = \sum_{i=0}^{n} i = \frac{n(n+1)}{2} \tag{2}$$

Therefore, if we compare the computational costs incurred to the center while using all the four (SHA-1, SF, DL, AF) hash operations, we obtain the graph shown in Figure 4. In this graph, the x axis represents the length of one-way chain, and y axis represents the time (in milliseconds), needed by the center to exhaust that chain. It is evident from Figure 4, that the three implementations of chameleon hash functions provide better

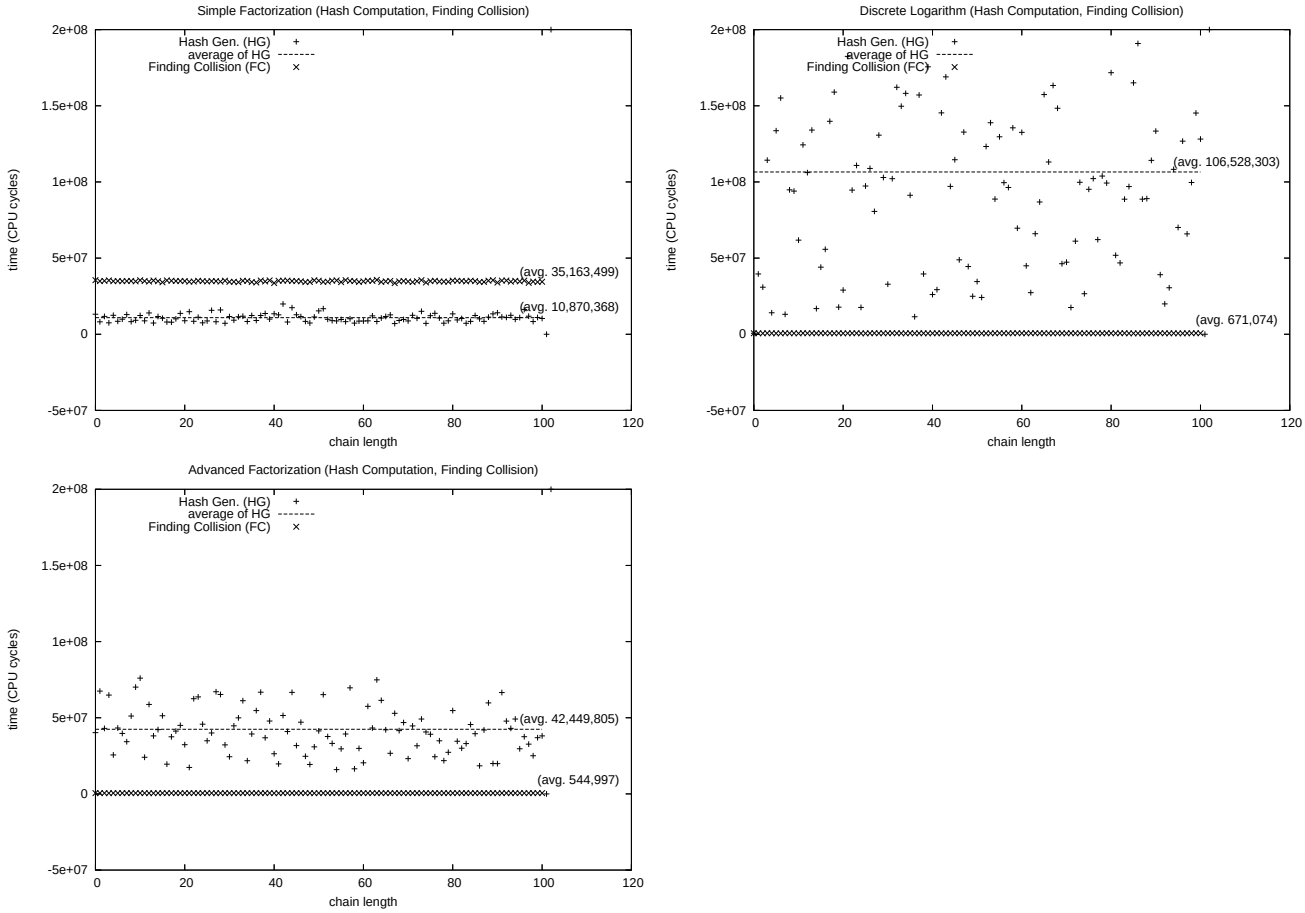

Figure 5: Run time analysis of chameleon one-way chain implementations

performances over the SHA-1 scheme, in long run. The computational advantage of SHA-1 disappears when the chain length is around 9,000. Thereafter, SHA-1 based one-way chains incur more cost. The computational cost for chameleon based schemes are given by $n \times [Cost(Find\ Collision) + Cost(Hash\ Gen.)]$, and for the $SHA-1$ based scheme, it is given by $O(n^2) \times Cost(hash)$ (cf. Eq. 2). To compare amongst chameleon based implementations, DL based implementation is the least efficient, while the SF and AF show overall similar performances (the two curves almost overlap in Figure 4). Simple factorization based implementation of our construction is suitable for setups where end users have limited computational capabilities. Also note that; SHA-1, MD5 etc., are the most optimized implementations in any standard cryptographic suit, unlike chameleon functions. We hope that the importance, and capability of chameleon schemes will bring forward more efficient implementations to existence. Furthermore, as we can see from Table 3, and Figure 5, hash generation phase requires a random number (that is provided by OpenSSL PRNG in our implementation); this phase can be considerably improved if the underlying application has a source of random numbers [5].

Nevertheless, even for a chain of small length (e.g., n = 5000), the average cost of an authentication under

SHA-1 scheme is higher than the cost of a hash under chameleon scheme. Note that, the computational cost advantage chameleon chain over SHA-1 chain increases as the value of n increases (cf. Figure 4).

Chameleon scheme	**SF**		**DL**		**AF**	
	Hash Gen.	Find Collision	Hash Gen.	Find Collision	Hash Gen.	Find Collision
	14.375 *ms*	46.503 *ms*	140.881 *ms*	0.887 *ms*	56.139 *ms*	0.720 *ms*
SHA-1 based scheme	the average cost (for the sender) to perform authentication using a chain with 5000 elements = $(\frac{5000(5000+1)}{2} \times 0.0322)/5000 = 80.516$ *ms*, where, 0.0322 *ms* is the cost of one SHA-1 hash					

Table 3: Comparative analysis of chameleon and SHA-1 chains based on their computational cost

5.3 Trade-offs between the SHA-1 based and the Chameleon based Schemes

These two schemes can be compared in terms of chain length, memory versus computational cost incurred to use the chains, durability of the message authenticity, and extra settings like time synchronizations, between communicating peers to realize the setup. Let us enumerate few undeniable advantages of our chain construction over the SHA-1 based one-way chains.

- **Practically unbounded one-way chain -** Unlike Lamport's one-way chain, this scheme provides a one-way hash chain whose length is restricted by the finiteness of the field the chain is built upon. As we have seen earlier, one-way chains using chameleon functions are always generated and used in same direction (i.e., the generator of the chain is just a few steps ahead than the last revealed value), whereas in Lamport's one-way chain the generation and usage proceed in opposite directions, therefore limiting the length of the chain.

- **Unit storage and computation requirement -** In one-way chains derived using SHA-1 hash function the applications require a trade-off between storage of the actual chain values and/or re-computation of the values (at the chain-owner's side). Such a constraint does not exist for chains based on chameleon functions because there is no need to compute the chain in advance in order to get the *anchor* of the chain.

- **Backward secrecy -** Knowing a value of any unit from the SHA-1 one-way chain, lets one derive all the values between that point up to the *anchor* of the chain. This eventually leads to the exposure of all previous communications secured using the hash values of such a chain. Whereas, in case of chameleon based one-way chains, one can derive (for authentication of newly received unit of the chain) only the previous value in the chain; not beyond that. This is an important requirement while using one-way chains

for managing group memberships. A newly joined/evicted member should not get undue privileges, other than the assigned ones, when the membership action is performed. We have shown this mechanism in section 4.

- **Time synchronization requirement -** The applications that use Lamport chains need to time-synchronize the communicating ends to establish authenticity of the distributed keys. In short, such a setup temporarily provides properties of asymmetric cryptography; where the asymmetry is introduced by the time difference between the sender and the receivers. In our construction, keys are self-verifiable and it is not possible to trace backward into the chain, the time synchronization between communicating peers is not necessary.

All the above mentioned properties can be realized with alternative mechanisms (possibly by the combination of SHA-1 chains and asymmetric cryptography) in *ad hoc* fashion, with increased costs and complexity.

Among the three Chameleon implementations we carried out, SF performs better in "Hash Computation" whereas AF performs fair in "Finding Collisions". With this observation, one can accommodate SF and AF into applications involving computationally weak verifiers (for instance, sensor networks) and computationally weak communication originator (a satellite engaging multiple base-stations), respectively.

6 Conclusions

In this paper, we have derived an analogy between Lamport's one-way chain and one-way chains based on chameleon functions. Our construction has the following advantages over the former: i) practically unbounded length, ii) backward secrecy, iii) constant storage and computational requirements, and iv) no time synchronization requirement for multicasts. Our construction is more efficient than SHA-1 based one-way chain, in storage constrained setups. Our scheme, in comparison with SHA-1 based one-way chain, has a linear overhead in computation, while the latter has quadratic scaling. We hope that the importance, and capability of chameleon schemes will bring forward more efficient implementations to existence.

Note that, we have explored chaining as one of the possible constructs, and shown its application for a typical scenario. There are several other security setups that require the properties provided by our construction. Chaining is only important if causality of signatures is needed. Furthermore, chameleon hash functions can be used just as easily to construct trees or even simpler star-like constructs that would eliminate the need for the verifier to store intermediate values.

References

[1] M. Abdalla and M. Bellare. Increasing the lifetime of a key: A comparative analysis of the security of re-keying techniques. In *ASIACRYPT*, pages 546–559, 2000.

[2] G. Ateniese, D. H. Chou, B. de Medeiros, and G. Tsudik. Sanitizable signatures. In *ESORICS*, pages 159–177, 2005.

[3] G. Ateniese and B. de Medeiros. Identity-based chameleon hash and applications. In *Financial Cryptography: 8th International Conference, FC 2004*, pages 164–180. Springer LNCS 3110, 2004.

[4] G. Ateniese and B. de Medeiros. On the key exposure problem in chameleon hashes. Cryptology ePrint Archive, Report 2004/243, 2004. http://eprint.iacr.org/.

[5] B. Barak, R. Impagliazzo, and A. Wigderson. Extracting randomness using few independent sources. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 384–393, Washington, DC, USA, 2004. IEEE Computer Society.

[6] J. Boyar, S. A. Kurtz, and M. W. Krentel. A discrete logarithm implementation of perfect zero-knowledge blobs. *J. Cryptology*, 2(2):63–76, 1990.

[7] S. Brands. *Rethinking Public Key Infrastructures and Digital Certificates; Building in Privacy*. MIT Press, 2000. ISBN 0-262-02491-8.

[8] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and Systems Sciences*, 37(2):156–189, 1988.

[9] X. Chen, F. Zhang, and K. Kim. Chameleon hashing without key exposure. In *Information Security Conference*, pages 87–98. Springer LNCS 3225, 2004.

[10] R. Di Pietro, A. Durante, and L. V. Mancini. A reliable key authentication scheme for secure multicast communications. In *Proceedings of the* 22^{nd} *IEEE Symposium on Reliable and Distributed Systems*, pages 231–240. IEEE press, 2003.

[11] R. Di Pietro and L. V. Mancini. Security and privacy issues of handheld and wearable wireless devices. *Communications of the ACM*, 46(9):75–79, 2003.

[12] W. Ford and M. S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption, Second Ed.* Prentice Hall, 2002.

[13] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[14] N. M. Haller. The S/KEY one-time password system. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 151–157, 1994.

[15] H. Krawczyk and T. Rabin. Chameleon signatures. In *ISOC Network and Distributed System Security Symposium (NDSS)*, pages 143–154, 2000.

[16] L. Lamport. Password authentification with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.

[17] A. Pannetrat and R. Molva. Efficient multicast packet authentication. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2003.

[18] J. M. Park, E. K. P. Chong, and H. J. Siegel. Efficient multicast packet authentication using signature amortization. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 227. IEEE Computer Society, 2002.

[19] A. Perrig, R. Canetti, D. Song, and J. D. Tygar. Efficient and secure source authentication for multicast. In *Network and Distributed System Security Symposium, NDSS'01*, pages 35–46. Internet Society, 2001.

[20] A. Perrig, R. Canetti, J. D. Tygar, and D. X. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, pages 56–73, 2000.

[21] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 355–367. Springer LNCS 2139, 2001.

[22] C. K. Wong, M. G. Gouda, and S. S. Lam. Secure group communications using key graphs. *IEEE/ACM Trans. Netw.*, 8(1):16–30, 2000.

[23] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam. Reliable group rekeying: a performance analysis. In *SIGCOMM*, pages 27–38, 2001.

A Chameleon Hash Algorithm (based on Discrete Logarithm)

Setting: Choose prime $q \equiv 3 \ mod \ 8$ and $q' \equiv 7 \ mod \ 8$.
Let $HK_R = c = qq'$ and $CK_R =< q, q' >$

Input: Message $m_0 = m_0[1] \ldots m_0[l]$;
Output: The value *hash*: A chameleon hash of m_0;

$HK_R = n$ as defined above
Choose random value $r_0 \in Z_n^*$;
$hash = r_0^2 \ mod \ c$
for $i = 1$ to l
  $hash = (4^{m_0[i]} hash^2) \ mod \ c$
next i
return(*hash*)

Table 4: An example of chameleon hash generation based on claw-free permutations [15].