New Second-Preimage Attacks on Hash Functions*

Elena Andreeva[1], Charles Bouillaguet[2], Orr Dunkelman[3,4],
Pierre-Alain Fouque[2], Jonathan J. Hoch[3], John Kelsey[5],
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] Faculty of Mathematics and Computer Science, Weizmann Institute of Science,
{Adi.Shamir,Yaakov.Hoch}@weizmann.ac.il
[4] Department of Computer Science, University of Haifa,
orrd@cs.haifa.ac.il
[5] National Institute of Standards and Technology,
john.kelsey@nist.gov

Abstract. In this work we present several new generic second-preimage attacks on hash functions. Our first attack is based on the herding attack, and applies to various Merkle-Damgård-based iterative hash functions. Compared to the previously known long-message second-preimage attacks, our attack offers more flexibility in choosing the second message in exchange for a small computational overhead. More concretely, in our attacks, the adversary may replace only a small number of blocks to obtain the second-preimage. As a result, the new attack is applicable to hash function constructions which were thought to be immune to the previously known second-preimage attacks. Such designs are the dithered hash proposal of Rivest, Shoup's UOWHF, and the ROX construction. We also suggest a few time-memory-data tradeoff variants for this type of attacks, allowing for a faster online phase, and even allow attacking significantly shorter messages than before.

We follow and analyze the properties of the dithering sequence used in Rivest's hash function proposal, and develop a time-memory tradeoff which allows us to apply our second-preimage attack to a wider range of dithering sequences, including sequences which are much stronger than those in Rivest's proposals. Parts of our results rely on the *kite generator*, a new time-memory tradeoff tool.

In addition to analysis of the Merkle-Damgård-like constructions, we analyze the security of the basic tree hash construction. We exhibit several second-preimage attacks on this construction, whose most notable variant is the time-memory-data tradeoff attack.

Finally, we show how both the existing second-preimage attacks and our new attacks can be applied even more efficiently when multiple shorter rather than a single long target messages are given.

Keywords: Cryptanalysis, Hash function, Dithering sequence, Second preimage attack, Herding attack, Kite Generator.

* A preliminary version of this paper appeared in Eurocrypt 2008.

1 Introduction

The recent years have been very active in the area of hash function cryptanalysis and results have come out that are of significant importance. New techniques, such as the ones by Wang *et al.* [48–51], Biham *et al.* [5], De Canniére *et al.* [13–15], Klima [29], Joux *et al.* [25], Mendel *et al.* [36, 37], Leurent [31, 32], and Sasaki *et al.* [35, 45], to name a few, have been developed to attack a wide spectrum of hash functions. These attacks target some actual constructions, while other attacks worked on more generic attacks. The results of Dean [16], Joux [23], Kelsey and Schneier [27], and Kelsey and Kohno [26], explore the resistance of the widely used Merkle-Damgård construction against several types of attacks, including multicollision attacks and second-preimage attacks.

Our work on second-preimage attacks has been motivated by the last advances, and mostly by the development of second-preimage attacks and new hash proposals that circumvent these attacks. One of the first works that describes a second-preimage attack against Merkle-Damgård constructions is in the Ph.D. thesis of Dean [16]. In his thesis, Dean presents an attack that works when fixed points of the compression function can be efficiently found. The attack has a time complexity of about $2^{n/2} + 2^{n-\kappa}$ compression function evaluations for n-bit digests where the target message is of 2^{κ} blocks.[1] Kelsey and Schneier [27] extended this result to work for *all* Merkle-Damgård hash functions (including those with no easily computable fixed points) by using the multicollision technique of Joux's [23]. Their result allows an adversary to find a second-preimage of a 2^{κ}-block target message in about $\kappa \cdot 2^{n/2+1} + 2^{n-\kappa}$ compression function calls. The main idea is to build an *expandable message*: a set of messages of varying lengths yielding the same intermediate hash result. Both mentioned attacks follow the basic approach of the *long-message attack* [38, p. 337], which computes second preimages of sufficiently long messages when the Merkle-Damgård strengthening is omitted.

Variants of the Merkle-Damgård construction that attempt to preclude the aforementioned attacks are the widepipe construction by Lucks [33], the HAIFA [6] mode of operation proposed by Biham and Dunkelman, and the "dithered" iteration by Rivest [43]. The widepipe strategy achieves the added-security by maintaining a double internal state (whilst consuming more memory and resources). A different approach is taken by the designers of HAIFA and the "dithered" hash function, who introduce an additional input to the compression function. While HAIFA uses the number of message bits hashed so far as the extra input, the dithered hash function decreases the size of the additional input to either 2 or 16 bits by using special dithering values [43]. Additionally, the properties of the "dithering" sequence were claimed by Rivest to be sufficient to avoid the second-preimage attacks of [16, 27] on the hash function.

Our new second-preimage attack is based on the *herding* attack of Kelsey and Kohno [26]. The herding attack is a method to perform a chosen-target preimage attack, whose main component is the *diamond structure*, computed offline. The diamond structure, is a collision tree of depth ℓ, with 2^{ℓ} leafs, i.e., 2^{ℓ} stating chaining values, that by a series of collisions, can all be connected to some chaining value which is in the root of the tree. This root (which we denote by $h_{\diamond}$) can be published as a target value for a message. Once the adversary is challenged with an arbitrary message prefix P, she constructs a suffix S, such that $H(P||S) = h_{\diamond}$. The suffix is composed of a block that links the prefix to the diamond structure and a series of blocks chosen according to the diamond structure. The herding attack on an n-bit hash function requires approximately $2^{(n+\ell)/2+2}$ offline computations, $2^{n-\ell}$ online computations, and 2^{ℓ} memory blocks.

1.1 Our Results

The main contribution of this paper is the development of new second-preimage attacks on the basic Merkle-Damgård hash function and most of its "dithered" variants. For Merkle-Damgård hash functions, our second-preimage attack uses a 2^{ℓ}-diamond structure [26] and works on messages of length 2^{κ} blocks in $2^{(n+\ell)/2+2}$ offline

[1] In this paper, we describe message lengths in terms of message blocks, rather than bits.

and $2^{n-\ell} + 2^{n-\kappa}$ online compression function evaluations. The attack achieves minimal total running time for $\ell \approx n/3$, yielding a total attack complexity of about $5 \cdot 2^{2n/3} + 2^{n-\kappa}$.

Our attack is slightly more expensive in terms of computation than the attack of Kelsey-Schneier [27], *e.g.*, for SHA-1 our attack requires 2^{109} time to be compared with 2^{105} for the attack of [27]. However, our new attack generates an extremely short patch: the new message differs from the original in only $\ell + 2$ blocks, compared with an average length of $2^{\kappa-1}$ blocks in [27], e.g., about 60 blocks instead of 2^{54} for SHA-1.

We also consider ways to improve one of the basic steps in long-message second-preimage attacks. In all previous results [16, 27, 38], as well as in ours, there is a step of the attack trying to connect to the chaining values of the target message. We show how to perform the connection using time-memory data tradeoff techniques. This approach reduces the online phase of the connection from $2^{n-\kappa}$ time to $2^{2(n-\kappa)/3}$ using $2^{n-\kappa}$ precomputation and $2^{2(n-\kappa)/3}$ auxiliary memory. Moreover, using this approach, one can apply the second-preimage attack for messages of lengths shorter than 2^{κ} in time which is faster than $2^{n-\lambda}$ for a 2^{λ}-block message. For example, for some reasonable values of n and κ, it is possible to produce second-preimages for messages of length $2^{n/4}$, in $\mathcal{O}\left(2^{n/2}\right)$ online time (after a $\mathcal{O}\left(2^{3n/4}\right)$ precomputation) using $\mathcal{O}\left(2^{n/2}\right)$ memory. In other words, after a precomputation which is equivalent to finding a single second preimage, the adversary can generate second preimages at the same time complexity as finding a collision in the compression function.

An important target of our new attack is the "dithered" Merkle-Damgård hash variant of [43]. For such hash functions, we exploit the short patch and the existence of repetitions in the dithering sequences. Namely, we show that the security of the dithered Merkle-Damgård hash function depends on the min-entropy of the dithering sequence, and that the sequence chosen by [43] is susceptible to our attack. For example, against the proposed 16-bit dithering sequence, our attack requires $2^{(n+\ell)/2+2} + 2^{n-\kappa+15} + 2^{n-\ell}$ work (for $\ell < 2^{13}$), which for SHA-1 is approximately 2^{120}. This is worse than the attacks against the basic Merkle-Damgård construction but it is still far less than the ideal 2^{160} second-preimage resistance expected from the dithered construction.

We further show the applicability of our attack to the universal one way hash function designed by Shoup [46], which exhibits some similarities with dithered hashing. The attack applies as well to constructions that derive from this design, *e.g.*, ROX [2]. Our technique yields the first published attack against these particular hash functions and confirms that Shoup's and ROX security bounds are tight, since there is asymptotically only a logarithmic factor (namely, $\mathcal{O}\left(\log(\kappa)\right)$) between the lower bounds given by their security proofs and our attack's complexity. To meet this end, we introduce the *multi-diamond* attack, which is a new tool that can handle more dithering sequences.

As part of our analysis of dithering sequences, we present a novel cryptographic tool — the *kite generator*. This tool can be used for long message second-preimage attacks for any dithering sequence over a small alphabet (even if the exact sequence is unknown during the precomputation phase).

We follow by presenting second-preimage attacks on tree hashes [39]. The naive version of the attack allows finding a second-preimage of a 2^{κ}-block message in time $2^{n-\kappa+1}$. We further time-memory-data tradeoff variant with time and memory complexities of $2^{2(n-\kappa+1)} = TM^2$, where T is the online time complexity and M is the memory (as long as $T \geq 2^{2\kappa}$).

Finally, we show that both the original second-preimage attacks of [16, 27] and our attacks can be extended to the case in which there are multiple target messages. We show that finding a second-preimage for any one of 2^t target messages of length 2^{κ} blocks each requires approximately the same work as finding a second-preimage for a message of $2^{\kappa+t}$ blocks.

1.2 Organization of the Paper

We describe our new second-preimage attack against the Merkle-Damgård construction in Section 2. In Section 3 we explore the use of time-memory-data tradeoff techniques in the connection step which is used in all long-message

second-preimage attacks and discuss second-preimage attacks on tree hashes. We introduce some terminology and describe the dithered Merkle-Damgård construction in Section 4, and then we extend our attack to tackle the dithered Merkle-Damgård proposals of Rivest in Section 5. We then offer a series of more general cryptanalytic tools that can attack more types of dithering sequences in Section 6. In Section 7, we show that our attacks work also against Shoup's UOWHF construction (and its derivatives). We conclude with Section 8 showing how to apply second-preimage attacks on a large set of target messages.

2 A New Generic Second-Preimage Attack

2.1 The Merkle-Damgård construction

We first describe briefly the strengthened Merkle-Damgård construction $H^f : \{0,1\}^* \to \{0,1\}^n$, which is built by iterating a compression function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. To hash a message m apply the following process:

- Pad and split the message M into r blocks $x_1, \ldots, x_r$ of m bits each.
- Set h_0 to the initialization value IV.
- For each message block i compute $h_i = f(h_{i-1}, x_i)$.
- Output $H^f(M) = h_r$.

Throughout the paper we shall use $\mathcal{T} = \{h_i\}$ to denote the set of all chaining values encountered while hashing the message m.

The common padding rule (referred to as the Merkle-Damgård strengthening) appends to the original message a single '1' bit followed by as many '0' bits as needed to complete an m-bit block after embedding the message length at the end. Merkle [39] and Damgård [12] proved independently that the scheme is collision-resistance preserving, in the sense that a collision on the hash function H^f implies a collision on the compression function f. As a side effect, the strengthening used defines a limit on the maximal length for admissible messages. In most deployed hash functions, this limit is 2^{64} bits, or equivalently 2^{55} 512-bit blocks. In the sequel, we denote the maximal number of admissible blocks by 2^κ.

2.2 Our Second-Preimage Attack on Merkle-Damgård hash

Our new technique to find second-preimages on Merkle-Damgård hash functions relies heavily on the diamond structure introduced by Kelsey and Kohno [26].

Diamond Structure: A diamond structure of size ℓ is a multicollision with the shape of a complete bineary tree of depth ℓ with2^ℓ leaves denoted by $\hat{h}_i$ (hence we often refer to it as a *collision tree*). The tree nodes are labeled by the n-bit chaining values, and the edges are labeled by the m-bit message blocks. A message block is mapped between two evolving states of the chaining value by the compression function f. Thus, there is a path labeled by the ℓ message blocks from any one of the 2^ℓ starting leaf nodes that leads to the same final chaining value $\hat{h}_\diamond$ at the root of the tree. We illustrate the diamond structure in Figure 1.

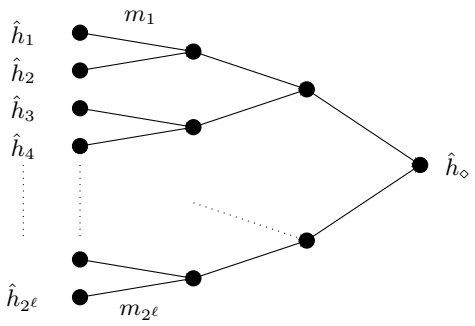


Fig. 1. A Diamond Structure

Algorithm 1 *Our New Attack Algorithm on Standard Merkle-Damgård Hash Functions*

1. *Construct a collision tree of depth ℓ with a final chaining value at the root $\hat{h}_\diamond$.*
2. *Try random message blocks B, until $f\left(\hat{h}_\diamond, B\right) \in \mathcal{T}$.* [2] *Let $B^{\searrow}$ be the message block and let $f\left(\hat{h}_\diamond, B^{\searrow}\right) = h_{i_0}$ for some i_0, $\ell + 1 \leq i_0 < \left|M\right|$.*
3. *Pick a prefix P of size $i_0 - \ell - 2$ blocks, and let h_P be the chaining value obtained after processing P by the hash function. Try random message blocks B, until $f\left(h_P, B\right) = \hat{h}_j$ for some $\hat{h}_j$ labeling a leaf of the diamond. Let $B^{\nearrow}$ denote this block, and let T be the chain of ℓ blocks traversing the diamond from $\hat{h}_j$ to $\hat{h}_\diamond$.*
4. *Form a message $M' = P||B^{\nearrow}||T||B^{\searrow}||M_{\geq i_0+1}$.*

The Attack: To illustrate our new second-preimage attack, let M be a target message of length 2^κ blocks. The main idea of our attack is that connecting the target message to a precomputed collision tree of size ℓ can be done with $2^{n-\ell}$ computations. Moreover, connecting the root of the tree to one of the 2^κ chaining values encountered during the computation of $H^f(M)$ takes only $2^{n-\kappa}$ compression function calls. Since a diamond structure can be computed in time much less than 2^n, we can successfully launch a second-preimage attack. The attack works in four steps as described in Algorithm 1 (and depicted in Figure 2).

The messages M' and M are of equal length and hash to the same value before strengthening, so they produce the same hash value with the added Merkle-Damgård strengthening.

Our new second-preimage attack applies identically to other Merkle-Damgård based constructions, such as prefix-free Merkle-Damgård [10], randomized hash [20], Enveloped Merkle-Damgård [3], etc. Keyed hash constructions like the linear and the XOR linear hash by [4] use unique per message block key, which foils this style of attacks in the connection step (as well as the attack of [27]).

Complexity. The first step allows for precomputation and its time and space complexity is about $2^{(n+\ell)/2+2}$ (see [26]). The second step of the attack is carried out online with $2^{n-\kappa}$ work, and the third step takes $2^{n-\ell}$ work. The total time complexity of the attack is then $2^{(n+\ell)/2+2}$ precomputation and $2^{n-\kappa} + 2^{n-\ell}$ online computations and their sum is minimal when $\ell = (n-4)/3$ for a total of about $5 \cdot 2^{2n/3} + 2^{n-\kappa}$ computations.

[2] Recall that $\mathcal{T}$ contains the chaining values encountered while hashing the target message m.

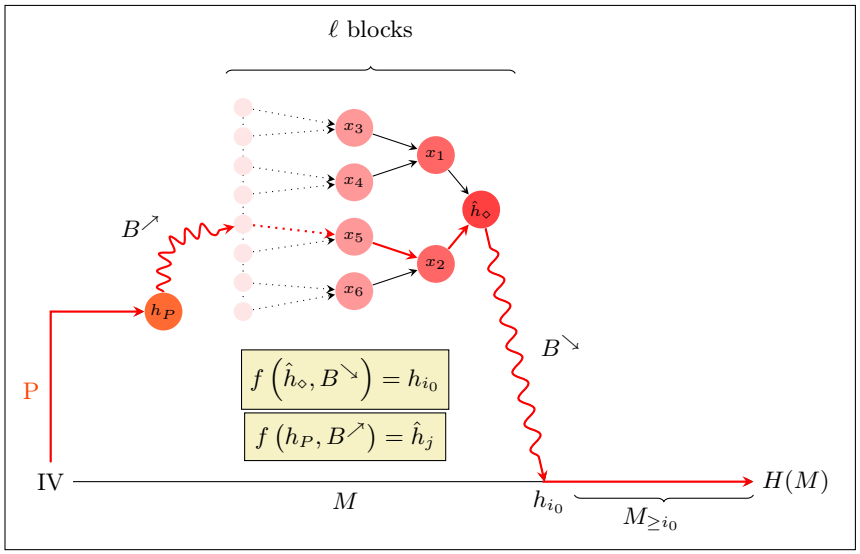


Fig. 2: Representation of Our New Attack on Standard Merkle-Damgård.

2.3 Attack Variants on Strengthened Merkle-Damgård

Variant 1: The above algorithm allows connecting in the third step of the attack only to the 2^ℓ chaining values at the first level of the diamond structure. It is possible, however, to use all the $2^{\ell+1}-1$ chaining values in the diamond structure by appending to $h_\diamond$ a short expandable message of lengths between $\log_2(\ell)$ and $\ell+\log_2(\ell)-1$. Thus, once the prefix P is connected to some chaining value in the diamond structure, it is possible to extend the length of the patch to be of a fixed length (as required by the attack). This variant requires slightly more work in the precomputation step and a slightly longer patch (of $\log_2(\ell)$ more blocks). The offline computation cost is about $2^{(n+\ell)/2+2}+\log_2(\ell)\cdot 2^{n/2+1}+\ell \approx 2^{(n+\ell)/2+2}$, while the online computation cost is reduced to $2^{n-\ell-1}+2^{n-\kappa}$ compression function calls.

Variant 2: A different variant of the attack suggests constructing the diamond structure by reusing the chaining values of the target message M as starting points. Here the diamond structure gets computed in the online phase. In this variant, the herding step becomes more efficient, as there is no need to find a block connecting to the diamond structure. In exchange, we need an expandable message at the output of the diamond structure (*i.e.*, starting from $h_\diamond$). The complexity of this variant is $2^{(n+\kappa)/2+2}+2^{n-\kappa}+\kappa\cdot 2^{n/2+1}+2^\kappa \approx 2^{(n+\kappa)/2+2}+2^{n-\kappa}+2^\kappa$ online compression function calls (note that 2^κ is also the size of the diamond structure).

2.4 Comparison with Dean [16] and Kelsey and Schneier [27]

The attacks of [16, 27] are slightly more efficient than ours. We present the respective offline and online complexities for these previous and our new attack in Table 1 and the comparison of these attacks for MD5 ($n=128, \kappa=55$), SHA-1 ($n=160, \kappa=55$), SHA-256 ($n=256, \kappa=118$), and SHA-512 ($n=512, \kappa=118$) in Table 2. Still, our technique gives the adversary more control over the second-preimage. For example, she could choose to reuse most of the target message, leading to a second preimage that differs from the original by only $\ell+2$ blocks.

Attack	Complexity			Avg. Patch Size	Message Length
	Offline	Online	Memory		
Dean [16]*	$2^{n/2+1}$	$2^{n-\kappa}$	2	$2^{\kappa-1}$	2^{κ}
Kelsey-Schneier [27]	$\kappa \cdot 2^{n/2+1} + 2^{\kappa}$	$2^{n-\kappa}$	$2 \cdot \kappa$	$2^{\kappa-1}$	2^{κ}
New	$2^{(n+\ell)/2+2}$	$2^{n-\ell} + 2^{n-\kappa}$	$2^{\ell+1}$	$\ell + 2$	2^{κ}
Variant 1	$2^{(n+\ell)/2+2}$	$2^{n-\ell-1} + 2^{n-\kappa}$	$2^{\ell+1} + 2 \cdot \log_2(\ell)$	$\ell + \log_2(\ell) + 2$	2^{κ}
Variant 2	—	$2^{(n+\kappa)/2+2} + 2^{n-\kappa} + 2^{\kappa}$	$2^{(n+\kappa)/2} + 2^{\kappa+1}$	$2^{\kappa-1}$	2^{κ}
First connection with with TMDTO (Sect. 3.2)	$2^{(n+\ell)/2+2} + 2^{n-\lambda}$	$2^{n-\ell} + 2^{2\lambda}$	$2^{\ell+1} + 2^{n-2\lambda}$	$\ell + 2$	2^{λ}

* — This attack assumes the existence of easily found fixed points in the compression function

Table 1. Comparison of Long Message Second-Preimage Attacks

The main difference between the older techniques and ours is that the previous attacks build on the use of expandable messages. We note that our attack just offers a short patch. At the same time, 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 $2^{(n+\ell)/2+2} + 2^{n-\ell}$. Although it is more expensive, this new technique can be adapted to work even when an additional dithering input is given, as we demonstrate in Section 5.

3 Time-Memory-Data Tradeoffs for Second-Preimage Attacks

In this section we discuss the first connection step (from the diamond structure to the message) and we show that it can be implemented using time-memory-data tradeoff. This allows speeding up the online phase in exchange for an additional precomputation and memory. An additional and important advantage is our ability to find second-preimages of significantly shorter messages. These ideas can also be used to offer second-preimage attacks on tree hashes.

3.1 Hellman's Time-Memory Tradeoff Attack

Time-memory Tradeoff attacks (TMTO) were first introduced in 1980 by Hellman [21]. The idea is to improve brute force attacks by trading the online time for memory and precomputation when inverting a function $f : \{0,1\}^n \to \{0,1\}^n$. Suppose we have an image element y and we wish to find a pre-image $x \in f^{-1}(y)$. One extreme would be to go over all possible elements x until we find one such that $f(x) = y$, while the other extreme would be to precompute a huge table containing all the pairs $(x, f(x))$ sorted by the second element. Hellman's idea is to consider what happens when applying f iteratively. We start at a random element x_0 and compute $x_{i+1} = f(x_i)$ for t steps saving only the start and end points of the generated chain (x_0, x_t). We repeat this process with different initial points and generate a total of c chains. Given an input y, we start generating a chain starting from y and checking if we reached one of the saved endpoints. If we have, we generate the corresponding chain, starting from the suggested starting point and hope to find a preimage of y. Notice that as the number of chains, c, increases beyond $2^n/t^2$, the contribution (*i.e.*, the number of new values that can be inverted) from additional chains decreases. To counter this birthday paradox effect, Hellman suggested to construct a number of tables,

Function (n, κ)		MD5 (128,55)	SHA-1 (160,55)	SHA-256 (256,118)	SHA-512 (512,118)
Dean [16]	Offline:	2^{65}	2^{81}	2^{129}	2^{257}
	Online:	2^{73}	2^{105}	2^{138}	2^{394}
	Memory:	2	2	2	2
	Patch Length:	2^{54}	2^{54}	2^{117}	2^{117}
Kelsey-Schneier [27]	Offline:	2^{71}	2^{87}	2^{136}	2^{264}
	Online:	2^{73}	2^{105}	2^{138}	2^{394}
	Memory:	110	110	234	234
	Patch Length:	2^{54}	2^{54}	2^{117}	2^{117}
New	Offline:	$2^{93.5}$	$2^{109.5}$	2^{189}	2^{317}
	Online:	2^{74}	2^{106}	2^{139}	2^{395}
	Memory:	2^{56}	2^{56}	2^{119}	2^{119}
	Patch Length:	57	57	120	120
Variant 1	Offline:	2^{93}	2^{109}	$2^{188.5}$	$2^{316.5}$
	Online:	2^{74}	2^{106}	2^{139}	2^{395}
	Memory:	2^{55}	2^{55}	2^{118}	2^{118}
	Patch Length:	62	62	126	126
Variant 2	Online:	$2^{87.7}$	2^{109}	2^{173}	2^{394}
	Memory:	$2^{84.7}$	2^{106}	2^{170}	2^{315}
	Patch Length:	$2^{40.3}$	2^{51}	2^{83}	2^{117}
	Length*:	$2^{41.3}$	2^{52}	2^{84}	2^{118}
First connection with TMDTO (setting online time equal to memory)	Offline:	$2^{98.3}$	$2^{122.3}$	$2^{194.3}$	2^{394}
	Online:	2^{65}	2^{81}	2^{129}	2^{257}
	Memory:	$2^{65.6}$	$2^{81.6}$	$2^{129.6}$	$2^{257.6}$
	Patch Length:	66	82	130	258
	Length*:	2^{32}	2^{40}	2^{64}	2^{118}

Memory, patch length, and message lengths are measured in blocks.
* — Length is given for cases where messages shorter than 2^κ can be used without effect on the time complexities.

Table 2. Comparison of the long-message second-preimage attacks on real hash functions (optimized for minimal online complexity)

each using a slightly different function f_i, such that knowing a preimage of y under f_i implies knowing such a preimage under f. Hellman's original suggestion, which works well in practice, is to use $f_i(x) = f(x \oplus i)$. Thus, if we create $d = 2^{n/3}$ tables each with different f_i's, such that each table contains $c = 2^{n/3}$ chains of length $t = 2^{n/3}$, about 80% of the 2^n points will be covered by at least one table. Notice that the online time complexity of Hellman's algorithm is $t \cdot d = 2^{2n/3}$ while the memory requirements are $d \cdot c = 2^{2n/3}$.

It is worth mentioning, that when multiple targets are given for inversion (*i.e.*, a set of possible targets $y_i = f(x_i)$), where it is sufficient to identify only one of the preimages (x_i for some i), one could offer better tradeoff curves. For example, given m possible targets, it is possible to reduce the number of tables stored by a factor of m, and trying for each of the possible targets, the attack (*i.e.*, apply the chain). This reduces the memory complexity (without affecting the online time complexity or the success rate), as long as $m \leq d$ (see [7] for more details concerning this constraint).

3.2 Time-Memory-Data Tradeoffs for Merkle-Damgård Second-Preimage Attacks

Both known long-message second-preimage attacks and our newly proposed second-preimage attack assume that the target message is long enough (up to the 2^κ limit). This enables the connection to the target message (namely, finding $B^{\searrow}$) to be done with complexity of about $2^{n-\kappa}$ compression function calls. In our new second preimage attack we also have a second connection phase: connecting from the message into the diamond structure (Step 3 of Alg. 1). In principle, both connection steps can be seen as finding the inverse of a function. Luckily, we can improve the first connection (which is common to all attacks) by using a time-memory-data tradeoff. The result of the tradeoff is that after a precomputation whose complexity is essentially that of finding a second preimage, the cost of finding subsequent second preimages becomes essentially that of finding collisions.

Recall that we search for a message block m such that $f(h_\diamond, m) = h_i$. As there are 2^κ targets (and finding the preimage of only one h_i's is sufficient), then we can run a time-memory-data tradeoff attack with a search space of $N = 2^n$, and $D = 2^\kappa$ available data points, time T, and memory M such that $N^2 = TM^2D^2$, after $P = N/D$ preprocessing (and $T \geq D^2$). Let 2^x be the online complexity of the time-memory-data tradeoff, and thus, $2^x \geq 2^{2\kappa}$, and the memory consumption is $2^{n-\kappa-x/2}$ blocks of memory. The resulting overall complexities are: $2^{n/2+\ell/2+2} + 2^{n-\kappa}$ preprocessing, $2^x + 2^{n-\ell}$ online complexity, and $2^{\ell+1} + 2^{n-\kappa-x/2}$ memory, for messages of $2^{x/2}$ blocks.

Given the constraints on the online complexity (*i.e.*, $x \geq 2\kappa$), it is sometime beneficial to consider shorter messages, *e.g.*, of 2^λ blocks (for $\lambda \leq \kappa$). For such cases, the offline complexity is $2^{n/2+\ell/2+2} + 2^{n-\lambda}$, the online complexity is $2^x + 2^{n-\ell}$, and the memory consumption being $2^{n-\lambda-x/2} + 2^{\ell+1}$. We can balance the online and memory complexities (as commonly done in time-memory-data tradeoff attacks), which results in picking x such that $2^x + 2^{n-\ell} \approx 2^{n-\lambda-x/2} + 2^{\ell+1}$. By picking $\lambda = n/4$, $x = 2\lambda = n/4$, and $\ell = n/2$, the online complexity is $2^{n/2+1}$, the memory complexity is $3 \cdot 2^{n/2}$, and the offline complexity is $5 \cdot 2^{3n/4}$. This of course holds as long as $n/4 = \lambda \leq \kappa$, *i.e.*, $4\kappa > n$.

When $4\kappa < n$, we can still balance the memory and the online computation by picking $T = 2^{n/2}$ and $\ell = n/2$. The memory consumption of this approach is still $\mathcal{O}\left(2^{n/2}\right)$, and the only difference is the preprocessing which increases to $2^{n-\kappa}$.

For this choice of parameters, we can find a second-preimage for a 2^{40}-block long message in SHA-1, with online time of 2^{81} operations, $2^{81.6}$ blocks of memory, and $2^{122.2}$ steps of precomputation. The equivalent Kelsey-Schneier attack takes 2^{120} online steps (and about $2^{85.3}$ offline computation).

One may consider comparison with a standard time-memory attack for finding preimages.[3] For an n-bit digests, for 2^n preprocessing, one can find a (second-) preimage using time 2^x and memory $2^{n-x/2}$. Hence, for the same 2^{40}-block message, with $2^{81.6}$ blocks of memory, the online computation is about $2^{156.8}$ SHA-1 compression function calls.

3.3 Time-Memory-Data Tradeoffs for Tree Hashes Second-Preimage Attacks

Another structure which is susceptible to the time-memory-data connection phase is tree hashes. Before describing our attacks, we give a quick overview of tree hashes.

Tree Hashes. Tree hashes were first suggested in [39]. Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a compression function used in the tree hash T_f. To hash a message M of length $|M| < 2^n$, M is initially padded with a single

[3] An attack that tries to deal with the multiple targets has to take care of the padding, which can be done by just starting from an expandable message. In other words, this is equivalent to using our new connection step in the Kelsey-Schneier attack.

'1' bit and as many '0' bits as needed to obtain $\mathrm{pad}_{\mathrm{TH}}(M) = m_1\|m_2\|\ldots\|m_L$, where each m_i is n-bit long, $L = 2^\kappa$ for $\kappa = \lceil \log_2(|M|+1)/n \rceil$. Consider the resulting message blocks as the leaves of a full binary tree of depth κ. Then, the compression function is applied to any two leaves with a common ancestor, and its output is assigned to the common ancestor. This procedure is followed in an iterative manner. A final compression function is applied to the output of the root and an extra final strengthening block, normally containing the length of the input message M. The resulting output is the final tree hash.

Formally, the tree hash function $T_f(M)$ is defined as:

1. $m_1\|m_2\|\ldots\|m_L \leftarrow \mathrm{pad}_{\mathrm{TH}}(M)$
2. For $j = 1$ to $2^{\kappa-1}$ compute $h_{1,j} = f(m_{2j-1}, m_{2j})$
3. For $i = 2$ to κ:
   - For $j = 1$ to $2^{\kappa-i}$ compute $h_{i,j} = f(h_{i-1,2j-1}, h_{i-1,2j})$
4. $T_f(M) \triangleq h_{\kappa+1} = f(h_{\kappa,1}, \langle |M| \rangle_n)$.

A Basic Second-Preimage Attack on Tree Hashes. Tree hashes that apply the same compression function to each message block (*i.e.*, the only difference between $f(m_{2i-1}, m_{2i})$ and $f(m_{2j-1}, m_{2j})$ for $i \neq j$ is the position of the resulting node in the tree) are vulnerable to a long-message second-preimage attack where the change is in at most two blocks of the message.

Recall that $h_{1,j} = f(m_{2j-1}, m_{2j})$ for $j = 1$ to $2^{\kappa-1}$ for a message M of length 2^κ blocks. Then, given the target message M, there are $2^{\kappa-1}$ chaining values $h_{1,j}$ that can be targeted.[4] An adversary that inverts one of these chaining values, *i.e.*, produces (m', m'') such that $f(m', m'') = h_{1,j}$ for some $1 \leq j \leq 2^{\kappa-1}$, computes successfully a second-preimage M'. Thus, a long-message second-preimage attack on message of length 2^κ requires about $2^{n-\kappa+1}$ trial inversions for $f(\cdot)$.

More precisely, the adversary just tries message pairs (m', m''), until $f(m', m'') = h_{1,j}$ for some $1 \leq j \leq 2^{\kappa-1}$. Then, the adversary replaces $(m_{2j-1}||m_{2j})$ with $m'||m'$ without affecting the computed hash value for M. Note that the number of modified message blocks is only two. This result also applies to other parallel modes where the exact position has no effect on the way the blocks are compressed.

Getting More for Less As can be seen, the previous attack tries to connect only to the first level of the tree. This fact stems from the fact that in order to connect to a higher level in the tree, one needs the ability to replace the subtree below the connection point.

Assuming that f is random enough, we can achieve this, by building the queries carefully. Consider the case where the adversary computes $n_1 = f(m'_1, m'_2)$ and $n_2 = f(m'_3, m'_4)$, for some message blocks $m'_1, \ldots, m'_4$. If neither n_1 nor n_2 are equal to some $h_{1,j}$, we can compute $o_1 = f(n_1, n_2)$. Now, if $o_1 = h_{1,j}$ for some j, we can offer a second preimage as before (replacing the corresponding message blocks by (n_1, n_2)). At the same time, if $o_1 = h_{2,j}$ for some j, we can replace the four message blocks $m_{4j-3}, \ldots, m_{4j}$ with $m'_1, \ldots, m'_4$. The probability of a successful connection is thus $3 \cdot 2^{\kappa-1-n} + 2^{\kappa-2-n} = 3.5 \cdot 2^{\kappa-1-n}$ for 3 compression function calls (rather than the expected $3 \cdot 2^{\kappa-1-n}$).

One can extend this approach, and try to connect to the third layer of the tree. This can be done by generating o_2 using four new message blocks, and if their connection fails, compute $f(o_1, o_2)$ and trying to connect it the first three levels of the tree. Hence, for a total of 7 compression function calls, we expect a success probability of $2 \cdot 3.5 \cdot 2^{\kappa-1-n} + 2^{\kappa-1-n} + 2^{\kappa-2-n} + 2^{\kappa-3-n} = 8.75 \cdot 2^{\kappa-1-n}$.

[4] We note that the number of possible locations for connection is $2^{\kappa-1}$ even if there are more compression function calls. This follows from the fact that the length of the second-preimage must be the same as for the original message, and thus, it is impossible to connect to a chaining value in the padding.

This approach can be further generalized, each time increasing the depth of the subtree which is replaced (up to κ). If the number of compression function calls needed to generate a subtree of depth t is $N_t = 2^t - 1$ and the probability of successful connection is p_t, then p_t follows the recursive formulas of:

$$p_{t+1} = 2p_t + \sum_{i=1}^{t+1} 2^{\kappa-i-n},$$

where $p_1 = 2^{\kappa-1-n}$. The time complexity advantage of this approach is $p_{t+1}/(N_t \cdot 2^{\kappa-1-n})$, as for the basic algorithm, after N_t compression function calls, the success rate is $N_t \cdot 2^{\kappa-1-n}$. Now, as $p_{t+1} < 2p_t + 2 \cdot 2^{\kappa-1-n}$, it is possible to upper bound p_κ by $2 + 4 \cdot 2^{\kappa-1}$, meaning that this attack is at most twice as fast as the original attack presented above.

The main drawback of this approach is the need to store the intermediate chaining values produced by the adversary. For a subtree of depth t, this sums up to $2^{t+1} - 1$ blocks of memory.

We notice that the utility of each new layer decreases. Hence, we propose a slightly different approach, where the utility is better. The improved variant starts by computing $n_1 = f(m'_1, m'_2)$ and $n_2 = f(m'_3, m'_4)$. At this point, the adversary computes 4 new values — $f(n_1, n_1), f(n_1, n_2), f(n_2, n_1)$, and $f(n_2, n_2)$. For these 6 compression function calls, the adversary has a probability of $6 \cdot 2^{\kappa-1-n} + 4 \cdot 2^{\kappa-2-n} = 8 \cdot 2^{\kappa-1-n}$ chance of connecting successfully to the message (either at the first level or the second level for the four relevant values). It is possible to continue this approach, and obtain 16 chaining values that can be connected in the first, second, or third levels of the tree.

This approach yields the same factor 2 improvement in the total time complexity with less memory, and with less restrictions on κ, namely, to obtain the full advantage, $\log_2(n)$ levels in the tree are needed (to be compared with n levels in the previous case).

Applying Time-Memory-Data Tradeoffs. As in the Merkle-Damgård second-preimage attacks, we model the inversion of f as a task for a time-memory-data attack [7]. The $h_{1,j}$ values are the multiple targets, which compose the available data points $D = 2^{\kappa-1}$. Using the time-memory-data curve of the attack from [7], it is possible to have an inversion attack which satisfy the relation $N^2 = TM^2D^2$, where N is the size of the output space of f, T is the online computation, and M is the number of memory blocks used to store the tables of the attack. As $N = 2^n$, we obtain that the curve for this attack is $2^{2(n-\kappa+1)} = TM^2$ (with preprocessing of $2^{n-\kappa+1}$). We note that the tradeoff curve can be used as long as $M < N, T < N$, and $T \geq D^2$. Thus, for $\kappa < n/3$, it is possible to choose $T = M$, and obtain the curve $T = M = 2^{2(n-\kappa+1)/3}$. For $n = 160$ with $\kappa = 50$, one can apply the time-memory-data tradeoff using 2^{110} preprocessing time and 2^{74} memory blocks, and find a second-preimage in 2^{74} online computation.

4 Dithered Hashing

The general idea of dithered hashing is to perturbate the hash process by using an additional input to the compression function, formed by the consecutive elements of a fixed *dithering* sequence. This gives the adversary less control over the inputs of the compression function, and makes the hash of a message block dependent on its position in the whole message.

The ability to "copy, cut, and paste" blocks of messages is a fundamental ingredient in many generic attacks, including for example the construction of expandable messages of [27] or of the diamond structure of [26]. To prevent such generic attacks, the use of some kind of dithering is now widely adopted, e.g., in the two SHA-3 finalists Blake and Skein.

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 $\mathcal{A}$ be a finite alphabet, and let the dithering sequence $\mathbf{z}$ be an eventually infinite word over $\mathcal{A}$. Let $\mathbf{z}[i]$ denote the i-th element of $\mathbf{z}$. The dithered Merkle-Damgård construction is obtained by setting $h_i = f\left(h_{i-1}, x_i, \mathbf{z}\left[i\right]\right)$ in the definition of the Merkle-Damgård scheme.

We demonstrate that the gained security (against our attack) of the dithering sequence is equal to its min-entropy of $\mathbf{z}$. This implies that to offer a complete security against our attacks, the construction must use a dithering sequence which contains as many different dithering inputs as blocks, *e.g.*, like suggested in HAIFA.

4.1 Background and Notations

Words and Sequences. Let ω be a word over a finite alphabet $\mathcal{A}$. We use the dot operator to denote concatenation. If ω can be written as $\omega = x.y.z$ (where x,y, or z can be empty), we say that x is a *prefix* of ω and that y is a *factor* of ω. A finite non-empty word ω is a *square* if it can be written as $\omega = x.x$, where x is not empty. A finite word ω is an *abelian square* if it can be written as $\omega = x.x'$ where x' is a permutation of x (*i.e.*, a reordering of the letters of x). A word is said to be *square-free* (respectively, *abelian square-free*) if none of its factors is a square (respectively, an abelian square). Note that abelian square-free words are also square-free.

Sequences Generated by Morphisms. We say that a function $\tau : \mathcal{A}^* \to \mathcal{A}^*$ is a *morphism* if for all words x and y, $\tau(x.y) = \tau(x).\tau(y)$. A morphism is then entirely determined by the images of the individuals letters. A morphism is said to be *r-uniform* (with $r \in \mathbb{N}$) if for any word x, $|\tau(x)| = r \cdot |x|$. If, for a given letter $\alpha \in \mathcal{A}$, we have $\tau(\alpha) = \alpha.x$ for some word x, then τ is *non-erasing* for α. Given a morphism τ and an initialization letter α, let u_n denote the n-th iterate of τ over α: $u_n = \tau^n(\alpha)$. If τ is r-uniform (with $r \geq 2$) and non-erasing for α, then u_n is a strict prefix of u_{n+1}, for all $n \in \mathbb{N}$. Let $\tau^\infty(\alpha)$ denote the limit of this sequence: it is the only fixed point of τ that begins with the letter α. Such infinite sequences are called *uniform tag sequences* [9] or *r-automatic* sequences [1].

An Infinite Abelian Square-Free Sequence. Infinite square-free sequences have been known to exist since 1906, when Axel Thue exhibited the Thue-Morse word over a ternary alphabet (there are no square-free sequences longer than four on a binary alphabet).

The question of the existence of infinite *abelian* square-free sequences was raised by 1961 by Erdös, and was solved by Pleasants [42] in 1970: he exhibited an infinite abelian square-free sequence over a five-letter alphabet. In 1992, Keränen [28] exhibited an infinite abelian square-free sequence $\mathbf{k}$ 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*. Before describing it, let us consider the permutation σ over $\mathcal{A}$ defined by:

$$\sigma(a) = b, \quad \sigma(b) = c, \quad \sigma(c) = d, \quad \sigma(d) = a$$

Surprisingly enough, the Keränen sequence is defined as the fixed point of a 85-uniform morphism τ, given by:

$$\tau(a) = \omega_a, \quad \tau(b) = \sigma\left(\omega_a\right), \quad \tau(c) = \sigma^2\left(\omega_a\right), \quad \tau(d) = \sigma^3\left(\omega_a\right),$$

where ω_a is some magic string of size 85 (given in [28, 43]).

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 $Fact_{\mathbf{z}}(\ell)$ the number of factors of size ℓ of the sequence $\mathbf{z}$.

Because they have a very strong structure, r-uniform sequences have special properties, especially with regard to their complexity:

Theorem 1 (Cobham, 1972, [9]). *Let* $\mathbf{z}$ *be an infinite sequence generated by an* r*-uniform morphism, and assume that the alphabet size* $\left|\mathcal{A}\right|$ *is finite. Then* $\mathbf{z}$ *has* linear *complexity bounded by:*

$$Fact_{\mathbf{z}}(\ell) \leq r \cdot |\mathcal{A}|^2 \cdot \ell.$$

A polynomial algorithm which computes the exact set of factors of a given length ℓ can be deduced from the proof of this theorem. It is worth mentioning that similar results exist in the case of sequences generated by non-uniform morphisms [17, 41], although the upper bound can be quadratic in ℓ. The bound given by this theorem, although attained by certain sequences, is relatively rough. For example, since the Keraïen sequence is 85-uniform, the theorem gives: $Fact_{\mathbf{k}}(\ell) \leq 1360 \cdot \ell$. For $\ell = 50$, this gives $Fact_{\mathbf{k}}(50) \leq 68000$, while the factor-counting algorithm reveals that $Fact_{\mathbf{k}}(50) = 732$. Hence, for small values of ℓ, the following upper bound may be tighter:

Lemma 1. *Let* $\mathbf{z}$ *be an infinite sequence over the alphabet* $\mathcal{A}$ *generated by an* r*-uniform morphism* τ*. For all* ℓ*,* $1 \leq \ell \leq r$*, we have :*

$$Fact_{\mathbf{z}}(\ell) \leq \ell \cdot \Big(Fact_{\mathbf{z}}(2) - |\mathcal{A}|\Big) + \Big[(r+1) \cdot |\mathcal{A}| - Fact_{\mathbf{z}}(2)\Big].$$

Proof. If $\ell \leq r$, then any factor of $\mathbf{z}$ of size ℓ falls in one of these two classes:

- Either it is a factor of $\tau(\alpha)$ for some letter $\alpha \in \mathcal{A}$. There are no more than $|A| \cdot (r - \ell + 1)$ such factors.
- Or it is a factor of $\tau(\alpha).\tau(\beta)$, for two letters $\alpha, \beta \in \mathcal{A}$ (and is not a factor of either $\tau(\alpha)$ or $\tau(\beta)$). For any given pair (α, β), there can only be $\ell - 1$ such factors. Moreover, $\alpha.\beta$ must be a factor of size 2 of $\mathbf{z}$.

So $Fact_{\mathbf{z}}(\ell) \leq |A| \cdot (r - \ell + 1) + Fact_{\mathbf{z}}(2) \cdot (\ell - 1)$. □

For the particular case of the Keränen sequence $\mathbf{k}$, we have $r = 85$, $\left|\mathcal{A}\right| = 4$ and $Fact_{\mathbf{k}}(2) = 12$ (all non-repeating pairs of letters). This yields $Fact_{\mathbf{k}}(\ell) \leq 8 \cdot \ell + 332$ when $\ell \leq 85$, which is tight, as for $\ell = 50$ it gives: $Fact_{\mathbf{k}}(50) \leq 732$.

Factor Frequency. Our attacks usually target the factor of highest frequency. If the frequency of the various factors is biased, *i.e.*, non uniform, then the attack should exploit this bias (just like in any cryptographic attack).

Formally, let us denote by $N_\omega(x)$ the number of occurrences of ω in x (which is expected to be a finite word), and by $\mathbf{z}[1..i]$ the prefix of $\mathbf{z}$ of size i. The *frequency* of a given word ω in the sequence $\mathbf{z}$ is the limit of $N_\omega(\mathbf{z}[1..i])/i$ when i goes to $+\infty$.

We denote by $2^{-H_\infty(\mathbf{z},\ell)}$ the frequency of the most frequent factor of length ℓ in the sequence $\mathbf{z}$. It follows immediately that $H_\infty(\mathbf{z}, \ell) \leq \log_2 Fact_{\mathbf{z}}(\ell)$. Hence, when the computation of $H_\infty(\mathbf{z}, \ell)$ is infeasible, $\log_2 Fact_{\mathbf{z}}(\ell)$ can be used as an upper-bound.

It is possible to determine precisely the frequency of certain words in sequences generated by uniform morphisms. For instance, it is easy to compute the frequency of individual letters: if x is some finite word and $\alpha \in \mathcal{A}$, then by definition of τ we find:

$$N_\alpha\left(\tau\left(x\right)\right) = \sum_{\beta \in \mathcal{A}} N_\alpha\left(\tau\left(\beta\right)\right) \cdot N_\beta\left(x\right) \tag{1}$$

In this formula, $N_\alpha(\tau(\beta))$ is easy to determine from the description of the morphism τ. Let us write:

$$\begin{aligned}\mathcal{A} &= \{\alpha_1, \ldots, \alpha_k\}, \\ U_s &= \left(\frac{N_{\alpha_j}\left(\tau^s\left(a\right)\right)}{\ell^s}\right)_{1\leq j\leq|\mathcal{A}|}, \\ M &= \left(\frac{N_{\alpha_i}(\tau(\alpha_j))}{\ell}\right)_{1\leq i,j\leq|\mathcal{A}|}.\end{aligned}$$

Then it follows from equation (1) that:

$$U_{s+1} = M \cdot U_s.$$

The frequency of individual letters is given by the vector $U_\infty = \lim_{s\to\infty} U_s$. Fortunately, this vector lies in the kernel of $M-1$ (and is such that its component sum up to one). For instance, for the Keränen sequence, and because of the very symmetric nature of τ, we find that M is a circulant matrix:

$$85 \cdot M = \begin{pmatrix} 19 & 18 & 27 & 21 \\ 21 & 19 & 18 & 27 \\ 27 & 21 & 19 & 18 \\ 18 & 27 & 21 & 19 \end{pmatrix}$$

We quickly obtain: $U_\infty = \frac{1}{4}(1,1,1,1)$, meaning that no letter occurs more frequently than the other — as can be expected. The frequencies of digrams (*i.e.*, two-letters words) are slightly more complicated to compute, as the digram formed from the last letter of $\tau(\alpha)$ and the first letter of $\tau(\beta)$ is automatically a factor of $\tau(\alpha\beta)$ but is not necessarily a factor of either $\tau(\alpha)$ or $\tau(\beta)$ individually. We therefore need a new version of equation (1) that takes this fact into account.

Let us define $\Omega_2 = \{\omega_1, \ldots, \omega_r\}$, the set of factors of size two of $\mathbf{z}$. If ω is such a factor, we obtain:

$$N_\omega\left(\tau\left(x\right)\right) = \sum_{\gamma\in\mathcal{A}} N_\omega\left(\tau\left(\gamma\right)\right)\cdot N_\gamma\left(x\right) + \sum_{\omega_j\in\Omega_2}\Big[N_\omega\left(\tau\left(\omega_j\right)\right) - N_\omega\left(\tau\left(\omega_j[1]\right)\right) - N_\omega\left(\tau\left(\omega_j[2]\right)\right)\Big]\cdot N_{\omega_j}\left(x\right) \qquad (2)$$

Again, in order to obtain a system of linear relations, we define:

$$\begin{aligned}V_s &= \left(\frac{N_{\omega_i}\left(\tau^s\left(a\right)\right)}{\ell^s}\right)_{1\leq i\leq|\Omega_2|}, \\ M_1 &= \left(\frac{N_{\omega_i}\left(\tau\left(\alpha_j\right)\right)}{\ell}\right)_{1\leq i\leq|\Omega_2|,1\leq j\leq|\mathcal{A}|}, \\ M_2 &= \left(\frac{N_{\omega_i}\left(\tau\left(\omega_j\right)\right) - N_{\omega_i}\left(\tau\left(\omega_j[1]\right)\right) - N_{\omega_i}\left(\tau\left(\omega_j[2]\right)\right)}{\ell}\right)_{1\leq i,j\leq|\Omega_2|},\end{aligned}$$

and equation (2) implies:

$$V_{s+1} = M_1 \cdot U_s + M_2 \cdot V_s$$

Again, we are interested in the limit V_∞ of V_s when s goes to infinity, and this vector is a solution of the equation: $V_\infty = M_2 \cdot V_\infty + M_1 \cdot U_\infty$. For the Keränen sequence $\mathbf{k}$, where $\Omega_2 = \{ab, ac, ad, ba, bc, bd, ca, cb, cd, da, db, dc\}$, we observe that:

$$85 \cdot M_1 = \begin{pmatrix} 6 & 3 & 9 & 9 \\ 8 & 5 & 8 & 5 \\ 4 & 10 & 10 & 7 \\ 7 & 4 & 10 & 10 \\ 9 & 6 & 3 & 9 \\ 5 & 8 & 5 & 8 \\ 8 & 5 & 8 & 5 \\ 10 & 7 & 4 & 10 \\ 9 & 9 & 6 & 3 \\ 3 & 9 & 9 & 6 \\ 5 & 8 & 5 & 8 \\ 10 & 10 & 7 & 4 \end{pmatrix}$$

Because the magic string that defines the Keränen sequence begins and ends with an "a", the digram formed by the last letter of $\tau(\alpha)$ and the first letter of $\tau(\beta)$ is precisely $\alpha.\beta$. Thus, M_2 is in fact $1/85$ times the identity matrix. We thus compute V_∞, to find that:

Factor	ab	ac	ad	ba	bc	bd	ca	cb	cd	da	db	dc
Frequency	$\frac{9}{112}$	$\frac{13}{168}$	$\frac{31}{336}$	$\frac{31}{336}$	$\frac{9}{112}$	$\frac{13}{168}$	$\frac{13}{168}$	$\frac{31}{336}$	$\frac{9}{112}$	$\frac{9}{112}$	$\frac{13}{168}$	$\frac{31}{336}$

Here, a discrepancy is visible, with "ba" being nearly 15% more frequent than "ab". Computing the frequency of factors of size less than ℓ is not harder, and the reasoning for factors of size two can be used as-is. In fact, equation (2) holds even if ω is a factor of $\mathbf{z}$ of size less than ℓ. Let us define:

$$S = \left(\frac{N_\omega\left(\tau\left(\alpha_j\right)\right)}{\ell} \right)_{1 \leq j \leq |\mathcal{A}|},$$
$$T = \left(\frac{N_\omega\left(\tau\left(\omega_j\right)\right) - N_\omega\left(\tau\left(\omega_j[1]\right)\right) - N_\omega\left(\tau\left(\omega_j[2]\right)\right)}{\ell} \right)_{1 \leq j \leq |\Omega_2|}.$$

Equation (2) then brings:

$$\frac{N_\omega\left(\tau^{s+1}\left(a\right)\right)}{\ell^{s+1}} = S \cdot U_s + T \cdot V_s$$

And the frequency of ω in $\mathbf{z}$ is then $S \cdot U_\infty + T \cdot V_\infty$. The frequency of any word could be computed using this process recursively, but we will conclude here, as we have set up the machinery we need later on.

4.2 Rivest's Dithered Proposals

Keränen-DMD. In [43] Rivest suggests to directly use the Keränen sequence as a source of dithering inputs. The dithering inputs are taken from the alphabet $\mathcal{A} = \{a, b, c, d\}$, and can be encoded by two bits. The introduction of dithering thus only takes two bits from the input datapath of the compression function, which improves the hashing efficiency (compared to longer encodings of dithering inputs). We note that the Keränen sequence can be generated online, one symbol at a time, in logarithmic space and constant amortized time.

Rivest's 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. Rivest's concrete proposal, which we refer to as DMD-CP (Dithered Merkle-Damgård – Concrete Proposal) reduces the need to generate the next Keränen letter. If the message M is r blocks long, then for $1 \leq i < r$ the i-th dithering symbol has the form:

$$\left(0, \mathbf{k}\left[\left\lfloor i/2^{13} \right\rfloor\right], i \bmod 2^{13}\right) \in \{0,1\} \times \mathcal{A} \times \{0,1\}^{13}$$

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. Finally, the last dithering symbol has a different form (recall that m is the number of bits in a message block):

$$(1, |M| \bmod m) \in \{0,1\} \times \{0,1\}^{15}$$

5 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. We first introduce the adapted attack in Section 5.1, and present the novel multi-diamond construction in Section 5.2 that offers a better attack on dithered Merkle-Damgård. In section 5.3, we adapt the attack of section 2 to Keränen-DMD, obtaining second-preimages in time $732 \cdot 2^{n-\kappa} + 2^{(n+\ell)/2+2} + 2^{n-\ell}$. We then apply the extended attack to DMD-CP, obtaining second-preimages with about $2^{n-\kappa+15}$ evaluations of the compression function. We then show some examples of sequences which make the corresponding dithered constructions immune to our attack.

5.1 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, as shown in Figure 3. A word of ℓ letters is then required for building the collision tree. We also need an additional letter to connect the collision tree to the message M. This way, in order to build a collision tree of depth ℓ, we have to fix a word ω of size $\ell+1$, use $\omega[i]$ as the dithering symbol of depth i, and use the last letter of ω to realize the connection to the given message.

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 ω and $\mathbf{z}$ match. The set of positions in the message where this is possible is then given by:

$$\mathit{Range} = \left\{ i \in \mathbb{N} \;\middle|\; \big(\ell + 1 \leq i\big) \wedge \big(\mathbf{z}[i-\ell] \ldots \mathbf{z}[i] = \omega\big) \right\}.$$

The adversary tries random message blocks B, computing $f(h_\diamond, B, \omega[\ell])$, until some h_{i_0} is encountered. If $i_0 \in \mathit{Range}$, then the second-preimage attack may carry on. Otherwise, another block B needs to be found. Therefore, the goal of the adversary is to build the diamond structure with a word ω which maximizes the cardinality of *Range*.

To attain the objective of maximizing the size of the range, ω should be the *most frequent* factor of $\mathbf{z}$ (amongst all factors of the same length). Its frequency, the log of which is the *min-entropy* of $\mathbf{z}$ for words of length ℓ, is

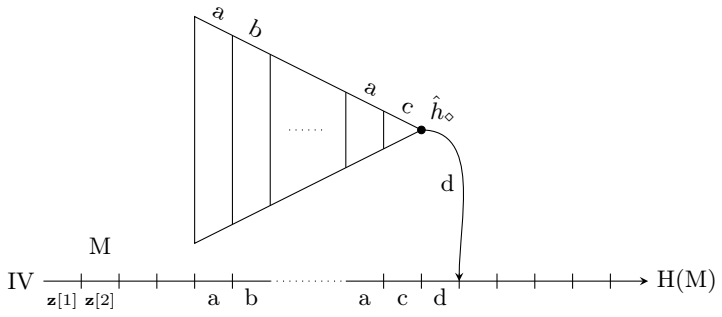


Fig. 3: A diamond built on top of a factor of the dithering sequence, connected to the message.

Algorithm 2 *Attack Algorithm for Dithered Merkle-Damgård Hash Functions*

1. *Let ω be the most frequent factor of length $\ell+1$ of $\mathbf{z}$.*
2. *Generate a collision tree of depth ℓ using the first ℓ symbols of ω as the dithering symbols in all the leaf-to-root paths. Let $\hat{h}_\diamond$ be the target value (root of the tree).*
3. *Try random message blocks B, until $f\left(\hat{h}_\diamond, B, \omega[ell]\right) = h_{i_0}$ for $i_0 \in Range$, where*

$$Range = \left\{ i \in \mathbb{N} \,\middle|\, \left(\ell+1 \leq i\right) \wedge \left(\mathbf{z}[i-\ell] \ldots \mathbf{z}[i] = \omega\right) \right\}.$$

   Let $B^{\searrow}$ be a message block satisfying this condition, i.e., $h_{i_0} = f(\hat{h}_\diamond, B^{\searrow}, \omega[\ell])$.
4. *Pick a prefix P of size $i_0 - \ell - 2$ blocks, and let h_P be the chaining value obtained after processing P by the hash function. Try random message blocks B, until $f(h_P, B) = \hat{h}_j$ for some $\hat{h}_j$ labeling a leaf of the diamond. Let $B^{\nearrow}$ denote this block, and let T be the chain of ℓ blocks traversing the diamond from $\hat{h}_j$ to $\hat{h}_\diamond$.*
5. *Form a message $M' = P||B^{\nearrow}||T||B^{\searrow}||M_{\geq i_0+1}$.*

therefore very important in computing the complexity of our attack. We denote it by $H_\infty(\mathbf{z}, \ell)$. The cost of finding the second-preimage for a given sequence $\mathbf{z}$ is

$$2^{\frac{n}{2}+\frac{\ell}{2}+2} + 2^{H_\infty(\mathbf{z},\ell+1)} \cdot 2^{n-\kappa} + 2^{n-\ell}.$$

When the computation of the exact $H_\infty(\mathbf{z}, \ell+1)$ is infeasible, we may use an upper-bound on the complexity of the attack by using the lower-bound on the frequency of *any* factor given in Section 4: in the worst case, all factors of size $\ell+1$ appear in $\mathbf{z}$ with the same frequency, and the probability that a randomly chosen factor of size $\ell+1$ in $\mathbf{z}$ is the word ω is $1/Fact_{\mathbf{z}}(\ell+1)$. This gives an upper bound on the attack's complexity:

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

A Time-Memory-Data Tradeoff Variant. As shown in Section 3, one can implement the connection into the message (Step 3 of Algorithm 2) using a time-memory-data tradeoff. It is easy to see that this attack can also

be applied here, as the dithering letter for the last block is known in advance. This allows reducing the online complexity to

$$2^{\frac{n}{2}+\frac{\ell}{2}+2} + 2^{2(n-\kappa+H_\infty(\mathbf{z},\ell+1)-t)} + 2^{n-\ell}.$$

in exchange for an additional 2^t memory and $2^{n-\kappa+H_\infty(\mathbf{z},\ell+1)}$ precomputation. As noted earlier, this may allow applying the attack at the same complexity to shorter messages, which in turn, may change the value of $H_\infty(\mathbf{z}, \ell+1)$.

5.2 Multi-Factor Diamonds

So far, we only used a single diamond, built using a single factor of the dithering sequence. As mentioned earlier, this diamond can only be used at specific locations, specified by its range (which corresponds to the set of locations of $\mathbf{z}$ where the chosen factor appears). We note that while the locations to connect into the message are determined by the dithering sequence, the complexity of connecting to the diamond structure depends (mostly) on the parameter ℓ, which can be chosen by the adversary. Hence, to make the online attack faster, we try to enlarge the range of our herding tool at the expense of a more costly precomputation and memory. We also note that this attack is useful for cases where the exact dithering sequence is not fully known in advance to the adversary, but there is a set of dithering sequences whose probabilities are sufficiently "high". Our tool of trade for this task, is the *multi-factor diamond* presented in the sequel.

Let ω_1 and ω_2 be two factors of size $\ell+2$ of the dithering sequence. Now, assume that they end with the same letter, say α. We can build two independent diamonds D_1 and D_2 using $\omega_1[1\ldots\ell]$ and $\omega_2[1\ldots\ell]$, respectively, to feed the dithering symbols. Assume that the root of D_1 (respectively, D_2) is labelled by $\hat{h}^1_\diamond$ (respectively, $\hat{h}^2_\diamond$). Now, we could find a colliding pair (m_1, m_2) such that $f(\hat{h}^1_\diamond, m_1, \omega_1[\ell+1]) = f(\hat{h}^2_\diamond, m_2, \omega_2[\ell+1])$. Let us denote by $\hat{h}_{\diamond\diamond}$ the resulting chaining value. Figure 4 illustrates our attack. Now, this last node can be connected to the message using α as the dithering symbol. We have "herded" together two diamonds with two different dithering words, and the resulting "multi-factor diamond" is more useful than any of the two diamonds separately. This claim is justified by the fact that the range of the new multi-factor diamond is the union of the two ranges of the two separate diamonds.

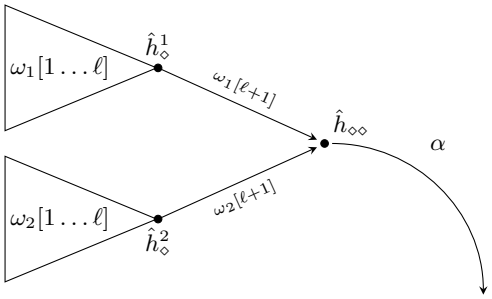


Fig. 4: A "Multi-diamond" with 2 words.

This technique can be used twice, to provide an even bigger range, as long as there four factors of $\mathbf{z}$ of size $\ell+3$ such that:

$$\begin{cases} \omega_1[\ell+3] = \omega_2[\ell+3] = \omega_3[\ell+3] = \omega_4[\ell+3] = \alpha \\ \omega_1[\ell+2] = \omega_2[\ell+2] = \beta \\ \omega_3[\ell+2] = \omega_4[\ell+2] = \gamma \end{cases}$$

A total number of 3 colliding pairs are needed to assemble the 4 diamonds together into this new multi-factor diamond.

Let us generalize this idea. We say that a set of 2^k words is *suffix-friendly* if all the words end by the same letter, and if after chopping the last letter of each word, the set can be partitioned into two suffix-friendly sets of size 2^{k-1} each. A single word is always suffix-friendly, and thus the definition is well-founded. Of course, a set of 2^k words can be suffix-friendly only if the words are all of length greater than k. If the set of factors of size $\ell+k+1$ of $\mathbf{z}$ contains a suffix-friendly subset of 2^k words, then the technique described here can be recursively applied k times.

A problem that arises for Merkle-Damgård hash functions is determining the biggest k such that a given set of words, Ω, contains a suffix-friendly subset of size 2^k. Fortunately, this task is doable in time polynomial in the sizes of Ω and $\mathcal{A}$.

Additionally, given a word ω, we define the *restriction* of a multi-factor diamond herding tree to ω by removing nodes from the original until all the paths between the leaves and the root are labelled by ω. For instance, restricting the multi-factor diamond of Figure 4 to ω_1 means keeping only the first sub-diamond and the path $\hat{h}^1_\diamond \to \hat{h}^1_{\diamond\diamond}$.

Now, assume that the set of factors of size $\ell+k+1$ of $\mathbf{z}$ contains a suffix-friendly subset of size 2^k, $\Omega = \{\omega_1, \ldots, \omega_{2^k}\}$. The multi-factor diamond formed by herding together the 2^k diamonds corresponding to the ω_i's can be used in place of any of them, as mentioned above. Therefore, its "frequency" is the sum of the frequency of the ω_i. However, once connected to the message, only its restriction to the $\ell+k+1$ letter of $\mathbf{z}$ before the connection can be used. This restriction is a diamond with 2^ℓ leaves (followed by a "useless" path of k nodes).

The cost of building a 2^k-multi-factor diamond is 2^k the time of building a diamond of size ℓ plus the cost of finding 2^k-1 additional collisions. Hence, the complexity is $2^k \cdot (2^{(n+\ell)/2+2} + 2^{n/2}) \approx 2^{k+(n+\ell)/2+2}$ compression function calls. The cost of connecting the prefix to the multi-factor diamond is still $2^{n-\ell}$ (this step is the same as in our original attack).

Lastly, the cost of connecting the multi-factor diamond to the message depends on the frequency of the factors chosen to build it, which ought to be optimized according to the actual dithering sequence. Similarly to the min-entropy, we denote by $H^k_\infty(\mathbf{z}, \ell+1)$ the min-entropy associated with a 2^k suffix-friendly set of length $\ell+1$ (i.e., the set of 2^k suffix-friendly dithering sequences of length $\ell+1$ which offers the highest probability).

The multi-factor diamond attack is demonstrated against Keränen-DMD in Section 5.3 and against Shoup's UOWHF in Section 7.3. In both cases, it is more efficient than the basic version.

5.3 Applications of the New Attacks

We now turn our attention to concrete instantiations of dithered hashing to which the attack can be applied efficiently.

Cryptanalysis of Keränen-DMD. The cost of the single-diamond attack against Keränen-DMD depends on the properties of the sequence $\mathbf{k}$ that have been outlined in Section 4. Let us emphasize again that since it has a very regular structure, $\mathbf{k}$ has an unusually low complexity, and despite being strongly repetition-free, the sequence offers an extremely weak security level against our attack. Following the ideas of section 4.1, the min-entropy of $\mathbf{k}$ for words of size $\ell \leq 85$ can be computed precisely: for $29 \leq \ell \leq 85$, the frequency of the most frequent

factor of size $\ell+1$ is $1/(4 \cdot 85) = 2^{-8.4}$ (if all the factors of length, say, 50 were equally frequent, this would have been $1/732 = 2^{-9.5}$). Therefore, $H_\infty(\mathbf{z}, \ell+1) = 8.4$, and the cost of our attack on Keränen-DMD, assuming that $29 \leq \ell \leq 85$:

$$2^{\frac{n}{2}+\frac{\ell}{2}+2} + 2^{n-\kappa+8.4} + 2^{n-\ell}.$$

If n is smaller than $3\kappa - 8.4$, the optimal value of ℓ is reached by fixing $\ell = (n-4)/3$. For n in the same order as 3κ, all the terms are about the same (for $n > 3\kappa$, the first term can be ignored). Hence, to obtain the best overall complexity (or to optimize the online complexity) we need to fix ℓ such that $2^{n-\kappa+8.4} = 2^{n-\ell}$, *i.e.*, $\ell = \kappa - 8.4$. For example, for $\kappa = 55$ the optimal value of ℓ is 46.6. The online running time (which is the majority of the cost for $n > 3\kappa$) is in this case $2^{n-46.6}$ which is much smaller than 2^n in spite of the use of dithering. For larger values of ℓ, *i.e.*, $85 \leq \ell < 128$, we empirically measured the min-entropy to be $H_\infty(\mathbf{k}, \ell+1) = 9.8$ *i.e.*, $\ell = \kappa - 9.8$ can be used when $n \approx 3\kappa$.

We also successfully applied the multi-factor diamond attack to Keränen-DMD. We determined the smallest ℓ such that the set of factors of size ℓ of the Keränen sequence $\mathbf{k}$ contains a 2^k suffix-friendly set, for various values of k:

k	$\min \ell$	$Fact_{\mathbf{z}}(\ell)$
4	4	88
5	6	188
6	27	540
7	109	1572
8	194	4256

From this table we conclude that our choice of k we will most likely be 6, or maybe 7 if κ is larger than 109 (which is the case for *e.g. SHA-512*). Choosing larger values of k would require ℓ to be larger than 194, and at the time of this writing most hash functions do not allow messages of 2^{194} blocks to be hashed. Thus, these choices would unbalance the cost of the two online connections steps.

Amongst all the possible suffix-friendly sets of size 2^6 found in the factors of size about 50 of $\mathbf{k}$, we chose one having a high frequency using a greedy algorithm making use of the ideas exposed in Section 4.1. We note that checking whether this yields *optimal* multi-factor diamonds is out of the scope of this paper. In any case, we found the frequency of our multi-factor diamond to be $2^{-3.97}$. Page size limitations prevents us from showing it, but we show a slighlty smaller multi-factor diamond of size 2^5 on fig 5.

If n is sufficiently large (for instance, $n = 256$), the offline part of the attack is still of negligible cost. Then, the minimal online complexity is obtained when $2^{n-\kappa+3.97} = 2^{n-\ell}$, *i.e.*, $\ell = \kappa - 3.97$. The complexity of the attack is then roughly $2 \cdot 2^{n-\kappa+4}$ for sufficiently large values of n. This represents a speed-up of about 21 compared to the single-diamond attack.

Cryptanalysis of DMD-CP. We now apply our attack to Rivest's concrete proposal. We first need to evaluate the complexity of its dithering sequence. Recall from Section 4.2 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 (we say that it results in the *dilution* of $\mathbf{k}$ with a 13-bit counter). The original motivation was to reduce the cost of the dithering, but it has the unintentional effect of increasing the resulting sequence complexity. It is possible to study this dilution operation generically, and to see to which extent it makes our attack more difficult.

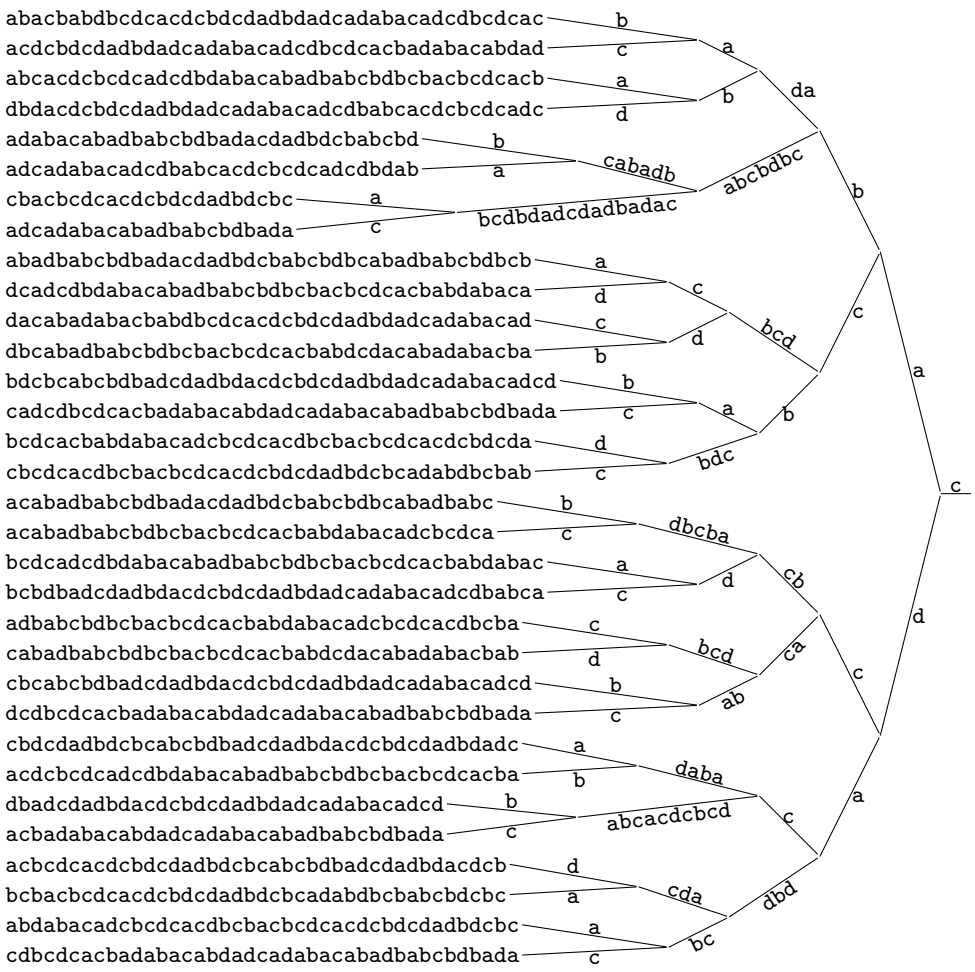


Fig. 5: A suffix-friendly set of 32 factors of size 50 from the Keränen sequence.

Lemma 2. *Let* $\mathbf{z}$ *be an arbitrary sequence over* $\mathcal{A}$*, and let* $\mathbf{d}$ *denote the sequence obtained by diluting* $\mathbf{z}$ *with a counter over* i *bits. Then for every* ℓ *not equal to 1 modulo* 2^i*, we have:*

$$\begin{aligned} Fact_{\mathbf{d}}(\ell) = \big(2^i - (\ell \mod 2^i) + 1\big) \cdot Fact_z\big(\lceil \ell \cdot 2^{-i} \rceil\big) \\ + \big((\ell \mod 2^i) - 1\big) \cdot Fact_{\mathbf{z}}\big(\lceil (\ell - 1) \cdot 2^{-i} \rceil + 1\big) \end{aligned}$$

Proof. The counter over i bits splits the diluted sequence $\mathbf{c}$ into chunks of size 2^i (a new chunk begins when the counter reaches 0). In a chunk, the letter from $\mathbf{z}$ does not change, and only the counter varies. To obtain the number of factors of size ℓ, let us slide a window of size ℓ over $\mathbf{d}$. This window overlaps at least $\lceil \ell \cdot 2^{-i} \rceil$ chunks (when the beginning of the window is aligned at the beginning of a chunk), and at most $\lceil (l-1) \cdot 2^{-i} \rceil + 1$

chunks (when the window begins just before a chunk boundary). These two numbers are equal if and only if $\ell \equiv 1 \mod 2^i$. When this case is avoided, then these two numbers are consecutive integers.

This means that by sliding this window of size ℓ over $\mathbf{d}$ we observe only factors of $\mathbf{z}$ of size $\lceil \ell \cdot 2^{-i} \rceil$ and $\lceil \ell \cdot 2^{-i} \rceil + 1$. Given a factor of size $\lceil \ell \cdot 2^{-i} \rceil$ of $\mathbf{z}$, there are $\left(2^i - (\ell \mod 2^i) + 1\right)$ positions of a window of size ℓ that allow us to observe this factor with different values of the counter. Similarly, there are $\left((\ell \mod 2^i) - 1\right)$ positions of the window that contain a given factor of $\mathbf{z}$ of size $\lceil \ell \cdot 2^{-i} \rceil + 1$. □

By taking $2 \leq \ell \leq 2^i$, we have that $\lceil \ell \cdot 2^{-i} \rceil = 1$. Therefore, only the number of factors of length 1 and 2 of $\mathbf{z}$ come into play. The formula can be further simplified into:

$$Fact_{\mathbf{d}}(\ell) = \ell \cdot \Big(Fact_{\mathbf{z}}(2) - Fact_{\mathbf{z}}(1)\Big) + (2^i + 1) \cdot Fact_{\mathbf{z}}(1) - Fact_2(\mathbf{z}).$$

For the Keränen sequence with $i = 13$, this gives: $Fact_{\mathbf{d}}(\ell) = 8 \cdot \ell + 32760$. Diluting over i bits makes the complexity 2^i times higher, but it does not change its asymptotic expression: it is still linear in ℓ, even though the constant term is bigger due to the counter. The cost of the attack is therefore:

$$2^{\frac{n}{2}+\frac{\ell}{2}+2} + (8 \cdot \ell + 32760) \cdot 2^{n-\kappa} + 2^{n-\ell}.$$

At the same time, for any $\ell \leq 2^i$, the most frequent factor of $\mathbf{d}$ is $(\alpha, 0), (\alpha, 1), \ldots, (\alpha, \ell - 1)$ when α is the most frequent letter of the Keränen sequence. However, as shown in section 4.1, all the letters have the same frequency, so most frequent factor of the diluted Keränen sequence $\mathbf{d}$ has a frequency of 2^{-15}. Hence, the cost of the above attack is:

$$2^{\frac{n}{2}+\frac{\ell}{2}+2} + 2^{n-\kappa+15} + 2^{n-\ell}.$$

This is an example where the most frequent factor has a frequency which is very close to the inverse of the number of factors (2^{-15} vs. $1/(8 \cdot \ell + 32760)$). In this specific case it may seem that the gain of using the most frequent element is small, but in some other cases, we expect much larger gains.

As before, if n is greater than 3κ (in this specific case $n \geq 3\kappa - 41$), the optimal value of ℓ is $\kappa - 15$, and the complexity of the attack is then approximately: $2 \cdot 2^{n-\kappa+15}$. For settings corresponding to SHA-1, a second preimage can be found in expected time of 2^{120} (for $78 > \ell > 40$).

5.4 Countermeasures

We just observed that the presence of a counter increases the complexity of the attack. If we simply use a counter over i bits as the dithering sequence, the number of factors of size ℓ is $Fact(\ell) = 2^i$ (as long as $i \leq \ell$). The complexity of the attack would then become: $2^{\frac{n}{2}+\frac{\ell}{2}+2} + 2^{n-\kappa+i} + 2^{n-\ell}$.

By taking $i = \kappa$, we obtain a scheme which is resistant to our attack. This is essentially the choice made by the designers of Haifa [6], or the UBI mode [19], but such a dithering sequence consumes (at least) κ bits of bandwidth.

Using a counter (*i.e.*, a big alphabet) is a simple way to obtain a dithering sequence of high complexity. Another, 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). However, in Section 6 we show how to perform some attacks on dithering sequences over small alphabet, which require a one-time heavy computation, but can then be used to find second preimages faster than exhaustive search.

There are Abelian Square-Free Sequences of Exponential Complexity. It is 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 $\{a, b, c, d\}$, and with another sequence $\mathbf{u}$ over $\{0, 1\}$ that has an exponential complexity. For example, such a sequence can be built by concatenating the binary encoding of all the consecutive integers. Then we can create a sequence $\tilde{\mathbf{z}}$ over the union alphabet $\mathcal{A} = \{a, b, c, d, 0, 1\}$ by interleaving $\mathbf{k}$ and $\mathbf{u}$: $\tilde{\mathbf{z}} = \mathbf{k}[1].\mathbf{u}[1].\mathbf{k}[2].\mathbf{u}[2]\ldots$. The resulting shuffled sequence inherits both properties: it is still abelian square-free, and has a complexity of order $\Omega\left(2^{\ell/2}\right)$.

Using this improved sequence, with $\ell = 2\kappa/3$, the total cost of the online attack is about $2^{n-2\kappa/3}$ (for $n > 8\kappa/3$). As a conclusion, we note that even with this exponentially complex dithering sequence, our attack is still more efficient than brute-force in finding second-preimages. Although it may be possible to find square-free sequences with even higher complexity, it is probably very difficult to achieve optimal protection, and the generation of the dithering sequences is likely to become more and more complex.

Pseudorandom Sequences. Another possible way to improve the resistance of Rivest'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 that the alphabet has size $\left|\mathcal{A}\right| = 2^i$. Then the expected number of ℓ-letter factors in a pseudo random word of size 2^κ is lower-bounded by: $2^{i\cdot\ell} \cdot \left(1 - \exp^{-2^{\kappa - i\cdot\ell}}\right)$ (refer to [22], theorem 2, for a proof of this claim). The total optimal cost of the online attack is then at least $2^{n-\kappa/(i+1)+2}$ and is obtained with $\ell = \kappa/(i+1)$. With 8-bit dithering symbols for $\kappa = 55$, the complexity of our attack is about 2^{n-5}, which still offers a small advantage over the generic exhaustive search.

6 Dealing with High Complexity Dithering Sequences

As discussed before, one possible solution to our proposed attacks is to use a high complexity sequence. In this section, we explore various techniques that can attack such sequences. We start with a simple generalization of our proposed attack. We then follow with two new attacks which have an expensive precomputation, in exchange for a much faster online phases: The kite generator and a variant of Dean's attack tailored to these settings.

6.1 Generalization of the Previous Attack

The main limiting factor of the previous construction is the fact that the diamond structure can be positioned only in specific locations. Once the sequence is of high enough complexity, then there are no sufficient number of "good" positions to apply the attack. To overcome this, we generate a converging tree in which each node is a $2|\mathcal{A}|$-collision. Specifically, for a pair of starting points w_0 and w_1 we find a $2|\mathcal{A}|$-collision under different dithering letters, *i.e.*, we find $m_0^1, \ldots, m_0^{|\mathcal{A}|}$ and $m_1^1, \ldots, m_1^{|\mathcal{A}|}$ such that

$$f(w_0, m_0^1, \alpha_1) = f(w_0, m_0^2, \alpha_2) = \ldots = f(w_0, m_0^{|\mathcal{A}|}, \alpha^{|\mathcal{A}|}) = f(w_1, m_1^{|\mathcal{A}|}, \alpha^{|\mathcal{A}|}) = \ldots = f(w_1, m_1^2, \alpha_2) = f(w_1, m_1^1, \alpha_1).$$

This way, we can position the diamond structure in any position, unrelated to the actual dithering sequence, as we are assured to be able to "move" from the i'th level to the $(i+1)$'th one, independently of the dithering sequence.

To build the required diamond structure we propose the following algorithm: First for each starting point (out of the 2^ℓ) find a $|\mathcal{A}|$-collision (under the different dithering letters). Now, it is possible to find collisions

between different starting points (just like in the original diamond structure, where we use a $|\mathcal{A}|$-collision rather than one message). Hence, the total number of $|\mathcal{A}|$-collisions which are needed from one specific starting point (in order to build the next layer of the collision tree) is $2^{n/2-\ell/2}$. The cost for building this number of $|\mathcal{A}|$ collisions is $2^{\frac{2|\mathcal{A}|-1}{2|\mathcal{A}|}n-\frac{\ell}{2|\mathcal{A}|}}$, or a total of $2^{\frac{2|\mathcal{A}|-1}{2|\mathcal{A}|}(n+\ell)+2}$ for the preprocessing step.

After the computation of the diamond structure (which may take more than 2^n), one can connect to any point in the message, independent of the used dithering letter. Hence, from the root of the diamond structure we try the most common dithering letter, and try to connect to all possible locations (this takes time $2^{n-\kappa+H_\infty(\mathbf{z},1)} \leq \left|\mathcal{A}\right| \cdot 2^{n-\kappa}$). Connecting from the message to the diamond structure takes $2^{n-\ell}$ as before.

The memory required for storing the diamond structure is $\mathcal{O}\left(|\mathcal{A}| \cdot 2^{\ell}\right)$. We note that the generation of the $|\mathcal{A}|$-collision can be done using the results of [24], which allow balancing between the preprocessing's time and its memory consumption.

Finally, given the huge precomputation step, it may be useful to consider a time-memory-data tradeoff for the first connection. This can be done by exploiting the $2^{n-\kappa+H_\infty(\mathbf{z},1)}$ possible targets as multiple data points. The analysis of this approach is the same as for the simple attack, and the resulting additional preprocessing is $2^{n+H_\infty(\mathbf{z},1)-\lambda}$, which along with an additional $2^{n+H_\infty(\mathbf{z},1)-2\lambda}$ memory reduces the online connection phase to $2^{n-\ell}+2^{2\lambda}$ (for $\lambda < \kappa - H_\infty(\mathbf{z},1)$).

6.2 The Kite Generator—Dealing with Small Dithering Alphabets

Even though the previous attack could handle any dithering sequence, it still relies on the ability to connect to the message. We can further reduce the online complexity (as well as the offline) by introducing a new technique, the *kite generator*. The kite generator shows that a small dithering alphabet is an inherent weakness, and after a $\mathcal{O}\left(2^n\right)$ preprocessing, second-preimages can be found for messages of length $2^l \leq 2^{n/4}$ in $\mathcal{O}\left(2^{2\cdot(n-l)/3}\right)$ time and space for any dithering sequence (even of maximal complexity). second-preimages for longer messages can be found in time $\max\left(\mathcal{O}\left(2^k\right), \mathcal{O}\left(2^{n/2}\right)\right)$ and memory $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-k}\right)$ (for k determined by the adversary).

Outline of the Attack. The kite generator uses a different approach, where the connections to and from the message are done for free, independent of the dithering sequence. In exchange, the precomputation phase is more computationally intensive, and the patch is significantly longer. In the precomputation phase the adversary builds a static data structure, the kite generator: she picks a set of $2^{n-\kappa}$ chaining values, B, that contains the IV. For each chaining value $x \in B$ and any dithering letter $\alpha \in \mathcal{A}$, the adversary finds two message blocks $m_{x,\alpha,1}$ and $m_{x,\alpha,2}$, such that $f(x, m_{x,\alpha,1}, \alpha), f(x, m_{x,\alpha,2}, \alpha) \in B$. The adversary then stores all $m_{x,\alpha,1}$ and all $m_{x,\alpha,2}$ in the data structure. Fig. 6 shows a toy kite generator.

In the online phase of the attack, given a message M, the adversary computes $h(M)$, and finds with high probability (thanks to the birthday paradox) an intermediate chaining value $\hat{h}_i \in B$ that equals to h_j obtained during the processing of M (for $n-\kappa < j < 2^\kappa$). The next step of the attack is to find a sequence of j blocks from the IV that leads to this $\hat{h}_i = h_j$. This is done in two steps. In the first step, the adversary performs a random walk in the kite generator, by just picking random $m_{x,\alpha,i}$ one after the other (according to the dithering sequence), until $\hat{h}'_{i-(n-\kappa)}$ is computed (this $\hat{h}_{i-(n-\kappa)}$ is independent of $\hat{h}_i = h_j$). At this point, the adversary stops her random walk, and computes from $\hat{h}_{i-(n-\kappa)}$ all the possible $2^{(n-\kappa)/2}$ chaining values reachable through any sequence of $m_{x,\alpha,1}$ or $m_{x,\alpha,2}$ (which agrees with the dithering sequence)—this amounts to consider all the paths starting from where the random walk stopped inside the kite generator and trying all the paths whose labels agree with the dithering sequence. Then, the adversary computes the "inverse" tree, starting from $\hat{h}_i$, and listing

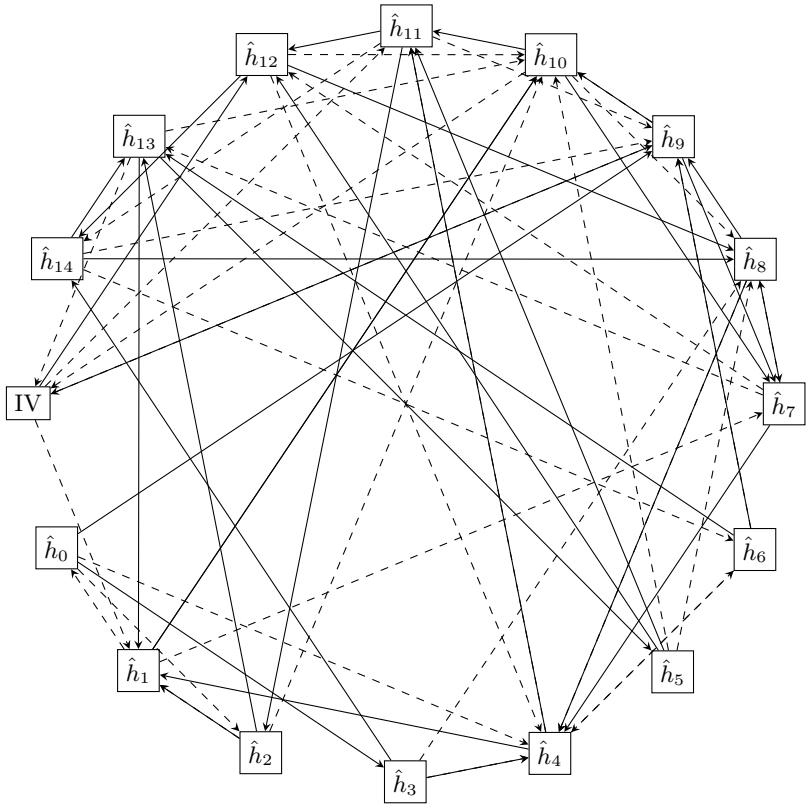


Fig. 6: A toy "Kite-Generator" with 16 nodes over a binary alphabet: Each node contains a chaining value, and each edge is labeled by a a message block and a dithering letter. Hard edges correspond to the first letter, and dashed edges to the second letter.

the expected $2^{(n-\kappa)/2}$ values[5] that may lead to it following the dithering sequence. If there is a collision between the two lists (which happens with high probability due to the birthday paradox), then the adversary just found the required path—she "connected" the IV to $\hat{h}_i$. Fig 7 illustrates the process.

The precomputation takes $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-\kappa} \cdot 2^{\kappa}\right) = \mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n}\right)$. The memory used to store the kite generator is $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-\kappa}\right)$. The online phase requires $\mathcal{O}\left(2^{\kappa}\right)$ compression function calls to compute the chaining values associated with M, and $\mathcal{O}\left(2^{(n-\kappa)/2}\right)$ memory and time for the meet-in-the-middle phase.[6] We conclude that the online time is $\max\left(\mathcal{O}\left(2^{\kappa}\right), \mathcal{O}\left(2^{(n-\kappa)/2}\right)\right)$ and the total used space is $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^{n-\kappa}\right)$. For the SHA-1 parameters of $n = 160$ and $\kappa = 55$, the time complexity of the new attack is 2^{55}, which is just the time needed to hash the original message. However, the size of the kite generator for the above parameters exceeds 2^{110}.

To some extent, the "converging" part of the kite generator can be treated as a diamond structure (for each end point, we can precompute this "structure"). Similarly, the expanding part, can be treated as the trials to connect to this diamond structure from $h'_{i-(n-\kappa)}$.

[5] See [18] for a formal justification of the size of the inverse "tree".
[6] The meet-in-the-middle can be done using memoryless variants as well.

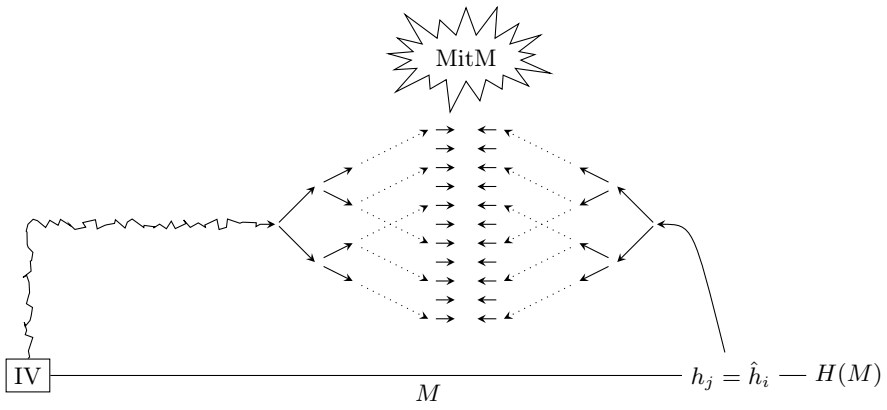


Fig. 7: A "Kite" connected to and from the message.

We note that the attack can also be applied when the IV is unknown in advance (*e.g.*, when the IV is time dependent or nonce), with essentially the same complexity. When we hash the original long message, we have to find two intermediate hash values h_i and h_j (instead of IV and h_i) which are contained in the kite 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 $\kappa < n/2$, 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 kite generator.

A "Connecting" Kite Generator In fact, the kite generator can be seen as an expandable message tolerating the dithering sequence, and we can use it in a more "traditional" way.

We first pick a special chaining value N in the kite generator. From this N we are going to connect to the message (following the approaches suggested earlier, as if N is the root of a diamond structure). Then, it is possible to connect from the IV to N inside the kite generator.

For a kite of 2^ℓ points, the offline complexity is $\mathcal{O}\left(\left|\mathcal{A}\right| \cdot 2^n\right)$, and the online complexity is $2^{n-\kappa+H_\infty(\mathbf{z},1)} + 2^\kappa + 2^{\ell/2+1}$. The memory required for the attack is $\mathcal{O}\left(2^\ell\right)$. It is easy to see that for $\kappa < n/2$, the heavy computation is the connection step, which seems a candidate for optimization.

We can also connect from N to the message using a time-memory-data tradeoff (just like in Section 3). In this case, given the $2^{\kappa-H_\infty(\mathbf{z},1)}$ targets, the precomputation is increased by $2^{n-\kappa+H_\infty(\mathbf{z},1)}$ (which is negligible with respect to the kite's precomputation). The online complexity is reduced to $2^{2(n-t-\kappa+H_\infty(\mathbf{z},1))}$ for an additional 2^t memory (as long as $2(n-t-\kappa+H_\infty(\mathbf{z},1)) \geq 2(\kappa - H_\infty(\mathbf{z},1))$, *i.e.*, $t \leq n - 2(\kappa - H_\infty(\mathbf{z},1))$). The overall online complexity is thus $2^{\ell/2+1} + 2^{2(n-t-\kappa+H_\infty(\mathbf{z},1))}$, which is lower bounded by $2^{\ell/2+1} + 2^{2(\kappa-H_\infty(\mathbf{z},1))}$.

6.3 A Variant of Dean's Attack for Small Dithering Alphabet

Given the fact that the connection into the message is the more consuming part of the attack, we now present a degenerate case of the kite generator. This construction can also be considered as an adaptation of Dean's attack to the case of small dithering alphabet.

Assume that the kite generator contains only one chaining value, namely, IV. For each dithering letter α, we find x_α such that $f(IV, x_\alpha, \alpha) = IV$. Then, we can "move" from IV to IV under any dithering letter. At

Attack	Complexity			Avg. Patch
	Offline	Online	Memory	
Adapted (Sect. 5.1)	$2^{(n+\ell)/2+2}$	$2^{n-\kappa+H_\infty(\mathbf{z},\ell+1)}+2^{n-\ell}$	$2^{\ell+1}$	$\ell+2$
Multi-Factor Diamond (Sect. 5.2)	$2^{k+(n+\ell)/2+2}$	$2^{n-\kappa+H^k_\infty(\mathbf{z},\ell+1)}+2^{n-\ell}$	$2^{k+\ell+1}$	$k+\ell+2$
Generalized (Sect. 6.1)	$2^{\frac{2\lvert\mathcal{A}\rvert-1}{2\lvert\mathcal{A}\rvert}\cdot(n+\ell)+2}$	$2^{n-\kappa+H_\infty(\mathbf{z},1)}+2^{n-\ell}$	$\lvert\mathcal{A}\rvert\cdot 2^{\ell+1}$	$\ell+2$
Kite Generator (Sect. 6.2)	$\lvert\mathcal{A}\rvert\cdot 2^n$	$2^\kappa+2^{(n-\kappa)/2+1}$	$\lvert\mathcal{A}\rvert\cdot 2^{n-\kappa+1}$	$2^{\kappa-1}$
"Connecting" Kite (Sect. 6.2)	$\lvert\mathcal{A}\rvert\cdot 2^n$	$2^\kappa+2^{n-\kappa+H_\infty(\mathbf{z},1)}+2^{\ell/2+1}$	$\lvert\mathcal{A}\rvert\cdot 2^{\ell+1}$	$2^{\kappa-1}$
"Self-loop" (Sect. 6.3)	$\lvert\mathcal{A}\rvert\cdot 2^n$	$2^{n-\kappa+H_\infty(\mathbf{z},1)}$	$\lvert A\rvert$	$2^{\kappa-1}$

$H^k_\infty(\mathbf{z},\ell+1)$ — the min-entropy of all sets of 2^k suffix-friendly dithering sequences of length $\ell+1$.

Table 3. Comparison of Long Message Second-Preimage Attacks on Dithered Hashing

this point, we connect from the IV to the message (either directly, or using time-memory-data tradeoff), and "traverse" the degenerate kite generator under the different dithering letters.

Hence, a standard implementation of this approach would require $\mathcal{O}\left(\lvert\mathcal{A}\rvert\cdot 2^n\right)$ precomputation and $2^{n-\kappa+H_\infty(\mathbf{z},1)}$ online computation (with $\lvert\mathcal{A}\rvert$ memory). A time-memory-data variant can reduce the online computation to $2^{2(n-t-\kappa+H_\infty(\mathbf{z},1))}$ in exchange for 2^t memory (as long as $t\le n-2(\kappa-H_\infty(\mathbf{z},1))$).

Table 3 compares all the techniques suggested for dithered hashing.

7 Matching the Security Bound on Shoup's UOWHF

In this section, we show that the idea of turning the herding attack into a second-preimage attack is generic enough to be applied to Shoup's Universal One-Way Hash Function (UOWHF) [46]. A UOWHF is a family of hash functions H for which any computationally bounded adversary A wins the following game with negligible probability. First, A chooses a message M, then a key K is chosen at random and given to A. The adversary wins if she generates a message $M'\neq M$ such that $H_K(M)=H_K(M')$. This security property, also known as target collision security or everywhere second preimage security [44] of a hash function, was first introduced in [40].

Bellare and Rogaway studied the construction of variable input length TCR hash functions from fixed input length TCR compression functions in [4]. They also demonstrated that the TCR property is sufficient for a number of signing applications. Shoup [46] improved on the former constructions by proposing a simpler scheme that also yields shorter keys (by a constant factor). It is a Merkle-Damgård-like mode of operation, but before every compression function evaluation in the iteration, the state is updated by XORing one out of a small set of possible masks into 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 (it just takes a few more operations).

We first briefly describe 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 (up to a logarithmic factor).

7.1 Description of Shoup's UOWHF

Shoup's construction has some similarities with Rivest's dithered hashing. It starts from a universal one way compression function f that is keyed by a key K, $f_K\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$. This compression function is then iterated, as described below, to obtain a variable input length UOWHF H_K^f.

The scheme uses a set of masks $\mu_0, \ldots, \mu_{\kappa-1}$ (where $2^\kappa - 1$ 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 K 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 $\mathcal{A} = \{0, \ldots, \kappa - 1\}$. The scheduling sequence is $\mathbf{z}[i] = \nu_2(i)$, for $1 \leq i \leq 2^\kappa$, where $\nu_2(i)$ denotes the largest integer ν such that 2^ν divides i. Let M be a message that can be split into r blocks $x_1, \ldots, x_r$ of m bits each and let h_0 be an arbitrary n-bit string. We define $h_i = f_K\left(h_{i-1} \oplus \mu_{\nu_2(i)}, x_i\right)$, and $H_K^f(M) = h_r$.

7.2 An Attack (Almost) Matching the Security Bound

In [46], Shoup proves the following security result:

Theorem 2 (Shoup, 2000, [46]). *If an adversary is able to break the target collision-resistance of H^f with probability ϵ in time T, then one can construct an adversary that breaks the target collision-resistance of f in time T, with probability $\epsilon/2^\kappa$.*

In this section we show that this bound is almost tight. First, we give an alternate definition of the dithering sequence $\mathbf{z}_{Shoup}$. In fact, the alphabet over which the sequence $\mathbf{z}_{Shoup}[i] = \nu_2(i)$ is built is not finite, as it is the set of all integers. In any case, we define:

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

As an example, we have $u_4 = 010201030102010$. The following facts about $\mathbf{z}_{Shoup}$ are easy to establish:

i) $|u_i| = 2^i - 1$
ii) The number of occurrences of u_i in u_j (with $i < j$) is 2^{j-i}.
iii) The frequency of u_i in the (infinite) sequence $\mathbf{z}$ is 2^{-i}.
iv) The frequency of a factor is the frequency of its highest letter.
v) Any factor of $\mathbf{z}_{Shoup}$ of size ℓ contains a letter greater or equal to $\lfloor \log_2(\ell) \rfloor$.

Let us consider a factor of size ℓ of $\mathbf{z}_{Shoup}$. It follows from the previous considerations that its frequency is upper-bounded by $2^{-\lfloor \log_2(\ell) \rfloor - 1}$, and that the prefix of size ℓ of $\mathbf{z}_{Shoup}$ has a greater or equal frequency. The frequency of this prefix is lower-bounded by the expression: $2^{-\lfloor \log_2(\ell) \rfloor - 1} \geq 1/(2 \cdot \ell)$.

Our attack can be applied against the TCR property of H^f as described above. Choose at random a (long) target message M. Once the key is chosen at random, build a collision tree using a prefix of $\mathbf{z}_{Shoup}$ of size ℓ, and continue as described in section 5. The cost of the attack is then:

$$T = 2^{\frac{n}{2} + \frac{\ell}{2} + 2} + 2 \cdot \ell \cdot 2^{n-\kappa} + 2^{n-\ell}.$$

This attack breaks the target collision-resistance with a constant success probability (of about 63%). Therefore, with Shoup's security reduction, one can construct an adversary against f with running time T and probability of success $0.63/2^\kappa$. If f is a black box, the best attack against f's TCR property is exhaustive search. Thus, the

best adversary in time T against f has success probability of $T/2^n$. When $n \geq 3\kappa$, $T \simeq (2\kappa+2)\cdot 2^{n-\kappa}$ (with $\ell = \kappa - 1$), and thus the best adversary running in time T has success probability $\mathcal{O}(\kappa/2^\kappa)$ when the success probability of the attack is $0.63/2^\kappa$. This implies that there is no attack better than ours by a factor greater than $\mathcal{O}(\kappa)$ or, in other words, there is only a factor $\mathcal{O}(\kappa)$ between Shoup's security proof and our attack.

We note that in this case, there is a very large gap between the frequency of the most frequent factor and the upper-bound provided by the inverse of the number of factors. Indeed, it can be seen that:

$$Fact_{u_i}(\ell) = \begin{cases} 0 & \text{if } |u_i| < \ell \\ 2^i - \ell & \text{if } |u_{i-1}| < \ell \leq |u_i| \\ \ell + Fact_{u_{i-1}}(\ell) & \text{if } |u_{i-1}| \geq \ell \end{cases}$$

And the expression of the number of factors follows:

$$Fact_{u_\kappa}(\ell) = 2^{\lceil \log_2(\ell+1)\rceil} + \left(\kappa - \lceil \log_2(\ell+1)\rceil - 1\right)\cdot \ell$$

Hence, if all of them would appear with the same probability, the time complexity of the attack would have been

$$T = 2^{\frac{n}{2}+\frac{\ell}{2}+2} + \left(2^{\lceil \log_2(\ell+1)\rceil} + \left(\kappa - \lceil \log_2(\ell+1)\rceil - 1\right)\cdot \ell\right)\cdot 2^{n-\kappa} + 2^{n-\ell},$$

which is roughly κ times bigger than the previous expression.

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

7.3 Application of the Multi-Factor Diamonds Attack

To apply the multi-factor diamond attack described in section 5.2, we need to identify a big enough suffix-friendly subset of the factors of $\mathbf{z}_{Shoup}$ of a given size, and to compute its frequency.

We choose to have end diamonds of size $\ell = 2^{2^i-1}$. Let us keep in mind that ℓ and κ must generally be of the same order to achieve the optimal attack complexity, which suggests that i should be close to $\log_2 \log_2 \kappa$.

Now, we need to identify a suffix-friendly set of factors of $\mathbf{z}_{Shoup}$ in order to build a multi-factor diamond. In fact, we focus on the factors that have u_i as a suffix. It is straightforward to check that they form a suffix-friendly set. It now remains to estimate its size and its frequency.

Lemma 3. *let Ω_j be the set of words ω of size $\ell = 2^{2^i-1}$ such that $\omega.u_i$ is a factor of u_j. Then:*

i) If $\kappa \geq 2^i$, then $|\Omega_\kappa| = \left(\kappa - 2^i + 1\right)\cdot 2^{2^i-i-1}$
ii) There are 2^{2^i-i-1} (distinct) words in Ω_κ whose frequency is 2^{-j} (with $2^i \leq j \leq \kappa$).

Proof. We first evaluate the size of Ω, and for this we define $f_i(\kappa)$, the number of factors of u_κ that can be written as $\omega.u_i$, with $|\omega| = 2^{2^i-1}$. We find:

$$|\Omega_\kappa| = \begin{cases} 0 & \text{if } 2^\kappa < 2^{2^i-1} + 2^i \\ |\Omega_{\kappa-1}| + 2^{2^i-i-1} & \text{if } 2^\kappa \geq 2^{2^i-1} + 2^i \end{cases} \quad (3)$$

The first case of this equality is rather obvious. The second case stems from the following observation: let x be a factor of u_j, for some j. Then either x is a factor of u_{j-1}, or u contains the letter "$j-1$" (both cases are

Function (n, κ)		MD5 (128,55)	SHA-1 (160,55)	SHA-256 (256,118)	SHA-512 (512,118)
Original (Sect. 7.2)	Offline:	2^{91}	2^{107}	2^{189}	2^{317}
	Online:	2^{80}	2^{112}	$2^{145.7}$	$2^{401.7}$
	Memory:	2^{50}	2^{50}	$2^{111.3}$	$2^{111.3}$
	Patch:	51	51	114	114
Multi-Factor Diamond (Sect. 7.3)	Offline:	$2^{96.5}$	$2^{112.5}$	$2^{194.5}$	$2^{322.5}$
	Online:	$2^{76.9}$	$2^{108.9}$	2^{142}	2^{398}
	Memory:	2^{59}	2^{59}	2^{123}	2^{123}
	Patch:	60	60	124	124

Table 4. Comparison of the Time Complexity of our Attacks on Shoup's UOWHF

mutually exclusive). Thus, we only need to count the numbers of factors of Ω_κ containing the letter "$\kappa - 1$" to write a recurrence relation.

If $2^\kappa \geq 2^{2^i-1} + 2^i$, then u_i appears $2^{\kappa-i}$ times in u_κ, at indices that are multiples of 2^i. The unique occurrence of the letter "$\kappa - 1$" in u_κ is at index $2^{\kappa-1} - 1$. Thus, elements of Ω_κ containing the letter "$\kappa - 1$" are present in u_κ at indices $2^{\kappa-1} - 2^{2^i-1} + \alpha \cdot 2^i$, with $0 \leq \alpha < 2^{2^i-i-1}$. Therefore, there are exactly 2^{2^i-i-1} distinct elements of Ω_κ containing "$\kappa - 1$" in u_κ (they are necessarily distinct because they all contain "$\kappa - 1$" only once and at different locations).

Now that Equation (3) is established, we can unfold the recurrence relation. We note that we have for $i \geq 1$, $\left\lceil \log_2 \left(2^{2^i-1} + 2^i\right) \right\rceil = 2^i$, and thus we obtain (assuming that $\kappa \geq 2^i$):

$$|\Omega_\kappa| = \left(\kappa - 2^i + 1\right) \cdot 2^{2^i-i-1}$$

Also, for $2^i \leq j \leq \kappa$, Ω_κ contains precisely 2^{2^i-i-1} words whose greatest letter is "$j-1$", and thus whose frequency in $\mathbf{z}_{Shoup}$ is 2^{-j}. □

By just selecting the factors of Ω_κ of the highest frequency, we would herd together $2^{2^i-i-1} = \ell / (1 + \log_2 \ell)$ diamonds, each one being of frequency $1/(2\ell)$. The frequency of the multi-factor diamond then becomes $1/(2 + 2\log_2 \ell)$.

The cost of the multi-factor diamond attack is thus roughly:

$$\frac{\ell}{1 + \log_2 \ell} \cdot \left(2^{(n+\ell)/2+2} + 2^{\frac{n}{2}}\right) + (1 + \log_2 \ell) \cdot 2^{n-\kappa+1} + 2^{n-\ell}.$$

If $n \gg 3\kappa$, the preprocessing will be negligible compared to the online time, and the cost of the attack is $\mathcal{O}(\log \kappa \cdot 2^{n-\kappa})$. Therefore, with the same proof as in the previous subsection, we can show that there is a factor $\mathcal{O}(\log \kappa)$ between Shoup's security proof and our attack. Note that, depending on the parameters, this improved version of the attack may be worse than the basic version.

We outline the complexities out our attacks (the regular and the multi-factor diamond ones) against MD5, SHA-1, SHA-256, and SHA-512 in Table 4.

8 Second-Preimage Attack with Multiple Targets

Both the older generic second-preimage results of [16, 27] 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 from the expandable message ([16, 27]). A set of 2^R messages, each of 2^κ blocks, has the same number of intermediate hash values as a single message of $2^{R+\kappa}$ blocks, and so the difficulty of finding a second-preimage for one of a set of 2^R such messages is no greater than that of finding a second-preimage for a single $2^{R+\kappa}$ 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 also linearly as long as the total number of message blocks, 2^S, satisfies $S < (n-4)/3$.

Consider for example an application which has used SHA-1 to hash 2^{30} different messages, each of 2^{20} message blocks. Finding a second-preimage for a given one of these messages using the attack of [27] requires about 2^{141} work. However, finding a second-preimage for *one* of these of these 2^{30} target messages requires 2^{111} work. (Naturally, the adversary 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 particular 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 (we note that in the case of Shoup's UOWHF and ROX, we require that the same masks were used for all the 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^{50} blocks has no practical relevance, as there are probably no applications which use messages of this length. A second-preimage attack which can be applied to a large set of messages of, say, 2^{24} blocks each, can offer a 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. Moreover, for hashes which use the same dithering sequence $\mathbf{z}$ in all invocations, this has an affect on the frequency of the most common factors (especially when the most common factor is relatively in the beginning of the dithering sequence, *e.g.*, Shoup's UOWHF with the same set of keys).

The long-message second-preimage attack on tree-based hashes offers approximately the same improvement, as the number of targets is increased. Thus, since a tree hash with an n-bit compression function output and 2^s message blocks offers a 2^{n-s+1} long-message second-preimage attack, a set of 2^r messages, each 2^s message blocks long and processed with a tree hash, will allow a second-preimage on one of those messages with about $2^{n-s-r+1}$ work.

Acknowledgments. Thanks to Lily Chen and Barbara Guttman for useful comments. Thanks to Jean-Paul Allouche, Jeffrey Shallit, and James D. Currie for pointing out the existence of abelian square-free sequences of high complexity.

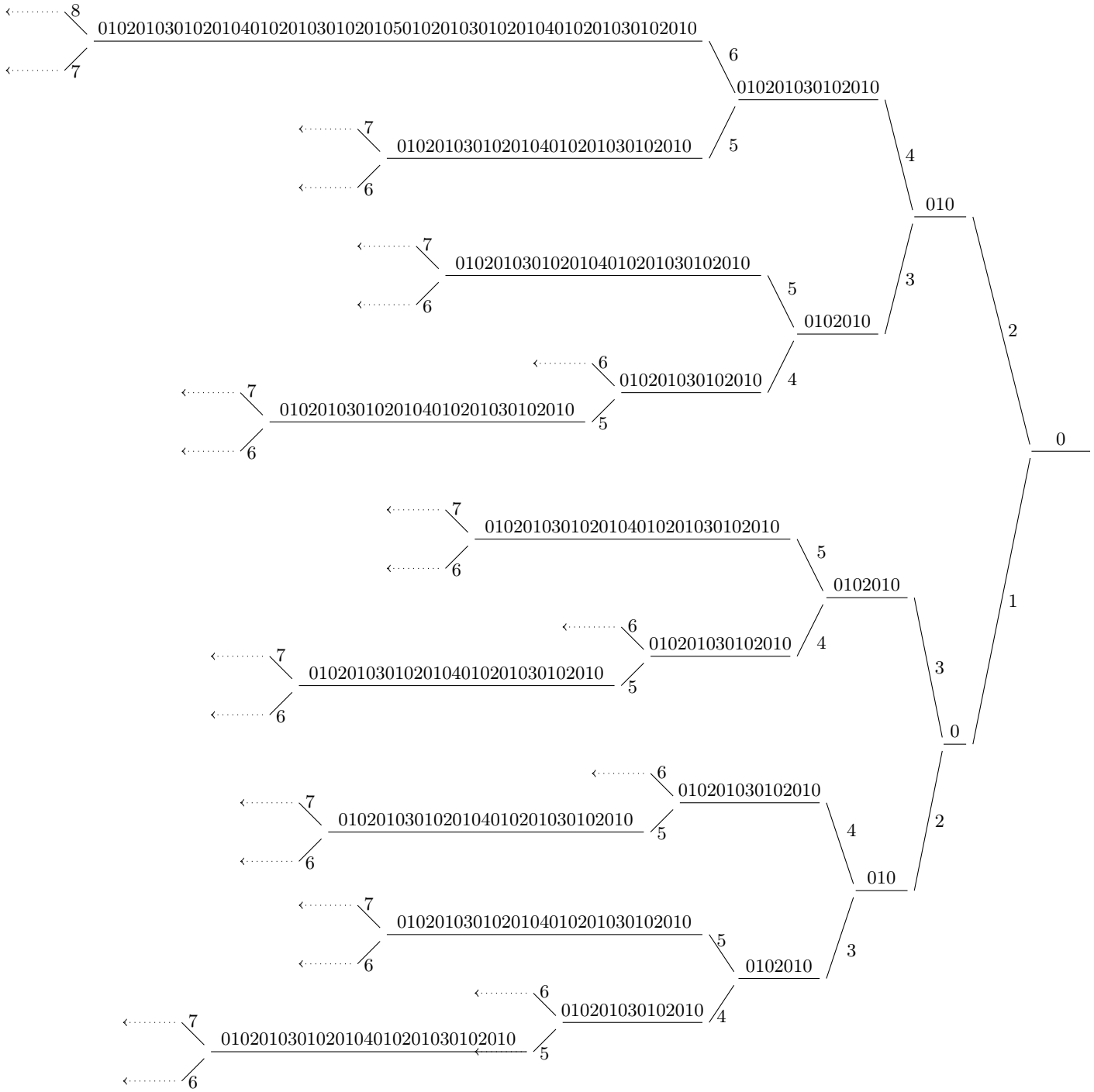
The work of the first author has been funded by a Ph.D. grant of the Flemish Research Foundation and supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy). The third author was partially supported by the France Telecome Chair and in part by ISF grant XXXXXX. This work was also partially supported by the European Commission through the IST Programme under contracts IST-2002-507932 ECRYPT and ICT-2007-216676 ECRYPT II.

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. Bellare, M., Ristenpart, T.: Multi-Property-Preserving Hash Domain Extension and the EMD Transform. [30] 299–314
4. Bellare, M., Rogaway, P.: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In Jr., B.S.K., ed.: CRYPTO. Volume 1294 of Lecture Notes in Computer Science., Springer (1997) 470–484
5. Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., Jalby, W.: Collisions of SHA-0 and Reduced SHA-1. [11] 36–57
6. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions — HAIFA. Presented at the second NIST hash workshop (August 24-25, 2006) (2006)
7. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Okamoto, T., ed.: ASIACRYPT. Volume 1976 of Lecture Notes in Computer Science., Springer (2000) 1–13
8. Brassard, G., ed.: CRYPTO '89, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. In Brassard, G., ed.: CRYPTO. Volume 435 of Lecture Notes in Computer Science., Springer (1990)
9. Cobham, A.: Uniform tag seqences. Mathematical Systems Theory **6**(3) (1972) 164–192
10. Coron, J.S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-damgård revisited: How to construct a hash function. In: CRYPTO'05. (2005) 430–448
11. 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)
12. Damgård, I.: A Design Principle for Hash Functions. [8] 416–427
13. de Cannière, C., Mendel, F., Rechberger, C.: Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Adams, C.M., Miri, A., Wiener, M.J., eds.: Selected Areas in Cryptography. Volume 4876 of Lecture Notes in Computer Science., Springer (2007) 56–73
14. de Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. [30] 1–20
15. de Cannière, C., Rechberger, C.: Preimages for Reduced SHA-0 and SHA-1. In Wagner, D., ed.: CRYPTO. Volume 5157 of Lecture Notes in Computer Science., Springer (2008) 179–202
16. Dean, R.D.: Formal Aspects of Mobile Code Security. PhD thesis, Princeton University (January 1999)
17. 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
18. Feller, W.: 12. In: An Introduction to Probability Theory and Its Applications. Volume 1. John Wiley & Sons (1971)
19. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein Hash Function Family. Submission to NIST (Round 1) (2008)
20. Halevi, S., Krawczyk, H.: Strengthening Digital Signatures Via Randomized Hashing. In Dwork, C., ed.: CRYPTO. Volume 4117 of Lecture Notes in Computer Science., Springer (2006) 41–59
21. Hellman, M.E.: A Cryptanalytic Time-Memory Trade Off. In: IEEE Transactions on Information Theory. Volume 26. (1980) 401–406
22. Janson, S., Lonardi, S., Szpankowski, W.: On average sequence complexity. Theor. Comput. Sci. **326**(1-3) (2004) 213–227
23. 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
24. Joux, A., Lucks, S.: Improved Generic Algorithms for 3-Collisions. [34] 347–363
25. 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
26. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In Vaudenay, S., ed.: EUROCRYPT. Volume 4004 of Lecture Notes in Computer Science., Springer (2006) 183–200
27. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. [11] 474–490

28. 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
29. Klima, V.: Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105 (2006) `http://eprint.iacr.org/`.
30. Lai, X., Chen, K., eds.: Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings. In Lai, X., Chen, K., eds.: ASIACRYPT. Volume 4284 of Lecture Notes in Computer Science., Springer (2006)
31. Leurent, G.: Md4 is not one-way. In Nyberg, K., ed.: FSE. Volume 5086 of Lecture Notes in Computer Science., Springer (2008) 412–428
32. Leurent, G.: Practical key-recovery attack against APOP, an MD5-based challenge-response authentication. IJACT **1**(1) (2008) 32–46
33. Lucks, S.: A Failure-Friendly Design Principle for Hash Functions. In Roy, B.K., ed.: ASIACRYPT. Volume 3788 of Lecture Notes in Computer Science., Springer (2005) 474–494
34. Matsui, M., ed.: Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009)
35. Matusiewicz, K., Naya-Plasencia, M., Nikolic, I., Sasaki, Y., Schläffer, M.: Rebound attack on the full lane compression function. [34] 106–125
36. Mendel, F., Peyrin, T., Rechberger, C., Schläffer, M.: Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Jr., M.J.J., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography. Volume 5867 of Lecture Notes in Computer Science., Springer (2009) 16–35
37. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman, O., ed.: FSE. Volume 5665 of Lecture Notes in Computer Science., Springer (2009) 260–276
38. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography
39. Merkle, R.C.: One Way Hash Functions and DES. [8] 428–446
40. Naor, M., Yung, M.: Universal One-Way Hash Functions and their Cryptographic Applications. In: STOC, ACM (1989) 33–43
41. 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
42. Pleasants, P.A.: Non-repetitive sequences. Mat. Proc. Camb. Phil. Soc. **68** (1970) 267–274
43. Rivest, R.L.: Abelian Square-Free Dithering for Iterated Hash Functions. Presented at ECRYPT Hash Function Workshop, June 21, 2005, Krakow, and at the Cryptographic Hash workshop, November 1, 2005, Gaithersburg, Maryland (2005)
44. Rogaway, P., Shrimpton, T.: Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance. In Roy, B.K., Meier, W., eds.: FSE. Volume 3017 of Lecture Notes in Computer Science., Springer (2004) 371–388
45. Sasaki, Y., Aoki, K.: Finding preimages in full md5 faster than exhaustive search. In Joux, A., ed.: EUROCRYPT. Volume 5479 of Lecture Notes in Computer Science., Springer (2009) 134–152
46. Shoup, V.: A Composition Theorem for Universal One-Way Hash Functions. In Preneel, B., ed.: EUROCRYPT'00. Volume 1807 of Lecture Notes in Computer Science., Springer (2000) 445–452
47. 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)
48. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. [11] 1–18
49. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. [47] 17–36
50. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. [11] 19–35
51. Wang, X., Yu, H., Yin, Y.L.: Efficient Collision Search Attacks on SHA-0. [47] 1–16

A A Suffix-Friendly Set for z_{Shoup}



The numbers mentioned in the figure refer to the masks in use (i.e., 0 corresponds to μ_0 and 0102 corresponds to four invocations of the compression function using $\mu_0, \mu_1, \mu_0, \mu_2$ as masks (in that order)).

Fig. 8: A suffix-friendly set of 2^k factors of for $\mathbf{z}_{Shoup}$.