Efficient Constructions for One-way Hash Chains[1]

Yih-Chun Hu Markus Jakobsson Adrian Perrig

5 November 2003

CMU-CS-03-220

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[1] Yih-Chun Hu is with UC Berkeley (`yihchun@eecs.berkeley.edu`), Markus Jakobsson is with RSA Laboratories (`mjakobsson@rsasecurity.com`), and Adrian Perrig is with Carnegie Mellon University (`perrig@cmu.edu`). Adrian Perrig's research was supported in part by the Center for Computer and Communications Security at Carnegie Mellon under grant DAAD19-02-1-0389 from the Army Research Office, and by gifts from Bosch and Intel Corporation. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of ARO, Bosch, Carnegie Mellon University, Intel, or the U.S. Government or any of its agencies.

Keywords: One-way hash chains, efficient constructions, broadcast authentication.

Abstract

One-way chains are an important cryptographic primitive in many security applications. Lamport first proposed to use one-way chains for one-time password authentication [19]. Subsequently, researchers proposed one-way chains as a basic building block for digital cash, for extending the lifetime of digital certificates, for constructing one-time signatures, for packet authentication, etc.

As one-way chains are very efficient to verify, they recently became increasingly popular for designing security protocols for resource-constrained mobile devices and sensor networks, as their low-powered processors can compute a one-way function within milliseconds, but would require tens of seconds or up to minutes to generate or verify a traditional digital signature [6]. Recent sensor network security protocols thus extensively use one-way chains to design protocols that scale down to resource-constrained sensors [20, 28]. Recently, researchers also proposed a variety of improvements to one-way hash chains to make storage and access more efficient [9, 17, 32], or to make setup and verification more efficient [16, 20].

In this paper we present two new constructions for one-way hash chains, which significantly improve the efficiency of one-way chains. Our first construction, the Sandwich-chain, provides a smaller bandwidth overhead for one-way chain values, and enables efficient verification of one-way chain values if the trusted one-way chain value is far away. Our second construction, Comb Skipchain, features a new lower bound for one-way chains in terms of storage and traversal overhead. In fact previously, researchers [9] cite a lower bound of $\log^2(n)$ for the product of per-value traversal overhead and memory requirements for one-dimensional chains. We show that one can achieve a lower bound by considering multi-dimensional chains. In particular, our two-dimensional construction requires $O(\log(n))$ memory and $O(1)$ traversal overhead, thereby improving on the one-dimensional bound. In addition, the setup cost for the one-way chain is in contrast only $O(n/\log(n))$. Other benefits for both constructions include a faster verification step than the traditional hash chains provide; a verifier can "catch up" efficiently, after having missed some number of previously released hash values (for the Sandwich-chain); and resistance against DoS attacks on authentication values. Moreover, we describe fractal traversal schemes for our proposed structures, bringing down the traversal costs for our structure to the same as those of the simpler "traditional" hash chain.

Our new construction is orthogonal to most previously proposed techniques, and can be used in conjunction with techniques for efficient setup or verification of one-way chains.

1 Introduction

One-way chains are a widely deployed cryptographic primitive. Lamport first proposed to use one-way chains for efficient authentication of one-time passwords [19], which Haller later refined to the S/KEY standard [12]. Since Lamport's work, many researchers proposed to use one-way chains as a basic building block for a variety of applications, for example for digital cash [2, 14, 26, 30], for extending the lifetime of digital certificates [1, 24], for constructing one-time signatures [10, 22, 23, 31], for authenticating link-state routing updates [8, 13, 33], or for efficient packet authentication [27].

Despite the computational efficiency of one-way functions, one-way chains are still challenging to use in resource-constrained environments, such as on small mobile devices or sensor networks. Especially some of the proposed sensor networks have significant resource limitations, as they use minimal hardware to lower the energy consumption [18]. In these resource-challenged environments the setup, traversal, verification, and storage of long one-way chains is a major challenge.

Recently, researchers proposed a variety of improvements to one-way hash chains to make setup, traversal, and storage more efficient. A good metric for one-way chain efficiency is the product of the per-value traversal overhead and the memory requirements.[1] For example, simply storing each value of a one-way chain with length n would result in a cost of $O(n)$, as storage requires $O(n)$ memory and traversal is $O(1)$ (no computation necessary, the one-way chain values are simply stored in an array). Another straightforward approach is to only store the seed of the chain, and derive each value on the fly, with an $O(n)$ efficiency again, as storage costs are $O(n)$ and traversal costs $O(1)$. Jakobsson [17], Coppersmith and Jakobsson [9], and Sella [32] propose new techniques that make make traversal and storage more efficient, and apply these techniques to traditional one-way chains. All of these techniques allow the computation of consecutive values in the hash chain at a cost of only $O(\log(n))$ one-way function computations (traversal cost), while also requiring $O(\log(n))$ storage, resulting in an efficiency of $O(\log^2(n))$. Given that the traversal techniques are applied to standard hash chains, the verification cost is not affected by the manner in which the values are represented and computed, making the verification cost $O(n)$. This is also the

[1] The traversal cost can be zero when the entire chain is stored, which would result that the product would be zero as well. We could deal with this by also accounting for memory accesses, or by adding 1 to the number of one-way function computations for the traversal cost. Both techniques result in a non-zero positive traversal cost.

computational cost of the setup phase, in which the value at the endpoint is computed given a randomly selected seed; this computation may be performed by a powerful and trusted device, as opposed to the resource constrained device that performs the traversal.

Hu et al. propose a new structure for one-way chains, in which more than one level of chains are used [16]. The main benefit of their approach structure is that it allows for more efficient verification: a verifier would only have to compute the sequence of hash function evaluations corresponding to a small portion of the total number of traversed values. We review their approach in Section 4 and refine their construction to design a new one-way chain that only requires $O(1)$ traversal overhead and $O(\log(n))$ storage, resulting in an efficiency of $O(\log(n))$.

Liu and Ning propose a two-level one-way chain, where the chains of the second level are derived from values of the first level [20]. Their scheme provides a linear speedup for setup and verification, and thus still requires $O(n)$ setup, storage, and verification overhead.

This paper makes the following contributions:

- **Framework.** We introduce a framework for comparing one-way chain techniques, considering setup, traversal, verification, storage, and communication overheads.

- **New two-dimensional chains.** We propose a new technique for authenticating chains below the first level of the hierarchy, producing three clear advantages in comparison to the related approach of [20]: First, it avoids jamming-based DoS attacks that focus on disrupting the transmission of the sensitive authentication values for secondary chains. Second, it allows users to store and forward the new authentication values for the benefit of other users, who did not receive them when first transmitted. This approach does not require users to trust one another. Note that we obtain this benefit without the use of digital signatures or other heavy-weight constructions; in fact, our approach has the same low computational demands as the approach taken in [20]. Third and finally, it allows a user who has missed some (potentially large) number of authentication values to "catch up" with a computational effort that is a fraction of number of transmitted authentication values she missed. This feature, which is not present in other proposals (whether those using traditional or hierarchical chains) may prove particularly beneficial in settings where nodes are mobile.

- **Light chains.** We propose the notion of *light one-way chains* to lower the communication overhead. These are relatively short hash chains whose values are shorter than what is normally needed to avoid attacks based on inverting or finding collisions; we show how to use these in a way that avoid attacks. Using known hash indexing techniques, we further make our proposal resistant against birthday attacks, that would otherwise reduce the security of the structure as its number of elements grows.

More specifically, we present two new one-way chain constructions: Sandwich-chain and Comb Skipchain. Sandwich-chain is a new hash chain structure with a fractal traversal algorithm requiring only $O(\log(n))$ computation and storage; a reduction of the bandwidth requirements; and a substantially reduced verification effort. The exact verification effort depends on parameter choices, and will be described in detail. Comb Skipchain is a new hash chain featuring only $O(n/\log(n))$ cost for setup, $O(\log(n))$ storage, and $O(1)$ cost for traversal. This construction is substantially more efficient than the lower bound for one-dimensional one-way chains. Like the new structures of [16, 20], our construction is not compatible with all previous uses of hash chains. However, and as will become evident, there are many common uses of traditional hash chains that can easily be adapted to use our structure.

The outline of the paper is as follows. Section 2 presents one-way chain basics and our evaluation framework. Section 3 introduces our Sandwich-chain, and Section 4 presents our Comb Skipchain construction. We describe the use of light chains in 5. Section 6 discusses different deployment scenarios and analyzes which one-way chain technique is most appropriate.

2 Background and Evaluation Framework

We present a framework for evaluating and comparing one-way chain proposals, comparing setup, traversal, verification, storage, and communication costs. Our constructions, like those of [20], are based on a hierarchy of chains, where values of some chains (which we refer to as *primary* chains) are used as roots for other chains (so-called *secondary* chains.)

One-Way Chain Setup. A one-way chain $(V_0 \ldots V_N)$ is a collection of values such that each value V_i (except the last value V_N) is a one-way function of the next value V_{i+1}. In particular, we have that $V_i = H(V_{i+1})$, for $0 \leq i < N$. Here, H is a one-way function, and is often selected as a cryptographic hash function,

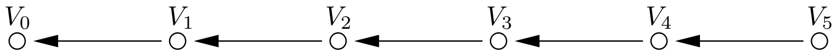

Figure 1: Standard one-level one-way chain.

which is why the structure is often also called a *hash chain*. For *setup* of the one-way chain, the *generator* chooses at random the *root* or *seed* of the chain, i.e., the value V_N, and derives all previous values V_i by iteratively applying the hash function H as described above. The value V_0, which we refer to as the *end-value*, is normally made public, and potentially linked to the identity of the user possessing the corresponding root value. An example of a standard hash chain is shown in Figure 1.

Verification of One-Way Chain Values. We assume that the *verifier* knows an authentic value of the generator's one-way chain, usually the end-value V_0. To verify an input value V_i of a chain, the verifier iteratively applies the one-way function H i times and compares the result to the trusted value V_0, i.e., verify that $H^i(V_i)$ equals V_0. If the computed and known values are equal, then the input value is said to be authentic. Note that if another value V_k, for $k < i$, is already known, then it suffices to iteratively apply the one-way function some $i - k$ times to the input value, and compare the result to this intermediate value.

One-way Chain Traversal. When the generator discloses successive values of the one-way chain, we call this one-way chain *traversal*. In the introduction, we mentioned two simple traversal techniques: in one the generator stores all values of the one-way chain in memory, and in the other the generator recomputes each value from the seed value. In so-called *fractal* traversal techniques the set of values that is stored is modified over time in a manner that reduces the (storage-times-computation) complexity.

One-way Chain Advantages and Disadvantages. Traditional one-way chains have many advantages. First of all, given only a trusted value V_i of the chain, it is intractable to find a value V_j, where $j > i$, such that $H^{j-i}(V_j) = V_i$ (assuming that H is a secure one-way function and that the output of H is sufficiently large, we further discuss the security of one-way chains below). However, it is easy to assess the validity of a value V_j, where $j > i$, by verifying that $H^{j-i}(V_j) = V_i$ (assuming that H provides weak collision resistance, also called second pre-image collision resistance).

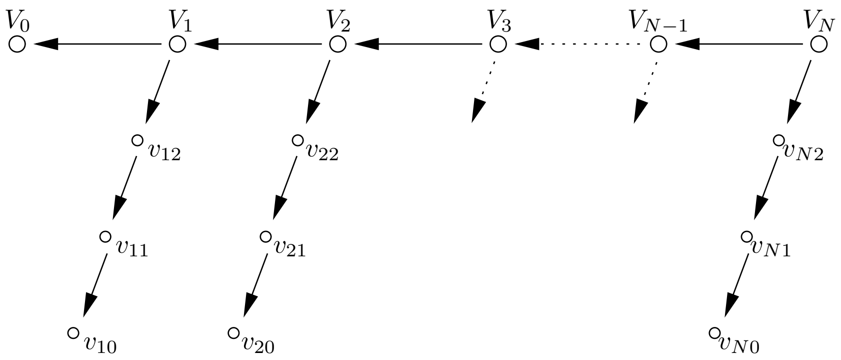

Figure 2: Hierarchical one-way chain, where $V_N \dots V_0$ are values on the one-way primary chain, and the values $v_{i0} \dots v_{i2}$ are values of a secondary light chain. There is no subchain under V_0 since it serves as the verification value.

A drawback of traditional one-way chains is that the verifier has to perform $j - i$ operations to validate V_j given V_i, which can be expensive if $j - i$ is large. Finally, the repeated disclosure of one-way chain values carries a cost related to the transmission of these values. The required bandwidth may be a burden to senders, especially in highly resource-constrained environments, such as in sensor networks.

Hierarchical One-Way Chains. A *hierarchical* one-way chain consists of two or more levels of chains, where values of a first-level ("primary") chain act as roots of a set of second-level ("secondary") chains. We refer to the secondary chain rooted in the ith value of the primary chain as the ith secondary chain. Here, all the values of the ith secondary chain are released before any of the values of the $i + 1$st chain is released; the primary chain value V_i is released in between. Figure 2 shows an example of such a structure.

As will be described later, different one-way functions may be used for primary and secondary chains, with the aim of lowering the communication costs.

To set up the hierarchical chain, the generator picks V_N at random and computes the primary chain $V_{N-1}, \dots, V_0$. The generator computes the secondary chain on the fly. A clear advantage is the very efficient setup, as only N/K operations are needed to compute V_0, where K is the length of the secondary chain.

To use this one-way chain, the generator traverses all the secondary chains in sequence (e.g., v_{00}, v_{01}, v_{02}, $v_{20}, \dots, v_{N0}, v_{N1}, v_{N2}$) and discloses the values of the primary one-way chain when possible.

A disadvantage of the hierarchical chain is the authentication of end-values of secondary chain. This hierarchical chain was proposed by Liu and Ning [20]. Liu and Ning propose to use the TESLA authentication protocol [27] using the primary chain to authenticate the end-values of the secondary chain. This approach has the shortcoming that the hierarchical chain can only be used in conjunction with the TESLA authentication protocol, as they propose to authenticate the end-values of the secondary chain with the TESLA authentication protocol using the primary chain. The disadvantage of that approach is that the loss of the authentication message prevents the verifier to authenticate secondary chain values until the next value of the primary chain is disclosed.[2] Another shortcoming of their approach is that the authentication is staged, as the generator can only send authentication values at transitions of the primary chain. The tradeoff is clear, on one hand we would like to have infrequent transitions in the primary chain, but on the other hand we prefer a short authentication delay.

Note that the all end-values need to be authenticated — both that of the primary chain and those of all secondary chains. As we discuss above, the authentication mechanism by Liu and Ning has several shortcomings. To overcome these shortcomings, we propose the Sandwich-chain, enabling efficient authentication of the end-values of the secondary chain at any moment, without assuming any additional authentication protocols.

Light Chains. To lower the communication overhead, we introduce the notion of the *light chain*. This is a sequence of values derived from their respective predecessors by means of a one-way function that results in a relatively short output. In contrast, we may refer to standard one-way chains as *heavy chains*. This permits us to refer to a chain consisting of both light and heavy components plainly as a *one-way chain*, or more specifically, a *hash chain*. The way we combine light and heavy chains is to use a heavy chain as a primary chain, and let each value of the heavy chain be the root of a light secondary chain.

As described below, we select the hash function used for a particular light chain from a large family of potential functions, disclosing what function was used not long before the disclosure of values of the light chain begins — two different light chains would use two different functions. This way, we avoid the threat of pre-computation attacks, since an attacker would not be able to begin building an input-output dictionary until the selection of the function is disclosed.

[2] They propose to send redundant messages to achieve higher robustness against packet loss, however, this approach increases the communication overhead.

Still, it is clear that the shorter the values of the light chains are, the smaller the effort will be for an attacker to find collisions, or to invert the function. This presents us with a tradeoff between the savings in communication cost (related to the size of individual values of the light chain) and the time a particular light chain can be used. The latter, in turn, puts constraints on the maximum length of the light chain, given a known rate at which this is traversed and its values disclosed.

Choice of One-Way Functions. As mentioned before, different one-way functions may be used for the primary respectively secondary chains. (Moreover, different one-way functions may be used for different segments of the secondary chain, but for simplicity, we do not consider that option here.)

The basic security requirement for the generating function of a one-way chain is *one-wayness*, preventing an attacker from deriving the following value V_{i+1} of the chain when knowing V_i. In addition, the generating function also needs to provide second preimage collision resistance, which is also known as weak collision resistance. Weak collision resistance prevents an attacker from finding another V'_{i+1} after the generator disclosed V_{i+1}, which also satisfies $V_i = H(V'_{i+1})$ and $V_{i+1} \neq V'_{i+1}$. This would enable the attacker to forge one value of the one-way chain, and prevent the verifier from verifying subsequent values of the one-way chain, as almost certainly $V'_{i+1} \neq H(V_{i+2})$.

Candidate one-way functions are based on cryptographic hash functions, such as SHA-1 [25] or MD5 [29]. Here, the one-way function used to compute the values of the primary chain is denoted H, and the one-way function used to derive the values of the ith secondary chain is referred to as h_i. We derive the salts using a third one-way function, which we refer to as H_s. We let $H(\cdot) = H_s(\cdot) = hash(\cdot)$ and $h_i(\cdot) = trunc(hash(salt_i, \cdot))$, where $trunc$ denotes a function that truncates the input to the desired length.

Another construction that also provides the required security properties are pseudo-random functions (PRF) [11]. A PRF F has a key K and is often used in the following construction to provide one-wayness and weak collision resistance: $V_i = F_{V_{i+1}}(0)$, where 0 denotes a constant string of zero bits. Note that the value V_{i+1} is used as the key to the PRF. A commonly used instantiation of a PRF is to use a message authentication code (MAC), for example HMAC is a popular choice [4]. A pseudo-random permutation (PRP) (e.g., a block cipher) can also be used as a PRF, if the PRP is only used as many times as the birthday bound allows [3]. This construction though often has the disadvantage that the generator has to run key setup function for the block cipher for each one-way chain value, which is often inefficient.

Resistance against Birthday Attacks. All previous proposals involving hash chains appear vulnerable to birthday attacks. More specifically, if an attacker selects a random seed and computes a hash chain from this, the chances for a collision between any one of the computed values and a value in an existing chain C increase with the length of C. This can be avoided by using a well-known technique[3] in which all hash functions used are indexed by their position in the chain. In other words, in order to compute the image V_i of a value V_{i+1}, one would compute the hash of $V_{i+1} \mid\mid i+1$ (where $\mid\mid$ denotes concatenation), as opposed to only the one-way chain value, as is commonly done. For simplicity of notation, we do not make this indexing explicit in the descriptions onwards. For a one-way chain value of m bits, the expected cost for an attacker to find a pre-image or even a second pre-image is 2^{m-1} one-way function computations.

Taxonomy of Hash Chain Usage. Though many proposed protocols use hash chains, we categorize these protocols into three categories: those which use every element (e.g. [19, 12]), those in which skipping elements is rare (e.g. [8, 13, 33]), and those in which skipping elements is common (e.g. [27]). In all protocols, fast traversal at the generator is desirable. In addition, when skipping elements is common, faster verification is desirable, especially when hash chain elements are used quickly. In some cases, such as wireless network protocols, network partitioning can force a node to perform a large number of hash functions to "catch up;" in these cases, faster verification can significantly improve protocol responsiveness.

Evaluation Metrics. We use the following five metrics to measure the efficiency of one-way chain techniques: setup, traversal, verification, storage, and communication cost. We describe each cost in more detail. The setup cost is measured by the number of hash function computations to derive the end-value of the one-way chain. The traversal cost is the average number of hash function computations that the generator performs to derive each one-way chain value, assuming that each chain value is consecutively disclosed. For real-time constrained environments, an upper bound on the traversal cost is desirable. The recently proposed efficient one-way chain schemes provide an upper bound of $O(\log(n))$ for the

[3] This technique is attributed to Micali and Leighton, who in an unpublished manuscript propose the use of this technique to avoid birthday attacks in Merkle trees.

traversal cost [9, 17]. The verification cost is measured in the number of one-way function computations that the verifier performs to verify a one-way chain value. The verification cost is usually $O(1)$ if the verifier receives all one-way chain values, however, a more interesting metric is the verification overhead in case the verifier needs to “catch up” because it missed many one-way chain values or started receiving values late, i.e., the verifier may need to traverse a large part of the one-way chain to verify. Advanced one-way chains feature efficient mechanisms to “catch up”. The storage cost is measured as the amount of memory that the generator needs to traverse the one-way chain. The communication cost measures the bandwidth overhead for distributing one-way chain values. We introduce the notion of light chains to lower the communication cost.

Requirements for Offline Verification. The chains we describe can be verified offline; that is, without communicating with the chain generator. However, this offline verification sometimes requires either reliable delivery or loose time synchronization. Of the chains we describe, the Sandwich-chain and light chains (Sections 3 and 5) require either reliable delivery or loose time synchronization to allow offline verification. On the other hand, Comb Skipchains (Section 4) do not have any special requirements for offline verification.

3 The Sandwich-chain Construction

We previously mentioned why the primary and secondary chains (as Figure 2 shows) have a significant disadvantage over the simple one-way chain: if the commitment to the end-value of the secondary chain is lost, the verifier has to wait until the generating value of the secondary chain (i.e., the value of the primary chain) is disclosed.

We now describe the Sandwich-chain, a new construction which removes this drawback, and which has several nice properties. Figure 3 shows an example of a Sandwich-chain.

The Sandwich-chain is a combination of a hierarchical one-way chain and another one-way chain $W_N \dots W_0$ used for verification of the end-values of the light chain. We now describe how the Sandwich-chain is generated, used, and how verifiers can authenticate values.

Sandwich-chain Setup. The generator picks V_N and W_N at random and computes $V_{N-1}, \dots, V_0$, where $V_i = H(V_{i+1})$. For each V_i, the generator derives

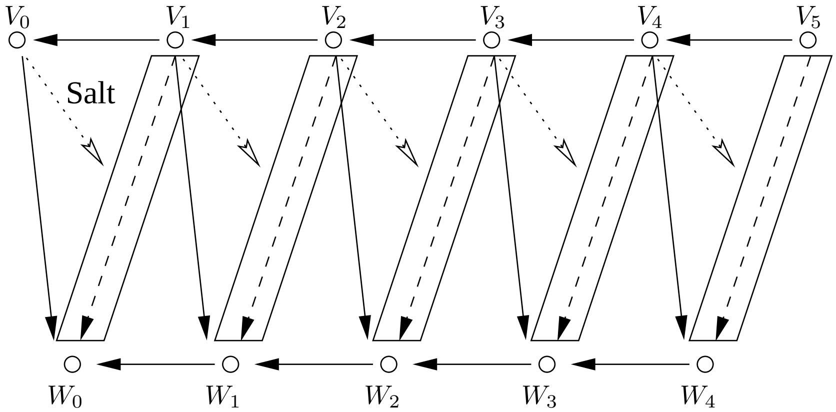

Figure 3: A sandwich chain. The upper horizontal chain is the primary chain; the diagonal chains are secondary chains; and the lower horizontal chain is used for verification of end-points of secondary chains. The dotted arrows correspond to salt derivation, which is used in the computation of the diagonal chains only.

the associated secondary chain $v_{i,K}, \ldots, v_{i,0}$ as follows: $v_{i,K} = h_s(V(i))$ (where V_{i-1} is used as the salt s), and $v_{i,j} = h_s(v_{i,j+1})$. Finally, the generator derives the W-value as follows: $W_{i-1} = H(W_i||v_{i,0}||V_{i-1})$. Figure 3 graphically represents these derivations.

Sandwich-chain Traversal. Similarly to the hierarchical chains described above, the generator uses the values of the light chains one after another, e.g.,

$$v_{1,0}, v_{1,1}, \ldots, v_{1,K}, v_{2,0}, \ldots, v_{N,0}, v_{N,1}, \ldots, v_{N,K}$$

. The V-values are disclosed after all the secondary chain values are published, for example after disclosing $v_{i,K}$, the generator can send V_i, which is needed for the salt of V_{i+1}'s secondary chain and also to authenticate W_i. W-values are disclosed as necessary, they are not secret, their use is only to authenticate the end-values of the secondary chain. The Sandwich-chain is traversed using fractal traversal [17, 9, 32].

Sandwich-chain Value Authentication. Initially, the verifier receives V_0 and W_0 over an authentic channel, and uses these values to authenticate all subsequent values. Using the trusted values V_0 and W_0, the verifier can easily authenticate the first value of the one-way chain $v_{1,0}$ by checking that $H(W_1||v_{1,0}||V_0$ equals W_0 (this assumes that the verifier also received W_1).

We now describe how the verifier authenticates the end-value $v_{i,0}$ of a general secondary chain, assuming that the verifier always kept up with receiving and verifying the chain, thus trusting values V_{i-1} and W_{i-1}. (In Section 3.2, we describe an efficient verification technique if the receiver did not keep up with the verification.) In the light chain spanned by V_i, the verifier needs to know W_i to verify the end-value $v_{i,0}$. The verifier first computes $H(W_i||v_{i,0}||V_{i-1})$ and checks that it matches the stored W_{i-1}, in which case both W_i and $v_{i,0}$ are authentic. Since value W_i is not secret, the generator can retransmit it periodically. The value V_{i-1} is used as the salt in the secondary chain $v_{i,j}$, thus V_{i-1} also needs to be transmitted periodically, and can easily be authenticated. As additional measure to authenticate the secondary chain values, the verifier can also check that $v_{i,k} = h_s(V_i)$ after V_i is disclosed.

3.1 Efficient Authentication Using One-Way Functions

We describe a novel mechanism to authenticate arbitrary values, without using a MAC function. We assume a secure weak collision resistant one-way function F (to derive the one-way chain), and a secure one-way function G (to produce commitments). The generator then generates a one-way chain $V_N, \ldots, V_0$, where $V_i = F(V_{i+1})$. We assume that the generator and verifiers are loosely time synchronized, with a maximum synchronization error of T_Δ. The generator specifies a regular disclosure schedule for values of the the one-way chain, disclosing V_i at time $T_i = T_0 + i * T_d$, where T_d is the time delay between the disclosure of two values, and T_0 is the time of disclosure of value V_0. To authenticate a value r, the generator publishes $r' = H(V_j||r)$, where V_j is a value that will be disclosed in the future. When a verifier gets r, r', j at time t, it verifies that the generator did not yet disclose V_j by checking that $t + T_\Delta < T_j$. If this condition holds, it accepts r' and waits for the disclosure of V_j to authenticate r. The verifier first verifies the authenticity of V_j, by following the one-way chain to the last authentic value. If V_j is authentic then r is authentic if $r' = H(V_j||r)$. This authentication is similar in nature to the TESLA authentication protocol, but it does not require a MAC computation.

3.2 Sandwich-chain Using our Efficient Authentication Technique

Our Sandwich-chain is especially constructed to also enable efficient authentication of the end-values of the light chain using the authentication technique described in Section 3.1. This has the advantage that a client who receives authentic values of the Sandwich-chain (i.e., V_0 and W_0), can efficiently authenticate secondary chain values in chain generated by V_i, without recomputing previous secondary chains. This approach thus substantially reduces the verification overhead for a new verifier that needs to "catch up" to current values of the chain.

We assume that the primary one-way chain $V_0, \ldots, V_N$ satisfies the requirements discussed in Section 3.1, i.e., the values V_i are disclosed after specific times T_i. A verifier can use the structure of the Sandwich-chain to authenticate the values of the W-chain without following it all the way back to the last authentic W value it trusts.

Consider a verifier that is time synchronized, who trusts value V_0, and who joins the transmission at time t_i, where $T_i < t_i < T_{i+1}$. The generator is thus currently traversing the values spanned by the secondary chain spanned by V_{i+1}, as V_i is already disclosed. At this point, the verifier cannot yet authenticate values $v_{i+1,j}$ it receives, as it may be computationally too expensive to recompute the Sandwich-chain all the way back to W_0. The generator periodically distributes W_{i+1}; as long as the verifier gets W_{i+1} before time $T_{i+1} - T_\Delta$, the verifier can later authenticate W_{i+1} after it receives V_{i+1}, W_{i+2}, and the value $v_{i+2,0}$. After successful authentication of value W_{i+1}, the verifier also knows that value W_{i+2} is authentic, and can use the values of the W-chain to immediately authenticate end-values of the following secondary chains.

Given loosely synchronized clocks, the Sandwich-chain thus enables computation-bound verifiers to very efficiently authenticate current one-way chain values (after waiting for a short time delay), without recomputing the majority of the Sandwich-chain.

4 The Comb Skipchain Construction

In [16], Hu et al. describe the skipchain mechanism. A skipchain is composed of a *signature chain* and a collection of secondary chains. A signature chain ($s_0 \ldots s_x$) is a chain of one-time signatures. Each value s_i of the signature chain spans a one-time signature scheme, where the one-time signature values are again reduced to a

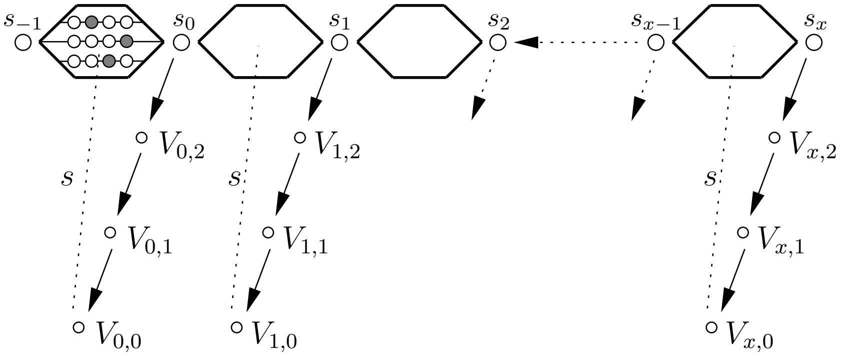

Figure 4: Comb Skipchain, where $s_y \dots s_0$ are values on the signature chain, s_{-1} represents the initial verification value, and $V_{i,0} \dots V_{i,2}$ represent light chains. Each hexagon in the figure represents one instance of a Merkle-Winternitz signature (a simplified version of such a signature is shown inside the first hexagon). The s denotes that each s_i is used to sign each $V_{i,0}$.

single value s_{i-1}, thus resembling a one-way chain. The value s_i can be thought of the private key of the one-time signature, and the value s_{i-1} is the corresponding public (verifying) key. (Similarly, s_{i-1} and s_{i-2} form the next private/public key pair.) Hu et al. describe such a signature chain [16] and propose to use the Merkle-Winternitz signature [10, 22, 31] as the one-time signature scheme.

Each value s_i in the signature chain is used to derive a secondary chain $V_{i,0} \dots V_{i,y}$, where $V_{i,y} = H(s_i)$ and the other values are generated as we describe in Section 3. In the same vein as the hierarchical chain construction in Section 3, these light chains represent a single one-way chain of length $(x+1)(y+1)$, where the jth value is $V_{\lfloor \frac{j}{y+1} \rfloor, j \mod y+1}$, and where x is the length of the secondary chain.

In this paper, we contribute a novel parameterization and traversal scheme that provides $O(1)$ computation, while retaining the $O(\log(n))$ storage requirement; a significant improvement over the previous $O(\log^2(n))$ bound [9].

Verification of Values of the Secondary Chain. Because values from the secondary chain cannot be verified through the repeated application of a one-way function, we must have a separate way of verifying them. Comb Skipchains use a one-time signature from each signature chain value to authenticate the end-value of each light chain. In particular, the ith one-time signature is used to sign $V_{i,0}$.

To verify this one-time signature, the verifier follows the signature chain all the way to a trusted value (in the same way as it follows a standard one-way chain). Note that the traversal of the primary chain makes previous one-time signatures redundant. In other words, while traversal in principle allows forgery of old one-time signatures, synchronization issues makes this a non-issue, just like knowledge of previously released chain values in general does not pose a security threat for the applications we consider.

Parameterization and Traversal For a chain of length n, we choose $x = \frac{n}{\log(n)}$ and $y = \log(n)$. The important intuition that allows us to achieve $O(1)$ computation is that the fractal traversal of the signature chain already requires $O(\log(n))$ storage, so we can store an entire secondary chain of length $O(\log(n))$ with no penalty in the asymptotic storage cost. As a result, the traversal of the secondary chains can be achieved with constant computation; and the traversal cost of the signature chain $O(\log(n))$ can be amortized over an entire light chain (length $O(\log(n))$), thus resulting in a constant computation cost per emitted value. In particular, we perform efficient traversal on the primary (signature) chain (at cost $\log(x)$ memory, and $\log(x)$ computation per step), but to amortize the traversal over the y steps in the secondary chain, for $O(1)$ computation. In addition, we can traverse the secondary chain for $O(y)$ memory and $O(1)$ computation by storing the entire chain. Finally, we amortize the computation of the next light chain, which costs $O(y)$ memory and $O(1)$ computation. The total cost is then $O(\log(n))$ memory and $O(1)$ computation.

Initialization. We perform the initialization necessary for fractal traversal of the signature chain. We also compute and store the entire first secondary chain. Finally, we prepare for the initialization of the second secondary chain.

Main routine. First, we perform a partial step in the signature chain. In particular, we follow the fractal traversal algorithm [9] until one hash operation is performed, for a cost of 1. After $\frac{1}{2}\log(n)$ steps are performed, the next signature chain value is ready. In particular, after $V_{i,\frac{y}{2}}$ is emitted, s_{i+1} becomes available.

Next, generate one more step of the next chain. In particular, when emitting $V_{i,j}$, we compute $V_{i-1,y-j}$.

Next, set the return value to the appropriate value previously computed. For example, when emitting $V_{i,j}$, we return the value computed when emitting $V_{i+1,y-j}$.

Analysis In our first step, we perform only one hash operation, and a constant amount of work, so the first step requires $O(1)$ computation and $O(\log(n))$ memory. Our second step performs one hash operation, so it requires $O(1)$ computation and $O(\log(n))$ memory. Our third step consists of a single load, so it requires $O(1)$ computation and $O(\log(n))$ memory. In fact, the memory used by the second and third step can be combined so that together they require just $y + 1$ hash values to be stored. The hash and signature are also constant time operations, and hence it also requires $O(1)$ computation. This represents a total cost of $O(1)$ computation and $O(\log(n))$ memory, for a memory-times-computation complexity of $O(\log(n))$. Our result substantially improves the previous result which provides memory-times-computation complexity of $O(\log^2(n))$, and establishes that as a lower bound.

Future Work Comb Skipchains provide other efficiency improvements; for example, in our scheme, the setup time is $O(\frac{n}{\log(n)})$. In future work, we intend to examine the use of hierarchical Comb Skipchains, with multiple levels of signature chains, as a mechanism for improving setup times.

5 Light Chains

In resource-constrained environments (e.g., sensor networks), communication bandwidth is at a premium, as data sending and receiving is expensive in terms of battery energy. To reduce the communication overhead, we propose one-way chains with reduced value size, but we need to be careful not to introduce new security vulnerabilities.

A *light* one-way chain $(v_0 \ldots v_n)$ is a collection of values such that each value $v_i = h(v_{i+1})$, for $0 \leq i < n$, where h is a one-way function with short output. That is, while the output of standard hash functions is typically 128 or 160 bits long, the output of h is much shorter, for example 64 bits long. Here, h is preferably a *salted* hash function (also referred to as a *keyed* hash function), whose output is truncated to some fixed length. Thus, each chain (or segment of a chain) has a salt s associated to it, which is appended to the input before the hash function evaluation. The effect of this is that the function h is selected from a family of functions, indexed by the salt.

Similar as for standard chains, v_0 is called the end-value, and v_n the root. The salt s of a chain (or chain segment) has to be selected before the chain is computed from the root value, but does not have to be released until a verifier needs it to verify the authenticity of a chain value.

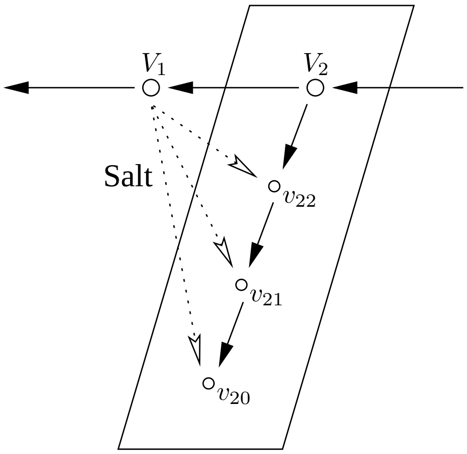

Figure 5: Salt-derived light one-way chain.

Shorter sized values may introduce pre-computation attacks, such as the attack proposed by Hellman [15]. For a chain with m-bit values, the attacker performs 2^m operation to setup a graph of size $2^{2 \cdot m/3}$. Given the graph, the attacker performs $2^{2 \cdot m/3}$ operations to find a pre-image of a value. To thwart this attack, we use two countermeasures: salted hash functions, as described below; and indexed hash functions as described in Section 2 in the paragraph "Resistance against Birthday Attacks".

Use of Salts. We have that outputs of h_i may be substantially shorter than outputs of H. Thus, while inverting H will be assumed to be computationally intractable, it may be possible to invert, or find collisions for, h_i given sufficient time. Since we want it to be infeasible to invert h_i during the time period it is associated with, it is important that this time interval is shorter than a lower estimate of the time it takes to invert the h_i. Evidently, it is also crucial to prevent pre-computation attacks, which is where the salt comes in.

The effect of salting the hash function h_i is that of randomly selecting this function from a large family of such functions. Given sufficiently long salts, this effectively makes it infeasible to compute a table of all input-output pairs of h_i without knowledge of the salt. Since the salts are not made public until just before the time period their use is associated with, this prevents pre-computation attacks.

6 Discussion

We give two examples of instances in which the our constructions are useful. In our examples, we use light chains with 64 bits of security, and heavy chains with 80 bits of security. For the one-time signatures in Comb Skipchain, we use the Merkle-Winternitz construction [10, 22, 31] to sign the 64 bits of the end-value of the light chain. To sign 64 bits, we use 32 signature chains of length two, and three checksum chains of length two, and one checksum chain of length one. This construction requires 71 hash function computations to generate one signature, and on average 35 hash function computations to verify.

Our first application is a sensor network with resource-limited sensors. In the sensor network, the nodes use the TESLA protocol to broadcast authentic data over their lifetime, which we assume to be 10 years. To achieve a relatively low authentication delay, yet have short one-way chains, the sensors use one value of the one-way chain during one second, thus requiring a total of 315 million values over the lifetime of the sensor network. We first study the overhead if we use the Sandwich-chain. One of the main features of the Sandwich-chain is that it has a low communication overhead and enables fast verification of chain values in case a receiver was absent for some time or is newly deployed. To select the length of the secondary chain (the light chain), we say that a sensor should wait for at most 30 minutes to efficiently authenticate the chain values after deployment, thus the light chain has a length of $30 \cdot 60 = 1800$ values. The primary chain thus has a length of $315,360,000/1800 = 175,200$ values. The setup cost is $315,360,000+175,200+175,200 = 315,710,400$ hash function computations to compute V_0 and W_0. Using fractal traversal [17, 9, 32], the traversal requires $O(\log(n))$ per-value computation and storage. The verification overhead is low, even if a sensor joins the network close to the end of its lifetime, it can verify the V-value in under three minutes, assuming that a hash function computation requires 1ms. In contrast, traditional one-way chains would take 1800 times longer. The communication cost is very low, as the light chain values are only 64 bits long, and only two heavy chain values of length 80 bits need to be disclosed after every 1800 values.[4]

To lower the traversal and setup cost at the expense of a higher communication overhead, we can use the Comb Skipchain mechanism. In choosing the length of the light chains, we want a length that allows a the signature chain to

[4] For robustness against packet loss, the generator may periodically publish the latest V and W values required for authentication.

be computed using a single hash operation per emitted element. Since each application of G in the signature chain requires 71 hash operations, and since each step in the signature chain requires a logarithmic number of applications of G, we need at least $71 \cdot \log_2(315, 360, 000) = 2004.5$ elements in the light chain. Since we use Sella's traversal [32], we can store ℓ elements from a chain of length $\frac{\ell(\ell+1)}{2}$ and provide reverse traversal with a single computation. As a result, we pick a minimum ℓ that allows us to traverse a chain of length at least 2005. In this case, $\ell = 63$, and each light chain is 2016 elements long. Our signature chains are of length $315, 360, 000/2016 = 156, 429$, and initialization requires traversing one light chain (2016 hash functions) and the entire signature chain ($156, 429 \cdot 71 = 11, 106, 459$ hash functions), for a total setup time of $11, 108, 475$. Traversal costs are 1 for each of the current light chain, the next light chain, and the signature chain, for a total of 3 applications. To perform these traversals, we require two sets of light chains to be stored simultaneously, at a cost of 63 pebbles each. In addition, $\log_2(315, 360, 000/2016) < 18$ pebbles are needed for the top chain, for a total of 144 pebbles. By contrast, Sella's scheme requires 533 pebbles to traverse the chain at a cost of three operations per element. The verification overhead is low; even if a sensor joins the network close to the end of its lifetime, it can verify the signature chain in around 3 minutes, assuming that a hash function computation requires 1ms. The communication cost is fairly low; again, the light chain values are only 64 bits long, and a 360 byte signature needs to be disclosed every 2016 values (possibly more often for robustness). This represents roughly 1.43 additional bits per chain value.

Another example is for micropayments for Internet or peer-to-peer traffic. Such payments could be used, for example, to pay an ISP for dialup access, in open-access wireless hotspots, or to pay for content transfer from a peer-to-peer file sharing network. To estimate pricing for such services, we consider that cable modem pricing is around \$45 for a cap of 15 GB [7], and assume that the ISP requires revenue of \$45 for 5 GB of transfer. If a micropayment provider sold in increments of \$10, then each chain will be 1.1 billion elements long. When using Comb Skipchains, we compute the length of the light chains as above: $71 \cdot \log_2(1, 111, 111, 111) = 2133.5$, so we choose $\ell = 65$ and a light chains length of 2145. As a result, our signature chains are of length $1, 111, 111, 111/2145 = 518, 001$. Initialization then requires $2145 + 518, 001 \cdot 71 = 36, 780, 216$ hash functions. Again, traversal costs are 1 for each of the current light chain, the next light chain, and the signature chain, for a total of 3 applications. To perform these traversals, we require two sets of light chains to be stored simultaneously, at a cost of 65 hash values each (these can be stored together with just 66 values). In addition, $\log_2(1,111,111,111/2145) < 19$ hash values are stored for top chain, for a total of 149 hash values (85 if better optimized). By contrast, Sella's scheme requires storage of 731 values to traverse the chain at a cost of three operations per element. The verification overhead is higher than before; if a network node receives a micropayment close to the end of the chain, it may take 36 seconds to verify, assuming that a hash function computation requires 1μs. The communication cost is fairly low; again, the light chain values are only 64 bits long, and a 360 byte signature needs to be disclosed every 2145 values (possibly more often for robustness). This represents roughly 1.34 additional bits per chain value.

7 Conclusion and Future Work

Our proposed constructions for one-way chains are useful in many settings, to speed up the current setup, traversal, and verification of one-way chains. Both constructions can be used as "drop-in" replacements for many current uses of one-way chains.

The Sandwich-chain is particularly useful in environments with low-bandwidth communication channels, and where the verifiers are computation constrained and may need to "catch up" (i.e., verify a chain value based on a distant trusted chain value).

The Comb Skipchain construction has a higher communication overhead than Sandwich-chain, but it provides a very efficient setup mechanism ($O(n/\log(n))$) and a very efficient traversal ($O(1)$) with small storage overhead ($O(\log(n))$). This construction beats the previously established lower bound for the product of memory overhead and traversal overhead, which was $O(\log^2(n))$. In contrast, our construction achieves a memory-times-computation complexity of $O(\log(n))$. This is possible (and does not contradict the previous lower bounds) given that we move from one-dimensional to two-dimensional (or hierarchical) chains.

Our future work includes investigating deeper hierarchies of our constructions, and to establish general bounds for the one-way chain costs.

References

[1] 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*. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1998.

[2] Ross Anderson, Harry Manifavas, and Chris Sutherland. A practical electronic cash system. personal communication, December 1995.

[3] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Advances in Cryptology - Crypto '94*, pages 341–358, 1994. Lecture Notes in Computer Science Volume 839.

[4] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 1–15. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 1996.

[5] Giles Brassard, editor. *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, Santa Barbara, CA, USA, 1990. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

[6] M. Brown, D. Cheung, D. Hankerson, J. Hernandez, M. Kirkup, and A. Menezes. PGP in constrained wireless devices. In *Proceedings of the 9th USENIX Security Symposium*, pages 247–261. USENIX, August 2000.

[7] Cable Datacom News. Time warner division implements consumption caps. Published at `http://www.cabledatacomnews.com/may03/may03-7.html`.

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

[9] 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.

[10] S. Even, O. Goldreich, and S. Micali. On-line/off-line digital signatures. In Brassard [5], pages 263–277.

[11] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1986.

[12] Neil Haller. The S/KEY one-time password system. RFC 1760, February 1995.

[13] 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.

[14] 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.

[15] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, July 1980.

[16] Yih-Chun Hu, Adrian Perrig, and David B. Johnson. Efficient security mechanisms for routing protocols. In *Network and Distributed System Security Symposium, NDSS '03*, pages 57–73, February 2003.

[17] 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.

[18] J. M. Kahn, R. H. Katz, and K. S. Pister. Mobile networking for smart dust. In *ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99)*, Seattle, WA, August 1999.

[19] Leslie Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, November 1981.

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

[21] Mark Lomas, editor. *Security Protocols—International Workshop*, volume 1189 of *Lecture Notes in Computer Science*, Cambridge, United Kingdom, April 1997. Springer-Verlag, Berlin Germany.

[22] Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378, Santa Barbara, CA, USA, 1988. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

[23] Ralph C. Merkle. A certified digital signature. In Brassard [5], pages 218–238.

[24] Silvio Micali. Efficient certificate revocation. Technical Report MIT/LCS/TM-542b, Massachusetts Institute of Technology, Laboratory for Computer Science, March 1996. Technical memo.

[25] National Institute of Standards and Technology (NIST). Secure hash standard, May 1993. Federal Information Processing Standards (FIPS) Publication 180-1.

[26] Torben Pryds Pedersen. Electronic payments of small amounts. In Lomas [21], pages 59–68.

[27] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Xiaodong Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Symposium on Security and Privacy*, May 2000.

[28] 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.

[29] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comment RFC 1321, Internet Engineering Task Force, April 1992.

[30] Ronald L. Rivest and Adi Shamir. PayWord and MicroMint: Two simple micropayment schemes. In Lomas [21], pages 69 – 88.

[31] Pankaj Rohatgi. A compact and fast hybrid signature scheme for multicast packet. In Gene Tsudik, editor, *Proceedings of the 6th ACM Conference on Computer and Communications Security*, pages 93–100, Singapore, November 1999. ACM Press.

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

[33] 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.