Second Preimage Attacks on Dithered Hash Functions

Elena Andreeva[1], Charles Bouillaguet[2], Pierre-Alain Fouque[2], Jonathan J. Hoch[3], John Kelsey[4], Adi Shamir[2,3], and Sebastien Zimmer[2]

[1] SCD-COSIC, Dept. of Electrical Engineering, Katholieke Universiteit Leuven, Elena.Andreeva@esat.kuleuven.be
[2] École normale supérieure (Département d'Informatique), CNRS, INRIA, {Charles.Bouillaguet,Pierre-Alain.Fouque,Sebastien.Zimmer}@ens.fr
[3] Weizmann Institute of Science, {Adi.Shamir,Yaakov.Hoch}@weizmann.ac.il
[4] National Institute of Standards and Technology, john.kelsey@nist.gov

Abstract. We develop a new generic long-message second preimage attack, based on combining the techniques in the second preimage attacks of Dean [8] and Kelsey and Schneier [16] with the herding attack of Kelsey and Kohno [15]. We show that these generic attacks apply to hash functions using the Merkle-Damgård construction with only slightly more work than the previously known attack, but allow enormously more control of the contents of the second preimage found. Additionally, we show that our new attack applies to several hash function constructions which are not vulnerable to the previously known attack, including the dithered hash proposal of Rivest [25], Shoup's UOWHF[26] and the ROX hash construction [2]. We analyze the properties of the dithering sequence used in [25], and develop a time-memory tradeoff which allows us to apply our second preimage attack to a wide range of dithering sequences, including sequences which are much stronger than those in Rivest's proposals. Finally, we show that both the existing second preimage attacks [8, 16] and our new attack can be applied even more efficiently to multiple target messages; in general, given a set of many target messages with a total of 2^R message blocks, these second preimage attacks can find a second preimage for one of those target messages with no more work than would be necessary to find a second preimage for a single target message of 2^R message blocks.

Keywords: Cryptanalysis, Hash Function, Dithering

1 Introduction

A number of recent attacks on hash functions have highlighted weaknesses of both specific hash functions, and the general Merkle-Damgård construction. Wang *et al.* [28–31], Biham *et al.* [3], Klima [19] and Joux *et al.* [14] all show that differential attacks can be used to efficiently find collisions in specific hash functions based on the MD4 design, such as MD5, RIPEMD, SHA-0 and SHA-1. This

type of result is important for at least two reasons. First, collision resistance is a required property for a hash function, and many applications of hash functions fail when collisions can be found. Second, efficiently found collisions permit additional attacks on hash functions using the Merkle-Damgård construction, as in Joux's [13] multicollision attack on cascade hashes, and the long-message second preimage attacks of Dean [8] and Kelsey and Schneier [16].

After Kelsey and Schneier published their attack, several researchers proposed a variant of the Merkle-Damgård construction, in which a third input to the compression function, called a "dithering sequence" in [25] and this paper, is used to block the attack. Specifically, using a dithering sequence prevents the construction of "expandable messages," required for both Dean and Kelsey and Schneier's attacks. In this paper, we develop a new kind of second preimage attack, which applies to some dithered variants of the Merkle-Damgård construction.

1.1 Related Work

The PhD thesis of Dean [8] presented a second preimage attack that works against a subset of hash functions using the Merkle-Damgård construction. Kelsey and Schneier [16] extended this result to work for *all* Merkle-Damgård hashes. For an n-bit hash function, their result allows an attacker to find a second preimage of a 2 block[1] target message with $k \cdot 2^{n-1} + 2^{n-k+1}$ evaluations of the compression function. The attack relies on the ability to construct an *expandable message*, a set of incomplete messages of widely varying length, all of which yield the same intermediate hash result. This attack can be seen as a variant of the *long message attack* [20], in which the expandable message is used to carry out the attack despite the Merkle-Damgård strengthening.

Variants of the Merkle-Damgård construction that attempt to preclude the aforementioned second preimage attacks are the HAIFA [2] [23] construction proposed by Biham and Dunkelman and the "dithered" Merkle-Damgård hash by Rivest [25]. HAIFA includes the number of message bits hashed so far in the message block. The simplest way to implement HAIFA is to shorten each data block by 64 bits, filling those 64 bits with the 64 bit counter used internally to track the length of the hash input so far. Rivest, on the other hand, introduced a clever way to decrease the number of bits used for this extra input to either 2 or 16, thus increasing the bandwidth available for actual data, by using a specific sequence of values to "dither" the actual inputs. The properties of this sequence were claimed by Rivest to be sufficient to avoid the second preimage attack on the hash function.

The herding attack of Kelsey and Kohno [15] can be seen as another variant of the long-message attack. In their attack, the attacker first does a large precomputation, and then commits to a hash value h. Later, upon being challenged with

[1] In this paper, we describe message lengths in terms of message blocks, rather than bits. Most common hash functions use blocks of length 512 or 1024 bits.

[2] We do not have any attacks more efficient than exhaustive search on HAIFA.

a prefix P, the attacker constructs a suffix S such that $\text{hash}(P||S) = h$. Their paper introduced the "diamond structure", which is reminiscent of a complete binary tree. It is a 2 -multicollision in which each message in the multicollision has a different initial chaining value, and which is constructed in the precomputation step of the attack. The herding attack on an n-bit hash function requires approximately 2^{2} $^{3\ 1}$ work.

1.2 Our Results

In this paper, we develop a new generic second preimage attack on Merkle-Damgård hash functions and dithered Merkle-Damgård variants, treating the compression functions as black boxes. Our basic technique relies on the diamond from the herding attack of [15]. If the diamond is a 2 -multicollision, we obtain a second preimage of a message of length 2 blocks with 2 ^{2} $^{2\ 2}+2^{\ -}+2^{\ -}$ compression function computations. The attack is optimized when $\ell \approx n/3$, yielding an attack of complexity $5 \cdot 2^{2}$ $^{3}+2^{\ -}$.

Our attack is slightly more expensive than the $k \cdot 2$ $^{2\ 1}+2^{\ -}$ complexity from [16] (for SHA-1, in which $n = 160$ and $k = 55$, the Kelsey-Schneier attack complexity is about 2^{1} work whereas ours is approximately 2^{1}). However, the new attack can be applied to Merkle-Damgård variants for which the attack of [16] is impossible. Our result also permits the attacker to leave most of the target message intact in the second preimage, or to arbitrarily choose the contents of roughly the first half of the second preimage, while leaving the remainder identical to the target message.

We can also apply our new second preimage attack to the dithered Merkle-Damgård hash variant of [25], exploiting the fact that the dithering sequences have many repetitions of some subsequences. For Rivest's proposed 16-bit dithering sequence, the attack requires 2 ^{2} $^{2\ 2}+(8\ell+32768)\cdot 2^{\ -}+2^{\ -}$ work, which for SHA-1 is approximately 2^{12} . This is slightly worse than the attacks against the basic Merkle-Damgård construction but it is still much smaller than the 2^{1} security which was expected for the dithered construction. We show that the security of a dithered Merkle-Damgård hash is dependent on the number of distinct ℓ-letter subwords in the dithering sequence, and that the sequence chosen by Rivest is very susceptible to our attack.

We also show that the attack on dithered hashes is subject to a time-memory tradeoff that enables the construction of second preimages for any dithering input defined over a small alphabet with only a small amount of online computation after an expensive precomputation stage.

We further apply our attack to a one way hash function designed by Shoup [26], which has some similarities with dithered hashing. The attack applies as well to constructions that derive from this design, such as ROX [2]. Our technique yields the first published attack against these particular hash functions. This additionally proves that Shoup's security bound is tight, since there is asymptotically only a factor of $\mathcal{O}(k)$ between his bound and our attack's complexity.

Finally, we show that both the original second-preimage attack of [8, 16] and our attack can be extended to the case in which there are multiple target

messages. In general, finding a second preimage for any one of 2 target messages of length 2 blocks each requires approximately the same work as finding a single second preimage for a message of 2 blocks.

1.3 Organization of the Paper

We describe our attack against the Merkle-Damgård construction in section 2. We introduce some terminology and describe the dithered Merkle-Damgård construction in section 3, and then we extend our attack to tackle dithered Merkle-Damgård in section 4. We apply it to Rivest's concrete proposal, as well as to some of the variations that he suggested. In section 5, we show that our attack works also against Shoup's UOWHF construction. We conclude with section 6, where we show how the second preimage attack may be applied to finding a second preimage for one of a large set of target messages.

2 A New Generic Second Preimage Attack

2.1 The Merkle-Damgård construction

We first describe briefly the classical Merkle-Damgård construction. An iterated hash function 0 1 0 1 is built by iterating a basic compression function 0 1 0 1 0 1 . The hash process works as follows:

- Pad and split a message into blocks 1 of bits each.
- Set h to the initialization value .
- For each message block compute $h =$ (h −1).
- Output ( ) = h .

The padding is usually done by appending a single '1' bit followed by as many '0' bits as needed to complete an -bit block. Merkle [21] and Damgård [7] independently proved in 1989 that making the binary encoding of the message length part of the padding improves the security of the construction: with this so-called *strengthening*, the scheme is proven to be Collision-Resistance Preserving, in the sense that a collision in the hash function would imply a collision in the compression function . As a side effect, the strengthening defines a limit over the maximal size of the messages that can be processed. In most deployed hash functions, this limit is 2 4 bits, or equivalently 2 512-bit blocks. In the sequel, we denote the maximal number of admissible blocks by 2 .

2.2 Second Preimage Attack on Merkle-Damgård hash

We now describe a new technique to find second preimages on a Merkle-Damgård hash. It relies heavily on the "diamond structure" introduced by Kelsey and Kohno [15].

A diamond of size ℓ is a multicollision that has the shape of a complete converging binary tree of depth ℓ, with 2 leaves (hence we often refer to it

as a *collision tree*). Its nodes are labelled by chaining values over n bits, and its edges are labelled by message blocks over bits, which map between the chaining values at the two ends of the edge by the compression function. Thus, from any one of the 2 leaves, there is a path labelled by ℓ message blocks that leads to the same target value h labelling the root of the tree.

Let be a target message of length 2 blocks. The main idea of our attack is that connecting a message to a collision tree can be done in less than 2 work. Moreover, connecting the root of the tree to one of the 2 chaining values encountered during the computation of ( ) takes only 2 $^{-}$ compression function calls. The attack works in 4 steps as described in figure 1.

1. Preprocessing step: compute a collision tree of depth ℓ with an arbitrary target value h . Note that this has to be done only once, and can be reused when computing second preimages of multiple messages.
2. Connect the target h to some chaining value in the message M. This can be done by generating random message blocks B, until $F \quad h, B \quad h_0$ for some i , $\ell \quad \leq i < |M|$. Let B be a message block satisfying this condition.
3. Generate an arbitrary prefix P of size $i \quad -\ell -$ blocks whose hash is one of the chaining values labelling a leaf. Let $h \quad H \quad P$ be this value, and let T be the chain of ℓ blocks traversing the tree from h to h .
4. Form a message $M' \quad P||T||B \; ||x_{0\ 1} \ldots x_{2^k}$.

Fig. 1: Summary of the attack on classic Merkle-Damgård.

Messages and are of equal length and hash to the same value, before strengthening, so they produce the same hash value despite the Merkle-Damgård strengthening.

A collision tree of depth ℓ can be constructed with time and space complexity $2^{\frac{n}{2}\ \frac{\ell}{2}\ 2}$ (see [15] for details). The second step of the attack can be carried out with 2 $^{-}$ work, and the third one with 2 $^{-}$ work. The total time complexity of the attack is then: $2^{\frac{n}{2}\ \frac{\ell}{2}\ 2} + 2^{-} + 2^{-}$. This quantity becomes minimal when $\ell = (n \quad 2)/3$, and in this setting, the total cost of our attack is about $5 \cdot 2^{2\ \ 3} + 2^{-}$.

2.3 Comparison With Kelsey and Schneier

On the original Merkle-Damgård construction, the attack of [16] is more efficient than ours (on SHA-1, they can find a second preimage of a message of size 2 with 2^1 work, whereas we need 2^1 calls to the compression function to obtain the same result).

However, our technique gives the adversary more control on the second preimage, since she can typically choose about the first half of the message in an arbitrary way. For example, she could choose to replicate most of the target message, leading to a second preimage that differs from the original by only $k+2$ blocks.

The main apparent difference between the two techniques is that the attack of Kelsey and Schneier relies on *expandable messages*. An expandable message is a family of messages with different number of blocks but with the same hash when the final length block is not included in the computation. Their attack constructs such an expandable message in time $k \cdot 2^{2} {}^{1}$. Our attack can also be viewed as a new, more flexible technique to build expandable messages, by choosing a prefix of the appropriate length and connecting it to the collision tree. This can be done in time $2^{2} {}^{2}+2^{-}$. Altough it is more expensive, this new technique can be adapted to work even when an additional dithering input is given, as we will demonstrate in the sequel.

3 Dithered Hashing

The general idea of dithered hashing is to perturb the hashing process by using an additional input to the compression function, formed by the consecutive elements of a fixed *dithering* sequence. This gives the attacker less control over the input of the compression function, and makes the hash of a message block dependent on its position in the whole message. In particular, the goal of dithering is to prevent attacks based on expandable messages.

Since the dithering sequence $\mathbf{z}$ has to be at least as long as the maximal number of blocks in any message that can be processed by the hash function, it is reasonable to consider infinite sequences as candidates for $\mathbf{z}$. Let be a finite alphabet, and let the dithering sequence $\mathbf{z}$ be an eventually infinite word over . Let $\mathbf{z}$ denote the -th element of $\mathbf{z}$. The dithered Merkle-Damgård construction is obtained by setting $h = (h_{-1}\ \mathbf{z})$ in the definition of the Merkle-Damgård scheme.

3.1 Words and Sequences

Notations and Terminology. Let be a word over the finite alphabet . The dot operator denotes concatenation. If can be written as = (where , or can be empty), we say that is a *prefix* of and that is a *factor* (or subword) of . A finite word is a *square* if it can be written as = , where is not empty. A finite word is an *abelian square* if it can be written as = where is a permutation of (*i.e.*, a reordering of the letters of). A word is said to be *square-free* (resp. *abelian square-free*) if none of its factors is a square (resp. an abelian square). Note that abelian square-free words are also square-free.

An Infinite Abelian Square-Free Sequence. In 1992, Keränen [17] exhibited an infinite abelian square-free word $\mathbf{k}$ over a four-letter alphabet (there are no infinite abelian square-free words over a ternary alphabet). In this paper, we call this infinite abelian square-free word the *Keränen sequence*. Details about this construction can be found in [17, 18, 25].

Sequence Complexity. The number of factors of a given size of an infinite word gives an intuitive notion of its *complexity*: a sequence is more complex (or richer) if it possesses a large number of different factors. We denote by $_{\mathbf{z}}(\ell)$ the number of factors of size ℓ of the sequence $\mathbf{z}$.

3.2 Rivest's Proposals.

Keränen-DMD. Rivest suggested to directly use the Keränen sequence as a source of dithering inputs. The dithering inputs are taken from the alphabet $=$, and can be encoded by two bits. The number of data bits in the input of the compression function is thus reduced by only two bits, which improves the hashing efficiency (compared to longer encodings of dither inputs). It is possible to generate the Keränen sequence online, one symbol at a time, in logarithmic space and constant amortized time.

Rivest's Concrete Proposal. Rivest's concrete proposal is referred to as DMD-CP (Dithered Merkle-Damgård – Concrete Proposal). To speed up the generation of the dithering sequence, Rivest proposed a slightly modified scheme, in which the dithering symbols are 16-bit wide. If the message is blocks long, then for 1 the -th dithering symbol has the form:

$$\left(0\ \mathbf{k}\left[\lfloor\ /2^{13}\rfloor\right]\qquad 2^{\ 13}\right)\quad 0\ 1\qquad 0\ 1\quad {}^{13}$$

The idea is to increment the counter for each dithering symbol, and to shift to the next letter in the Keränen sequence, only when the counter overflows. This "diluted" dithering sequence can essentially be generated 2^{13} times faster than the Keränen sequence. The last dithering symbol has a different form (recall that is the number of bits in a message block):

$$(1\ |\ |\qquad)\quad 0\ 1\quad 0\ 1\quad {}^{1}$$

4 Second Preimage Attacks on Dithered Merkle-Damgård

In this section, we present the first known second preimage attack on Rivest's dithered Merkle-Damgård construction. In section 4.1, we adapt the attack of section 2 to Keränen-DMD, obtaining second preimages in time $(k + \ 0\ 5) \cdot 2^{\ -\ 3}$. We then apply the extended attack to DMD-CP, obtaining second preimages with about $2^{\ -\ 1}$ evaluations of the compression function. We show some examples of sequences which make the corresponding dithered constructions immune to our attack. This notably covers the case of Haifa [23]. Lastly, in section 4.2 we present a variation of the attack, which includes an expensive preprocessing, but which is able to cope with sequences of high complexity over a small alphabet with a very small online cost.

4.1 Adapting the Attack to Dithered Merkle-Damgård

Let us now assume that the hashing algorithm uses a dithering sequence $\mathbf{z}$. When building the collision tree, we must choose which dithering symbols to use. A simple solution is to use the same dithering symbol for all the edges at the same depth in the tree. A tuple of ℓ letters is then required to build the collision tree. We will also need an additional letter to connect the tree to the message . This way, in order to build a collision tree of depth ℓ, we have to fix a word of size $\ell+1$, use as the dithering symbol of depth , and use the last letter of to realize the connection.

The dithering sequence makes the hash of a block dependent on its position in the whole message. Therefore, the collision tree can be connected to its target only at certain positions, namely, at the positions where and $\mathbf{z}$ match. The set of positions in the message where this is possible is then given by:

$$n = \mathbb{N} \left(\ell+1 \right) \left(\mathbf{z} \quad \ell \quad \mathbf{z} = \right)$$

Note that finding a connecting block in the second step defines the length of the prefix that is required. If n , it will be possible to build the second preimage. Otherwise, another block has to be found.

To make sure that n is not empty, has to be a factor of $\mathbf{z}$. Ideally, should be the factor of length $\ell+1$ which occurs most frequently in $\mathbf{z}$, as the cost of the attack ultimately depends on the number of connecting blocks tried before finding a useful one (with n). What is the probability that a factor appears at a random position in $\mathbf{z}$? Although this is highly sequence-dependent, it is possible to give a generic lower bound: in the worst case, all factors of size $\ell+1$ appear in $\mathbf{z}$ with the same frequency. In this setting, the probability that a randomly chosen factor of size $\ell+1$ in $\mathbf{z}$ is the word is $1/ \ _{\mathbf{z}}(\ell+1)$.

The main property of $\mathbf{z}$ influencing the cost of our attack is its complexity (which is related to its min-entropy), whereas its repetition-freeness influences the cost of Kelsey and Schneier type attacks.

1. Choose the most frequent factor of $\mathbf{z}$, with $| \ | \ \ell$.
2. Build a collision tree of depth ℓ using as the dithering symbols in all the leaf-to-root paths. Let h be the target value of the tree.
3. Find a connecting block B mapping h to anyone of the h (say h_0), by using ℓ as the dithering letter. Repeat until i .
4. Carry the remaining steps of the attack as described in Fig. 1.

Fig. 2: Summary of the attack when a dithering sequence $\mathbf{z}$ is used.

The cost of finding this second preimage for a given sequence $\mathbf{z}$, in the worst-case situation where all factors appear with the same frequency, is given by:

$$2^{\frac{n}{2}} \ ^{\frac{\ell}{2}} \ ^{2} + \ _{\mathbf{z}}(\ell+1) \cdot 2^{-} + 2^{-}$$

Cryptanalysis of Keränen-DMD. The cost of the extended attack against Keränen-DMD depends on the complexity of the sequence **k**. Since it has a very regular structure, **k** has an unusually low complexity.

Lemma 1. *For* ℓ 85*, we have:*

$$(\ell) \quad 8 \cdot \ell + 332$$

Despite being strongly repetition-free, the sequence **k** offers an extremely weak security level against our attack. We illustrate this by evaluating the cost of our attack on Keranen-DMD:

$$2^{\frac{n}{2}} \; {}^{\frac{\ell}{2}} \; {}^{2} + (8 \cdot \ell + 3 \;\; 0) \cdot 2^{\;-} \;\; + 2^{\;-}$$

If n is of the same order than about $3k$, then the first term of this sum is of the same order than the other two, and if n $3k$ then it can simply be neglected. We will use this approximation several times in the sequel. By setting $\ell = k$ 3, the total cost of the attack is about: $(k + \;\; 0 \;\; 5) \cdot 2^{\;-} \;\; {}^{3}$ which is much smaller than 2 in spite of the dithering.

Cryptanalysis of DMD-CP. We now apply the attack to Rivest's concrete proposal. We first need to evaluate the complexity of its dithering sequence. Recall from section 3.2 that it is based on the Keränen sequence, but that we move on to the next symbol of the sequence only when a 13 bit counter overflows. The original motivation was to reduce the cost of the dithering, but it has the unintentional effect of increasing the resulting sequence complexity. However, it is possible to prove that this effect is quite small:

Lemma 2. *Let* *denote the sequence obtained by diluting* **k** *with a 13-bit counter. Then for every* 0 ℓ $2^{\;13}$*, we have:*

$$(\ell) = 8 \cdot \ell + 32760$$

The dilution does not generate a sequence of a higher asymptotic complexity: it is still linear in ℓ, even though the constant term is bigger due to the counter. The cost of the attack is therefore:

$$2^{\frac{n}{2}} \; {}^{\frac{\ell}{2}} \; {}^{2} + (8 \cdot \ell + 32768) \cdot 2^{\;-} \;\; + 2^{\;-}$$

Again, if n is greater than about $3k$, the best value of ℓ is k 3, and the complexity of the attack is then approximately: $(k + \;\; 0 \;\;) \cdot 2^{\;-} \;\; {}^{3} \;\;\; 2^{\;-} \;\; {}^{1}$ For settings corresponding to SHA-1, a second preimage can be computed in time 2^{12} .

Countermeasures. Even though the dilution does not increase the asymptotic complexity of a sequence, the presence of a counter increases the complexity of the attack. If we simply used a counter over bits as the dithering sequence,

the number of factors of size ℓ would be $(\ell) = 2$ (as long as ℓ). The complexity of the attack would then become: $2^{\frac{n}{2}} {}^{\frac{\ell}{2}\ 2} + 2^{-} + 2^{-}$

In practice, the dominating term is 2^{-}. By taking $= k$, we would obtain a scheme which is resistant to our attack. This is essentially the choice made by the designers of HAIFA [23], but such a dithering sequence consumes k bits of bandwidth. Note that as long as the counter does not overflow, no variation of the attack of Kelsey and Schneier can be applied to the dithered construction.

Using a counter (*i.e.*, a big alphabet) is a simple way to obtain a dithering sequence of high complexity. An other, somewhat orthogonal, possibility to improve the resistance of Rivest's dithered hashing to our attack is to use a dithering sequence of high complexity over a *small* alphabet (to preserve bandwidth). In appendix A we show that there is an abelian square-free sequence over 6 letters with complexity greater than $2^{\ 2}$. Then, with $\ell = 2k/3$, the total cost of the online attack is about $2^{-2\ \ 3}$.

Another possible way to improve the resistance of Rivets's construction against our attack is to use a pseudo random sequence over a small alphabet. Even though it may not be repetition-free, its complexity is almost maximal. Suppose the alphabet has size $= 2$. Then the expected number of ℓ-letter factors in a pseudo random word of size 2 is lower-bounded by: $2 \cdot \left(1 \quad 2^{-}\right)$ (refer to [12], theorem 2, for a proof of this claim)). The total optimal cost of the online attack is then at least $2^{-} \quad {}^{1\ \ 2}$ and is obtained with $\ell = k/(\ + 1)$. With 8-bit dithering symbols and if $k = 55$, as in the SHA family, the complexity of the attack is 2^{-} .

4.2 A Generic Attack on any Dithering Scheme With a Small Alphabet

The attacks described so far exploited the low complexity of Rivest's specific dithering sequences. In this section we show that the weakness is more general, and that after an $\mathcal{O}(2\ )$ preprocessing, second preimages can be found for messages of length $2 \quad 2^{\ 4}$ in $\mathcal{O}\left(2^{2\ \ -\ \ 3}\right)$ time and space for any dithering sequence (even of maximal complexity) if the dithering alphabet is small. Second preimages for longer messages can be found in $\left(\mathcal{O}\left(2\ \right)\ \ \mathcal{O}\left(2\ \ ^{2}\right)\right)$ time and $\left(\mathcal{O}\left(2^{-}\right)\ \ \mathcal{O}\left(2\ \ ^{2}\right)\right)$ memory.

Outline of the Attack. The new attack can be viewed as a type of time-memory tradeoff. For any given compression function, we precompute a fixed data structure which can then be used to find additional preimages for *any* dithering sequence and *any* given message of sufficient length. In the attack we will need to find connecting blocks leading from the message to our data structure and from our data structure to the message. The data structure will allow us to generate a sequence of blocks of the required length, leading from the entry point to the exit point, using the given dithering sequence.

A simple structure of this type is the *kite generator*[3] which will allow us to find a second preimage for a message made of $\mathcal{O}(2\ )$ message blocks in time $(\mathcal{O}(2\ )\ \mathcal{O}(2^{\ -\ 2}))$ and $\mathcal{O}(\ \cdot 2^{\ -}\ )$ space. Note that for the SHA-1 parameters of $n = 160$ and $k = 55$, the time complexity of the new attack is $2\ $, which is just the time needed to hash the original message. However, the size of the kite generator for the above parameters exceeds 2^{11} . The kite generator is a labelled directed graph whose $2^{\ -}$ vertices are labelled by some easily recognized subset of the chaining values that includes the (e.g., the tiny fraction of hash values which are extremely close to). Each directed edge (which can be traversed in both directions) is labelled by one letter from the dithering alphabet and one message block , and it leads from vertex h_1 to vertex h_2 if $(h_1\ \ \ ) = h_2$. Each vertex in the generator should have exactly two outgoing edges labelled by each dithering letter, and thus the *expected* number of ingoing edges labelled by each letter is also 2. The generator is highly connected in the sense that there is an exponentially large diverging binary tree with any desired dithering sequence starting at any vertex, and an exponentially large converging tree [4] with any desired dithering sequence (whose degrees are not always 2) ending at most vertices. It can be viewed as a generalization of the collision tree of Kelsey and Kohno [15], which is a single tree with a single root in only the converging direction and with no dithering labels.

Once computed (during an unbounded precomputation stage), we can use the generator to find a second preimage for any given message with 2 blocks and any dithering sequence. We first hash the long input to find (with high probability) some intermediate hash value h which appears in the generator. We then use the generator to replace the first blocks in the message by a different set of blocks. We start from the generator vertex labelled by , and follow some path in the generator of length $(n\ \ k)$ which has the desired dithering sequence (there are exponentially many paths we can choose from). It leads to some hash value h in the generator. We then evaluate the full diverging tree of depth $(n\ \ k)/2$ and the desired dithering sequence starting at h , and the full converging tree of depth $(n\ \ k)/2$ and the desired dithering sequence ending at h . Since the number of leaves in each tree is $\mathcal{O}(2^{\ -\ 2})$ and they

[3] We call it a kite generator since we use it to generate *kites* of the form

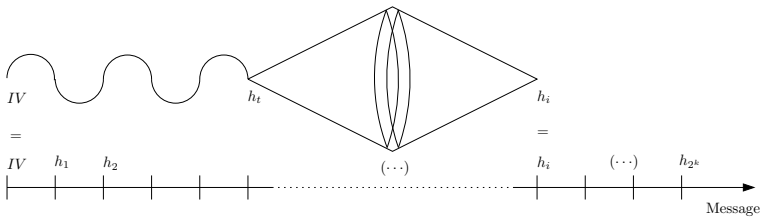

Fig. 3: A Kite

[4] See [10] for a formal justification of this claim.

are labelled by only 2^{-} possible values, we expect by the birthday paradox to find a common chaining value among the two sets of leaves. We can now combine the long random chain of length $(n \quad k)$ with the two short tree chains of length $(n \quad k)/2$ to find a *kite*-shaped structure of the same length and with the same dithering sequence as the original message between the two chaining values and h . Note that the common leaf of the two trees can be found with no additional space by using a variant of Pollard's rho method which traverses pseudo-randomly chosen paths in the two trees until it cycles.

This attack can be applied with essentially the same complexity even when the is not known during the precomputation stage (e.g., when it is time dependent). When we hash the original long message, we have to find two intermediate hash values h and h (instead of and h) which are contained in the generator and connect them by a properly dithered kite-shaped structure of the same length.

The main problem of this technique is that for the typical case in which $k \quad n/2$, it uses more space than time, and if we try to equalize them by reducing the size of the kite generator, we are unlikely to find any common chaining values between the given message and the generator. Finding a way to connect the generator back into the message will require $2^{-\ 1}$ additional steps, which will make the time complexity too high. To bypass this difficulty, we will use the classic time-memory tradeoff of Hellman tables.

Hellman's TMTO attack. Time/memory Tradeoffs (TMTO) were first introduced in 1980 by Hellman [11]. The idea is to improve brute force attacks by trading time for memory when inverting a function $0\ 1 \quad 0\ 1$. Suppose we have an image element and wish to find a pre-image $^{-1}(\ )$. One extreme would be to go over all possible elements until we find one such that $(\ ) =$, while the other extreme would be to pre-compute a huge table containing pairs $(\quad (\ ))$ sorted by the second element. Hellman's idea was to consider what happens when applying iteratively. We start at a random element and compute $_1 = (\ )$ for steps saving only the start and end points of the generated chain $(\quad)$. We repeat this process with different initial points and generate a total of chains. Now on input we start generating a chain starting from and check if we reach one of the saved endpoints. If we have, we generate the corresponding chain, starting from the original starting point and hope to find a preimage of . Notice that as the number of chains increases beyond $2\ /\ ^2$, the contribution from additional chains decreases with the number of chains. To counter this birthday paradox effect, Hellman suggested to construct a number of tables, each using a slightly different function , such that knowing a preimage of under implies knowing such a preimage under . Hellman's original suggestion, which works well in practice, was to use $(\ ) = (\quad)$. Thus if we create $= 2^{\ 3}$ tables each with a different , such that each table contains $= 2^{\ 3}$ chains of length $= 2^{\ 3}$, about 88% of the 2 points will be covered by at least one table. Notice that the running

time of Hellman's algorithm is $\cdot = 2^{2\ 3}$ while the memory requirement is $\cdot = 2^{2\ 3}$.

The Attack. As mentioned above, we need to find a linking block from the kite-generator to the message when its size is too small to have a common point. To solve this problem, we denote one of the vertices in the kite-generator by and construct for each a set of Hellman tables with chains, each of length , such that $\cdot \cdot = 2^{-}$ by iterating the basic function $( ) = ($). During the online phase, for each intermediate hash value h in the message, we use the set of tables corresponding to the dithering character used to reach h and try to find a block leading from the specified vertex to h using . Since the tables cover approximately 2^{-} elements, the probability of finding such a block for h is 2^{-}. As the message is of length 2, we expect to find on average one connecting h. Notice that although we create chains for the Hellman tables, they do *not* correspond to the chain of hash values of a message, and thus we do not have to use the correct dithering sequence along these paths. The only purpose of the chains is to invert the function and thereby find a single block linking to one of the intermediate hash values along the given message.

Now that we have a method for connecting a predetermined hash value to a message, we can replace the role of the kite-generator of finding a prefix which ends at with a simpler construction. Since we were not constrained in our choice of we can simplify the kite generator to the single point with a self loop for each dithering symbol . During the preprocessing, we exhaustively search for each a block such that ( ) = . Given such self loops, we use in each step the block corresponding to the current dithering symbol and thus we can generate a message of any length starting and ending with . This serves as the point in Hellman's algorithm. Note that this construction does not have the advantage of the original kite generator that can be unknown during the preprocessing stage.

Combining the two steps, we first find a linking block from to one of the intermediate hash values of the message using the correct dithering symbol. Then, using the self loops, we construct a prefix of the required length linking back to . During the preprocessing, the cost of constructing the Hellman tables is $\cdot \cdot \cdot = \mathcal{O}(\cdot 2^{-})$ time and $\cdot \cdot$ space, while constructing the self loops takes $\mathcal{O}(\cdot 2)$ time and space. As the cost of finding the self loops is the dominating factor, the total time used in the preprocessing phase is $\mathcal{O}(\cdot 2)$ and the total space used is $\cdot \cdot$. In the online phase, generating the prefix takes time $\mathcal{O}(2)$ and finding a linking block to one of the 2 intermediate hash values takes time $\mathcal{O}(2 \cdot \cdot)$, so the total time spent in the online phase is $\mathcal{O}(2 \cdot \cdot)$. For constant sized alphabets this leads to the following complexities: for $k \quad n/$, a tradeoff balancing the time and memory costs is $= 2^{-\ 3} = 2^{\ 2\ 3} = 2^{-4\ 3}$ giving total time and memory complexities of $\mathcal{O}(2^{2\ -\ 3})$. For $n/ \quad k \quad n/2$ the balanced time/memory tradeoff is achieved by using for each a single table with parameters $= 2^{\ 2}$ and $= 2^{\ 2-}$ giving a flat time and memory complexities of $\mathcal{O}(2^{\ 2})$. For

a non-constant sized alphabet , the general time-memory tradeoff curve is . $^2 \cdot 2^2 = 2^2 \cdot {}^2$ for k $n/$ and $2^{\ 2}$.

5 An Attack on Shoup's UOWHF

In this section, we show that our attack is generic enough to be applied against hash functions enjoying a different security property, namely Universal One-Way Hash Functions (UOWHF). A UOWHF is a family of hash functions for which any computationally bounded adversary wins the following game with negligible probability. First chooses a message , then a key is chosen at random and given to . The adversary wins if she violates the Target Collision Resistance (TCR) of , that is if she generates a message different from that collides with for the key (*i.e.*, such that ( ) = ( ) with =).

Shoup [26] proposed a simple construction for a UOWHF that hashes messages of arbitrary size, given a UOWHF that hashes messages of fixed size. It is a Merkle-Damgård-like mode of operation, but before every iteration, one of several possible masks is XORed to the chaining value. The number of masks is logarithmic in the length of the hashed message, and the order in which they are used is carefully chosen to maximize the security of the scheme. This is reminiscent of dithered hashing, except that here the dithering process does not decrease the bandwidth available to actual data.

We first describe briefly Shoup's construction, and then show how our attack can be applied against it. The complexity of the attack demonstrates that for this particular construction, Shoup's security bound is nearly tight.

5.1 Description

This construction has some similarities with Rivest's dithered hashing. It starts from a universal one way compression function that is keyed by a key , 0 1 0 1 0 1 . This compression function is then iterated, as described below, to obtain a variable input length UOWHF .

The scheme uses a set of masks $_{-1}$ (where 2 1 is the length of the longest possible message), each one of which is a random n-bit string. The key of the whole iterated function consists of and of these masks. After each application of the compression function, a mask is XORed to the chaining value. The order in which the masks are applied is defined by a specified sequence over the alphabet = 0 k 1 . The scheduling sequence is $\mathbf{z} = {}_2( )$, for 1 2 , where ${}_2( )$ denotes the largest integer such that 2 divides . Let be a message that can be split into blocks $_1$ and let h be an arbitrary n-bit string. We define $h = (h_{-1} {}_2)$, and ( ) = h .

5.2 An Attack Matching the Security Bound

In [26], Shoup proves the following security result:

Theorem 1 (Main result of [26]). *If an adversary is able to break the target collision resistance of with probability in time , then one can construct an adversary that breaks the target collision resistance of in time , with probability /2 .*

In this section we show that this bound is almost tight. First, we give an alternate definition of the dithering sequence **z**. We define:

$$= \begin{cases} 0 & \text{if } = 1, \\ {}_{-1}(\ 1)\ {}_{-1} & \text{otherwise.} \end{cases}$$

As an example, we have $_4 = 010201030102010$. It is clear that $| \ | = 2 \ 1$, and it is easy to show that for all , is a prefix of **z**. The dithering sequence is thus simply .

The most frequently-occurring factor of size $\ell \ 2$ in **z** is the prefix of size ℓ of **z**. It is a prefix of with $= \ _2(\ell + 1)$, and itself occurs about $2^{\ -}$ times in $\mathbf{z} =$. The probability for a random factor of **z** of size ℓ to be exactly this candidate is equal to the number of occurrences of this candidate divided by the number of ℓ-bit strings in **z**. Thus this probability is $\frac{2^{k-}}{2^{k-}}$. This can in turn be lower-bounded by: $2^{-} \ \frac{1}{2 \ 1}$. Our attack can be applied against the TCR property of as described above. Choose at random a (long) target message . Once the key is chosen at random, build a collision tree using a prefix of **z** of size ℓ for the dithering, and continue as described in section 4. The cost of the attack is then:

$$= 2^{\frac{n}{2}} \ ^{\frac{\ell}{2}} \ ^2 + 2(\ell + 1) \cdot 2^{\ -} \ + 2^{\ -}$$

This attack breaks the target collision resistance with probability nearly 1. Therefore, with Shoup's result, one can construct an adversary against with running time and probability of success 1/2 . If is a black box, the best attack against 's TCR property is the exhaustive search. Thus, the best attacker in time against has success probability /2 . When $n \ 3k$, $(2k + 3) \cdot 2^{\ -}$ (with $\ell = k \ 1$), and thus the best adversary running in time has success probability $\mathcal{O}\left(k/2^{\ }\right)$ when success probability of is 1/2 . This implies that there is no attack better than ours by a factor greater than $\mathcal{O}(k)$ or, in other words, there is only a factor $\mathcal{O}(k)$ between Shoup's security proof and our attack.

The ROX construction by [2], which also uses the Shoup's mask sequence to XOR with the chaining values is susceptible to the same type of attack, which is also provably near-optimal.

5.3 Comparing the Shoup and Rivest Dithering Techniques

An intriguing connection between Shoup's and Rivest's ideas shows up as soon as we notice that the scheduling sequence **z** chosen by Shoup is abelian square-free. In fact, one year after Shoup's construction was published, Mironov [22] proved that an even stronger notion of repetition-freeness was necessary: **z** is,

and has to be, *even-free*. A word is even-free if all of its non-empty factors contain at least one letter an odd number of times. Note that all even-free words are abelian square-free. We believe that the role these non-trivial sequences play in iterated constructions in cryptography (such as hashing) has yet to be completely understood.

6 Second Preimage Attack with Multiple Targets

Both the older generic second preimage results of [8, 16] and our results can be applied efficiently to multiple target messages. The work needed for these attacks depends on the number of intermediate hash values of the target message, as this determines the work needed to find a linking message from the collision tree (our attack) or expandable message ([8, 16]). A set of 2 messages, each of 2 blocks, has the same number of intermediate hash values as a single message of 2 blocks, and so the difficulty of finding a second preimage for one of a set of 2 such messages is no greater than that of finding a second preimage for a single 2 block target message. In general, for the older second preimage attacks, the total work to find one second preimage falls linearly in the number of target messages; for our attack, it falls linearly so long as the total number of blocks 2 satisfies $(n \quad)/3$.

Consider for example an application which has used SHA-1 to hash 2^3 different messages, each of 2^2 message blocks. Finding a second preimage for a given one of these messages using the attack of [16] requires about 2^{141} work. However, finding a second preimage for *any one* of these of these 2^3 target messages requires 2^{111} work. (Naturally, the attacker cannot control for *which* target message he finds a second preimage.)

This works because we can consider each intermediate hash value in each message as a potential target to which the root of the collision tree (or an expandable message) can be connected, regardless of the message it belongs to, and regardless of its length. Once we connect to an intermediate value, we have to determine to which particuliar target message it belongs. Then we can compute the second preimage of that message. Using similar logic, we can extend our attack on Rivest's dithered hashes, Shoup's UOWHF, and the ROX hash construction to apply to multiple target messages.

This observation is important for two reasons: First, simply restricting the length of messages processed by a hash function is not sufficient to block the long message attack; this is relevant for determining the necessary security parameters of future hash functions. Second, this observation allows long-message second preimage attacks to be applied to target messages of practical length. A second preimage attack which is feasible only for a message of 2 blocks has no practical relevance, as there are probably no applications which use messages of that length. A second preimage attack which can be applied to a large set of messages of, say, 2^{24} blocks, might have some practical impact. While the computational requirements of these attacks are still infeasible, this observation shows that the attacks can apply to messages of practical length.

Acknowledgments. Thanks to Orr Dunkelman for many useful suggestions and discussions, to Lily Chen and Barbara Guttman for useful comments, and to J.-P. Allouche, J. Shallit and J. Curie for pointing out the existence of abelian square-free sequences of high complexity.

The work of the first author has been funded by a Ph.D. grant of the Flemish Research Foundation and supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). This work is partly supported by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT.

References

1. Allouche, J.P.: Sur la complexité des suites infinies. Bull. Belg. Math. Soc. **1** (1994) 133–143
2. Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Seven-Property-Preserving Iterated Hashing: ROX. In Kurosawa, K., ed.: ASIACRYPT'07. Volume 4833 of Lecture Notes in Computer Science., Springer (2007) 130–146
3. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. [6] 36–57
4. Brassard, G., ed.: CRYPTO '89, Santa Barbara, California, USA, August0-24, 1989, Proceedings. In Brassard, G., ed.: CRYPTO. Volume 435 of Lecture Notes in Computer Science., Springer (1990)
5. Cobham, A.: Uniform tag seqences. Mathematical Systems Theory **6**(3) (1972) 164–192
6. Cramer, R., ed.: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. In Cramer, R., ed.: EUROCRYPT'05. Volume 3494 of Lecture Notes in Computer Science., Springer (2005)
7. Damgård, I.: A Design Principle for Hash Functions. [4] 416–427
8. Dean, R.D.: Formal Aspects of Mobile Code Security. PhD thesis, Princeton University (January 1999)
9. Ehrenfeucht, A., Lee, K.P., Rozenberg, G.: Subword Complexities of Various Classes of Deterministic Developmental Languages without Interactions. Theor. Comput. Sci. **1**(1) (1975) 59–75
10. Feller, W.: 12. In: An Introduction to Probability Theory and Its Applications. Volume 1. John Wiley & Sons (1971)
11. Hellman, M.E.: A cryptanalytic time-memory trade off. In: IEEE Transactions on Information Theory. Volume IT-26. (1980) 401–406
12. Janson, S., Lonardi, S., Szpankowski, W.: On average sequence complexity. Theor. Comput. Sci. **326**(1-3) (2004) 213–227
13. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Franklin, M.K., ed.: CRYPTO'04. Volume 3152 of Lecture Notes in Computer Science., Springer (2004) 306–316
14. Joux, A., Peyrin, T.: Hash Functions and the (Amplified) Boomerang Attack. In Menezes, A., ed.: CRYPTO. Volume 4622 of Lecture Notes in Computer Science., Springer (2007) 244–263

15. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In Vaudenay, S., ed.: EUROCRYPT'06. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 183–200
16. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. [6] 474–490
17. Keränen, V.: Abelian Squares are Avoidable on 4 Letters. In Kuich, W., ed.: ICALP. Volume 623 of Lecture Notes in Computer Science., Springer (1992) 41–52
18. Keränen, V.: On abelian square-free DT0L-languages over 4 letters. In Harju, T., ed.: WORDS'03. Volume 27., TUCS General Publication (2003) 95–109
19. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105 (2006) `http://eprint.iacr.org/`.
20. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography
21. Merkle, R.C.: One Way Hash Functions and DES. [4] 428–446
22. Mironov, I.: Hash Functions: From Merkle-Damgård to Shoup. In Pfitzmann, B., ed.: EUROCRYPT. Volume 2045 of Lecture Notes in Computer Science., Springer (2001) 166–181
23. Orr Dunkelman, E.B.: A Framework for Iterative Hash Functions — HAIFA. Presented at the second NIST hash workshop (August 24–25 2006)
24. Pansiot, J.J.: Complexité des facteurs des mots infinis engendrés par morphismes itérés. In Paredaens, J., ed.: 11th ICALP, Antwerpen. Volume 172 of LNCS., Springer (july 1984) 380–389
25. Rivest, R.L.: Abelian Square-Free Dithering for Iterated Hash Functions. Presented at ECrypt Hash Function Workshop, June 21, 2005, Cracow, and at the Cryptographic Hash workshop, November 1, 2005, Gaithersburg, Maryland (August 2005)
26. Shoup, V.: A Composition Theorem for Universal One-Way Hash Functions. In: EUROCRYPT'00. (2000) 445–452
27. Shoup, V., ed.: Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005)
28. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. [6] 1–18
29. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. [27] 17–36
30. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. [6] 19–35
31. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. [27] 1–16

A Some Sequence-Complexity Related Results

Sequences Generated by Morphisms. We say that a function is a *morphism* if for all words and , ( ) = ( ) ( ). A morphism is then entirely determined by the images of the individuals letters. A morphism is said to be -*uniform* (with ℕ) if for all word , | ( )| = · | |. If, for a given letter , we have ( ) = for some word , then is *non-erasing* for . Given a morphism and an initialization letter , let denote the n-th iterate of over : = ( ). If is -uniform (with 2) and non-erasing for , then is a strict prefix of $_1$, for all n ℕ. Let ( ) denote the

limit of this sequence: it is the only fixed point of that begins with the letter . Such infinite sequences are called *uniform tag sequences* [5] or *-automatic* sequences [1]. Because they have a very regular structure, there is a spectacular result [5] regarding the complexity of infinite sequences generated by uniform morphisms:

Theorem 2 (Cobham, 1972). *Let* $\mathbf{z}$ *be an infinite sequence generated by an -uniform morphism, and assume that the alphabet size is constant. Then* $\mathbf{z}$ *has* linear *complexity:*

$$_{\mathbf{z}}(\ell) \quad \cdot | \ | \ ^{2} \cdot \ell$$

It is worth mentioning that similar results exist in the case of sequences generated by non-uniform morphisms [24, 9], although the upper bound can be quadratic in ℓ. Since the Keraïen sequence is 85-uniform [17, 18, 25], the result of theorem 2 gives: $(\ell) \quad 1360 \cdot \ell$. This upper-bound is relatively rough, and for particular values of ℓ, it is possible to obtain a much better approximation, such as the one given in lemma 1 (which is tight). The interested reader should consult the full version of this paper.

There are Abelian Square-Free Sequences of Exponential Complexity
It is indeed possible to construct an infinite abelian square-free sequence of exponential complexity, although we do not know how to do it without slightly enlarging the alphabet.

We start with the abelian square-free Keraïen sequence $\mathbf{k}$ over , and with another sequence over 0 1 that has an exponential complexity. Such a sequence can be built for example by concatenating the binary encoding of all the consecutive integers. Then we can create a sequence $\mathbf{z}$ over the union alphabet = 0 1 by interleaving $\mathbf{k}$ and : $\mathbf{z} = \mathbf{k}\ 1 \quad 1\ \mathbf{k}\ 2 \quad 2$. The resulting shuffled sequence inherits both properties: it is still abelian square-free, and has a complexity of order $\left(2^{\ 2}\right)$.