Fast Verification of Hash Chains

Marc Fischlin*

Department of Computer Science & Engineering,
University of California, San Diego, USA

mfischlin@cs.ucsd.edu
http://www-cse.ucsd.edu/~mfischlin/

Abstract. A hash chain is a sequence of hash values $x_i = \mathsf{hash}(x_{i-1})$ for some initial secret value x_0. It allows to reveal the final value x_n and to gradually disclose the pre-images $x_{n-1}, x_{n-2}, \ldots$ whenever necessary. The correctness of a given value x_i can then be verified by re-computing the chain and comparing the result to x_n. Here we present a method to speed up the verification by outputting some extra information in addition to the chain's end value x_n. This information allows to relate the verifier's workload to a variably chosen security bound. That is, on input a putative chain value the verifier determines a security level (i.e., security against adversaries with at most T steps and success probability ϵ) and performs only a fraction $p = p(T, \epsilon)$ of the original work by using the additional information. We also show lower bounds for the length of this extra information.

1 Introduction

A hash chain, introduced by Lamport [Lam81], is a sequence of hash values $x_i = \mathsf{hash}(x_{i-1})$ for a seed x_0 where hash is some collision-intractable hash function (or some other publicly computable one-way function). Such a chain allows the owner of the seed to publish the chain's end value x_n and to stepwise release the pre-images $x_{n-1}, x_{n-2}, \ldots$ such that revealing x_{n-i} at step i does not help to find some of the values $x_{n-i-1}, \ldots, x_0$. The receiver can check the validity of some received x_{n-i} by re-calculating the chain starting with x_{n-i} up to x_n.

Hash chains have numerous applications. One of the best known is Micali's suggestion to use them as certificate chains [Mic96]. Roughly, for the user's public key pk of a signature or encryption scheme the certification authority (CA) publishes a certificate of x_n and pk (and possibly further information). For the i-th time period of some pre-determined length the CA hands the pre-image x_{n-i} to the user who can then provide this value as a certificate for his public key during this time period. To revoke the certificate the CA stops delivering the pre-images. Since it is infeasible to find the pre-image of the previously given value, forgery of certificates for future time periods is unlikely.

* This work was supported by the Emmy Noether Programme Fi 940/1-1 of the German Research Foundation (DFG).

A preliminary version appears in
RSA Security 2004 — Cryptographer's Track,
Lecture Notes in Computer Science, Springer-Verlag.

Other applications areas of hash chains include the design of micropayments schemes [HSW96,RS97], the S/KEY one-time authentication (RFC 1760) [Hal94,Hal95], securing routing information (e.g., [HPT97,HJP02,HPJ03]) and spam-fighting protocols [DGN03]. Similarly, one-way chains —where the hash function is replaced by a one-way function— have been deployed for the BiBa signature scheme [Per01] and for multicast authentication [PCST02].

In some of the aforementioned areas the verification procedure can be shortened significantly. Namely, if the verifier stores a previously verified chain value x_m for $m < n$, then the next time a value is presented, the verifier merely has to re-calculate the chain up to the stored value x_m. However, considering certificates for example, the owner of the seed may visit some sites only sporadically. Similarly, for routing protocols the information may be passed unfrequently. Or, the verifier may not be able to store previous chain values due to memory limitations or other restrictions. Finally, in some solutions, like the anti-spam solution of Dwork et al. [DGN03], the values are not released gradually but rather require the verifier to re-compute a full hash chain. Hence there are cases where large parts of the chain may still have to be verified.

Related Results. The need for faster verification of hash chains has immediately lead to so-called hash trees [Mer88]. Such constructs condense the long chains to tree-like structures such that the path from any value to the published root shrinks to logarithmic length. Unfortunately, in order to verify a given value the user has to supply logarithmically many inner nodes of the tree as a proof of correctness. Hence, two of the advantages of hash chains, low communication complexity and structural simplicity, vanish and are traded for faster verification.

Interestingly, quite a few efforts have dealt with the problem of fast *computation* of intermediate values x_i, for both chains and trees [CJ02,JLMS03,Sel03]. That is, if the user only stores the seed x_0 then, in order to release x_i of a hash chain, he needs to re-calculate the chain starting with x_0 up to x_i. It is preferable for the seed owner, of course, to keep some intermediate values $x_{i_1}, \ldots, x_{i_k}$ confidentially, and to recover any x_i from these values much faster.

The results in [CJ02,JLMS03,Sel03] give constructions for storing and recovering intermediate values of chains and trees. They also give lower bounds showing that the constructions are optimal with respect to time/storage tradeoffs. However, none of these solutions improves the *verification* time. This is especially true for the hash chains, and which would lessen the disadvantage of chains versus trees.

Our Results. Our solution is to let the owner of the seed generate some supplementary information which is published together with the chain's end value x_n. This extra information then allows to improve the verification time when the verifier is presented an allegedly correct chain value. Specifically, for security bound T and ϵ on the adversarial running time and success probability, respectively, our construction allows to decide correctness after roughly a fraction $p = (\log T + \log \frac{1}{\epsilon})/128$ of the original workload. Here, the workload is the number of hash function evaluations (i.e., it is equal to i if x_{n-i} is given).

Alternative choices for the constant 128, depending on parameter settings, are possible.

The interesting property of our construction is that the two security parameters T and ϵ can be chosen individually by any verifier, even differently for each verification run. Once the verifier has selected "his" security level this determines the fraction $p = p(T, \epsilon)$ of hash chain computations. In other words, the more liberal the verifier chooses the security level the less work he has to carry out. In this sense, the property is related to the notion of progressive verification, recently introduced in the context of message authentication in [Fis03].

We emphasize that the security level (T, ϵ) for the fast verification should not be confused with security of the hash functions against collision-finders. As for collisions we know that, by the birthday paradox, collisions for hash functions with n-bits output can be generated with probability more than $1/2$ within $2^{n/2}$ steps. Once such a collision is found the complete hash chain becomes disaffected. In our model, we simply assume that finding such collision is beyond feasible attacks. Security here refers to attacks in which the verifier should be forced to perform more than a fraction $p(T, \epsilon)$ of the work; even if the adversary overcomes this security bound the verifier can may still raise the level for the next verification.

In our solution the extra information the seed owner attaches to x_n is called a check-bit vector. As explained, this check-bit vector is a universal parameter enabling different security/workload levels for the verifiers. Another interesting characteristic, in addition to the time improvement, is the length of such check-bit vectors: very long vectors may outweigh the gain in verification time. We therefore investigate lower bounds for this length.

The bounds on the length of check-bit vectors vary with the way the vectors are created. In the most simple case the extra information is chosen according to the time period i and merely consist of some fixed number of the bits of the intermediate value x_{n-i}. For this type of schemes, under which our construction falls, we show that approximately[1] $(\log T - \log n + \log \frac{1}{\epsilon}) \log n$ bits are required. In comparison, our solution produces check-bit vectors of about $128(\log_2 n - 4)$ bits, which for $n = 1,024$, $T = 2^{40}$ and $\epsilon = 2^{-20}$ and $p = 48.5\%$ for example, yields respectable 768 bits. Still, this is better than the usual $160 \log_2 n = 1,600$ bits to communicate the inner nodes of a tree, and the communication of the public check-bit vector amortizes over the time periods. Yet, hash trees are usually much faster verifiable, in particular with respect to "standard" security levels.

Organization. In Section 2 we define check-bit schemes and their security formally. We present our lower bounds in Section 3 and our construction appears in Section 4. We conclude with a brief discussion in Section 5.

2 Definition

In the most simple form, a hash chain (for a given length parameter n) can be described by two algorithms $\mathcal{G}$ and $\mathcal{V}$, the generator and verification algorithm.

[1] The bound depends on further parameters not discussed here in the introduction.

The former algorithm simply chooses a random x_0 and computes the chain up to x_n, and the verifier on input x_n and some putative chain value x for time period i merely checks that $\mathsf{hash}^i(x) = x_n$.

Check-Bit Schemes. Here we augment the basic hash chain generation and verification. Algorithm $\mathcal{G}$, when generating the chain for seed x_0, repeatedly runs a deterministic selection algorithm $\mathcal{S}$ as subroutine for each hash function iteration. For each such execution, for $i = n-1$ down to 0, algorithm $\mathcal{S}$ produces a string cb_i (possibly the empty string λ), which is determined by the time period number i, the intermediate value $x_{n-i} = \mathsf{hash}^{n-i}(x_0)$ and the preceding strings $\mathsf{cb}_{i+1}, \ldots, \mathsf{cb}_{n-1}$.

The so-called check-bit vector cb is the concatenation of all strings $\mathsf{cb}_0, \mathsf{cb}_1, \ldots, \mathsf{cb}_{n-1}$, ordered according to the release time. We assume that the position of cb_i within cb and its length are recoverable from cb; this clearly inhibits lossy encodings and we thus call the constructions allowing to recover cb_i *schemes with lossless encoding*. Nonetheless, since we only deal with such schemes throughout the paper we often drop this appendix. Let $\mathsf{cb}_{\geq i}$ be the string $\mathsf{cb}_i || \ldots || \mathsf{cb}_{n-1}$ and set $\mathsf{cb}_{>i} = \mathsf{cb}_{\geq i+1}$ for $i \leq n-1$ (where $\mathsf{cb}_{>n-1} = \lambda$).

As before, the verifier $\mathcal{V}$ takes a value x and integer i together with the chain's end value x_n as input, and verifies that x is the correct pre-image for time period i. This time, however, the verifier also gets the check-bit vector cb as extra input and uses this value to shorten the verification: For each hash function iteration in time period j the verifier now also calls $\mathcal{S}(j, \mathsf{hash}^{j-i}(x), \mathsf{cb}_{>j})$ and compares the result to the given cb_j. If a mismatch occurs then reject, else continue (possibly up to the chain's end).

Moreover, the verifier gets two parameters T and ϵ representing the bounds on the adversarial running time and success probability (both characteristics are specified below). Instructively, one may think of these two variable security parameters as determined by $\mathcal{V}$ before starting the verification, although for ease of notation we sometimes set these parameters instead and then provide these fixed values to the verifier.

Definition 1. *A check-bit scheme with lossless encoding and for parameter n is a triple $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ of algorithms (of which $\mathcal{G}$ is probabilistic) such that*

Algorithm $\mathcal{G}$:

- *picks a seed x_0 according to some efficiently samplable distribution,*
- *computes $x_i = \mathsf{hash}(x_{i-1})$ for $i = 1, 2, \ldots, n$,*
- *computes $\mathsf{cb}_i = \mathcal{S}(i, x_{n-i}, \mathsf{cb}_{>i})$ for $i = n-1, \ldots, 0$,*
- *outputs (x_0, x_n, cb).*

Algorithm $\mathcal{V}$:

- *gets inputs x_n, cb and x, an integer i as well as T and ϵ,*
- *repeats the following until $i = 0$ or halt:*
  - *if $\mathsf{cb}_i \neq \mathcal{S}(i, x, \mathsf{cb}_{>i})$ then reject and stop*[2]
  - *else set $i \leftarrow i-1$ and $x \leftarrow \mathsf{hash}(x)$*

[2] Here $\mathcal{V}$ recovers $\mathsf{cb}_i, \mathsf{cb}_{>i}$ from cb.

– *if* $x = x_n$ *then accept, else reject.*

Algorithm $\mathcal{S}$:

– *takes an integer* i*, a value* x *and a string* $\mathsf{cb}_{>i}$ *as input,*
– *computes and returns* $\mathsf{cb}_i = \mathcal{S}(i, x, \mathsf{cb}_{>i})$.

In addition, the scheme is complete, i.e., the verifier never rejects a valid input x_n, cb, i *and* $x = x_{n-i}$ *produced by* $\mathcal{G}$*, independently of* T *and* ϵ.

Note that the selection algorithm $\mathcal{S}$ is defined to be deterministic. On one hand, this simplifies the definition and analysis significantly. On the other hand, it does not weaken the model too much. Namely, for a given hash function hash define $\mathsf{hash}'(x_i||r) = \mathsf{hash}(x_i)||r$ such that r remains unchanged during the iterations. If $x_0||r$ is chosen at random by $\mathcal{G}$ then $\mathcal{S}$ can use r as externally provided random coins. This corresponds, of course, to public coins, as the right part of the chain's end value $x_n||r$ is output, too. However, public randomness ensures that *any* verifier can re-calculate the selection algorithm's output and compare it to the given check-bit vector.

Attacks. In order to define security we have to specify the attack mode first. We measure the running time T of the adversary by counting the hash function evaluations only. Formally, we therefore provide the attacker with an oracle $\mathsf{hash}(\cdot)$ which she can access, but for which "guessing" images, i.e., generating images without querying the oracle, is infeasible. We let $\epsilon \in [0, 1)$ represent a bound on the adversary's success probability

We also introduce a parameter $p \in [0, 1)$ which bounds the fraction of the original work the verifier performs, at least if the position of the given value exceeds a certain distance Δ from the end. This offset allows to overcome the problem of verifying values within the given bound if the values are too close to the end, e.g., checking the second to last value with less than 50% of the work. Nonetheless, the offset should be small and, in particular, not depend on the chain length n in order to rule out trivial solutions like $\Delta = n$. It may, however, depend on the adversarial bounds because Δ is likely to depend on p which, in turn, varies with T and ϵ.

We now define the following experiment for a check-bit scheme $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ with parameter n:

Experiment $\mathrm{Exp}_{\mathcal{A}}(T, \epsilon, p, \Delta)$:

– Algorithm $\mathcal{G}$ generates (x_0, x_n, cb).
– The adversary $\mathcal{A}$ gets as input (x_n, cb). The adversary also gets access to the hash oracle $\mathsf{hash}(\cdot)$ and an oracle $\mathsf{Release}(\cdot)$ which takes integers j as input and returns x_{n-j}. Let r denote the maximum over all queries to $\mathsf{Release}$ (where $r = 0$ if $\mathcal{A}$ has never queried the oracle).
– In addition to oracle queries the adversary performs internal computations and finally outputs (x, k).
– The verifier $\mathcal{V}$ is invoked on $(x_n, \mathsf{cb}, x, k, T, \epsilon)$ and returns a decision after V hash function evaluations.

We say that adversary $\mathcal{A}$ wins experiment $\mathrm{Exp}_{\mathcal{A}}(T, \epsilon, p, \Delta)$,

- if the adversary makes at most T hash function evaluations, and
- if the adversary has queried the hash oracle about $\mathsf{hash}^i(x)$ for all $i = 0, 1, \ldots, \lfloor pk \rfloor - 1$, and
- if the adversary has queried the oracle Release only about values smaller than k, i.e., if $k > r$, and
- if the position k exceeds the offset, i.e., if $k \geq \Delta$, and
- if the verifier does not reject within $V \leq pk$ hash function evaluations.

The mere purpose of letting the adversary query about the output is to charge the adversary's running time also for the time to verify the output.

Security. Informally, a check-bit scheme is (T, ϵ, p, Δ)-verifiable if no adversary running in time T can cause the verifier to perform a fraction p or more of the work with probability more than ϵ. Here, the work refers to the number k of hash function evaluations required to verify the correct value x_{n-k} at time period k. As explained above, we usually envision the security bound as chosen by the verifier, and that this bound then determines the required fraction of the work. In this sense, $p = p(T, \epsilon)$ is a function of the security level, and we call a check-bit scheme (p, Δ)-verifiable if for any (T, ϵ) it is $(T, \epsilon, p(T, \epsilon), \Delta)$-verifiable. More formally,

Definition 2. *A check-bit scheme $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ with parameter n is (T, ϵ, p, Δ)-verifiable if, for any adversary $\mathcal{A}$ running in time at most T, the probability of $\mathcal{A}$ winning experiment $\mathrm{Exp}_{\mathcal{A}}(T, \epsilon, p, \Delta)$ is at most ϵ. The scheme is (p, Δ)-verifiable if, for any adversary $\mathcal{A}$ and any T, ϵ, the probability of $\mathcal{A}$ winning experiment $\mathrm{Exp}_{\mathcal{A}}(T, \epsilon, p(T, \epsilon), \Delta)$ is at most ϵ.*

We have chosen a relative bound to measure the work to be performed, i.e., if $p = 1/2$ then at time period $3n/4$ the verifier needs $3n/8$ hash evaluations, at time period n the verifier has to compute $n/2$ hash values etc. Alternatively, one may define an absolute bound saying that the verifier has to do $w = w(T, \epsilon)$ (or less) hash function evaluations, independently of the time period. But first note that such an absolute bound easily follows if we set $w = pn$. Second, some applications may bear in mind that verification is faster for the first time periods. In this case, it is preferable to have a relative work reduction saying that you save up to 50%, for instance, at any time period and independent of the length n of the chain.

3 Lower Bounds

We first show a lower bound for special check-bit procedures in Section 3.1. This bound holds for arbitrary security parameters T, ϵ and thus even yields a bound for the more liberal case of (T, ϵ, p, Δ)-verifiable schemes. The bound says that the selection algorithm $\mathcal{S}$ must essentially generate check-bit vectors of $(\log T - \log hn + \log \frac{1}{\epsilon}) \log \frac{n}{\Delta}$ bits, where h is the maximum number of hash

function evaluations for each of the n iterations (including the ones for the computation of $\mathcal{S}$).

The bound above holds for selection algorithms where the length of the output cb_i may depend on the position i but not the intermediate value. We call such schemes *position-driven* selection algorithms:

Definition 3. *Let $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ be a check-bit scheme (for parameter n). Algorithm $\mathcal{S}$ is position-driven if for any two seeds x_0, y_0 we have $|\mathsf{cb}_i(x_0)| = |\mathsf{cb}_i(y_0)|$.*

In general, the length of cb_i may depend on the preceding values or check bits as well, and thus $|\mathsf{cb}_i(x_0)|$ can be different from $|\mathsf{cb}_i(y_0)|$. In this case, the generator $\mathcal{G}$ possibly outputs some seeds x_0 with very short check-bit vectors. For such schemes we yet show in Section 3.2 that check-bit vectors with only a slightly smaller length than above must still be produced with high probability.

For both bounds, i.e., even if $|\mathsf{cb}_i(x_0)| \neq |\mathsf{cb}_i(y_0)|$, we make the following assumption which basically says that the adversary will find matching check bits for random samples with at least the guessing probability.

Assumption 1. *Let $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ be a check-bit scheme with lossless encoding and parameter n. Then, for any i, we assume that for random x_0, y_0 the probability that $\mathsf{cb}_i(x_0) = \mathsf{cb}_i(y_0)$ is at least $2^{-\min\{|\mathsf{cb}_i(x_0)|, |\mathsf{cb}_i(y_0)|\}}$. The probability is over the choice of x_0 and y_0.*

3.1 Lower Bound for Position-Driven Selection Algorithms

Throughout this section we use the following notation (visualized in Figure 1): We let $[1, n]$ be the set of integers between 1 and n. Each integer represents the number of hash function evaluations that are required to verify a given value x at time period i.

We divide $[1, n]$ into disjoint intervals. For this, let $\alpha_0, \ldots, \alpha_I$ be a sequence of increasing values with $\alpha_0 = 0$ and $\alpha_I = 1$ for an appropriate integer I (which we will specify later). For $\ell = 1, \ldots, I$ define the ℓ-th interval $\mathcal{I}_\ell$ to be $[\alpha_{\ell-1} n + 1, \alpha_\ell n]$, where we assume for simplicity that all $\alpha_\ell n$'s are integers.

Let $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ be a (T, ϵ, p, Δ)-verifiable check-bit scheme with a position-driven selection algorithm. For a seed x_0 chosen by $\mathcal{G}$ let c_ℓ be the number of check-bit positions in the interval $\mathcal{I}_\ell$. Note that, by assumption, c_ℓ does not depend on x_0. The sum over all c_ℓ's for $\ell > \log \Delta$ is therefore the total number of check bits for which we prove our lower bound.

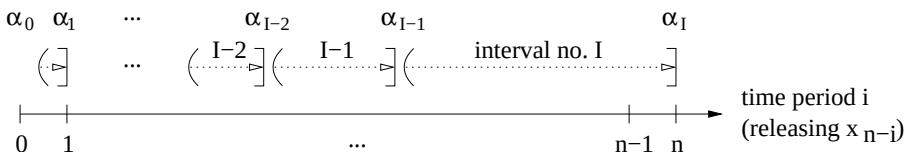

Fig. 1. Idea of Lower Bound

In the sequel we set $q = 1 - p$ for $p > 0$ of the (T, ϵ, p, Δ)-verifiable scheme and we let $\alpha_\ell = q\alpha_{\ell+1}$ for $\ell = I-1, \ldots, 1$. Then $\alpha_\ell = q^{I-\ell}$ for $\ell \geq 1$ and each interval $\mathcal{I}_\ell$ is of size $p\alpha_\ell n$ and by a factor $1/q$ larger than the previous one. Recall that we also assume that $\alpha_\ell n$ is an integer for all ℓ, thus n must be a power of $1/q$ and we must have $I = \log_{1/q} n$ for the number I of intervals. For instance, for $p = 1/2$ we have $\log_2 n$ intervals, each one half the size of the following one.

Let h be the maximum of $\mathcal{G}$'s hash function evaluations when computing x_i and cb_i in some i-th step. Then h includes the single evaluation to derive the next chain value and at most $h - 1$ hash function computations of $\mathcal{S}$.

Lemma 1. *For all* $\ell = \lceil \log_{1/q} \Delta \rceil, \ldots, I$ *we have*

$$c_\ell \geq \log_2 T - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon}$$

Note that, for very small ϵ, the term $\log_2 \ln \frac{1}{1-\epsilon}$ becomes roughly $\log_2 \epsilon$. Hence, the smaller the error should be the more check bits are requried.

Proof. Suppose that for some interval $\mathcal{I}_\ell$ the number c_ℓ is strictly less than the given bound. We show how to construct an adversary $\mathcal{A}$ then that runs at most $T = 2^t$ steps and succeeds with probability more than ϵ in making the verifier evaluate more than a fraction p of the $k = \alpha_\ell n$ iterations for $x_{n-\alpha_\ell n}$, i.e., beyond interval $\mathcal{I}_\ell$. Note that $k \geq \Delta$ by construction.

Adversary $\mathcal{A}$ repeats the following at most $r = T/hn$ times. $\mathcal{A}$ selects a random seed y_0 and iterates the hash function until all check bits c_ℓ in interval $\mathcal{I}_\ell$ have been computed. If these check bits match the original ones, then output $x = \mathsf{hash}^{n-\alpha_\ell n}(y_0)$ and stop, else repeat.

Note that the adversary's running time is certainly bounded above by T. This holds since the computation of the c_ℓ check bits via the position-driven selection algorithm in each round requires at most hn hash function iterations, and since the number of repetitions is at most r.

It remains to calculate the success probability. In each loop the probability of $\mathcal{A}$ finding a value x for which the check bits match is, by Assumption 1, at least 2^{-c_ℓ}. Hence, the probability that $\mathcal{A}$ does not find a suitable x during all r rounds is at most:

$$\begin{aligned}
\left(1 - 2^{-c_\ell}\right)^r &\leq \exp\left(-r2^{-c_\ell}\right) = \exp\left(-2^{t-\log_2 hn-c_\ell}\right) \\
&< \exp\left(-2^{t-\log_2 hn-\left(t-\log_2 hn-\log_2 \ln \frac{1}{1-\epsilon}\right)}\right) \\
&= \exp\left(-2^{\log_2 \ln \frac{1}{1-\epsilon}}\right) = \exp\left(-\ln \tfrac{1}{1-\epsilon}\right) \\
&= 1 - \epsilon
\end{aligned}$$

The probability of $\mathcal{A}$ finding such an x is therefore strictly more than ϵ, contradicting the security of the scheme. Therefore, the assumption about c_ℓ falling below the bound must be false. □

We immediately get from the previous lemma:

Theorem 2. *Let $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ be a (T, ϵ, p, Δ)-verifiable check-bit scheme with lossless encoding and parameter n. Let $\mathcal{S}$ be a position-driven selection algorithm and assume that Assumption 1 holds. Presume further that the computation of a chain of length n requires at most hn hash function evaluations. Then the length of the check-bit vector is at least*

$$\left(\log_2 T - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon}\right) \log_{1/(1-p)} \tfrac{n}{\Delta}$$

Proof. According to the lemma, for each $\ell \geq \lceil \log_{1/q} \Delta \rceil$ and interval $\mathcal{I}_\ell$ we have for the number of check bits:

$$c_\ell \geq t - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon}$$

It follows for the overall number of check bits:

$$\sum_{\ell=\lceil \log \Delta \rceil}^{I} c_\ell \;\geq\; \left(t - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon}\right) \cdot \left(\log_{1/q} n - \log_{1/q} \Delta\right)$$

This proves the lower bound. □

For example, if $n = 1,024$, $h = 1$ and the verifier chooses a security level of $T = 2^{40}$ and $\epsilon = 2^{-20}$, then for $p = 1/2 = q$ and offset $\Delta = 64$ we need approximately $(40 - 10 + 20) \cdot (10 - 6) = 200$ bits.

3.2 Lower Bound for General Check-Bit Schemes

For non-position-driven selection algorithms the size of the output cb_i may vary with the intermediate values. Luckily, we can modify the proof above to obtain a slightly relaxed bound.

Take all the values α_ℓ, $\mathcal{I}_\ell$ etc. as in the previous section and let $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ be a check-bit scheme, not necessarily with a position-driven selection algorithm. Let c_ℓ denote again the number of check bits in interval $\mathcal{I}_\ell$ —which now is a random variable over $\mathcal{G}$'s choice. In addition, fix some constant $a \in (0, 1)$.

Lemma 2. *The probability that $\mathcal{G}$ picks a seed x_0 such that*

$$c_\ell \geq \log_2 T - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon^{1-a}} \quad \textit{for all } \ell = \lceil \log_{1/q} \Delta \rceil, \ldots, I$$

is at least $1 - (I - \log_{1/q} \Delta)\epsilon^a$.

Substituting $\log_2 \ln \frac{1}{1-\epsilon^{1-a}}$ by the approximation $\log_2 \epsilon^{1-a}$ again, the success probability now enters as $(1-a)\log_2 \epsilon$. Hence, the smaller a the larger the vector length —but the smaller the probability of outputting such a long vector as well.

Proof. Suppose for sake of contradiction that this probability is less than $1-(I-\log_{1/q} \Delta)\epsilon^a$. Then there exists a fixed ℓ_0 in the range such that the probability of $\mathcal{G}$ picking a seed x_0 such that

$$c_{\ell_0} < \text{bound}_{\ell_0} := \log_2 T - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon^{1-a}}$$

is at least ϵ^a.

Next, as in the previous case, we construct an adversary $\mathcal{A}$ trying to cause more than a fraction p of the work for interval $\mathcal{I}_{\ell_0}$ with probability more than ϵ. $\mathcal{A}$ repeats the following $r = T/hn$ times. $\mathcal{A}$ selects a random seed y_0 and computes the chain for this seed up to time period $\alpha_{\ell_0} n$. Let x be $\mathsf{hash}^{n-\alpha_{\ell_0} n}(y_0)$. The adversary continues to iterate the hash function $(\alpha_{\ell_0} - \alpha_{\ell_0 - 1})n$ times. If the check bits do not match the given ones then repeat the process. Else return x.

The running time of $\mathcal{A}$ is bounded above by T since the adversary makes at most hn hash function iterations for each of the r tries. As for the success probability, condition on the event that $\mathcal{G}$ outputs some x_0 for which $c_{\ell_0} < \mathrm{bound}_{\ell_0}$. This happens with probability at least ϵ^a. Next note that the adversary succeeds if the at most bound_{ℓ_0} bits of the attempt match. This happens with probability at least $2^{-\mathrm{bound}_{\ell_0}}$ according to Assumption 1.

Hence, under the condition that $\mathcal{G}$'s seed x_0 causes c_{ℓ_0} to be less than the bound, it follows as before that $\mathcal{A}$ fails with probability

$$\left(1 - 2^{-\mathrm{bound}_{\ell_0}}\right)^r < 1 - \epsilon^{1-a}$$

The probability that $\mathcal{A}$ succeeds in the experiment is therefore more than ϵ^{1-a} times the probability that $c_{\ell_0} < \mathrm{bound}_{\ell_0}$ for $\mathcal{G}$'s output. Multiplying these two probabilities we obtain a successful attack with probability more than ϵ. Thus the initial assumption must have been wrong. □

Theorem 3. *Let $(\mathcal{G}, \mathcal{V}, \mathcal{S})$ be a (T, ϵ, p, Δ)-verifiable check-bit scheme with lossless encoding and parameter n. Presume that the computation of a chain of length n requires at most hn hash function evaluations and let $a \in (0, 1)$ be a constant. Then, under Assumption 1, with probability at least $1 - \epsilon^a \log_{1/(1-p)} \frac{n}{\Delta}$ (over $\mathcal{G}$'s seed choice) the check-bit vector has at least*

$$\left(\log_2 T - \log_2 hn - \log_2 \ln \tfrac{1}{1-\epsilon^{1-a}}\right) \log_{1/(1-p)} \tfrac{n}{\Delta}$$

bits.

4 Constructions of Check-Bit Schemes

In this section we present our check-bit scheme. We start with an elementary attempt which provides an *absolute* work bound of $w = \log_2 T + \log_2 \frac{1}{\epsilon}$ hash function evaluations for the desired security parameter. However, the *relative* performance (relative to the time period and the original number of hash function evaluations) is rather bad, so we elaborate on a construction with relative bound $p = (\log_2 T + \log_2 \frac{1}{\epsilon})/128$. This, unfortunately, comes with an increase in the length of check-bit vector.

4.1 Construction with Absolute Bound

In our construction with absolute bound the selection algorithm $\mathcal{S}$ simply outputs the least significant bit of intermediate value x_{n-i} for each percent of computation (i.e., if $i = \lfloor jn/100 \rfloor$ for some j). Here, the value 100 is chosen rather

arbitrarily; any other granularity may be selected as well. The verifier, when checking some input x, i, then merely compares the least significant bits of the intermediate values when re-calculating the chain, and stops if a mismatch occurs.

Construction 4. *The check-bit scheme* $(\mathcal{G}_{abs}, \mathcal{V}_{abs}, \mathcal{S}_{abs})$ *with parameter* $n > 100$ *is described by the following selection algorithm:*

Algorithm $\mathcal{S}_{abs}(x, i)$*:*

if $i = \lfloor jn/100 \rfloor$ *for some* $j \in \{1, \ldots, 100\}$
then output $\mathsf{cb}_i =$ *[least signifcant bit of* x*]*
else output $\mathsf{cb}_i = \lambda$

Note that the length of the check-bit vector is constant and adds 100 bits to the public chain's end value x_n of typically 160 or 256 bits.

The idea of the scheme is as follows. Suppose that the distribution of the bits is approximately uniform, and that the adversary cannot do better than computing chains for randomly chosen seeds. Then, for such a seed the probability of hitting $w = \log_2 T + \log_2 \frac{1}{\epsilon}$ of the given check bits is at most $2^{-w} = \epsilon T^{-1}$. Hence, the overall success probability of the adversary making T or less steps is at most ϵ.

The scheme, as is, does not provide a reasonable *relative* security level, though. For instance, consider the time period k which is $\log_2 \frac{1}{\epsilon} - 1$ percent from the end value n. An adversary that outputs a random x together with k makes $\mathcal{V}$ evaluate the whole hash function till the end with probability 2ϵ (because there are at most $\log_2 \frac{1}{\epsilon} - 1$ check bits in this interval). Hence, for any given ϵ the verifier performs 100% of the original computation with probability more than ϵ for some point k. We remark that this argument still holds if there is a small offset Δ because $k = (\log_2 \frac{1}{\epsilon} - 1)n/100$ eventually surpasses this value for large n's.

Because of our interest in relative bounds we omit a formal security statement and analysis of this scheme here and turn to the next construction instead.

4.2 Construction with Relative Bound

The problem with the approach in the previous subsection is that the check bits are distributed equidistantly over the chain of length n. Yet, the workload of the verifier varies with the distance to the end value and is thus relative to the position. The idea is now to increase the density of check bits towards the end of the chain such that the number of check bits compensates for the reduced work towards the first time periods.

Construction. For our construction we also let the time period number enter into the hash function computation. We formally reduce this to the basic case as follows. Let $\langle \cdot \rangle$ be some fixed-length encoding (such that the encoding is one-to-one for integers $0, 1, \ldots, n$) and let $\mathsf{hash}'(\cdot)$ be a hash function. Then we define a function hash for inputs $x_i = \langle i \rangle || x'_i$ by

$$\mathsf{hash}(x_i) = \langle i + 1 \rangle || \mathsf{hash}'(\langle i \rangle || x'_i)$$

For random x'_0 we can now set the start value as $x_0 = \langle 0 \rangle || x'_0$ and derive the chain by iterating hash on x_0. The verifier, when presented a hash value $x_i = \langle i \rangle || x'_i$, should now also check that i matches the current time period. Observe that the least significant bits of the chain values are still well distributed if hash' is an appropriate hash function.

Next we specify our construction. We partition the chain of length n into $I = \log_2 n$ intervals of length $1, 2, 4, \ldots, n/4, n/2$. For ease of notation we presume that n is a power of 2. For $\ell = 1, \ldots, I$ interval $\mathcal{I}_\ell$ ranges from $2^{\ell-1}$ to $2^\ell - 1$ (and we add the point n to interval $\mathcal{I}_I$).[3] We also introduce an even value B, the base, which determines the density of the check bits. Typically, $B = 100$ as in the previous section or, for ease of implementation, B should be a power of two, say 128.

In interval $\mathcal{I}_I$ we let $\mathcal{S}_{\mathrm{rel}}$ output the least significant bit of the intermediate values at positions jn/B. In interval $\mathcal{I}_{I-1}$ we double the check bits by outputting the bits of each value at position $jn/2B$. In general, we output the least significant bit of value x_{n-i} for $i \in I_\ell$ if $i = jn/(B \cdot 2^{I-\ell}) = j2^\ell/B$ (with appropriate rounding).

Another refinement is to return the b least significant bits instead of a single one only. This improves the error detection probability. We thus define our check-bit scheme with respect to parameters b and B which are arbitrary integers (except that B is even) and which are fixed for a specific instance.

Construction 5. *The check-bit scheme $(\mathcal{G}_{rel,b,B}, \mathcal{V}_{rel,b,B}, \mathcal{S}_{rel,b,B})$ with parameter $n > B$ is described by the following selection algorithm:*

Algorithm $\mathcal{S}_{rel,b,B}(x, i)$:

`if` $i \in \mathcal{I}_\ell$ *and* $i = \lfloor j2^\ell/B \rfloor$ *for some* $j \in \{\frac{1}{2}B, \frac{1}{2}B + 1, \ldots, B - 1\}$
`then` *output* $\mathsf{cb}_i = [b$ *least significant bits of* $x]$
`else` *output* $\mathsf{cb}_i = \lambda$

For each interval $\mathcal{I}_\ell$ such that $|\mathcal{I}_\ell| = 2^{\ell-1} \geq B/2$ the variable j runs through $B/2$ values, for each such values producing b bits output. The union of the remaining intervals with $\ell < \log_2 B$ ranges (at most) from 1 to B. Hence, there are at most B further indices producing a non-empty check-bit output. The overall length of a check-bit vector is therefore bounded by:

$$\tfrac{1}{2}bB \cdot (I - \log_2 B + 1) + bB = \tfrac{1}{2}bB \cdot \big(\log_2 \tfrac{n}{B} + 3\big).$$

If we choose $B = 128$ and $b = 2$, for instance, then we get a check-bit vector of maximal $128(\log_2 n - 4)$ bits, and for $n = 1,024$ the check-bit vector is at most 768 bits. We remark that we have rounded off some values for sake of readability; depending on the choice of parameters the actual length may even be smaller.

To eliminate the division with rounding the base B can be chosen as a power of two, $B = 2^\beta$ with, say, $\beta = 7$. Then, with j running through $2^{\beta-1}$ and $2^\beta - 1$, the rounded values of $j2^\ell/B = j2^{\ell-\beta}$ equal the integers $2^{\ell-1}, \ldots, 2^\ell - 1$. To check that i matches one of these values, it therefore suffices to check that $i \gg (\ell - 1)$, i.e., i with the least significant $\ell - 1$ bits knocked off, equals 1.

[3] In contrast to the lower bound we move the intervals one position to the left to simplify the implementation.

Assumptions. In order to show security we first need to specify the assumptions about the hash function. The first assumption basically says that finding pre-images for the given chain is impossible, at least within a given bound like $T_0 \approx 2^{60}$ and $\epsilon_0 \approx 2^{-40}$. In particular, giving away some bits of pre-images of chain values must not facilitate the search.

We remark that the parameters T_0, ϵ_0 should not be confused with the flexible and lightweight parameters T, ϵ of our improved verification procedures. The values T_0, ϵ_0 are large bounds for adequate security against inverters and collision-finders. They are determined by the choice of the hash function and are usually fixed. Still, they limit of course the choices for T, ϵ because the least requirement to make our fast verification algorithm work, is that the chain is not compromised.

To formulate the assumption we can use the description of experiment $\mathrm{Exp}_{\mathcal{A}}$:

Assumption 6. *For the scheme $(\mathcal{G}_{rel,b,B}, \mathcal{V}_{rel,b,B}, \mathcal{S}_{rel,b,B})$ in Construction 5 for any adversary $\mathcal{A}$ running in time at most T_0 the probability of $\mathcal{A}$ winning experiment $\mathrm{Exp}_{\mathcal{A}}(T_0, \epsilon_0, 1, 1)$, i.e., the experiment with bounds $p = 1$ and $\Delta = 1$, is at most ϵ_0.*

Note that the bound $p = 1$ implies that the verifier can check the complete chain and, in particular, also compares the final value to the original end value. The assumption therefore says that the adversary is not able to find (x, k) within the success bound (T_0, ϵ_0) such that x has not been released before and such that $x_n = \mathsf{hash}^k(x)$. It follows that $x_{n-i} \neq \mathsf{hash}^{k-i}(x)$ for any i with $1 = \Delta \leq i \leq k$. In conclusion, according to the assumption the adversary is not able to find collisions (or possibly the pre-image itself) except with some very small probability, even if given some additional information in form of the check-bit vector.

Although the adversary may not be able to find a pre-image of the chain's end value, it might still be possible to find a related pre-image such that large parts of the check-bit vectors coincide. The following assumption rules this out:

Assumption 7. *For any two seeds x_0, y_0 such that $\mathsf{hash}^i(x_0) \neq \mathsf{hash}^i(y_0)$ for all $i = 0, 1, \ldots, n$ we assume that the check-bit vectors $\mathsf{cb}(x_0)$ and $\mathsf{cb}(y_0)$ generated by $\mathcal{S}_{rel,b,B}$ are uniformly and independently distributed strings of the corresponding length (where the probability is over the choice of the hash function hash).*

Note how this assumption captures the adaptive queries of the adversary to $\mathsf{Release}(\cdot)$ in an attack. Specifically, the assumption quantifies over all seeds and thus, even if the adversary knows a seed x_0 generated by $\mathcal{G}_{\mathrm{rel},b,B}$, it is infeasible to find a different seed complying with (parts of) $\mathsf{cb}(x_0)$ better than with trial-and-error.

Both assumptions are satisfied if hash is for example modelled as a random oracle [BR93]. In this case, inverting the chain or finding collisions is extremely unlikely and the bits are then uniformly and independently distributed. From a practical point of view, well-known hash functions like SHA-1 and RIPEMD-160 seem to approximate the assumptions quite well. To best of our knowledge the distribution of the least significant bits is not known to be biased significantly. Similarly, providing very few bits of a pre-image is not known to substantially help inverting the hash function. See [BK03] for results.

Security. We next prove security of our construction under the stated assumptions:

Theorem 8. *Under Assumption 6 (for parameters T_0, ϵ_0) and Assumption 7 the check-bit scheme $(\mathcal{G}_{rel,b,B}, \mathcal{V}_{rel,b,B}, \mathcal{S}_{rel,b,B})$ in Construction 5 is a (p, Δ)-verifiable check-bit scheme for*

$$p = \frac{\log_2 T + \log_2 \frac{1}{\epsilon - \epsilon_0} + b}{\frac{1}{2} bB} \qquad \text{and} \qquad \Delta = B/2$$

if $T \leq T_0$ and $\epsilon_0 \leq \epsilon$ (and $p = 1$ otherwise). For chains of length n the check-bit vectors have at most

$$\tfrac{1}{2} bB \cdot \left(\log_2 \tfrac{n}{B} + 3 \right)$$

bits.

Proof. The case $p \geq 1$ is trivial, so we condition on $p < 1$, $T \leq T_0$ and $\epsilon_0 \leq \epsilon$. Consider the adversary's final output (x, k) with $k \geq \Delta$. Let ℓ be the interval number in which k lies, i.e.,

$$\frac{n}{2 \cdot 2^{I-\ell}} \leq k < \frac{n}{2^{I-\ell}}$$

Then, one percent of the work to verify the pair (x, k) corresponds to at least $\frac{1}{2} \cdot \frac{1}{2^{I-\ell}}$ percent of the work to verify the whole chain. By construction, on the other hand, the density of the check bits in each interval $\mathcal{I}_i$ for $i \leq \ell$ is at least $2^{I-i} \geq 2^{I-\ell}$ times the one of check bits in $\mathcal{I}_I$. Hence, if we perform a fraction p of the verification work for (x, k), reading at least a fraction $p2^{-(I-\ell+1)}$ of the complete chain, then we consult at least

$$\begin{aligned} & b \lfloor p 2^{-(I-\ell+1)} \cdot B 2^{I-\ell} \rfloor \\ & \quad = b \lfloor pB/2 \rfloor = b \lfloor \tfrac{\log_2 T + \log_2 \frac{1}{\epsilon - \epsilon_0} + b}{b} \rfloor \geq \log_2 T + \log_2 \tfrac{1}{\epsilon - \epsilon_0} \end{aligned}$$

check bits in total. Note that $\lfloor pB/2 \rfloor \leq p\Delta \leq k$ and therefore we do not reach the end of the chain before accessing all these check bits. In summary, the adversary must find an x such that it matches at least $\log_2 T + \log_2 \frac{1}{\epsilon - \epsilon_0}$ check bits.

During the attack the adversary may probe at most T inputs $x = (\langle i \rangle, x')$ by forwarding them to the hash function oracle. First notice that none of these samples yields a collision $\mathsf{hash}(x) = x_{i+1}$ for any unreleased part of the chain, except with probability ϵ_0. Else this would contradict Assumption 6, because we could easily devise an algorithm from the adversary, winning experiment $\exp_{\mathcal{A}}(T_0, \epsilon_0, 1, 1)$ with probability more than ϵ_0. In the sequel we condition on the event that no such pre-image of the chain pops up.

Fix some sample $x = \langle i \rangle || x'$ and perform the thought experiment of iterating $\mathcal{S}_{\mathrm{rel},b,B}$ for input $i, x, \mathsf{cb}_{>i}$ until $b \lfloor pB/2 \rfloor$ check bits have been produced (or the end is reached). Here we exploit the fact that the sample implicitly defines the position i. We can assume that no chain value x_j for $i < j$ such that $x = \mathsf{hash}^{j-i}(x_j)$ has been released before; else the adversary could derive the hash value of x for free by simply querying $\mathsf{Release}$ instead. In this case, it follows

again by Assumption 6 that none of the values during the "virtual" iteration matches the values in the chain. According to Assumption 7, the probability that the "virtual" check bits match the $b\lfloor pB/2 \rfloor$ corresponding bits in cb is then at most $2^{-b\lfloor pB/2 \rfloor}$. Hence, the probability that any of the at most T samples of the adversary matches is at most

$$T \cdot 2^{-b\lfloor pB/2 \rfloor} \leq T \cdot 2^{-\log_2 T - \log_2 \frac{1}{\epsilon - \epsilon_0}} = \epsilon - \epsilon_0.$$

Together with the bound ϵ_0 of the probability that the adversary finds a preimage of the given chain, the result now follows. □

Returning to our example with $n = 1,024$, $B = 128$ and $b = 2$, for $T = 2^{40}$ and $\epsilon = 2^{-20}$ (and $\epsilon_0 \approx 0$) the verifier requires about $p = 48.5\%$ of the original workload. For $b = 3$ and vectors of $1,152$ bits the work in this case even reduces to 33%.

We finally contrast the length of check-bit vectors in our solution to the lower bound. Suppose we want to achieve a reduction of 50% and fix $p = 1/2$. We further fix the adversarial parameters T, ϵ, or at least some upper bound for them. Assume for simplicity that $\epsilon_0 = 0$. Then the lower bound says check-bit vectors must be roughly $(\log_2 T + \log_2 \frac{1}{\epsilon} - \log_2 n) \log_2 \frac{n}{\Delta}$ bits. If we now set $bB = 4(\log_2 T + \log_2 \frac{1}{\epsilon})$ in our construction, then we achieve $p \approx 1/2$ and the vector length becomes $2(\log_2 T + \log_2 \frac{1}{\epsilon})(\log_2 \frac{n}{\Delta} + 2)$ for $\Delta = B/2$. In this sense, our solution is close to the lower bound.

5 Discussion

We have presented constructions to improve the verification time of hash chains. Our solutions enable the verifier to select a flexible security level and to relate the work to be done to this security level. Our constructions and lower bounds rely on so-called check-bit schemes where basically some bits of the intermediate values are output. Fortunately, such schemes are very simple and can be integrated quite easily; they preserve the simplicity of hash chains and are applicable in general. Disadvantageously, as we have shown, those schemes cannot go below certain bounds when it comes to the length of the check-bit vectors.

It remains an open problem to provide other check-bit schemes with shorter vectors, e.g., by using lossy encoding techniques. Yet, those schemes should have comparable simplicity as the basic scheme in this paper, otherwise the running time may be dominated by the additional effort, invalidating the benefits of faster verification. Similarly, it would be interesting to show lower bounds for more general check-bit schemes.

Acknowledgment

We thank the anonymous reviewers of RSA-CT 2004 for valuable comments.

References

[BK03] M. Bellare and T. Kohno. *Hash Function Balance and its Impact on Birthday Attacks*. Number 2003/65 in Cryptology eprint archive. `eprint.iacr.org`, 2003.

[BR93] M. Bellare and P. Rogaway. *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*. Proceedings of the Annual Conference on Computer and Communications Security (CCS). ACM Press, 1993.

[CJ02] D. Coppersmith and M. Jakobsson. *Almost Optimal Hash Sequence Traversal*. Financial Cryptography (FC) 2002, Volume 2357 of Lecture Notes in Computer Science. Springer-Verlag, 2002.

[DGN03] C. Dwork, A. Goldberg, and M. Naor. *On Memory-Bound Funtions for Fighting Spam*. Advances in Cryptology — Crypto 2003, Volume 2729 of Lecture Notes in Computer Science. Springer-Verlag, 2003.

[Fis03] M. Fischlin. *Progressive Verification: The Case of Message Authentication*. Progress in Cryptology — Indocrypt 2003, Volume 2904 of Lecture Notes in Computer Science. Springer-Verlag, 2003.

[Hal94] N. Haller. *The S/KEY One-Time Password Scheme*. Symposium on Network and Distributed Systems Security, pages 151–157. Internet Society, 1994.

[Hal95] N. Haller. *The S/KEY One-Time Password Scheme*, 1995.

[HJP02] Y.-C. Hu, D. Johnson, and A. Perrig. *SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks*. Workshop on Mobile Computing Systems and Applications (WMCSA) 2002. IEEE Computer Society Press, 2002.

[HPJ03] Y.-C. Hu, A. Perrig, and D. Johnson. *Efficient Security Mechanisms for Routing Protocols*. Annual Symposium on Network and Distributed System Security (NDSS) 2003. Internet Society, 2003.

[HPT97] R. Hauser, A. Przygienda, and G. Tsudik. *Reducing the Cost of Security in Link State Routing*. Annual Symposium on Network and Distributed System Security (NDSS)'97. Internet Society, 1997.

[HSW96] R. Hauser, M. Steiner, and M. Waidner. *Micro-Payments Based on iKP*. Proceedings of SECURICOM'96, Worldwide Congress on Computer and Communications Security and Protection, pages 67–82. ???, 1996.

[JLMS03] M. Jakobsson, T. Leighton, S. Micali, and M. Szydlo. *Fractal Merkle Tree Representation and Traversal*. Topics in Cryptology — Cryptographer's Track, RSA Conference (CT-RSA) 2003, Volume 2612 of Lecture Notes in Computer Science, pages 314–326. Springer-Verlag, 2003.

[Lam81] L. Lamport. *Password Authentication with Insecure Communication. Communications of the ACM*, 24(11):770–772, 1981.

[Mer88] R. Merkle. *A Digital Signature Based on a Conventional Encryption Function*. Advances in Cryptology — Crypto'87, Volume 293 of Lecture Notes in Computer Science, pages 369–378. Springer-Verlag, 1988.

[Mic96] S. Micali. *Efficient Certificate Revocation*. Technical Report MIT/LCS/TM-542b, MIT Laboratory for Computer Science, 1996.

[PCST02] A. Perrig, R. Canetti, D. Song, and D. Tygar. *The TESLA Broadcast Authentication Protocol*. CryptoBytes, Volume 5, pages 2–13. RSA Security, 2002.

[Per01] A. Perrig. *The BiBa One-Time Signature and Broadcast Authentication Protocol*. Proceedings of the Annual Conference on Computer and Communications Security (CCS), pages 28–37. ACM Press, 2001.

[RS97] R. Rivest and A. Shamir. *PayWord and MicroMint: Two Simple Micropayment Schemes*. Security Protocols, Volume 1189 of Lecture Notes in Computer Science, pages 69–87. Springer-Verlag, 1997.

[Sel03] Y. Sella. *On the Computation-Storage Trade-Offs of Hash Chain Traversals*. Financial Cryptography (FC) 2003, Lecture Notes in Computer Science. Springer-Verlag, 2003.