Generic Attacks on Hash Combiners

Zhenzhen Bao, Itai Dinur, Jian Guo, Gaëtan Leurent, Lei Wang

Generic Attacks on Hash Combiners

Zhenzhen Bao[1,2], Itai Dinur[3], Jian Guo[1], Gaëtan Leurent[4], and Lei Wang[5,6]

[1] Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
baozhenzhen10@gmail.com, guojian@ntu.edu.sg
[2] Strategic Centre for Research in Privacy-Preserving Technologies and Systems, Nanyang Technological University, Singapore
[3] Department of Computer Science, Ben-Gurion University, Israel
dinuri@cs.bgu.ac.il
[4] Inria, France
gaetan.leurent@inria.fr
[5] Shanghai Jiao Tong University, Shanghai, China
[6] Westone Cryptologic Research Center, Beijing, China
wanglei_hb@sjtu.edu.cn

Abstract. Hash combiners are a practical way to make cryptographic hash functions more tolerant to future attacks and compatible with existing infrastructure. A combiner combines two or more hash functions in a way that is hopefully more secure than each of the underlying hash functions, or at least remains secure as long as one of them is secure. Two classical hash combiners are the exclusive-or (XOR) combiner $\mathcal{H}_1(M) \oplus \mathcal{H}_2(M)$ and the concatenation combiner $\mathcal{H}_1(M) \| \mathcal{H}_2(M)$. Both of them process the same message using the two underlying hash functions in parallel. Apart from parallel combiners, there are also cascade constructions sequentially calling the underlying hash functions to process the message repeatedly, such as Hash-Twice $\mathcal{H}_2(\mathcal{H}_1(IV, M), M)$ and the Zipper hash $\mathcal{H}_2(\mathcal{H}_1(IV, M), \overleftarrow{M})$, where $\overleftarrow{M}$ is the reverse of the message M.

In this work, we study the security of these hash combiners by devising the best-known generic attacks. The results show that the security of most of the combiners is not as high as commonly believed. We summarize our attacks and their computational complexities (ignoring the polynomial factors) as follows:

1. Several generic preimage attacks on the XOR combiner:
   - A first attack with a best-case complexity of $2^{5n/6}$ obtained for messages of length $2^{n/3}$. It relies on a novel technical tool named Interchange Structure. It is applicable for combiners whose underlying hash functions follow the Merkle-Damgård construction or the HAIFA framework.
   - A second attack with a best-case complexity of $2^{2n/3}$ obtained for messages of length $2^{n/2}$. It exploits properties of functional graphs of random mappings. It achieves a significant improvement over

[§] This paper is a combination and extension of three conference papers [LW15,Din16,BWGG17].

the first attack but is only applicable when the underlying hash functions use the Merkle-Damgård construction.
- An improvement upon the second attack with a best-case complexity of $2^{5n/8}$ obtained for messages of length $2^{5n/8}$. It further exploits properties of functional graphs of random mappings and uses longer messages.

These attacks show a rather surprising result: regarding preimage resistance, the sum of two n-bit narrow-pipe hash functions following the considered constructions can never provide n-bit security.

2. A generic second-preimage attack on the concatenation combiner of two Merkle-Damgård hash functions. This attack finds second preimages faster than 2^n for challenges longer than $2^{2n/7}$ and has a best-case complexity of $2^{3n/4}$ obtained for challenges of length $2^{3n/4}$. It also exploits properties of functional graphs of random mappings.
3. The first generic second-preimage attack on the Zipper hash with underlying hash functions following the Merkle-Damgård construction. The best-case complexity is $2^{3n/5}$, obtained for challenge messages of length $2^{2n/5}$.
4. An improved generic second-preimage attack on Hash-Twice with underlying hash functions following the Merkle-Damgård construction. The best-case complexity is $2^{13n/22}$, obtained for challenge messages of length $2^{13n/22}$.

   The last three attacks show that regarding second-preimage resistance, the concatenation and cascade of two n-bit narrow-pipe Merkle-Damgård hash functions do not provide much more security than that can be provided by a single n-bit hash function.

Our main technical contributions include the following:

1. The interchange structure, which enables simultaneously controlling the behaviours of two hash computations sharing the same input.
2. The simultaneous expandable message, which is a set of messages of length covering a whole appropriate range and being multi-collision for both of the underlying hash functions.
3. New ways to exploit the properties of functional graphs of random mappings generated by fixing the message block input to the underlying compression functions.

Keywords: Hash function, Generic attack, Hash combiner, XOR combiner, Concatenation combiner, Zipper hash, Hash-Twice, (Second) Preimage attack

1 Introduction

A cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ maps arbitrarily long messages to n-bit digests. It is a fundamental primitive in modern cryptography and is among the main building blocks of many widely utilized cryptographic protocols and cryptosystems. There are three *basic* security requirements on a hash function $\mathcal{H}$:

– **Collision resistance**: It should be computationally infeasible to find a pair of different messages M and M' such that $\mathcal{H}(M) = \mathcal{H}(M')$.
– **Preimage resistance**: Given an arbitrary n-bit value V, it should be computationally infeasible to find any message M such that $\mathcal{H}(M) = V$.
– **Second-preimage resistance**: Given a challenge message M, it should be computationally infeasible to find any different message M' such that $\mathcal{H}(M) = \mathcal{H}(M')$.

As the birthday and brute-force attack requires $2^{n/2}$ and 2^n computations, respectively, to find a collision and a (second) preimage, a secure hash function is expected to provide the same level of resistance.

Unfortunately, widely deployed standards (such as MD5 and SHA-1) fail to provide the expected resistance because of cryptographic weaknesses [WYY05,WY05]. Moreover, Kelsey and Schneier have demonstrated a generic second-preimage attack against all hash functions based on the classical Merkle-Damgård construction (such as MD5, SHA-1, and SHA-2) when the challenge message is long [KS05]. As a result, countermeasures have been proposed in order to build more tolerant hash functions, and to protect oneself against future attacks, while keeping the same interface for compatibility. A practical way is to combine the outputs of two (or more) independent hash functions to provide better security in case one or even both hash functions are weak. In particular, this reasoning was used by the designers of SSL [FKK11] and TLS [DA99b], who combined MD5 and SHA-1 in various ways. More precisely, the Key Derivation Function of TLS v1.0/v1.1 uses a sum of HMAC-MD5 and HMAC-SHA-1.[7] The designers explain [DA99b], "In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure." Formally, we call the resulting construction a *hash function combiner* (or combiner for short). The goal of a hash combiner is to achieve *security amplification*, *i.e.*, the hash combiner has higher security than its underlying hash functions, or to achieve *security robustness*, *i.e.*, the hash combiner is secure as long as (at least) one of its underlying hash functions is secure.

There are two classical hash combiners, the concatenation combiner and the exclusive-or (XOR) combiner. Both of them process the same message using two (independent) hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ in parallel. Then, the former concatenates their outputs, $\mathcal{H}_1(M) \parallel \mathcal{H}_2(M)$, and the latter XORs them, $\mathcal{H}_1(M) \oplus \mathcal{H}_2(M)$. More generally[8], cryptographers have also studied cascade constructions of two (or more) hash functions, that is, to compute $\mathcal{H}_1$ and $\mathcal{H}_2$ in sequential order. Examples are Hash-Twice, $\mathcal{HT}(M) \triangleq \mathcal{H}_2(\mathcal{H}_1(IV, M), M)$[9], and the Zipper hash [Lis06], $\mathcal{H}_2(\mathcal{H}_1(IV, M), \overleftarrow{M})$, where $\overleftarrow{M}$ is the message with the same blocks as M but in reversed order. Such kinds of cascade constructions

[7] We note that this MD5/SHA-1 combiner has been replaced by primitives based on single hash function (*e.g.*, SHA-256) since TLS v1.2 [DR08].

[8] Here, we generalize the syntax of hash functions to also regard the initial value IV as an input parameter.

[9] The original specification of Hash-Twice is $\mathcal{HT}(M) \triangleq \mathcal{H}(\mathcal{H}(IV, M), M)$, which processes the same message twice using a single hash function as shown in [ABDK09].

are not strictly black-box hash combiners compared with the XOR combiner and the concatenated combiner. That is because the initial vector of the subsequent hash functions are not fixed as in their specifications. Instead, for such constructions, to get black-box access to the underlying hash functions, the initial vector of these hash functions is required to be an input parameter. However, apart from this point, other parts of the hash functions are all accessed as black boxes by the constructions. Considering that such constructions can be classified into the method of designing hash functions using multiple existing hash functions, and for the sake of conciseness, we regard these cascade constructions of hash functions as hash combiners in this paper. In the sequel, for these combiners, we call the computation of the first (resp. the second) hash function the first (resp. the second) computation pass (or simply, the *first pass*, resp. the *second pass*).

In this paper, we study the security of these hash combiners. We focus on combiners of *iterated* hash functions. Iterated hash functions commonly first pad and split the message M into message blocks of fixed length (*e.g.*, b-bit), *i.e.*, $M = m_1 \parallel m_2 \parallel \ldots \parallel m_L$. They then process message blocks by iteratively applying a series of compression function h_i. Those compression functions update an internal state, initialized with a public value IV, using the previous state value and the current message block, *i.e.*, $x_i = h_i(x_{i-1}, m_i)$. Finally, the internal state is updated by a finalization function which can be either the compression function or another independent function, and the output of the finalization function is outputted as the hash digest. For simplicity of description, we assume that the finalization function is the same as the compression function in the rest of the paper, but we stress that our attacks also work in a straightforward way with an independent finalization function. In this work, we mainly focus on hash functions whose internal state size is equal to its output size, which are known as "narrow-pipe" designs. We will discuss the applicability of our proposed attacks on "wide-pipe" designs whose internal state size is (not much) larger than its output size at the end of the paper. In particular, we consider hash functions following the classical Merkle-Damgård construction [Dam89,Mer89] and the more general HAIFA framework [BD07], for which we naturally take the length padding mentioned next as the message padding scheme. The Merkle-Damgård construction (MD) applies the same compression function h in all iterations (see Fig. 1) and adds a padding of the message length to the final message block (known as length padding, or Merkle-Damgård strengthening). The HAIFA framework is similar to the MD construction but uses extra inputs to the compression function with the number of message bits hashed so far and a salt value; this is equivalent to using a different compression function for each block. The primary goal of that construction is to thwart some narrow-pipe attacks (*e.g.*, Kelsey and Schneier's long message second-preimage attack on MD hash [KS05]). HAIFA framework is formally defined as follows:[10]

$$x_0 = IV \qquad x_i = h(x_{i-1}, m_i, \#bits, salt) \qquad \mathcal{H}(M) = h(x_{L-1}, m_L).$$

[10] For simplicity of description, we omit the computation of the initial value $IV_n = h(IV, n, 0, 0)$, which is used to support variable hash size in the specification of HAIFA in [BD07]. This does not influence the attacks.

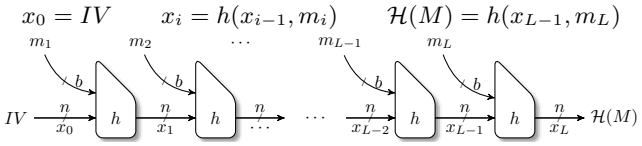


Fig. 1: Narrow-pipe Merkle-Damgård hash function

1.1 Related Works

Combiners have been studied in several settings, including generic attacks and security proof. For generic attacks, the compression functions are modelled as random functions to devise attacks that do not use any weakness of the compression function. Thus, they provide upper bounds on the security of the combiners.

Security proof, which is more theoretical work, focuses on the notion of *robustness* and *security amplification*. A robust combiner is secure regarding property α as long as one of the underlying hash functions is secure regarding α. Lines of research for those security notions include the study of advanced combiners in [Her05,CRS+07,FL07,FL08,FLP08,Her09,Leh10,FLP14]. A series of studies on the minimum output length of robust combiners have shown that robust combiners for collision resistance and preimage resistance cannot have an output length significantly shorter than the sum of the output length of the underlying hash functions [BB06,Pie07,Pie08,Rja09] (more recent works include [MP14,Mit12]).

There are also some works that assume that the compression function is a *weak* random oracle (*i.e.*, the attacker is given additional interfaces to receive random preimages of the compression functions) and prove that some constructions are still secure in this model [Lis06,HS08] or provide efficient attacks on the combiners by exploiting weakness of the underlying hash functions [CJ15,JN15].

Analysis of the concatenation combiner. The concatenation combiner $\mathcal{H}_1(M) \,\|\, \mathcal{H}_2(M)$ (see Fig. 2) is probably the most well-known and most studied hash combiner. This combiner was described already in 1993 [Pre93].

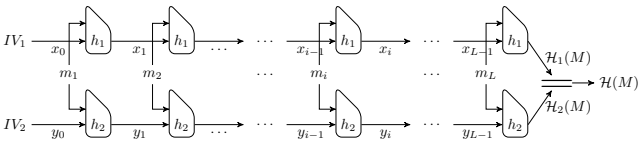


Fig. 2: The concatenation combiner

Generic attacks. In 2004, Joux [Jou04] described surprising attacks on the concatenation of two narrow-pipe iterated hash functions using multi-collisions: while the output size is $2n$ bits, the concatenation combiner merely provides at most $n/2$-bit security for collision resistance and n-bit security for preimage resistance[11] (see Sect. 2.1 for a description). In particular, the concatenation combiner is not security-amplifying (it does not increase collision and preimage resistance).

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 MD construction is less than 2^n), a natural question is whether there exists a second-preimage attack on the concatenation combiner of MD 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 [DP07]. In fact, although the works of [Jou04,KS05] have attracted a significant amount of follow-up research on countermeasures against second-preimage attacks (such as Hash-Twice or dithered hash) and attacks that break them [ABD+16,ABDK09,ABF+08], there has been no progress concerning second-preimage attacks on the basic concatenation combiner. In this paper, we try to provide an answer to this question by devising the first second-preimage attack on the concatenation combiner of MD hash functions, which is faster than 2^n.

Security proof. From the theoretical side, the concatenation combiner is robust for collision resistance, *e.g.*, a collision $\mathcal{H}_1(M) \parallel \mathcal{H}_2(M) = \mathcal{H}_1(M') \parallel \mathcal{H}_2(M')$ implies $\mathcal{H}_1(M) = \mathcal{H}_1(M')$ and $\mathcal{H}_2(M) = \mathcal{H}_2(M')$.

Hoch and Shamir [HS08] evaluated the security of the concatenation combiner with two weak hash functions. More precisely, the two hash functions are narrow-pipe MD, and the compression functions are modelled as weak random oracles (as defined by Liskov [Lis06]), *i.e.*, the adversary is given additional interfaces to receive (random) preimages of the compression functions. They have proven that in this model, the concatenation combiner is still indifferentiable from a random oracle with $n/2$-bit security, implying (at least) the same security bound for collision resistance and preimage resistance. The bound is matched by Joux's attack for collisions, but there is a gap with Joux's attack for preimages, with complexity 2^n, which might be interesting to investigate further.

Analysis of dedicated instantiations. Mendel *et al.* analysed some dedicated instantiations of the concatenation combiner [MRS09], in particular using the hash function MD5. We omit the details and refer interested readers to [MRS09].

Analysis of the XOR combiner. The XOR combiner $\mathcal{H}_1(M) \oplus \mathcal{H}_2(M)$ (see Fig. 3) has received less analysis.

[11] The attacks essentially only require one of the functions to be iterated.

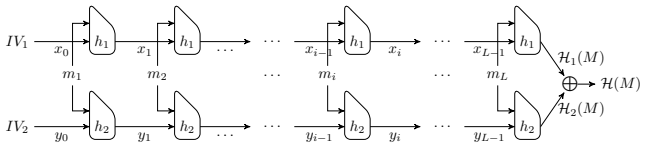


Fig. 3: The XOR combiner

Generic attacks. To the best of our knowledge, no preimage attacks have been shown against the XOR combiner. Therefore, the preimage security of the XOR combiner against generic attacks is still an open problem and will be one of the main topics of our work.

Security proof. Theoretically, the XOR combiner is robust concerning PRF (Pseudo-Random Function) and MAC (Message Authentication Code) in the black-box reduction model [Leh10]. Since the XOR combiner is length-preserving, from the conclusions regarding the minimum output length of robust combiners, it is not robust for collision resistance and preimage resistance. However, the work of Hoch and Shamir [HS08] actually proves the security of the XOR combiner as an intermediate result: it is also indifferentiable from a random oracle up to $2^{n/2}$ queries in the weak random oracle model. In particular, this proves there are no generic attacks with complexity less than $2^{n/2}$. For collision resistance, the bound is tight, since it is matched with the generic birthday attack bound. On the other hand, for preimage resistance, there exists a gap between the $n/2$-bit proven bound and the n-bit expected ideal security bound. Note that the non-robustness result regarding preimage security does not imply that the XOR of two concrete hash functions is weak, and the simplicity and short output of this construction still make it quite attractive.

Analysis of Hash-Twice. Hash-Twice is a folklore hash construction that hashes a (padded) message twice, with the output of the first hash value as the value of the initialization vector of the second hash. In its original definition [ABDK09], the two underlying hash functions are identical, *i.e.*, $\mathcal{HT}(M) \triangleq \mathcal{H}(\mathcal{H}(IV, M), M)$; here, we consider a generalized version, where the underlying hash functions are independent, *i.e.*, $\mathcal{HT}(M) \triangleq \mathcal{H}_2(\mathcal{H}_1(IV, M), M)$ (see Fig. 4).

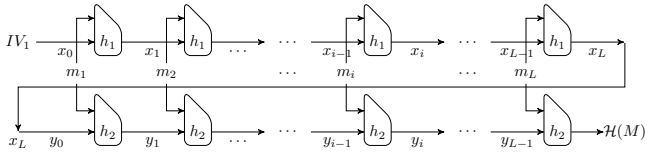


Fig. 4: The Hash-Twice

Generic attacks. Towards the three basic security requirements, a second-preimage attack on Hash-Twice ($\mathcal{HT}(M) \triangleq \mathcal{H}(\mathcal{H}(IV, M), M)$) has been published by An-

dreeva *et al.* in [ABDK09]. The attack is based on a herding attack, which exploits the diamond structure originally used in the herding attack on a single hash function [KK06] (see Sect. 2.3 for an introduction). The complexity of the attack is approximately $2^{(n+t)/2} + 2^{n-\ell} + 2^{n-t}$, where 2^t is the width of the diamond structure, and 2^ℓ is the length of the challenge.

Security proof. To the best of our knowledge, there is no published formal proof regarding the security of Hash-Twice. However, we can claim that they are at least as secure as the original functions: a generic collision attack requires at least $2^{n/2}$ (because we need a collision in one of the compression functions); a preimage attack requires at least 2^n (because we need a preimage for the finalization function); a second-preimage requires at least $2^{n/2}$ (because it implies a collision).

Analysis of the Zipper hash. The Zipper hash has been proposed with the goal of constructing an ideal hash function from weak ideal compression functions (by "weak ideal", it means that the compression function is vulnerable to strong forms of attack but is otherwise random). Similar to Hash-Twice, it cascades two independent hash functions evaluating the same (padded) message. The difference is that the second hash processes the message blocks in reverse order, *i.e.*, $\mathcal{ZH} \triangleq \mathcal{H}_2(\mathcal{H}_1(IV_1, M), \overleftarrow{M})$ (see Fig. 5). Note that the messages are first padded by a padding scheme and split into message blocks, and then they are processed in forward and reverse order sequentially. Thus, the padded message block m_L is processed at the end of the first hash computation and at the beginning of the second hash computation, *i.e.*, in the middle of the whole processing procedure. The padding scheme of Zipper was specified to be any injective function of the message [Lis06]. In this paper, and as for all other combiners, we take the length padding of the MD construction as the padding scheme.

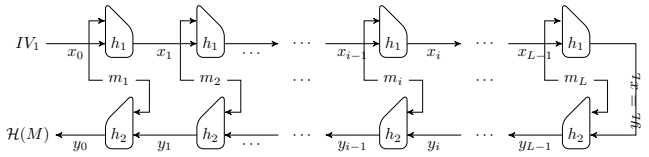


Fig. 5: The Zipper hash

Generic attacks. To the best our knowledge, no generic attacks on the Zipper hash regarding the three basic security notions have been shown. However, there are a number of works that consider other security notions, such as multi-collision, herding attack or attacks assuming weak compression functions. Examples include [NS07,HS06,ABDK09,CJ15,JN15], some of which also consider the corresponding security of Hash-Twice.

Security proof. Zipper hash is proved ideal in the sense that the overall hash function is indistinguishable from a random oracle (up to the birthday bound) when instantiated with weak ideal compression functions. More precisely, its provable security is $2^{\min(b,n)/2}$ for collision resistance and $2^{\min(b,n)}$ for (second-) preimage resistance, where b is the size of the message block and n is the size of the internal state and the hash value (considering "narrow-pipe" design).

1.2 Our Results

In this work, we study the upper bound of the security of these hash combiners by devising the best-known generic preimage attacks and second-preimage attacks with a long challenge. We do not assume any weakness of the compression functions, *i.e.*, they are accessed as black boxes in our attacks. The compression functions are chosen uniformly at random from all $n+b$ to n-bit functions, which implies that our analysis applies to most compression functions. Table 1 summarizes the updated security status of various hash combiners after integrating our new results. It shows that the security of most combiners is not as high as commonly believed. Regarding certain basic security requirements, these combiners of two (or even more) n-bit hash functions fail to provide more (or even the same) security than that provided by a single n-bit ideal hash function.

Next, we briefly introduce our main attacks on combiners of two narrow-pipe hash functions and their computational complexities (ignoring the polynomial factors). In addition to the attacks summarized next, we also present improved but more complex attacks and apply our techniques to attack combiners of more than two hash functions.

Preimage attacks on the XOR combiner. We present several generic attacks:

- *A first attack with a best-case complexity of* $2^{5n/6}$ *computations obtained for messages of length* $2^{n/3}$. This attack involves a meet-in-the-middle procedure enabled by building a novel technical tool named *interchange structure*. This structure consists of a sequence of basic building modules named *switches* among different hash computation lanes such that it breaks the pairwise dependency between the internal states of these hash computations on the same message. This attack is applicable for the XOR combiner with underlying hash functions following a wide range of iterated constructions (*e.g.*, the classical MD construction and the more general HAIFA framework) and for the Cryptophia's short combiner [Mit13,MP14].
- *A second attack with a best-case complexity of* $2^{2n/3}$ *computations obtained for messages of length* $2^{n/2}$. This attack also involves a meet-in-the-middle procedure, but instead of using the interchange structure, it exploits properties of functional graphs of random mappings. The random mappings are generated by fixing the message block input to the underlying compression functions. The functional graphs of these random mappings are formed by

successive iteration of the mappings, whose nodes are all possible values of input/output and whose edges are all from preimages to images. We exploited special nodes that are images of a large number of iterations of the mappings. We named them *deep iterates* because they are located at deep strata in the functional graphs. By exploiting such deep iterates, the variability of the number of iterations provides extra freedom to find a linking message fragment mapping a pair of starting states to a predefined pair of states. When using this freedom, to overcome the hurdle set by the length padding, we construct a structure named *simultaneous expandable message*, which is a set of messages of length covering a whole appropriate range and being multi-collisions for both of the underlying hash functions. This attack achieves a trade-off of $2^n \cdot L^{-2/3}$ between the maximal allowed message length L and the time complexity of attack. This improves the trade-off of $2^n \cdot L^{-1/2}$, obtained by the first interchange-structure-based attack. On the other hand, it only applies when both underlying hash functions combined use the MD construction.

– *An improvement upon the second attack with a best-case complexity of* $2^{5n/8}$ *obtained for messages of length* $2^{5n/8}$. In this attack, we exploit more special nodes in functional graphs of random mappings, which are called cyclic nodes, and we utilize a technique named *multi-cycles*. Linking pairs of states through cyclic nodes and allowing to loop around the cycles makes the attack more efficient. The complexity improvement of the resulting preimage attack is $2^{n/24}$. We point out that this attack has the limitation that the length of the message must be at least $2^{n/2}$ blocks. Therefore, its practical impact is limited and its main significance is theoretical. This attack shows that the security level regarding preimage resistance of the XOR combiner is quite close to the provable security level $2^{n/2}$, which is also the level of collision resistance.

These attacks on the XOR combiner show a rather surprising result – regarding preimage resistance, the sum of two n-bit iterated hash functions can never provide n-bit security. In general, the sum is *weaker* than each of the two hash functions.

Second-preimage attack on the concatenation combiner. We describe the first generic second-preimage attack faster than 2^n on the concatenation combiner of two MD hash functions. The general framework follows that of the long message second-preimage attack on a single MD hash [KS05]. It is faster than 2^n computations by overcoming two main challenges. The first is to overcome the length padding. We solve this by using the *simultaneous expandable message*. The second is to speed up the connection from a crafted message to the challenge message on chaining states. In the case of hash combiners, one must connect to the challenge message on *a pair of* n-bit states, while in the second-preimage attack on a single MD hash, one needs only to connect on a single n-bit state. Thus, the attempt is essentially to reach a $2n$-bit state (in a set of L states, where L is the message length in blocks, and thus $L < 2^n$) faster than

2^n computations. We solve this by exploiting again deep iterates in functional graphs. In this attack, we choose a pair of deep iterates as target chaining states, such that the connection from our crafted message to the challenge message is more efficient. Indeed, this attack is closely related with our deep-iterates-based preimage attack on the XOR combiners.

We obtain a trade-off between the complexity of the attack and the length of the challenge message. This second-preimage attack is faster than 2^n for input messages of length at least[12] $2^{2n/7}$. The best-case complexity[13] is $2^{3n/4}$, obtained for (very) long challenges of length $2^{3n/4}$. Again, due to these constraints, the practical impact of our second-preimage attack is limited and its main significance is theoretical. Namely, it shows that regarding second-preimage resistance, the concatenation of two n-bit MD hash functions is not as strong as a single n-bit ideal hash function.

Second-preimage attack on the Zipper hash. We show the first generic second-preimage attack on the Zipper hash. This attack combines multiple tools, including Joux's multi-collision, the simultaneous expandable message, deep iterates and multi-cycles in functional graph of random mappings. The general framework is similar to that of above ones on combiners of MD hashes. However, some special specifications of the Zipper hash allow the attacker to choose an optimal configuration on message length for the attack, and to launch a more efficient meet-in-the-middle connecting procedure in the attack. The best-case complexity of this attack is $2^{3n/5}$, obtained for challenge message of length $2^{2n/5}$. This result shows that combination of two MD hash functions using a Zipper can be vulnerable to second-preimage attack with long challenges.

Second-preimage attack on Hash-Twice. We give an improved second-preimage attack on Hash-Twice. This attack also combines multiple tools including Joux's multi-collision, the diamond structure, the interchange structure, the simultaneous expandable message, deep iterates and multi-cycles in functional graphs. Like all our previous functional-graph-based attacks, it improves a previous attack from [ABDK09] because of the efficiency brought by exploiting the special nodes in the functional graphs. The best-case complexity of this attack is $2^{13n/22}$, obtained for challenge message of length $2^{13n/22}$. This attack shows that regarding second-preimage resistance, hashing a message twice using two Merkle-Damgård hash functions does not provide much more security compared with hashing the message only once.

Finally, we highlight the technical interests of this paper. We believe that the tools introduced in this paper – the interchange structure, the simultaneous expandable message, deep iterates and multi-cycles in random functional

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

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

graphs – are important technical advances and will hopefully have further applications or lead to new technical developments in related settings. Particularly, we point out that we can use the interchange structure in order to optimize the complexity of functional-graph-based attacks on the XOR combiner, concatenation combiner, and Hash-Twice. 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.

$\mathcal{H}$	Collision		Preimage		2nd Preimage	(challenge length)
Ideal	$2^{0.5n}$	Birthday	2^n	Brute-force	2^n	Brute-force
HAIFA	-	-	-	-	-	-
MD	-	-	-	-	$2^{0.5n}$	[KS05] DS ($2^{0.5n}$)
Provable	$2^{0.5n}$		2^n		$2^{0.5n}$	

$\mathcal{H}_1 \oplus \mathcal{H}_2$	Collision		Preimage		2nd Preimage	
Ideal	$2^{0.5n}$	Birthday	2^n	Brute-force	2^n	Brute-force
HAIFA	-	-	$2^{0.833n}$	[Sect. 3.1] IS		
MD	-	-	$2^{0.833n}$	[Sect. 3.1] IS		
	-	-	$2^{0.667n}$	[Sect. 4.1] SEM+FGDI		
	-	-	$2^{0.643n}$	[Sect. 4.3] SEM+FGDI+IS		
	-	-	$2^{0.625n}$	[Sect. 5.2] SEM+FGMC		
	-	-	$2^{0.612n}$	[Sect. 5.3] SEM+FGMC+IS		
Provable	$2^{0.5n}$	[HS08]	$2^{0.5n}$	[HS08]	$2^{0.5n}$	[HS08]

$\mathcal{H}_1 \parallel \mathcal{H}_2$	Collision		Preimage		2nd Preimage	(challenge length)
Ideal	2^n	Birthday	2^{2n}	Brute-force	2^{2n}	Brute-force
HAIFA	$2^{0.5n}$	[Jou04] MC	2^n	[Jou04] MC		
MD	$2^{0.5n}$	[Jou04] MC	2^n	[Jou04] MC		
	-	-	-	-	$2^{0.75n}$	[Sect. 6.1] SEM+FGDI ($2^{0.75n}$)
	-	-	-	-	$2^{0.735n}$	[Sect. 6.3] SEM+FGDI+IS ($2^{0.735n}$)
Provable	$2^{0.5n}$	[HS08]	$2^{0.5n}$	[HS08]	$2^{0.5n}$	[HS08]

Zipper	Collision		Preimage		2nd Preimage	(challenge length)
Ideal	$2^{0.5n}$	Birthday	2^n	Brute-force	2^n	Brute-force
MD	-	-	-	-	$2^{0.625n}$	[Sect. 7] SEM+FGDI [$L < 2^{n/2}$] ($2^{0.375n}$)
	-	-	-	-	$2^{0.6n}$	[Sect. 7] SEM+FGMC [$L > 2^{n/2}$] ($2^{0.4n}$)
Provable	$2^{0.5r}$	[Lis06]	$2^{0.5r}$	[Lis06]	$2^{0.5r}$	[Lis06]

Hash-Twice	Collision		Preimage		2nd Preimage	(challenge length)
Ideal	$2^{0.5n}$	Birthday	2^n	Brute-force	2^n	Brute-force
MD	-	-	-	-	$2^{0.667n}$	[ABDK09] EM+MC+DS ($2^{0.333n}$)
	-	-	-	-	$2^{0.612n}$	[Sect. 8.3] SEM+IS+DS+FGDI ($2^{0.5n}$)
	-	-	-	-	$2^{0.591n}$	[Sect. 8.2] SEM+IS+DS+FGMC ($2^{0.591n}$)
Provable	$2^{0.5}$	-	2^n	-	$2^{0.5n}$	-

r : $\min(b, n)$ L : 2nd preimage length MC: Joux's Multi-Collisions
EM: Kelsey and Schneier's Expandable Message DS: Diamond Structure
IS: Interchange Structure SEM: Simultaneous Expandable Message
FGDI: Deep-Iterates in Functional Graph FGMC: Multi-Cycles in Functional Graph
All computational complexities of the attacks are presented ignoring the polynomial factors.

Table 1: Security status of various combiners of two narrow-pipe hashes

1.3 Notations and Roadmap of the Rest of Paper

Notations. We summarize below notations shared across various attacks.

$\mathcal{H}_1$,	$\mathcal{H}_2$	: Underlying hash functions in a hash combiner
IV_1,	IV_2	: Initialization vectors of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively
h_1,	h_2	: Compression functions of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively
h_1^*,	h_2^*	: Compression functions iterated over several blocks (in particular, $\mathcal{H}_i(\boldsymbol{M}) = h_i^*(IV_i, \boldsymbol{M})$ for $i \in \{1, 2\}$)
V		: Targeted image
m		: Message block
M		: Message chunk
$[m]^q$		: Message chunk formed by concatenating q message blocks m, with $[m] = [m]^1$
$M_{\|q}$		: Message chunk with q message blocks
$\boldsymbol{M} = m_1 \| \ldots \| m_L$		: Target message or computed preimage (of L massage blocks)
L		: Length of $\boldsymbol{M}$ (measured in the number of blocks)
ℓ		: The binary logarithm of the length of the message $\boldsymbol{M}$, *i.e.*, denote $L = 2^\ell$
$\boldsymbol{M}'$		: Computed second preimage
L'		: Length of $\boldsymbol{M}'$ (measured in the number of blocks)
$a_0, \ldots, a_L$		: Sequence of internal states computed during the invocation of h_1 on $\boldsymbol{M}$, $a_0 = IV_1$
$b_0, \ldots, b_L$		: Sequence of internal states computed during the invocation of h_2 on $\boldsymbol{M}$, $b_0 = IV_2$
x,	y	: Computed internal states
$\vec{a}_j$,	$\vec{b}_k$	: Chains of internal states for $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. $\vec{a}_j$ denotes a generic chain, while $\vec{a}_{j_0}$ denotes a particular chain
A_j,	B_k	: End points (final states) of the chains
n		: Bit-size of the output of each underlying hash function ($\mathcal{H}_1$ and $\mathcal{H}_2$); In addition, we suppose their compression functions h_1 and h_2 have n-bit internal states (*i.e.*, suppose the underlying hash functions are narrow-pipe)
b		: Bit-size of a message block
N		: The considered random mappings are from a finite N-set domain to a finite N-set range, and $N = 2^n$
$\mathcal{FG}_{f_1}$,	$\mathcal{FG}_{f_2}$	: The functional graphs of random mappings $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$ generated by fixing a message block m to the compression functions
$\mathcal{M}$		: A set of messages
$\mathcal{M}_{\texttt{MC/EM/SEM/DS/IS}}$		: The set of messages in a standard Joux's multi-collision or an expandable message or a simultaneous expandable message or a diamond structure or an interchange structure

$x \xrightarrow{m} x'$, $x \xrightarrow{\hat{M}} x'$	:	We say that m (resp. $\hat{M}$) maps state x to state x' if $x' = h(x, m)$ (resp. $x' = h^*(x, \hat{M})$) and denote this by $x \xrightarrow{m} x'$ (resp. $x \xrightarrow{\hat{M}} x'$, the compression function h is clear from the context).
$h_{[m]}$	:	An n-bit random mapping obtained by feeding an arbitrary fixed message block m into a compression function h with n-bit state.
$\tilde{O}$	:	Soft-O, is used as a variant of big O notation that ignores logarithmic factors. Thus, $f(n) \in \tilde{O}(g(n))$ is shorthand for $\exists k : f(n) \in O(g(n) \log^k g(n))$.

Roadmap. Section 2 exhibits preliminaries including generic tools and known attacks on hash constructions. Particularly, sections 2.5, 2.6, 2.7 demonstrate those new tools – the interchange structure, the simultaneous expandable message, deep iterates and multi-cycles in random functional graphs – which make the presented attacks possible. Sections 3, 4, and 5 illustrate preimage attacks on the XOR combiner using interchange structure, deep iterates, and multi-cycles in functional graphs, respectively. Sections 6, 7, and 8 describes the second-preimage attacks on the concatenation combiner, Zipper hash, and Hash-Twice, respectively. Within the description of each attack, we provide an overview followed by detailed steps. In Section 9, we discuss more applications and extensions of the proposed attacks, including applications to combiners of wide-pipe hash functions and extensions to the combination of more than two hash functions. Section 10 summarizes attacks presented in this paper and discusses open problems.

2 Preliminaries

In this section, we introduce the technical tools and general concepts used in our attacks. Those tools briefly introduced in Sect. 2.1 through Sect. 2.4 are all existing and well-known tools. Those tools described in detail in Sect. 2.5 through Sect. 2.7 are our new tools exploited in different attacks.

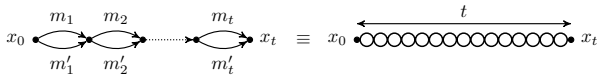
2.1 Joux's Multi-Collision (MC) and Its Applications in Attacks on the Concatenation Combiner [Jou04]

In 2004, Joux introduced multi-collisions on narrow-pipe Merkle-Damgård hash functions. Given a hash function $\mathcal{H}$, a multi-collision refers to a set of messages $\mathcal{M} = \{M_1, M_2, \ldots\}$ whose hash digests are all the same, *i.e.*, $\mathcal{H}(M_i) = \mathcal{H}(M_j)$ for any pair $M_i, M_j \in \mathcal{M}$. The computational complexity of generic brute-force search increases exponentially when the target size $|\mathcal{M}|$ increases; more precisely, it is approximately $2^{(|\mathcal{M}|-1)\cdot n/|\mathcal{M}|}$. Utilizing the iterative nature of Merkle-Damgård structure, Joux's algorithm (see Alg. 1, whose pseudo-code is given in Appendix A) is able to find multi-collision of size 2^t with a complexity of $t \cdot 2^{n/2}$, *i.e.*, a complexity not much greater than that of finding a single collision.

Algorithm 1: Building a 2^t-Joux's Multi-Collision

Require: Given an iterated hash function $\mathcal{H}$ with a compression function h, and an initial value x_0.

1. Initialize $\mathcal{M}_{\texttt{MC}}$ as a data structure of pairs of message blocks
2. For $i = 1, \ldots, t$:
   (a) Find a pair of message blocks (m_i, m_i') such that $h(x_{i-1}, m_i) = h(x_{i-1}, m_i') = x_i$. This can be done with a complexity of $2^{n/2}$ due to birthday paradox.
   (b) Append (m_i, m_i') to $\mathcal{M}_{\texttt{MC}}$
3. Output $(x_t, \mathcal{M}_{\texttt{MC}})$



It is trivial to see the message set $\mathcal{M} = \{\overline{m}_1 \parallel \overline{m}_2 \parallel \cdots \parallel \overline{m}_t \mid \overline{m}_i = m_i \text{ or } m_i' \text{ for } i = 1, 2, \ldots, t\}$ forms a multi-collision of size 2^t, and the overall complexity is $\mathcal{O}(t \cdot 2^{n/2})$. Moreover, a data structure $\mathcal{M}_{\texttt{MC}}$ of t pairs of message blocks can fully define the set $\mathcal{M}$ of 2^t colliding messages.

With Joux's multi-collision at hand, one can immediately deploy a collision attack and a preimage attack on concatenation combiner with complexities $n \cdot 2^{n/2}$ and $n \cdot 2^n$, respectively. The collision attack goes as follows: first, build a $2^{n/2}$-Joux's multi-collision for one of the underlying hash function (iterated), and then exploit the messages in the structure to launch a birthday attack for the other hash function to find a collision among the outputs. The preimage attack follows a similar framework (see [BGW18] for an illustration and Joux's original paper [Jou04] for more details).

Since its invention, Joux's multi-collisions have been employed in numerous cryptanalysis of hash functions, including the following most relevant ones and works such as [HS06,NS07].

2.2 Expandable Message (EM) and the Long Message Second-Preimage Attack [KS05]

In [DA99a], 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 challenge message $\boldsymbol{M} = m_1 \parallel m_2 \parallel \ldots \parallel m_L$, the attacker computes the sequence of internal states $a_0, a_1, \ldots, a_L$ generated during the invocation of the compression function on $\boldsymbol{M}$. A simplified attack would now start from the state $x_0 = IV$ and evaluate the compression function with arbitrary message blocks until a collision $h(x_0, m) = a_p$ is found for some message block m and index p. The attacker can now append the message suffix

$m_{p+1}\|\ldots\|m_L$ to m, hoping to obtain the target hash value $\mathcal{H}(\boldsymbol{M})$. However, this approach does not work due to the final padding of the message length, which will be different if the message prefixes are of different lengths. The solution of Dean was to compute an *expandable message* that 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 $M_{\|\kappa}$ of κ blocks that maps x_0 to $\hat{x}$. Thus, the algorithm first finds a collision $h(\hat{x}, m) = a_p$, and the second preimage is computed as $M_{\|p-1}\|m\|m_{p+1}\|\ldots\|m_L$. The assumption that it is easy to find fixed points in the compression function is used in efficient construction of the expandable message.

In [KS05], 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 firstly picking an arbitrary prefix of size 2^{i-1} of m'_i denoted by $\hat{m}_i$, say $[0]^{2^{i-1}}$, computing $h(x_{i-1}, \hat{m}_i) = x'_i$ and then looking for a collision $h(x_{i-1}, m_i) = h(x'_i, \check{m}_i)$ using a final block $\check{m}_i$ (namely, $m'_i = \hat{m}_i \,\|\, \check{m}_i$) (see Fig. 6).

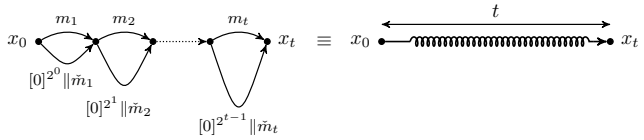


Fig. 6: The expandable message and its condensed representation [JN15]

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 are in the interval $[t, 2^t+t-1]$ (such a message structure is denoted as a $(t, 2^t + t - 1)$-expandable message). A message of length $t \le \kappa \le 2^t + t - 1$ 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). In the sequel, we denote this type of expandable message by $\mathcal{M}_{\texttt{EM}}$. Given that the challenge message $\boldsymbol{M}$ is of $L \le 2^{n/2}$ blocks, the construction of the expandable message in the first phase of the attack requires less than $n \cdot 2^{n/2}$ computations, while obtaining the collision with one of the states computed during the computation of $\boldsymbol{M}$ requires approximately $1/L \cdot 2^n$ computations according to the birthday paradox.

2.3 Diamond Structure (DS) [KK06]

Like Joux's multi-collisions and expandable message, the diamond structure is also a type of multi-collision. The difference is that instead of mapping a common starting state to a final state, each message in a diamond maps a different state

to a final state. A 2^t-diamond contains 2^t specially structured messages mapping 2^t starting states to a final state, and it forms a complete binary tree of depth t. The 2^t starting states are leaves, and the final state is the root. A 2^t-diamond can be built by launching several collision attacks requiring about $\sqrt{t} \cdot 2^{\frac{(n+t)}{2}}$ messages and $n \cdot \sqrt{t} \cdot 2^{\frac{(n+t)}{2}}$ computations in total [BSU12]. In the sequel, we denote the set of messages in a diamond by $\mathcal{M}_{\mathtt{DS}}$. The diamond was primarily invented by Kelsey and Kohno to devise herding attacks against MD hash functions [KK06], in which the attacker first commits to the digest value of a message using the root of his diamond and later "herds" any given prefix of a message to his commitment by choosing an appropriate message from his diamond as the suffix. Later, Andreeva *et al.* successfully exploited it to launch herding and/or second-preimage attack beyond MD hash constructions, such as the dithered hash, Hash-Twice, the Zipper hash, and hash trees [ABF+08,ABDK09,ABD+16]. Concretely, the second-preimage attack on Hash-Twice in [ABDK09] leverages techniques in herding attack and techniques in the above-mentioned second-preimage attack. One key point of this attack is that it builds a long Joux's multi-collision in the first pass, exploits messages in this multi-collision to build a diamond structure in the second pass, and finally uses the diamond as a connector to connect one crafted message to the challenge message on some states. Let 2^t be the width of the diamond and 2^ℓ be the length of the message; the complexity of this attack is approximately $2^{(n+t)/2} + 2^{n-\ell} + 2^{n-t}$.

2.4 Distinguished Points (DP)

The memory complexity of many algorithms that are based on functional graphs (*e.g.*, parallel collision search [vOW99]) 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 $\mathcal{G}$ computed in Alg. 5, but without storing all the 2^t nodes in memory. The idea is to define a set of 2^t distinguished points (nodes) using a simple predicate (*e.g.*, the $n-t$ LSBs of a node are zero). The nodes of $\mathcal{G}$ contain approximately $2^t \cdot 2^{t-n} = 2^{2t-n}$ distinguished points, and only they are stored in memory. A collision of an arbitrary chain with $\mathcal{G}$ is expected to occur at depth of about 2^{n-t} and will be detected at the next distinguished point which is located (approximately) after traversing additional 2^{n-t} nodes. Consequently, we can detect the collision with a small overhead in time complexity, but a significant saving factor of 2^{n-t} in memory.

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

2.5 Interchange Structure (IS)

In this subsection, we present how to build a structure that enables us to simultaneously control two (or more) hash computation lanes sharing the same input message and succeed in further relaxing the pairwise relation between the

internal states of computation lanes. We name the structure the *interchange structure*.

The main idea is to consider several chains of internal states reached by processing a common message $\boldsymbol{M}$ from different starting points (note that the message $\boldsymbol{M}$ is not fixed in advance, but will be determined when building the structure). More precisely, the message $\boldsymbol{M}$ is denoted as the *primary* message and divided into several chunks: $\boldsymbol{M} = M_0 \,\|\, M_1 \,\|\, \ldots$ (as discussed later, a chunk consists of approximately $n/2$ message blocks). We denote chains of internal states for $\mathcal{H}_1$ as $\vec{a}_j$ and the individual states of the chain as $\vec{a}_j^i$, with $h_1^*(\vec{a}_j^i, M_i) = \vec{a}_j^{i+1}$. Similarly, we denote chains for $\mathcal{H}_2$ as $\vec{b}_k$, with $h_2^*(\vec{b}_k^i, M_i) = \vec{b}_k^{i+1}$. When considering both hash functions, message block M_i leads from the pair of states $(\vec{a}_j^i, \vec{b}_k^i)$ to $(\vec{a}_j^{i+1}, \vec{b}_k^{i+1})$, which is denoted as

$$(\vec{a}_j^i, \vec{b}_k^i) \overset{M_i}{\rightsquigarrow} (\vec{a}_j^{i+1}, \vec{b}_k^{i+1}).$$

Switch Structure. To construct a desired interchange structure, we first create the basic building blocks to jump between chains in a controlled way; we named them *switches*. A switch allows to jump from a specific pair of chains $(\vec{a}_{j_0}, \vec{b}_{k_0})$ to a different pair of chains $(\vec{a}_{j_0}, \vec{b}_{k_1})$ using a secondary message chunk M_i', in addition to the normal transitions using chunk M_i of the primary message $\boldsymbol{M}$:

$$(\vec{a}_j^i, \vec{b}_k^i) \overset{M_i}{\rightsquigarrow} (\vec{a}_j^{i+1}, \vec{b}_k^{i+1}): \quad \text{normal transition for each chain}$$

$$(\vec{a}_{j_0}^i, \vec{b}_{k_0}^i) \overset{M_i'}{\rightsquigarrow} (\vec{a}_{j_0}^{i+1}, \vec{b}_{k_1}^{i+1}): \quad \text{jump from chains } (\vec{a}_{j_0}, \vec{b}_{k_0}) \text{ to } (\vec{a}_{j_0}, \vec{b}_{k_1})$$

To simplify the notation, we often omit the chunk index to show only the chains that are affected by the switch.

The main message chunk M_i and the secondary message chunk M_i' are determined when building the switch, and the main message defines the next state of all the chains. We note that the secondary message chunk M_i' should only be used when the state is $(\vec{a}_{j_0}^i, \vec{b}_{k_0}^i)$. A simple example is depicted in Fig. 7.

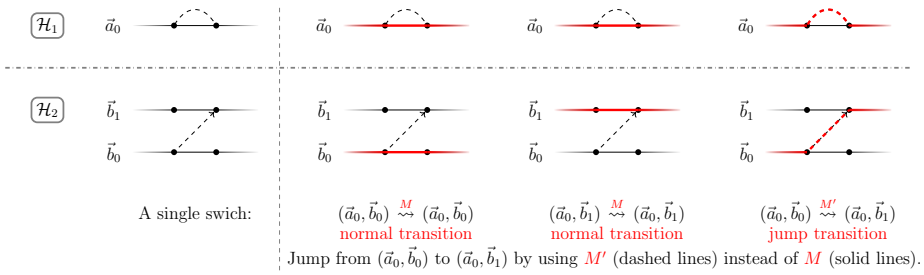


Fig. 7: A single switch

Alternatively, a switch can be designed to jump from $(\vec{a}_{j_0}, \vec{b}_{k_0})$ to $(\vec{a}_{j_1}, \vec{b}_{k_0})$. It can be built with a complexity of $\tilde{O}(2^{n/2})$.

We now explain how to build the switch structure at the core of some of our attacks. This construction is strongly based on the multi-collision technique of Joux presented in Sect.2.1.

Given states $\vec{a}^i_{j_0}$, $\vec{b}^i_{k_0}$ and $\vec{b}^i_{k_1}$, we want to build message chunks M_i and M'_i in order to have the following transitions:

$$\begin{aligned}
(\vec{a}^i_{j_0}, \vec{b}^i_{k_0}) &\overset{M_i}{\rightsquigarrow} (\vec{a}^{i+1}_{j_0}, \vec{b}^{i+1}_{k_0}): \quad \text{normal transition} \\
(\vec{a}^i_{j_0}, \vec{b}^i_{k_1}) &\overset{M_i}{\rightsquigarrow} (\vec{a}^{i+1}_{j_0}, \vec{b}^{i+1}_{k_1}): \quad \text{normal transition} \\
(\vec{a}^i_{j_0}, \vec{b}^i_{k_0}) &\overset{M'_i}{\rightsquigarrow} (\vec{a}^{i+1}_{j_0}, \vec{b}^{i+1}_{k_1}): \quad \text{jump transition}
\end{aligned}$$

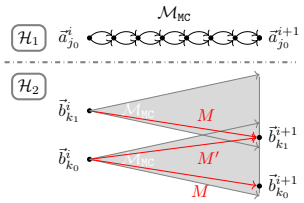
The main message chunk M_i is used to define the next state of all the remaining chains, while the secondary message chunk M'_i will be used to jump from chains $(\vec{a}_{j_0}, \vec{b}_{k_0})$ to $(\vec{a}_{j_0}, \vec{b}_{k_1})$. We note that M'_i will only be used when the state is $(\vec{a}^i_{j_0}, \vec{b}^i_{k_0})$. In particular, M_i and M'_i must satisfy the following:

$$\begin{aligned}
\vec{a}^{i+1}_{j_0} &= h_1^*(\vec{a}^i_{j_0}, M_i) = h_1^*(\vec{a}^i_{j_0}, M'_i) \\
\vec{b}^{i+1}_{k_1} &= h_2^*(\vec{b}^i_{k_1}, M_i) = h_2^*(\vec{b}^i_{k_0}, M'_i) \\
\vec{b}^{i+1}_{k_0} &= h_2^*(\vec{b}^i_{k_0}, M_i) \neq \vec{b}^{i+1}_{k_1}
\end{aligned}$$

The full building procedure is shown in Alg. 2 whose pseudo-code is given in Appendix A; it requires approximately $n/2 \cdot 2^{n/2}$ evaluations of the compression functions.

Algorithm 2: Building a single switch

1. Build a multi-collision for h_1^*, starting from state $\vec{a}^i_{j_0}$, *i.e.*, a set $\mathcal{M}_{\texttt{MC}}$ of $2^{n/2}$ messages that all reach the same state $\vec{a}^{i+1}_{j_0}$ ($\forall M \in \mathcal{M}_{\texttt{MC}}$, $h_1^*(\vec{a}^i_{j_0}, M) = \vec{a}^{i+1}_{j_0}$). As shown in Sect. 2.1, this can be done efficiently by sequentially building $n/2$ collisions. Thus, each M is comprised of $n/2$ message blocks.
2. Evaluate $h_2^*(\vec{b}^i_{k_0}, M)$ and $h_2^*(\vec{b}^i_{k_1}, M)$ for all the messages M in the set $\mathcal{M}_{\texttt{MC}}$. With high probability there is match between the sets of values[a]. Denote the colliding messages as M_i and M'_i, so that we have $h_2^*(\vec{b}^i_{k_0}, M'_i) = h_2^*(\vec{b}^i_{k_1}, M_i)$.
3. Compute the missing chains using the message M_i: $\vec{a}^{i+1}_j = h_1^*(\vec{a}^i_j, M_i)$, $\vec{b}^{i+1}_k = h_2^*(\vec{b}^i_k, M_i)$. With high probability all the chains reach distinct values; if this is not the case, restart the construction with a new multi-collision.[b]



First, M and M' are selected from $\mathcal{M}_{\text{MC}}$ to generate a collision (defining the new $\vec{b}_{k_1}$), then $\vec{b}_{k_0}$ is evaluated using M.

[a] If this is not the case, we build a new multi-collision.
[b] We omit the illustration of step 3 for simplicity in the figure.

Interchange Structure. By combining several simple switches, we can build an interchange structure with starting points IV_1 and IV_2 and ending points $\{A_j \mid j = 0 \dots 2^t - 1\}$ and $\{B_k \mid k = 0 \dots 2^t - 1\}$, so that we can select a message ending in any state (A_j, B_k). An interchange structure with 2^t chains for each function requires about 2^{2t} switches. Since we can build a switch for a cost of $\tilde{O}(2^{n/2})$, the total structure is built with $\tilde{O}(2^{2t+n/2})$ operations.

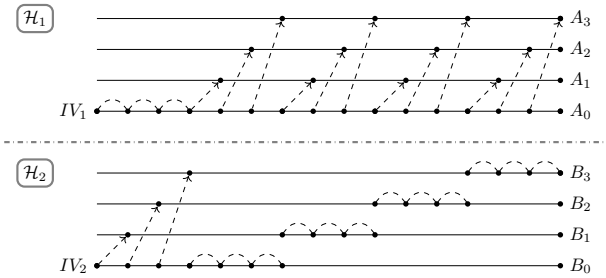
Let us now describe the combination of switch structures into an interchange structure. The goal of this structure is to select the final value of the $\mathcal{H}_1$ computation and the $\mathcal{H}_2$ computation independently. More precisely, the structure defines two sets of final values A_j and B_k, and a set of messages $\boldsymbol{M}_{jk}$ such that

$$(IV_1, IV_2) \overset{\boldsymbol{M}_{jk}}{\rightsquigarrow} (A_j, B_k).$$

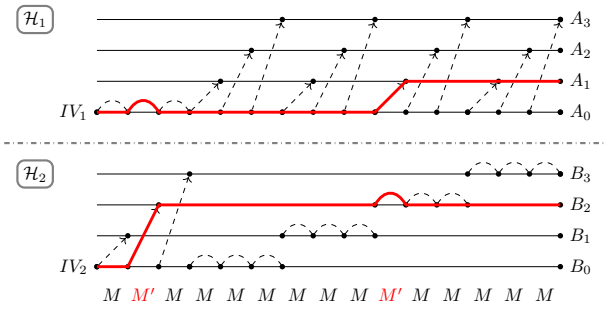
Algorithm 3: Building and using a 2^t-interchange structure

1. Initialize the first chains with $\vec{a}^0_0 = IV_1$, $\vec{b}^0_0 = IV_2$, and set the other starting points randomly.
2. Build switches to jump for an already reachable pair $(\vec{a}_{j_0}, \vec{b}_{k_0})$ to a different pair $(\vec{a}_{j_0}, \vec{b}_{k_1})$ (or to $(\vec{a}_{j_1}, \vec{b}_{k_0})$, respectively). By using $2^{2t} - 1$ switches, we can make all pairs reachable. There are many ways to combine the switches; a simple one can be described as follows:
   (a) first, build switches from $(\vec{a}_0, \vec{b}_0)$ to each of the $(\vec{a}_0, \vec{b}_k)$'s;
   (b) then, for each k, build a series of switches from $(\vec{a}_0, \vec{b}_k)$ to all the $(\vec{a}_j, \vec{b}_k)$'s.

   To reach the chains $(\vec{a}_j, \vec{b}_k)$, one would activate the k-th switch in the first part to jump from $(\vec{a}_0, \vec{b}_0)$ to $(\vec{a}_0, \vec{b}_k)$, and then the j-th switch in the k-th series of the second part to jump from $(\vec{a}_0, \vec{b}_k)$ to $(\vec{a}_j, \vec{b}_k)$.



One possible way to build an interchange structure



Select a given message in the interchange structure

Algorithm 3 describes the combination of switches to build an interchange structure. Its pseudo-code is given in Appendix A, where the INTERCHANGE functions builds the structure, and the SELECTMESSAGE function extracts the message reaching $(\vec{a}_j, \vec{b}_k)$.

The structure can be somewhat optimized using the fact that the extra chains have no prespecified initial values. We show how to take advantage of this in Appendix B, using multi-collision structures in addition to the switch structures. However, this does not significantly change the complexity: we need $(2^t-1)(2^t-1)$ switches instead of $2^{2t}-1$. In total, we need approximately $n/2 \cdot 2^{2t+n/2}$ evaluations of the compression functions to build a 2^t-interchange structure.

We believe that a 2^t-interchange structure based on switches will need at least $\Theta(2^{2t})$ switches, because every switch can only increase the number of reachable pairs $(\vec{a}_j, \vec{b}_k)$ by one. As shown in Appendix B some switches can be saved in the beginning, but it seems that new ideas are needed to reduce the total complexity below $\Theta(2^{2t+n/2})$.

2.6 Simultaneous Expandable Messages (SEM)

In this subsection, we build a *simultaneous expandable message* for two MD hash functions basing on the multi-collision described in Sect. 2.1 and the expandable message for a single MD hash function described in Sect. 2.2. 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 $M_{\|\kappa}$ of κ blocks that maps (IV_1, IV_2) to $(\hat{x}, \hat{y})$. A similar construction of an expandable message over two hash functions was proposed in the independent paper [JN15] by Jha and Nandi, which analyses the Zipper hash assuming weak compression functions. We describe our construction approach of this simultaneous expandable message in detail next.

We set $C \approx n/2 + \log(n)$ as a parameter that depends on the state size n. Our basic building block consists of two pairs of states (x_0, y_0) and (x_1, y_1) and two message fragments ms and ml that map the state pair (x_0, y_0) to (x_1, y_1). The message ms is the (shorter) message fragment of fixed size C, while ml is of size $i > C$. We will show how to construct this building block for any state pair (x_0, y_0) and length $i > C$ in Alg.4.

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 $j \in \{1, \ldots, t\}$. See Fig.8 for a visual illustration.

Given a length κ in the 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 $(\kappa-\kappa')/C$.

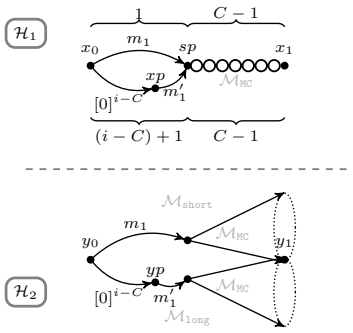
Construction of the Building Block. Given state pair (x_0, y_0) and length $i > C$, the algorithm for constructing the building block for the expandable message is based on multi-collisions, as described below; its pseudo-code is given in Appendix A.

Algorithm 4: Constructing a building block for an SEM

1. Pick an arbitrary prefix of size $i-C$ blocks, say $[0]^{i-C}$, and compute $xp = h_1^*(x_0, [0]^{i-C})$.
2. Find collision (m_1, m_1', sp) s.t. $h_1(x_0, m_1) = h_2(xp, m_1') = sp$, where m_1 and m_1' are single message blocks.
3. Build a 2^{C-1} standard Joux's multi-collision in h_1 starting from sp, and denote its endpoint by x_1. Altogether, we have a multi-collision in h_1 with 2^C messages that map x_0 to x_1. Out of these 2^C messages,

- 2^{C-1} are of length C (obtained by first selecting m_1, *i.e.*, $m_1 \times \mathcal{M}_{\texttt{MC}}$), and we denote this set of short messages by $\mathcal{M}_{\texttt{short}}$;
- 2^{C-1} are of length i (obtained by first selecting $[0]^{i-C} \parallel m_1'$, *i.e.*, $([0]^{i-C} \parallel m_1') \times \mathcal{M}_{\texttt{MC}}$), and we denote this set of long messages by $\mathcal{M}_{\texttt{long}}$.

4. Evaluate $yp = h_2^*(y_0, [0]^{i-C})$ and store the result. Next, evaluate h_2 from y_0 on the two sets $\mathcal{M}_{\texttt{short}}$ and $\mathcal{M}_{\texttt{long}}$ (using the stored yp to avoid recomputing $h_2^*(y_0, [0]^{i-C})$) and find a collision between them (such a collision is very likely to occur since $C - 1 > n/2$). The collision gives the required $ms \in \mathcal{M}_{\texttt{short}}$ and $ml \in \mathcal{M}_{\texttt{long}}$ of appropriate sizes such that $y_1 \triangleq h_2^*(y_0, ms) = h_2^*(y_0, ml)$ and $x_1 \triangleq h_1^*(x_0, ms) = h_1^*(x_0, ml)$.



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

Complexity Analysis of the Full Building Procedure. The full expandable message requires computing $C - 1 + t$ building blocks whose sum of length parameters (dominated by the final building block) is approximately $C \cdot 2^t \approx n \cdot 2^t$. Assuming that $t < n$, we construct $C - 1 + t \approx n$ building blocks, and the total time complexity of constructing the expandable message is approximately $n \cdot 2^t + n^2 \cdot 2^{n/2}$. Our attacks require 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 a time complexity of approximately

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

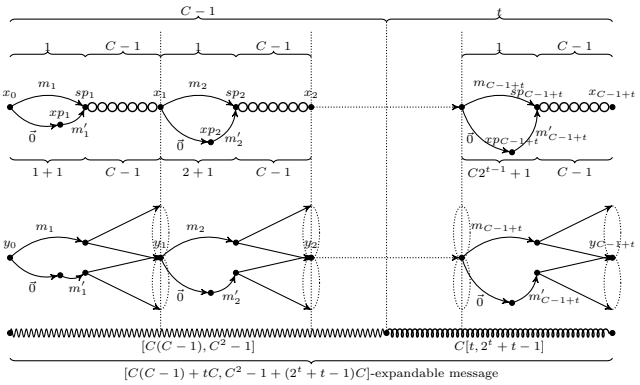


Fig. 8: Simultaneous expandable message

2.7 Functional Graph (FG) of Random Mappings

In many 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 $f(x) = h_{[m]}(x) = h(x, m)$. The mapping f yields a directed functional graph.

The functional graph of a random mapping f is defined via successive iteration on this mapping.

> Let f be an element in $\mathcal{F}_N$ that is the set of all mappings with the set N as both the domain and range. The functional graph of f is a directed graph whose nodes are the elements $0, \ldots, N-1$ and whose edges are the ordered pairs $\langle x, f(x)\rangle$, for all $x \in \{0, \ldots, N-1\}$. If starting from any x_0 and iterating f, that is $x_1 = f(x_0), x_2 = f(x_1), \ldots$, we will find that before N iterations, a value x_j equal to one of $x_0, x_1, \ldots, x_{j-1}$; suppose that the collided one is x_i. In this case, we say the path $x_0 \to x_1 \to \cdots \to x_{i-1} \to x_i$ connects to a *cycle* $x_i \to x_{i+1} \to \cdots \to x_{j-1} \to x_i$. If we consider all possible starting points x_0, paths exhibit confluence and form trees; trees grafted on cycles form components; a collection of components forms a functional graph. That is, a functional graph can be viewed as a set of connected components; a component is a cycle of trees; a tree is recursively defined by appending a node to a set of trees; a node is a basic atomic object and is labelled by an integer [FO89].

Structures of functional graph of random mappings have been studied for a long time, and some parameters have accurate asymptotic evaluations [FO89]. Below, we list some of the most relevant ones. These properties have been extensively studied and exploited in cryptography, *e.g.*, in the classical works of Hellman [Hel80] and van Oorschot and Wiener [vOW99], and much more recently in generic attacks on hash-based MACs [DL14,GPSW14,LPW13,PK14,PW14] (refer to [BGW18] for a systematization of knowledge regarding the applications of random functional graphs in generic attacks).

Theorem 1 ([FO89]). *The expected number of components, number of cyclic nodes (nodes belong to a cycle), number of terminal nodes (nodes without preim-*

age: $f^{-1}(x) = \emptyset$), number of image nodes (nodes with preimage), and number of k-th iterate image nodes (image nodes of the k-th iterate f^k of f) in a random mapping of size N have the following asymptotic forms as $N \to \infty$:

1. *# Components* $\frac{1}{2}\log N = 0.5 \cdot n$
2. *# Cyclic nodes* $\sqrt{\pi N/2} \approx 1.2 \cdot 2^{n/2}$
3. *# Terminal nodes* $e^{-1}N \approx 0.37 \cdot 2^n$
4. *# Image nodes* $(1-e^{-1})N \approx 0.62 \cdot 2^n$
5. *# k-th iterate image nodes* $(1 - \tau_k)N$, *where* τ_k *satisfies the recurrence relation* $\tau_0 = 0$, $\tau_{k+1} = e^{-1+\tau_k}$

Seen from an arbitrary node x_0, we call the length (measured by the number of edges) of the path starting from x_0 and before entering a cycle the *tail length* of x_0 and denote it by $\lambda(x_0)$; term the length of the cycle connected with x_0 the *cycle length* of x_0 and denote it by $\mu(x_0)$; name the length of the non-repeating trajectory of the node x_0 the *rho-length* of x_0 and denote it by $\rho(x_0) = \lambda(x_0) + \mu(x_0)$.

Theorem 2 ([FO89]). *Seen from a random point (any of the N nodes in the associated functional graph is taken equally likely) in a random mapping of $\mathcal{F}_N$, the expected tail length, cycle length, rho-length have the following asymptotic forms:*

1. *Tail length* (λ) $\sqrt{\pi N/8} \approx 0.62 \cdot 2^{n/2}$
2. *Cycle length* (μ) $\sqrt{\pi N/8} \approx 0.62 \cdot 2^{n/2}$
3. *Rho-length* (ρ) $\sqrt{\pi N/2} \approx 1.2 \cdot 2^{n/2}$

Theorem 3 ([FO89]). *The expected maximum cycle length (μ^{max}), maximum tail length (λ^{max}) and maximum rho length (ρ^{max}) in the functional graph of a random mapping of $\mathcal{F}_N$ satisfy the following:*

1. $\mathbf{E}\{\mu^{max} \mid \mathcal{F}_N\} = 0.78248 \cdot 2^{n/2}$
2. $\mathbf{E}\{\lambda^{max} \mid \mathcal{F}_N\} = 1.73746 \cdot 2^{n/2}$
3. $\mathbf{E}\{\rho^{max} \mid \mathcal{F}_N\} = 2.41490 \cdot 2^{n/2}$

Theorem 4 ([FO89]). *Assuming the smoothness condition, the expected value of the size of the largest tree and the size of the largest connected component in a random mapping of $\mathcal{F}_N$ are asymptotically*

1. *Largest tree:* $0.48 \cdot 2^n$
2. *Largest component:* $0.75782 \cdot 2^n$

The results from these theorems indicate that in a random mapping, most of the points tend to be grouped together in a single giant component. This component is therefore expected to have very tall trees and a large cycle [FO89].

A useful algorithm for expanding the functional graph of f is given below (see Alg. 5 whose pseudo-code is given in Appendix A). This algorithm is not new and has been previously used (for example, in [GPSW14,PW14]). It takes an input parameter $t \geq n/2$ that determines the number of expanded nodes (and the running time of the algorithm).

Algorithm 5: Expanding the functional graph of f

1. Initialize $\mathcal{G} = \emptyset$ as a data structure of evaluated nodes.
2. Until $\mathcal{G}$ contains 2^t 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 hits a point in $\mathcal{G}$. Add the points of the chain to $\mathcal{G}$.

Deep Iterates in the Functional Graphs (FGDI). Next, we describe our observations on functional graph of random mappings. The efficiencies of our following attacks are mostly based on these observations on special nodes in functional graphs.

In our attacks, we are particularly interested in nodes of f that 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 $x' = f^i(x)$, *i.e.*, x' is an i-th iterate image node (or say *i-th iterate* for short). If i is relatively large, we say that x' is a deep iterate. Deep iterates are usually obtained 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 $\vec{x}$. The following two observations regarding deep iterates make them helpful in the proposed attacks:

Observation 1. It is easy to obtain a large set of deep iterates. Specifically, by running Alg. 5 with input parameter t ($t \geq n/2$), one can obtain a set of 2^t nodes, among which a constant fraction ($\Theta(2^t)$) are 2^{n-t}-th iterates. The theoretical reasoning is as follows. After we have executed the algorithm and developed 2^t nodes, then another chain from an arbitrary starting point is expected to collide with the evaluated graph at depth of roughly 2^{n-t}. This is a direct consequence of the birthday paradox. Moreover, for two chains from two different starting points x and y, the probability that $\Pr[f^{2^{n-t}}(x) = f^{2^{n-t}}(y)] = \Theta(2^{-t})$ [DL14, Lemma 1] (note that $n-t < n/2$). That is, for $t \geq n/2$, when the number of new chains (of length 2^{n-t} and from arbitrary starting points) is less than 2^t, they are expected to collide with the evaluated graph at distinct points. In particular, this observation implies that most chains developed by the algorithm will be extended to depth $\Omega(2^{n-t})$ (without colliding with $\mathcal{G}$ of cycling); therefore, a constant fraction of the developed nodes are iterates of depth 2^{n-t}. In total, the algorithm develops $\Theta(2^t)$ iterates of f of depth 2^{n-t} in 2^t time. This conclusion was also verified experimentally.

Observation 2. A deep iterate has a relatively high probability to be encountered during the evaluation of a chain from an arbitrary starting node. Let f_1 and f_2 be two independent n to n-bit mappings. Suppose $\bar{x}$ (resp. $\bar{y}$) is an iterate of depth 2^g in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$); then, it is an endpoint of a chain of states of length 2^g. Let d be in the interval $[1, 2^g]$ and x_0 (resp. y_0) be a random point. Then, according to Lemma 1, $\Pr[x_d = \bar{x} \approx d \cdot 2^{-n}]$ (resp. $\Pr[y_d = \bar{y} \approx d \cdot 2^{-n}]$), which is the probability that $\bar{x}$ (resp. $\bar{y}$) will be encountered at distance d from x_0 (resp. y_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 $[1, 2^g]$, we conclude that the probability that $\bar{x}$ and $\bar{y}$ will be encountered at the same distance is approximately $(2^g)^3 \cdot 2^{-2n} = 2^{3g-2n}$.

The probability calculation in Observation 2 yields the conclusion that we need to compute approximately 2^{2n-3g} chains from different starting points to find a pair of starting points (x_0, y_0) reaching a pair of 2^g-th iterates $(\bar{x}, \bar{y})$ at the same distance. This conclusion was verified experimentally. Note that since various trials performed by selecting different starting points for the chains are dependent, the proof of this conclusion is incomplete. However, this dependency is negligible in our attacks, and thus we can ignore it. More details can be found in Appendix C.

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 $x_d = f(x_{d-1})$ for integer $d \geq 1$. 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 $D \leq 2^{n/2}$. For $x_d = x'_D$ to occur, x_{d-i} should collide with x'_{D-i} for[14] some $0 \leq i \leq d$. For a fixed i, the probability for this collision is roughly[15] 2^{-n}, and summing over all $0 \leq i \leq d$ (all events are disjointed), we get that the probability is approximately $d \cdot 2^{-n}$. □

Multi-Cycles in Functional Graphs (FGMC) Next, we study a property of some more special nodes — *cyclic nodes* in random functional graphs. There are efficient cycle search algorithms (with $O(2^{n/2})$ time complexity) to detect the cycle length and collect cyclic nodes in the largest component of a random functional graph [Jou09, Chapter 7], and cycles has been exploited in generic attacks on hash-based MACs [GPSW14,LPW13]. Here, we exploit them in a new way. Each cyclic node in a functional graph defined by f loops along the cycle when computed by f iteratively and goes back to itself after a (multi-) cycle-length number of function calls. This property can be utilized to provide extra degrees of freedom when estimating the distance of other nodes to a cyclic node in the functional graph, *i.e.*, it can be expanded to a set of discrete values by using multi-cycles. For example, let x and x' be two nodes in a component of the functional graph defined by f, x be a cyclic node, and the cycle length of the component be denoted as L. Clearly, there exists a path from x' to x as they are in the same component, and the path length is denoted by d. Then, we have the following:

$$f^d(x') = x; \quad f^L(x) = x \implies f^{(d+i\cdot L)}(x') = x \quad \text{for any positive integer } i.$$

Suppose it is limited to use at most t cycles (limitation imposed by the length of the message). Then, the distance from x' to x is expanded to a set of $t+1$ values $\{d + i \cdot L \mid i = 0, 1, 2, ..., t\}$.

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

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

Now, let us consider a special case of reaching two deep iterates from two random starting nodes: *select two cyclic nodes within the largest components in the functional graphs as the deep iterates.* More specifically, let $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ be two functional graphs defined by f_1 and f_2. Let $\bar{x}$ and x_0 be two nodes in a common largest component of $\mathcal{FG}_{f_1}$, where $\bar{x}$ is a cyclic node. Let L_1 denote the cycle length of the component and d_1 denote the path length from x_0 to $\bar{x}$. Similarly, we define notations $\bar{y}$, y_0, L_2 and d_2 in $\mathcal{FG}_{f_2}$. We are interested in the probability of linking x_0 to $\bar{x}$ and y_0 to $\bar{y}$ at a common distance. Thanks to the usage of multiple cycles, the distance values from x_0 to $\bar{x}$ and from y_0 to $\bar{y}$ can be selected from two sets $\{d_1 + i \cdot L_1 \mid i = 0, 1, 2, \ldots, t\}$ and $\{d_2 + j \cdot L_2 \mid j = 0, 1, 2, \ldots t\}$, respectively. Hence, as long as there exists a pair of integers (i, j) such that $0 \leq i, j \leq t$ and $d_1 + i \cdot L_1 = d_2 + j \cdot L_2$, we obtain a common distance $d = d_1 + i \cdot L_1 = d_2 + j \cdot L_2$ such that

$$f_1^d(x_0) = \bar{x}, \qquad\qquad f_2^d(y_0) = \bar{y}.$$

Next, we evaluate the probability amplification of reaching $(\bar{x}, \bar{y})$ from a random pair (x_0, y_0) at the same distance. Without loss of generality, we assume $L_1 \leq L_2$. Let $\Delta L \triangleq L_2 \mod L_1$. Then, it follows that

$$\begin{aligned} d_1 + i \cdot L_1 &= d_2 + j \cdot L_2 && \Longrightarrow \\ d_1 - d_2 &= j \cdot L_2 - i \cdot L_1 && \Longrightarrow \\ (d_1 - d_2) \mod L_1 &= j \cdot \Delta L \mod L_1 \end{aligned}$$

Letting j range over all integer values in internal $[0, t]$, we will collect a set of $t + 1$ values $\mathcal{D} = \{j \cdot \Delta L \mod L_1 \mid j = 0, 1, \ldots, t\}$.[16] Since $d_1 = \mathcal{O}(2^{n/2})$, $d_2 = \mathcal{O}(2^{n/2})$ and $L_1 = \Theta(2^{n/2})$, it follows that $|d_1 - d_2| = \mathcal{O}(L_1)$, and we assume $|d_1 - d_2| < L_1$ by ignoring the constant factor. Therefore, for a randomly sampled pair (x_0, y_0) that encounters $(\bar{x}, \bar{y})$, we are able to derive a pair of (i, j) such that $d_1 + i \cdot L_1 = d_2 + j \cdot L_2$, as long as their distance bias $d_1 - d_2$ is in the set $\mathcal{D}$. In other words, we are able to *correct such a distance bias by using multi-cycles.* Hereafter, the set $\mathcal{D}$ is referred to as the set of *correctable distance bias.* Thus, the probability of reaching $(\bar{x}, \bar{y})$ from a random pair (x_0, y_0) at a common distance is amplified by roughly t times, where t is the number of cycles to the maximum.

3 Preimage Attack on XOR Combiners Based on the Interchange Structure

In this section, we introduce our first attack — the preimage attack on the XOR combiner. In this attack, we are given an n-bit target value V, and our goal is to find a message $\boldsymbol{M}$ such that $\mathcal{H}_1(\boldsymbol{M}) \oplus \mathcal{H}_2(\boldsymbol{M}) = V$. Notice that, if the goal is to find two messages $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ such that $\mathcal{H}_1(\boldsymbol{M}_1) \oplus \mathcal{H}_2(\boldsymbol{M}_2) = V$,

[16] This is a very low probability that the set contains repeated values, particularly when t is significantly small compared with L_1. Here, we omit the discussion.

we can immediately launch a meet-in-the-middle procedure to find a solution of the equation $\mathcal{H}_1(\boldsymbol{M}_1) = \mathcal{H}_2(\boldsymbol{M}_2) \oplus V$ with $2^{n/2}$ computations. That is because in the last equation, the left-hand side and right-hand side are independent. By separately computing $2^{n/2}$ values on each side, we obtain 2^n pairs and will find a match with high probability due to the birthday paradox. Thus, the above is essentially to find a collision, which is an easy challenge. However, in the real challenge, the collision must be generated from the same message. Thus, the computations on the two side of the equation are pairwise related, *i.e.*, the computation on one side of the equation can only pair with a single computation on the other side. Consequently, unlike in the easy challenge, $2^{n/2}$ computations on each side can only generate $2^{n/2}$ pairs instead of 2^n. Therefore, to launch a similar meet-in-the-middle procedure as we did in the easy challenge for the real challenge, a crucial part of our attack is to construct a structure breaking the pairwise dependency between the two computations. That structure playing the important role is the *interchange structure* introduced in Sect. 2.5. Next, we provide an overview of our attack based on the interchange structure and then give detailed attack procedures.

3.1 Overview of the Attack

Next, we give an overview of the first preimage attack on the XOR combiner. Let V denote the target value. The two hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$ share the same input message, and hence the internal states of their iterative compression function computations are pairwise related. We first manage to simultaneously control the computation chains of $\mathcal{H}_1$ and $\mathcal{H}_2$ by constructing an interchange structure including a message structure $\mathcal{M}$ and two sets of internal states $\mathcal{A}$ (for $\mathcal{H}_1$) and $\mathcal{B}$ (for $\mathcal{H}_2$) such that for any state A picked from $\mathcal{A}$ and B picked from $\mathcal{B}$, we can easily derive a message $M_{A,B}$ from $\mathcal{M}$ such that $\mathcal{H}_1(M_{A,B})$ produces A and $\mathcal{H}_2(M_{A,B})$ produces B. Hence, we can select states from $\mathcal{A}$ and $\mathcal{B}$ independently in the next phase of the attack. In the next phase, we use a birthday match to find a message block m, a state A in $\mathcal{A}$ and a state B in $\mathcal{B}$ such that $h_1(A, m) \oplus h_2(B, m)$ equals the target hash digest V, where h_1 and h_2 are the compression functions of $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. Finally, given states A and B, we derive the message $M_{A,B}$ from $\mathcal{M}$, and output $M_{A,B} \,\|\, m$ as a preimage of V.[17] The birthday match in the second phase of the attack is essentially a meet-in-the-middle procedure enabled by the interchange structure built in the first phase of the attack. Thus, the entire attack is more efficient than a brute-force attack.

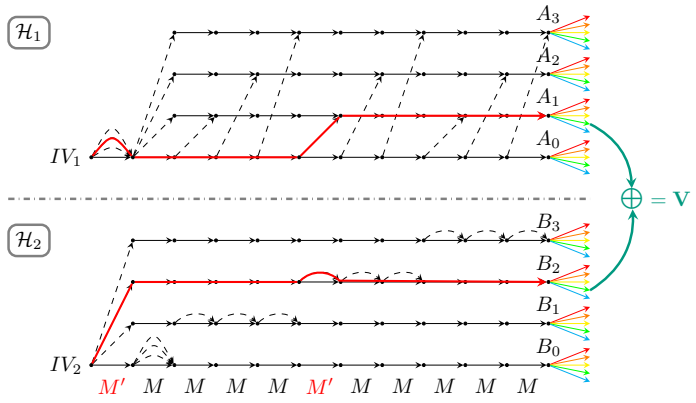
Attack 1: Preimage attack on XOR combiner based on the interchange structure

[17] Note that for simplicity of description, we omit the description of the finalization transformation on the internal state with the padding block and refer to Sect. 3.2 for the formal description.

– **Phase 1:** Build a 2^t-interchange structure using 2^{2t} switches that enables to jump between chains (see. Sect.2.5). This structure has starting points IV_1 and IV_2 and ending points $\{A_j \mid j = 0 \dots 2^t - 1\}$ and $\{B_k \mid k = 0 \dots 2^t - 1\}$, so that for any state pair (A_j, B_k), we can easily select a message ending in it.
– **Phase 2:** Select a random message block m, and compute two lists by evaluating the compression functions after the interchange structure: $\{A'_j = h_1(A_j, m) \mid j = 0 \dots 2^t - 1\}$ and $\{B'_k = V \oplus h_2(B_k, m) \mid k = 0 \dots 2^t - 1\}$. We expect a match between the lists with probability 2^{2t-n}. After about 2^{n-2t} random choices of m, we obtain a match (j^*, k^*):

$$h_1(A_{j^*}, m) = V \oplus h_2(B_{k^*}, m) \quad i.e. \quad h_1(A_{j^*}, m) \oplus h_2(B_{k^*}, m) = V.$$

Therefore, we can construct a preimage of V by concatenating the message leading to (A_{j^*}, B_{k^*}) in the interchange structure and m (we ignore the finalization function here).



The complexity of the preimage search is approximately 2^{n-t} evaluations of the compression function, using an interchange structure with 2^t endpoints.

Complexity Analysis. Building the interchange structures requires approximately $2^{2t+n/2}$ evaluations of the compression function, while the preimage search requires approximately 2^{n-t}. The optimal complexity[18] is reached when both steps take the same time, *i.e.*, $t = n/6$. This gives a complexity of $\tilde{O}(2^{5n/6})$. Since it uses messages of length at least $n/2 \cdot 2^{2t}$, the optimal complexity is obtained for messages of length at least $2^{n/3}$. For messages shorter than $2^{n/3}$, it provides a trade-off of $2^n \cdot L^{-1/2}$ between the maximal allowed message length L and the time complexity of attack (see Fig. 12 for a trade-off curve).

[18] From now on, we will use "optimal complexity" to mean the minimized complexity under the optimal choice of parameters for each attack.

3.2 Details of the Preimage Attack on XOR Combiners Using the Interchange Structure

Now, we describe the full preimage attack in detail. We first build an interchange structure with 2^t chains for each of $\mathcal{H}_1$ and $\mathcal{H}_2$. We denote the ending points as $\{A_j \mid j = 0 \dots 2^t - 1\}$ and $\{B_k \mid k = 0 \dots 2^t - 1\}$, and we know how to select a message $\boldsymbol{M}_{jk}$ to reach any state (A_j, B_k). When adding message fragment $m \parallel pad$, to one of the messages $\boldsymbol{M}_{jk}$ in the interchange structure, where m is a message block and pad is the final block padded with the length L of the preimage message, the output of the combiner can be written as follows:

$$\mathcal{H}_1(\boldsymbol{M}_{jk}\|m\|pad) \oplus \mathcal{H}_2(\boldsymbol{M}_{jk}\|m\|pad) = h_1(h_1(A_j, m), pad) \oplus h_2(h_2(B_k, m), pad),$$

Note that we fix the finalization functions of $\mathcal{H}_1$ and $\mathcal{H}_2$ as their compression functions, h_1 and h_2, respectively.

To reach a target value V, we select a random block m, and we evaluate $\{A'_j = h_1(h_1(A_j, m), pad) \mid j = 0 \dots 2^t - 1\}$ and $\{B'_k = V \oplus h_2(h_2(B_k, m), pad) \mid k = 0 \dots 2^t - 1\}$. If there is a match (j^*, k^*) between the two lists, we have the following:

$$\begin{aligned} A'_{j^*} = B'_{k^*} &\Leftrightarrow h_1(h_1(A_{j^*}, m), pad) = V \oplus h_2(h_2(B_{k^*}, m), pad) \\ &\Leftrightarrow \mathcal{H}_1(\boldsymbol{M}_{jk} \parallel m \parallel pad) \oplus \mathcal{H}_2(\boldsymbol{M}_{jk} \parallel m \parallel pad) = V. \end{aligned}$$

For a random choice of m, we expect that a match exists with probability 2^{2t-n}, and testing it requires approximately 2^t operations[19]. We will have to repeat this procedure 2^{n-2t} times on average; therefore, the total cost of the preimage search is approximately 2^{n-t} evaluations of h_1 and h_2.

As explained in the previous section, building a 2^t-interchange structure requires approximately $n/2 \cdot 2^{2t+n/2}$ operations. Using $t = n/6$, we balance the two steps of the attack and reach the optimal complexity of approximately $n/2 \cdot 2^{5n/6}$ operations for this preimage attack.

4 Improved Preimage Attack on XOR Combiners Based on Deep Iterates

The first attack works identically for the case in which the combined hash functions use the HAIFA mode, and the case in which they use the MD construction. However, when they are limited to use the MD construction, we can launch a more efficient attack than the first. In this case, pairwise dependency between internal states can be broken efficiently by using repeated message blocks. Explicitly, we use a different approach to get the two sets of states $\mathcal{A} = \{A_j \mid j = 0 \dots 2^t - 1\}$ and $\mathcal{B} = \{B_k \mid k = 0 \dots 2^t - 1\}$ such that for any pair of states $(A_j, B_k \mid A_j \in \mathcal{A}, B_k \in \mathcal{B})$, we can manage to find a message $M_{A,B}$ such that $(IV_1, IV_2) \xrightarrow{M_{A,B}} (A_j, B_k)$. For convenience, we name such an

[19] It takes $O(t \cdot 2^t)$ operations by sorting the lists, but only $2 \cdot 2^t$ using a hash table.

abstract procedure `GenPairableStates`, which is implemented and utilized by quite different approaches in different attacks – the first attack implements it using interchange structure, this second attack implements it using deep iterates in functional graphs, and as will be seen, the third attack implements it using cyclic nodes in functional graphs.

The first step is to fix an arbitrary message block m to the compression functions, giving rise to n to n-bit random mappings $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$. Such random mappings and their functional graphs have many interesting properties and have been extensively studied and used in cryptanalysis, as shown in Sect. 2.7. However, to attack hash combiners, we exploit them in new ways. In this attack, instead of precisely controlling every computational step in chains of equal length to obtain two sets of endpoints as in building an interchange structure, here, we loosely herd computational chains of various length to collect two sets of states. These collected states have large offsets in the chains. These chains are iteratively computed using the above defined random mappings f_1 and f_2. Thus, the collected states are essentially *deep iterates* in the functional graphs of f_1 and f_2, which are introduced in Sect. 2.7. As has been shown in Sect. 2.7, such special states are relatively easy (*i.e.*, with high probability) to be reached from randomly selected starting states. This is where the advantage of the attack mainly comes from.

In this attack, given a pair of such special states (A_j, B_k), finding a common message mapping a pair of starting states to them under the two hash computations is not as efficient as selecting a message from an interchange structure in Attack 1. However, collecting those final states by expanding functional graphs is much more efficient than computing endpoints by building an interchange structure. This attack amortizes computational costs to different steps. Thus, it provides better balance between different attack steps. Moreover, it also provides a better trade-off between the message length and time complexity.

However, unlike the interchange-structure-based attack, this approach uses chains of various lengths, which implies that lengths of message fragments in intermediate attack steps are not fixed in advance. However, the length of the preimage needs to be predefined in this attack. Thus, the length padding at the end of the hash computations will be a problem. We overcome this problem using our tool, the simultaneous expandable message for two MD hash functions, which is introduced in Sect. 2.6.

Next, we provide a high-level overview of this attack and then present the detailed attack steps.

4.1 Overview of the Attack

Suppose that we are given a target n-bit preimage value V and our goal is to find a message $\boldsymbol{M}$ such that $\mathcal{H}_1(\boldsymbol{M}) \oplus \mathcal{H}_2(\boldsymbol{M}) = V$. Although the formal problem does not restrict $\boldsymbol{M}$ in any way, several concrete hash functions restrict the length of $\boldsymbol{M}$. Therefore, we will first assume that the size of $\boldsymbol{M}$ is bounded by a parameter L.

The attack is composed of three main phases.

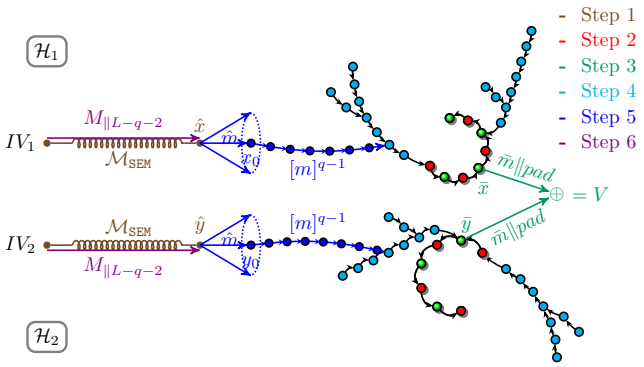
Attack 2: Preimage attack on XOR combiner based on deep iterates

- **Phase 1:** Build a simultaneously expandable message $\mathcal{M}_{\texttt{SEM}}$ for $\mathcal{H}_1$ and $\mathcal{H}_2$, starting from the initial state pair (IV_1, IV_2) and ending with 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 $M_{\|\kappa}$ of κ blocks that maps (IV_1, IV_2) to $(\hat{x}, \hat{y})$.
- **Phase 2:** Find a set $\mathcal{S}$ (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.
- **Phase 3:** Start from $(\hat{x}, \hat{y})$ and compute a message fragment $\hat{M}_{\|q}$ of length q (shorter than $L - 2$) such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}_{\|q}} (\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$. For this tuple, denote $(\bar{a}, \bar{b}) \triangleq h_{1,2}((\bar{x}, \bar{y}), \bar{m})$.

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

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

Altogether, we obtain the padded preimage for the XOR combiner: $\boldsymbol{M} = M_{\|L-q-2} \,\|\, \hat{M}_{\|q} \,\|\, \bar{m} \,\|\, pad$.



Complexity Analysis. Denote $L = 2^\ell$. For parameters $g_1 \geq max(n/2, n - \ell)$ and $s \geq 0$, the complexity of 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 be $2^{n/3+2g_1/3} + 2^{\ell+2g_1-n}$. 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[20] is

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

The optimal complexity is $2^{2n/3}$, obtained for messages of length $2^{n/2}$ (see Fig. 12 for a trade-off curve).

4.2 Details of the Preimage Attack on XOR Combiners using Deep Iterates

Details of Phase 1 can be found in Sect.2.6. In the following, we describe details of the other two phases.

Details of Phase 2: Finding a Set of Target State Pairs. In the second phase, we fix an arbitrary message block m, giving rise to the functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ defined by the random mappings $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$. Given parameters $g_1 \geq n/2$ and $s \geq 0$, our goal is to compute a set $\mathcal{S}$ (of size 2^s) of tuples of the form $((x, y), w)$, where w is a single block such that for each tuple, the following hold:

1. The state x is a 2^{n-g_1}-th iterate in $\mathcal{FG}_{f_1}$, and y is a 2^{n-g_1}-th iterate in $\mathcal{FG}_{f_2}$.
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.

This algorithm resembles the algorithm used in the final phase in previous interchange-structure-based preimage attack (Attack 1 in Sect. 3), as both look for state pairs (x, y) that give $h_1(x, w \parallel pad) \oplus h_2(y, w \parallel pad) = V$ (for some message block w). The difference is that in previous interchange-structure-based attack, (x, y) is an arbitrary endpoint pair of the interchange structure, while in this case, we look for x and y that are deep iterates.

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

Phase 2 of Attack 2: Finding a set of target state pairs

1. Fix an arbitrary single-block value m, and construct $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$.
2. Expand $\mathcal{FG}_{f_1}$ using Alg. 5 with parameter g_1. Store all encountered 2^{n-g_1}-th iterates in a table $\mathcal{T}_1$.
3. Expand $\mathcal{FG}_{f_2}$ using Alg. 5 with parameter g_1. Store all encountered 2^{n-g_1}-th iterates in a table $\mathcal{T}_2$.
4. Allocate a set $\mathcal{S} = \emptyset$. For single-block values $w = 0, 1, \ldots$, perform the following steps until $\mathcal{S}$ contains 2^s elements:
   (a) For each node $x \in \mathcal{T}_1$ evaluate $h_1(x, w \,\|\, pad)$, and store the results in a table $\mathcal{T}_1'$, sorted according $h_1(x, w \,\|\, pad)$.
   (b) For each node $y \in \mathcal{T}_2$ 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 $\mathcal{T}_1'$. For each match, add the tuple $((x, y), w)$ to $\mathcal{S}$.

The time complexity of steps 2 and 3 is approximately 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 $\mathcal{S}$.

According to Observation 1 in Sect. 2.7, the expected size of $\mathcal{T}_1$ and $\mathcal{T}_2$ (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

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

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 $\hat{M}_{\|q}$ of length q (is valid as long as shorter than $L-2$) such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}_{\|q}} (\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$. We use in a strong way the fact that states $\bar{x}$ (and $\bar{y}$) in $\mathcal{S}$ are deep iterate (of depth 2^{n-g_1}) in $\mathcal{FG}_{f_1}$ (and $\mathcal{FG}_{f_2}$).

The goal of this phase is to find a pair of starting points of chains, reaching some $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$ at same distance. This phase is carried out by picking an arbitrary starting message block $\hat{m}$, which gives points $x_0 = h_1(\hat{x}, \hat{m})$ and $y_0 = h_2(\hat{y}, \hat{m})$. 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 $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$. In case in which for all pairs of $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$, $\bar{x}$ 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 $\hat{m}$ and start again. Once we find such a value for $\hat{m}$ and pair of iterates $(\bar{x}, \bar{y})$, this gives the required $\hat{M}_{\|q} \triangleq \hat{m} \,\|\, [m]^{q-1}$.

According to Observation 2 in Sect. 2.7, for a pair of 2^{n-g_1}-th iterates $\bar{x}$ and $\bar{y}$, the probability that they will be encountered at the same distance from arbitrary starting points x_0 and y_0 of chains is $(2^{n-g_1})^3 \cdot 2^{-2n} = 2^{n-3g_1}$. Since $\mathcal{S}$ contains 2^s elements, the probability calculation yields the conclusion that we need to compute about 2^{3g_1-n-s} chains from different starting points to find such a value for $\hat{m}$ generating starting points (x_0, y_0) reaching a pair of deep iterates $(\bar{x}, \bar{y})$ in $\mathcal{S}$ at the same distance.

The next question which we address is to what maximal length L' should we evaluate chains $\vec{x}$ and $\vec{y}$. 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 $L - 3$, this impose the restriction $L' = 2^{n-g_1} < L - 3$, or $2^{g_1} < 2^n/L$.

The naive algorithm described above performs about 2^{3g_1-n-s} trials, where each trial evaluates chains of length $L' = 2^{n-g_1}$ from arbitrary points, giving a total time complexity of approximately $2^{3g_1-n-s+n-g_1} = 2^{2g_1-s}$. Since $g_1 \geq n/2$, the time complexity of this phase is at least 2^{n-s}, and after making a balance with that of the Phase 2, the time complexity can be $2^{3n/4}$ by setting $s = n/4$.

However, it is possible to optimize this naive algorithm by further expanding 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 look-ahead procedure to calculate the distance of the chain's starting point from $\bar{x}$ (or $\bar{y}$) in each tuple $((x, y), w) \in \mathcal{S}$. This look-ahead procedure resembles the one used in attacks on hash-based MACs [GPSW14,PW14] (although the setting and actual algorithm in our case are obviously different).

We define an $\mathcal{S}$-*node* (for f_1) as a node x such that there exists a node y and a message block w such that $((x, y), w) \in \mathcal{S}$. An $\mathcal{S}$-node for f_2 is defined in a similar way. To avoid heavy update operations for the distances from all the $\mathcal{S}$-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 $\mathcal{S}$-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.

Phase 3 of Attack 2: Hitting a target state pair

1. Develop (about) 2^{g_2} nodes in $\mathcal{FG}_{f_1}$ (and $\mathcal{FG}_{f_2}$) (as specified in Alg. 5) with the following modifications.
   - Store only distinguished points for which the $n - g_2$ LSBs are zero.
   - Once an $\mathcal{S}$-node is encountered, update its distance in the previously encountered distinguished point (which is defined with high probability[a]).
   - Stop evaluating each chain once it hits a stored distinguished point.
   - The evaluated distinguished points for f_1 (resp. f_2) are stored in the data structure $\mathcal{G}_1$ (resp. $\mathcal{G}_2$).

2. For single-block values $\hat{m} = 0, 1, \ldots$, compute $x_0 = h_1(\hat{x}, \hat{m})$ and $y_0 = h_2(\hat{y}, \hat{m})$ and repeat the following step:
   (a) Compute chains $\vec{x}$ and $\vec{y}$ 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 $\mathcal{G}_1$ and $\mathcal{G}_2$ (respectively).
      – Then, allocate a table $\mathcal{T}_1$ for f_1 (and $\mathcal{T}_2$ for f_2) and continue traversing (only) the distinguished points of the chain (using the links in $\mathcal{G}_1$ and $\mathcal{G}_2$) up to depth $L-2$, while updating $\mathcal{T}_1$ (resp. $\mathcal{T}_2$): for each visited distinguished point, add all its stored $\mathcal{S}$-nodes to $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) with its distance from x_0 (resp. y_0).
      – Once the maximal depth $L-2$ is reached, sort $\mathcal{T}_1$ and $\mathcal{T}_2$. 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 $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$. If such $\bar{x} \in \mathcal{T}_1$ and $\bar{y} \in \mathcal{T}_2$ exist, return the message $\hat{M}_{\|q} = \hat{m} \parallel [m]^{q-1}$ and $\bar{m}$ (retrieved from $\mathcal{S}$) as output. Otherwise (no such $\bar{x}$ and $\bar{y}$ were found), return to Step 2.

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

The time complexity of Step 1 is approximately 2^{g_2} (note that we always perform a constant amount of work per developed node).

For time complexity of Step 2, the analysis is as follows. As concluded above, the expected number of values for $\hat{m}$ we need to test until we find a pair of starting point (x_0, y_0) of chains encounter at the same distance to a pair of 2^{n-g_1}-th iterates $(\bar{x}, \bar{y})$ in $\mathcal{S}$ is approximately 2^{3g_1-n-s}.

The analysis of the complexity of Step 2.(a) is as follows. First, we estimate the expected number of nodes that we visit during the computation of a chain. Initially, we compute approximately 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 approximately $2^{n-g_2} + L \cdot 2^{g_2-n}$ nodes while computing a chain. Finally, we need to account for all the $\mathcal{S}$-nodes encountered while traversing the chains of depth L. Basically, there are 2^s $\mathcal{S}$-nodes which are iterates of depth 2^{n-g_1}, (essentially) randomly chosen in Phase 2 out of approximately 2^{g_1} such deep iterates. As a result, the probability of such a deep iterate to be a $\mathcal{S}$-node is approximately 2^{s-g_1} (while other nodes have probability 0). Therefore, while traversing chains of depth L, we expect to encounter at most $L \cdot 2^{s-g_1}$ $\mathcal{S}$-nodes (which is a bound on the sizes of $\mathcal{T}_1$ and $\mathcal{T}_2$). 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 a 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 $\mathcal{S}$-nodes.

4.3 Optimizing the Deep-Iterates-Based Preimage Attack on XOR Combiners using the Interchange Structure

The above deep-iterates-based preimage attack on XOR combiners can be slightly improved using an interchange structure. Recall that the interchange structure helps to break the dependency between two hash computations on a common message. When building a 2^r-interchange structure starting from the pair of endpoints $(\hat{x}, \hat{y})$ of the simultaneous expandable message and ending at two sets of states $\mathcal{A} = \{A_1, A_2, \ldots, A_{2^r}\}$ and $\mathcal{B} = \{B_1, B_2, \ldots, B_{2^r}\}$, any $A_i \in \mathcal{A}$ can make a pair with any $B_j \in \mathcal{B}$ (for any such a pair, one could easily find a common message mapping $(\hat{x}, \hat{y})$ to this pair). Therefore, by using a single message block $\hat{m}$ to generate two sets of 2^r random starting nodes respectively from $\mathcal{A}$ and $\mathcal{B}$, we can get 2^{2r} pairs of starting nodes. As a result, the required number of samplings on the random massage block $\hat{m}$ is reduced by a factor of 2^{2r}.

The detailed complexity analysis of the attack using a 2^r-interchange structure is as follows: Denote $L = 2^\ell$. For parameters $g_1 \geq \max(n/2, n-\ell)$, $g_2 \geq 0$, $s \geq 0$ and $0 \leq r \leq \ell/2$ (because the length 2^{2r} of the interchange structure should be less than the message length 2^ℓ), the complexity of phases of the attack 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: $2^{g_2} + 2^{3g_1-n-s-2r} \cdot 2^r \cdot (2^{n-g_2} + 2^{\ell+g_2-n} + 2^{\ell+s-g_1}) + 2^{n/2+2r}$

$$= 2^{g_2} + 2^{3g_1-g_2-s-r} + 2^{3g_1+g_2+\ell-2n-s-r} + 2^{2g_1+\ell-n-r} + 2^{n/2+2r}$$

Compared with the complexity of the attack in Sect.4.1, the difference lies in Phase 3. In the complexity formula of Phase 3, the term 2^{g_2} is the number of nodes developed in the look-ahead procedure; the term $2^{3g_1-n-s-2r}$ is the required number of samplings on the value of message block $\hat{m}$ to get pairs of starting nodes, which is reduced by a factor of 2^{2r} when building a 2^r-interchange structure; the term $2^r \cdot (2^{n-g_2} + 2^{\ell+g_2-n} + 2^{\ell+s-g_1})$ is the time complexity for computing distances of pairs of starting nodes (generated using the same value for $\hat{m}$) from all 2^s target nodes; the term $2^{n/2+2r}$ is the time complexity for building the 2^r-interchange structure.

We first balance the first two terms in Phase 3 by setting $g_2 = 3g_1 - g_2 - s - r$, which gives $g_2 = 3g_1/2 - s/2 - r/2$. Thus, the complexity of Phase 3

becomes $2^{3g_1/2-s/2-r/2}+2^{9g_1/2-3s/2-3r/2+\ell-2n}+2^{2g_1+\ell-n-r}+2^{n/2+2r}$. We then balance Phase 2 and Phase 3 by setting $n+s-g_1 = 3g_1/2-s/2-r/2$, which gives $s = 5g_1/3-r/3-2n/3$. The sum of all dominant terms turns to be $2^{\ell}+2^{n/3+2g_1/3-r/3}+2^{2g_1+\ell-n-r}+2^{n/2+2r}$. Finally, we pick 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 $g_1 = n-\ell$. The sum of dominant terms turns to be $2^{\ell}+2^{n-2\ell/3-r/3}+2^{n-\ell-r}+2^{n/2+2r}$. Considering $n-2\ell/3-r/3 > n-\ell-r$ always holds, the sum of dominant terms is

$$2^{n-2\ell/3-r/3}+2^{n/2+2r}.$$

Note that there is a restriction on r, that is, $r \leq \ell/2$. As a result (see Fig. 12 for a trade-off curve),

- For the case $\ell \leq 3n/11$, we have $n-2\ell/3-r/3 > n/2+\ell > n/2+2r$. We set $r = \ell/2$ to optimize the complexity. Thus, the sum of dominant terms is $2^{n-5\ell/6}$. The optimal complexity is $2^{17n/22}$ obtained for message of length $2^{\ell} = 2^{3n/11}$.
- For the case $3n/11 < \ell \leq n/2$, we set $r = 3n/14-2\ell/7$ to make a balance, which fulfils $r < \ell/2$ in this case. The sum of dominant terms is $2^{13n/14-4\ell/7}$. The optimal complexity is $2^{9n/14}$ obtained for message of length $2^{\ell} = 2^{n/2}$.

5 Improved Preimage Attack on XOR Combiners Based on Multi-Cycles

When the underlying hash functions use the MD construction, and the maximum length of the message is allowed to exceed $2^{n/2}$ blocks, we can further improve previous deep-iterates-based preimage attack. The idea is that we utilize more special nodes in function graphs, which are called cyclic nodes, and exploit a technique named *multi-cycles* as introduced in Sect. 2.7. Recall that, in the deep-iterates-based attack, a key step is to find two starting nodes x_0 and y_0 in functional graphs of f_1 and f_2, such that they reach the selected target nodes $\bar{x}$ and $\bar{y}$ at a common distance. We find that when selecting cyclic nodes as target nodes $\bar{x}$ and $\bar{y}$, the probability of a pair of random node (x_0, y_0) reaching them at a common distance can be greatly amplified. Indeed, cyclic nodes are essentially special deep iterates that are located not only *deep* in the functional graph but also in a *cycle* of the graph. Therefore, for two cyclic nodes in two independent functional graphs, by looping around the cycles, some differences between distances from two random nodes to the two cyclic nodes can be corrected by the difference between the two cycle lengths. More precisely, if the members of a target node pair $(\bar{x}, \bar{y})$ are both cyclic nodes within the largest components in two functional graphs, the probability of a random pair (x_0, y_0) reaching $(\bar{x}, \bar{y})$ at a common distance is amplified by $\#C$ times, the maximum number of cycles that can be used, by using the set of correctable distance bias as stated in Sect. 2.7. Moreover, such a probability amplification comes with almost no increase of complexity at Step 2, which leads to a new complexity trade-off between Steps 2 and 3. Thus, the usage of cyclic nodes and multi-cycles enables

us to reduce the computational complexity of preimage attacks on the XOR combiner.

5.1 Overview of the Attack

Here, we briefly list the *main* steps of our preimage attack on the XOR combiner.

Attack 3: Preimage attack on the XOR combiner based on multi-cycles

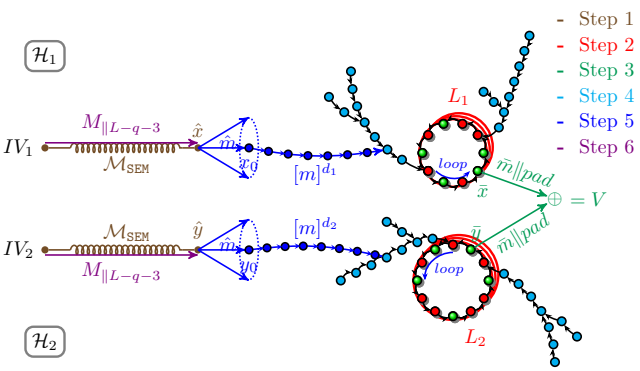
- **Phase 1:** Build a simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ for $\mathcal{H}_1$ and $\mathcal{H}_2$, starting from (IV_1, IV_2) and ending with $(\hat{x}, \hat{y})$.
- **Phase 2:** Collect cyclic nodes within the largest components of functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ and compute the set of correctable distance bias
$$\mathcal{D} = \{i \cdot \Delta L \mod L_1 \mid i = 0, 1, \ldots, \#C\},$$
where L_1 and L_2 are the cycle lengths of the largest components of $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, respectively, and $\Delta L = L_2 - L_1 \mod L_1$.
- **Phase 3:** Find a set $\mathcal{S}$ (of size 2^s) of tuples of the form $((x, y), w)$ such that x and y are cyclic nodes located in the largest components of $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, respectively, and $(x, y) \xrightarrow{w} (a, b)$ 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, V is the target hash digest.
- **Phase 4:** Find a message fragment $M_{\texttt{Link}}$ that maps $(\hat{x}, \hat{y})$ to a pair of target nodes $(\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$.
That is done by first start from $(\hat{x}, \hat{y})$, enumerate a message block $\hat{m}$ to find a pair of states (x_0, y_0) with $x_0 = h_1(\hat{x}, \hat{m})$ and $y_0 = h_2(\hat{y}, \hat{m})$, such that in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, their distance difference $d_1 - d_2 \mod L_1$ from $(\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$ belongs to $\mathcal{D}$. Suppose the common distance after correcting by the cycle lengths is $q \triangleq d_1 + i \cdot L_1 = d_2 + j \cdot L_2$, and define a message fragment $M_{\texttt{Link}} \triangleq \hat{m} \parallel [m]^q$.

Up to now, we are able to derive a message $M_{\|L-q-3}$ from the simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ with an appropriate length $L-q-3$ and produce a preimage message:

$$\begin{aligned} \boldsymbol{M} &\triangleq M_{\|L-q-3} \parallel M_{\texttt{Link}} \parallel \bar{m} \parallel pad \\ &= M_{\|L-q-3} \parallel \hat{m} \parallel [m]^q \parallel \bar{m} \parallel pad. \end{aligned}$$

such that

$$(IV_1, IV_2) \xrightarrow{M_{\|L-q-3}} (\hat{x}, \hat{y}) \xrightarrow{\hat{m} \| [m]^q} (\bar{x}, \bar{y}) \xrightarrow{\bar{m} \,\|\, pad} (\mathcal{H}_1(\boldsymbol{M}), \mathcal{H}_2(\boldsymbol{M})) : \mathcal{H}_1(\boldsymbol{M}) \oplus \mathcal{H}_2(\boldsymbol{M}) = V$$



By balancing the complexities of these steps, we obtain an optimal complexity of $2^{5n/8}$.

5.2 Details of the Preimage Attack on the XOR Combiner Using Multi-Cycles

In this section, we present the completed description of the attack procedure and complexity evaluation. We point out the length of our preimage is at least $2^{n/2}$ blocks due to the usage of (multi-) cycles.

Attack Procedure. Denote by V the target hash digest. Suppose the attacker is going to produce a preimage message with a length L. The value of L will be discussed later. The attack procedure is described below.

Detailed Steps of Attack 3

1. Build a simultaneous expandable message structure $\mathcal{M}_{\mathtt{SEM}}$ with a range of length being roughly $[n^2, L]$, starting from the initial state pair (IV_1, IV_2) and ending with a pair of final state $(\hat{x}, \hat{y})$ such that for each positive integer κ of an integer interval, there is a message $M_{\|\kappa}$ with a block length κ in $\mathcal{M}_{\mathtt{SEM}}$ such that $(IV_1, IV_2) \xrightarrow{M_{\|\kappa}} (\hat{x}, \hat{y})$.
2. Fix a single-block message m, and construct $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$.
   (a) Run the cycle search algorithm several times to find the cycle length L_1 (resp. L_2) and cyclic nodes within the largest components in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$). Store all the cyclic nodes in a table $\mathcal{T}_1$ (resp. $\mathcal{T}_2$), denote $\mathcal{T}_1 = \{x_1, x_2, \ldots, x_{L_1}\}$ (resp. $\mathcal{T}_2 = \{y_1, y_2, \ldots, y_{L_2}\}$). Without loss of generality, assume $L_1 \leq L_2$.
   (b) Compute $\#C = \lfloor L/L_1 \rfloor$ as the maximum number of cycles that can be used to correct the distance bias. Compute $\Delta L = L_2 \mod L_1$,

and then compute the set of correctable distance bias:

$$\mathcal{D} = \{i \cdot \Delta L \mod L_1 \mid i = 0, 1, 2, \ldots, \#C\}.$$

3. Find a set

$$\mathcal{S} = \{((\bar{x}_1, \bar{y}_1), \bar{m}_1), ((\bar{x}_2, \bar{y}_2), \bar{m}_2)\}, \cdots, ((\bar{x}_{2^s}, \bar{y}_{2^s}), \bar{m}_{2^s})$$

such that $\bar{x}_i$ and $\bar{y}_i$ are cyclic nodes respectively located in the largest components of $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, and $(x_i, y_i) \xrightarrow{w} (a_i, b_i)$ and $h_1(a_i, pad) \oplus h_2(b_i, pad) = V$, where pad is the final block of the (padded) preimage message of length L, V is the target hash digest. The search procedure is described as follows.

   (a) Initialize a set $\mathcal{S}$ as empty.
   (b) Select a random single-block message w.
   (c) Compute $h_1^*(x, w \,\|\, pad)$ for each cyclic node x in $\mathcal{T}_1$, and store them in a table $\mathcal{T}_1'$.
   (d) Similarly, for each cyclic node y in $\mathcal{T}_2$, compute $h_2^*(y, w \,\|\, pad) \oplus V$, and look for matches with elements in $\mathcal{T}_1'$. If it is matched to some $h_1^*(x, w \,\|\, pad)$, we have $h_2^*(y, w \,\|\, pad) \oplus h_1^*(x, w \,\|\, pad) = V$, store $((x, y), w)$ in $\mathcal{S}$.
   (e) If $\mathcal{S}$ contains less than 2^s elements, goto Step 3b and repeat the search procedure.

   Hereafter, we call the pair of nodes $(\bar{x}_i, \bar{y}_i)$ in $((\bar{x}_i, \bar{y}_i), \bar{m}_i) \in \mathcal{S}$ a pair of target nodes.
4. Run Alg.5 with a parameter t to develop 2^t nodes in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$), and store them in a data structure $\mathcal{G}_1$ (resp. $\mathcal{G}_2$). Moreover,
   (a) Store at each node its distance from a particular target node (say target node $\bar{x}_1$ (resp. $\bar{y}_1$), together with its distance from the cycle (*i.e.*, its height, similar to Phase 3 in Section 5 of [PW14]).
   (b) Store the distance of other target nodes $\bar{x}_i$ (resp. $\bar{y}_i$) to this particular target node $\bar{x}_1$ (resp. $\bar{y}_1$) in a table $\mathcal{T}_x$ (resp. $\mathcal{T}_y$) by iterating f_1 (resp. f_2) along the cycle.
   (c) Thus, when the distance of a node from the particular target node and that from the cycle is known from $\mathcal{G}_1$ (resp. $\mathcal{G}_2$), the distances of this node from all the other target nodes can be immediately deduced from $\mathcal{T}_x$ (resp. $\mathcal{T}_y$). Specifically, suppose the distance of a node x_0 from $\bar{x}_1$ is d_1 and its height is e_1, and suppose the distance of a target node $\bar{x}_i$ from $\bar{x}_1$ is d_i; then, the distance of x_0 from $\bar{x}_i$ is $d_1 - d_i$ if $d_i \leq (d_1 - e_1)$, and $L_1 - d_i + d_1$ if $d_i > (d_1 - e_1)$.
5. Find a message $M_{\texttt{Link}}$ that maps $(\hat{x}, \hat{y})$ to a pair of target nodes $(\bar{x}, \bar{y})$ for some $((\bar{x}, \bar{y}), \bar{m}) \in \mathcal{S}$. We search for such a linking message among a set of special messages: $M_{\texttt{Link}} = \hat{m} \,\|\, m \,\|\, m \,\|\, \cdots \,\|\, m$, where $\hat{m}$ is a random single-block message, and m is the fixed message at Step 2. The search procedure is as follows.

(a) Select a random message block w, and compute $x_0 = h_1(\hat{x}, w)$ and $y_0 = h_2(\hat{y}, w)$.
(b) Compute a chain by iteratively applying f_1 (resp. f_2) to update x_0 (resp. y_0) until either of the following two cases occurs.
- The chain length reaches 2^{n-t}. In this case, goto Step 5a;
- The chain encounters a node stored in $\mathcal{G}_1$ (resp. $\mathcal{G}_2$). In this case, compute the distance of x_0 (resp. y_0) to every target node $\bar{x}_i$ (resp. $\bar{y}_i$) as described in Step 4c, and denote it as $d_{\bar{x}_i}$ (resp. $d_{\bar{y}_i}$). Examine whether $(d_{\bar{x}_i} - d_{\bar{y}_i} \mod L_1)$ is in $\mathcal{D}$. If it is, set $d_1 \triangleq d_{\bar{x}_i}$ and $d_2 \triangleq d_{\bar{y}_i}$, derive the corresponding j and k such that $d_1 + j \cdot L_1 = d_2 + k \cdot L_2$ holds. Set $q \triangleq d_1 + j \cdot L_1 = d_2 + k \cdot L_2$, and then $M_{\mathtt{Link}} \triangleq \hat{m} \parallel [m]^q$. Besides, set $\hat{m} \triangleq w$ and $\bar{m} \triangleq \bar{m}_i$. Otherwise, goto Step 5a.

6. Derive a message $M_{\|L-q-3}$ with a block length of $L - q - 3$ from the expandable message $\mathcal{M}_{\mathtt{SEM}}$.
7. Produce a preimage $\boldsymbol{M}$ of the target hash digest V as follows:

$$\begin{aligned}\boldsymbol{M} &\triangleq M_{\|L-q-3} \parallel M_{\mathtt{Link}} \parallel \bar{m} \parallel pad \\ &= M_{\|L-q-3} \parallel \hat{m} \parallel [m]^q \parallel \bar{m} \parallel pad.\end{aligned}$$

Complexity Analysis. For parameters $L \geq 2^{n/2}$, $s \geq 0$ and $t \geq 0$, the complexity of the steps of the attack is given below (ignore the constant and polynomial factors for simplicity of description).

- **Step 1:** $L + n^2 \cdot 2^{n/2}$ (refer to Sect. 2.6);
- **Step 2:** $2^{n/2} + L/L_1 \approx 2^{n/2} + 2^{-n/2} \cdot L \approx 2^{n/2}$;
- **Step 3:** $2^{s+n/2}$;
  One execution of the search procedure has a complexity of $L_1 + L_2$, and contributes to $L_1 \cdot L_2$ pairs. As $L_1 \cdot L_2 = \Theta(2^n)$, one tuple can be obtained by a constant number of executions. Hence, the number of necessary executions is $\Theta(2^s)$, and the complexity of this step is $\Theta(2^{s+n/2})$.
- **Step 4:** $2^t + 2^{n/2}$;
  The complexity of developing 2^t nodes and computing their distance to a particular target node is 2^t (refer to Alg. 5 and Step 4a). The complexity to compute the distance of all the other target nodes to the particular target node is upper bounded by $2^{n/2}$ (refer to the expectation of the maximum cycle length in Thm. 3). Hence, the complexity of this step is $2^t + 2^{n/2}$.
- **Step 5:** $2^{2n-t-s}/L$;
  One execution of the search procedure requires a time complexity of 2^{n-t}. Clearly, a constant factor of both chains encounter nodes stored in $\mathcal{G}_1$ and $\mathcal{G}_2$. We mainly need to evaluate the probability of deriving a common distance for each chain. For every pair of target nodes $(\bar{x}_i, \bar{y}_i)$, the value of $d_{x_i} - d_{y_i}$ is equal to a correctable distance bias in $\mathcal{D}$ with a probability of $\#C \cdot 2^{-n/2} \approx L \cdot 2^{-n}$. Since there are 2^s pairs of target nodes, the success probability of each chain

is $L \cdot 2^{s-n}$. Hence, the total number of chains is $2^{n-s}/L$, and the complexity of this step is $2^{n-t} \cdot 2^{n-s}/L = 2^{2n-t-s}/L$.
- **Steps 6 and 7:** $\mathcal{O}(L)$.

The overall complexity is computed as (denote $L = 2^\ell$)

$$2^\ell + 2^{s+n/2} + 2^t + 2^t + 2^{n/2} + 2^{2n-t-s-\ell},$$

where the complexity of Step 2 is ignored.

Now, we search for parameters t and s that give the lowest complexity. First, we balance Step 3 and Step 4 by setting $s + n/2 = t$. That gives $s = t - n/2$. The complexity becomes (ignoring constant factors) $2^\ell + 2^t + 2^{5n/2-2t-\ell}$. We then make a balance by setting $t = 2n - 2t + n/2 - \ell$, *i.e.*, $t = 5n/6 - \ell/3$. Thus, the total complexity becomes

$$2^\ell + 2^{5n/6-\ell/3}.$$

Hence,

- for the case $n/2 \leq \ell \leq 5n/8$, the final complexity is $2^{5n/6-\ell/3}$;
- for the case $5n/8 < \ell$, the final complexity is 2^ℓ.

The optimal complexity is $2^{5n/8}$, obtained for messages of length $2^{5n/8}$ (see Fig. 12 for a trade-off curve).

5.3 Optimizing the Multi-Cycles-Based Preimage Attack on the XOR Combiner Using the Interchange Structure

Again, similar to the previous deep-iterates-based preimage attack, this multi-cycles-based preimage attack on the XOR combiner can also be improved using an interchange structure. The complexity of the attack using a 2^r-interchange structure is analysed as follows.

Denote $L = 2^\ell$. For parameters $t \geq n/2$, $\ell \geq n/2$ and $0 \leq r \leq \ell/2$ (because the length 2^{2r} of the interchange structure should be no larger than the message length $L = 2^\ell$), the complexity of each step is as follows:

Step 1: $2^\ell + n^2 \cdot 2^{n/2}$ **Step 2:** $2^{n/2} + 2^{\ell-n/2}$ **Step 3:** $2^{s+n/2}$

Step 4: $2^t + 2^{n/2}$ **Step 5:** $2^{n-t+r} \cdot 2^{n-s-2r-\ell} + 2^{n/2+2r}$ **Step 6:** 2^ℓ

The sum of dominant terms is

$$2^\ell + 2^{s+n/2} + 2^t + 2^{2n-t-s-r-\ell} + 2^{n/2+2r}.$$

We first balance the last four terms by setting $s + n/2 = t = 2n - t - s - r - \ell = n/2 + 2r$. Thus, $t = 11n/14 - 2\ell/7$, $s = 2n/7 - 2\ell/7$, and $r = n/7 - \ell/7$. Note that for $\ell \geq n/2$, we have $n/7 - \ell/7 \leq n/14 < n/2$. Thus, the restriction $r \leq \ell/2$ always holds in this setting. The total complexity turns to be

$$2^\ell + 2^{11n/14-2\ell/7}.$$

Hence,

– for the case $n/2 \leq \ell \leq 11n/18$, the final complexity is $2^{11n/14-2l/7}$;
– for the case $11n/18 < \ell$, the final complexity is 2^{ℓ}.

The optimal complexity is $2^{11n/18}$, obtained for messages of length $2^{11n/18}$ (see Fig. 12 for a trade-off curve).

6 Second-Preimage Attack on Concatenation Combiners Based on Deep Iterates

In this section, we introduce the first second-preimage attack faster than 2^n on concatenation combiners of MD hashes.

In this attack, we are given a challenge message $\boldsymbol{M} = m_1 \parallel m_2 \parallel \ldots \parallel m_L$, and our goal is to find another message $\boldsymbol{M}'$ such that $\mathcal{H}_1(\boldsymbol{M}') \parallel \mathcal{H}_2(\boldsymbol{M}') = \mathcal{H}_1(\boldsymbol{M}) \parallel \mathcal{H}_2(\boldsymbol{M})$ (or equivalently $\mathcal{H}_1(\boldsymbol{M}') = \mathcal{H}_1(\boldsymbol{M})$ and $\mathcal{H}_2(\boldsymbol{M}') = \mathcal{H}_2(\boldsymbol{M})$). We denote the sequence of internal states computed during the invocation of h_1 (respectively, h_2) on $\boldsymbol{M}$ by $a_0, a_1, \ldots, a_L$ (respectively, $b_0, b_1, \ldots, b_L$).

The general framework of our attack is similar to the one of the long message second-preimage attack on a single MD hash proposed by Kelsey and Schneier and described in Sect.2.2. 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 challenge message $\boldsymbol{M} = m_1 \parallel \ldots \parallel m_L$. Our goal is then to "connect" to one of the state pairs (a_i, b_i) using a different message prefix of the same size. Once we manage to achieve this, we can reuse the same message suffix as in $\boldsymbol{M}$ and obtain a second preimage.

There are two main challenges in this approach, where the main challenge is to connect to some state pair (a_i, b_i) generated by $\boldsymbol{M}$ from a different message. The secondary challenge is to ensure that the connected message prefixes are of the same length. We overcome the secondary challenge by building a simultaneous expandable message for two Merkle-Damgård hash functions, as described in Sect. 2.6.

A much more difficult challenge is to actually connect to the challenge 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* introduced in Sect.3. Recall that, the interchange structure consists of an initial state pair (a, b), a set of message fragments $\mathcal{M}$ and two sets of internal states $\mathcal{A}$ (for $\mathcal{H}_1$) and $\mathcal{B}$ (for $\mathcal{H}_2$) such that for any value $A \in \mathcal{A}$ and any value $B \in \mathcal{B}$, it is possible to efficiently construct a message $M_{A,B} \in \mathcal{M}$ that maps (a, b) to (A, B). Assume that there exists an index $i \in \{1, 2, \ldots, L\}$ such that $a_i \in \mathcal{A}$ and $b_i \in \mathcal{B}$; 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 $\mathcal{A}$ and $\mathcal{B}$ is not efficient enough.

Here, as in the deep-iterate-based preimage attack on XOR combiner, we use deep iterates in functional graphs of $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2 \triangleq h_2(\cdot, m)$ (as a

result, it is not applicable when any one of the underlying hash functions are of the HAIFA framework). More specifically, our goal is to find a state pair (a_p, b_p) composed of two deep iterates in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, respectively.[21] 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 simultaneous expandable message, this gives the desired second preimage.

Next, as in previous attack, we start by providing a high-level overview of the attack and then give technical details.

6.1 Overview of the Attack

The attack is composed of three main phases.

Attack 4: Second-preimage attack on the concatenation combiner based on deep iterates

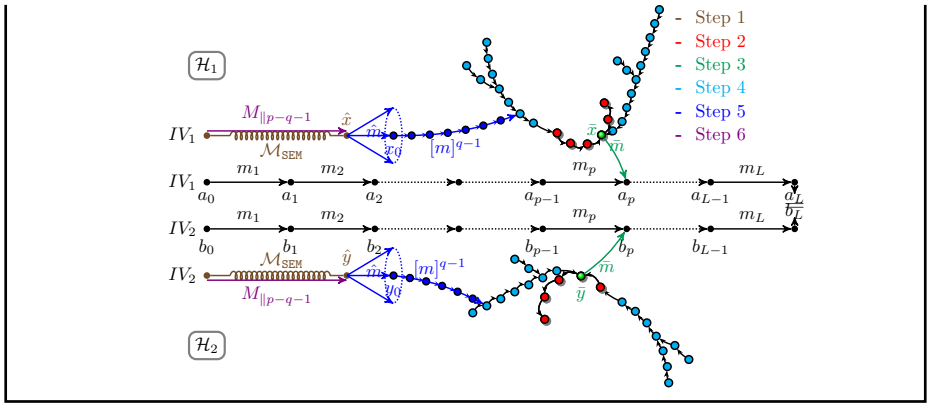
- **Phase 1:** Build a simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ for $\mathcal{H}_1$ and $\mathcal{H}_2$ as described in Sect.2.6, starting from the initial state pair (IV_1, IV_2) and ending at a final state pair $(\hat{x}, \hat{y})$.
- **Phase 2:** Find a pair of states $(\bar{x}, \bar{y})$, 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 challenge message). Moreover, the state pair $(\bar{x}, \bar{y})$ should have the special property that will be defined in the detailed description of this phase.
- **Phase 3:** Start from $(\hat{x}, \hat{y})$ and compute a message fragment $\hat{M}_{\|q}$ (shorter than $p-1$) such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}_{\|q}} (\bar{x}, \bar{y})$. This phase can be performed efficiently due to the special property of $(\bar{x}, \bar{y})$.

To compute the second preimage, we pick $M_{\|p-q-1}$ from the simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$, giving $(IV_0, IV_1) \xrightarrow{M_{\|p-q-1}} (\hat{x}, \hat{y})$, and concatenate $M_{\|p-q-1} \,\|\, \hat{M}_{\|q} \,\|\, \bar{m}$ 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

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

Altogether, we obtain the second preimage: $\boldsymbol{M'} = M_{\|p-q-1} \,\|\, \hat{M}_{\|q} \,\|\, \bar{m} \,\|\, m_{p+1} \,\|\, \ldots \,\|\, m_L$.

[21] 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.



Complexity Analysis. 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 trade-off 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 $\ell = 3n/4$ (see Fig. 12 for a trade-off curve). The attack is faster than 2^n (Joux's preimage attack) for[22] $\ell > n/3$. 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 Sect. 6.3.

6.2 Details of the Second-Preimage Attack on Concatenation Combiners using Deep Iterates

Details of Phase 1 can be found in Sect. 2.6.

Details of Phase 2: Finding a Target State Pair. In the second phase, we fix an arbitrary message block m, giving rise to the functional graph $\mathcal{FG}_{f_1}$ of $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $\mathcal{FG}_{f_2}$ of $f_2(\cdot) \triangleq h_2(\cdot, m)$ 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 a 2^{n-g_1}-th iterate in $\mathcal{FG}_{f_1}$ and $\bar{y}$ is a 2^{n-g_1}-th iterate in $\mathcal{FG}_{f_2}$.
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.

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

Phase 2 of Attack 4: Finding a target state pair

1. Fix an arbitrary single-block value m and get $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$.
2. Expand $\mathcal{FG}_{f_1}$ using Alg. 5 with parameter g_1. Store all encountered 2^{n-g_1}-th iterates in a table $\mathcal{T}_1$.
3. Expand $\mathcal{FG}_{f_2}$ using Alg. 5 with parameter g_1. Store all encountered 2^{n-g_1}-th iterates in a table $\mathcal{T}_2$.
4. For single-block values $m' = 0, 1, \ldots$, perform the following steps:
   (a) For each node $x \in \mathcal{T}_1$, 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 $\mathcal{T}_1'$, sorted according to the index i of a_i.
   (b) Similarly, for each node $y \in \mathcal{T}_2$ 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 $\mathcal{T}_1'$. 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 less 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 Alg. 5) is approximately 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 $\mathcal{T}_1$ and $\mathcal{T}_2$ 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. According to Observation 1 in Sect. 2.7, the expected size of $\mathcal{T}_1$ and $\mathcal{T}_2$ (that contain iterates of depth 2^{n-g_1}) is close to 2^{g_1}. According to the birthday paradox, the expected size of $\mathcal{T}_1'$ is approximately $L \cdot 2^{g_1-n}$. Similarly, the number of matches $y' = b_j$ is also approximately $L \cdot 2^{g_1-n}$. 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 approximately $1/L \cdot 2^{2n-2g_1}$ 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 $p = \Theta(L)$.

Details of Phase 3: Hitting the Target State Pair. In the third and final phase, we start from the pair of endpoints $(\hat{x}, \hat{y})$ of the simultaneous expandable message constructed in Phase 1 and compute a message fragment $\hat{M}_{\|q}$ of length $q < p - 1$ such that $(\hat{x}, \hat{y}) \xrightarrow{\hat{M}_{\|q}} (\bar{x}, \bar{y})$. Like in the deep-iterates-based preimage attack on the XOR combiner, here, we again 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 $\hat{m}$, which gives points $x_0 = h_1(\hat{x}, \hat{m})$ and $y_0 = h_2(\hat{y}, \hat{m})$. 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 $L' = 2^{n-g_1}$. 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 $\hat{M} = \hat{m} \,\|\, [m]^{q-1}$ (where $[m]^{q-1}$ denotes the concatenation of $q-1$ message blocks m), which is of length $q < p-1$. If $\bar{x}$ 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 $\hat{m}$ and start again.

Since $\bar{x}$ (resp. $\bar{y}$) is an iterate of depth 2^{n-g_1} in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$), it is an endpoint of a chain of states of length $L' = 2^{n-g_1}$ (such a chain was computed in Phase 2). According to Observation 2, the probability that $\bar{x}$ and $\bar{y}$ will be encountered at the same distance from arbitrary starting points x_0 and y_0 of chains is 2^{n-3g_1}. The probability calculation gives rise to the conclusion that we need to compute approximately 2^{3g_1-n} chains from different starting points. Each chain is of length up to $L' = 2^{n-g_1}$. This gives 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 the attack is not faster than Joux's preimage attack.

To optimize the algorithm, as we did in the deep-iterates-based preimage attack, we use a look-ahead procedure by further expanding the graphs of f_1 and f_2. The difference is that since we only select a single pair of deep iterate as the target, we do not use the distinguished point technique[23] [24]. We pick a parameter $g_2 > g_1$ and execute the algorithm below.

Phase 3 of Attack 4: Hitting the target state pair

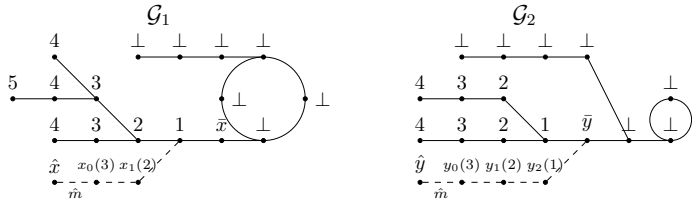
1. Develop 2^{g_2} nodes in $\mathcal{FG}_{f_1}$ (and $\mathcal{FG}_{f_2}$) (as specified in Alg. 5) with the following modifications.

[23] One may ask why we did not compute a larger set $\mathcal{S}$, as we did in Phase 3 of Attack 2. The reason for this is that it can be shown that in this case, a set of size 1 is optimal.

[24] One may also ask why we did not use cyclic nodes and multi-cycles to further improve this second-preimage attack on concatenation combiners as we did for the preimage attack on XOR combiners. The reason is that optimization on Phase 2 of Attack 4 has reached it limitation because of the limited number of candidate state pairs for $(\bar{x}, \bar{y})$. Thus, the complexity of Phase 2 becomes the bottleneck and cannot be improved using cyclic nodes.

- Store at each node its distance from $\bar{x}$ in $\mathcal{FG}_{f_1}$ (or $\bar{y}$ in $\mathcal{FG}_{f_2}$) (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 $\bar{y}$) 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 $\perp$. This special value is also used for nodes whose distance from $\bar{x}$ is greater than L'.
- The evaluated nodes for $\mathcal{FG}_{f_1}$ ($\mathcal{FG}_{f_2}$) are stored in a data structure $\mathcal{G}_1$ ($\mathcal{G}_2$).

2. For single-block values $\hat{m} = 0, 1, \ldots$, compute $x_0 = h_1(\hat{x}, \hat{m})$ and $y_0 = h_2(\hat{y}, \hat{m})$ and repeat the following step:
   (a) Compute the chains $\vec{x}$ and $\vec{y}$ up to maximal length $L' = 2^{n-g_1}$ or until they hit $\mathcal{G}_1$ and $\mathcal{G}_2$ (respectively).
   - If $\vec{x}$ (or $\vec{y}$) does not hit $\mathcal{G}_1$ ($\mathcal{G}_2$), return to Step 2.
   - Otherwise, once $\vec{x}$ ($\vec{y}$) hits $\mathcal{G}_1$ ($\mathcal{G}_2$), 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 $\hat{M} = \hat{m} \parallel [m]^i = \hat{m} \parallel [m]^{q-1}$ as output.



The time complexity of Step 1 is approximately 2^{g_2}. As concluded above, in Step 2, we perform approximately 2^{3g_1-n} trials on the value of $\hat{m}$ before finding two starting points x_0 and y_0 at the same distance from $\bar{x}$ and $\bar{y}$. According to the analysis of Sect. 2.7, each trial requires approximately 2^{n-g_2} computation (before hitting $\mathcal{G}_1$ and $\mathcal{G}_2$). 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 a 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 has been presented in the closely related algorithm in Sect. 4.2.

6.3 Optimizing the Deep-Iterates-Based Second-Preimage Attack on Concatenation Combiners using an Interchange Structure

Similar to the deep-iterates-based preimage attack on the XOR combiner, this deep-iterates-based second-preimage attack can also be slightly improved using an interchange structure. The detailed complexity analysis of the attack using a 2^r-interchange structure is as follows:

Denote $L = 2^\ell$. For parameters $g_1 \geq \max(n/2, n-\ell)$, $g_2 \geq 0$, and $0 \leq r \leq \ell/2$ (because the length 2^{2r} of the interchange structure should be less than the message length 2^ℓ), the complexity of the phases of the attack is (ignoring constant factors)

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

Phase 3: $2^{g_2} + 2^{3g_1-n-2r} \cdot 2^{r+n-g_2} + 2^{n/2+2r} = 2^{g_2} + 2^{3g_1-g_2-r} + 2^{n/2+2r}$

Compared with the complexity of the attack in Sect.6.1, the difference lies in Phase 3. In the complexity formula of Phase 3, the term 2^{g_2} is the number of nodes developed in the look-ahead procedure; the term 2^{3g_1-n-2r} is the required number of samplings on the random massage block $\hat{m}$ when trying to find a pair of starting nodes reaching the pair of 2^{n-g_1}-th iterates $(\bar{x}, \bar{y})$ at a common distance; the term 2^{r+n-g_2} is the time complexity for computing the distances of a set of 2^r starting nodes (generated using the same value for $\hat{m}$) from a target node; and the term $2^{n/2+2r}$ is the time complexity for building a 2^r-interchange structure.

We first balance the first two terms in Phase 3 by setting $g_2 = 3g_1 - g_2 - r$, which gives $g_2 = 3g_1/2 - r/2$. The sum of all dominant terms is

$$2^\ell + 2^{2n-g_1-\ell} + 2^{3g_1/2-r/2} + 2^{n/2+2r}.$$

For $\ell > 7n/17$, we set $2n - g_1 - \ell = 3g_1/2 - r/2 = n/2 + 2r$, which gives $g_1 = 19n/22 - 5\ell/11$ and $r = 7n/22 - 3\ell/11$. The total complexity is then $2^\ell + 2^{25n/22-6\ell/11}$. For $\ell > 25n/34$, we have $25n/22 - 6\ell/11 < \ell$. Thus, the time complexity is 2^ℓ for $\ell > 25n/34$. Note that, $g_1 = 19n/22 - 5\ell/11$ fulfils the restriction $g_1 \geq \max(n/2, n-\ell)$ for $n/4 \leq \ell \leq 4n/5$, and $r = 7n/22 - 3\ell/11$ fulfils the restriction $2r < l$ as long as $\ell > 7n/17$. Thus, the time complexity is $2^{25n/22-6\ell/11}$ for $7n/17 < \ell \leq 25n/34$.

For $\ell \leq 7n/17$, we directly set $r = \ell/2$ (the maximum under the restriction $2r < \ell$) to optimize the complexity (because as shown next, the balanced sum $2^{2n-g_1-\ell} + 2^{3g_1/2-r/2}$ is greater than $2^{n/2+2r}$ under the restrictions $g_1 \geq \max(n/2, n-\ell)$, $2r < \ell$ and $\ell \leq 7n/17$). The formula is $2^{2n-g_1-\ell} + 2^{3g_1/2-\ell/4} + 2^{n/2+\ell}$. We balance the first two terms by setting $2n - g_1 - \ell = 3g_1/2 - \ell/4$, from which we deduced that $g_1 = 4n/5 - 3\ell/10$ (fulfills the restriction $g_1 \geq \max(n/2, n-\ell)$ as long as $\ell > 2n/7$). Then, the complexity is $2^{6n/5-7\ell/10} + 2^{n/2+\ell}$. Since $6n/5 - 7\ell/10 \geq n/2 + \ell$ for $\ell \leq 7n/17$, the total time complexity is then $2^{6n/5-7\ell/10}$. It is no less than 2^n for $\ell < 2n/7$.

Thus, the final time complexity of this attack using interchange structure is summarized as follows:

– For the case $\ell < 2n/7$, the complexity is 2^n achieved by Joux's attack;
– For the case $2n/7 \leq \ell \leq 7n/17$, the complexity of this attack is $2^{6n/5-7\ell/10}$;
– For the case $7n/17 < \ell \leq 25n/34$, the complexity of this attack is $2^{25n/22-6\ell/11}$;
– For the case $\ell > 25n/34$, the complexity of this attack is 2^{ℓ}.

The optimal complexity is $2^{25n/34}$, obtained for messages of length $2^{25n/34}$ (see Fig. 12 for a trade-off curve).

7 Second-Preimage Attack on the Zipper Hash

In this section, we present the first second-preimage attack on the Zipper hash, which is applicable for *idealized* compression functions and hence a generic attack. Again, the attack is based on the deep iterates and multi-cycles in the functional graphs defined by $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$ with a fixed single-block message value m. The general framework is similar to that of the above ones on combiners of MD hashes. However, some special specifications on the Zipper hash allow the attacker to choose an optimal configuration for the attack and to launch a more efficient connecting phase in the attack. More precisely, as opposed to the two parallel combiners, in the Zipper hash, the message length is placed in the middle of the two passes. Thus, when we first connect our crafted message to the challenge message on an internal state in the second pass, the message prefix of our crafted message is fixed. This prefix does not include the length padding. As a result, the length of our crafted message is not necessarily equal to the length of the challenge message. Thus, we can choose a proper length for our crafted message that optimizes the attack complexity. A further uniqueness of the Zipper hash is that its second pass processes the message blocks in a reversed order. Thus, in the attack, when looking for a pair of nodes $(\check{x}, \check{y})$ reaching two predefined nodes of deep iterates $(\bar{x}, \bar{y})$ at a common distance, $\check{x}$ and $\check{y}$ are computed with different message blocks. This enables us to launch an efficient meet-in-the-middle procedure during the connecting phase. Accordingly, Joux's multi-collision (see Sect.2.1) is used to facilitate the meet-in-the-middle procedure, and the previous simultaneous expandable message in Sect.2.6 is fine-tuned to adapt to the Zipper hash.

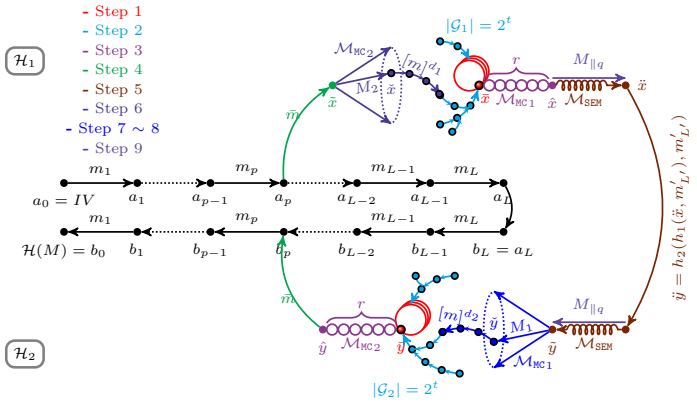
7.1 Overview of the Attack

Given a message $\boldsymbol{M} = m_1 \| \cdots \| m_L$, the goal of the second-preimage attack on the Zipper hash is to find another message $\boldsymbol{M'}$ such that $\mathcal{H}_2(\mathcal{H}_1(IV, \boldsymbol{M}), \overleftarrow{\boldsymbol{M}}) = \mathcal{H}_2(\mathcal{H}_1(IV, \boldsymbol{M'}), \overleftarrow{\boldsymbol{M'}})$, where $\overleftarrow{\boldsymbol{M}}$ is a message generated by reversing the order of message blocks of $\boldsymbol{M}$ (we call $\overleftarrow{\boldsymbol{M}}$ the reverse of $\boldsymbol{M}$), *i.e.*, $\overleftarrow{\boldsymbol{M}} = m_L \| m_{L-1} \| \cdots \| m_1$, and $\overleftarrow{\boldsymbol{M'}}$ is the reverse of $\boldsymbol{M'}$. Here, we briefly list the *main* steps of the attack.

Attack 5: Second-preimage attack on the Zipper hash

- **Phase 1:** Get a cyclic node $\bar{x}$ (resp. $\bar{y}$) located in the largest component of $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$); Get the cycle length L_1 (resp. L_2), and compute the set of correctable distance bias $\mathcal{D} = \{i \cdot \Delta L \mod L_1 \mid i = 0, 1, \ldots, \#C\}$, where $\Delta L = L_2 \mod L_1$ and $\#C$ is the number of cycles to the maximum. After that, generate a Joux's multi-collision $\mathcal{M}_{\texttt{MC1}}$ (resp. $\mathcal{M}_{\texttt{MC2}}$) starting from $\bar{x}$ (resp. $\bar{y}$) and denote its final endpoints by $\hat{x}$ (resp. $\hat{y}$).
- **Phase 2:** Build a cascade simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ across the two passes starting from $\hat{x}$ and denote its final endpoint by $\tilde{y}$.
- **Phase 3:** Find a message block $\bar{m}$ mapping the final endpoint $\hat{y}$ of the second Joux's multi-collision $\mathcal{M}_{\texttt{MC2}}$ to one of the chaining states b_p in the second pass of the original message; Then, in the first pass, use $\bar{m}$ to update the corresponding chaining state a_p to a state $\tilde{x}$.
- **Phase 4:** Find a message $M_{\texttt{Link}}$ such that $(\tilde{x}, \tilde{y}) \xrightarrow{M_{\texttt{Link}}} (\bar{x}, \bar{y})^a$.
  This is a meet-in-the-middle procedure. First, exploit the messages in the two Joux's multi-collisions $\mathcal{M}_{\texttt{MC1}}$ and $\mathcal{M}_{\texttt{MC2}}$ to map $(\tilde{x}, \tilde{y})$ to two independent sets of starting nodes, compute and store their distances from target node $\bar{x}$ and $\bar{y}$ in two tables $\mathcal{T}_1$ and $\mathcal{T}_2$ independently. Then, find a match between $\mathcal{T}_1$ and $\mathcal{T}_2$ (check whether the difference between some stored distances is correctable by values in $\mathcal{D}$), denote the matched distances by d_1 and d_2, set the common distance $d \triangleq d_1 + i \cdot L_1 = d_2 + j \cdot L_2$ for some i and j, and retrieve the corresponding messages $M_2 \in \mathcal{M}_{\texttt{MC2}}$ and $M_1 \in \mathcal{M}_{\texttt{MC1}}$ that generate the corresponding starting nodes.

At the end, select a message suffix $M_{\|q}$ with a proper block length $q = L' - p - 2r - d$ from the simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ and construct a second preimage $\boldsymbol{M'}$: $m_1 \,\|\, \cdots \,\|\, m_p \,\|\, M_2 \,\|\, [m]^d \,\|\, M_1 \,\|\, M_{\|L'-p-2r-d}$.



[a] Rigorously, we should write $\tilde{y} \xleftarrow{M_{\mathtt{Link}}} \bar{y}$.

There are two main differences between the attack on the Zipper hash and the second-preimage attack on concatenation combiners (Attack 4) and the preimage attacks on XOR combiners (Attacks 2 and 3). One is that linking $\tilde{x}$ to $\bar{x}$ and $\tilde{y}$ to $\bar{y}$ can be carried out independently, resulting in a meet-in-the-middle-like effect. The other is that the message length is embedded inside the expandable message $\mathcal{M}_{\mathtt{SEM}}$, which enables us to choose the length of second preimage to optimize the complexity.

7.2 Details of the Second-Preimage Attack on the Zipper Hash

In this subsection, we present the detailed procedure of Attack 5.

Detailed Steps of Attack 5

1. Fix an arbitrary single-block message value m and get $f_1(\cdot) = h_1(\cdot, m)$ and $f_2(\cdot) = h_2(\cdot, m)$.
   (a) Run the cycle search algorithm several times to locate the largest cycles $\mathcal{C}_1$ and $\mathcal{C}_2$ in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, get the cycle lengths L_1 and L_2. Without loss of generality, assume $L_1 \leq L_2$.
   (b) Pick a cyclic node $\bar{x}$ located in $\mathcal{C}_1$ and a cyclic node $\bar{y}$ located in $\mathcal{C}_2$.
   (c) Compute the set of correctable distance bias $\mathcal{D} = \{i \cdot \Delta L \mod L_1 \mid i = 0, 1, \ldots, \#C\}$ as in Step 2 of Attack 3.
2. Run Alg. 5 with a parameter t to develop 2^t nodes in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$), compute their distance from the target node $\bar{x}$ (resp. $\bar{y}$). Store these nodes in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$) in a data structure $\mathcal{G}_1$ (resp. $\mathcal{G}_2$).
3. Build a 2^r-Joux's multi-collision $\mathcal{M}_{\mathtt{MC1}}$ (resp. $\mathcal{M}_{\mathtt{MC2}}$) starting from the cyclic node $\bar{x}$ (resp. $\bar{y}$) and denote its final endpoint by $\hat{x}$ (resp. $\hat{y}$).
4. Construct a simultaneous expandable message $\mathcal{M}_{\mathtt{SEM}}$ across the two hash functions that starts from the state $\hat{x}$ in the first pass, and denote its final endpoint by $\tilde{y}$ in the second pass. The details of constructing such an expandable message are provided in Sect. 7.3.
5. Find a single-block $\bar{m}$ mapping $\hat{y}$ to some internal state b_p in the second pass of the original message $\boldsymbol{M}$. The search procedure is trivial and hence omitted. Then, use $\bar{m}$ updating the corresponding chaining state a_p in the first pass to a state $\tilde{x}$, *i.e.*, compute $\tilde{x} = h_1(a_p, \bar{m})$.
6. For each message M_2' in $\mathcal{M}_{\mathtt{MC2}}$,
   (a) Compute $\check{x} = h_1^*(\tilde{x}, M_2')$.
   (b) Compute a chain $\vec{x}$ by applying f_1 to update $\check{x}$ iteratively until up to a maximum length 2^{n-t} or until it hits $\mathcal{G}_1$. In the latter case, compute the distance d_1 of $\check{x}$ to $\bar{x}$, and store (d_1, M_2') in a table $\mathcal{T}_1$.
7. For each message M_1' in $\mathcal{M}_{\mathtt{MC1}}$,

(a) Compute $\breve{y} = h_2^*(\tilde{y}, M_1')$;
(b) Compute a chain $\vec{y}$ by applying f_2 to update $\breve{y}$ up to a maximum length 2^{n-t} or until it hits $\mathcal{G}_2$. In the latter case, compute the distance d_2 of $\breve{y}$ to $\bar{y}$, and store (d_2, M_1') in a table $\mathcal{T}_2$.

8. Find (d_1, M_2') in $\mathcal{T}_1$ and (d_2, M_1') in $\mathcal{T}_2$ such that $(d_1 - d_2 \mod L_1) \in \mathcal{D}$. The search is a meet-in-the-middle procedure to match elements between $\mathcal{T}_1$ and $\mathcal{T}_2$. Denote the common distance corrected by values in $\mathcal{D}$ by $d \triangleq d_1 + i \cdot L_1 = d_2 + j \cdot L_2$ for some i and j, and the corresponding messages $M_1 \triangleq M_1'$ and $M_2 \triangleq M_2'$ (retrieved from $\mathcal{T}_1$ and $\mathcal{T}_2$).
9. Derive a message $M_{\|q}$ with a block length $q = L' - p - 1 - r - d - r$ from $\mathcal{M}_{\texttt{SEM}}$, where L' is the length of the constructed second-preimage. Construct a message $\boldsymbol{M'} = m_1 \,\|\, m_2 \,\|\, \cdots \,\|\, m_p \,\|\, \bar{m} \,\|\, M_2 \,\|\, [m]^d \,\|\, M_1 \,\|\, M_{\|q}$ and output $\boldsymbol{M'}$ as a second-preimage.

Complexity Analysis. The complexities of each step in this attack are as follows (ignoring constant factors and the factor n):

Step 1:	$2^{n/2}$	**Step 2:**	2^t	**Step 3:**	$2^{n/2}$
Step 4:	$2^{\ell'} + 2^{n/2+2\log_2(n)+1}$	**Step 5:**	$2^{n-\ell}$	**Step 6:**	$2^r \cdot 2^{n-t}$
Step 7:	$2^r \cdot 2^{n-t}$	**Step 8:**	2^r	**Step 9:**	$2^{\ell'}$

The sum of dominant terms is

$$2^t + 2^{\ell'} + 2^{n-\ell} + 2^r \cdot 2^{n-t},$$

where 2^t is the complexity for developing more nodes in the look-ahead procedure; $2^{\ell'}$ is the complexity for building the simultaneous expandable message; $2^r \cdot 2^{n-t}$ is the complexity for generating tables $\mathcal{T}_1$ and $\mathcal{T}_2$.

We first balance the first term with the fourth term by setting $t = r + n - t$, which gives $t = n/2 + r/2$. As a result, the sum of dominant terms is

$$2^{n/2+r/2} + 2^{\ell'} + 2^{n-\ell}.$$

- When the allowed length L' of the second preimage is limited by $2^{n/2}$, we set $\ell' = n/2$ to optimize the complexity. The multi-cycle technique is not applicable. The required number of samplings on pairs of starting nodes before finding one pair reaching any one of the 2^{2r} pairs of target nodes at a common distance is $2^{2n-3n/2}$. Thus, it is required that $2^{2r} = 2^{2n-3n/2} = 2^{n/2}$. This gives $r = n/4$. The total complexity is then $2^{5n/8}$ for all $\ell \geq 3n/8$.
- When the allowed length L' of the second preimage is not limited and can be greater than $2^{n/2}$, multi-cycles can be used. In this case, the required number of samplings on pairs of starting nodes before finding one pair reaching any one of the 2^{2r} pairs of target nodes at a common distance is $2^{2n-3n/2-(\ell'-n/2)}$. Thus, $2^{2r} = 2^{n-\ell'}$. This gives $r = n/2 - \ell'/2$. The total complexity is then $2^{3n/4-\ell'/4} + 2^{\ell'} + 2^{n-\ell}$. We then balance the first two terms by setting $3n/4 - \ell'/4 = \ell'$, which gives $\ell' = 3n/5$.

- For the case $\ell < 2n/5$, the total complexity is $2^{n-\ell}$;
- For the case $2n/5 \leq \ell$, the total complexity is stabilized at $2^{3n/5}$.

(see Fig. 13 for trade-off curves).

7.3 Step 4: Constructing an Expandable Message

The constructing method is similar to that in Sect. 2.6, with slight modifications. Detailed steps and the method used are shown in Alg. 6, where $C \approx n/2+\log(n)$:

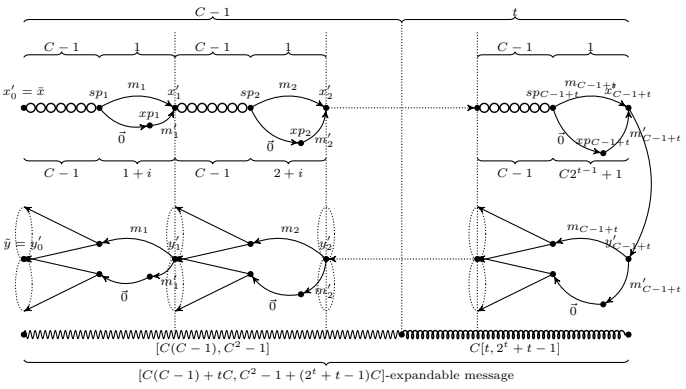
Algorithm 6: Building and using a simultaneous expandable message applicable to attacks on the Zipper hash

1. $x'_0 \leftarrow \hat{x}$
2. For $i \leftarrow 1,\ 2, \cdots, C-1+t$:
   (a) Build a 2^{C-1} standard Joux's multi-collision in h_1 starting from x'_{i-1}, and denote its final endpoint by sp_i.
   (b) Compute $xp_i = h_1^*(sp_i, [0]^s)$, where $[0]^s$ is an all-zero message of size s blocks, where $s = i$ if $i \leq C-1$ and $s = C2^{i-(C-1)-1}$ if $C-1 < i \leq C-1+t$.
   (c) Find a collision $h_1(sp_i, m_i) = h_1(xp_i, m'_i)$ with single block messages $m_i,\ m'_i$. Denote the collision by x'_i.
   (d) We obtain a multi-collision in h_1 with 2^C messages that map x'_{i-1} to x'_i.
      i. Out of these messages, 2^{C-1} are of length b (obtained by combining one of the 2^{C-1} Joux's multi-collisions with m_i). We denote this set of messages by $\mathcal{M}_{\texttt{short}i}$, where $b = C$.
      ii. Out of these messages, 2^{C-1} are of length b (obtained by combining one of the 2^{C-1} Joux's multi-collisions with $[0]^s \parallel m'_i$) and we denote this set of messages by $\mathcal{M}_{\texttt{long}_i}$, where $b = C+i$ if $i \leq C-1$ and $b = C(2^{i-(C-1)-1}+1)$ if $C-1 < i \leq C-1+t$.
3. Denote the last collision state x'_{C-1+t} by $\ddot{x}$, and compute $\ddot{y} = h_2(h_1(\ddot{x}, m'_{L'}), m'_{L'})$, where $m'_{L'}$ is a message block padded with the length L' of the second preimage.
4. $y'_{C-1+t} \leftarrow \ddot{y}$, $\mathcal{M}_{\texttt{SEMshort}} \leftarrow \emptyset$, $\mathcal{M}_{\texttt{SEMlong}} \leftarrow \emptyset$.
5. For $i \leftarrow C-1+t,\ C-1+t-1,\ \ldots,\ 2,\ 1$:
   (a) For each $\vec{ms}_i \in \mathcal{M}_{\texttt{short}i}$, compute $u_i = h_2^*(y'_i, \overleftarrow{\vec{ms}_i})$ where $\vec{ms}_i = ms_{i,1} \parallel ms_{i,2} \parallel \cdots \parallel ms_{i,C-1} \parallel ms_{i,C}$ and $\overleftarrow{\vec{ms}_i} = ms_{i,C} \parallel ms_{i,C-1} \parallel \cdots \parallel ms_{i,1}$. Store each pair $(u_i,\ \vec{ms}_i)$ in a table $\mathcal{U}_i$ indexed by u_i. The final size of $\mathcal{U}_i$ is 2^{C-1}.
   (b) For each $\vec{ml}_i \in \mathcal{M}_{\texttt{long}_i}$, compute $v_i = h_2^*(y'_i, \overleftarrow{\vec{ml}_i})$ where $\vec{ml}_i = ml_{i,1} \parallel ml_{i,2} \parallel \cdots \parallel ml_{i,s-1} \parallel ml_{i,s}$ and $\overleftarrow{\vec{ml}_i} = ml_{i,s} \parallel ml_{i,s-1} \parallel \cdots \parallel ml_{i,1}$, where $s = C(2^{i-(C-1)-1}+1)$ if $C-1 < i \leq C-1+t$ and $s = C+i$ if $1 \leq i \leq C-1$. Store each pair $(v_i,\ \vec{ml}_i)$ in a table $\mathcal{V}_i$ indexed by v_i. The final size of $\mathcal{V}_i$ is 2^{C-1}.

(c) Find a match $u_i = v_i$ between $\mathcal{U}_i$ and $\mathcal{V}_i$, and denote the matched state by $y'_{i-1} \triangleq u_i = v_i$. Combine the corresponding message fragment $\vec{ms}_i$ indexed by y'_i with $\mathcal{M}_{\texttt{SEMshort}}$ and $\vec{ml}_i$ indexed by y'_i with $\mathcal{M}_{\texttt{SEMlong}}$, *i.e.*, $\mathcal{M}_{\texttt{SEMshort}} = \vec{ms}_i \parallel \mathcal{M}_{\texttt{SEMshort}}$ and $\mathcal{M}_{\texttt{SEMlong}} = \vec{ml}_i \parallel \mathcal{M}_{\texttt{SEMlong}}$.

Then, the whole simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ can be fully defined by $\mathcal{M}_{\texttt{SEMshort}}$ and $\mathcal{M}_{\texttt{SEMlong}}$. For any length κ lying in the appropriate range of $[C(C-1)+tC,\ C^2-1+C(2^t+t-1)]$, one can construct a message $M_{\|\kappa}$ mapping $\hat{x}$ to $\tilde{y} = y'_0$ through h_1 and h_2 by picking messages fragment either from $\mathcal{M}_{\texttt{SEMshort}}$ or from $\mathcal{M}_{\texttt{SEMlong}}$ as also described in Sect.2.6:

1. Select the length $\kappa' \in [C(C-1), C^2-1]$ such that $\kappa' = \kappa \bmod C$, defining the first $C-1$ message fragment choices: selecting the message fragment $\vec{ms}_i$ in $\mathcal{M}_{\texttt{SEMshort}}$ for $1 \le i \le C-1$ and $i \ne \kappa' - C$; selecting the message fragment $\vec{ml}_i$ in $\mathcal{M}_{\texttt{SEMlong}}$ for $i = \kappa' - C$.
2. Compute $kp \leftarrow (\kappa - \kappa')/C$, 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 $kp - t$.



8 Second-Preimage Attack on Hash-Twice

In this section, we present an efficient second-preimage attack on another cascade hash construction – Hash-Twice (a generalized specification $\mathcal{HT}(M) \triangleq \mathcal{H}_2(\mathcal{H}_1(IV, M), M)$). Similar to the previous second-preimage attack on Hash-Twice in [ABDK09], this attack builds a diamond structure for one hash function by exploiting messages in a long multi-collision built for the other hash function. Like all our previous functional-graph-based (deep-iterates-based and multi-cycles-based) attacks, it improves the attack from [ABDK09] because of the efficiency brought by exploiting the special nodes in the functional graphs. It follows the same structure as the second-preimage attack on the concatenation

combiner, but the result shows that attacking Hash-Twice can be much more efficient than attacking the concatenation combiner. That is mainly because the attack tries to connect to an n-bit internal state in the case of Hash-Twice, instead of a $2n$-bit internal state in the case of the concatenation combiner. Note that similar to all previous functional-graph-based attacks, this attack applies when the underlying hash functions use the MD construction.

8.1 Overview of the Attack

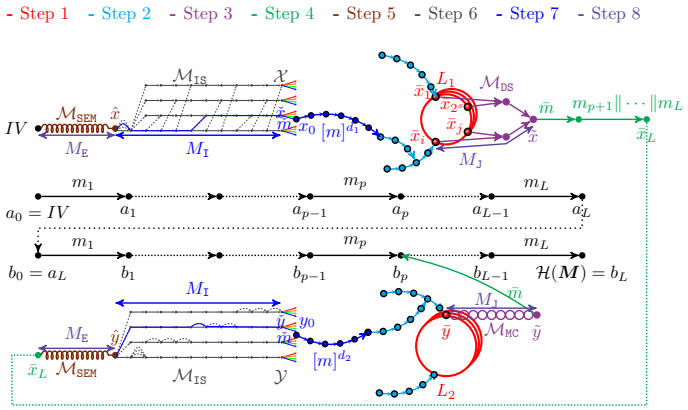
Given a challenge message $\boldsymbol{M} = m_1 \parallel m_2 \parallel \cdots \parallel m_L$, the goal of the second-preimage attack on Hash-Twice is to find another message $\boldsymbol{M}'$ such that $\mathcal{H}_2(\mathcal{H}_1(IV, \boldsymbol{M}'), \boldsymbol{M}') = \mathcal{H}_2(\mathcal{H}_1(IV, \boldsymbol{M}), \boldsymbol{M})$. The framework of this attack is sketched as follows.

Attack 6: Second-preimage attack on Hash-Twice

- **Phase 1:** Generate a set of pairs of nodes $\mathcal{S} = \{(\bar{x}_1, \bar{y}), (\bar{x}_2, \bar{y}), \ldots, (\bar{x}_{2^s}, \bar{y})\}$, where $\bar{x}_i$'s are cyclic nodes randomly located in the largest cycle of $\mathcal{FG}_{f_1}$, and $\bar{y}$ is a cyclic node in the largest cycle of $\mathcal{FG}_{f_2}$.
  In addition, like in Step 2 of Attack 3, compute the set of correctable distance bias $\mathcal{D} = \{j \cdot \Delta L \mod L_1 \mid j = 0, 1, \ldots, \#C\}$ with parameters L_1, L_2 and $\#C$ (where $\Delta L = L_2 \mod L_1$, L_1 and L_2 are the two cycle lengths, $\#C$ is the number of cycles to the maximum).
- **Phase 2:** Build a long Joux's multi-collision $\mathcal{M}_{\texttt{MC}}$ starting from $\bar{y}$ and denote its endpoint by $\tilde{y}$. Then, by exploiting messages in $\mathcal{M}_{\texttt{MC}}$, build a diamond structure $\mathcal{M}_{\texttt{DS}}$ with the cyclic nodes $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{2^s}$ as leaves and denote its root by $\tilde{x}$. Find a message block $\bar{m}$ mapping the endpoint $\tilde{y}$ of Joux's multi-collision to one of the chaining states b_p in the second pass of the original message. After finding $\bar{m}$, starting from the root $\tilde{x}$ of the diamond $\mathcal{M}_{\texttt{DS}}$, compute the final state $\bar{x}_L$ in the first pass with message fragment $\bar{m} \parallel m_{p+1} \parallel \cdots \parallel m_L$.
- **Phase 3:** Build a simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ (with lengths covering roughly $[n^2, L]$) starting from $(IV, \bar{x}_L)$. Denote its endpoints by $(\hat{x}, \hat{y})$.
- **Phase 4:** Find a message fragment $M_{\texttt{Link}}$ such that $(\hat{x}, \hat{y}) \xrightarrow{M_{\texttt{Link}}} (\bar{x}, \bar{y})$ for some $(\bar{x}, \bar{y}) \in \mathcal{S}$.
  This is done by first build a 2^r-interchange structure $\mathcal{M}_{\texttt{IS}}$ starting from $(\hat{x}, \hat{y})$. Denote its two sets of endpoints by $\mathcal{X}$ and $\mathcal{Y}$. Then, for states in $\mathcal{X}$ and $\mathcal{Y}$, launch a meet-in-the-middle procedure to find a pair of nodes (x_0, y_0) with $x_0 = h_1(\check{x}, \check{m})$ and $y_0 = h_1(\check{y}, \check{m})$ and $\check{x} \in \mathcal{X}$ and $\check{y} \in \mathcal{Y}$, such that in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, the distance difference of x_0 and y_0 from some targeted nodes $(\bar{x}, \bar{y}) \in \mathcal{S}$ is correctable by the values in $\mathcal{D}$. Denote the common distance after corrected as d. Retrieve the message fragment $M_{\texttt{I}}$ from $\mathcal{M}_{\texttt{IS}}$ such that $(\hat{x}, \hat{y}) \xrightarrow{M_{\texttt{I}}} (\check{x}, \check{y})$. The desired $M_{\texttt{Link}}$ is then defined as $M_{\texttt{Link}} \triangleq M_{\texttt{I}} \parallel \check{m} \parallel [m]^d$.

At the end, select a message fragment $M_{\mathtt{E}} \in \mathcal{M}_{\mathtt{SEM}}$ with a proper block length, and a message fragment $M_{\mathtt{J}} \in \mathcal{M}_{\mathtt{DS}}$ such that $\bar{x} \xrightarrow{M_{\mathtt{J}}} \tilde{x}$, and construct a second preimage:

$$\begin{aligned} \boldsymbol{M}' &\triangleq M_{\mathtt{E}} \parallel M_{\mathtt{Link}} \parallel M_{\mathtt{J}} \parallel \bar{m} \parallel m_{p+1} \parallel \cdots \parallel m_L \\ &= M_{\mathtt{E}} \parallel M_{\mathtt{I}} \parallel \check{m} \parallel [m]^d \parallel M_{\mathtt{J}} \parallel \bar{m} \parallel m_{p+1} \parallel \cdots \parallel m_L. \end{aligned}$$



8.2 Details of the Second-Preimage Attack on Hash-Twice Based on Multi-Cycles, Diamond and Interchange Structure for Long Messages

The detailed steps of the second-preimage attack on Hash-Twice are follows:

Detailed Steps of Attack 6

1. Fix an arbitrary single-block message m and construct $f_1(\cdot) \triangleq h_1(\cdot, m)$ and $f_2(\cdot) \triangleq h_2(\cdot, m)$.
   (a) Run the cycle search algorithm several times to locate the largest cycles in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$, and obtain the cycle lengths L_1 and L_2. Without loss of generality, assume $L_1 \leq L_2$.
   (b) Generate a set of 2^s pairs of target nodes

$$\mathcal{S} = \{(\bar{x}_1, \bar{y}), (\bar{x}_2, \bar{y}), \ldots, (\bar{x}_{2^s}, \bar{y})\},$$

   where $\bar{x}_i$ are cyclic nodes randomly located in the largest cycle of $\mathcal{FG}_{f_1}$ and $\bar{y}$ is a cyclic node in the largest cycle of $\mathcal{FG}_{f_2}$.
   (c) Compute the set of correctable distance bias

$$\mathcal{D} = \{j \cdot \Delta L \mod L_1 \mid j = 0, 1, \ldots, \#C\}$$

with parameters L_1, L_2 and $\#C$ (where $\#C$ is the allowed maximum number of cycles and $\Delta L = L_2 \mod L_1$), as in Step 2 of Attack 3.

2. Develop 2^t nodes in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$) by running Alg. 5, and record these together with their distances from a particular target node $\bar{x}_i$ (resp. $\bar{y}$) in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$) in a table $\mathcal{T}_x$ (resp. $\mathcal{T}_y$). Note that the distance of a node from all other target nodes $\bar{x}_j$ can be directly deduced from the distance of it from the particular target node $\bar{x}_i$, as explained in Step 4c of Attack 3.
3. Build a long Joux's multi-collision $\mathcal{M}_{\texttt{MC}}$ of length $s \cdot n/2$ (blocks) starting from state $\bar{y}$, and denote its endpoint by $\tilde{y}$. Then, by exploiting messages in $\mathcal{M}_{\texttt{MC}}$, build a diamond structure $\mathcal{M}_{\texttt{DS}}$ starting from cyclic nodes $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{2^s}$ (as leaves) and denote its root by $\tilde{x}$.
4. Find a message block $\bar{m}$ mapping the endpoint $\tilde{y}$ of Joux's multi-collision to one of the chaining values b_p in the second pass of the original message, as done in the second-preimage attack on a single MD [KS05]. After finding $\bar{m}$, starting from the root $\tilde{x}$ of the diamond, directly compute a final state (denoted by $\bar{x}_L$) in the first pass with message fragment $\bar{m} \parallel m_{p+1} \parallel \cdots \parallel m_L$. The initial state in the second pass on the second preimage is then determined to be $\bar{x}_L$.
5. Build a parallel simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ starting from $(IV, \bar{x}_L)$, and denote their endpoints by $(\hat{x}, \hat{y})$.
6. Build a 2^r-interchange structure $\mathcal{M}_{\texttt{IS}}$ starting from $(\hat{x}, \hat{y})$, and denote its two sets of endpoints by $\mathcal{X}$ and $\mathcal{Y}$.
7. Find a message fragment $M_{\texttt{Link}}$ such that $(\hat{x}, \hat{y}) \xrightarrow{M_{\texttt{Link}}} (\bar{x}, \bar{y})$ for some $(\bar{x}, \bar{y}) \in \mathcal{S}$. The search procedure is described as follows.
   (a) Select a random single-block message w.
   (b) Initialize a table $\mathcal{T}_y$ as empty.
   (c) For each of the 2^r states $\check{y}' \in \mathcal{Y}$:
      – Compute $y_0 = h_2(\check{y}', w)$
      – Derive the distance $d_{\bar{y}}$ of y_0 from the target node $\bar{y}$ in $\mathcal{FG}_{f_2}$ (if it is undefined $\perp$, go to Step 7a), store $(\check{y}', d_{\bar{y}})$ in $\mathcal{T}_y$.
   (d) For each of the 2^r states $\check{x}' \in \mathcal{X}$:
      – Compute $x_0 = h_1(\check{x}', w)$
      – For each of the 2^s target node $\bar{x}_i \in \{\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_{2^s}\}$
         • Derive the distance of x_0 from $\bar{x}_i$ (as described in Step 4c of Attack 3), denote it by $d_{\bar{x}_i}$.
         • Make a match between $d_{\bar{x}_i}$ and elements in $\mathcal{T}_y$ by checking whether there exists a $d_{\bar{y}} \in \mathcal{T}_y$ such that $(d_{\bar{x}_i} - d_{\bar{y}} \mod L_1) \in \mathcal{D}$. If one exists, retrieve $\check{y}'$ corresponding to $d_{\bar{y}}$ from $\mathcal{T}_y$. Derive the common distance $d \triangleq d_{\bar{x}_i} + j \cdot L_1 = d_{\bar{y}} + k \cdot L_2$. Set $\check{m} \triangleq w$, $\check{x} \triangleq \check{x}'$, $\check{y} \triangleq \check{y}'$, and $\bar{x} \triangleq \bar{x}_i$. Retrieve the message fragment $M_{\texttt{I}}$ from $\mathcal{M}_{\texttt{IS}}$ such that $(\hat{x}, \hat{y}) \xrightarrow{M_{\texttt{I}}} (\check{x}, \check{y})$. Set $M_{\texttt{Link}} \triangleq M_{\texttt{I}} \parallel \check{m} \parallel [m]^d$, and go to the next step.

8. Select a message fragment $M_{\mathtt{E}} \in \mathcal{M}_{\mathtt{SEM}}$ with a proper length $q = p - (2^{t+1} + (2^t - 1)^2 \cdot n/2) - 1 - d - s \cdot n/2 - 1 = p - 2^{t+1} - (2^t - 1)^2 \cdot n/2 - d - s \cdot n/2 - 2$, select the message fragment $M_{\mathtt{J}} \in \mathcal{M}_{\mathtt{DS}}$ such that $\bar{x} \xrightarrow{M_{\mathtt{J}}} \tilde{x}$, and construct a second preimage:

$$\begin{aligned} \boldsymbol{M'} &\triangleq M_{\mathtt{E}} \parallel M_{\mathtt{Link}} \parallel M_{\mathtt{J}} \parallel \bar{m} \parallel m_{p+1} \parallel \cdots \parallel m_L \\ &= M_{\mathtt{E}} \parallel M_{\mathtt{I}} \parallel \breve{m} \parallel [m]^d \parallel M_{\mathtt{J}} \parallel \bar{m} \parallel m_{p+1} \parallel \cdots \parallel m_L. \end{aligned}$$

Complexity Analysis. The complexity of each step in this second-preimage attack on Hash-Twice is as follows (denote $L = 2^\ell$):

Step 1: $2^{n/2}$ **Step 2:** 2^t **Step 3:** $n\sqrt{s} \cdot 2^{n/2+s/2}$

Step 4: $2^{n-\ell} + 2^\ell$ **Step 5:** $2^\ell + n^2 \cdot 2^{n/2}$ **Step 6:** $2^{n/2+2r}$

Step 7: $2^r \cdot 2^{n-t} \cdot 2^{n-2r-s-\ell}$ **Step 8:** 2^ℓ

For $\ell > n/2$, we have $2^{n-\ell} < 2^{n/2}$. The sum of dominant terms is (ignoring polynomial factors)

$$2^t + 2^{n/2+s/2} + 2^\ell + 2^{n/2+2r} + 2^{2n-t-r-s-\ell}.$$

We balance different terms by setting $t = n/2 + 2r = n/2 + s/2 = 2n - t - r - s - \ell$, *i.e.*, $r = t/2 - n/4$, $s = 2t - n$ and $t = 13n/18 - 2\ell/9$. Consequently, the total complexity is approximately

$$2^\ell + 2^{13n/18-2\ell/9}.$$

The improved attack is valid for all $\ell > n/2$ (even when we account for the message length 2^{2r} of the interchange structure, which should be less the 2^ℓ). The optimal complexity for this attack is $2^{13n/22}$, obtained when $2^\ell = 2^{13n/22}$. Compared with the original optimal complexity $2^{2n/3}$ for messages of length $2^{13n/22}$, the improvement is $2^{5n/66}$ (see Fig. 13 for a trade-off curve).

8.3 Details of the Second-Preimage Attack on Hash-Twice Based on Deep-iterates, Diamond and Interchange Structure for Short Messages

Note that for $L = 2^\ell \leq 2^{n/2}$, we can no longer apply the multi-cycles technique. However, we can still choose deep-iterates with depth less than $2^{n/2}$ as target nodes and selected proper iterate depth to improve the second-preimage attack on Hash-Twice for short messages.

The procedure of this new attack is similar to that of the previous attack in Sect.8.2. The difference lies in that in Step 1, we collect a set of 2^s pairs of target nodes $\{(\bar{x}_1, \bar{y}), (\bar{x}_2, \bar{y}), \ldots, (\bar{x}_{2^s}, \bar{y})\}$, where $\bar{x}_i$'s are 2^{n-g}-th iterate nodes in $\mathcal{FG}_{f_1}$ for $1 \leq i \leq 2^s$ and $\bar{y}$ is a 2^{n-g}-th iterate node in $\mathcal{FG}_{f_2}$. It is required

that $g \geq \max(n/2, n-\ell)$. In addition, in Steps 2 and 7, we use the distinguish-point technique in the look-ahead procedure as we did in the preimage attack on XOR combiner in Sect.4.2. The procedures in other steps are the same as those in Sect.8.2.

In this case, required number of samplings before finding a pair of $(\check{x}, \check{y})$ such that they reach one of a target pair of nodes $(\bar{x}_i, \bar{y})$ at a common distance changes to be $2^{3g-n-s-2r}$. Consequently, the complexity of Step 7 is $2^r \cdot 2^{3g-n-s-2r} \cdot (2^{n-t} + 2^{\ell+t-n} + 2^{\ell+s-g})$. Thus, complexity of each step in this second-preimage attack on Hash-Twice for short message is as follows:

Step 1: 2^g **Step 2:** 2^t **Step 3:** $n\sqrt{s} \cdot 2^{n/2+s/2}$

Step 4: $2^{n-\ell} + 2^\ell$ **Step 5:** $2^\ell + n^2 \cdot 2^{n/2}$ **Step 6:** $2^{n/2+2r}$

Step 7: $2^{3g-t-s-r} + 2^{3g+t-s-r+\ell-2n} + 2^{2g+\ell-n-r}$ **Step 8:** 2^ℓ

For $\ell \leq n/2$, we have $2^{n-\ell} \geq 2^{n/2}$. Thus, the sum of dominant terms is

$$2^g + 2^t + 2^{n/2+s/2} + 2^{n-\ell} + 2^{n/2+2r} + 2^{3g-s-t-r} + 2^{3g+t-s-r+\ell-2n} + 2^{2g+\ell-n-r}.$$

We first set $t = n/2+s/2 = n/2+2r = 3g-s-t-r$, *i.e.*, $t = 2g/3+5n/18$, $s = 4g/3-4n/9$, and $r = g/3-n/9$ to make a balance. The sum of dominant terms is $2^g + 2^{2g/3+5n/18} + 2^{n-\ell} + 2^{2g+\ell-7n/6} + 2^{5g/3+\ell-8n/9}$. We optimize the attack by picking the minimal value of g under the restriction $g \geq \max(n/2, n-\ell)$, *i.e.*, $g = n-\ell$ for $\ell \leq n/2$. Consequently, the total complexity becomes (for $g = n-\ell$ and $\ell < n/2$, with the last two terms always less than $2^{n-\ell}$, the distinguished points method allowed us to resolve with no overhead the complication of keeping track of distances from 2^s target nodes):

$$2^{17n/18-2\ell/3} + 2^{n-\ell}.$$

The improved attack is better than that in [ABDK09] for $\ell > 5n/12$ (in which case $17n/18 - 2\ell/3 < 2n/3$, the message length 2^{2r} of the interchange structure is less the 2^ℓ, thus applicable). The optimal complexity for this attack is $2^{11n/18}$, obtained when $\ell = n/2$. Compared with the previous best-known complexity $2^{2n/3}$ at message length $L = 2^{n/2}$, the improvement is $2^{n/18}$ (see Fig. 13 for a trade-off curve).

9 More Applications and Extensions

9.1 Applications Beyond MD Construction and Beyond XOR Operation

Application to HAIFA mode. The first preimage attack on the XOR combiner purely bases on interchange structure. Thus, it works identically if the hash functions use the HAIFA mode rather than the plain Merkle-Damgård iteration, whereas the other attacks all based on functional graphs requiring identical compression functions and thus cannot work if the underlying hash functions use the HAIFA mode.

Application to Cryptophia's short combiner. All of our attacks on XOR combiner can also be applied to Cryptophia's short combiner, as proposed by Mittelbach [Mit13], and to the revised version of Mennink and Preneel [MP14]. This combiner computes the sum of two hash functions with some pre-processing of the message to allow non-independent functions:

$$\begin{aligned} C(M) &= \mathcal{H}_1\left(\tilde{m}_1^1 \parallel \ldots \parallel \tilde{m}_\ell^1\right) \oplus \mathcal{H}_2\left(\tilde{m}_1^2 \parallel \ldots \parallel \tilde{m}_\ell^2\right) \\ \tilde{m}_j^1 &= \mathcal{H}_1(0 \parallel l_1 \parallel m_j \oplus k_1) \oplus \mathcal{H}_2(0 \parallel l_2 \parallel m_j \oplus k_2) \\ \tilde{m}_j^2 &= \mathcal{H}_1(1 \parallel l_1 \parallel m_j \oplus k_1) \oplus \mathcal{H}_2(1 \parallel l_2 \parallel m_j \oplus k_2) \end{aligned}$$

where k_1, k_2, l_1, l_2 is a randomly chosen key. The security proof in the ideal model shows that C is optimally preimage resistant if at least one of the hash functions is ideal.

However, if both $\mathcal{H}_1$ and $\mathcal{H}_2$ are narrow-pipe, we can apply our preimage attacks with the same corresponding complexity. This does not violate the security proof because we need both functions to be narrow-pipe and hence not n-bit ideal[25]. From a practical point of view, though, they show that in many cases (*e.g.* using SHA-512 and Whirlpool) the combiner is *weaker* than the initial functions.

Application beyond XOR. All preimage attacks on XOR combiner can easily be extended to $\mathcal{H}_1(M) \odot \mathcal{H}_2(M)$ where $\odot$ denotes an easy to invert group operation (for instance, a modular addition rather than the exclusive or). These attacks can also be extended to hash functions $\mathcal{H}_1$ and/or $\mathcal{H}_2$ using an internal checksum, such as the GOST family of hash functions, using pairs of blocks with a constant sum.

9.2 Applications to the Combination of Wide-Pipe Hash Functions

Our attacks can also be used when the internal state size n' is (not much) larger than the output size n.

The interchange-structure-based preimage attack on the XOR combiner. In Attack 1, the complexity of building a 2^t-interchange structure is related to n' as $(n'/2) \cdot 2^{2t+n'/2}$. On the other hand, the complexity of the meet-in-the-middle preimage search is related to n as 2^{n-t}. The optimal complexity is $(n'/2) \cdot 2^{2n/3+n'/6}$ by matching the two complexities with $t = n/3 - n'/6$. Therefore, our attack can be applied as long as $n' + 6\log(n') \leq 2n$ holds. For instance, we can compute preimages of SHA-224 $\oplus$ BLAKE-224 using the interchange-structure-based attack with complexity roughly 2^{199}.[26]

[25] A large multi-collisions can be built with a cost of roughly $2^{n/2}$ in a narrow-pipe function, but this costs almost 2^n for an ideal hash function.

[26] However, the message length can be a problem with some hash functions that do not accept long inputs. For example, SHA-256 and SHA-224 are only defined for

The deep-iterates-based preimage attack on the XOR combiner. For parameters $g_1 \geq \max(n'/2,$
$n' - \ell)$ and $s \geq 0$, the complexity of each phase in Attack 2 is as follows:

Phase 1: $2^{\ell} + {n'}^2 \cdot 2^{n'/2}$ **Phase 2:** 2^{n+s-g_1}

Phase 3: $2^{3g_1/2-s/2} + 2^{\ell+9g_1/2-2n'-3s/2} + 2^{\ell+2g_1-n'}$

We balance the time complexities of different phases similar to what we did before. In case of $\ell \leq n'/2$, we set $g_1 = n' - \ell$, the total time complexity of the attack is $2^{n/3+2n'/3-2\ell/3} + 2^{n'-\ell}$. The optimal complexity is $2^{n/3+n'/3}$,[27] obtained for $\ell = n'/2$. It is less than 2^n when $n' < 2n$.

The multi-cycles-based preimage attack on the XOR combiner. The complexity of each detailed step in Attack 3 is no longer related to the output size n but related to the internal state size n' except for Step 3. For Step 3, the complexity is $2^{s+n-n'/2}$. For other steps, the complexity can be obtained by simply replacing n with n' from the original formula. Then, the overall complexity is approximately (note that $\ell \geq n'/2$)

$$2^{\ell} + 2^{s+n-n'/2} + 2^{t} + 2^{2n'-t-s-\ell}.$$

We balance the last three terms by setting $s + n - n'/2 = t = 2n' - t - s - \ell$, *i.e.*, $s = n' - 2n/3 - \ell/3$, $t = n'/2 + n/3 - \ell/3$. Then, the total complexity is $2^{\ell} + 2^{n'/2+n/3-\ell/3}$. The optimal complexity is $2^{3n'/8+n/4}$ for $\ell = 3n'/8 + n/4$. The complexity is less than 2^n when $n'/2 < \ell < n$ and $n' < 2n$.

The deep-iterates-based second-preimage attack on the concatenation combiner. The complexity of Attack 4 is no longer related to the output size n but rather is related to the internal state size n' and the message length $L = 2^{\ell}$. The time complexity is $2^{(3/5)\cdot(2n'-\ell)}$ as long as $\ell \leq 3n'/4$ and is less than 2^n for $\ell > 2n' - 5n/3$. Therefore, this attack can be applied when $n' < 4n/3$.

The second-preimage attack on the Zipper hash. The complexity of Attack 5 is no longer related to the output size n but is related to the internal state size n'. We can get the complexity by simply replacing n with n' in the formula. Accordingly, when the length L' of the second preimage is limited by $2^{n'/2}$, the optimal complexity is $2^{5n'/8}$ for all $\ell \geq 3n'/8$. It is less than 2^n when $n' < 8n/5$. When the length L' is not limited, the optimal complexity is $2^{3n'/5}$ for all $\ell \geq 2n'/5$. It is less than 2^n when $n' < 5n/3$.

---

messages with less than 2^{64} bits (*i.e.*, 2^{55} blocks). In this case, one can apply the attack with a smaller value of t: this reduces the length of the messages at the cost of more time spent in the preimage search step. Thus, to mount a preimage attack against SHA-224 $\oplus$ BLAKE-224, we should use $t = 24$ instead of $t = 32$. Then, the optimal complexity is 2^{200} instead of 2^{199}.

[27] Note that $n/3 + n'/3 > n'/2$ when $n' < 2n$.

The second-preimage attack on Hash-Twice. The complexity of Attack 6 is no longer related to the output size n but rather is related to the internal state size n'. Accordingly, when the length $L = 2^\ell$ is limited by $2^{n'/2}$, the total complexity is $2^{n'-\ell} + 2^{17n'/18-2\ell/3}$. The optimal complexity is $2^{11n'/18}$, obtained for $\ell = n'/2$, which is less than 2^n when $n' < 18n/11$. When the length is not limited, the total complexity is $2^\ell + 2^{13n'/18-2\ell/9}$. The optimal complexity is $2^{13n'/22}$, obtained for $\ell = 13n'/22$, which is less than 2^n when $n' < 22n/13$.

9.3 Extensions to the combination of three or more hash functions

The interchange-structure-based preimage attack on the XOR combiner. The interchange-structure-based attack on the XOR combiner, *i.e.*, Attack 1, can be extended to the sum of three or more hash functions. To attack the sum of k functions, two different strategies are possible: either we use a simpler structure that only gives two degrees of freedom and fixes $k-2$ functions to a constant value, or we build an interchange structure to control all the k functions independently.

Controlling only two functions. The easiest way to extend the attack is to use a single chain in the $k-2$ extra hash functions. The procedure to build a switch is modified in order to use multi-collisions for $k-1$ functions instead a simple multi-collisions for one function; this costs $O(n^{k-1} \cdot 2^{n/2})$ using Joux's method [Jou04]. As in the basic attack, we need $O(t^2)$ switches to generate a 2^t-interchange for two functions, and the preimage search costs $O(2^{n-t})$; the optimal complexity is therefore $O(n^{k-1} \cdot 2^{5n/6})$ with $t = n/6$.

Controlling all the functions. Alternatively, we can build a more complex interchange structure in order to control all the functions independently. When attacking three functions, we will use the switch structure to jump from chains $(\vec{a}_{j_0}, \vec{b}_{k_0}, \vec{c}_{l_0})$ to $(\vec{a}_{j_0}, \vec{b}_{k_0}, \vec{c}_{l_1})$ (or $(\vec{a}_{j_0}, \vec{b}_{k_1}, \vec{c}_{l_0})$ or $(\vec{a}_{j_1}, \vec{b}_{k_0}, \vec{c}_{l_0})$). We need $2^{3t}-1$ switches in the interchange structure to reach all the 2^{3t} triplets of chains (a switch makes only one new triplet reachable). Each switch is built using a $2^{n/2}$-multi-collision on two functions, which can be built for a cost of $O(n^2 \cdot 2^{n/2})$ following Joux's technique [Jou04]. Therefore, we can build a 2^t-interchange for a cost of $O(n^2 \cdot 2^{3t+n/2})$. More generally, for the sum of k hash functions, we can build an interchange structure for k functions for a cost of $O(n^{k-1} \cdot 2^{kt+n/2})$.

In the preimage search phase, we generate k lists of size 2^t, and we need to detect efficiently whether we can combine then to generate a zero-sum. This problem can be solved using an algorithm similar to Wagner's generalized birthday algorithm [Wag02]. If $k = 2^\kappa$, we find a solution with probability $O(2^{(\kappa+1)\cdot t-n})$ for a cost of $O(k \cdot 2^t)$. Therefore, the preimage search costs $O(k \cdot 2^{n-\kappa t})$. With $k = 4$ (*i.e.*, $\kappa = 2$), this yields a complexity of $O(n^3 \cdot 2^{5n/6})$. However, this approach is less efficient than the previous one for $k = 3$ and for $k > 4$.

To summarize, attacking the sum of k hash functions ($k \geq 2$) using interchange structure costs $O(n^{k-1} \cdot 2^{5n/6})$. Controlling chains independently in more than

two hash function might be useful for further work, but it does not improve the preimage attack on the sum of k hash functions.

The deep-iterates-based preimage attack on the XOR combiner. Suppose the combiner outputs the sum of k hash digests of n bits. To extend Attack 2, one needs to first construct a simultaneous expandable message for k independent hash functions, this costs $O(2^\ell + n^{2(k-1)} \cdot 2^{n/2})$ by setting $C \approx (n/2 + \log(n))^{k-1}$ (see Fig.9a).

To extend Phase 2, one collects 2^{g_1} deep-iterates for each of the k random functional graphs. Then, tries to find a set of 2^s tuples of k deep-iterates by mapping a single-block message to states whose sum equals the target V. Again, this can be solved using an algorithm similar to Wagner's generalized birthday algorithm, with k lists of size 2^{g_1}. Let $k = 2^\kappa$, and then finding 2^s tuples costs $O(2^s \cdot k \cdot 2^{n-\kappa g_1}) \approx 2^{n+s-\kappa g_1}$.

To extend Phase 3, one expands the k functional graphs independently with parameter g_2 in the look-ahead procedure and then tries to find a tuple of k starting nodes simultaneously hitting one of the 2^s tuples of k deep-iterates obtained in Phase 2. Note that for one tuple of k deep-iterates of depth 2^d in independent random functional graphs, the probability for a tuple of k random nodes reaching them simultaneously is approximately $2^{-kn} \sum_{i=1}^{2^d} i^k \approx 2^{(k+1)d-kn}$. In the attack, $d = n - g_1$. Thus, the complexity of Phase 3 becomes

$$\begin{aligned} &2^{g_2} + 2^{kn-(k+1)(n-g_1)-s} \cdot (2^{n-g_2} + 2^{l+g_2-n} + 2^{l+s-g_1}) \\ =&2^{g_2} + 2^{(k+1)g_1-g_2-s} + 2^{(k+1)g_1+g_2+l-2n-s} + 2^{kg_1+l-n} \end{aligned}$$

Note that in this attack, the restrictions are $\ell \leq n/2$ and $g_1 \geq \max(n/2, n-l)$. Thus, the last term in the complexity of Phase 3, *i.e.*, 2^{kg_1+l-n}, cannot be less than 2^n for $k \geq 3$. In other words, when combining more than two hashes, under any configuration for parameters of the attack, the distinguished points method no longer allowed us to resolve with no overhead the complication of keeping track of distances from the $\mathcal{S}$-nodes. Thus, this extended attack is not more efficient than 2^n.

The multi-cycles-based preimage attack on the XOR combiner. The extension of Attack 3 is very similar to the above extension of the deep-iterates-based preimage attack. However, because this attack does not need to use the distinguished points method, it can be more efficient than 2^n for $k \geq 3$. Let $k = 2^\kappa$. Following a similar analysis for the complexity of each step to the above one, we have the following:

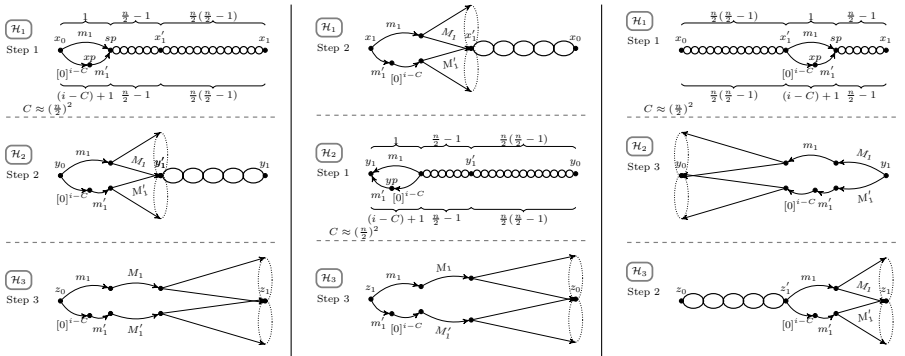
Step 1: $2^\ell + n^{2(k-1)} \cdot 2^{n/2}$	**Step 2:** $2^{n/2} + 2^{\ell-n/2}$	**Step 3:** $2^{s+n-\kappa \cdot n/2}$
Step 4: $k \cdot 2^t$	**Step 5:** $2^{n-t+kn/2-s-\ell}$	**Step 6:** 2^ℓ

We make a balance by setting $s + n - \kappa \cdot n/2 = t = n - t + kn/2 - s - \ell$, *i.e.*, $s = (k+2\kappa-2)n/6 - \ell/3$, and $t = (k-\kappa)n/6 + 2n/3 - \ell/3$. Then, the complexity

becomes $2^{\ell} + 2^{(k-\kappa)n/6+2n/3-\ell/3}$. It is optimally $2^{(k-\kappa+4)n/8}$ by setting $\ell = (k-\kappa+4)n/8$ (in this attack, we assume $n/2 < \ell < n$).

Accordingly, it costs less than 2^n computations for $k < 7$. For $k = 3$, the optimal complexity is approximately $2^{(7-\log_2 3)n/8} \approx 2^{0.677n}$. For $k = 4$, the optimal complexity is approximately $2^{3n/4} \approx 2^{0.75n}$. For $k = 5$ and $k = 6$, this extension is less efficient than the above extended interchange-structure-based attack.

The deep-iterates-based second-preimage attack on the concatenation combiner. Suppose the combiner outputs the concatenation of k hash digests of n bits. The direct way to extend Attack 4 is to simultaneously control k hash functions. To do that, again, one needs to first construct a simultaneous expandable message for k independent hash functions, this costs $O(2^{\ell} + n^{2(k-1)} \cdot 2^{n/2})$ (see Fig.9a). To extend Phase 2, one collects 2^{g_1} deep iterates for each of the k random functional graphs and then tries to find a tuple of k deep iterates hitting k internal states at the same offset p which uniformly distributed in the interval $[1, 2^{\ell}]$. The complexity of Phase 2 is $2^{g_1} \cdot 2^{-\ell} \cdot 2^{kn-kg_1} = 2^{kn-(k-1)g_1-\ell}$. To extend Phase 3, one expands the k functional graphs with parameter g_2 independently and tries to find a tuple of k starting nodes simultaneously hitting the tuple of k deep iterates obtained in Phase 2. As calculated above, for a tuple of k deep iterates of depth 2^{n-g_1} in independent random functional graphs, the probability for a tuple of k random nodes reaching them simultaneously is about $2^{(k+1)(n-g_1)-kn}$. Thus, the complexity of Phase 3 becomes $2^{(k+1)g_1/2}$ by setting $g_2 = (n-g_2) + kn - (k+1)(n-g_1)$, *i.e.*, $g_2 = (k+1)g_1/2$. After making a balance between different phases by setting $g_1 = 2(kn-\ell)/(3k-1)$, one will find that the optimal complexity is $2^{(k+1)n/4}$, obtained for $l = (k+1)n/4$. Thus, for $k \geq 3$, the attack is not more efficient than 2^n.



(a) Parallel (b) Zipper (built in the front) (c) Zipper (built at the end)

Fig. 9: Construct a building block for a 3-pass simultaneous expandable message

The second-preimage attack on the Zipper hash. When the Zipper hash combines $k > 2$ hash functions by alternatively forward computing and reverse computing, one can still extend Attack 5.

To construct a simultaneous expandable message adaptable for the Zipper hash, one first constructs partial building blocks for all the computational passes in one direction and then constructs them for all the computational passes in the other direction. Note that in each of the building blocks in a simultaneous expandable message for the Zipper hash, there are two pieces of Joux's multi-collisions, each at one of the two ends of the building block. The left Joux's multi-collisions are for synchronizing the reverse computations, and the right Joux's multi-collisions are for synchronizing the forward computations. The complexity to build a cascade simultaneous expandable message for k hash functions can be the same as that of building a parallel one, which is approximately $O(2^\ell + n^{2(k-1)} \cdot 2^{n/2})$. Note that for even k, the simultaneous expandable message is placed at the end of each pass, and the length of the second preimage encoded in the simultaneous expandable message can be chosen to optimize the attack, while for odd k, the simultaneous expandable message is placed in the front end of each pass (see Fig.9b and 9c).

Next, we take $k = 3$ for example to briefly describe the extended attack (see Fig.10). The attack first collects a triple of cyclic nodes $(\bar{x}, \bar{y}, \bar{z})$, each node located in each of the three random functional graphs generated using the three compression functions. Start from $\bar{z}$, we build a $2^{r \cdot \frac{n}{2}}$-Joux's multi-collisions $\mathcal{M}_{\texttt{MC3}}$, ending with $\hat{z}$. By exploiting messages in $\mathcal{M}_{\texttt{MC3}}$, we can start from $\bar{x}$ and build a 2^r-Joux's multi-collisions $\mathcal{M}_{\texttt{MC1}}$, ending with $\hat{x}$. This is essentially computing a 2^r-simultaneous Joux's multi-collision starting from state pair $(\bar{x}, \bar{z})$ and ending with $(\hat{x}, \hat{z})$. We denoted it by $\mathcal{M}_{\texttt{MC13}}$. Start from $\bar{y}$, we build a 2^r-Joux's multi-collisions $\mathcal{M}_{\texttt{MC2}}$ for the reverse computation, ending with $\hat{y}$. Then, find a message block $\bar{m}$ mapping $\hat{z}$ to an internal state c_p in the original computation with $\boldsymbol{M}$. Afterwards, the suffix of the second preimage is fixed to be $m_{p+1} \parallel m_{p+2} \parallel \cdots \parallel m_L$.

Starting from $\hat{y}$, we compute the cascade simultaneous expandable message. Note that, only after we completed the computation in the second pass (which is a reverse computation in the middle of the two forward computations) and get the terminal point $\ddot{y}$, can we start the computation in the first and third passes. The first pass starts from IV and the third pass starts from the terminal point of the second pass, *i.e.*, $\ddot{z} = \ddot{y}$. Because there is only one reverse pass, in each building block, only the right piece of Joux's multi-collision is required (see Fig.9b, if $k \geq 4$, there should be another piece of Joux's multi-collision at the left end in each building block). At last, we launch a meet-in-the-middle procedure using messages in $\mathcal{M}_{\texttt{MC13}}$ and $\mathcal{M}_{\texttt{MC2}}$, to find a triple of starting nodes $(\check{x}, \check{y}, \check{z})$ simultaneously reach the triple of deep-iterates $(\bar{x}, \bar{y}, \bar{z})$, and output the concatenation of the obtained message fragments.

We analysis the complexity of the extended attack in general supposing that there are k computational passes. Note that, among the k computation passes, there are $\lceil k/2 \rceil$ forward computations and $\lfloor k/2 \rfloor$ reverse computations. We have

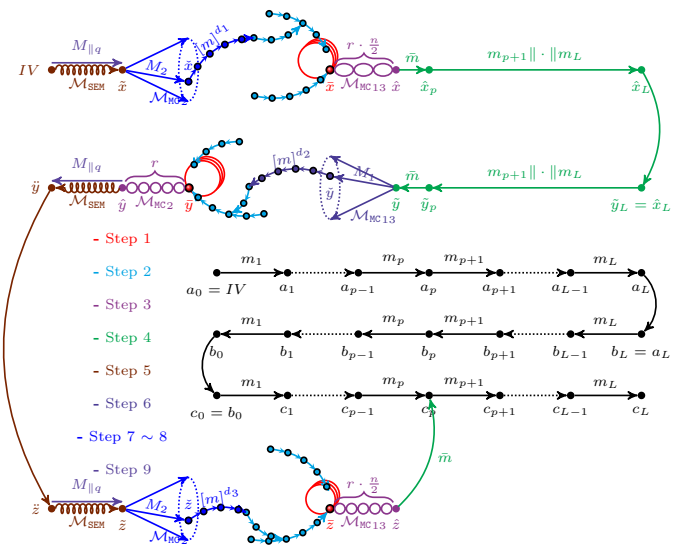


Fig. 10: Second-preimage attacks on 3-pass Zipper

2^{2r} pairs of starting nodes in the meet-in-the-middle procedure in the extended attack, which is the Cartesian product between the set of 2^r nodes for the forward computations and the set of 2^r nodes for the reverse computations. Let the deep-iterates nodes be of depth 2^d; then, each pair succeeds with probability $2^{(k+1)d-kn}$ without using multi-cycles, and the probability amplified to be $2^{(k+1)n/2-kn-n/2+\ell'}$ by using multi-cycles.

If the message length is limited to be no more than $2^{n/2}$, we cannot use the multi-cycles technique. The success of the attack requires $2r = kn - (k+1) \cdot d$, and the attack complexity is approximately $2^t + 2^{\ell'} + 2^{n-\ell} + 2^r \cdot 2^{n-t}$. Setting $d = n/2$ and $\ell' = \ell = n/2$, we get $r = kn/4 - n/4$, and the optimal complexity is $2^{kn/8+3n/8}$, which is less than 2^n for $k < 5$. Concretely, the complexity is $2^{3n/4}$ for $k = 3$ and $2^{7n/8}$ for $k = 4$.

If the message length is not limited, we can use the multi-cycles technique. The success of the attack requires $2r = kn-(k+1)\cdot n/2+n/2-\ell'$, *i.e.*, $r = kn/4-\ell'/2$. The attack complexity is approximately $2^t + 2^{\ell'} + 2^{n-\ell} + 2^{kn/4-\ell'/2+n-t}$. We balance the terms by setting $t = kn/4 - \ell'/2 + n - t = \ell'$ and achieve the optimized complexity $2^{kn/10+2n/5}$ for $n - \ell < \ell'$ and $\ell' = kn/10 + 2n/5$. It is less than 2^n when $k < 6$. Concretely, the complexity is $2^{7n/10}$, $2^{4n/5}$, $2^{9n/10}$ for $k = 3,\ 4,\ 5$ respectively.

The second-preimage attack on Hash-Twice. Attack 6 can be extended using two different strategies: either we only build an interchange structure for two of the k computational passes or we build an interchange structure to control all the k computational passes. The second strategies cannot be more efficient

than the first. Thus, we only describe the extended attack following the first strategy here.

In the extended attack on hashing k times, the frameworks for constructing the first and last passes are almost identical to that of the original attack (see Fig.11). We collect 2^s special nodes in the functional graph of the first hash function. We collect one special node in each of the last $k-1$ hash functions. If the message length is not limited, we select cyclic node as special nodes to be targeted (when using multi-cycles technique, we only consider the correctable distances between two of the k cycles); if the message length is limited, we select deep-iterate nodes as the special nodes to be targeted. We thus get 2^s tuples of k target nodes. From those target nodes in the last $k-1$ passes, we build a $2^{s\cdot n/2}$-simultaneous Joux's multi-collision $\mathcal{M}_{\mathtt{MC}}$ (that is, a set of messages that is Joux's multi-collision for the $k-1$ independent hash functions simultaneously, and is a $2^{s\cdot(n/2)^{k-i+1}}$-Joux's multi-collision for the i-th hash). We herd the 2^s target nodes in the first pass to a single state $\tilde{x}$ by building a diamond using messages in $\mathcal{M}_{\mathtt{MC}}$. We try to hit an internal state in the last computational pass from the endpoint of the last Joux's multi-collision. After that, suffix of the second preimage is fixed. We can then compute the final states of the first $k-1$ passes, which are also initial states of the last $k-1$ passes. We then start from the initial states in the k computational passes, compute a k-pass simultaneous expandable message (see Fig.9a). Starting from the terminal states of the simultaneous expandable message, we build an interchange structure in which two of the k passes have 2^r chains and the remaining $k-2$ passes have a single chain (which essentially is a simultaneous Joux's multi-collision for the remaining $k-2$ passes). We finally try to use the endpoints of the interchange structure to find a tuple of k starting nodes reaching one of the 2^s tuples of k target nodes at the same distance.

If the message length is limited to be no more than $2^{n/2}$, we use deep-iterates nodes (with depth 2^{n-g}, where $g \geq \max(n/2, n-\ell)$) as targeted nodes in the above extended attack. Following an analysis similar to the one in Sect. 8.3, one finds that the complexity is as follows (ignore the constant and polynomial factors):

Step 1: $k \cdot 2^g$ **Step 2:** $k \cdot 2^t$ **Step 3:** $n\sqrt{s} \cdot 2^{n/2+s/2} + s \cdot n^{k-1} \cdot 2^{n/2}$

Step 4: $2^{n-\ell} + 2^{\ell}$ **Step 5:** $2^{\ell} + n^{2k+2} \cdot 2^{n/2}$ **Step 6:** $n^{k-1} \cdot 2^{n/2+2r}$

Step 7: $2^{(k+1)g-t-s-r} + 2^{(k+1)g+t-s-r+\ell-2n} + 2^{kg+\ell-n-r}$

In Sect. 8.3, for the case $k = 2$, we balance the complexity using the first term in the formula of Step 7, because the last two terms in the formula of Step 7 is less than the first term under the best configuration. Here, for the case $k \geq 3$, we balance the complexity using the third term in the formula of Step 7, because the third term is greater than the first two terms. Specifically, we set $t = n/2 + s/2 = n/2 + 2r = kg + \ell - n - r$, *i.e.*, $t = 2kg/3 + 2\ell/3 - n/2$, $s = 4kg/3 + 4\ell/3 - 2n$, and $r = kg/3 + \ell/3 - n/2$. The complexity of the dominant terms is $2^g + 2^{2kg/3+2\ell/3-n/2} + 2^{-4kg/3+g-7\ell/3+3n}$, in which the mid term is always greater than 2^n for $k \geq 4$ under the restrictions $g \geq \max(n/2, n-\ell)$ and $\ell \leq n/2$. Thus, the attack can be more efficient than 2^n only for $k < 4$. For $k = 3$, we set

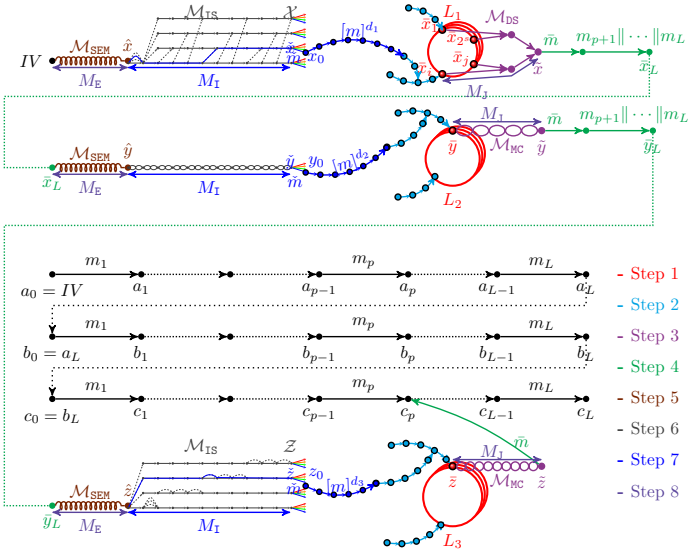


Fig. 11: Second-preimage attacks on 3-pass Hash-Twice

$g = n - \ell$ to optimize the complexity. In this case, $r = k(n-\ell)/3 + \ell/3 - n/2$, which fulfils the restriction $2r \leq \ell$ when $k \leq 3$. The complexity is then

$$2^{n-\ell} + 2^{(4k-3)n/6-2(k-1)\ell/3}.$$

It is optimally $(2k-1)n/6$, obtained for $\ell = n/2$. As a result, for $k = 3$, the optimal complexity is $2^{5n/6}$.

If the message length is not limited and $\ell > n/2$, we use the multi-cycles technique. Following an analysis similar to the one in Sect. 8.2, one can find that the complexity is as follows (ignoring the constant and polynomial factors):

Step 1: $k \cdot 2^{n/2}$ **Step 2:** $k \cdot 2^t$

Step 3: $n\sqrt{s} \cdot 2^{n/2+s/2} + s \cdot n^{k-1} \cdot 2^{n/2}$

Step 4: $2^{n-\ell} + 2^{\ell}$ **Step 5:** $2^{\ell} + n^{2k+2} \cdot 2^{n/2}$

Step 6: $n^{k-1} \cdot 2^{n/2+2r}$ **Step 7:** $2^{(k+2)n/2-t-r-s-\ell}$ **Step 8:** 2^{ℓ}

We make a balance by setting $t = n/2+s/2 = n/2+2r = (k+2)n/2-t-r-s-\ell$, *i.e.*, $t = (2k+9)n/18 - 2\ell/9$, $s = 2kn/9 - 4\ell/9$, and $r = kn/18 - \ell/9$. The complexity is then

$$2^{\ell} + 2^{(2k+9)n/18-2\ell/9}.$$

It is optimally $2^{(2k+9)n/22}$ when $\ell = (9+2k)n/22$. Thus, the attack costs less than 2^n for $k < 7$. Concretely, it is $2^{15n/22}$, $2^{17n/22}$, $2^{19n/22}$, $2^{21n/22}$ for $k = 3,\ 4,\ 5,\ 6$ respectively.

Remark. From the above results, in the functional-graph-based attacks, when use a set of deep iterates as targeted states for attacks with short messages, one has to use the distinguished points method, which is not adequately efficient to attack combiners of more than three hash functions. In contrast, when using a set of cyclic nodes for attacks with long messages, one does not need the distinguished points method and thus can extend the attacks to combiners with n-bit output of up to six Merkle-Damgård hash functions.

10 Summary and Open Problems

In this paper, we study the security of various of hash combiners by devising generic attacks. These attacks show rather surprising results — the security upper bounds of most hash combiners are not as high as commonly believed. Regarding basic security requirements (preimage resistance, second-preimage resistance), they fail to provide more (or even the same) security than that provided by a single ideal hash function, or even less than that provided by its underlying hash functions. See Tab. 1 for a summary of their current security status. In Fig. 12 and Fig. 13, we summarize their detailed security status by drawing trade-off curves between the length of the message and the complexity of the attacks. From these trade-off curves, for combiners with underlying hash functions using Merkle-Damgård construction, the gaps between the security upper bounds and the security lower bounds provided by security proof are quite narrow. However, that is true only for very long messages. For short messages, the gap remains large. That mainly results from the limitation of the key techniques used in our attacks. Our attacks highly exploit the iterated property of the underlying hash functions. Particularly, most of our attacks exploit properties of functional graphs of random mappings generated by fixing a message block input to the compression functions. Thus, they usually involve iterating the compression functions with fixed message block many times. Therefore, our crafted messages are very long, and they are typically composed of a large number of repeated message blocks (which can be easily recognized). Thus, one open problem is how to extend the attacks to apply to short messages or with small patches. Another open problem is how to extend the attacks to combiners with at least one underlying hash function following the HAIFA framework.

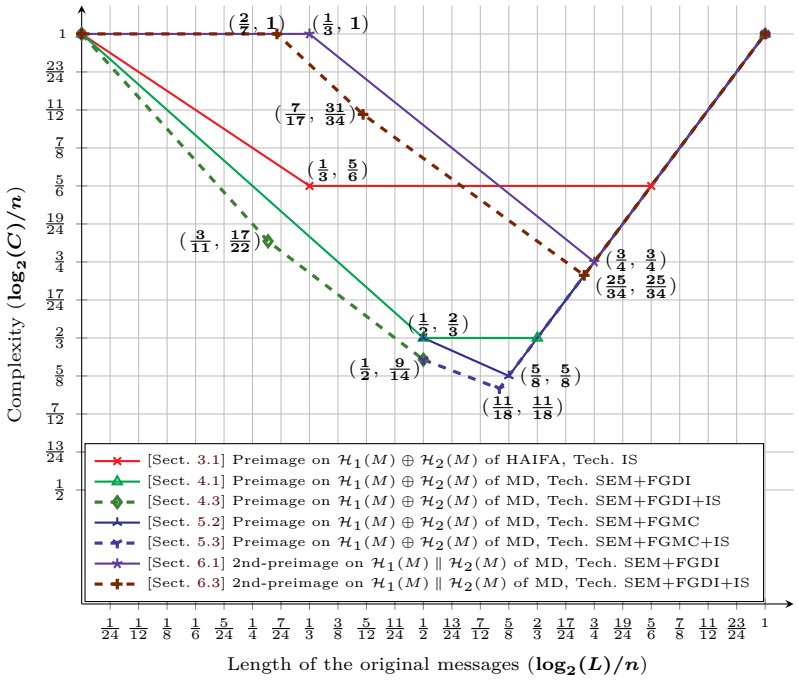


Fig. 12: Trade-offs between the message length and the complexity of attacks on parallel hash combiners

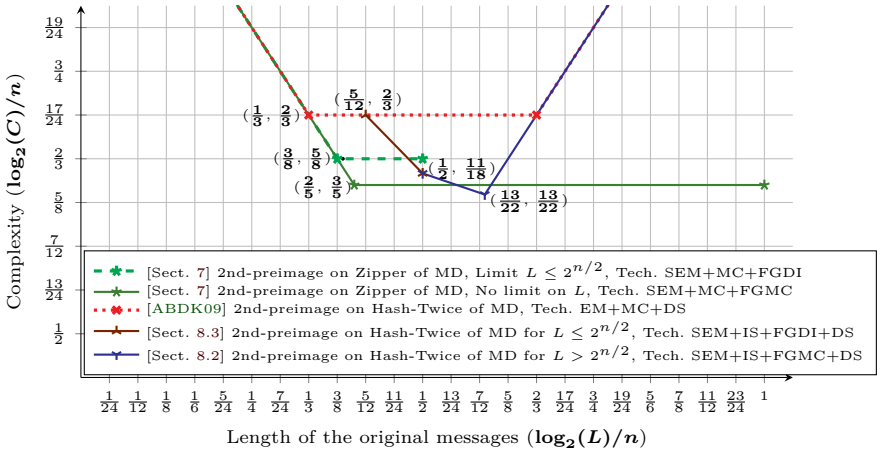


Fig. 13: Trade-offs between the message length and the complexity of attacks on cascade hash combiners

Acknowledgments

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its Strategic Capability Research Centres Funding

Initiative, Nanyang Technological University under research grant M4082123, and Singapore's Ministry of Education under grant M4012049. Itai Dinur is supported in part by the Israeli Science Foundation through grant No. 573/16. Lei Wang is supported by National Natural Science Foundation of China (61602302, 61472250, 61672347), Natural Science Foundation of Shanghai (16ZR1416400), Shanghai Excellent Academic Leader Funds (16XD1401300), 13th five-year National Development Fund of Cryptography (MMJJ20170114).

References

ABD+16. Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. New Second-Preimage Attacks on Hash Functions. *J. Cryptology*, 29(4):657–696, 2016.

ABDK09. Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, and John Kelsey. Herding, second preimage and trojan message attacks beyond merkle-damgård. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh 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.

ABF+08. Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In Nigel 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.

ADG+08. Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor 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*. Springer, 2008.

BB06. Dan Boneh and Xavier Boyen. On the Impossibility of Efficiently Combining Collision Resistant Hash Functions. In Cynthia 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.

BD07. Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. *IACR Cryptology ePrint Archive*, 2007:278, 2007.

BGW18. Zhenzhen Bao, Jian Guo, and Lei Wang. Functional Graphs and Their Applications in Generic Attacks on Iterated Hash Constructions. *IACR Trans. Symmetric Cryptol.*, 2018(1):201–253, 2018.

Bra90. Gilles 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.

BSU12. Simon R. Blackburn, Douglas R. Stinson, and Jalaj Upadhyay. On the complexity of the herding attack and some related attacks on hash functions. *Des. Codes Cryptography*, 64(1-2):171–193, 2012.

BWGG17. Zhenzhen Bao, Lei Wang, Jian Guo, and Dawu Gu. Functional graph revisited: Updates on (second) preimage attacks on hash combiners. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 404–427. Springer, 2017.

CJ15. Shiwei Chen and Chenhui Jin. A second preimage attack on zipper hash. *Security and Communication Networks*, 8(16):2860–2866, 2015.

Cra05. Ronald 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.

CRS+07. Ran Canetti, Ronald L. Rivest, Madhu Sudan, Luca Trevisan, Salil P. Vadhan, and Hoeteck Wee. Amplifying Collision Resistance: A Complexity-Theoretic Treatment. In Menezes [Men07], pages 264–283.

DA99a. Richard Drews Dean and Andrew Appel. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University Princeton, 1999.

DA99b. Tim Dierks and Christopher Allen. The TLS Protocol Version 1.0. *RFC*, 2246:1–80, 1999.

Dam89. Ivan Damgård. A Design Principle for Hash Functions. In Brassard [Bra90], pages 416–427.

Din16. Itai Dinur. New Attacks on the Concatenation and XOR Hash Combiners. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 484–508. Springer, 2016.

DL14. Itai Dinur and Gaëtan Leurent. Improved Generic Attacks against Hash-Based MACs and HAIFA. In Garay and Gennaro [GG14], pages 149–168.

DP07. Orr Dunkelman and Bart Preneel. Generalizing the Herding Attack to Concatenated Hashing Schemes. In *In ECRYPT Hash Function Workshop*. Citeseer, 2007.

DR08. Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.2. *RFC*, 5246:1–104, 2008.

FKK11. Alan O. Freier, Philip Karlton, and Paul C. Kocher. The secure sockets layer (SSL) protocol version 3.0. *RFC*, 6101:1–67, 2011.

FL07. Marc Fischlin and Anja Lehmann. Security-Amplifying Combiners for Collision-Resistant Hash Functions. In Menezes [Men07], pages 224–243.

FL08. Marc Fischlin and Anja Lehmann. Multi-property Preserving Combiners for Hash Functions. In Ran 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.

FLP08. Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust Multi-property Combiners for Hash Functions Revisited. In Aceto et al. [ADG+08], pages 655–666.

FLP14. Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust Multi-Property Combiners for Hash Functions. *J. Cryptology*, 27(3):397–428, 2014.

FO89. Philippe Flajolet and Andrew M. Odlyzko. Random Mapping Statistics. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology - EUROCRYPT '89, Workshop on the Theory and Application of of Cryptographic Techniques, Houthalen, Belgium, April 10-13, 1989, Proceedings*, volume 434 of *Lecture Notes in Computer Science*, pages 329–354. Springer, 1989.

GG14. Juan A. Garay and Rosario 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.

GPSW14. Jian Guo, Thomas Peyrin, Yu Sasaki, and Lei Wang. Updates on Generic Attacks against HMAC and NMAC. In Garay and Gennaro [GG14], pages 131–148.

Hel80. Martin E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Trans. Information Theory*, 26(4):401–406, 1980.

Her05. Amir Herzberg. On Tolerant Cryptographic Constructions. In Alfred Menezes, editor, *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*, volume 3376 of *Lecture Notes in Computer Science*, pages 172–190. Springer, 2005.

Her09. Amir Herzberg. Folklore, practice and theory of robust combiners. *Journal of Computer Security*, 17(2):159–189, 2009.

HS06. Jonathan J. Hoch and Adi Shamir. Breaking the ICE - finding multicollisions in iterated concatenated and expanded (ICE) hash functions. In Matthew 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.

HS08. Jonathan J. Hoch and Adi Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In Aceto et al. [ADG+08], pages 616–630.

JN15. Ashwin Jha and Mridul Nandi. Some Cryptanalytic Results on Zipper Hash and Concatenated Hash. *IACR Cryptology ePrint Archive*, 2015:973, 2015.

Jou04. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew 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.

Jou09. Antoine Joux. *Algorithmic cryptanalysis*. Chapman and Hall/CRC, 2009.

KK06. John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.

KS05. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Cramer [Cra05], pages 474–490.

Leh10. Anja Lehmann. *On the security of hash function combiners*. PhD thesis, Darmstadt University of Technology, 2010.

Lis06. Moses Liskov. Constructing an Ideal Hash Function from Weak Ideal Compression Functions. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography, 13th International Workshop, SAC 2006, Montreal, Canada, August 17-18, 2006 Revised Selected Papers*, volume 4356 of *Lecture Notes in Computer Science*, pages 358–375. Springer, 2006.

LPW13. Gaëtan Leurent, Thomas Peyrin, and Lei Wang. New Generic Attacks against Hash-Based MACs. In Kazue Sako and Palash 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.

LW15. Gaëtan Leurent and Lei Wang. The Sum Can Be Weaker Than Each Part. In Elisabeth Oswald and Marc 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.

Men07. Alfred Menezes, editor. *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*. Springer, 2007.

Mer89. Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [Bra90], pages 428–446.

Mit12. Arno Mittelbach. Hash Combiners for Second Pre-image Resistance, Target Collision Resistance and Pre-image Resistance Have Long Output. In Ivan Visconti and Roberto De 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.

Mit13. Arno Mittelbach. Cryptophia's Short Combiner for Collision-Resistant Hash Functions. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security - 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings*, volume 7954 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2013.

MP14. Bart Mennink and Bart Preneel. Breaking and Fixing Cryptophia's Short Combiner. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis 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.

MRS09. Florian Mendel, Christian Rechberger, and Martin Schläffer. MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners. In Mitsuru 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.

NS07. Mridul Nandi and Douglas R. Stinson. Multicollision Attacks on Some Generalized Sequential Hash Functions. *IEEE Trans. Information Theory*, 53(2):759–767, 2007.

Pie07. Krzysztof Pietrzak. Non-trivial Black-Box Combiners for Collision-Resistant Hash-Functions Don't Exist. In Moni 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.

Pie08. Krzysztof Pietrzak. Compression from Collisions, or Why CRHF Combiners Have a Long Output. In David A. Wagner, editor, *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, volume 5157 of *Lecture Notes in Computer Science*, pages 413–432. Springer, 2008.

PK14. Léo Perrin and Dmitry Khovratovich. Collision Spectrum, Entropy Loss, T-Sponges, and Cryptanalysis of GLUON-64. In Carlos Cid and Christian 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.

Pre93. Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit te Leuven, 1993.

PW14. Thomas Peyrin and Lei Wang. Generic Universal Forgery Attack on Iterative Hash-Based MACs. In Phong Q. Nguyen and Elisabeth 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.

Rja09. Michal Rjasko. On Existence of Robust Combiners for Cryptographic Hash Functions. In Peter Vojtás, 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.

vOW99. Paul C. van Oorschot and Michael J. Wiener. Parallel Collision Search with Cryptanalytic Applications. *J. Cryptology*, 12(1):1–28, 1999.

Wag02. David A. Wagner. A Generalized Birthday Problem. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2002.

WY05. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [Cra05], pages 19–35.

WYY05. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor 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.

A Pseudo-codes of Algorithms

Algorithm 1 Building a 2^t-Joux's Multi-Collision

function JouxMultiCollision(h, x_0, t)
  $\mathcal{M}_{\mathtt{MC}} \leftarrow \{\}$
  for $1 \le i \le t$ **do**
    $(x_i, m, m') \leftarrow$ Collision1(h, x_{i-1})
    $\mathcal{M}_{\mathtt{MC}} \leftarrow \mathcal{M}_{\mathtt{MC}} \| (m, m')$
  end for
  return $(x_t, \mathcal{M}_{\mathtt{MC}})$
end function

function Collision1(h, x)
  $\mathcal{T} \leftarrow \{\}$
  loop
    $m \leftarrow \$,\ y \leftarrow h(x, m)$
    if $\mathcal{T}[y]$ exists **then return** $(y, m, \mathcal{T}[y])$
    else $\mathcal{T}[y] \leftarrow m$
    end if
  end loop
end function

Algorithm 2 Building a single switch

function Switch(h_1, h_2, a, b, b')
  $x \leftarrow \varnothing,\ \mathcal{M}_{\mathtt{MC}} \leftarrow \emptyset$
  $(x, \mathcal{M}_{\mathtt{MC}}) \leftarrow$ JouxMultiCollision($h_1, a, n/2$)
  $\mathcal{T} \leftarrow \{\}$
  for each $M \in \mathcal{M}_{\mathtt{MC}}$ **do**
    $y \leftarrow h_2^*(b, M),\ \mathcal{T}[y] \leftarrow M$
  end for
  for each $M \in \mathcal{M}_{\mathtt{MC}}$ **do**
    $y \leftarrow h_2^*(b', M)$
    if $\mathcal{T}[y]$ exists **then**
      return $(\mathcal{T}[y], M)$
    end if
  end for
end function

Algorithm 3 Building and using a T-interchange structure

function INTERCHANGE(h_1, h_2, IV_1, IV_2)
  $a_0 \leftarrow IV_1$, $b_0 \leftarrow IV_2$
  for $1 \leq k < T$ **do**
    $a_k \leftarrow \$$, $b_k \leftarrow \$$
  end for
  for $1 \leq j < T$ **do**
    $(M, M') \leftarrow$ SWITCH(h_1, h_2, a_0, b_0, b_j)
    $\boldsymbol{M} \leftarrow \boldsymbol{M} \parallel M$, $\boldsymbol{M'} \leftarrow \boldsymbol{M'} \parallel M'$
    for $0 \leq k < T$ **do**
      $a_k \leftarrow h_1^*(a_k, M)$, $b_k \leftarrow h_2^*(b_k, M)$
    end for
  end for
  for $1 \leq j < T$ **do**
    for $1 \leq i < T$ **do**
      $(M, M') \leftarrow$ SWITCH(h_2, h_1, b_j, a_0, a_i)
      $\boldsymbol{M} \leftarrow \boldsymbol{M} \parallel M$, $\boldsymbol{M'} \leftarrow \boldsymbol{M'} \parallel M'$
      for $0 \leq k < T$ **do**
        $a_k \leftarrow h_1^*(a_k, M)$, $b_k \leftarrow h_2^*(b_k, M)$
      end for
    end for
  end for
  return $(\boldsymbol{M}, \boldsymbol{M'})$
end function

function SELECTMESSAGE($\boldsymbol{M}, \boldsymbol{M'}, j, k$)
  $\mu \leftarrow \boldsymbol{M}$
  if $k \neq 0$ **then**
    $\mu[k-1] \leftarrow \boldsymbol{M'}[k-1]$
  end if
  if $j \neq 0$ **then**
    $\mu[(k+1) \cdot (T-1) + j - 1] \leftarrow \boldsymbol{M'}[(k+1) \cdot (T-1) + j - 1]$
  end if
  return μ
end function

Algorithm 4 Constructing a building block for a simultaneous expandable message

function SEMBLOCK(x_0, y_0, i)
  $xp \leftarrow h_1^*(x_0, [0]^{i-C})$
  $(sp, m_1, m_1') \leftarrow$ COLLISION2(h_1, x_0, xp)
  $(x_1, \mathcal{M}_{\texttt{MC}}) \leftarrow$ JOUXMULTICOLLISION($h_1, sp, C-1$)
  $\mathcal{M}_{\texttt{short}} \leftarrow m_1 \times \mathcal{M}_{\texttt{MC}}$, $\mathcal{M}_{\texttt{long}} \leftarrow ([0]^{i-C} \parallel m_1') \times \mathcal{M}_{\texttt{MC}}$
  $\mathcal{Y}_{\texttt{short}} \leftarrow \emptyset$
  for each $ms' \in \mathcal{M}_{\texttt{short}}$ **do**
    $ys' \leftarrow h_2^*(y_0, ms')$, $\mathcal{Y}_{\texttt{short}} \leftarrow_{insert} (ys', ms')$
  end for
  for each $ml' \in \mathcal{M}_{\texttt{long}}$ **do**
    $yl' \leftarrow h_2^*(y_0, ml')$
    if $yl' \in \mathcal{Y}_{\texttt{short}}$ **then**
      $(y_1, ms, ml) \leftarrow (yl', ms', ml')$
    end if
  end for
  return (x_1, y_1, ms, ml)
end function

function COLLISION2(h, x, x')
  $\mathcal{T} \leftarrow \{\}$
  for $1 \le i \le 2^{n/2}$ **do**
    $m \leftarrow \$$, $y \leftarrow h(x, m)$, $\mathcal{T}[y] \leftarrow m$
  end for
  loop
    $m' \leftarrow \$$, $y' \leftarrow h(x', m')$
    if $\mathcal{T}[y']$ exists **then**
      return $(y', \mathcal{T}[y'], m')$
    end if
  end loop
end function

Algorithm 5 Expanding the functional graph of f (generating 2^t nodes in $\mathcal{FG}_f$)

1: **procedure** GEN(t)
2: $\mathcal{G} \leftarrow \emptyset$
3: **while** $|\mathcal{G}| < 2^t$ **do**
4: $\mathcal{C} \leftarrow \emptyset$, $x \leftarrow_{\$} \{0, 1, \ldots, 2^n - 1\} \setminus \mathcal{G}$
5: **while** true **do**
6: **if** $x \in \mathcal{G}$ or $x \in \mathcal{C}$ **then**
7: $\mathcal{G} \leftarrow_{merge} \mathcal{C}$, **go to line** 3
8: **else** $\mathcal{C} \leftarrow_{insert} x$, $x \leftarrow f(x)$
9: **end if**
10: **end while**
11: **end while**
12: **return** $\mathcal{G}$
13: **end procedure**

B Optimized Interchange Structure

We now describe an optimized attack using only $(2^t-1)(2^t-1)$ switches rather than $2^{2t}-1$. The attack also requires multi-collision structures, as introduced by Joux [Jou04].

We replace the first 2^t-1 switches with a 2^t- Joux's multi-collision in $\mathcal{H}_1$, and we use those messages to initialize all the b_k chains in $\mathcal{H}_2$. We can also optimize the first series of switches in $\mathcal{H}_2$ in the same way: we build a 2^t-multi-collision in $\mathcal{H}_2$ starting from b_0, and we use those messages to initialize the a_j chains in $\mathcal{H}_1$. This is illustrated by Fig. 14, and the detailed attack is given in Alg. 6.

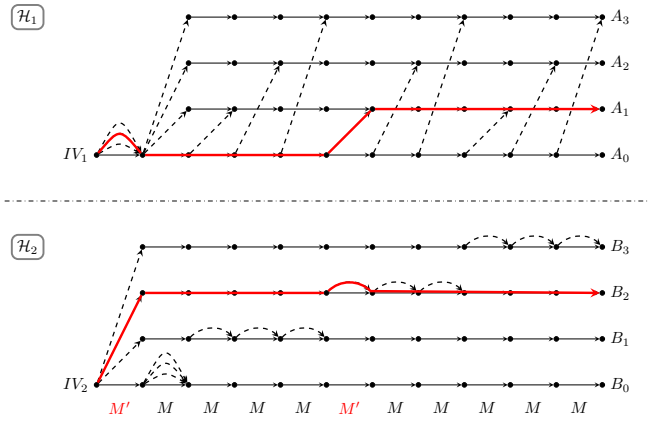


Fig. 14: Optimized interchange structure

Algorithm 6 Optimized T-interchange structure (denote $T = 2^t$)

function INTERCHANGE(h_1, h_2, IV_1, IV_2, T)
  $a_0 \leftarrow IV_1$, $b_0 \leftarrow IV_2$
  $(a_0, \mathcal{M}_{\texttt{MC0}}) \leftarrow$ JOUXMULTICOLLISION(h_1, a_0, t)
  for $0 \le k < T$ **do**
    $b_k \leftarrow h_2^*(b_0, \mathcal{M}_{\texttt{MC0}}[k])$
  end for
  $(b_0, \mathcal{M}_{\texttt{MC1}}) \leftarrow$ JOUXMULTICOLLISION(h_2, b_0, t)
  $a_0 \leftarrow h_1^*(a_0, \mathcal{M}_{\texttt{MC1}}[0])$
  for $1 \le k < T$ **do**
    $a_k \leftarrow h_1^*(a_0, \mathcal{M}_{\texttt{MC1}}[k])$
    $b_k \leftarrow h_2^*(b_k, \mathcal{M}_{\texttt{MC1}}[0])$
  end for
  for $2 \le j < T$ **do**
    for $1 \le i < T$ **do**
      $(M, M') \leftarrow$ SWITCH(h_2, h_1, b_j, a_0, a_i)
      $\boldsymbol{M} \leftarrow \boldsymbol{M} \parallel M$, $\boldsymbol{M'} \leftarrow \boldsymbol{M'} \parallel M'$
      for $0 \le k < T$ **do**
        $a_k \leftarrow h_1^*(a_k, M)$
        $b_k \leftarrow h_2^*(b_k, M)$
      end for
    end for
  end for
  return $(\mathcal{M}_{\texttt{MC0}}, \mathcal{M}_{\texttt{MC1}}, \boldsymbol{M}, \boldsymbol{M'})$
end function

function SELECTMESSAGE($\mathcal{M}_0, \mathcal{M}_1, \boldsymbol{M}, \boldsymbol{M'}, j, k$)
  if $j = 0$ **then**
    return $\mathcal{M}_0[k] \parallel \mathcal{M}_1[0] \parallel M$
  else if $k = 0$ **then**
    return $\mathcal{M}_0[0] \parallel \mathcal{M}_1[j] \parallel M$
  else
    $\mu \leftarrow M$
    $\mu[(k-1)\cdot(T-1)+j-1] \leftarrow M'[(k-1)\cdot(T-1)+j-1]$
    return $\mathcal{M}_0[k] \parallel \mathcal{M}_1[0] \parallel \mu$
  end if
end function

C On Problem Raised by Dependency Between Chain Evaluations

Suppose $\bar{x}$ and $\bar{y}$ are both of depth 2^{n-g_1}. From Observation 2 in Sect. 2.7, we conclude that the probability of encountering $\bar{x}$ and $\bar{y}$ at the same distance in chains (of f_1 and f_2) evaluated from x_0 and y_0 is approximately 2^{n-3g_1}. Thus, in Sect. 4.2 and Sect. 6.2, we conclude that if the trials of chain evaluations are independent, we need to compute about 2^{3g_1-n} chains from different starting points. However, since various trials performed by selecting different starting points for the chains are dependent, it might require further proof for the conclusion.

More specifically, when the number of nodes evaluated along chains exceeding 2^{n-d}, a new chain of length 2^d is very likely to collide with a previously evaluated node due to the birthday paradox ($2^d \times 2^{n-d} = 2^n$). Thus, the outcome of this chain evaluation is determined. As a result, new chains are all related with already evaluated chains, and the dependency between them affects the outcome non-negligibly after having evaluated 2^{n-d} nodes.

However, we notice that in our attacks, the actual birthday bound for the non-negligible dependency between trials is 2^{2n-2d} instead of 2^{n-d} because in each trail, there are two chain evaluations. One is in $\mathcal{FG}_{f_1}$, and the other is in $\mathcal{FG}_{f_2}$. The chain evaluation in $\mathcal{FG}_{f_1}$ can be seen as independent with a chain evaluation in $\mathcal{FG}_{f_2}$. After having evaluated 2^{n-d} nodes in each of the two functional graphs, there is indeed a high probability for each new chain colliding with previously evaluated chains. However, for a new *pair of* chain evaluations, the probability for both chains colliding with the chains evaluated in a previous trial is significant only after having evaluated 2^{2n-2d} nodes due to the birthday paradox. That is, trials cannot be seen as independent only after having evaluated 2^{2n-2d} nodes. Note that in our attacks, required number of trials is 2^{2n-3d}, thus the total evaluated number of nodes is $2^{2n-3d} \cdot 2^{d+1} \approx 2^{2n-2d}$ which exactly falls on the birthday bound. Thus, the dependency between the trials is negligible and the complexity analysis of the corresponding attacks is justified.