Herding, Second Preimage and Trojan Message Attacks Beyond Merkle-Damgård

Elena Andreeva[1], Charles Bouillaguet[2], Orr Dunkelman[2], and John Kelsey[3]

[1] ESAT/SCD — COSIC, Dept. of Electrical Engineering,
Katholieke Universiteit Leuven and IBBT
elena.andreeva@esat.kuleuven.be
[2] Ecole Normale Supérieure
{charles.bouillaguet, orr.dunkelman}@ens.fr
[3] National Institute of Standards and Technology
john.kelsey@nist.gov

Abstract. In this paper we present new attack techniques to analyze the structure of hash functions that are not based on the classical Merkle-Damgård construction. We extend the herding attack to concatenated hashes, and to certain hash functions that process each message block several times. Using this technique, we show a second preimage attack on the folklore "hash-twice" construction which process two concatenated copies of the message. We follow with showing how to apply the herding attack to tree hashes. Finally, we present a new type of attack — the trojan message attack, which allows for producing second preimages of unknown messages (from a small known space) when they are appended with a fixed suffix.
Key words: Herding attack, Second preimage attack, Trojan message attack, Zipper hash, Concatenated hash, Tree hash.

1 Introduction

The works of Dean [8] showed that fixed points of the compression function can be transformed into a long message second preimage attack on the Merkle-Damgård functions in $\mathcal{O}\left(2^{n/2}\right)$ time (n is the size in bits of the chaining and digest values). Later, the seminal work by Joux [10] suggested a new method to efficiently construct *multicollisions*, by turning ℓ pairs of colliding message blocks into 2^ℓ colliding messages. [1] In [12], Kelsey and Schneier applied the multicollision ideas of Joux to Dean's attack, and eliminated the need for finding fixed points in the compression function by building *expandable messages*, which are a set of 2^ℓ colliding messages each of a distinct length.

Another result in the same line of research is the *herding attack* by Kelsey and Kohno in [11]. The attack is a chosen-target prefix attack, *i.e.*, the adversary commits to a digest value h and is then presented with a challenge prefix P. Now, the adversary efficiently computes a suffix S, such that $H(P||S) = h$.

[1] We note that in [6] the same basic idea was used for a dedicated preimage attack.

The underlying technique uses a *diamond structure*, a precomputed data structure, which allows 2^ℓ sequences of message blocks to iteratively converge to the same final digest value. The latter result, together with the long message second preimage attack, was used in [1] to exhibit a new type of second preimage attack that allows to construct second preimages differing from the original messages only by a small number of message blocks.

We note that the work of Joux in [10] also explores concatenated hashing, *i.e.*, the hash function $H(M) = H_1(M)||H_2(M)$. It appears that if one of the underlying hash functions is iterative, then the task of finding a collision (resp., a preimage) in $H(M)$ is only polynomially harder than finding a collision (resp., a preimage) in any of the $H_i(M)$.

1.1 Our Contributions

Our results may be summarized as follows: All examples assume 128-bit hashes, and set the work for precomputation equal to the online work of the attack for concreteness.

Applying herding attacks to more constructions. In this paper, herding attacks are applied to four non-Merkle-Damgård hash constructions. For reference, an n bit Merkle-Damgård hash can be herded with a precomputation of about $2^{\frac{n+\ell}{2}+2}$ operations, and an online cost of about $2^{n-\ell}$ operations. An Merkle-Damgård hash with a 128-bit state, with the precomputed and online work set equal, can be herded for about 2^{88} operations.

Applying long message second preimage attacks to new constructions In this paper, long-message second preimage attacks are applied to two new constructions. For reference, a long-message second preimage attack on an Merkle-Damgård hash with 128 bit state and a 2^{50} block target message requires about 2^{78} work.

Trojan Message Attacks This paper also introduces a new kind of attack, called the trojan message attack. This involves an attacker producing a kind of poisoned suffix S (the "Trojan message") such that, when the victim prepends one of a constrained set of possible prefixes to it, producing the message $P||S$, the attacker can produce a second preimage for that message. The attack comes in a less powerful form and a more powerful form, as is described in Section 9. Note that for both forms of the attack, the number of message blocks in the Trojan message is at least as large as the number of possible prefixes.

1.2 Organization of the Paper

Section 2 outlines the definitions used in the paper. Prior work is surveyed in Section 3. In Section 4 we introduce the diamond structures on κ pipes and use

Concatenated Hashes: *(κ concatenated hashes, each n bits wide, using a 2^ℓ-wide diamond structure.)*	
Precomputation:	$\left(n - \ell + \frac{n\ell}{2}\right) \cdot (\kappa - 1) \cdot (n/2)^{\kappa-2} \cdot 2^{n/2} + (2 \cdot \kappa - 1) \cdot 2^{\frac{n+\ell}{2}+2}$
Online:	$2^{n-\ell} \cdot [1 + 2 \cdot (\kappa - 1)] = (2\kappa - 1) \cdot 2^{n-\ell}$
Example:	*(Two 128 bit hashes concatenated)* 2^{94} work total.

Hash Twice: *(n bit hash state, using a 2^ℓ-wide diamond structure.)*	
Precomputation:	$3 \cdot 2^{(n+\ell)/2+2}$
Online:	$3 \cdot 2^{n-\ell}$
Example:	*(128-bit wide hash)* $3 \cdot 2^{88}$ work.

Zipper Hash: *(n bit hash state, using a 2^ℓ-wide diamond structure.)*	
Precomputation:	$2 \cdot 2^{(n+\ell)/2+2} + (n - \ell + \frac{n\ell}{2}) \cdot 2^{n/2}$
Online:	$2 \cdot 2^{n-\ell}$
Example:	*(128-bit hash)* $3 \cdot 2^{88}$ work.

Tree Hash: *(n bit hash state, binary hash tree with $2^{\ell+1}$ message blocks)* *(Note: Width of diamond is limited by message length.)*	
Precomputation:	$15.08 \cdot 2^{\frac{n+\ell}{2}}$
Online:	$2^{n-(\ell-1)}$
Example:	*(128 bit binary tree hash, 2^{30} message blocks.)* 2^{99} work total.

Hash Twice: *(n bit hash, using a 2^ℓ-wide diamond structure, 2^κ block target message.)*	
Precomputation:	$2^{\frac{n-\ell}{2}+3}$
Online:	$2^{n-\kappa}$
Example:	*(128 bit hash, 2^{50} block message)* about 2^{88} work total.

Tree Hash: *(n bit hash, binary hash tree, 2^κ block target message.)*	
Online:	$2^{n-\kappa}$
Example:	*(128 bit hash, 2^{50} block message.)* 2^{70} work total.

Less Powerful Version: *(n bit hash, N possible prefixes)* *(Second preimage changes only small part of S)*	
Precomputation:	$N \cdot 2^{n/2}$
Online:	Negligible
Example:	*(128 bit hash, 1024 possible prefixes)* about 2^{74} work total.

More Powerful Version: *(n bit hash, N possible prefixes)* *(Second preimage gives attacker full choice of prefix.)*	
Precomputation:	$2^{\frac{n+\ell}{2}+2} + N \cdot 2^{n/2}$
Online:	$2^{n-\ell}$
Example:	*(128 bit hash, 1024 possible prefixes)* 2^{88} work.

them to apply the herding attack to concatenated hashes. We use the same ideas to present herding attacks on the hash-twice and zipper hash in Section 5. These herding attacks are used in Section 6 to present a second preimage attacks on hash-twice, The new results on herding tree hashes are presented in Section 7, and in Section 8 we present second preimage attacks on tree hashes. We follow to introduce the new trojan message attack, in Section 9 and conclude with Section 10.

2 Background

General notation. Let n be a positive integer, then $\{0,1\}^n$ denotes the set of all bitstrings of length n, and $\{0,1\}^*$ be the set of all bitstrings. If x, y are strings, then $x\|y$ is the concatenation of x and y. We denote by $|x|$ the length of the bitstring x, but in some places we use $|M|$ to denote the length of the message M in blocks. For $M = m_1\|m_2\|\ldots\|m_L$ we define $\widetilde{M} = m_L\|m_{L-1}\|\ldots\|m_1$.

Let $f : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$ be a compression function taking as inputs bitstrings of length n and b, respectively. Unless stated explicitly we denote the n-bit input values by h (chaining values) and the b-bit values by m (message blocks). A function $H : \{0,1\}^* \to \{0,1\}^n$ built on top of some fixed compression function f is denoted by H^f. To indicate the use of fixed initialization vectors for some hash functions, with $H^f(IV, M)$ we explicitly denote the result of evaluating H^f on inputs the message M and the initialization vector IV.

Merkle-Damgård hash function. $\mathcal{MDH}^f : \{0,1\}^* \to \{0,1\}^n$ takes a message $M \in \{0,1\}^*$ as an input to return a digest value of length n bits. Given a fixed initialization vector IV and a padding function $\mathrm{pad}_{\mathrm{MD}}$ that appends to M fixed bits (a single 1 bit and sufficiently many 0 bits) together with the message length encoding of M to obtain a message multiple of the blocksize b, the $\mathcal{MDH}^f$ function is defined as:

1. $m_1, \ldots, m_L \leftarrow \mathrm{pad}_{\mathrm{MD}}(M)$.
2. $h_0 = IV$.
3. For $i = 1$ to L compute $h_i = f(h_{i-1}, m_i)$.
4. $\mathcal{MDH}^f(M) \triangleq h_L$.

Often in the sequel we consider chaining values obtained by hashing a given prefix of the message. When P is a message whose size is a multiple of b bits, we denote by $f^*(P)$ the chaining value resulting from the Merkle-Damgård iteration of f without the padding scheme being applied.

Concatenated hash. $\mathcal{CH}$ of $\kappa \geq 2$ pipes is defined as:

$$\mathcal{CH}(M) = H^{f_1}(IV_1, M) \,||\, H^{f_2}(IV_2, M) \,||\, \ldots \,||\, H^{f_\kappa}(IV_\kappa, M).$$

"Hash-Twice". This is a folklore hashing mode of operation that hashes two consecutive copies of the (padded) message. Formally, it is defined by:

$$\mathcal{HT}(M) \triangleq H^f(H^f(IV, M), M).$$

ZIPPER HASH. It is proposed in [13] and is proven indifferentiable from a random oracle up to the birthday bound even if the compression functions in use, f_1 and f_2, are weak (*i.e.*, they can be inverted and collisions can be found efficiently). The zipper hash $\mathcal{ZH}$ is defined as:

$$\mathcal{ZH}(M) \triangleq H^{f_2}\left(H^{f_1}(IV, M), \widetilde{M}\right).$$

Throughout the paper, we assume that all H^{f_i} are applications of the Merkle-Damgård mode of operation and thus the respective padding $\mathrm{pad}_{\mathrm{MD}}$ is present in the syntax of M, and also $\widetilde{M}$ in the case of Zipper hash.

TREE HASH. The first suggested tree hash construction dates back to [14]. Let $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a compression function and $\mathrm{pad}_{\mathrm{TH}}$ be a padding function that appends a single 1 bit and as many 0 bits as needed to the message M to obtain $\mathrm{pad}_{\mathrm{TH}}(M) = m_1 \| m_2 \| \ldots \| m_L$, where $|m_i| = n$, $L = 2^d$ for $d = \lceil \log_2(|M| + 1) \rceil$. Then, the tree hash function $\mathcal{Tree}$ is defined as:

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

3 Existing Attack Techniques

3.1 Herding attack.

The herding attack is a chosen-target preimage attack on Merkle-Damgård constructions [11]. In the attack, an adversary commits to a public digest value h_T. After the commitment phase, the adversary is challenged with a prefix P which she has no control over, and she is to produce a suffix S for which $h_T = H^f(P \| S)$. Of course, h_T is specifically chosen after a precomputation phase by the adversary. The main idea behind this attack is to store 2^ℓ possible chaining values $D = \{h_i\}$ from which the adversary knows how to reach h_T. To construct this data structure, which is in fact a tree, the adversary picks about $2^{n/2-\ell/2+1/2}$ single-block messages m_j, and evaluates $f(h_i, m_j)$ for all i and j. Due to the large number of values, it is expected that collisions occur, and it is expected that the adversary knows for all 2^ℓ values of h_i a corresponding message block $m_{\alpha(i)}$ such that the set $\left\{f\left(h_i, m_{\alpha(i)}\right)\right\}$ contains only $2^{\ell-1}$ distinct values. The process is then repeated $\ell - 1$ more times until a final digest value is found. These values (and the corresponding message blocks) are called a diamond structure, as presented in Figure 1.

In the online phase of the attack, the adversary tries at random message blocks m^* until $f^*(P||m^*) \in D$. Once such a value is found, it is possible to follow the path connecting this value to the committed hash (which is at the "root" of the diamond) and produce the required suffix S.
The total time complexity of the attack is about $2^{n/2+\ell/2+2}$ offline compression function evaluations, and $2^{n-\ell}$ online compression function evaluations.

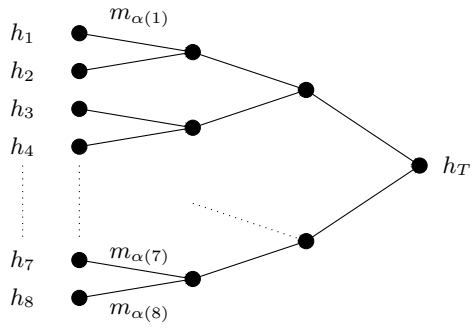


Fig. 1. A Diamond Structure

3.2 Collisions on Concatenated Hashes.

We describe the collision attack of [10] against the concatenated hash $\mathcal{CH}$ with two pipes. Starting from two fixed chaining values IV_1 and IV_2 in the two pipes, the adversary first finds a $2^{n/2}$-multicollision for the first function f_1. The adversary then evaluates f_2 on the $2^{n/2}$ messages of the multicollision, all yielding the same chaining value for f_1, while yielding a set of $2^{n/2}$ chaining values for f_2, as shown in figure 2. The adversary then looks for the expected collision in this set. To construct a 2^ℓ-*multicollision* on the two pipes, just replay Joux's attack using the two-pipe collision finding algorithm described above ℓ times.

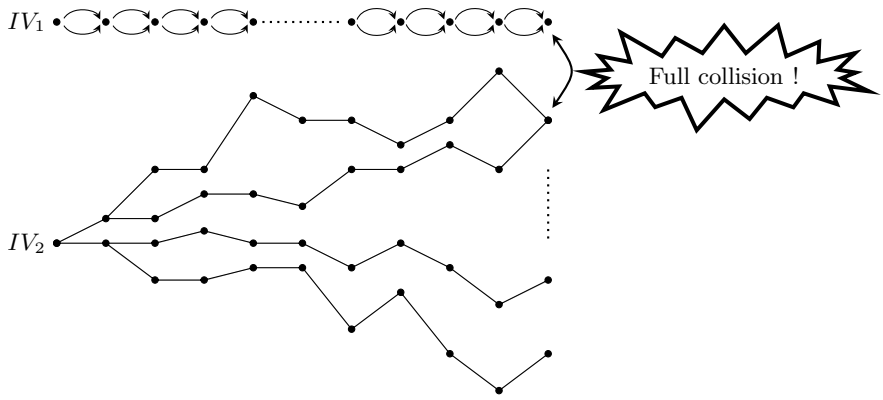


Fig. 2. Joux's attack against concatenated hashes

Joux also shows that this idea can be extended to find (multi)collisions in the concatenation of κ hash functions. To build a collision on κ parallel pipes, the adversary proceeds inductively: first construct a $2^{n/2}$-multicollision on the first $\kappa - 1$ pipes and hash the $2^{n/2}$ messages in the last pipe. Then, by the birthday bound, a collision is expected amongst the set of $2^{n/2}$ values generated in the last

pipe. This collision is present in all the previous $\kappa - 1$ pipes, and hence results in a full collision on all the κ pipes.

The cost of building a collision on κ pipes is the cost of building the multi-collision, plus the cost of hashing the $2^{n/2}$ messages of length $(n/2)^{\kappa-1}$. Solving the recurrence yields a time complexity of $\kappa \cdot \left(\frac{n}{2}\right)^{\kappa-1} \cdot 2^{\frac{n}{2}}$ compression function calls. More generally, the complexity of building a 2^{ℓ}-multicollision on κ pipes is exactly ℓ times the preceding expression, or $\ell \cdot \kappa \cdot \left(\frac{n}{2}\right)^{\kappa-1} \cdot 2^{\frac{n}{2}}$.

4 Herding Attack on Concatenated Hashes

We start by showing how to adapt the herding attack to concatenated hashes. The main idea behind the new attack is to construct *multi-pipe diamonds*, which can be done on top of multicollisions. We recall that a multicollision on $(\kappa - 1)$ pipes can be used to construct a collision on κ pipes. In the same vein, we succeed in herding κ pipes by building a κ-pipe diamond using a $(\kappa - 1)$-pipe diamond and $(\kappa - 1)$-pipe multicollision.

Assume that the adversary succeeded in herding $\kappa - 1$ pipes. Then, she faces the problem of herding the last pipe. Now, if the adversary tries to connect in the κ-th pipe with a random block, she is very likely to lose the control over the previous pipes. However, if she uses a "block" which is part of a multicollision on the first $\kappa - 1$ pipes, she still maintains the control over the previous pipes, while offering enough freedom for herding the last pipe.

4.1 Precomputation Phase

In the precomputation phase, the adversary starts with the $(\kappa - 1)$-diamond which is already known. The first step is the randomization step: given the concatenated chaining value the adversary constructs a $2^{n-\ell}$-multicollision on the first $\kappa - 1$ pipes. Let the resulting chaining value be $(h^1, h^2, \ldots, h^{\kappa-1})$.

The second step is the actual diamond construction. The adversary picks at random 2^{ℓ} values for $D_{\kappa} = \{h_i^{\kappa}\}$. Then, she generates a set of further (in addition to the preceding $2^{n-\ell}$ multicollisions) $2^{n/2}$-multicollisions[2] on the first $\kappa-1$ pipes, starting from $(h^1, h^2, \ldots, h^{\kappa-1})$. For each possible message in the multicollision, and any starting point $(h^1, h^2, \ldots, h^{\kappa-1}, h_i^{\kappa})$, the adversary computes the new chaining values, expecting to reach enough collisions, such that for any h_i^{κ}, there exists a "message" m_i (*i.e.*, a sequence of message blocks in the multicollision) where $\#\left\{f_{\kappa}{}^{*}\left(h_i^{\kappa}, m_i\right)\right\} = 2^{\ell-1}$. After this step, the same process is repeated. Figure 3 depicts the process for $\kappa = 2$.

The running time is dominated by the generation of the last diamond structure. First, we need to generate $2^{n-\ell+\frac{n\ell}{2}}$-multicollisions on $\kappa - 1$ pipes, which

[2] We note that fewer multicollisions are needed (herding the first layers takes less messages). However, for the ease of description we shall assume all layers of the diamond structure require the same number of multicollisions. Hence, the total of $2^{n\ell/2}$-multicollisions, can be reduced to $2^{\ell(n+1-\ell)/2}$-multicollisions.

takes $(n-\ell+\frac{n\ell}{2})\cdot(\kappa-1)\cdot(n/2)^{\kappa-2}\cdot 2^{n/2}$ compression function calls. Then, we need to "hash" 2^ℓ values under $2^{\frac{n-\ell+1}{2}}$ message sequences (for the last layer of the diamond structure). While at a first glance it may seem that we need a very long time for each message sequence, it can be done efficiently if we take into consideration the fact that there is no need to recompute all the chaining values only if the last block was changed. Hence, the actual time required to construct the diamond structure is $2\cdot 2^{\frac{n+\ell}{2}+2}$ (twice the time needed for a classic diamond structure). In total, the preprocessing takes

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

One may ask what is the reason for the randomization step. As demonstrated in the online phase of the attack, the need arises from the fact that herding the values in the first $\kappa-1$ pipes fixes the value in the κ-th pipe. Hence, we need enough "freedom" to randomize this chaining value, without affecting the already solved pipes.

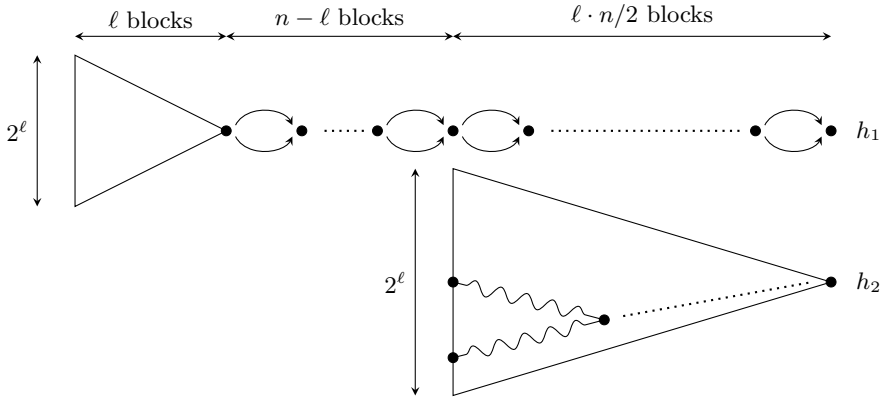


Fig. 3. Diamond Structure on two pipes

4.2 Online Phase

Given a precomputed κ-diamond structure, it is possible to apply the herding attack to κ concatenated hash functions. The adversary is given a prefix P, and tries various message blocks m^* until ${f_1}^*(P||m^*)$ gives one of the 2^ℓ values in D_1 of the diamond structure on the first pipe. Then, the adversary traverses the first diamond structure to its root, finding the first part of the suffix S_1 (so far all computations are done in the first pipe). At this point, the adversary starts computing ${f_2}^*(P||m^*||S_1)$, and for all $2^{n-\ell}$ paths of the multicollision in the randomization path, until one of them hits one of the 2^ℓ values in D_2. At this point, the adversary can use the paths inside this second diamond (built upon

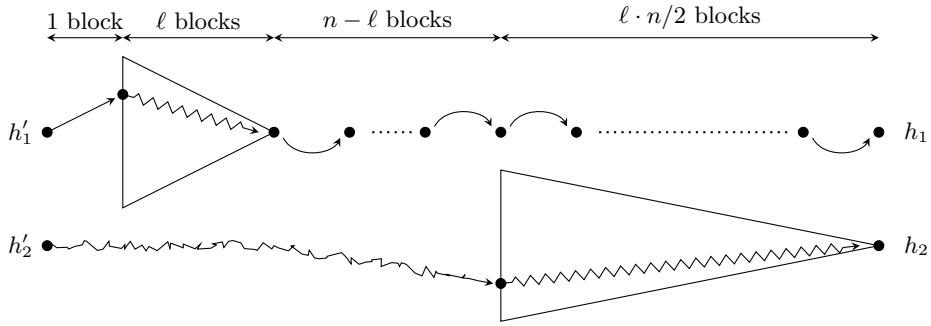


Fig. 4. The Online Phase of the Herding Attack for $\kappa = 2$

a multicollision). This process can start again (with a randomization part, and traversing the diamond structure) until all κ pipes were herded correctly. We outline the process for $\kappa = 2$ in Figure 4.

We note that once a pipe is herded, there is no longer a need to compute it (as the multicollision predicts its value), and then it is possible to start analyzing the next pipe. In each new pipe, we need to evaluate $2^{n-\ell}$ "messages" (for all pipes but the first one, these messages are multicollisions on the previous pipes), each takes on average (in an efficient implementation) two compression function calls (besides the first layer). Hence, the online time complexity of the attack is

$$2^{n-\ell} \cdot [1 + 2 \cdot (\kappa - 1)] = (2\kappa - 1) \cdot 2^{n-\ell}$$

compression function calls.

5 Herding Beyond Merkle-Damgård

In this section we show that the previous technique can be applied to two other hash constructions which previously appeared to be immune to herding attacks — the Hash Twice construction and Zipper Hash. Both attacks make use of the two-pipe diamond structure described above.

5.1 Herding the Hash-Twice Function

It follows from the very general result of [9, 15] that it is possible to build Joux-style multicollisions efficiently on the hash-twice construction. In this section, we extend their results by describing a herding attack against the hash-twice construction. This attack can then be adapted into a full second-preimage attack, following the ideas of [1, 12] (as described in Section 6).

Because each message block enters the hashing process twice, choosing a message block in the second pass may change not only the chaining value going *out* but also the chaining value going *into* the second pass. Choices of the message

intended to affect the second pass must thus be done in a way that does not randomize the result of the first pass.

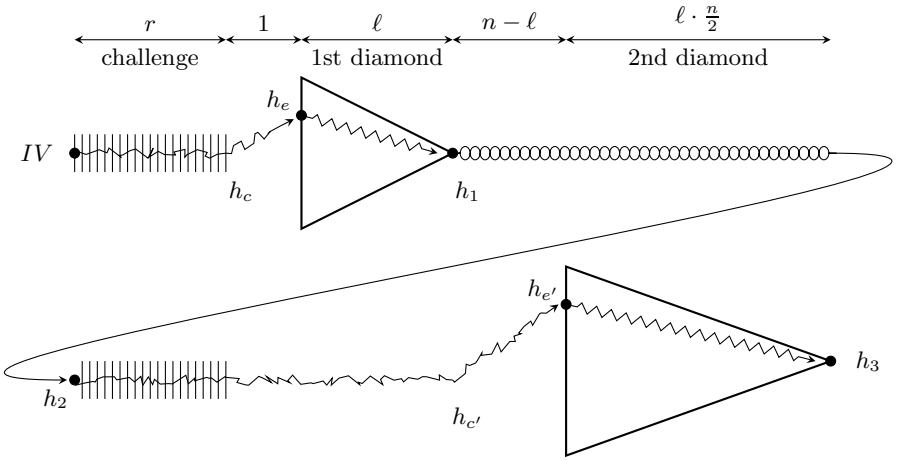


Fig. 5. Herding the Hash-Twice construction

Apart from this technicality, the attack is essentially the same as the one against the concatenation of two hash functions, as shown in figure 5. The adversary commits to h_3, and is then being challenged with an unknown prefix P. Hashing the prefix yields a chaining value h_c. Starting from h_c, she chooses a message block m^* connecting to a chaining value h_e which is one of the starting points of the first diamond, then a path S_1 inside it yields the chaining value h_1 on the first pass, from which we traverse a precomputed $2^{n-\ell+n\ell/2}$-multicollision, producing h_2 as the input chaining value to the second pass. Starting from h_2, the challenge prefix P leads to a random chaining value $h_{c'}$ in the second pass. Then, the second pass can be herded without losing control of the chaining value in the first pipe thanks to the diamond built on top of a multicollision. Amongst the $2^{n-\ell}$ messages in the multicollision following the first diamond, we expect one to connect to the chaining value $h_{e'}$ in the starting points of the second diamond. We can then follow a path inside the second diamond, which is also a path in the multicollision of the first pipe, that yields the chaining value at the root of the second diamond, namely h_3.

The offline complexity of the attack is the time required for generating a diamond structure of 2^ℓ starting points (which takes $2^{(n+\ell)/2+2}$), finding $(n-\ell)+n\cdot\ell/2$ collisions (which takes $[(n-\ell)+n\cdot\ell/2]\cdot 2^{n/2}$), and constructing a two-pipe diamond (which takes $2\cdot 2^{(n+\ell)/2+2}$). The total offline complexity is therefore $3\cdot 2^{(n+\ell)/2+2}$.

The online complexity is composed of finding two connecting "messages". The first search takes $2^{n-\ell}$, while the second takes $2\cdot 2^{n-\ell}$, or a total of $3\cdot 2^{n-\ell}$.

Attacks on Hash-Thrice. It is relatively clear that the attack can be generalized to the case where the message is hashed three or more times (by using multicollisions on 3 pipes, or the respective number of passes). The complexity of the attack becomes polynomially higher, though.

5.2 Herding the Zipper Hash Function

It is also possible to mount a modified herding attack against the zipper-hash. The regular herding attack is not feasible, because the last message block going into the compression function is the first message block of the challenge. Therefore, an adversary who is capable of doing the herding attack can be used to invert the compression function. We therefore consider a variant of the herding attack where the challenge is placed at the end: the adversary commits to a hash value h_T, then she is challenged with a suffix S, and has to produce a prefix P such that $\mathcal{ZH}(P \,||\, S) = h_T$.

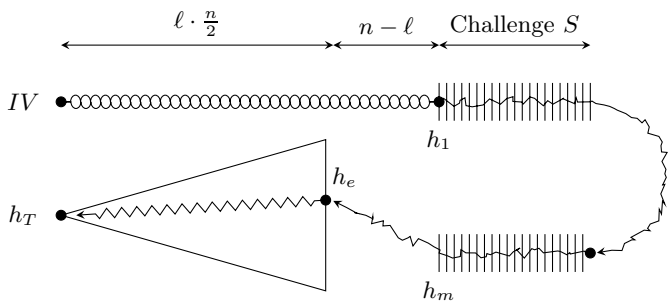


Fig. 6. Herding the Zipper Hash

The attack is relatively similar to the hash-twice case. The offline part is as follows:

1. Starting from the IV, build a $2^{n\ell/2+n-\ell}$-multicollision that yields a chaining value h_1.
2. Build a diamond structure on top of the *reversed* multicollision (*i.e.*, where the order of colliding messages in the multicollision is reversed). The chaining value at the root of the second diamond is h_T.
3. Commit to h_T.

And the online part:

1. Given a challenge suffix S, compute the chaining value after the two copies of the challenge: $h_m = {f_2}^* \left({f_1}^* \left(h_1, S \right), \widetilde{S} \right)$.
2. From h_m, find a connecting path in the part of (reversed) multicollision that is just before the diamond (in the second run) yielding a chaining value

$h_e \in D_1$ of the diamond structure. Then find a path inside the (reversed) diamond structure towards the committed hash h_T.

We note that the fact that two different hash functions f_1 and f_2 are used in the two passes has no impact on our results, as the attack technique is independent of the actual functions used. The precomputation phase takes $2 \cdot 2^{(n+\ell)/2+2} + (n - \ell + \frac{n\ell}{2}) \cdot 2^{n/2}$, and the online computation takes $2 \cdot 2^{n-\ell}$ compression function calls.

6 From Herding to Second Preimages: Hash-Twice

If a construction is susceptible to the herding attack, then it is natural to ask whether the second preimage attack of [1] is applicable. The general idea of this attack is to connect the root of the diamond structure to some chaining value encountered during the hashing of the target message, and then connect into the diamond structure (either from the corresponding location in the original message or from a random prefix). This ensures that the new message has the same length (foiling the Merkle-Damgård strengthening).

In this section, we present a second preimage attack against the Hash-Twice construction. The general strategy is to build a diamond structure, and try to connect it to the challenge message (in the second pass). Some complications appear, because the connection may happen anywhere, and the diamond only works on top of a multicollision that has to be located somewhere in the first pass. However, we can use an expandable message [12] to move the multicollision (and therefore the diamond) around. Here is a complete description of the attack. Let us assume that the adversary is challenged with a message M of 2^κ blocks.

The offline processing is as follows:

1. Generate a Kelsey-Schneier expandable message which can take any length between κ and $2^\kappa + \kappa - 1$, starting from the IV yielding a chaining value h_a.
2. Starting from h_a, generate a multicollision of length $(n - \ell) + \ell \cdot n/2$ blocks, that yields a chaining value h_b.
3. Build a diamond structure on top of the multicollision. It yields a chaining value h_x. It is used to herd the second pass.

The online phase, given a message M, is as follows (depicted in Figures 7 and 8):

1. Given h_x, select at random message blocks m^* until $f(h_x, m^*)$ equals to a chaining value h_{i_0} appearing in the second pass of the hashing of M. Let us denote by m the right message block.
2. To position the end of the diamond at the $i_0 - 1$-th block of M, instantiate the expandable message in length of $i_0 - 1 - n \cdot \ell/2 - (n - \ell)$ blocks.
3. Let $h_c = f^*(h_b, m_{i_0} || m_{i_0+1} || \ldots || m_{2^\kappa})$. Compute the second pass on the expandable message, until h_d is reached. Now, using the freedom in the first $n - \ell$ blocks of the multicollision, find a message that sends h_d to a chaining value h_e occurring in the starting points of the diamond in the second pass.

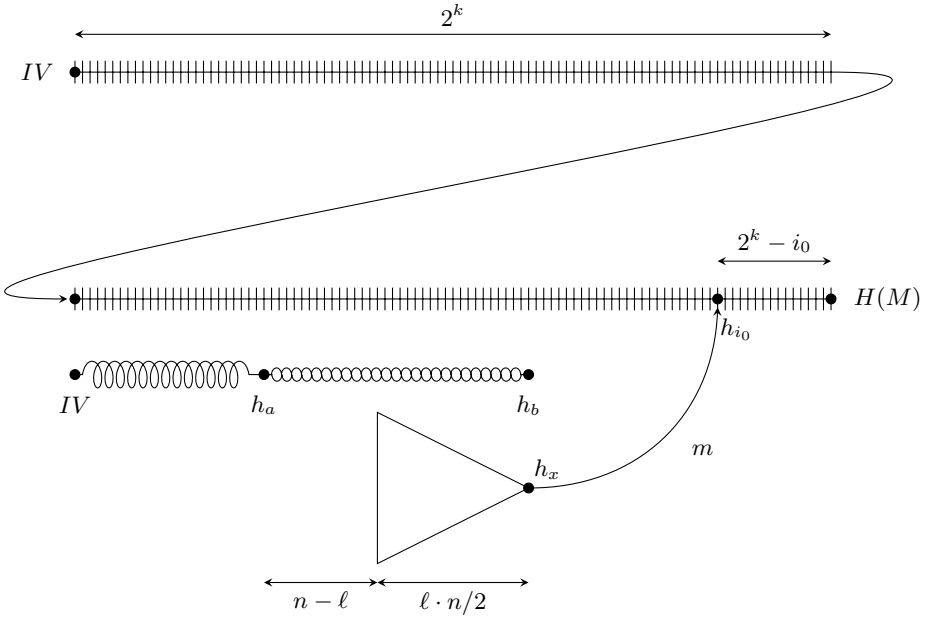


Fig. 7. Second preimage attack on Hash-Twice: first online step

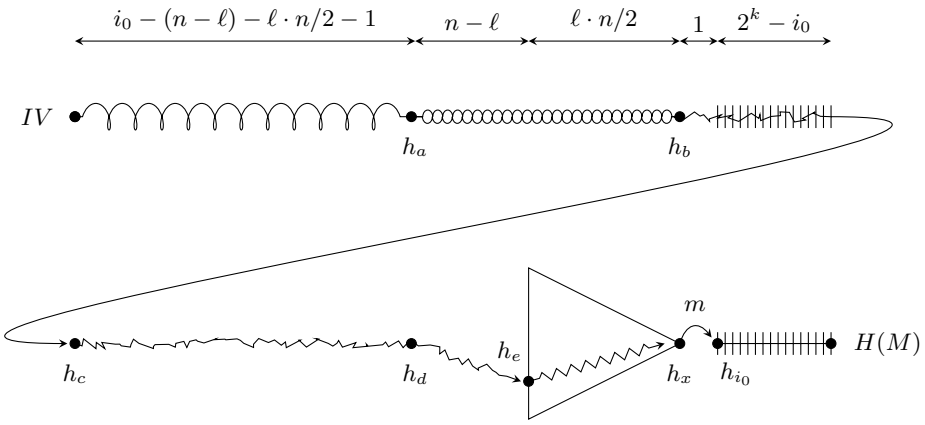


Fig. 8. Second preimage attack on Hash-Twice: online steps 2 to 5

4. Find a path inside the diamond in the second pass (this is also a path inside the multicollision of the first pass). It yields the chaining value h_x at the root of the diamond in the second pipe.
5. Append the connection block m and the suffix of M to obtain the second preimage.

Note that the message forged by assembling the right parts has the same length as M, therefore the padding scheme act the same way on both.

The offline complexity of the attack is the mostly dominated by the need to construct a diamond structure on two pipes, *i.e.*, $2 \cdot 2^{(n+\ell)/2+2}$. The online time complexity is $2^{n-\kappa}$ for finding m, and $2 \cdot 2^{n-\ell}$ connecting to the diamond structure. Hence, the total online time is $2^{n-\kappa} + 2^{n+1-\ell}$.

7 Herding Tree Hashes

In this section we introduce a new method for mounting herding attacks on tree hashes. As in the previous attacks, our method is composed of two steps: offline computation (presented in Section 7.1) and online computation (presented in Section 7.2).

The main differences with the regular herding attacks is the fact that in the case of tree hashes the adversary may suggest embedding the challenge in any block she desires (following the precomputation step). Moreover, the adversary, may publish in advance a great chunk of the answer to the challenge.

7.1 Precomputation Phase

In the offline precomputation phase of the herding attack, the adversary determines the position for inserting the challenge block and commits to the digest value h_T. The diamond-like structure built in this attack allows for freedom in choosing the location of the challenge. Let the length of the padded message (the answer to the challenge, after the embedding of it) be 2^ℓ n-bit blocks, and assume that the compression function is $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$. The details of the offline computation are as follows:

1. Determine the location for the challenge block, *i.e.*, m_3.
2. Choose some set A_1 of $2^{\ell-1}$ arbitrary chaining values for $h_{1,2}$.
3. Fix the message block m_2 (alternatively fix m_1, or parts of m_1 and m_2). For arbitrary m_1^j, and compute $h_{1,1}^j = f(m_1^j, m_2)$. For each $h_{1,2}^i$ find a value $h_{1,1}^i$, such that $\#A_2 \triangleq \{h_{2,1} = f(h_{1,1}^i, h_{1,2}^j)\} = 2^{\ell-2}$.
4. Fix m_6, m_7 and m_8. For arbitrary m_5^j compute $h_{2,2}^j = f(f(m_5^j, m_6), f(m_7, m_8))$. For each $h_{2,1}^i$, find a value $h_{2,2}^i$, such that $\#A_3 \triangleq \{h_{3,1} = f(h_{2,1}^i, h_{2,2}^j)\} = 2^{\ell-3}$.
5. Repeat the above step (each time with a larger set of fixed values), until fixing $m_{2^{\ell-1}+2}, m_{2^{\ell-1}+3}, \ldots, m_{2^\ell}$. For $m_{2^{\ell-1}+1}$, find two possible values, such that $h_{\ell,1}$ collides for the two values in $A_{\ell-1}$.

6. Commit to $h_T = f(h_{\ell,1}, |M|)$.

The chosen points in the set A_1 serve as target values for the online stage of the computation. The goal is to compute the hash digest h_T, such that it is reachable from all points in A_1. For that we reduce the size of A_1 by a factor of 2 to form A_2 by means of collision search through the possible values of m_1. The same principle is followed until the root hash value is computed.
To reduce the complexity of the precomputation it is more efficient for the adversary to fix the known message blocks from the precomputation to constants, rather than to store the exact values needed for each collision in the tree. For a tree of depth ℓ, the adversary can fix all but $\ell + 1$ message blocks, leaving one message block for the challenge, and controlling the paths in the tree through the remaining ℓ blocks. The adversary also can publish the fixed message blocks in advance. However, this is not a strict requirement since these message blocks are already under the control of the adversary.

The time complexity of the precomputation with $2^{\ell-1}$ starting points is about $2 \cdot 2^{\frac{n+(\ell-1)+1}{2}}$ for finding the first layer of collisions. This follows from the fact that we need to try about $2^{\frac{n-(\ell-1)+1}{2}}$ possible message blocks for m_1 (or its equivalent) to find collisions between any pair in the target set A_1, and we need to perform 2 compression function calls to evaluate $h_{2,1}$. For the collisions on the second level $3 \cdot 2^{\frac{n+(\ell-1)-1+1}{2}} + 1$ compression function calls are needed (the last term is due to the computation of $f(m_7, m_8)$ which can be done once). The third level requires $4 \cdot 2^{\frac{n+(\ell-1)-2+1}{2}} + 4$ compression function calls. Hence, in total we have

$$\sum_{j=1}^{\ell-1} \left[(j+1) \cdot 2^{\frac{n+\ell-j+1}{2}} + 2^{j+1} - (j+2) \right] \leq \left(\sum_{j=1}^{\ell-1} (j+1) \cdot 2^{-j/2} \right) \cdot 2^{\frac{n+\ell+1}{2}} + 2^{\ell+1} - \frac{\ell^2}{2}$$

The sum in the right-hand side admits $\frac{1-2\sqrt{2}}{2\sqrt{2}-3} \leq 10.66$ as a limit when ℓ goes to infinity, which yields an approximate offline complexity of $15.08 \cdot 2^{\frac{n+\ell}{2}}$ compression function calls. The space complexity here is $2^\ell - 1$ and is determined by the amount of memory blocks that are required for the storage of the target points in A_1 and the precomputed values for the non-fixed message blocks (in this example chosen to be $m_1^*, m_5^*, m_9^*, \ldots$).

7.2 Online Phase

Here the adversary obtains the challenge P, and has to:

1. Find m_4^*, such that $f(P, m_4^*) = h_{1,2}$ where $h_{1,2} \in A_1$. Note that $h_{1,2}$ fixes the rest of the message blocks $m_1^*, m_5^*, m_9^*, \ldots, m_{2^{\ell-1}+1}^*$.
2. Retrieve the stored value for m_1^* for which $f(f(m_1^*, m_2), h_{2,1}) \in A_2$. Tracing the correct intermediate chaining values, arrive to the correct value for $m_{2^{\ell-1}+1}^*$ which leads to $h_{\ell,1}$ and h_T.
3. Output $m_1^*, m_2, P, m_4^*, m_5^*, m_6, m_7, m_8, m_9^*, m_{10}, \ldots, m_\ell$ as the answer.

The workload in the online phase of the computation reflects the cost of linking to a point contained in the set A_1. Approximately $2^{n-(\ell-1)}$ compression function calls are required to link correctly to one of the $2^{(\ell-1)}$ points in A_1.

7.3 Variants and Applications of the Herding Attack on Tree Hash Functions

Precomputed challenge messages. If there exists a limited set of possible challenges, it is possible to precompute the points in A_1. This allows for a very efficient connection in the online stage, however, at the cost of losing flexibility– only the precomputed message blocks can be "herded" to h_T.

Herding sequences of adjacent message blocks. The herding attack also allows for inserting sequences (instead of a single block) of adjacent challenge blocks. In this case the set of target chaining values A_1 has to be embedded deeper in the tree structure. This results in larger online complexity due to the evaluation of additional nodes on the path to the target linking set A_1.

Herding both sides of the hash tree. The diamond-like structure used for herding trees can accommodate the insertion of a challenge message blocks on both sides of the hash tree due to symmetry of the structure. It is thus no more expensive to construct a diamond structure that allows $2^{\ell-1}$ choices on both sides of the root. This means that an adversary can either herd one message block on the left half of the message, and another on the right, or satisfying two challenges simultaneously with the same diamond structure.

Applicability. We note that the proposed herding attack is applicable to other variants of the tree hash function. Even if the employed compression functions in the tree are distinct (e.g., as considered in MD6 [17]), it is still possible to apply the attack, because an adversary knows (and controls) the location of the challenges.[3] The attack also works irrespective of the known random XOR masks (e.g., tree constructions of the XOR tree type [2, 18]) applied on the chaining values at each level.

8 Long-Message Second Preimages in 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 $f(m_{2j-1}, m_{2j})$ is the position of the resulting node in the tree) are vulnerable to a long-message second preimage attack which changes at most two blocks of the message.

We know that $h_{1,j} = f(m_{2j-1}, m_{2j})$ for $j = 1$ to $L/2$ for a message M of length $L = 2^\kappa$ blocks. Then given the target message M, there are $2^{\kappa-1}$ chaining values $h_{1,j}$ that can be targeted. If the adversary is able to invert even one of

[3] Still, note that the herding attack on MD6 has increased offline complexity (compared to our estimates) because of its large internal state and subsequent truncation in the final output transformation.

these chaining values, i.e., to produce (m', m'') such that $f(m', m'') = h_{1,j}$ for some $1 \leq j \leq 2^{\kappa-1}$, then he has successfully produced a second preimage M'. Note, however that (m', m'') shall differ than the corresponding pair of message blocks in the original target message 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.

Furthermore, it is also possible to model the inversion of f as a task for a time-memory-data attack [4]. The $h_{1,j}$ values are the multiple targets, which compose $D = 2^{\kappa-1}$ data points. Using the time-memory-data curve of the attack from [4], 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 trade-off curve can be used as long as $M < N, T < N$, and $T \geq D^2$ (see [3] for more details about the last constraint). 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 $n = 128$ with $\kappa = 30$, one can apply the time-memory-data tradeoff attack using 2^{99} pre-processing time and 2^{66} memory blocks, and find a second preimage in 2^{66} online computation.

The described long message second preimage attack on trees applies to not only strengthened Merkle trees, but also to XOR-Trees [2] and optimized variants of these hash functions [18].

9 New Trojan Message Attacks on Merkle-Damgård Hash Functions

> "*Do not trust the horse, Trojans. Whatever it is, I fear the Greeks even when they bring gifts*" (Virgil's Aeneid, Book 2, 19 BC)

In this section, we introduce a new generic attack on many hash function constructions, called the Trojan Message attack. A Trojan message is a string S which is produced offline by an attacker, and is then provided to a victim. The victim then selects some prefix P from a constrained set of choices, and produces the message $P \,||\, S$. However, due to the way S was chosen, the attacker is now able to find a second preimage for $P \,||\, S$.

Given a Merkle-Damgård hash for which collisions may be found, Trojan messages may be produced. In general, the Trojan message requires at least one message input block, and one collision search, per possible value of P. If there are 1024 possible values of P, an attacker may produce a 1024-block Trojan message, requiring 1024 collision searches.

One can imagine a Trojan message attack being practical against applications which use MD5, and which permit an attacker to provide some victim with "boilerplate" text for the end of his document, while imposing a relatively constrained set of choice for his part of the document.

Against Merkle-Damgård hashes, Trojan message attacks take two forms:

1. If only straightforward collisions of the compression function are possible, second preimages for the full message keep the victim's choice of P, but introduce a limited change in S. That is, the attacker finds $S' \neq S$ such that $H(P \,||\, S) = H(P \,||\, S')$.
2. If collisions of the compression function starting from different chaining values are possible, second preimages for the full message give the attacker a choice of P, and leave S mostly unchanged. That is, the attacker finds P' and S' such that $H(P \,||\, S) = H(P' \,||\, S')$.

Let $\mathcal{P} = \{P_1, \ldots, P_N\}$ be a set of N known prefix messages and h_0^i be the intermediate chaining value resulting from the computation of $f^*(P_i)$. Note, that without loss of generality, we can assume that all the prefixes have the same length (otherwise, we just consider padded versions). Therefore, we safely disregard strengthening and padding issues.

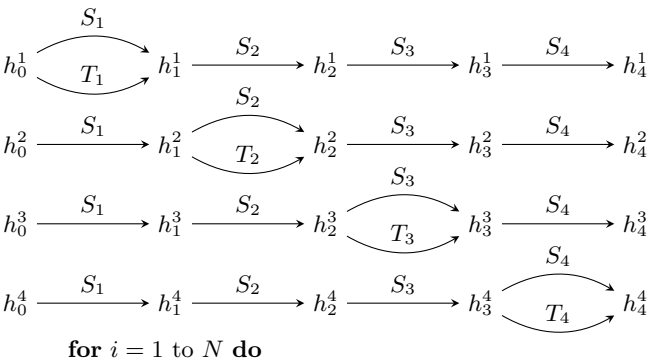
9.1 The Collision Trojan Attack

The collision variant of the trojan message attack makes use of a collision finding algorithm **IndenticalPrefixCollision** which takes a chaining value as parameter and produces a pair of messages colliding from this chaining value. The attack proceeds as follows:

1. $\mathcal{A}$ computes N colliding message pairs (S_i, T_i) using the algorithm of figure 9.
2. $\mathcal{A}$ sends $\mathcal{B}$ a suffix message $S = S_1 \,||\, \ldots \,||\, S_N$.
3. $\mathcal{B}$ commits to $h = H^f(P_i \,||\, S)$ where P_i is in $\mathcal{P}$.
4. $\mathcal{A}$ finds out P_i through exhaustive search amongst the N possible choices and outputs:
$$M' = P_i \,||\, S_1 \,||\, \ldots \,||\, T_i \,||\, \ldots \,||\, S_N$$

We have that $H^f(M') = h$. The hashing of $P_i \,||\, S$ and $P_i \,||\, S'$ differs only when T_i replaces S_i, but because these two blocks collide, both hash processes do not diverge.

The only non-trivial part of the attack for $\mathcal{A}$ is the first step where $\mathcal{A}$ precomputes N collisions for each prefix from the set $\mathcal{P}$ (in time $N \cdot 2^{n/2}$), and evaluates the compression function N^2 times. If finding a collision for the hash function is easy, e.g., like the legacy hash function MD5 [16] the attack can be even practical. It has recently been shown that finding a collision in MD5 takes about 2^{16} evaluations of the compression function [19]. For instance, one can forge in a matter of seconds a suffix S of 46720 bytes permitting to find second preimages for MD5 if the prefix set $\mathcal{P}$ is the set of the days of the year.



for $i = 1$ to N **do**
  $(S_i, T_i) \leftarrow$ **IndenticalPrefixCollision**$\left(h_{i-1}^i\right)$
  for $j = 1$ to N **do**
    $h_i^j \leftarrow f\left(h_{i-1}^j, S_i\right)$
  end for
end for

Fig. 9: Trojan Message Attack, Collision Variant

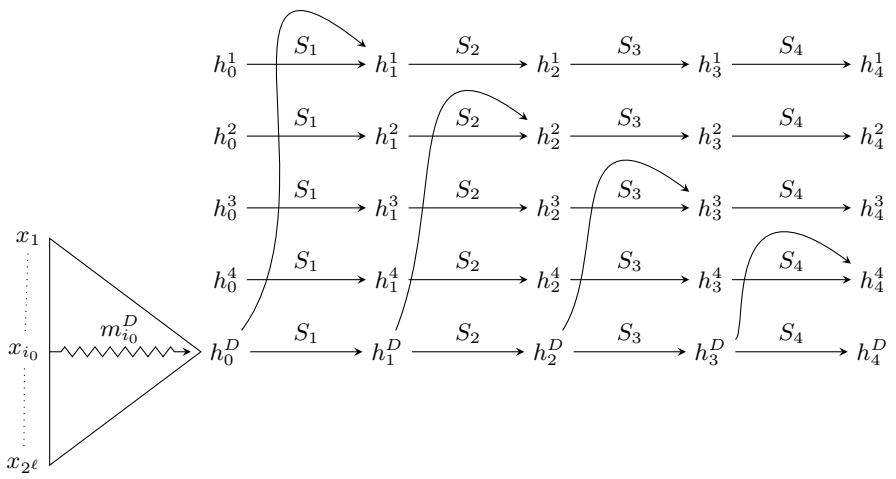
9.2 The Herding Trojan Attack

The herding variant of the trojan message attack is stronger, and allows for more freedom for the attacker. In exchange, the preprocessing and the online running times are larger.

Let K denote the length of all possible prefixes in $\mathcal{P}$. We can extend K to be as large as we wish. The herding variant of the trojan message attack makes use of a different, more sophisticated "chosen-prefix" collision finding algorithm **ChosenPrefixCollision**(h_1, h_2) that returns the messages m_1 and m_2, such that $f(h_1, m_1) = f(h_2, m_2)$. In some specific cases this collision is harder to find (for instance in MD5, such collision takes 2^{41} compression function evaluations [19]).

Another difference between this variant and the previous one, is that in this variant, the adversary is challenged by a second prefix P', not controlled by him, which he has to herd to the same value as $H^f(P_i \,||\, S)$. The attack proceeds as follows:

1. $\mathcal{A}$ computes a diamond structure with 2^ℓ entry points, denoted by $D_1 = \{\mathbf{h}_i\}$, converging to the hash value h_0^D, with the constraint that $\ell < K - 2$.
2. $\mathcal{A}$ generates N colliding message pairs using the algorithm of figure 10.
3. $\mathcal{A}$ sends $\mathcal{B}$ a suffix message $S = S_1 \,||\, \ldots \,||\, S_N$.
4. $\mathcal{B}$ commits to $h = H^f(P_i \,||\, S)$ where $P_i \in \mathcal{P}$.
5. $\mathcal{A}$ is challenged with an arbitrary prefix P' of size at most $K - \ell - 1$ blocks, not necessarily in the known prefix set.
6. $\mathcal{A}$ finds (by random trials) a connecting message C of size $K - \ell - |P'|$ blocks such that $h_{i_0} = f^*(P' \,||\, C) \in D_1$.
7. $\mathcal{A}$ forges a new prefix $Q = P' \,||\, C \,||\, m_{i_0}^D$, which is such that $f^*(Q) = h_0^D$.



for $i = 1$ to N **do**
  $(S_i, T_i) \leftarrow$ **ChosenPrefixCollision**$\left(h_{i-1}^i, h_{i-1}^D\right)$
  for $j = 1$ to N **do**
    $h_j^i \leftarrow f\left(h_j^{i-1}, S_i\right)$
  end for
  $h_i^D \leftarrow f\left(h_{i-1}^D, S_i\right)$
end for

Fig. 10: Trojan Message Attack, Herding Variant

8. As in the collision version, $\mathcal{A}$ outputs $Q \,||\, S'$, where $S' = S_1 \,||\, \ldots \,||\, T_i \,||\, \ldots \,||\, S_N$.

As in the collision variant, we have that $H^f(Q \,||\, S') = h$. The reasoning to establish this fact is essentially the same.

The workload of the attack is step one where $\mathcal{A}$ constructs a diamond structure with 2^ℓ starting points and N collisions for each prefix from the set P. Thus, the precomputation complexity is of order $2^{n/2+\ell/2+2} + N \cdot 2^{n/2}$. The online cost is the connection step for computing the prefix P' and is of order $2^{n-\ell}$.

9.3 Applications of the Trojan Attacks

The trojan attack can is highly useful in instances with a set of predictable prefixes, and where the attacker is able to suggest a suffix to introduce to the message. Such a case is the X.509 certificate, where the adversary may generate a second certificate (with the same identification) but with different public keys. Another possible application is a time stamping service, which signs $\mathcal{MDH}(ts, M)$ where ts is a time stamp and M is the document.

Trojan Attacks on Tree Hashes. The processing of the prefix in tree hashes is independent of the suffix processing. Thus, $\mathcal{A}$ computes independent collisions for each message input node. In fact, it is only enough that $\mathcal{A}$ produces a single

colliding suffix block $f(S_i) = f(T_i)$ in the first level of the tree evaluation. Then, for all $P_i \in \mathcal{P}$, $\mathcal{A}$ can compute $\mathcal{Tree}(P_i\|S) = \mathcal{Tree}(P_i\|S')$ where $S = S_1\| \ldots \|S_i\| \ldots \|S_L$, $S' = S_1\| \ldots \|T_i\| \ldots \|S_L$ and L may be different than $|\mathcal{P}|$. The latter is true, because as opposed to Merkle-Damgård, here the length of the suffix is independent of the size of the prefix set $\mathcal{P}$. This completes the collision variant of the trojan message attack on tree hashes.

The herding trojan message attack on tree hashes could be applied as follows. $\mathcal{A}$ executes first the herding attack on tree hashes. Instead of selecting arbitrary chosen target set, here $\mathcal{A}$ fixes the target set T to consist of all intermediate hash values of the known prefixes $P_i \in \mathcal{P}$. Then, as in the collision tree variant of the attack, $\mathcal{A}$ computes S' by creating collision(s) on the top tree node(s) (distinct from the target set). Now, challenged on P', $\mathcal{A}$ finds P'_s, such that $f(P'\|P'_s) \in T$ where $|P'\|P'_s| = |P_i|$ and $|P'_s|$ is at least $\log_2(2^n/|\mathcal{P}|)$. Then $\mathcal{Tree}(P'\|P'_s\|S') = \mathcal{Tree}(P_i\|S)$, which concludes the herding variant of the attack.

10 Summary and Conclusions

Our results enhance the understanding of the multi-pipe and multi-pass modes of iteration, such as concatenated hashes, zipper hash, hash-twice, and tree hash functions. The presented attacks reconfirm the knowledge that there is only a limited gain by concatenating the output of hash functions when it comes to security, and that the hash twice construction is not secure.

Moreover, we show that all of the investigated constructions suffer the herding attack. An interesting result is that domain separation (equivalent to distinct internal compression function evaluation, e.g., by means of a counter separation) does not protect any of the existing hash functions against herding attacks. And while domain separation often does offer protection against second preimage attacks, it appears to be unable to also mitigate herding attacks. An open question remains to exhibit either a generic herding protective mechanism or a mode of operation optimally secure against standard and herding attacks.

Acknowledgments

We would like to thank Lily Chen, Barbara Guttman and the anonymous referees for their useful feedback. This work was supported in part by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. The first author is supported by a Ph.D. Fellowship from the Flemish Research Foundation (FWO–Vlaanderen). The third author was supported by the France Telecom Chaire.

References

1. Andreeva, E., Bouillaguet, C., Fouque, P.A., Hoch, J.J., Kelsey, J., Shamir, A., Zimmer, S.: Second Preimage Attacks on Dithered Hash Functions. In Smart,

N.P., ed.: EUROCRYPT. Volume 4965 of Lecture Notes in Computer Science, Springer (2008) 270–288
2. Bellare, M., Rogaway, P.: Collision-Resistant Hashing: Towards Making UOWHFs Practical. In Kaliski, B.S.J., ed.: CRYPTO. Volume 1294 of Lecture Notes in Computer Science, Springer (1997) 470–484
3. Biryukov, A., Mukhopadhyay, S., Sarkar, P.: Improved time-memory trade-offs with multiple data. In Preneel, B., Tavares, S.E., eds.: Selected Areas in Cryptography. Volume 3897 of Lecture Notes in Computer Science, Springer (2005) 110–127
4. 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
5. 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)
6. Coppersmith, D.: Another birthday attack. In Williams, H.C., ed.: CRYPTO. Volume 218 of Lecture Notes in Computer Science, Springer (1985) 14–17
7. Damgård, I.: A Design Principle for Hash Functions. [5] 416–427
8. Dean, R.D.: Formal Aspects of Mobile Code Security. PhD thesis, Princeton University (January 1999)
9. Hoch, J.J., Shamir, A.: Breaking the ice - finding multicollisions in iterated concatenated and expanded (ice) hash functions. In Robshaw, M.J.B., ed.: FSE. Volume 4047 of Lecture Notes in Computer Science, Springer (2006) 179–194
10. 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
11. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In Vaudenay, S., ed.: EUROCRYPT'06. Volume 4004 of Lecture Notes in Computer Science, Springer (2006) 183–200
12. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer, R., ed.: EUROCRYPT'05. Volume 3494 of Lecture Notes in Computer Science, Springer (2005) 474–490
13. Liskov, M.: Constructing an Ideal Hash Function from Weak Ideal Compression Functions. In Biham, E., Youssef, A.M., eds.: Selected Areas in Cryptography. Volume 4356 of Lecture Notes in Computer Science, Springer (2006) 358–375
14. Merkle, R.C.: One Way Hash Functions and DES. [5] 428–446
15. Nandi, M., Stinson, D.R.: Multicollision attacks on some generalized sequential hash functions. IEEE Transactions on Information Theory **53**(2) (2007) 759–767
16. Rivest, R.L.: The MD5 message-digest algorithm. RFC1321 (April 1992)
17. Rivest, R.L., Agre, B., Bailey, D.V., Crutchfield, C., Dodis, Y., Fleming, K.E., Khan, A., Krishnamurthy, J., Lin, Y., Reyzin, L., Shen, E., Sukha, J., Sutherland, D., Tromer, E., Yin, Y.L.: The md6 hash function, a proposal to nist for sha-3 (2008)
18. Sarkar, P.: Construction of universal one-way hash functions: Tree hashing revisited. Discrete Applied Mathematics **155**(16) (2007) 2174–2180
19. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate. Cryptology ePrint Archive, Report 2009/111 (2009) `http://eprint.iacr.org/`.