New Attacks on the Concatenation and XOR Hash Combiners

Itai Dinur

Department of Computer Science, Ben-Gurion University, Israel

Abstract. We study the security of the concatenation combiner $H_1(M) \| H_2(M)$ for two independent iterated hash functions with n-bit outputs that are built using the Merkle-Damgård construction. In 2004 Joux showed that the concatenation combiner of hash functions with an n-bit internal state does not offer better collision and preimage resistance compared to a single strong n-bit hash function. On the other hand, the problem of devising second preimage attacks faster than 2^n against this combiner has remained open since 2005 when Kelsey and Schneier showed that a single Merkle-Damgård hash function does not offer optimal second preimage resistance for long messages.

In this paper, we develop new algorithms for cryptanalysis of hash combiners and use them to devise the first second preimage attack on the concatenation combiner. The attack finds second preimages faster than 2^n for messages longer than $2^{2n/7}$ and has optimal complexity of $2^{3n/4}$. This shows that the concatenation of two Merkle-Damgård hash functions is not as strong a single ideal hash function.

Our methods are also applicable to other well-studied combiners, and we use them to devise a new preimage attack with complexity of $2^{2n/3}$ on the XOR combiner $H_1(M) \oplus H_2(M)$ of two Merkle-Damgård hash functions. This improves upon the attack by Leurent and Wang (presented at Eurocrypt 2015) whose complexity is $2^{5n/6}$ (but unlike our attack is also applicable to HAIFA hash functions).

Our algorithms exploit properties of random mappings generated by fixing the message block input to the compression functions of H_1 and H_2. Such random mappings have been widely used in cryptanalysis, but we exploit them in new ways to attack hash function combiners.

Keywords: Hash function, cryptanalysis, concatenation combiner, XOR combiner.

1 Introduction

Hash functions are among the main building blocks of many cryptographic protocols used today. A hash function H takes as an input a message M of an arbitrary length, and maps it to an output $H(M)$ of a fixed length n. The main security properties expected from a hash function are:

1. **Collision resistance**: It should be difficult to find a pair of different messages M and M' such that $H(M) = H(M')$.

2. **Preimage resistance**: Given an arbitrary n-bit value V, it should be difficult to find any message M such that $H(M) = V$.
3. **Second preimage resistance**: Given a target message M, it should be difficult to find any different message M' such that $H(M) = H(M')$.

When the hash is function viewed as a "random oracle", it offers collision resistance up to a security level of $2^{n/2}$ due to the birthday paradox. The expected security level against preimage and second preimage attacks is 2^n.

In practice, widely deployed standards (such as MD5 and SHA-1) have failed to provide the expected security level [37, 38]. As a result, researchers have proposed to combine the outputs of two hash functions to provide better security in case one (of even both) hash functions are weak. In fact, hash function combiners were also used in practice in previous versions of the SSL and TLS protocols [10, 15].

The most well-known hash function combiner is the concatenation combiner that concatenates the outputs of two hash functions $H_1(M) \| H_2(M)$. This combiner was described already in 1993 [34] and has been subject to extensive analysis in various settings (e.g., [20, 26]). From the theoretical side, researchers have studied the notion of a *robust combiner*, which is secure in case at least one of the combined hash functions is secure. Clearly, the concatenation combiner is secure with respect to the main security properties defined above, e.g., a collision $H_1(M) \| H_2(M) = H_1(M') \| H_2(M')$ implies $H_1(M) = H_1(M')$ and $H_2(M) = H_2(M')$. Lines of research regarding robust combiners include the study of advanced combiners in [13, 14] and study of the minimal output length of robust combiners [5, 33, 35] (more recently works include [27, 29]).

We are interested in this paper in the security of combiners of iterated hash functions that break the message M into blocks $m_1 \| m_2 \| \ldots \| m_L$ of fixed length and processes them by iterative applications of a compression function h (or several compression functions) that updates an internal state x_i using the previous state and the current message block $x_i = h(x_{i-1}, m_i)$. Hash functions whose internal state size is n-bits are known as "narrow-pipe" constructions and typically output the final state x_L. A very common way to build iterated hash functions is the Merkle-Damgård construction [8, 28] which applies the same compression function h in all iterations, and adds a padding of the message length to final message block (known as Merkle-Damgård strengthening). Iterated hash function constructions (and in particular, the Merkle-Damgård construction) are very efficient and widely used in practice.

In a breakthrough paper published in 2004 [22], Joux showed that the collision and preimage resistance of the concatenation combiner of iterated narrow-pipe hash functions[1] is not much greater than that of a single n-bit hash function. The result of Joux was highly influential and resulted in numerous followup works in hash function analysis. The main technique introduced in [22] (known as Joux multi-collisions) generates a large multi-collision (many message that map to the same value) in a narrow-pipe hash function in time which is not much greater than $2^{n/2}$ (the time required to generate a single collision).

[1] In fact, only one of the hash functions has to be iterated.

Joux's multi-collisions were used shortly afterwards by Kelsey and Schneier in order to show that the second preimage resistance of the Merkle-Damgård construction is less than the optimal 2^n for a long input target message M [23]. The idea is to compute the specific sequence of n-bit states $a_1, \ldots, a_L$ that is generated during the application of the compression function h on the message M. We then try to "connect" to some internal state a_i in this sequence with a different message prefix, and reuse the message suffix $m_{i+1} \| \ldots \| m_L$ to obtain the second preimage M'. However, this approach is foiled by the Merkle-Damgård strengthening if M and M' are not of the same length. The main idea of Kelsey and Schneier was use Joux's multi-collisions to construct an *expandable message* which essentially allows to expand the connected message prefix of M' to the desired length i, and overcome the length restriction.

Following the results of Joux (which showed that the concatenation combiner does not increase collision and preimage resistance), and the later results of Kelsey and Schneier (which showed that the second preimage resistance of the Merkle-Damgård construction is less than 2^n), a natural question is whether there exists a second preimage attack on the concatenation combiner of Merkle-Damgård hash functions that is faster than 2^n. Interestingly, the problem of devising such an attack remained open for a long time despite being explicitly mentioned in several papers including [12], and much more recently in [25]. In fact, although the works of [22, 23] have attracted a significant amount of followup research on countermeasures against second preimage attacks (such as "hash twice" or "dithered hash") and attacks that break them [1–3], there has been no progress with respect to second preimage attacks on the basic concatenation combiner.

In this paper, we devise the first second preimage attack on the concatenation combiner of Merkle-Damgård hash functions which is faster than 2^n. As in related attacks (and in particular, [23]) we obtain a tradeoff between the complexity of the attack and the length of the target message. In particular, our second preimage attack is faster than 2^n only for input messages of length at least[2] $2^{2n/7}$. The optimal complexity[3] of our attack is $2^{3n/4}$, and is obtained for (very) long messages of length $2^{3n/4}$. Due to these constraints, the practical impact of our second preimage attack is limited and its main significance is theoretical. Namely, it shows that the concatenation of two Merkle-Damgård hash functions is not as strong a single ideal hash function.

The general framework of our attack is similar to the one of the long message second preimage attack of Kelsey and Schneier. Namely, we first compute the sequences of internal states $a_1, \ldots, a_L$ and $b_1, \ldots, b_L$ by applying the compression functions h_1 and h_2 on the target message $M = m_1 \| \ldots \| m_L$. Our goal is then to "connect" to one of the state pairs (a_i, b_i) using a different message prefix of

[2] For example, for $n = 160$ and message block of length 512 bits (as in SHA-1), the attack is faster than 2^{160} only for messages containing at least 2^{46} blocks, or 2^{52} bytes.

[3] The complexity formulas do not take into account (small) constant factors, which are generally ignored throughout this paper.

the same size. Once we manage to achieve this, we can reuse the same message suffix as in M and obtain a second preimage.

There are two main challenges in this approach, where the main challenge is connect to some state pair (a_i, b_i) generated by M from a different message. The secondary challenge is to ensure that the connected message prefixes are of the same length. We overcome the later challenge by building a (simultaneous) expandable message for two Merkle-Damgård hash functions with complexity that is not much larger than the complexity of building it for a single hash function [23]. The expandable message is built by using a cascaded form of Joux's multi-collisions, a technique which was used in previous works starting from Joux's original paper [22] and up to subsequent works such as [19, 30]. A similar construction of an expandable message over two hash functions was proposed in the independent paper [21] by Jha and Nandi, which analyzes the zipper hash.

A much more difficult challenge is to actually connect to the target message on a state pair (a_i, b_i) from a different message of arbitrary (smaller) length. Indeed, the obvious approach of attempting to reach an arbitrary $2n$-bit state pair by trying random messages requires more than 2^n time, since the number of target state pairs is equal to the message length which is smaller than 2^n. A more promising approach is to use the *interchange structure*, which was recently introduced at Eurocrypt 2015 by Leurent and Wang [25]. Indeed, this structure is very useful in analysis of hash combiners as it breaks the dependency between the sequences of states computed during the computation of h_1 and h_2 on a common message. More specifically, the structure consists of an initial state pair (as, bs) and two sets of internal states A for H_1 and B for H_2 such that: for any value $a \in A$ and any value $b \in B$, it is possible to efficiently construct a message $M_{a,b}$ that sends (as, bs) to (a, b). Assume that there exists an index $i \in \{1, 2, \ldots, L\}$ such that $a_i \in A$ and $b_i \in B$, then we can connect to (a_i, b_i) using M_{a_i,b_i} as required. Unfortunately, this does not result in an efficient attack, essentially because the complexity of building an interchange structure for sufficiently large sets A and B is too high.

In this paper we use a different approach whose first step is to fix an arbitrary message block m, giving rise to functional graphs generated by the random n to n-bit mappings $f_1(x) = h_1(x, m)$ and $f_2(y) = h_2(y, m)$. Such random mappings have many interesting properties and have been extensively studied and used in cryptography in the classical works of Hellman [18] and van Oorschot and Wiener [36], and much more recently in [11, 17, 24, 31, 32]. However, in our case, the use of random mappings may seem quite artificial and unrelated to our goal of connecting to the arbitrary target message. Nevertheless, we will show how to exploit random mappings to search for a "special" state pair (a_p, b_p) whose states can be reached relatively easily from an arbitrary starting state pair (using the fixed message block m), thus connecting to the target message.

More specifically, we are particularly interested in "special" states of $a_1, \ldots, a_L$ and $b_1, \ldots, b_L$ that are located deep in the functional graphs defined by f_1 and f_2, i.e., these states can be reached after iterating f_1 and f_2 many times. Such special states (called *deep iterates*) are relatively easy to reach from arbitrary starting states. Our goal is to find a state pair (a_p, b_p) composed of two deep iterates in $f_1(x)$ and $f_2(y)$, respectively.[4] Once we find such a "special" state pair, we show how to simultaneously reach both of its states in an efficient manner from an arbitrary state pair. Combined with the expandable message, this gives the desired second preimage.

Interestingly, our algorithms for cryptanalysis of hash combiners are related to recent attacks on hash-based MACs [11, 17, 24, 32], as in both settings the notion of distance of a node in the functional graph from a deep iterate plays an important role in the attack.

Our techniques are quite generic and can be applied to attack other Merkle-Damgård hash function combiners. In particular, they are applicable to another well-known combiner defined as $H_1(M) \oplus H_2(M)$ and known as the XOR combiner. At Eurocrypt 2015 [25], Leurent and Wang devised the first preimage attack against the XOR combiner (using the aforementioned interchange structure) with optimal complexity of $2^{5n/6}$. Here, we improve this attack for Merkle-Damgård hash functions and obtain an optimal complexity of $2^{2n/3}$. In practice, many concrete hash functions limit the maximal allowed message length L, and our attack on the XOR combiner gives a trade-off of $2^n \cdot L^{-2/3}$ (between L and the time complexity of attack). This improves the tradeoff of $2^n \cdot L^{-1/2}$, obtained by Leurent and Wang's attack. For a particular example mentioned in [25], we improve the complexity of finding a preimage of the XOR of two well-known Merkle-Damgård hash functions SHA-512 $\oplus$ WHIRLPOOL by a factor of about 2^{21} (from 2^{461} to 2^{440}). On the other hand, we stress that our techniques only apply in case that both hash functions combined use the Merkle-Damgård construction. In particular, our attacks are not applicable in case at least one combined hash function in built using the HAIFA mode [4] (that uses a block counter in every compression function call). In this case, the attack of Leurent and Wang remains the only published preimage attack on the XOR combiner that is faster than exhaustive search.

Finally, we point out that we can use the interchange structure [25] in order to optimize the complexity of our attacks on both the concatenation and XOR combiners. Although this does not lead to a very big improvement, it further demonstrates the wide applicability of this structure in cryptanalysis of hash function combiners.

The rest of the paper is organized as follows. In Section 2 we describe some preliminaries. In Section 3 we describe our second preimage attack on the concatenation combiner, while our preimage attack on the XOR combiner is described is Section 4. Finally, we conclude the paper in Section 5.

2 Preliminaries

In this section, we describe preliminaries that are used in the rest of the paper.

[4] The actual attack is slightly different, as it searches for deep iterates from which (a_p, b_p) can be reached with a common message block.

2.1 The Merkle-Damgård Construction [8, 28]

The Merkle-Damgård construction [8, 28] is a common way to build a hash function from a compression function $h : \{0,1\}^n \times \{0,1\}^k \rightarrow \{0,1\}^n$, where n denotes the size of the chaining value[5] and k denotes the block size.

The input message M is padded with its length (after additional padding which ensures that the final message length is a multiple of the block size k). The message is then divided into k-bit blocks $m_1\|m_2\|\ldots\|m_L$. The initial state of the hash function is an n-bit initial chaining value $x_0 = IV$. The compression function is then applied L times

$$x_i = h(x_{i-1}, m_i).$$

The hash value is set as $H(M) = x_L$. We extend the definition of h recursively to process an arbitrary number of k-bit message blocks, namely $h(x, m_1\|m_2\|\ldots\|m_L) = h(h(x, m_1), m_2\|\ldots\|m_L)$. Moreover, we donote by $|M|$ the length of M in blocks.

Given a state x and a message m such that $x' = h(x, m)$, we say that m *maps* the state x to the state x' (the compression function h used for this mapping will be clear from the context) and denote this by $x \xrightarrow{m} x'$. Throughout this paper we assume that the compression function is chosen uniformly at random from all $n + k$ to n-bit functions, which implies that our analysis applies to most (but not all) compression functions.

2.2 Hash Function Combiners

Given two hash functions H_1 and H_2, the concatenation combiner is defined as $H_1(M)\|H_2(M)$ while the XOR combiner is defined as $H_1(M) \oplus H_2(M)$. In this paper we are interested in the security of these combiners in the case that both H_1 and H_2 are based on the Merkle-Damgård construction with independent compression functions. We denote the IVs of H_1 and H_2 by IV_1 and IV_2 (respectively), their compression functions by h_1 and h_2, (respectively), and assume that both the chaining values and outputs are of size n (our techniques can also be extended to other cases). An additional technicality has to do with the message block sizes of H_1 and H_2, and we assume that both are equal to k. Once again, our techniques also apply in the case that the block sizes are different. A generic (although not always to most efficient) way to deal with this case is to align the block sizes by defining a “superblock” whose size is the least common multiple of the two block sizes.

We pair the (extended) compression functions h_1 and h_2 using the notation of $h_{1,2}$. Given two states x, y and a message m such that $x' = h_1(x, m)$ and $y' = h_2(y, m)$, we write $(x', y') = h_{1,2}((x, y), m)$. In this case, we say that m *maps* (or sends) the state pair (x, y) to the state pair (x', y') (the compression functions h_1, h_2 are clear from the context) and denote this by $(x, y) \xrightarrow{m} (x', y')$.

[5] In this paper, we mainly consider “narrow-pipe” constructions in which the sizes of the chaining value and the hash function output are the same, but our techniques and analysis extend naturally (with additional cost) to “wide-pipe” constructions in which the chaining value size is larger.

2.3 Joux's Multi-collisions [22]

In a well-known result [22], Joux observed that finding a large multi-collision in an iterated hash function H is not much more expensive than finding a single collision. The algorithm starts from a state x_0 and evaluates the compression function for arbitrary message blocks until a collision is found. According to the birthday paradox, $2^{n/2}$ compression function evaluations are sufficient to find a collision $h(x_0, m_1) = h(x_0, m'_1)$ for some m_1 and m'_1. Next, we set $x_1 = h(x_0, m_1)$ and continue the process iteratively t times. This gives 2^t t-block messages that reach x_t from x_0, as in the i'th block we can select either m_i or m'_i. Altogether, 2^t collisions are found with about $t \cdot 2^{n/2}$ compression function evaluations.

Joux's multi-collisions have numerous applications in cryptanalysis of hash functions. Next, we focus on one of the most relevant applications for this paper.

2.4 The Long Message Second Preimage Attack [23]

In [9], Dean devised a second preimage attack for long messages on specific Merkle-Damgård hash functions for which it is easy to find fixed points in their compression function. Given a target message $M = m_1 \| m_2 \| \ldots \| m_L$, the attacker computes the sequence of internal states $IV = a_0, a_1, \ldots, a_L$ generated during the invocation of the compression function on M. A simplified attack would now start from the state $IV = x_0$ and evaluate the compression function with arbitrary message blocks until a collision $h(x_0, m) = a_i$ is found for some message block m and index i. The attacker can now append the message suffix $m_{i+1} \| \ldots \| m_L$ to m, hoping to obtain the target hash value $H(M)$. However, this approach does not work due to the final padding of the message length which would be different if the message prefixes are of different lengths.

The solution of Dean was to compute an *expandable message* which consists of the initial state x_0 and another state $\hat{x}$ such that for each length κ (in some range) there is a message M_κ of κ blocks that maps x_0 to $\hat{x}$. Thus, the algorithm first finds a collision $h(\hat{x}, m) = a_i$, and the second preimage is computed as $M_{i-1} \| m \| m_{i+1} \| \ldots \| m_L$.

The assumption that it is easy to find fixed points in the compression function is used in [9] in efficient construction of the expandable message. In [23], Kelsey and Schneier described a more generic attack that uses multi-collisions of a special form to construct an expandable message, removing the restriction of Dean regarding fixed points. As in Joux's original algorithm, the multi-collisions are constructed iteratively in t steps. In the $i'th$ step, we find a collision between some m_i and m'_i such that $|m_i| = 1$ (it is a single block) and $|m'_i| = 2^{i-1} + 1$, namely $h(x_{i-1}, m_i) = h(x_{i-1}, m'_i)$. This is done by picking an arbitrary prefix of size 2^{i-1} of m'_i denoted by m', computing $h(x_{i-1}, m') = x'$ and then looking for a collision $h(x_{i-1}, m_i) = h(x', m'')$ using a final block m'' (namely, $m'_i = m' \| m''$).

The construction of Kelsey and Schneier gives an expandable message that can be used to generate messages starting from x_0 and reaching $\hat{x} = x_t$ whose (integral) sizes range in the interval $[t, 2^t + t - 1]$ (it is denoted as a $(t, 2^t + t - 1)$-expandable message). A message of length $t \leq \kappa \leq 2^t + t - 1$ is generated by looking at the t-bit binary representation of $\kappa - t$. In iteration $i \in \{1, 2, \ldots, t\}$, we select the long message fragment m'_i if the $i'th$ LSB of $\kappa - t$ is set to 1 (otherwise, we select the single block m_i).

Given that the target message M is of length $L \leq 2^{n/2}$ blocks, the construction of the expandable message in the first phase of the attack requires less than $n \cdot 2^n$ computation, while obtaining the collision with one of the states computed during the computation of M requires about $1/L \cdot 2^n$ compression function evaluations according to the birthday paradox.

2.5 Functional Graph

In various phases of our attacks, we evaluate a compression function h with a fixed message input block m (e.g., the zero block), and simplify our notation by defining $f(x) = h(x, m)$. The mapping f gives rise a directed functional graph in which nodes are the n-bit states and an edge from node x to y is defined if and only if $f(x) = y$.

In this paper, we are particularly interested in nodes of f which are located deep in the functional graph. More specifically, x' is an iterate of depth i if there exists some ancestor node x' such that $x' = f^i(x)$. Deep iterates can be reached using *chains* evaluated from an arbitrary starting point x_0 by computing a sequence of nodes using the relation $x_{i+1} = f(x_i)$. We denote this sequence by $\overrightarrow{x}$.

A useful algorithm for expanding the functional graph of f is given below. This algorithm is not new and has been previously used (for example, in [17, 32]). It takes an input parameter $g \geq n/2$ which determines the number of expanded nodes (and the running time of the algorithm).

```
1. Initialize $G = \emptyset$ as a data structure of evaluated nodes.
2. Until $G$ contains $2^g$ nodes:
   (a) Pick an arbitrary starting point $x_0$ and evaluate the chain $x_{i+1} = f(x_i)$ until it cycles (there exists $x_i = x_j$ for $i \neq j$) or it hits a point in $G$. Add the points of the chain to $G$.
```

A simple but important observation that we exploit is that after we have executed the algorithm and developed 2^g nodes, then another chain from an arbitrary starting point is expected to collide with the evaluated graph at depth of roughly 2^{n-g}. This is a direct consequence of the birthday paradox. In particular, this observation implies that most chains developed by the algorithm will be extended to depth $\Omega(2^{n-g})$ (without colliding with G of cycling), and therefore a constant fraction of the developed nodes are iterates of depth 2^{n-g}. In total, the algorithm develops $\Theta(2^g)$ iterates of f of depth 2^{n-g} in 2^g time.

In this paper we will be interested in the probability of encountering a specific deep iterate at each stage of the evaluation of a chain from an arbitrary starting point.

Lemma 1. *Let f be an n-bit random mapping, and x'_0 an arbitrary point. Let $D \leq 2^{n/2}$ and define the chain $x'_i = f(x'_{i-1})$ for $i \in \{1, \ldots, D\}$ (namely, x'_D is an iterate of depth D). Let x_0 be a randomly chosen point, and define $x_d = f(x_{d-1})$. Then, for any $d \in \{1, \ldots, D\}$, $Pr[x_d = x'_D] = \Theta(d \cdot 2^{-n})$.*

Proof (Sketch). We can assume that the chains do not cycle (i.e., each chain contains distinct nodes), as $D \leq 2^{n/2}$. In order for $x_d = x'_D$ to occur then x_{d-i} should collide with x'_{D-i} for[6] some $0 \leq i \leq d$. For a fixed i, the probability for this collision is roughly[7] 2^{-n}, and summing over all $0 \leq i \leq d$ (are events are disjoint), we get that the probability is about $d \cdot 2^{-n}$.

Distinguished Points The memory complexity of many algorithms that are based on functional graphs (e.g., parallel collision search [36]) can be reduced by utilizing the *distinguished points* method (which is attributed to Ron Rivest). Assume that our goal is to detect a collision of a chain (starting from an arbitrary node) with the nodes of G computed above, but without storing all the 2^g nodes in memory. The idea is to define a set of 2^g distinguished points (nodes) using a simple predicate (e.g. the $n-g$ LSBs of a node are zero). The nodes of G contain approximately $2^g \cdot 2^{g-n} = 2^{2g-n}$ distinguished points, and only they are stored in memory. A collision of an arbitrary chain with G is expected to occur at depth of about 2^{n-g}, and will be detected at the next distinguished point which is located (approximately) after traversing additional 2^{n-g} nodes. Consequently, we can detect the collision with a small overhead in time complexity, but a significant saving factor of 2^{n-g} in memory.

Interestingly, in the specific attack of Section 4, the distinguished points method is essential for reducing the time complexity of the algorithm.

3 A New Long Message Second Preimage Attack on the Concatenation Combiner

In this attack we are given a target message $M = m_1 \| m_2 \| \ldots \| m_L$ and our goal is to find another message M' such that $H_1(M') \| H_2(M') = H_1(M) \| H_2(M)$ (or equivalently $H_1(M') = H_1(M)$ and $H_2(M') = H_2(M)$). We denote the sequence of internal states computed during the invocation of h_1 (respectively, h_2) on M by $a_0, a_1, \ldots, a_L$ (respectively, $b_0, b_1, \ldots, b_L$). We start with a high-level overview of the attack and then give the technical details.

The attack is composed of three main phases. In the first phase, we build (a special form of) an expandable message, similarly to the second preimage attack on a single hash function [23]. This expandable message essentially consists of the initial state pair (IV_1, IV_2) and final state pair $(\hat{x}, \hat{y})$ such that for each length κ in some appropriate range (which is roughly $[n^2, L]$) there is a message M_κ of κ blocks that maps (IV_1, IV_2) to $(\hat{x}, \hat{y})$.

[6] A collision between x_{d-i} and x'_{D-i} occurs if $x_{d-i} = x'_{D-i}$ but $x_{d-i-1} \neq x'_{D-i-1}$.

[7] A more accurate analysis would take into account the event that the chains collide before x_{d-i}, but the probability for this is negligible.

In the second phase our goal is to find a pair of states $(\bar{x}, \bar{y})$, a message block $\bar{m}$ and an index p such that $(\bar{x}, \bar{y}) \xrightarrow{\bar{m}} (a_p, b_p)$ (note that the state pair (a_p, b_p) is computed during the evaluation of the target message). Moreover, the state pair $(\bar{x}, \bar{y})$ should have a special property which is formally defined in the full description of the second phase.

In the third and final phase, we start from $(\hat{x}, \hat{y})$ and compute a message fragment $\hat{M}$ of length q (shorter than $p - 1$) such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}} (\bar{x}, \bar{y})$. This phase can be performed efficiently due to the special property of $(\bar{x}, \bar{y})$.

In order to compute the second preimage, we pick M_{p-q-1} using the expandable message, giving $(IV_0, IV_1) \xrightarrow{M_{p-q-1}} (\hat{x}, \hat{y})$, and concatenate $M_{p-q-1} \| \hat{M} \| \bar{m}$ in order to reach the state pair (a_p, b_p) from (IV_1, IV_2) with a message of appropriate length p. Indeed, we have

$$(IV_0, IV_1) \xrightarrow{M_{p-q-1}} (\hat{x}, \hat{y}) \xrightarrow{\hat{M}} (\bar{x}, \bar{y}) \xrightarrow{\bar{m}} (a_p, b_p).$$

Altogether, we obtain the second preimage

$$M' = M_{p-q-1} \| \hat{M} \| \bar{m} \| m_{p+1} \| \ldots \| m_L.$$

This attack can be optimized using the interchange structure, as described in Appendix A.

Notation For the sake of clarity, we summarize below the notation that is shared across the various phases. We note that each phase also introduces additional "internal" notation whose scope is only defined within the phase.

$\boldsymbol{M = m_1 \| \ldots \| m_L}$	**:** Target Message.
$\boldsymbol{a_0, \ldots, a_L}$ $\boldsymbol{(b_0, \ldots, b_L)}$	**:** Sequence of internal states computed during the invocation of h_1 (h_2) on M.
$\boldsymbol{M'}$	**:** Computed second preimage.
$\boldsymbol{(\hat{x}, \hat{y})}$	**:** Endpoint pair of expandable message (computed in Phase 1).
$\boldsymbol{(a_p, b_p)}$	**:** State pair (in the sequences $a_0, \ldots, a_L$ and $b_0, \ldots, b_L$) on which the computation of M and M' coincides (computed in Phase 2).
$\boldsymbol{(\bar{x}, \bar{y}),\ \bar{m}}$	**:** "Special" state pair and message block used to reach (a_p, b_p) (computed in Phase 2).
$\boldsymbol{\hat{M}}$	**:** Message fragment that maps $(\hat{x}, \hat{y})$ to $(\bar{x}, \bar{y})$ (computed in Phase 3).
$\boldsymbol{q}$	**:** Length of $\hat{M}$ (smaller than $p - 1$).

Complexity Evaluation Denote $L = 2^{\ell}$. For a parameter $g_1 \geq max(n/2, n - \ell)$, the complexity of the phases of the attack (as computed in their detail description) is given below (ignoring constant factors).

Phase 1: $L + n^2 \cdot 2^{n/2} = 2^{\ell} + n^2 \cdot 2^{n/2}$
Phase 2: $1/L \cdot 2^{2n-g_1} = 2^{2n-g_1-\ell}$
Phase 3: $2^{3g_1/2}$

We balance the second and third phases by setting $2n - g_1 - \ell = 3g_1/2$, or $g_1 = 2/5 \cdot (2n - \ell)$, giving time complexity of $2^{3/5 \cdot (2n-\ell)}$. This tradeoff holds as long as $2^{\ell} + n^2 \cdot 2^{n/2} \leq 2^{3/5(2n-\ell)}$, or $\ell \leq 3n/4$. The optimal complexity is $2^{3\ell/4}$, obtained for $\ell = 3n/4$. The attack is faster than 2^n (Joux's preimage attack) for[8] $\ell > n/3$. The message range for which the attack is faster than 2^n can be slightly improved to $L \geq 2^{2n/7}$ using the optimized attack, described in Appendix A.

3.1 Details of Phase 1: Constructing an Expandable Message

In this phase, we build a simultaneous expandable message for two Merkle-Damgård hash functions. This expandable message consists of the initial states (IV_1, IV_2) and final states $(\hat{x}, \hat{y})$ such that for each length κ in some appropriate range (determined below) there is a message M_κ of κ blocks that maps (IV_1, IV_2) to $(\hat{x}, \hat{y})$.

We set $C \approx n/2 + \log(n)$ as a constant. Our basic building block consists of two pairs of states (xa, ya) and (xb, yb) and two message fragments ms and ml that map the state pair (xa, ya) to (xb, yb). The message ms is the (shorter) message fragment of fixed size C, while ml is of size $i > C$. Below, we will show how to construct this building block for any state pair (xa, ya) and length $i > C$.

Given this building block and a positive parameter t, we build an expandable message in the range of $[C(C-1) + tC, C^2 - 1 + C(2^t + t - 1)]$. This is done by utilizing a sequence of $C - 1 + t$ basic building blocks. The first $C - 1$ building blocks are built with parameters $i \in \{C+1, C+2, \ldots, 2C-1\}$. It is easy to see that these structures give a $(C(C-1), C^2-1)$–expandable message by selecting at most one longer message fragment from the sequence, where the remaining $C - 2$ (or $C - 1$) fragments are of length C. The final t building blocks give a standard expandable message, but it is built in intervals of C. These t building blocks are constructed with parameters $i = C(2^{j-1} + 1)$ for $j \in \{1, \ldots, t\}$.

Given a length κ in the appropriate range of $[C(C-1)+tC, C^2-1+C(2^t+t-1)]$, we can construct a corresponding message by first computing κ (modulo C). We then select the length $\kappa' \in [C(C-1), C^2-1]$ such that $\kappa' \equiv \kappa$ (modulo C), defining the first $C - 1$ message fragment choices. Finally, we compute $(\kappa - \kappa')/C$ which is an integer in the range of $[t, 2^t + t - 1]$ and select the final t message fragment choices as in a standard expandable message using the binary representation of $(\kappa - \kappa')/C$.

[8] Note that for $\ell > n/3$, $g_1 = 2/5 \cdot (2n - \ell) > 2n/3 > max(n/2, n - \ell)$, as required.

Construction of the Building Block Given state pair (xa, ya) and length $i > C$, the algorithm for constructing the building block for the expandable message is based on multi-collisions as described below.

1. Pick an arbitrary prefix mp of size $i - C$ blocks and compute $xp = h_1(xa, mp)$.
2. Find a collision $h_1(xa, m_1) = h_1(xp, m'_1) = x_2$ with single block messages m_1, m'_1.
3. Build a 2^{C-1} standard Joux multi-collision in h_1 starting from x_2 and denote its final endpoint by xb. Altogether we have a multi-collision in h_1 with 2^C messages that map xa to xb. Out of these messages, 2^{C-1} are of length C (obtained by first selecting m_1) and we denote this set of messages by S_1. The remaining 2^{C-1} messages are of length i (obtained by first selecting the $(i - C + 1)$-block prefix $mp \| m'_1$), and we denote this set of messages by S_2.
4. Evaluate $yp = h_2(ya, mp)$ and store the result. Next, evaluate h_2 from ya on the two sets S_1 and S_2 (using the stored yp to avoid recomputing $h_2(ya, mp)$) and find a collision between them (such a collision is very likely to occur since $C - 1 > n/2$). The collision gives the required $ms \in S_1$ and $ml \in S_2$ of appropriate sizes such that $yb \triangleq h_2(ya, ms) = h_2(ya, ml)$ and $xb \triangleq h_1(xa, ms) = h_1(xa, ml)$.

The complexity of Step 1 is less than i compression function evaluations. The complexity of Step 2 is about $2^{n/2}$, while the complexity of Step 3 is about $C \cdot 2^{n/2} \approx n \cdot 2^{n/2}$. The complexity of Step 4 is about $i + n \cdot 2^{n/2}$. In total, the complexity of constructing the basic building block is about $i + n \cdot 2^{n/2}$ (ignoring small factors).

Complexity Analysis of the Full Phase The full expandable message requires computing $C - 1 + t$ building blocks whose sum of length parameters (dominated by the final building block) is about $C \cdot 2^t \approx n \cdot 2^t$. Assuming that $t < n$, we construct $C - 1 + t \approx n$ building blocks and the total time complexity of constructing the expandable message is about $n \cdot 2^t + n^2 \cdot 2^{n/2}$. Our attack requires the $(C(C-1)+tC, C^2-1+C(2^t+t-1))$–expandable message to extend up to length L, implying that $L \approx n \cdot 2^t$, and giving time complexity of about

$$L + n^2 \cdot 2^{n/2}.$$

3.2 Details of Phase 2: Finding a Target State Pair

In the second phase, we fix some message block m, giving rise to the functional graphs $f_1(x) = h_1(x, m)$ and $f_2(y) = h_1(y, m)$ and let $g_1 \geq n/2$ be a parameter (to be determined later). Our goal is to find a pair of states $(\bar{x}, \bar{y})$, a message block $\bar{m}$ and an index p such that the following two conditions hold:

1. The state $\bar{x}$ is an iterate of depth 2^{n-g_1} in the functional graph of $f_1(x)$ and $\bar{y}$ is an iterate of depth 2^{n-g_1} in the functional graph of $f_2(y)$.

2. The state pair $(\bar{x}, \bar{y})$ is mapped to (a_p, b_p) by $\bar{m}$, or $(\bar{x}, \bar{y}) \xrightarrow{\bar{m}} (a_p, b_p)$.

The algorithm of this phase is given below.

```
1. Fix an arbitrary single-block value $m$.
2. Expand the functional graph of $f_1$ using the procedure of Section 2.5 with parameter $g_1$. Store all encountered iterates of depth $2^{n-g_1}$ in a table $T_1$.
3. Similarly, expand the functional graph of $f_2$ using the procedure of Section 2.5 with parameter $g_1$. Store all encountered iterates of depth $2^{n-g_1}$ in a table $T_2$.
4. For single-block values $m' = 0, 1, \ldots$, perform the following steps:
   (a) For each node $x \in T_1$ evaluate $x' = h_1(x, m')$ and store the matches $x' = a_i$ with the[a] sequence $a_1, \ldots, a_L$ in a table $T_1'$, sorted according to the index $i$ of $a_i$.
   (b) Similarly, for each node $y \in T_2$ evaluate $y' = h_2(y, m')$ and look for matches $y' = b_j$ with the sequence $b_1, \ldots, b_L$. For each match with some $b_j$, search for the index $j$ in the table $T_1'$. If a match $i = j$ is found, set $p \triangleq i$ (namely, $(a_p, b_p) \triangleq (x', y')$), $\bar{m} \triangleq m'$ and $(\bar{x}, \bar{y}) \triangleq (x, y)$. This gives $(\bar{x}, \bar{y}) \xrightarrow{\bar{m}} (a_p, b_p)$ as required. Otherwise (no match $i = j$ is found), go back to Step 4.
```

[a] More precisely, due to the minimal length restriction of the expandable message, matches $x' = a_i$ with i smaller than (approximately) $C^2 \approx n^2$ cannot be exploited in the full attack. Moreover, the maximal exploitable value of i is $L - 2$. However, the fraction of these nodes is very small and can be ignored in the complexity analysis.

The time complexity of steps 2 and 3 (which execute the algorithm of Section 2.5) is about 2^{g_1}. The time complexity of step 4.(a) and step 4.(b) is also bounded by 2^{g_1} (given that $a_1, \ldots, a_L$ and $b_1, \ldots, b_L$ are sorted in memory), as the size of T_1 and T_2 is at most 2^{g_1} and the number of matches found in each step can only be smaller.

We now calculate the expected number of executions of Step 4 until the required (a_p, b_p) is found. Using the analysis of Section 2.5, the expected size of T_1 and T_2 (that contain iterates of depth 2^{n-g_1}) is close to 2^{g_1}. According to the birthday paradox, the expected size of T_1' is about $L \cdot 2^{g_1 - n}$. Similarly, the number of matches $y' = b_j$ is also about $L \cdot 2^{g_1 - n}$. The probability of a match $i = j$ in Step 4.(b) is computed using a birthday paradox on the L possible indexes, namely, $1/L \cdot (L \cdot 2^{g_1-n})^2 = L \cdot 2^{2g_1 - 2n}$. As a result, Step 4 is executed about $1/L \cdot 2^{2n-2g_1}$ times until the required (a_p, b_p) is found (the executions with different blocks m' are essentially independent). Altogether, the total time complexity of this step is

$$2^{g_1} \cdot 1/L \cdot 2^{2n-2g_1} = 1/L \cdot 2^{2n-g_1}.$$

Since the index p is uniformly distributed in the interval $[1, L]$, we will assume that $p = \Theta(L)$.

3.3 Details of Phase 3: Hitting the Target State Pair

In the third and final phase, we start from $(\hat{x}, \hat{y})$ and compute a message fragment $\hat{M}$ of length $q < p-1$ such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}} (\bar{x}, \bar{y})$. We use in a strong way the fact that the state $\bar{x}$ (and $\bar{y}$) is a deep iterate (of depth 2^{n-g_1}) in the functional graph of $f_1(x)$ $(f_2(y))$.

This phase is carried out by picking an arbitrary starting message block ms, which gives points $x_0 = h_1(\hat{x}, ms)$ and $y_0 = h_2(\hat{y}, ms)$. We then continue to evaluate the chains $x_i = h_1(x_{i-1}, m)$ and $y_j = h_2(y_{j-1}, m)$ up to a maximal length L' (determined below). We hope to encounter $\bar{x}$ at some distance $q-1$ from x_0 and to encounter $\bar{y}$ at the same distance $q-1$ from y_0. Given that $q-1 < p$, this will give the required $\hat{M} = ms \| [m]^{q-1}$ (where $[m]^{q-1}$ denotes the concatenation of $q-1$ message blocks m), which is of length $q < p-1$. In case $\bar{x}$ and $\bar{y}$ are encountered at different distances in the chains, or at least one of them is not encountered at all, we pick a different value for ms and start again.

The next question which we address is to what maximal length L' should we evaluate $\overrightarrow{x}$ and $\overrightarrow{y}$. As we wish to reach iterates $\bar{x}$ and $\bar{y}$ of depth 2^{n-g_1}, it can be shown that $L' = 2^{n-g_1}$ is optimal. Since the total chain length should be less than $p-1$, this imposes the restriction $L' = 2^{n-g_1} < p-1 < L$, or $2^{g_1} < 2^n/L$.

We now estimate the probability that $\bar{x}$ and $\bar{y}$ will be encountered at the same distance from the arbitrary starting points of the chains x_0 and y_0. This probability will allow us to compute the number of chains from different starting points that we need to evaluate in this phase of the attack, which is an important parameter in the complexity evaluation.

Since $\bar{x}$ is an iterate of depth 2^{n-g_1} in $f_1(x)$, it is an endpoint of a chain of states of length $L' = 2^{n-g_1}$ (such a chain was computed in Section 3.2). Let d be in the interval $[1, L'] = [1, 2^{n-g_1}]$, then according to Lemma 1, $Pr[x_d = \bar{x}] \approx d \cdot 2^{-n}$ (this is the probability that $\bar{x}$ will be encountered at distance d from x_0). Due to the independence of f_1 and f_2, $Pr[x_d = \bar{x} \bigwedge y_d = \bar{y}] \approx (d \cdot 2^{-n})^2$. Summing the probabilities of the (disjoint) events over all distances d in the interval $[1, 2^{n-g_1}]$, we conclude that the probability that $\bar{x}$ and $\bar{y}$ will be encountered at the same distance is about $(2^{n-g_1})^3 \cdot 2^{-2n} = 2^{n-3g_1}$.

The probability calculation seems to give rise to the conclusion that we need to compute about 2^{3g_1-n} chains from different starting points in this phase of the attack. This conclusion was verified experimentally, but its proof is incomplete since the various trials performed by selecting different starting points for the chains are dependent. More details can be found in Appendix B.

The Algorithm of Phase 3 The naive algorithm described above performs about 2^{3g_1-n} trials, where each trial evaluates chains of length $L' = 2^{n-g_1}$ from arbitrary points, giving a total time complexity of about $2^{3g_1-n+n-g_1} = 2^{2g_1}$. Since $g_1 \geq n/2$, the time complexity of the full algorithm is at least 2^n and it is not faster than Joux's preimage attack.

In order to optimize the algorithm, we further expand the graphs of f_1 and f_2. As a result, the evaluated chains are expected to collide with the graphs sooner (before they are evaluated to the full length of 2^{n-g_1}). Once a collision occurs, we use a lookahead procedure to calculate the distance of the chain's starting point from $\bar{x}$ (or $\bar{y}$). This lookahead procedure resembles the one used in recent attacks on hash-based MACs [17, 32] (although the setting and actual algorithm in our case are obviously different).

More specifically, we pick a parameter $g_2 > g_1$ and execute the algorithm below (see Figure 1 for illustration).

1. Develop 2^{g_2} nodes in the functional graphs of f_1 (and f_2) (as specified in Section 2.5) with the following modifications.
   - Store at each node its distance from $\bar{x}$ (or $\bar{y}$ in f_2) (the maximal stored distance is $L' = 2^{n-g_1}$): for each computed chain, once it hits a previously visited node in the graph, obtain its stored distance from $\bar{x}$ (or $\bar{y}$ in f_2) and update it in all the computed nodes in the current chain up to the maximal value $L' = 2^{n-g_1}$.
   - If a chain does not hit $\bar{x}$, then the distance of its nodes is undefined and stored as a special value $\perp$. Similarly, this special value is used for nodes whose distance from $\bar{x}$ is larger than L'.
   - The evaluated nodes for f_1 (f_2) are stored in the data structure G_1 (G_2).
2. For single-block values $ms = 0, 1, \ldots$, compute $x_0 = h_1(\hat{x}, ms)$ and $y_0 = h_2(\hat{y}, ms)$ and repeat the following step:
   (a) Compute the chains $\overrightarrow{x}$ and $\overrightarrow{y}$ up to maximal length $L' = 2^{n-g_1}$, or until they hit G_1 and G_2 (respectively).
      - If $\overrightarrow{x}$ (or $\overrightarrow{y}$) does not hit G_1 (G_2), return to Step 2.
      - Otherwise, once $\overrightarrow{x}$ ($\overrightarrow{y}$) hits G_1 (G_2), obtain the stored distance from $\bar{x}$ ($\bar{y}$) at the collision point. If the distance to $\bar{x}$ (or $\bar{y}$) is undefined, return to Step 2.
      - Compute the respective distances i and j of x_0 and y_0 from $\bar{x}$ and $\bar{y}$. If $i \neq j$, return to Step 2.
      - Otherwise ($i = j$), denote $q = i + 1$. If $q \geq p - 1$, return to Step 2.
      - Otherwise ($q < p - 1$), return the message $\hat{M} = ms \| [m]^i = ms \| [m]^{q-1}$ as output.

The time complexity of Step 1 is 2^{g_2}. As previously computed, in Step 2 we perform about 2^{3g_1-n} trials before encountering two starting points with the same distance to $\bar{x}$ and $\bar{y}$. According to the analysis of Section 2.5, each trial requires about 2^{n-g_2} computation (before hitting G_1 and G_2). Therefore, the total time complexity of this phase is $2^{g_2} + 2^{3g_1-n} \cdot 2^{n-g_2} = 2^{g_2} + 2^{3g_1-g_2}$. The complexity is minimized by setting $g_2 = 3g_1/2$ which balances the two terms and gives time complexity of

$$2^{3g_1/2}.$$

Finally, we note that the memory complexity of this algorithm can be optimized using distinguished points. A detailed way to achieve this will be presented in the closely related algorithm of Section 4.2.

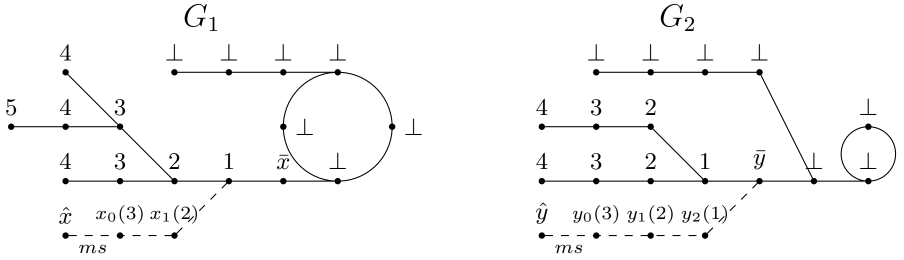

Fig. 1. Phase 3 of the Attack

4 A New Preimage Attack on the XOR Combiner

In this attack we are given a target n-bit preimage value V and our goal is to find a message M such that $H_1(M) \oplus H_2(M) = V$. Although the formal problem does not restrict M in any way, several concrete hash functions restrict the length of M. Therefore, we will also assume that the size of M is bounded by a parameter L. As in the previous attack, we start with a high-level overview and then give the technical details.

The attack is composed of three main phases which are similar to the second preimage attack on the concatenation combiner of Section 3. The first phase is identical to the first phase of the attack of Section 3. Namely, we build an expandable message that consists of the initial states (IV_1, IV_2) and final states $(\hat{x}, \hat{y})$ such that for each length κ in an appropriate range there is a message M_κ of κ blocks that maps (IV_1, IV_2) to $(\hat{x}, \hat{y})$. The description of this phase is given in Section 3.1 and is not repeated below.

In the second phase of the attack, we find a set S (of size 2^s) of tuples of the form $((x, y), w)$ such that w is a single block, $(x, y) \xrightarrow{w} (a, b)$, and $h_1(a, pad) \oplus h_2(b, pad) = V$, where pad is the final block of the (padded) preimage message of length L. Moreover, (x, y) has a special property that will be defined in the detailed description of this phase.

In the third and final phase, we start from $(\hat{x}, \hat{y})$ and compute a message fragment $\hat{M}$ of length q (shorter than $L - 2$) such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}} (\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in S$. For this tuple, denote $(\bar{a}, \bar{b}) \triangleq h_{1,2}((\bar{x}, \bar{y}), \bar{m})$.

Finally, we pick M_{L-q-2} using the expandable message, giving $(IV_0, IV_1) \xrightarrow{M_{L-q-2}}$ $(\hat{x}, \hat{y})$, and concatenate $M_{L-q-2} \| \hat{M} \| \bar{m}$ in order to reach the state pair $(\bar{a}, \bar{b})$ from (IV_1, IV_2) with a message of appropriate length $L - 1$. Indeed, we have

$$(IV_0, IV_1) \xrightarrow{M_{L-q-2}} (\hat{x}, \hat{y}) \xrightarrow{\hat{M}} (\bar{x}, \bar{y}) \xrightarrow{\bar{m}} (\bar{a}, \bar{b}).$$

Altogether, we obtain the padded preimage for the XOR combiner

$$M = M_{L-q-2} \| \hat{M} \| \bar{m} \| pad.$$

We note that this attack can be optimized using the interchange structure, similarly to the attack on the concatenation combiner. However, the improvement is rather small and we do not give the details here.

Notation We summarize below the notation that is shared across the various phases.

$\boldsymbol{V}$	: Target preimage.
$\boldsymbol{M}$	: Computed preimage.
$\boldsymbol{L}$	: Length of M.
$\boldsymbol{pad}$	: Final block of (the padded) M.
$(\boldsymbol{\hat{x}}, \boldsymbol{\hat{y}})$	: Endpoint pair of expandable message (computed in Phase 1).
$\boldsymbol{S}$:	Set containing tuples of the form $((x, y), w)$ such that w is a single block, $(x, y) \xrightarrow{w} (a, b)$, and $h_1(a, pad) \oplus h_2(b, pad) = V$ (computed in Phase 2).
$\boldsymbol{2^s}$	: Size of S.
$((\boldsymbol{\bar{x}}, \boldsymbol{\bar{y}}), \boldsymbol{\bar{m}})$	: State pair and message block in S used in M (computed in Phase 3).
$(\boldsymbol{\bar{a}}, \boldsymbol{\bar{b}})$	: State pair defined as $(\bar{a}, \bar{b}) \triangleq h_{1,2}((\bar{x}, \bar{y}), \bar{m})$ (computed in Phase 3).
$\boldsymbol{\hat{M}}$	: Message fragment used in M that maps $(\hat{x}, \hat{y})$ to $(\bar{x}, \bar{y})$ (computed in Phase 3).
$\boldsymbol{q}$	: The length of $\hat{M}$ (smaller than $L - 2$).

Complexity Evaluation Denote $L = 2^{\ell}$. For parameters $g_1 \geq max(n/2, n - \ell)$ and $s \geq 0$, the complexity of the phases of the attack (as computed in their detail description) is given below (ignoring constant factors).

Phase 1: $2^{\ell} + n^2 \cdot 2^{n/2}$
Phase 2: 2^{n+s-g_1}
Phase 3: $2^{3g_1/2-s/2} + L \cdot 2^{9g_1/2-2n-3s/2} + L \cdot 2^{2g_1-n} = 2^{3g_1/2-s/2} + 2^{\ell+9g_1/2-2n-3s/2} + 2^{\ell+2g_1-n}$

We balance the time complexities of the second phase and the first term in the expression of the third phase by setting $n + s - g_1 = 3g_1/2 - s/2$, or $s = 5g_1/3 - 2n/3$, giving a value of $2^{n/3+2g_1/3}$ for these terms. Furthermore, $\ell + 9g_1/2 - 2n - 3s/2 = \ell + 2g_1 - n$ and the time complexity expression of Phase 3 is simplified to $2^{n/3+2g_1/3} + 2^{\ell+2g_1-n}$. Since g_1 is a positive factor in all the terms, we optimize the attack by picking the minimal value of g_1 under the restriction $g_1 \geq max(n/2, n-\ell)$. In case $\ell \leq n/2$, we set $g_1 = n - \ell$ and the total time complexity of the attack[9] is

$$2^{n/3+2(n-\ell)/3} = 2^{n-2\ell/3}.$$

The optimal complexity is $2^{2n/3}$, obtained for $\ell = n/2$ by setting $g_1 = n/2$.

4.1 Details of Phase 2: Finding a Set of Target State Pairs

In the second phase, we fix some message block m, giving rise to the functional graphs defined by the random mappings $f_1(x) = h_1(x, m)$ and $f_2(y) = h_1(y, m)$. Given parameters $g_1 \geq n/2$ and $s \geq 0$, our goal is to compute a set S (of size 2^s) of tuples of the form $((x, y), w)$ where w is a single block such that for each tuple:

1. The state x is an iterate of depth 2^{n-g_1} in the functional graph of $f_1(x)$ and y is an iterate of depth 2^{n-g_1} in the functional graph of $f_2(y)$.
2. $(x, y) \xrightarrow{w} (a, b)$ and $h_1(a, pad) \oplus h_2(b, pad) = V$, where *pad* is a final block of the (padded) preimage message of length L.

The algorithm of this phase is (obviously) somewhat different from the algorithm of Section 3.2 due to the fact that the goal of this attack and the actual combiner scheme attacked are different. This algorithm resembles the algorithm used in the final phase in Leurent and Wang's attack [25], as both look for state pairs (x, y) that give $h_1(x, w\|pad) \oplus h_2(y, w\|pad) = V$ (for some message block w). The difference is that in the attack of [25], (x, y) was an arbitrary endpoint pair of the interchange structure, while in our case, we look for x and y that are deep iterates.

```
1. Fix an arbitrary single-block value $m$.
2. Expand the functional graph of $f_1$ using the procedure of Section 2.5
   with parameter $g_1$. Store all encountered iterates of depth $2^{n-g_1}$ in a
   table $T_1$.
3. Similarly, expand the functional graph of $f_2$ using the procedure of
   Section 2.5 with parameter $g_1$. Store all encountered iterates of depth
   $2^{n-g_1}$ in a table $T_2$.
4. Allocate a set $S = \emptyset$. For single-block values $w = 0, 1, \ldots$, perform the
   following steps until $S$ contains $2^s$ elements:
   (a) For each node $x \in T_1$ evaluate $h_1(x, w\|pad)$, and store the results
       in a table $T_1'$, sorted according $h_1(x, w\|pad)$.
   (b) Similarly, for each node $y \in T_2$ evaluate $h_2(y, w\|pad) \oplus V$, and look
       for matches $h_2(y, w\|pad) \oplus V = h_1(x, w\|pad)$ with $T_1'$. For each
       match, add the tuple $((x, y), w)$ to $S$.
```

[9] Note that $\ell + 2g_1 - n = n - \ell < n - 2\ell/3$.

The time complexity of steps 2 and 3 is about 2^{g_1}. The time complexity of step 4.(a) and step 4.(b) is also bounded by 2^{g_1}. We now calculate the expected number of executions of Step 4 until 2^s matches are found and inserted into S.

According to the analysis of Section 2.5, the expected size of T_1 and T_2 (the number of deep iterates) is close to 2^{g_1}. Thus, for each execution of Step 4, the expected number of matches on n-bit values $h_2(y, w\|pad) \oplus V = h_1(x, w\|pad)$ is 2^{2g_1-n}. Consequently, Step 4 is executed 2^{s+n-2g_1} times in order to obtain 2^s matches. Altogether, the total time complexity of this step is

$$2^{n+s-2g_1+g_1} = 2^{n+s-g_1}.$$

4.2 Details of Phase 3: Hitting a Target State Pair

In the third and final phase, we start from $(\hat{x}, \hat{y})$ and compute a message $\hat{M}$ of length q (shorter than $L-2$) such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}} (\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in S$.

This phase is carried out by picking an arbitrary starting message block ms, which gives points $x_0 = h_1(\hat{x}, ms)$ and $y_0 = h_2(\hat{y}, ms)$. We then continue to evaluate the chains $x_{i+1} = h_1(x_i, m)$ and $y_{j+1} = h_2(y_j, m)$ up to length at most $L-3$. We hope to encounter $\bar{x}$ at some distance $q-1$ from x_0 and to encounter $\bar{y}$ at the same distance $q-1$ from y_0, where $((\bar{x}, \bar{y}), \bar{m}) \in S$ for some single block value $\bar{m}$. This gives the required $\hat{M} = ms\|[m]^{q-1}$.

The goal of this algorithm is very similar to one of the algorithm of Section 3.3, where the difference is the size of the set S, which essentially contained a single element[10] in Section 3.3, but can now have a larger size. This difference results in a complication that arises when the algorithm builds the functional graph of f_1 (and f_2), and has to keep track of distances of encountered nodes from all the 2^s nodes x (and y) that are in tuples of S (instead of merely keeping track of distances from a single node as in Section 3.3).

More formally, we define an *S-node* (for f_1) as a node x such that there exists a node y and a message block w such that $((x, y), w) \in S$. An S-node for f_2 in defined in a similar way. In order to avoid heavy update operations for the distances from all the S-nodes, we use distinguished points. Essentially, each computed chain is partitioned into intervals according to distinguished points, where each distinguished point stores only the distances to all the S-nodes that are contained in its interval up to the next distinguished point. Given a parameter $g_2 > g_1$, the algorithm for this phase is described below.

1. Develop (about) 2^{g_2} nodes in the functional graphs of f_1 (and f_2) (as specified in Section 2.5) with the following modifications.
   - Store only distinguished points for which the $n - g_2$ LSBs are zero.
   - Once an S-node is encountered, update its distance in the previously encountered distinguished point (which is defined with high probability[a]).

[10] One may ask why we did not compute a larger set S in Section 3.2. The reason for this is that it can be shown that in the previous case a set of size 1 is optimal.

   - Stop evaluating each chain once it hits a stored distinguished point.
   - The evaluated distinguished points for f_1 (f_2) are stored in the data structure G_1 (G_2).
2. For single-block values $ms = 0, 1, \ldots$, compute $x_0 = h_1(\hat{x}, ms)$ and $y_0 = h_2(\hat{y}, ms)$ and repeat the following step:
   (a) Compute chains $\overrightarrow{x}$ and $\overrightarrow{y}$ as specified below.
      - First, compute the chains in a standard way by evaluating the compression functions h_1 and h_2, until they hit stored distinguished points in G_1 and G_2 (respectively).
      - Then, allocate a table T_1 (T_2 for f_2) and continue traversing (only) the distinguished points of the chain (using the links in G_1 and G_2) up to depth $L-2$, while updating T_1 (T_2): for each visited distinguished point, add all its stored S-nodes to T_1 (T_2) with its distance from x_0 (y_0).
      - Once the maximal depth $L-2$ is reached, sort T_1 and T_2. Search for nodes $\bar{x}$ and $\bar{y}$ that were encountered at the same distance $q-1$ from x_0 and y_0 (respectively), such that $((\bar{x}, \bar{y}), \bar{m}) \in S$. If such $\bar{x} \in T_1$ and $\bar{y} \in T_2$ exist, return the message $\hat{M} = ms \| [m]^{q-1}$ and $\bar{m}$ (retrieved from S) as output. Otherwise (no such $\bar{x}$ and $\bar{y}$ were found), return to Step 2.

[a] Since $g_2 > g_1$, S-nodes are deeper iterates than distinguished points, and thus distinguished points are likely to be encountered in an arbitrary chain before an S-node.

The time complexity of Step 1 is about 2^{g_1}, as described in Section 2.5 (note that we always perform a constant amount of work per developed node). Compared to the second step of the algorithm of Section 3.3, S contains 2^s elements (instead of 1), and this reduces by the same factor the expected number of trials we need to execute in order to reach some $((\bar{x}, \bar{y}), \bar{m}) \in S$ in Step 2. Reusing the analysis of Section 3.3, the expected number of trials (executions of Step 2) is reduced from 2^{3g_1-n} to 2^{3g_1-n-s}.

The analysis of the complexity of Step 2.(a) is somewhat more involved compared to the corresponding step of Section 3.3. First, we estimate the expected number of nodes that we visit during the computation of a chain. Initially (as in Section 3.3), we compute about 2^{n-g_2} nodes until we hit stored distinguished points. Then, we continue by traversing (only) distinguished points up to depth of about L. The expected number of such points is $L \cdot 2^{g_2-n}$. Therefore, we expect to visit about $2^{n-g_2} + L \cdot 2^{g_2-n}$ nodes while computing a chain.

Finally, we need to account for all the S-nodes encountered while traversing the chains of depth L. Basically, there are 2^s S-nodes which are iterates of depth 2^{n-g_1}, (essentially) randomly chosen in Phase 2 out of about 2^{g_1} such deep iterates. As a result, the probability of such a deep iterate to be an S-node is about 2^{s-g_1} (while other nodes have probability 0). Therefore, while traversing chains of depth L, we expect to encounter at most $L \cdot 2^{s-g_1}$ S-nodes (which is a bound on the sizes of T_1 and T_2). Altogether, the expected time complexity of a single execution of Step 2.(a) is at most $2^{n-g_2} + L \cdot 2^{g_2-n} + L \cdot 2^{s-g_1}$.

The total time complexity of this phase is $2^{g_2} + 2^{3g_1-n-s} \cdot (2^{n-g_2} + L \cdot 2^{g_2-n} + L \cdot 2^{s-g_1}) = 2^{g_2} + 2^{3g_1-g_2-s} + L \cdot 2^{3g_1+g_2-2n-s} + L \cdot 2^{2g_1-n}$. We set $g_2 = 3g_1/2 - s/2$ which balances the first two terms and gives time complexity of

$$2^{3g_1/2-s/2} + L \cdot 2^{9g_1/2-2n-3s/2} + L \cdot 2^{2g_1-n}.$$

The time complexity evaluation of the full attack at the beginning of this section shows that for the optimal parameters of this attack, the extra two terms $L \cdot 2^{9g_1/2-2n-3s/2} + L \cdot 2^{2g_1-n}$ are negligible compared to the other terms in the complexity equation. In other words, the distinguished points method allowed us to resolve with no overhead the complication of keeping track of distances from the S-nodes.

5 Conclusions and Open Problems

In this paper we devised the first second preimage attack on the concatenation combiner and improved the preimage attack on the XOR combiner (due to Leurent and Wang) in case both hash functions use the Merkle-Damgård construction. Since both of our second preimage and preimage attacks on the concatenation and XOR combiners have higher complexities than the lower bounds ($2^n/L$ and $2^{n/2}$, respectively), it would be interesting to further improve them, and it particular, extend the second preimage attack to shorter messages. There are many additional interesting future work items such as extending our algorithms to combine more than two hash functions. Indeed, while it is easy to extend the expandable message to more than two hash functions with small added complexity, extending the random mapping techniques is less obvious. Yet another open problem is to improve preimage attack of Leurent and Wang on the XOR combiner in case only one of the functions uses the Merkle-Damgård construction.

References

1. E. Andreeva, C. Bouillaguet, O. Dunkelman, P.-A. Fouque, J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. New Second-Preimage Attacks on Hash Functions. *Journal of Cryptology (to appear)*, 2015.
2. E. Andreeva, C. Bouillaguet, O. Dunkelman, and J. Kelsey. Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård. In M. J. J. Jr., V. Rijmen, and R. Safavi-Naini, editors, *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers*, volume 5867 of *Lecture Notes in Computer Science*, pages 393–414. Springer, 2009.
3. E. Andreeva, C. Bouillaguet, P. Fouque, J. J. Hoch, J. Kelsey, A. Shamir, and S. Zimmer. Second Preimage Attacks on Dithered Hash Functions. In N. P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International*

Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings, volume 4965 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2008.

4. E. Biham and O. Dunkelman. A Framework for Iterative Hash Functions - HAIFA. *IACR Cryptology ePrint Archive*, 2007, 2007. http://eprint.iacr.org/2007/278.
5. D. Boneh and X. Boyen. On the Impossibility of Efficiently Combining Collision Resistant Hash Functions. In C. Dwork, editor, *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings*, volume 4117 of *Lecture Notes in Computer Science*, pages 570–583. Springer, 2006.
6. G. Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.
7. R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
8. I. Damgård. A Design Principle for Hash Functions. In Brassard [6], pages 416–427.
9. R. D. Dean. Formal Aspects of Mobile Code Security. PhD thesis, Princeton University, 1999.
10. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, 2008. https://www.ietf.org/rfc/rfc5246.txt.
11. I. Dinur and G. Leurent. Improved Generic Attacks against Hash-Based MACs and HAIFA. In Garay and Gennaro [16], pages 149–168.
12. O. Dunkelman and B. Preneel. Generalizing the Herding Attack to Concatenated Hashing Schemes. ECRYPT Hash Workshop, 2007.
13. M. Fischlin and A. Lehmann. Multi-property Preserving Combiners for Hash Functions. In R. Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008.*, volume 4948 of *Lecture Notes in Computer Science*, pages 375–392. Springer, 2008.
14. M. Fischlin, A. Lehmann, and K. Pietrzak. Robust Multi-Property Combiners for Hash Functions. *J. Cryptology*, 27(3):397–428, 2014.
15. A. O. Freier, P. Karlton, and P. C. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101, 2011. http://www.ietf.org/rfc/rfc6101.txt.
16. J. A. Garay and R. Gennaro, editors. *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*. Springer, 2014.
17. J. Guo, T. Peyrin, Y. Sasaki, and L. Wang. Updates on Generic Attacks against HMAC and NMAC. In Garay and Gennaro [16], pages 131–148.
18. M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, 1980.
19. J. J. Hoch and A. Shamir. Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In M. J. B. Robshaw, editor, *Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, volume 4047 of *Lecture Notes in Computer Science*, pages 179–194. Springer, 2006.
20. J. J. Hoch and A. Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, editors, *Automata,*

Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations, volume 5126 of *Lecture Notes in Computer Science*, pages 616–630. Springer, 2008.

21. A. Jha and M. Nandi. Some Cryptanalytic Results on Zipper Hash and Concatenated Hash. *IACR Cryptology ePrint Archive*, 2015:973, 2015.
22. A. Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In M. K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.
23. J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer [7], pages 474–490.
24. G. Leurent, T. Peyrin, and L. Wang. New Generic Attacks against Hash-Based MACs. In K. Sako and P. Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2013.
25. G. Leurent and L. Wang. The Sum Can Be Weaker Than Each Part. In E. Oswald and M. Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 345–367. Springer, 2015.
26. F. Mendel, C. Rechberger, and M. Schläffer. MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners. In M. Matsui, editor, *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*, volume 5912 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2009.
27. B. Mennink and B. Preneel. Breaking and Fixing Cryptophia's Short Combiner. In D. Gritzalis, A. Kiayias, and I. G. Askoxylakis, editors, *Cryptology and Network Security - 13th International Conference, CANS 2014, Heraklion, Crete, Greece, October 22-24, 2014. Proceedings*, volume 8813 of *Lecture Notes in Computer Science*, pages 50–63. Springer, 2014.
28. R. C. Merkle. One Way Hash Functions and DES. In Brassard [6], pages 428–446.
29. A. Mittelbach. Hash Combiners for Second Pre-image Resistance, Target Collision Resistance and Pre-image Resistance Have Long Output. In I. Visconti and R. D. Prisco, editors, *Security and Cryptography for Networks - 8th International Conference, SCN 2012, Amalfi, Italy, September 5-7, 2012. Proceedings*, volume 7485 of *Lecture Notes in Computer Science*, pages 522–539. Springer, 2012.
30. M. Nandi and D. R. Stinson. Multicollision attacks on some generalized sequential hash functions. *IEEE Transactions on Information Theory*, 53(2):759–767, 2007.
31. L. Perrin and D. Khovratovich. Collision Spectrum, Entropy Loss, T-Sponges, and Cryptanalysis of GLUON-64. In C. Cid and C. Rechberger, editors, *Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers*, volume 8540 of *Lecture Notes in Computer Science*, pages 82–103. Springer, 2014.
32. T. Peyrin and L. Wang. Generic Universal Forgery Attack on Iterative Hash-Based MACs. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014.*

Proceedings, volume 8441 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2014.

33. K. Pietrzak. Non-trivial Black-Box Combiners for Collision-Resistant Hash-Functions Don't Exist. In M. Naor, editor, *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 23–33. Springer, 2007.
34. B. Preneel. Analysis and design of cryptographic hash functions. PhD thesis, KU Leuven, 1993.
35. M. Rjasko. On Existence of Robust Combiners for Cryptographic Hash Functions. In P. Vojtáš, editor, *Proceedings of the Conference on Theory and Practice of Information Technologies, ITAT 2009, Horský hotel Kralova studna, Slovakia, September 25-29, 2009*, volume 584 of *CEUR Workshop Proceedings*, pages 71–76. CEUR-WS.org, 2009.
36. P. C. van Oorschot and M. J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.
37. X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. In V. Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.
38. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In Cramer [7], pages 19–35.

A Optimizing the Second Preimage Attack Using the Interchange Structure

The interchange structure [25] is built with a parameter that we denote by r. It consists of a starting state pair (as, bs) and two sets of 2^r internal states A for H_1 and B for H_2 such that: for any value $a \in A$ and any value $b \in B$, it is possible to efficiently construct a message $M_{a,b}$ (of length 2^{2r}) such that $(as, bs) \xrightarrow{M_{a,b}} (a, b)$. We now describe how to use the interchange structure as a black box in order to optimize the second preimage attack of Section 3.

The idea is to insert the interchange structure after the expandable message in order to reach $(\bar{x}, \bar{y})$ more efficiently in the third phase of the attack. More specifically, consider Step 2 in the attack of Section 3.3. There, we start computing from the state pair $(\hat{x}, \hat{y})$ and evaluate chains independently for each single-block value $ms = 0, 1, \ldots$. In the optimized attack, we build the interchange structure with the starting state pair $(\hat{x}, \hat{y})$ and sets of 2^r states A, B. We pick some single-block value ms and compute two sets of 2^r chains starting from the states of A and B. Our goal is to find any pair of states $a \in A$ and $b \in B$ such that $(a, b) \xrightarrow{ms\|[m]^{q-1}} (\bar{x}, \bar{y})$ for some $q \leq p - 2r - 1$. Therefore, we have $(\hat{x}, \hat{y}) \xrightarrow{M_{a,b}} (a, b) \xrightarrow{ms\|[m]^{q-1}} (\bar{x}, \bar{y})$, and we set $\hat{M} = M_{a,b}\|ms\|[m]^{q-1}$.

In the original attack, we evaluate and compare two chains for each execution of Step 2. In contrast, in the optimized attack, in each execution of modified

Step 2 we evaluate $2 \cdot 2^r$ chains and compare them in $2^r \cdot 2^r = 2^{2r}$ pairs. Consequently, the time complexity of (modified) Step 2 is increased by 2^r, but we have to execute it 2^{2r} less times. The complexity evaluation of the attack is rather technical as we need to balance several parameters and account for the message length 2^{2r} of $M_{a,b}$.

We sketch the complexity evaluation for short messages, where the length of $M_{a,b}$ is roughly equal to L, i.e., we have $L = 2^{\ell} = 2^{2r}$ or $r = \ell/2$. According to Section 3.3, the complexity of Phase 3 in the original attack is $2^{g_2} + 2^{3g_1 - n} \cdot 2^{n-g_2}$. Since building the interchange structure requires $2^{n/2+2r}$ time, the modified complexity is $2^{g_2} + 2^{3g_1-n-2r} \cdot 2^{n-g_2+r} + 2^{n/2+2r} = 2^{g_2} + 2^{3g_1-g_2-\ell/2} + 2^{n/2+\ell}$ (setting $r = \ell/2$). We balance the first two terms and obtain $g_2 = 3g_1/2 - \ell/4$, giving time complexity of $2^{3g_1/2-\ell/4} + 2^{n/2+\ell}$. Recall from Section 3.2 that the complexity of Phase 2 is $2^{2n-g_1-\ell}$. We balance the second and third phases by setting $g_1 = 4n/5 - 3\ell/10$, which gives time complexity of $2^{6n/5-7\ell/10}$ for small values of ℓ in which the term $2^{n/2+\ell}$ is negligible. Therefore, we obtain an attack faster than 2^n for messages of length $L > 2^{2n/7}$ (which is a small improvement compared to $L \geq 2^{n/3}$, obtained without this optimization).

For larger values of ℓ we need to redo the computation and account for the term $2^{n/2+2r}$ in the complexity evaluation. However, since the improvement over the original attack in not very significant, we do not give the details here.

B On the Number of Required Chain Evaluations in Section 3.3

In Section 3.3 we concluded that the probability of encountering $\bar{x}$ and $\bar{y}$ at the same distance in chains (of f_1 and f_2) evaluated from arbitrary starting points x_0 and y_0 is about 2^{n-3g_1}. If the trials of chain evaluations were independent, this would have led to the conclusion that we need to compute about 2^{3g_1-n} chains from different starting points in Phase 3. However, the trials are dependent as explained below.

Reconsider Lemma 1 in case we select more than $2^n/D^2$ starting points x_0^i such that the chains (of length D) evaluated from them contain in total more than $D \cdot 2^n/D^2 = 2^n/D$ nodes. In this case, a new chain of length D (evaluated from x_0) is very likely to collide with a previously evaluated node before colliding with the original chain (evaluated from x_0') due to the birthday paradox. After a collision of the new chain with a previously evaluated node, the outcome of the trial is determined and cannot be analyzed probabilistically. Of course, this does not imply that Lemma 1 does not hold for more than $2^n/D^2$ trials, but it does imply that in a formal proof we need to account for the dependency of the trials when applying this lemma with more than $2^n/D^2$ trials.[11]

A potential way to handle this is to select more targets[12] of the form $(\bar{x}, \bar{y})$ in Phase 2, which reduces the number of trials that we need to perform in Phase

[11] Note that in our case $D = 2^{n-g_1}$ or $2^{g_1} = 2^n/D$, so $2^{3g_1-n} = 2^{2n}/D^3 > 2^n/D^2$.

[12] In a similar way to the algorithm of Section 4.

3 (as we need to reach only one target). This will enable us to complete the theoretical analysis of the attack, but it does result in a performance degradation (although the attack remains faster than 2^n for a modified set of parameters).

However, based on simulations (described below), we strongly believe that indeed about 2^{3g_1-n} trials are required in Section 3.3 in order to reach arbitrary iterates $\bar{x}$ and $\bar{y}$ of depth 2^{n-g_1} at the same distance. We note that assumptions of this type are not uncommon in analysis of random functions. A recent and related conjecture was made in [17].

Experimental Results In our simulations we preformed hundreds of experiments on independent n-bit random mappings f_1 and f_2 for $n \in \{12, \dots, 28\}$ with several[13] values of $n/3 \leq g_1 < n/2$. In the beginning of each experiment, we chose different mappings f_1 and f_2 and arbitrary deep iterates $\bar{x}, \bar{y}$ of depth 2^{n-g_1}. Each experiment was carried out by performing $2 \cdot 2^{3g_1-n}$ trials (with chains evaluated from arbitrary different starting points), trying to hit $\bar{x}$ and $\bar{y}$ at the same distance (up to 2^{n-g_1}). The success rate of the experiments was more than 50% and did not drop as the value of n increased.

[13] Smaller values of n where chosen for smaller values of g_1, as these experiments are more expensive.