Functional Graphs and Their Applications in Generic Attacks on Iterated Hash Constructions

Zhenzhen Bao[1], Jian Guo[1] and Lei Wang[2,3]

[1] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[2] Shanghai Jiao Tong University, Shanghai, China
[3] Westone Cryptologic Research Center, Beijing, China
baozhenzhen10@gmail.com,guojian@ntu.edu.sg,wanglei_hb@sjtu.edu.cn

Abstract. We provide a survey about generic attacks on cryptographic hash constructions including hash-based message authentication codes and hash combiners.
We look into attacks involving iteratively evaluating identical mappings many times. The functional graph of a random mapping also involves iteratively evaluating the mapping. These attacks essentially exploit properties of the functional graph. We map the utilization space of those properties from numerous proposed known attacks, draw a comparison among classes of attacks about their advantages and limitations. We provide a systematic exposition of concepts of cycles, deep-iterate images, collisions and their roles in cryptanalysis of iterated hash constructions. We identify the inherent relationship between these concepts, such that case-by-case theories about them can be unified into one knowledge system, that is, *theories on the functional graph of random mappings*. We show that the properties of the cycle search algorithm, the chain evaluation algorithm and the collision search algorithm can be described based on statistic results on the functional graph. Thereby, we can provide different viewpoints to support previous beliefs on individual knowledge.
In that, we invite more sophisticated analysis of the functional graph of random mappings and more future exploitations of its properties in cryptanalysis.

Keywords: Functional graph · hash-based MAC · Hash combiner · Cycle · Deep-iterate image · Collision · State recovery attack · Forgery attack · (Second) Preimage attack

1 Introduction

Cryptographers build cryptographic functions using an *iterated construction* (which is the de-facto standard) to simplify the security proof of the new designs when developing them in theory and to ease the implementation of the designs when using them in practice. Iterated constructions are expected to provide a simple and secure way to process data of various length, e.g., iterated cryptographic hash functions. Iterated cryptographic hash functions map messages of arbitrary length to digests of fixed length. To achieve this, they iterate on fixed length input compression functions. Usually, those compression functions iterated on within each construction are identical (or equivalent up to the addition of constants).

To make security proof possible and to meet the security expectation on the iterated hash construction, the compression function iterated on should behave like an ideal primitive — usually a random mapping from a finite set into itself. Structures of a random mapping can be visualized by the corresponding functional graph. The functional graph of a random mapping is defined by the successive iteration of this mapping. An interesting primary application of functional graphs in cryptography is the cycling experiments on standard block cipher DES and the discovery of functional graph representing an iteration

IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2018, No. 1, pp. 201–253
DOI:10.13154/tosc.v2018.i1.201-253

structure of DES [KRS88]. Apparent structures of the functional graph of a random mapping might reveal intermediate information of the processing procedure under the mapping. Characteristics and special parts of the functional graph of an underlying function might be exploited to attack the entire iterated construction. Instances of these kinds will be exhibited in this paper. Specifically, we will show how some statistical properties of the functional graph are utilized in generic attacks against various cryptographic hash constructions built on a classical iterated construction — the Merkle-Damgård construction [Mer89, Dam89].

The Merkle-Damgård construction (MD) iteratively applies the same compression function to process different message blocks evenly split from the given padded message. It is broadly adopted by hash function designs such as MD5 and SHA-1 which are widely used to construct more advanced hash constructions, such as hash-based message authentication codes (MACs) and hash combiners. The hash-based MACs use keys to provide authentication for data. The hash combiners combine two or more hash functions to provide more security or to be robust so they will remain secure as long as at least one hash function is secure. Various generic attacks have revealed weaknesses in the Merkle-Damgård construction, including the multi-collision attack [Jou04], second-preimage attacks [Dea99, KS05, ABD+16] and the herding attack [KK06]. These attacks commonly exploit the iterative property of the construction which allows inner collisions to propagate to the output. They encourage more generic attacks on hash designs built on MD construction including hash-based MACs and hash combiners. These generic attacks profoundly exploit the iterative property and are efficient by taking advantage of observations on the functional graph of the underlying mapping.

We note that large body of the knowledge on the statistical properties of the random functional graph can be cited from [FO89], derived using approaches in an independent research area — analytic combinatorics. Actually, random mapping is not an exclusive for the design of cryptographic hash function, it is a fundamental model for many aspects of cryptology. More than that, it has also been intensely studied and widely applied in various independent parallel lines of research, such as random number generation, computational number theory, and the analysis of algorithms. New knowledge on characteristics of random mapping from one research line has a potential influence on the others. The influence might become significant until several decades later. This paper shows one instance of such influence. Explicitly, we show how old knowledge from research on random graphs and trees [RS67, Pro74, Mut88, FO89, FS09] became the theoretical basis of recent generic attacks on hash constructions [PSW12, LPW13, PW14, GPSW14, DL14, DL17, Din16, BWGG17]. In the former line, researchers usually use profound and solid mathematical methods including combinatorial mathematics, complex analysis, and probability and statistics. In the latter line, researchers usually use delicate and smart cryptanalysis techniques which might be based on reasonable heuristic assumptions, intuitive probabilistic arguments and experimental verifications. Our motivation in this paper is to show the potential for the latter to be supported further by the former and the potential of further more fruitful cryptanalysis results. And we call for more joint effort between the two research areas to build a complete knowledge system.

1.1 Our Contributions

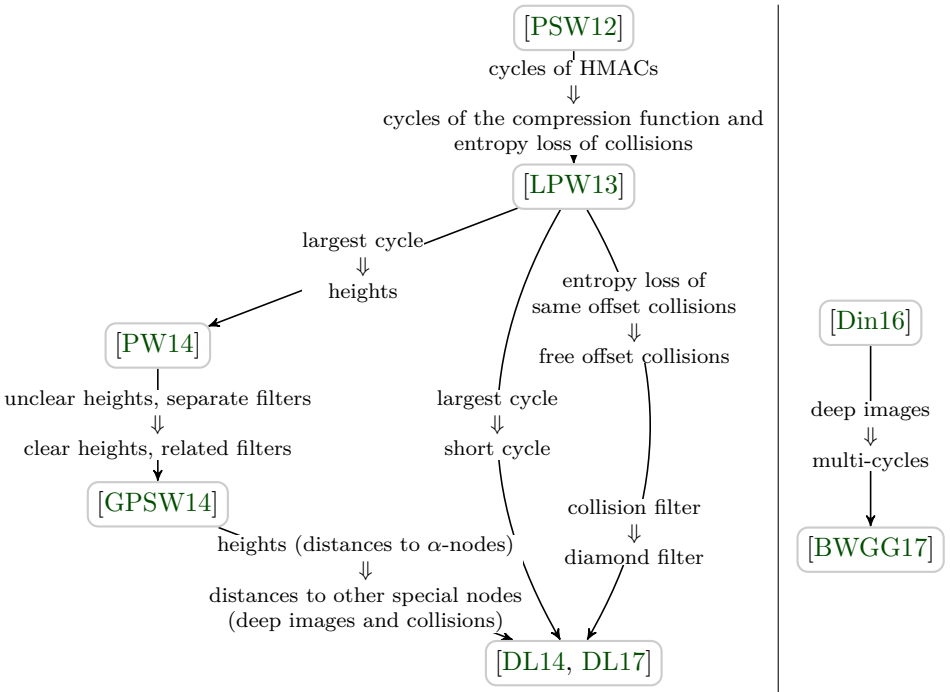
We provide a survey about generic attacks on cryptographic hash constructions. Specific hash constructions concerned include hash-based MACs and hash combiners with Merkle-Damgård underlying hash functions. The survey covers a toolbox used in generic attacks. For hash-based MACs, the survey covers various generic attacks, including distinguishing, state recovery attacks and forgery attacks. For hash combiners, the survey covers generic preimage and second-preimage attacks on various hash combiners. We try to provide a uniform description of these attacks, and try to provide brief complexity analysis which

shows the inherent limitations or advantages over related attacks (refer to Fig.1 for a list of the main surveyed papers and their technical relations).

We look into attacks involving iteratively evaluating a mapping many times. The functional graph of a random mapping also involves iteratively evaluating the mapping. These attacks are typical applications of the functional graph. With their invention, more and more properties of functional graph are exploited and better trade-offs of the attack complexity are obtained. We map the utilization space of those properties from these numerous proposed attacks, draw a comparison among classes of attacks of their advantages and limitations, and discuss the fundamental reasons for the advantages and limitations.

We provide a systematic exposition of concepts — cycles, deep-iterate images, collisions and their roles in cryptanalysis on iterated hash constructions. We identify the inherent relationship between these concepts, such that case-by-case theories about them can be unified into one knowledge system, that is, *theories on the functional graph of random mappings*. We show that the presented attacks commonly exploit properties of some probabilistic algorithms restoring part of the functional graph, that is the cycle search algorithm, the chain evaluation algorithm and the collision search algorithm. Outputs of these algorithms are essentially special nodes in the functional graph which can be analysis using statistical results on parameters of functional graph. Thereby, we can provide different viewpoints to support previous beliefs on individual knowledge.

In that, we invite more sophisticated analysis of properties of the functional graph of random mappings, which might stimulate insightful analysis of generic properties of the iterated construction, and which might lead potential future exploitations of these properties in cryptanalysis.



(a) On Hash-based MACs

(b) On Hash combiners

Figure 1: Main surveyed papers about generic attacks related to the functional graph

1.2 Notations and Roadmap in the Rest of Paper

Notations. We summarize below the notation that is shared across various attacks.

$\mathcal{H}$,	$\mathcal{H}_1, \mathcal{H}_2$	: Hash functions (the underlying hash functions in the MAC algorithms or hash combiners)
IV,	IV_1, IV_2	: Initialization vectors of $\mathcal{H}$, $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively
h,	h_1, h_2	: Compression functions of $\mathcal{H}$, $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively
h^*,	h_1^*, h_2^*	: Compression functions iterated over several blocks (in particular, $\mathcal{H}(M) = h^*(IV, M)$, and $\mathcal{H}_i(M) = h_i^*(IV_i, M)$ for $i =\in \{1, 2\}$)
V		: Targeted image
m		: Message block
$C_{\|i}$		: The first i blocks of a challenge message C
$[m]^q$		: Message fragment formed by concatenating q message blocks m, with $[m] = [m]^1$
$M_{\|q}$		: Message fragment with q message blocks
$\hat{M}$		: Message fragment
$M = m_1 \| \ldots \| m_L$		: Target message or computed preimage (of L message blocks)
M'		: Computed second preimage
$\mathcal{M}$		: A set of messages
$\mathcal{M}_{\texttt{MC/EM/SEM/DS/IS}}$		: The set of messages in a standard multi-collision or an expandable message or a simultaneous expandable message or a diamond structure or an interchange structure
$a_0, \ldots, a_L$		: Sequence of internal states computed during the invocation of h_1 on M, $a_0 = IV_1$
$b_0, \ldots, b_L$		: Sequence of internal states computed during the invocation of h_2 on M, $b_0 = IV_2$
x,	y	: Computed internal states
L		: Length of M (measured in the number of blocks)
L'		: Length of M' (measured in the number of blocks, in most attacks, $L' = L$)
l		: In Sect. 4, we assign l to be the bit-size of the internal state of the hash function (following conventions of previous literatures); In Sect. 5, we reassign l to represent the binary logarithm of the length of the message M, i.e., suppose the message is of $L = 2^l$ blocks (also following conventions of related literature).
n		: Bit-size of the output of a hash function; In Sect. 2.4 and Sect. 5, we suppose the compression function h has an n-bit internal state (narrow-pipe hash functions)
b		: Bit-size of a message block
N		: In Sect. 3, we suppose those considered random mappings are from a finite N-set domain to a finite N-set range ($N = 2^n$).
$\mathcal{FG}_f$		: The functional graph of a random mapping f
$x \xrightarrow{m} x'$	$x \xrightarrow{\hat{M}} x'$	: 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 provides basics conceptions, security requirements and attack techniques (a toolbox) related to the hash functions, the hash-based MACs, and the hash combiners. Section 3 introduces and summarizes important properties of a central subject — the functional graph. These properties are heavily utilized by the following attacks. Sections 4 and 5 then separately present various attacks against hash-based MACs and hash combiners. These considered attacks are all typical applications of the properties of functional graph. We show how more and more properties of the functional graph are discovered and cleverly exploited by the series of attacks. These attacks share common structures and techniques, so we try to highlight the changes from one to another, thereby shows the limitations or advantages of each attack over the related attacks. Finally, Section 6 provides a systematic exposition of concepts exploited in various attacks, identifies their inherent relationship with the properties of functional graph, discusses and compares different attacks by plotting trade-off curves between their complexities and the message length, and raises some open problems.

2 Preliminaries

2.1 Hash Functions and Iterative Constructions

A cryptographic hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ maps messages of arbitrary length to short digests of fixed length n-bit, e.g. 256, 384 or 512 bits. Such functions are widely used in various cryptosystems to provide integrity, authentication, and randomness. Applications include software integrity verifications, digital signatures, password storage, proof-of-work systems, secure keys generations, etc. Different applications might require different attack resistances, three main security requirements on cryptographic hash functions are:

- **Collision resistance:** It should be *computationally* hard to find two distinct messages M and M' such that $\mathcal{H}(M) = \mathcal{H}(M')$.
- **Preimage resistance:** Given a target V, it should be *computationally* hard to find a message M such that $\mathcal{H}(M) = V$.
- **Second preimage resistance:** Given a message M, it should be *computationally* hard to find a distinct message $M' \neq M$ such that $\mathcal{H}(M) = \mathcal{H}(M')$.

A secure hash function with n-bit output is expected to offer $n/2$-bit security for collision resistance and n-bit for (second) preimage resistance, which are respectively limited by the ability of the generic birthday attack and the brute-force attack.

Merkle-Damgård Construction (MD) [Dam89, Mer89]. Most hash functions used in practice are of an iterated construction. To be able to process messages of various length, the iterated construction uses a small compression function with fixed-size input. The input is composed of an n-bit part and a b-bit part, the former is an internal state x, and the latter is a message block m. A standard iterated construction is the Merkle-Damgård construction. For a given message M, MD first appends some padding bits

and the message length $|M|$ to M so that the total length is divisible by b. It then splits the padded message into a sequence of message blocks $m_1 \| m_2 \| \ldots \| m_L$. Note that, the last block m_L is encoded with the message length $|M|$, that is called *length padding* or *Merkle-Damgård strengthening*. After that, it feeds the message blocks one after another to the compression function and updates the internal state x iteratively which is initialized by a public initialization vector IV. After processing the last message block, it updates the internal state under a finalization transformation and outputs the state as the digest of the message (see Fig. 2, because the finalization transformation does not affect those generic attacks, we omit it hereafter).

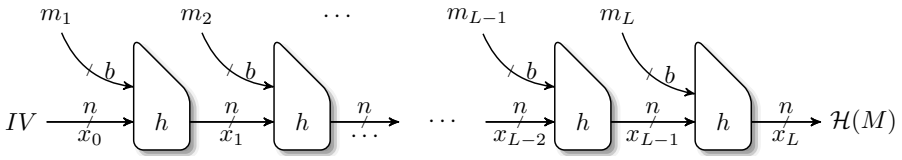


Figure 2: Narrow-pipe Merkle-Damgård hash function

There are narrow-pipe iterative hash functions, *i.e.*, the size of the internal state is the same as the size of the output. There are also wide-pipe iterative hash functions, *i.e.*, the size of the internal state is larger than the size of the output.

2.2 Hash-based MACs

The cryptographic hash function is one of the common primitive used to construct Message Authentication Codes (MACs) which are expected to provide integrity and authentication. In this usage scenario of the hash function, a secret key K is required as input besides the message M. Again, output $T = \mathtt{MAC}(K, M)$ is a fixed-length *tag* used to authenticate the message. The receiving party should use the same key K to verify it, which determines the symmetric attribute of such MACs. The key K is usually used at both the beginning and the end of the updating procedure of the internal state inside a hash-based MAC, this is mostly necessary for narrow-pipe-based designs. Two classical hash-based MACs are NMAC and HMAC [BCK96]. Besides, there are also envelope-MAC [Tsu92, KR95], i.e., $\mathcal{H}(K\|pad\|M\|K')$ and $\mathcal{H}(K\|pad\|M\|K)$, and sandwich-MAC [Yas07], i.e., $\mathcal{H}(K\|pad\|M\|pad'\|K)$. Here we make an explicit convention about the size of security parameters for the hash-based MACs, so that we can describe typical instances of them and define various security requirements on them with respect to these parameters later: k-bit key, b-bit message blocks, l-bit internal states, n-bit tag.

NMAC [BCK96]. NMAC calls an underlying hash function twice and requires to regard the initialization vector IV of the underlying hash function as a secret input parameter instead of a fixed public value. It requires two keys K_{in} and K_{out}. In the first call, it first replaces the public IV with K_{in}, then computes the digest of the message M under the underlying hash function, and finally gets the intermediate result $T = \mathcal{H}_{K_{in}}(M)$. In the second call, it first replaces the public IV with K_{out}, then computes the digest of the first intermediate result T, and finally gets the output $\mathcal{H}_{K_{out}}(\mathcal{H}_{K_{in}}(M))$. Essentially, NMAC sequentially calls a keyed hash function twice with two different keys being the initialization vector and outputs the digest of the first digest. NMAC is formally defined as:

$$\mathtt{NMAC}(K_{out}, K_{in}, M) = \mathcal{H}_{K_{out}}(\mathcal{H}_{K_{in}}(M)).$$

HMAC [BCK96]. Similar to NMAC, HMAC also sequentially calls an underlying hash function twice. Unlike NMAC, HMAC does not require the underlying hash function

to be keyed. It concatenates the padded key $K \oplus ipad$ with message M and updates the internal state which is initialized by the original fixed public IV of the underlying hash function. After the first call, it gets an intermediate result $T = \mathcal{H}(K \oplus ipad \| M)$. Then, it concatenates the padded key $K \oplus opad$ with the generated intermediate result T and goes through the underlying hash function updating the internal state which is also initialized by the original fixed public IV. Essentially, HMAC can be viewed as a single-key version of NMAC. It uses a single k-bit key K to derive two different l-bit keys, that is: $K_{in} = h(IV, K \oplus ipad)$ and $K_{out} = h(IV, K \oplus opad)$, where $ipad$ and $opad$ are two b-bit constants. And then, it does exactly the same as NMAC. HMAC is formally defined as:

$$\texttt{HMAC}(K, M) = \mathcal{H}(K \oplus opad \| \mathcal{H}(K \oplus ipad \| M)).$$

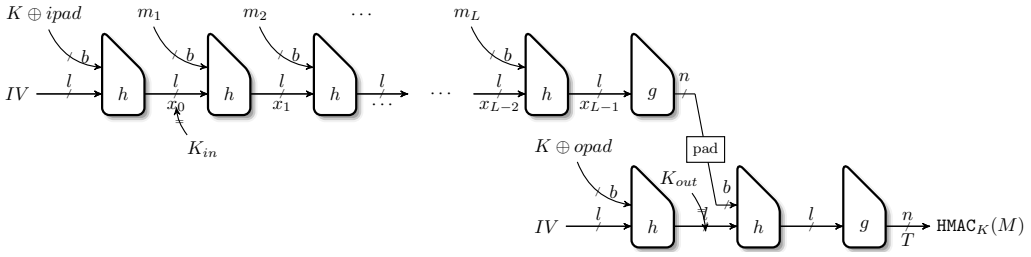


Figure 3: HMAC with a Merkle-Damgård hash function

HMAC/NMAC has been proven to be variable-length input pseudo-random functions (VI-PRF) assuming the underlying compression function is itself a fixed-length input pseudo-random function (FI-PRF).

Compared with NMAC, HMAC has an interesting property for practical utilization, that is, it can use arbitrary key size and can be instantiated with an unkeyed hash function (like MD5, SHA-1, etc). Thus, it has been standardized (by NIST, ANSI, IETF, and ISO) and widely deployed, in particular for banking processes and Internet security protocols (e.g. SSL, TLS, SSH, IPSec) and has APIs in public libraries and modules of various program languages (e.g., Python hashlib, Crypto++ Library, Java Cryptography Extension).

Security Requirements. MAC algorithms are usually expected to meet the following security requirements:

- **Key recovery resistance:** It should be impossible to recover the secret key faster than by exhaustive search. That is, key recovery should cost no less than 2^k computations for a perfectly secure k-bit key MAC.
- **State recovery resistance:** It should be impossible to recover the intermediate state faster than by exhaustive search. That is, state recovery should cost no less than $\min(2^k, 2^l)$ computations for a perfectly secure k-bit key and l-bit state MAC.
- **Forgery resistance:** It should be *computationally* impossible for an adversary to forge a valid tag of M without knowing the secret key and without having queried M, when the message M is
  - **Existential forgery:** chosen by the adversary before or after interacting with the oracle.
  - **Selective forgery:** chosen and first committed on by the adversary before interacting with the oracle.

– **Universal forgery:** given to the adversary as a challenge.

That is, forgery attack should cost no less than $\min(2^k, 2^n)$ computations for a perfectly secure k-bit key and n-bit output MAC. However, for NMAC and HMAC whose underlying hash functions are iterated, there is a generic existential forgery attack (named extension attack) requiring $2^{l/2}$ computations [PvO95]: the adversary first asks birthday bound number of queries to obtain an internal collision between two messages (M, M'), then he appends an arbitrary extra block m to both of the colliding messages. Thereby, he obtains another pair of messages $(M\|m, M'\|m)$ which lead to colliding MAC outputs. By querying for $M\|m$, the attacker can simply forge a valid tag for $M'\|m$.

Besides, cryptanalysists also concern security notions based on indistinguishable property of hash-based MACs, that is, distinguishing-R and distinguishing-H [KBPH06b, KBPH06a]. Note that, the notion of distinguishing-R is identical to the notion of PRF-security. The notion of distinguishing-H is useful when we want to check which cryptographic hash function is embedded in the MAC oracle.

- **Distinguishing-R:** It should be impossible to distinguish $\mathtt{MAC}$ (e.g., HMAC/NMAC) from a random function. Explicitly, the goal of the adversary is to distinguish between two cases: let $\mathcal{F}_n$ be the set of all n-bit output functions and denote F_K the oracle on which the adversary $\mathcal{A}$ can make queries. In the first case, the oracle is instantiated with $\mathtt{MAC}_K$ (with K being a randomly chosen k-bit key), i.e. $F_K = \mathtt{MAC}_K$; In the second case, the oracle is instantiated by randomly choosing a function from $\mathcal{F}_n$, i.e. $F_K = R_K$. The adversary's advantage is defined as

$$Adv(\mathcal{A}) = \left|\Pr\left[\mathcal{A}(\mathtt{MAC}_K) = 1\right] - \Pr\left[\mathcal{A}(R_K) = 1\right]\right|.$$

  The expected security of HMAC/NMAC against distinguishing-R attacks is $2^{l/2}$ computations, essentially because of the collision-based forgery attack.

- **Distinguishing-H:** It should be impossible to distinguish the hash-based MAC construction $\mathtt{MAC}$ instantiated with existing hash functions (e.g., HMAC-SHA1) from the same construction $\mathtt{MAC}$ built with a random function (e.g., HMAC-PRF). Explicitly, the adversary is given access to an oracle $\mathtt{MAC}_K$, the goal of the adversary is to distinguish between two cases: In the first case, the oracle is instantiated with a known dedicated compression function h, e.g. $\mathtt{HMAC}^h_K$; In the second case, the oracle is instantiated with a randomly chosen function r from $\mathcal{F}_l^{b+l}$ which is the set of all $(b+l)$-bit to l-bit output functions, e.g. $\mathtt{HMAC}^r_K$. The adversary's advantage is defined as

$$Adv(\mathcal{A}) = \left|\Pr\left[\mathcal{A}(\mathtt{MAC}^h_K) = 1\right] - \Pr\left[\mathcal{A}(\mathtt{MAC}^r_K) = 1\right]\right|.$$

  The expected security of HMAC/NMAC against distinguishing-H attacks is 2^l computations.

There are also related key attacks against hash-based MACs.

- **Related key attacks:** In this model, the adversary has the power to force the user of a MAC algorithm to use two keys with a specific relation (e.g., with a particular XOR difference as in [PSW12]).

Security proofs of hash-based MACs are usually up to the birthday bound. These bounds are known to be tight for some security notions, such as the above mentioned existential forgery resistance and distinguishing-R resistance, because of the internal-collision-based attacks. However, for other types of security notions, there remain gaps between the provable lower bound and the known upper bound.

2.3 Hash Combiners

Among various approaches to designing a hash function, one approach to achieve more tolerant and more compatible hash constructions is to use the so-called hash combiner. That is, combining two (or more) existing hash functions in such a way that the resulting function could provide security amplification, i.e., it is more secure than its underlying hash functions; Or, it could provide security robustness, i.e., it is secure as long as any one of its underlying hash functions is secure. Hash combiners are deployed in widely used Internet security protocols, e.g., SSLv3 ($\texttt{MD5}\|\texttt{SHA1}$) [FKK11], TLS 1.0/1.1 ($\texttt{MD5} \oplus \texttt{SHA1}$) [AD99, DR06], or cryptosystems. They acted to gain confidence on the security of the design although they had not received a thorough analysis.

Two classical hash combiners are the *concatenation combiner* $(\mathcal{H}_1(IV_1, M)\|\mathcal{H}_2(IV_2, M))$ and the *XOR combiner* $(\mathcal{H}_1(IV_1, M)\oplus\mathcal{H}_2(IV_2, M))$. Assume we already have two unrelated n-bit hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$, and we want to get one hash function which could provide security amplification or security robustness by combining them. The concatenation combiner simply feeds the same sequence of message blocks into the two hash functions, computes the two n-bit digests in parallel and outputs their concatenation which is of $2n$ bits (see Fig. 4 L.H.S). The concatenation combiner is expected to behave like an ideal $2n$-bit hash function — though it is usually considered in settings where this is far from being true, e.g., with MD-hashes, as shown in Sect. 5. It is robust with respect to collision resistance since any collision on the combiner can be traced back to collisions on both underlying functions. Its disadvantage is its poor suitability for practical applications where the length of the output is a crucial parameter, because it increases the length of the output of the hash function from n to $2n$ bits. Similar to the concatenation combiner, the XOR combiner also computes two n-bit digests of the same message under the two hash functions in parallel, but it outputs their XOR-sum which remains to be n-bit (see Fig. 4 R.H.S). The XOR combiner is expected to behave like an ideal n-bit hash function. The XOR combiner is robust with respect to PRF and MAC in the black-box reduction model [Leh10]. However, it is not robust with respect to collision resistance, e.g., $\mathcal{H}_2(M) = \mathcal{H}_1(M)\oplus\texttt{const}$. There are impossibility results about efficiently combining collision-resistant hash functions according to research ([BB06, Pie07, Pie08]). It was shown that the output of a (black-box) collision-resistant combiner could not be shorter than the concatenation of both outputs. Advanced security amplification combiners and robust multi-property combiners for hash functions have been constructed [FL07, FL08, FLP08, FLP14]. More generally[1], 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. Well-known examples are Hash Twice: $\mathcal{H}_2(\mathcal{H}_1(IV, M), M)$ and Zipper Hash [Lis06]: $\mathcal{H}_2(\mathcal{H}_1(IV, M), \overleftarrow{M})$, where $\overleftarrow{M}$ is the reversed (block) order of the original message M (see Fig. 5). We can also regard these cascade constructions of hash functions as hash combiners. It turns out that techniques and toolbox used in generic attacks against cascade combiners have large overlaps with that used in attacks on parallel combiners (and those used in generic attacks against hash-based MACs), particular for the applications of the functional graph.

2.4 A Toolbox for Generic Attacks on Hash Constructions

In this section, we exhibit innovation tools widely used in generic attacks on narrow-pipe iterated hash constructions.

[1] Here we need to generalize the syntax of hash functions such that the initial value is also regarded as an input parameter.

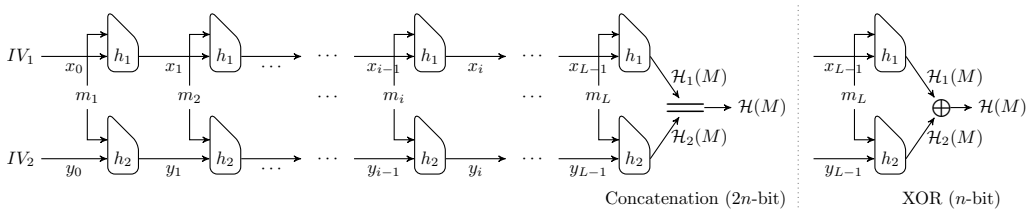


Figure 4: The concatenation combiner and the XOR combiner

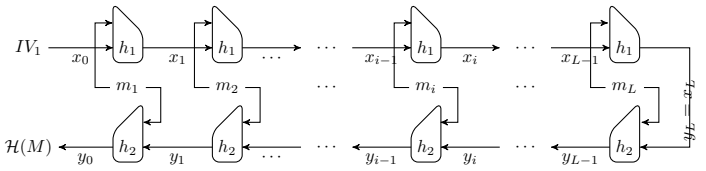


Figure 5: The Zipper hash

2.4.1 Multi-Collision (MC [Jou04]) and Attacks on Concatenation Combiner

In a seminal paper published in 2004 [Jou04], Joux proposes a highly influential tool known as multi-collision. This tool is applicable for narrow-pipe iterated hash functions. By iteratively generating t collisions with the n-bit state compression function, one can get 2^t messages all of which maps a starting state x_0 to a common state x_t (see Fig. 6). Interestingly, the complexity is of $t \cdot 2^{n/2}$ compression function computations which is not much greater than that of finding a single collision. Besides, the whole set of 2^t messages can be described by only $2t$ message blocks: $\mathcal{M}_{\texttt{MC}} = (m_1, m_1') \times (m_2, m_2') \times \cdots \times (m_t, m_t')$. To enumerate messages in this structure, one enumerates an integer from 0 to $2^t - 1$ and observes its binary representation. For $1 \leq i \leq t$, if the i-th bit is '0', then select m_i to be the i-th message block, otherwise select m_i'.

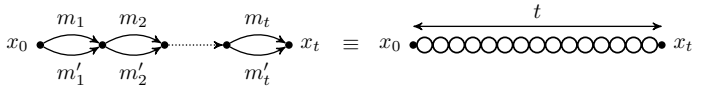
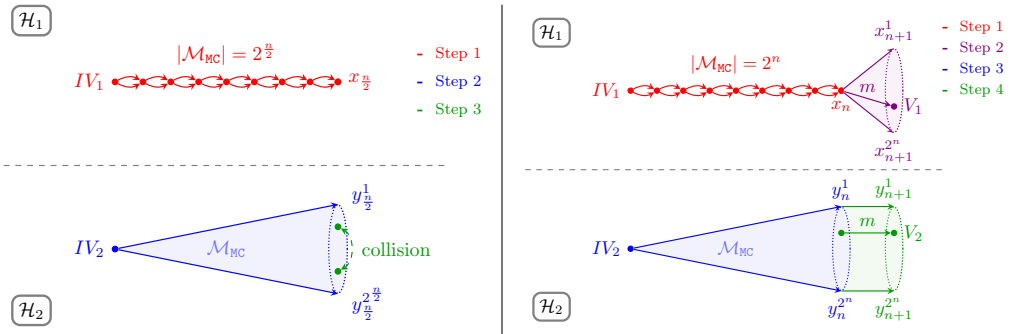


Figure 6: Multi-collision and its condensed representation in R.H.S. [JN15]

With multi-collision at hand, one can directly launch a collision attack with $n \cdot 2^{n/2}$ computations and a preimage attack with $n \cdot 2^n$ computations against concatenation combiner. That is done by building a multi-collision for one of the iterated underlying hash functions, then exploit messages in the multi-collision to deploy a birthday attack for the other underlying hash function (see Fig.7a and 7b). Note that, these attacks works as long as one of the underlying function is narrow-pipe iterated hash, and they show that in this case, the collision resistance and preimage resistance of the concatenation combiner is not much greater than that of a single hash function.

2.4.2 Expandable Message (EM [KS05]) and the Long Message Second-Preimage Attack on MD Hash

For iterated hash functions without Damgård-Merkle strengthening, there is a generic way known two decades ago to violate the second-preimage resistance: Given a target message $M = m_1 \| \ldots \| m_L$ of L blocks, the attacker first computes the sequence of internal states $IV = a_0, a_1, \ldots, a_L$ in processing M. He then starts from IV and evaluates the compression function with arbitrary message blocks until he finds a collision $h(IV, m') = a_p$



(a) Collision attack **(b)** Preimage attack

Figure 7: Attacks against concatenation combiner using multi-collision

for some message block m' and index $p \in \{1, \ldots, L\}$. He finally appends the message suffix $m_{p+1} \| \cdots \| m_L$ to m' to form a second preimage. The expected number of computations is $2^n/L$, because each attempt on m' succeeds with probability $L/2^n$. For MD hash functions, this attack is foiled by the length padding in the last block.

In [Dea99], Dean discovers that one can bypass this defense and carry out a second preimage on MD hash using fixed points in the compression function. Evaluating the compression function on the fixed point various times results in unchanged state and messages of various lengths, which helps correct the length of the crafted message. See Fig.9a for an illustration of Dean's attack which combines the fixed points with the above long message second-preimage attack. The required assumption of Dean's attack is that it is easy to find fixed points in the compression function. However, if the compression functions is not of Davies-Meyer construction, finding fixed points is usually difficult, especially when we assume they are random mappings.

Kelsey and Schneier invent another kind of multi-collision known as the *expandable message* [KS05]. Similar to building a Joux's multi-collision, by iteratively generating t collisions, but with message fragments of carefully chosen length, one can get 2^t colliding messages whose lengths cover the whole range of $[t, t + 2^t - 1]$ (see Fig. 8). The complexity is of $2^t + t \cdot 2^{n/2}$ computations, which is also not much greater than that of finding a single collision. And similar to Joux's multi-collision, the whole set of 2^t messages can be described by only $2t$ message blocks: $\mathcal{M}_{\text{EM}} = (m_1, [0]^{2^0} \| m'_1) \times (m_2, [0]^{2^1} \| m'_2) \times \cdots \times (m_t, [0]^{2^{t-1}} \| m'_t)$. To choose a message of length $s \in [t, t + 2^t - 1]$, one observes the binary representation of $s' = s - t$. For $1 \le i \le t$, if the i-th bit is '0', then select m_i to be the i-th message fragment, otherwise select $[0]^{2^i} \| m'_i$. This expandable message makes the above long message second-preimage attack work as well on narrow-pipe MD hash without assumption on the weakness of the compression function (see Fig. 9b).

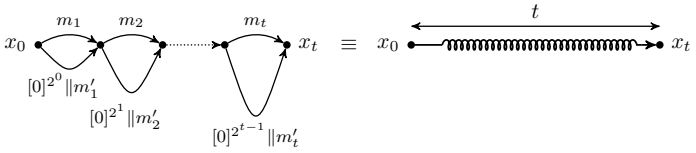
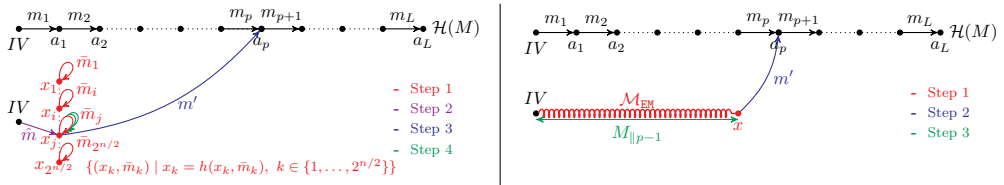


Figure 8: Expandable message and its condensed representation in R.H.S. [JN15]



(a) Using Fixed Points [Dea99] **(b)** Using Kelsey and Schneier's EM [KS05]

Figure 9: The long message second-preimage attack on MD hash functions

2.4.3 Diamond Structure (DS [KK06]) and the Second-Preimage Attack on Hash-Twice [ABDK09]

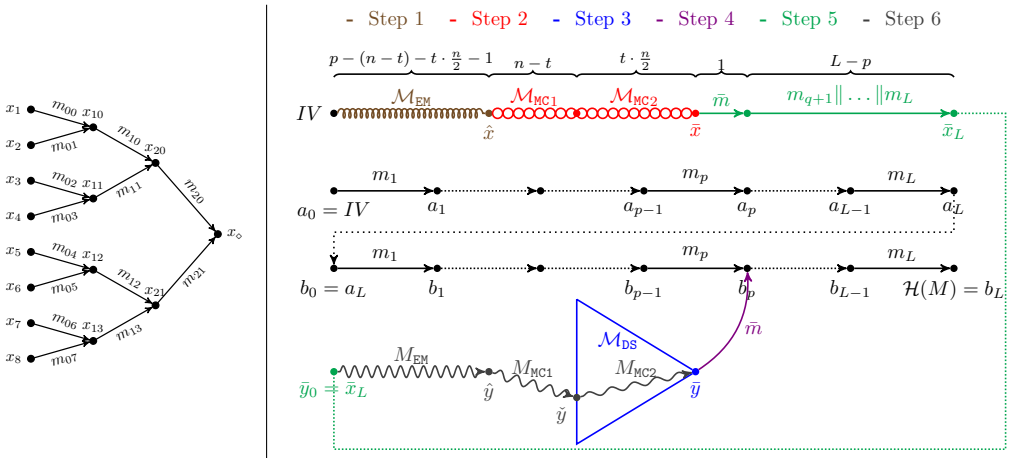
Kelsey and Kohno invent the diamond structure to devise herding attacks against hash functions. In a herding attack, the adversary privately constructs a structure of many collisions on the hash function. He first commits to the digest value of a message in public, and later "herd" any given prefix of a message to that committed digest value by choosing an appropriate suffix from the precomputed structure. This structure of collisions is named a diamond. Similar to multi-collisions and the expandable message, diamond is also a kind of multi-collision built by launching several collision attacks. The difference is that, instead of mapping a single starting state to a final state, a 2^t-diamond maps a set of 2^t starting states (denoted by $\{x_1, x_2, \ldots, x_{2^t}\}$) to a common final state (denoted by $x_\diamond$). Specifically, it contains a complete binary tree of depth t in which those 2^t starting states are leaves, and the common final state is the root (see Fig. 10a). For any starting state x_i, we can directly pick a message from the diamond $\mathcal{M}_{\text{DS}}$ mapping the state to the final state $x_\diamond$, i.e., for any x_i, there exists an $M_i \in \mathcal{M}_{\text{DS}}$ such that $x_i \xrightarrow{M_i} x_\diamond$. Building a 2^t-diamond structure using Kelsey and Kohno's primary approach [KK06] was proved to require $\sqrt{t} \cdot 2^{\frac{(n+t)}{2}}$ messages and $n \cdot \sqrt{t} \cdot 2^{\frac{(n+t)}{2}}$ computations [BSU12]. In [KK13], authors propose a new and also more intricate method with message complexity being $O(2^{n+t})$.

Although the diamond structure is originally used in the herding attacks on hash functions [KK06], it is actually a quite general tool. In [ABF+08, ABDK09], Andreeva *et al.* exploit the diamond structure to develop a generic second-preimage attacks, combining with the techniques in the generic second-preimage attack in [Dea99, KS05]. The general idea of their second-preimage attack on Hash-Twice is to use a long standard multi-collision in the first pass to build a diamond structure in the second pass. And use the diamond structure to connect some internal state computed for crafted messages to one of the internal states computed for the target message (see Fig. 10b). The complexity of the resulted attack is about $2^{(n+t)/2} + 2^{n-l} + 2^{n-t}$, where 2^t is the width of the diamond structure and 2^l is the length of the target message (in block).

With multi-collisions, the expandable message and the diamond structure at hand, our community refit these tools to more advanced ones (such as the following interchange structure, the simultaneous expandable message, and the diamond filter) that can assist evaluating the vulnerability of enhanced hash constructions.

2.4.4 Interchange Structure (IS [LW15]) and the Preimage Attack on XOR combiner

To devise a preimage attack faster than 2^n on the XOR combiner $(\mathcal{H}_1(IV_1, M) \oplus \mathcal{H}_2(IV_2, M))$, Leurent and Wang invent an advanced tool named interchange structure. This structure breaks the pairwise dependency between the internal states of the two underlying hash computations $\mathcal{H}_1$ and $\mathcal{H}_2$. The internal states of $\mathcal{H}_1$ and $\mathcal{H}_2$ are dependent because the computations share the same input message. By building an interchange structure, one creates a set of messages $\mathcal{M}_{\text{IS}}$ and two sets of states $\mathcal{A}$ and $\mathcal{B}$, such that for any pair of



(a) A 2^3-diamond **(b)** Second-preimage attack on Hash-Twice using diamond structure

Figure 10: Diamond structure and the second-preimage attack on Hash-Twice

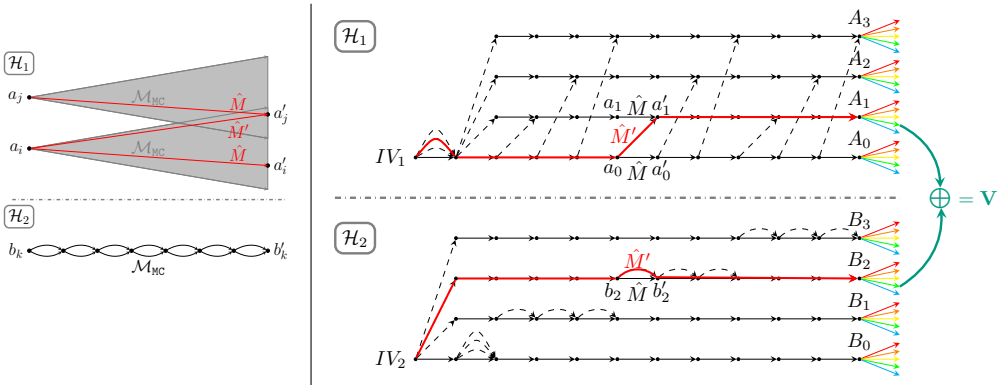
states $(A_i, B_j \mid A_i \in \mathcal{A},\ B_j \in \mathcal{B})$, one can directly pick a message M from $\mathcal{M}_{\text{IS}}$ such that $A_i = \mathcal{H}_1(IV_1, M)$ and $B_i = \mathcal{H}_2(IV_2, M)$. The interchange structure might be reminiscent of a complete bipartite graph (biclique) in the mathematical field of graph theory. The effect brought by the interchange structure is indeed like a biclique, that is, each state in $\mathcal{A}$ is related by a unique message to each state in $\mathcal{B}$, thus any state in $\mathcal{A}$ and any state in $\mathcal{B}$ can be made a pair. However, it is constructed and pictured quite differently from a biclique. Details are as follows.

We refer to an interchange structure with 2^t states for each hash function as a 2^t-interchange structure. To build a 2^t-interchange structure, one need to cascade $2^{2t} - 1$ building modules named switches (a slightly optimized version requires $(2^t - 1)^2$ switches). Each switch contains one pair of message fragment $(\hat{M}, \hat{M}')$, and three pairs of states, two (denote as (a_i, a_i') and (a_j, a_j')) in the computation chain of one hash function (say $\mathcal{H}_1$), and one (denote as (b_k, b_k')) in the computation chain of the other hash function (say $\mathcal{H}_2$). The effect of a switch is that $h_1^*(a_i, \hat{M}) = a_i'$ and $h_1^*(a_i, \hat{M}') = h_1^*(a_j, \hat{M}) = a_j'$ and at the same time $h_2^*(b_k, \hat{M}) = h_2^*(b_k, \hat{M}') = b_k'$, i.e., a state in one computation chain of a hash function can make pair with two states in two computation chains of another hash function. Such a switch can be built using multi-collisions and the birthday attack (see Fig. 11a). By carefully selecting computation chains to make combinations, the cascading of about 2^{2t} such switches will form the above mentioned desired $\mathcal{M}_{\text{IS}}$, $\mathcal{A}$ and $\mathcal{B}$. The total complexities to build a 2^t-interchange structure is of $\tilde{O}(2^{2t+n/2})$ computations.

In the preimage attack on the XOR combiner, the interchange structure enables a follow-up meet-in-the-middle procedure to find a preimage for a given target image V with optimal complexity $\tilde{O}(2^{5n/6})$ (see Fig. 11b, and refer to [LW15] for more details).

2.4.5 Simultaneous Expandable Message (SEM [Din16])

Based on multi-collision and the expandable message, authors of [JN15, Din16] develop a way to construct a multi-collision structure that is expandable with respect to two hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$, named as *simultaneous expandable message*. The construction of a simultaneous expandable message in [Din16] consists of creating a sequence of basic building modules. Each building module contains two pairs of states (x, y) and (x', y'), and one pair of messages (ms, ml), such that $h_1^*(x, ms) = h_1^*(x, ml) = x'$ and $h_2^*(y, ms) = h_2^*(y, ml) = y'$, where ms is a short message of fixed length C and ml is a long message of length $i > C$. The length C of the short message ms is set as a constant $C \approx n/2 + \log(n)$ due to the



(a) Building a switch

(b) Interchange structure and preimage attack on XOR combiner

Figure 11: Interchange structure and the preimage attack on XOR combiner

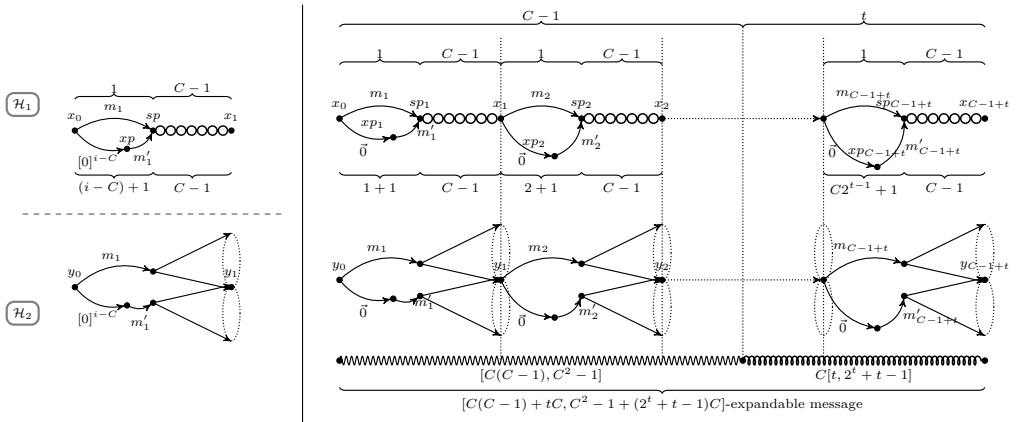
birthday paradox, so that one can successfully construct a building module; The length i of the long message ml is carefully selected, so that the connection of the building modules could form a set of messages of length covering a whole appropriate range. We denote a building module in which the long message is of length i by $\mathcal{M}_{\|i}$. To build $\mathcal{M}_{\|i}$, one can use a procedure similar to the collision attack on concatenation combiner shown in Fig. 7a. The difference is that, the first pair of message blocks in the 2^C-multi-collision is replaced by a pair of message fragments in which one is of single block and the other is of $i - C + 1$ blocks as shown in Fig. 12a.

Given a positive parameter t, by consecutively creating $C - 1 + t$ building modules, one can build the set of multi-collisions whose length covers the whole range of $[C(C-1) + tC, C^2 - 1 + C(2^t + t - 1)]$, i.e., form a whole simultaneous expandable message. The first $C - 1$ building modules $\mathcal{M}_{\|i}$ are built with parameter $i \in \{C+1, C+2, \ldots, C+C-1\}$ so that they form an expandable message of length covering the range $[C(C-1), C^2 - 1]$; The other t building modules $\mathcal{M}_{\|i}$ are built with parameter $i = C(2^{j-1} + 1)$ for $j \in \{1, \ldots, t\}$ which resembles the standard expandable message but with C-block as the unit of length instead of with 1-block (see Fig. 12b). Given a length κ, to pick a message of length κ, we first compute $\kappa' \in [C(C-1), C^2 - 1]$ such that $\kappa' \equiv \kappa \mod C$ to define the first $C - 1$ message fragment choices (select at most one longer message fragment from the first $C - 1$ building modules), and then compute $s = (\kappa - \kappa')/C$ to define the last t message fragment choices (select as in a standard expandable message using the binary representation of s from the last t building modules). This is like using "C" digits $0 \sim C - 1$ instead of using the decimal digits $0 \sim 9$ to represent numbers.

The full construction requires about $n \cdot 2^t + n^2 \cdot 2^{n/2}$ computations and $(n + t) \cdot 2^{n/2}$ message blocks to build a simultaneous expandable message to extend up to length $n \cdot 2^t + n^2$.

In [BWGG17], authors fine-tune the simultaneous expandable message to adapt to Zipper hash. The constructed simultaneous expandable message is placed in the middle of the two passes of the combiner. That is, at the end of the first hash and at the beginning of the second hash. Different from the original parallel construction, this modified one is constructed in a sequential order, because one has to finish the entire building process in the first pass before starting the collision finding process in the second pass (see Fig. 13). Complexity for building simultaneous expandable message for Zipper hash equals to that for parallel combiners.

As shown in Sect. 5, simultaneous expandable message is applied to launch preimage and second-preimage attack on various hash combiners [JN15, Din16, BWGG17].



(a) A building module **(b)** The full construction

Figure 12: Simultaneous expandable message for parallel combiners

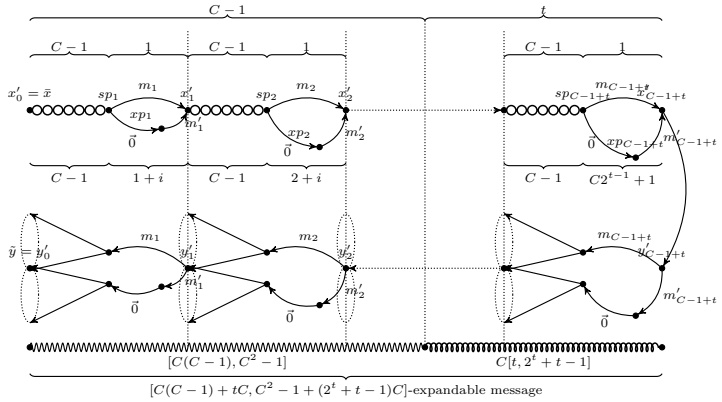


Figure 13: Simultaneous expandable message for cascade combiners

2.4.6 Filters

In recent generic attacks on hash-based MACs, the attacker usually tries to recover some internal state computed by the oracle on his queries. To do that, he usually first computes a set of states by himself. He then manages to make a match between these known states computed by himself and those unknown states obtained from the oracle. A filter usually plays a role in recognizing a match when the attacker does not control a secret suffix (e.g., as in HMAC). Specifically, suppose we have a known state a, and we know that the pair of message blocks (m, m') leads the mapping on a to a collision, i.e., $h(a, m) = h(a, m')$. Suppose u is the unknown state computed by the MAC oracle on a message prefix $\hat{M}$. To test whether u equals a, we simply query the MAC oracle with two crafted messages $M = \hat{M}\|m$ and $M' = \hat{M}\|m'$ and observe the outputs (the computation of the MAC oracle will involve $h(u, m)$ and $h(u, m')$). If the two outputs collide, then this unknown state u equals a with a high probability (compared with the probability 2^{-n} if $u \neq a$, where $n < l$ and n is the bit-size of the output, l is the bit-size of the internal state). This pair of message blocks (m, m') is called a *collision filter* for state a, introduced in [PSW12]. One can build a collision filter by birthday attack with $2^{l/2}$ computations.

In some attacks, to find a match, one has to build filters for a large number of states. Separately constructing filters for each state would be costly. To solve this problem, [DL14] presents a *diamond filter*. To build filters for 2^t states $\{x_1, \ldots, x_{2^t}\}$, one first herds these

states to a single state $x_\diamond$ by building a diamond (recall Sect. 2.4.3). Then, one builds a collision filter (m, m') for this single state $x_\diamond$, i.e. find a pair of message blocks such that $h(x_\diamond, m) = h(x_\diamond, m')$. After that, for any state $x_i \in \{x_1, \ldots, x_{2^t}\}$, one can directly get a collision filter. That is done by concatenating the message fragment $\hat{M}_i$ mapping x_i to $x_\diamond$ in the diamond with each message block in the collision filter (m, m') for $x_\diamond$, i.e. $(\hat{M}_i \| m,\ \hat{M}_i \| m')$. The complexity for building a diamond filter for 2^t states is dominated by the complexity of building the diamond, which is about $2^{(l+t)/2}$. The average complexity for building a filter for a state decreases to $2^{(l-t)/2}$.

It is also possible to build on-line filters for unknown state by querying the MAC oracle many times and observe collisions on the outputs [DL14]. When building off-line filter (for the known state computed off-line), matching between the known state and the unknown state is done on-line (i.e., by querying the MAC oracle as shown above). While, when building on-line filter (for the unknown state by querying the MAC oracle), matching between the unknown state and the known state is done off-line (i.e., by evaluating the compression function with the known state and the filter without querying). Thus, building on-line filters cost more than building off-line filters, but matching using on-line filters is more efficient than matching using off-line filters.

3 Functional Graph of Random Mappings

In this section, we introduce and summarize important properties of the central subject — the functional graph (FG) of a random mapping, whose properties are profoundly exploited by the series of attacks presented in the following sections.

Denote $\mathcal{F}_N$ the set of all mappings from a finite N-set into itself. Let f be an arbitrary mapping in $\mathcal{F}_N$. The functional graph of f (denoted by $\mathcal{FG}_f$) 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 $x \in \{0, \ldots, N-1\}$. It is constructed by successively iterating on f. Starting the development by taking an arbitrary node x_0 as input and taking the output as the input of next iteration on f, that is, $x_1 = f(x_0), x_2 = f(x_1), \ldots$, before N iterations, a value x_j will equal to one of $x_0, x_1, \ldots, x_{j-1}$ (because f is a mapping), suppose the one is x_i. We say x_i is an α-node through which the path $x_0 \to x_1 \to \cdots \to x_{i-1} \to x_i$ enters a *cycle* $x_i \to x_{i+1} \to \cdots \to x_{j-1} \to x_i$. If we consider all possible starting nodes, paths exhibit confluence and form into trees; trees grafted on cycles form components; a collection of components forms a functional graph [FO89]. In the functional graph, we call those nodes that belong to a cycle the cyclic nodes; those nodes without predecessors (preimages, i.e. $f^{-1}(x) = \emptyset$) the terminal nodes; those nodes with at least one predecessor (preimage) the image nodes. Seen from an arbitrary node x_0, we call the length of the path (measured by the number of edges) starting from x_0 and before entering a cycle the tail length of x_0 and denote this by $\lambda(x_0)$; we call the length of the cycle connected with x_0 (measured by the number of edges or nodes) the cycle length of x_0 and denote this by $\mu(x_0)$; we call the length of the non repeating trajectory of the node x_0 the rho-length of x_0 and denote this by $\rho(x_0) = \lambda(x_0) + \mu(x_0)$. Fig.14 is an illustration of the functional graph of a chopped AES-128 (obtained by fixing an arbitrary key and 122 bits of the input and taking 6 bits as output, in this case $N = 2^6 = 64$).

The structure of the functional graph of random mappings has been studied for a long time. Lots of parameters have accurate asymptotic evaluation, which are summarized in the following subsections. A remarkable method achieving these results is analytic combinatorics (the use of the symbolic method, generating functions and singularity analysis) [FO89, FS09]. These results on statistical properties of the random functional graph can provide various knowledge about the iteration of a random mapping, e.g., the expected number of iterations before one encounters a collision starting from a random node; a quantitative evaluation on the lost entropy of the output when iterating a random

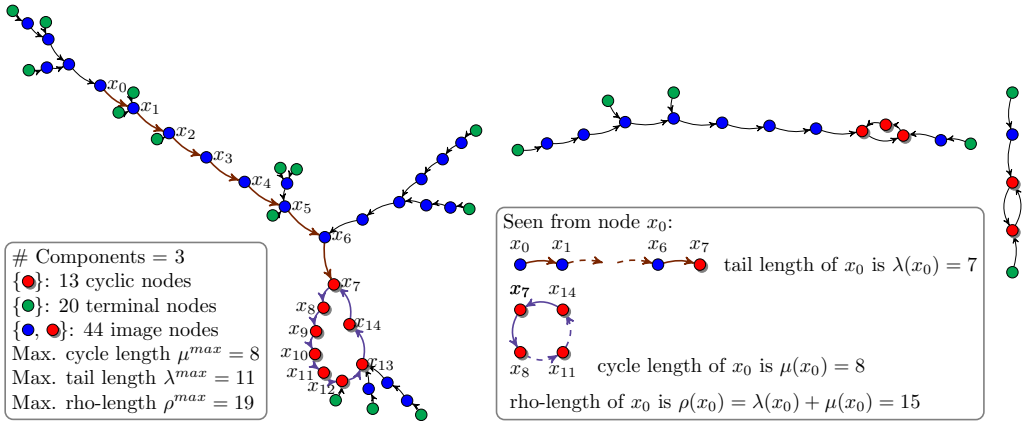


Figure 14: An illustration of the functional graph of chopped AES-128

function many times (discussed in detail in Sect. 6). These results have stimulated fruitful results on the cryptanalysis of iterated hash constructions, as shown in the following sections.

3.1 Known Results on Parameters of Functional Graph [FO89]

Definition 1 (k-th iterate image node)**.** A k-th iterate image node in the functional graph of a random mapping $f \in \mathcal{F}_N$ is an image of the k-th iterate f^k of f. We sometimes simply call it a k-th iterate, and say k is its depth.

Theorem 1 (Direct Parameters [FO89])**.** *The expectations of parameters, number of components, number of cyclic nodes, number of terminal nodes, number of image nodes, and number of k-th iterate image nodes in a random mapping of size N have the 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 the τ_k satisfies the recurrence* $\tau_0 = 0$, $\tau_{k+1} = e^{-1+\tau_k}$.

Note that, the conclusion on the expected number of k-th iterate image nodes implies an important observation on the entropy loss in the output of the iteration of a random mapping, as discussed in detail in Sect. 6.1.2. It essentially facilitates several interesting attacks, e.g., the chain-based attacks on the hash-based MACs in Sect. 4.3 and the deep-iterates-based attacks on hash combiners in Sect. 5.1.

Theorem 2 (Cumulative Parameter Estimates [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 expectations of parameters tail length, cycle length, rho-length, tree size, component size, and predecessors size have the following asymptotic forms, where the tail length, cycle length and rho-length of a node x are defined at the beginning of Sect. 3; the tree size parameter of node x means the size of the maximal tree (rooted on a cycle) containing x; the component size means the size of the connected component that contains x; the predecessors size of x is the size of the tree rooted at x or equivalently the number of iterated preimages of x:*

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) $\sqrt{\pi N/2} \approx 1.2 \cdot 2^{n/2}$
4. *Tree size* $N/3 \approx 0.34 \cdot 2^n$
5. *Component size* $2N/3 \approx 0.67 \cdot 2^n$
6. *Predecessors size* $\sqrt{\pi N/8} \approx 0.62 \cdot 2^{n/2}$

Theorem 3 (Probability Distributions: r-configurations [FO89])**.** *For any fixed integer r, the parameters number of r-nodes, number of predecessor trees of size r, number of cycles trees of size r and number of components of size r, have the following asymptotic* ***mean*** *values, where an r-node is a node of in-degree r; a cycle tree is a tree rooted on a cycle; a predecessor tree is an arbitrary tree in the functional graph:*

1. *r-nodes:* $N \cdot e^{-1}/r!$
2. *r-predecessor trees:* $N \cdot t_r e^{-1}/r!$
3. *r-cycle trees:* $\sqrt{\pi N/2} \cdot t_r e^{-1}/r!$
4. *r-cycles:* $1/r$
5. *r-components:* $c_r e^{-r}/r!$

where t_r is the number of trees having r nodes, $t_r = r^{r-1}$, and $c_r = r![z^r]c(z)$ is the number of connected mappings of size r, $c(z)$ is the generating function of the connected mappings and $[z^r]c(z)$ denotes the coefficient of z^r in the expansion of $c(z)$.

Theorem 4 (Extremal Parameters: Longest Paths [FO89])**.** *The expectation of the maximum cycle length (μ^{max}), maximum tail length (λ^{max}) and maximum rho-length (ρ^{max}) in the functional graph of random mappings of $\mathcal{F}_N$ respectively satisfies:*

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 5 (Extremal Parameters: Largest Configurations [FO89])**.** *Assuming the smoothness condition*[2]*, 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:* $0.75782 \cdot 2^n$.

Results from these theorems indicate that, in a random mapping, most of the nodes tend to be grouped together in a single giant component. This component might therefore be expected to have very tall trees and a large cycle. With these knowledge, one can ensure that by running the cycle search algorithm several times, the cycle length of the giant component in the functional graph could be detected (as discussed in Sect. 6.1.1). There are various applications of this observation, e.g., those cycle-based attacks on hash-based MACs in Sec.4.1, and those multi-cycles-based (second) preimage attacks on hash combiners in Sec.5.2.

3.2 Height Distribution and the λ-th Stratum

The tail length $\lambda(x)$ of a node x is also called the height (or altitude) of the node. Recall that it is the length of the unique path connecting the node with a cycle node. The set of all nodes with height λ is usually called the λ-th stratum of the random graph. Let W_λ be the number of nodes with height λ, i.e., the size of λ-th stratum of the functional

[2]Let ξ^{max} be one of the parameters of random mappings, largest tree size or largest component size. We shall say that the parameter is smooth if the following condition is satisfied: There exists δ such that $\delta = \lim_{n\to\infty} \frac{1}{n}\mathbf{E}\{\xi^{max} \mid \mathcal{F}_N\}$. The reason why introduce it here is to bypass some intrinsic difficulty in the singular behaviour of truncated Taylor series, which involved in the generating function equations associated to the largest tree and the largest connected component.

graph of random mapping. Proskurin [Pro74] provided the distribution of W_λ of a random mapping, and Mutafchiev [Mut88] provided an explicit estimation of the mean value of W_λ as follows.

Theorem 6 ([Mut88], Lemma 2)**.** *Let $\mu(N,\lambda)$ be the mean value of W_λ over all functional graph of random mappings. Then*

$$\mu(N,\lambda) = \sqrt{\pi N/2}\cdot(1+o(1)) \text{ if } N\to\infty \text{ and } \lambda = o(N^{1/2})$$

Corollary 1 ([PW14], Corollary 1)**.** *Let W'_λ be the number of nodes in the λ-th stratum in the giant component of random functional graph. Let $\mu'(N,\lambda)$ be the mean value of W'_λ over all functional graph of random mappings. Then*

$$\mu'(N,\lambda) = 0.64\sqrt{\pi N/2} = \sqrt{\pi N/2} - \sqrt{(\pi N\cdot 0.2418)/2} \text{ if } N\to\infty \text{ and } \lambda = o(N^{1/2})$$

Corollary 2 ([PW14], Corollary 2)**.** *Let W'_λ and $\mu'(N,\lambda)$ be the same as defined in Corollary 1. Let $\delta(n)$ be any function such that $\delta(n)\to\infty$ as $n\to\infty$. Then there exists a positive value n_0* s.t.

$$\mu'(N,\lambda) = 0.64\cdot 2^{n/2} \text{ if } n\to\infty \text{ and } 0\le\lambda\le 2^{n/2}/\delta(n)) \text{ for } \forall\ n > n_0.$$

Conjecture 1 ([PW14], Conjecture 1)**.** *With $s\le 2^{n/6}$, there is only a negligible probability that a collision exists among the heights of s random values in a functional graph of an n-bit random mapping.*

Algorithm 1 is used in various attacks, which expands 2^t nodes in the functional graph of a random mapping f and records their heights for a given parameter t.

Conjecture 2 ([GPSW14], Conjecture 1)**.** *If in total 2^t distinct nodes, where $n/2\le t\le n$ holds, are collected following Algorithm 1, then for any integer λ satisfying $1\le\lambda\le 2^{n/2}/n$, there are $\Theta(2^{t-n/2})$ nodes collected with the height value λ.*

Algorithm 1 Generating 2^t nodes and record their heights in $\mathcal{FG}_f$

```
1: procedure GEN(t)
2:     G ← ∅    ▷ a data structure like a graph storing nodes in FG_f (each node contains information including its value, height, predecessors and successor), G is a partial FG.
3:     while |G| < 2^t do
4:         C ← ∅,   y.value ←$ {0,1,...,2^n − 1} \ G,   y.height ← 0
5:         while true do
6:             if y.value ∈ G or y.value ∈ C then
7:                 if y.value ∈ G then
8:                     Get the height value of y from G, denote it by λ. For each predecessor of y in C, update its height to be λ + d, where d is its distance from y.
9:                 else   ▷ y.value ∈ C, the chain connected to a cycle
10:                    Denote by y_i the node colliding with y in C, set height of all nodes from y_i to y as 0. For each predecessor of y_i, update its height to be d, where d is its distance from y_i.
11:                end if
12:                G ←_merge C,   go to line 3
13:            else C ←_insert y,   y ← f(y)
14:            end if
15:        end while
16:    end while
17:    return G
18: end procedure
```

Based on these corollaries and conjectures on height distribution of nodes in the functional graph of a random mapping, generic universal forgery attacks on iterative hash-base MACs are proposed in [PW14, GPSW14], as described in Sect. 4.2.

3.3 Turn the Compression Function Into Random Mapping on Itself

When feeding an arbitrary fixed message block m (of b-bit width) into the compression function h, we turn the compression function which takes $n+b$-bit input into an n to n-bit random mapping. We denote this n-bit random mapping by $h_{[m]}$ and denote the functional graph of $h_{[m]}$ by $\mathcal{FG}_{h_{[m]}}$.

4 Attacks on Hash-based MAC Based on Functional Graph

In this section, we look into generic attacks on hash-based MACs which involve iteratively evaluating a mapping many times, i.e., which relate to properties of the functional graph of random mappings (refer to Fig. 1a for a list of main surveyed papers and their technical relations). Here, we recall the explicit convention about the size of security parameters for MACs: b-bit message blocks, l-bit internal states, n-bit tag.

4.1 Cycle-based Attacks

In [PSW12], Peyrin, Sasaki and Wang create distinguishing attacks, inner state recovery attacks and forgery attacks against HMAC in the related-key setting. These attacks are mainly based on an observation that is the cycle structure of HMAC is the same when instantiated with a key K or with a related key $K' = K \oplus ipad \oplus opad$. More generally speaking, functional graphs of the two mappings $\texttt{HMAC}_K = f_{K_{out}} \circ f_{K_{in}}$ and $\texttt{HMAC}_{K'} = f_{K_{in}} \circ f_{K_{out}}$ have a common structure. This property should not exist for a randomly chosen function embedded with K and K', thus can be used as a distinguisher.

Distinguishing-R [PSW12]. Details about the distinguishing-R attack are as follows: Let F_K and $F_{K'}$ be the two oracles instantiated with two related keys $K' = K \oplus ipad \oplus opad$. They are either $\texttt{HMAC}_K = f_{K_{out}} \circ f_{K_{in}}$ and $\texttt{HMAC}_{K'} = f_{K_{in}} \circ f_{K_{out}}$ or R_K and $R_{K'}$. To distinguish the two cases, the adversary first queries a birthday bound number of adaptive chosen messages to the two oracles, so that he can respectively detect the cycle length L of F_K and L' of $F_{K'}$. Specifically, queries are chosen as follows: He first chooses a random n-bit message M_A (resp. M_B) and initializes the queries $q_0^A = M_A$ (resp. $q_0^B = M_B$). After getting the answer of the previous one, he continues by querying the answer received from the last query to the oracle, i.e. $q_i^A = F_K(q_{i-1}^A)$ (resp. $q_i^B = F_{K'}(q_{i-1}^B)$) for $i > 0$, and repeats this for $2^{n/2} + 2^{n/2-1}$ iterations. When the oracles are instantiated with $\texttt{HMAC}_K = f_{K_{out}} \circ f_{K_{in}}$ and $\texttt{HMAC}_{K'} = f_{K_{in}} \circ f_{K_{out}}$, then the first half part of the hash computation (inner hashing) on the queries of the adversary together with the second half part of the hash computation (outer hashing) form random walks, denoted by $\texttt{walk}_A$ (resp. $\texttt{walk}_B$), along the chain of alternating computations of $f_{K_{in}}$ and $f_{K_{out}}$ (resp. $f_{K_{out}}$ and $f_{K_{in}}$) as depicted in Fig.15. Thus, if the computation chains loop, the cycle length of the two walks should be equal. Note that, there are efficient algorithms to detect the cycle length [Jou09, Chapter 7] (also refer to Sect. 6.1.1). Accordingly, the adversary outputs 1 if he successfully detects the cycle lengths and the two cycle lengths are equal, and outputs 0 otherwise.

The computational complexity of this distinguishing-R attack is $2^{n/2+1}$ which is less than that of the trivial generic attack based on internal collision, which is $2^{l/2}$, as long as the length of the tag is less than the size of the internal state $n < l$, i.e. wide-pipe underlying hash function. Besides, this distinguishing-R attack can be further extended into an internal state recovery attack, forgery attacks and distinguishing-H attacks with similar requirement of queries and $2^{n/2+2} + 2^{l-n+1}$ computations [PSW12].

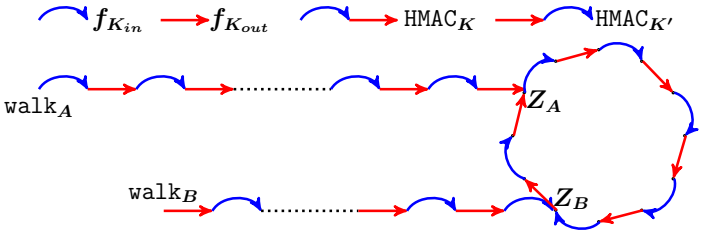


Figure 15: The cycle built with access to $\mathtt{HMAC}_K$ and $\mathtt{HMAC}_{K'}$ [PSW12]

Distinguishing-H [LPW13]. In [LPW13], instead of studying the cycle structure of an entire HMAC, Leurent, Peyrin and Wang study the cycle structure of the internal compression function h with a fixed message block (see Sect. 3.3). In that, the proposed attack works in the single-key model instead of in the related-key model. By off-line identifying the property of a dedicated underlying compression function h, one can distinguish whether an HMAC oracle is embedded with the known dedicated compression function h or with a randomly chosen function r without knowing the secret key, which is exactly the setting of the distinguishing-H attack.

Attacks in [LPW13] are based on the following observations: It is easy to detect the cycle length of the giant component in the functional graph of $h_{[0]}$ (denote by $\mathcal{FG}_{h_{[0]}}$, where [0] is an instance of the fixed message block which can take an arbitrary value, see Sect. 3.3 and Sect. 6.1.1,). Suppose the cycle length is μ, then the node x located in the cycle must satisfy $x = h^{\mu}_{[0]}(x)$. That is, if $x = h(m)$ and x is a cyclic node and μ is the cycle length of $\mathcal{FG}_{h_{[0]}}$, then $h^{\mu+1}(m\|[0]^{\mu}) = h(m)$. In addition, it is possible to detect such a cycle on-line if the oracle is embedded with the known h. Specifically, to detect such a cycle, we craft two related but distinct messages which satisfy that such a cycle happens with high probability in both of the computations of the two messages, and if that happens, then the outputs of these two messages collide. Thus, we observe the outputs of the oracle on these two carefully crafted messages to determine whether the oracle is indeed embedded with h. The distinguishing-H attack based on these observations is sketched as follows (see Fig.16):

- **Off-line:** Detect the cycle length μ of the giant component in $\mathcal{FG}_{h_{[0]}}$ by running the cycle search algorithm several times (explained in Sect. 6.1.1).
- **On-line:** Randomly select an initial message block m, query the oracle with two long messages $M_1 = m\|[0]^{l/2}\|[1]\|[0]^{l/2+\mu}$ and $M_2 = m\|[0]^{l/2+\mu}\|[1]\|[0]^{l/2}$.
- **Output:** If the tags of the two messages collide, output 1. Otherwise, output 0.

The complexity of this distinguishing-H attack is: $\mathcal{O}(2^{l/2})$ off-line computations and 2 on-line queries of about $3 \cdot 2^{l/2}$ message blocks. The adversary's advantage is 0.14. Reasons are as follows: If the HMAC oracle is instantiated with the real compression function h, the collision between the two tags of M_1 and M_2 happens when: 1) message block m sets the on-line computations in the giant component of $\mathcal{FG}_{h_{[0]}}$ (happens with probability 0.75782 because the giant component is expected to occupy 0.75782 percent of the whole nodes) and the first message fragment $[0]^{l/2}$ sets the follow-up on-line computations in the cycle (happens with probability 1/2 because the the tail length is expected to be less than $2^{l/2}$), 2) message block [1] and the second message fragment $[0]^{l/2}$ fulfills the same corresponding conditions as in 1). Thus, the collision probability is $(0.75782 \cdot 1/2)^2 \approx 0.14$; If the HMAC oracle is instantiated with a random function, the corresponding collision happens when the last $2^{l/2}$ [0] message blocks leads the two computations to an internal collision (happens with probability $2^{l/2} \cdot 2^{-l} = 2^{-l/2}$). Overall, $Adv(\mathcal{A}) = |0.14 - 2^{-l/2}| \approx 0.14$.

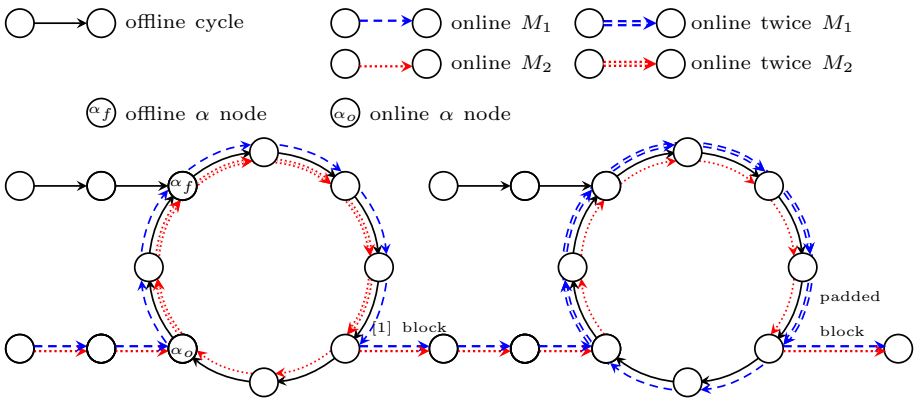


Figure 16: Distinguishing-H attack based on cycle structure [LPW13]

Cycle-based State Recovery Attack [LPW13]. By extending previous cycle-based distinguishing-H attack, it is also possible to launch an internal state recovery attack. The node through which the computation chain enters the first cycle is the targeted recovered state (see α_o in Fig.16). This α-node has a good probability (denote by p_α which is about $0.48/0.75782 \approx 0.63$) to be the root of the largest tree in the giant component of the functional graph. Such root can be computed off-line using several runs of the cycle search algorithm (see Sect. 6.1.1). Besides those techniques used in the previous distinguishing-H attack, an additional technique required is the *binary search*. Binary search can sped-up locating the α-node. The goal is searching from an integer range $[0, 2^{l/2}]$ for the minimum number of blocks [0] in M_1 and M_2 before the two computation chains simultaneously entering the first cycle. The binary search goes as follows ([LPW13, PW14]):

1. Initialize two integer variables X_1 and X_2 to the value 0 and $2^{l/2}$ respectively.
2. Set $X' = (X_1 + X_2)/2$. Query to the MAC oracle with at most $\beta \log(l)$ distinct message pairs $M_1' = m\|[0]^{X'}\|[i]\|[0]^{2^{l/2}+\mu}$ and $M_2' = m\|[0]^{X'+\mu}\|[i]\|[0]^{2^{l/2}}$, where $[i]$ can take arbitrary value different from [0]. If at least one received tag pair collides, set $X_2 = X'$. Otherwise, set $X_1 = X'$.
3. If $X_1 + 1 = X_2$, output X_2 as the minimum block index of α-node. Otherwise, go back to the previous step.

The value of β in Step 2 is set to be 4.5 to ensure the average success probability of this binary search is $p_{bs} = (1 - 1/l)^{l/2} \geq e^{-1/2}$ as explained in [LPW13].

Launching the above binary search after several runs of a procedure like the previous cycle-based distinguish-H attack (run several times to ensure the on-line computations is in the cycle in the giant component of $\mathcal{FG}_{h_{[0]}}$), this performs a state recovery attack. This cycle-based internal state recovery attack has complexity about $13.5 \cdot l \cdot \log(l) \cdot 2^{l/2}$ and has success probability about $p_\alpha \cdot p_{bs} \approx 0.38$ with long messages queries, where $p_\alpha \approx 0.63$ is the probability of α_o being the root of the largest tree in the giant component and $p_{bs} \geq e^{-1/2} \approx 0.6$ is the success probability of the binary search.

Note that, the following presented height-based universal forgery attacks use this cycle-based internal state recovery attack as a sub-procedure. Actually, in this state recovery attack, the minimum number of blocks [0] before a computation chain entering the first cycle, i.e., the minimum block index of α_o from x, is equivalent to the height of x in $\mathcal{FG}_{h_{[0]}}$. This binary search procedure is actually finding the height of x. This binary search is also used in [DL14, DL17] for launching collision-based state recovery attacks (refer Sect. 4.3). The difference is that, here we use the binary search to get the height of a node, and there

we use the binary search to get the offset of a collision. They are essentially identical. Nevertheless, α-nodes are collisions located on cycles.

Short Cycle-based State Recovery Attack [DL17]. To make a trade-off between the message length and the attack complexity, instead of using the *largest cycle* in the giant component in the functional graph, one can use *shorter cycles* as proposed in [DL17]. For messages of length 2^s for $s \leq l/2$, the complexity of the short cycle-based state recovery attack is 2^{2l-3s}.

Besides the state recovery attack, the cycle-based distinguishing-H attack was also directly extended into a selective forgery attack in [GPSW14]. The key is that by querying the MAC oracle with a message $M' = m^{2^{l/2}+\mu} \| m' \| m^{2^{l/2}}$ and getting (M', T), the adversary can output a valid forgery (M, T) where M is the committed message and $M = m^{2^{l/2}} \| m' \| m^{2^{l/2}+\mu}$ and μ is the cycle length in the giant component of $\mathcal{FG}_{h_{[m]}}$. The overall complexity and the adversary's advantage is the same as that of the distinguishing-H attack.

4.2 Height-based Attacks

For the universal forgery, suppose the challenge message given to the adversary is $M = m_1 \| m_2 \| \cdots \| m_L$. Almost all known generic universal forgery attacks first split M into two parts $M_1 \| M_2$: $M_1 = m_1 \| m_2 \| \cdots \| m_{2^s}$, $M_2 = m_{2^s+1} \| m_{2^s+2} \| \cdots \| m_L$. And then carry out the following two phases:

- **Phase 1 [State recovery attack]**: Recover one of the internal states x_i in $X = \{x_1, x_2, \cdots, x_{2^s}\}$, where X is the ordered set of unknown states during the processing of M_1 under the inner hash function.
- **Phase 2 [Second-preimage attack]**: Based on the knowledge of the recovered internal state, launch a classical second-preimage-like attack to find a different message suffix M' colliding with M_2 under the inner hash function . Query the MAC oracle with $M_1 \| M'$ and get the tag T, then output T as the tag of M.

Phase 2 of all known universal forgery attacks are the same (using the long message second-preimage attack on narrow-pipe MD hash functions in [KS05]. Thus, all presented universal forgery attacks are only applicable for narrow-pipe MD hash-based MACs) and the attack complexity of Phase 2 is relatively less than the complexity of Phase 1. Thus, improvements focus on Phase 1.

Note that, Phase 1 is much harder than an internal state recovery attack in which the message is completely chosen by the adversary. In the aforementioned internal state recovery attack, the recovered state is the collision node through which the chain enters a cycle. The cycle length of the functional graph is a key property exploited. The cycle length is essentially an attribute of an entire functional graph and thus can characterize a particular random mapping. However, it can not character any particular node in the functional graph. Thus, it can be exploited in the distinguishing attack, the internal state recovery attack, and the existential forgery attack, in all of which the adversary can choose the message, but hardly be exploited in the universal forgery attack in which the message is given to the adversary as a challenge. To carry out a universal forgery attack, the adversary has to exploit much more powerful properties. Unlike the cycle length, the height of a node contains more information about individual node. Theorem 6 shows that the numbers of nodes in low strata and the number of cyclic nodes of a random functional graph are identically distributed as $n \to \infty$. That finally implies Conjecture 1 which shows that if one collects relatively small number of nodes in a functional graph, one will find the

heights of those nodes are distinct with high probability. Suppose a particular unknown state of the hash computation is known to be a tail node in a functional graph, and one knows its height. Then, one can greatly narrow the search space of nodes computed off-line to find a match so that one can recover the unknown state efficiently. By exploiting the height property, [PW14] presents the first generic universal forgery attack on iterative hash-based MACs, which was further updated in [GPSW14].

Universal Forgery Attack — The First [PW14]. The attack in [PW14] exploits the heights of those internal states in $X = \{x_1, x_2, \ldots, x_{2^s}\}$ in a particular functional graph (size of X is 2^s). Those heights make it efficient when matching between elements of X and elements of an off-line precomputed set $Y = \{y_1, y_2, \ldots, y_{2^{l-s}}\}$ (size of Y is 2^{l-s}, thus Y has an overlap with X with good probability according to the birthday paradox). The height is a characteristic to classify elements of Y into subsets. Thus, matching between X and Y can be speed up by first matching height and then matching between subsets characterized by height. Note that, from Conjecture 1, elements of set X have distinct heights. Additionally, these subsets of Y are all disjoint. Thus, the total number of pairs of elements of X and elements of Y that need to be matched is at most 2^{l-s}, namely the size of Y, instead of the straightforward $2^s \cdot 2^{l-s} = 2^l$. This is exactly where the advantage comes from.

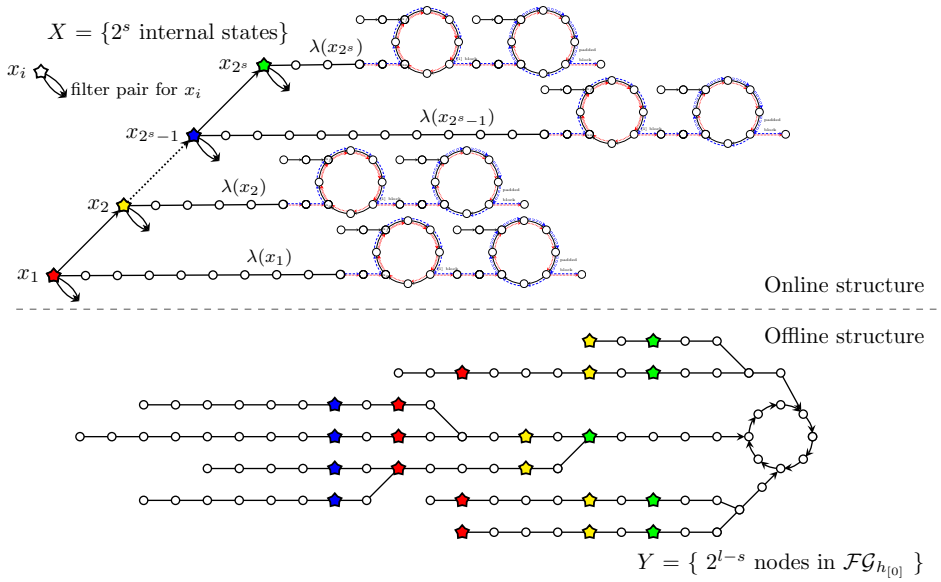
The procedure of the height-based universal forgery attack is sketched as follows (Steps 1–4 compose Phase 1 depicted in Fig. 17, Steps 5–6 compose Phase 2):

- **Step 1 (on-line)**. For each unknown internal states $x_1, x_2, \ldots, x_{2^s}$ in X, use the cycle-based internal state recovery attack in Sect. 4.1 as sub-procedure to recover their height in $\mathcal{FG}_{h_{[0]}}$. Denote by $\lambda(x_i)$ the height of node x_i.
- **Step 2 (on-line)**. Generate filters for each x_i in X. That is, finding a pair of one-block messages (m, m') such that $m_1 \| \ldots \| m_i \| m$ and $m_1 \| \ldots \| m_i \| m'$ leads collision on outputs of the MAC. That is done using birthday attack.
- **Step 3 (off-line)**. Develop 2^{l-s} nodes and record their height in $\mathcal{FG}_{h_{[0]}}$ using Algorithm 1. Store them in a set Y sorted according to the height values.
- **Step 4 (off-line)**. For each x_i in set X, try to recover it by matching it with elements of Y. The matching is divided into two steps. In the first step, match according to the height $\lambda(x_i)$ to get a subset $Y_{\lambda(x_i)}$ of Y in which all elements have the same height as x_i. In the second step, matching elements of $Y_{\lambda(x_i)}$ using the filter pair (m, m') of x_i. Find out the element y_i such that $h(y_i, m) = h(y_i, m')$. As long as we find one match y_t, output the value of y_t as the value of internal state x_t.
- **Step 5 (off-line)**. Based on the knowledge of the recovered internal state x_i, compute the value of x_{2^s} under the inner hash function. Find a second preimage M' for the processing of M_2 using classical second-preimage attack [KS05] (regarding x_{2^s} as the IV). The length of M_2 is $L - 2^s$ blocks.
- **Step 6 (on-line)**. Forge a valid tag for the challenge M by querying message $M_1 \| M'$ to the MAC oracle, and receive its tag T. Output (M, T).

Complexity of each steps in this height-based universal forgery attack is:

Step 1:	$O(l \cdot \log(l) \cdot 2^{s+l/2})$	**Step 2:**	$2^{2s+l/2}$	**Step 3:**	2^{l-s}
Step 4:	2^{l-s}	**Step 5:**	$2^l/(L-2^s)$	**Step 6:**	L

The optimal complexity of this attack for challenges of different length are:



Only match elements in X and elements in Y at same height (same color impling same height).

Figure 17: Phase 1 of height based universal forgery attack

- If $0 \leq L \leq 2^{l/6}$, Steps 3–5 are dominated. We set $2^{l-s} = 2^l/(L-2^s) \Rightarrow 2^s = L - 2^s$ to get an optimal complexity of $O(2^l/L)$ computations.
- If $2^{l/6} < L \leq 2^{5l/6}$, Step 2 is dominated. We set $s = l/6$ to get an optimal complexity of $O(2^{5l/6})$ computations.
- If $L > 2^{5l/6}$, Step 6 is dominated. We set $s = l/6$ to get an optimal complexity of L computations.

Universal Forgery Attack — The Updated [GPSW14]. In the previous presented universal forgery attack, distribution of height values of elements of the off-line precomputed set Y is unclear, because nodes in Y are collected using Algorithm 1 instead of randomly selected. According to observation and experimental verification, [GPSW14] provides a reasonable conjecture which identifies the distribution of height values in Y (refer to Sect. 3.2 Conjecture 2). More explicitly, among those 2^t nodes collected using Algorithm 1, there are $\Theta(2^{t-l/2})$ nodes with height λ for all $\lambda \leq 2^{l/2}/l$. Under this heuristic assumption, the universal forgery attack can be updated as follows (describe only the differences in the sequel):

- **Step 2 (on-line).** Different from the original Step 2 in which one generates filters for each internal states separately, in the updated attack, one only generates a filter (m, m') for the last internal state x_{2^s} in X. Then, using (m, m') one directly build a filter $(m_{i+1}\| \cdots \|m_{2^s}\|m, m_{i+1}\| \cdots \|m_{2^s}\|m')$ for other internal state x_i.
- **Step 3 (off-line).** Different from the original Step 3, one only stores the nodes with height value λ satisfying $\lambda \leq 2^{l/2}/l$. Moreover, according to Conjecture 2, we now know that for each such height λ such that $\lambda \leq 2^{l/2}/l$ holds, there are $2^{l/2-s}$ nodes in Y that have height λ.

Note that, although it is Steps 2 and 3 that are updated, major complexity updates happen to Steps 2 and 4, both of which turn to be $2^{s+l/2}$. New balances between different steps are achieved for messages of different lengths.

- If $0 \leq L \leq 2^{l/4}$, Steps 3 and 5 are dominated. We set $2^{l-s} = 2^l/(L-2^s) \Rightarrow 2^s = L-2^s$ to get an optimal complexity of $O(2^l/L)$ computations.
- If $2^{l/4} < L \leq 2^{3l/4}$, Steps 1–5 are dominated. We set $s = l/4$ to get an optimal complexity of $O(2^{3l/4})$ computations.
- If $L > 2^{3l/4}$, Step 6 is dominated. We set $s = l/4$ to get an optimal complexity of $O(L)$ computations.

4.3 Attacks Based on Entropy Loss of Chain Evaluation and Collision Search

Cycle-based (height-based) attacks have a disadvantage, that is, queries to the MAC oracle are of very long length. However, in practice, many widely deployed hash functions have a limit on the maximal message length (e.g. SHA-1 and SHA-2). Thus, these cycle-based and height-based attacks actually bring no harm to many concrete HMACs instantiated with those length-limited hash functions. Motivated by that, Leurent, Wang and Dinur devised several chain-based and collision-based short message attacks in [LPW13, DL14, DL17].

4.3.1 Collision Search Algorithm

As shown in Algorithm 2, the collision search algorithm evaluates chains up to a fixed maximum length (e.g. 2^s) starting from randomly selected nodes. More specifically, starting from node x_0, the algorithm constructs chains iteratively using equation $x_i = f(x_{i-1})$ until $i = 2^s$. These chains are essentially branches in functional graph of f. Collisions are distinguished as different types. One type is the so-called same-offset collision (generated using procedure SameOffsetCollision in Algorithm 2), another is called the free-offset collision (generated using procedure FreeOffsetCollision in Algorithm 2). For the same-offset collisions, their distance from the two starting nodes are exactly the same. Namely, for the two colliding chains $\vec{x}$ and $\vec{y}$, their colliding point satisfies $x_i = y_j$ and $i = j$. While, for free-offset collisions on two chains $\vec{x}$ and $\vec{y}$, their colliding point satisfies $x_i = y_j$ for any i, j. Essentially, cycles also imply a type of collision. They are collisions on a single chain.

Algorithm 2 Collision finding with chains of length 2^s [LPW13]

```
1:  procedure SameOffsetCollision(s)
2:      T ← ∅,   x ←$ {0,...,2^l − 1}
3:      loop ▷ g_{2^s} = f_{2^s} ∘ ··· ∘ f_1
4:          x ← x + 1,   y ← g_{2^s}(x)
5:          if T[y] = ∅ then T[y] ← x
6:          else
7:              x' ← T[y],   i ← 1
8:              repeat
9:                  x ← f_i(x), x' ← f_i(x'),
10:                 i ← i + 1
11:             until x = x'
12:             return x
13:         end if
14:     end loop
15: end procedure
16: procedure FreeOffsetCollision(s)
17:     T ← ∅,   x ←$ {0,...,2^l − 1}
18:     loop
19:         y_0 ← x,   x ← x + 1,   i ← 0
20:         repeat
21:             if y_i ∈ T then return y_i
22:             else
23:                 T ←_insert y_i,   y_{i+1} ← f(y_i)
24:             end if
25:             i ← i + 1
26:         until i = 2^s
27:     end loop
28: end procedure
```

Based on formal evaluation on the entropy loss in the output of classical collision search algorithm on random functions, Leurent and Wang in [LPW13] and Dinur and Leurent in [DL14, DL17] propose more efficient attacks for short messages.

Same-offset collision search. For the same-offset collision, the numbers of iterations before the colliding point on the two colliding chains are equal. When the message length is appended to messages, only such same-offset internal collisions between equal-length queries to the MAC oracle are detectable. Thus, in known collision-based attacks, collisions detected on-line are always same-offset collisions. Additionally, even if the underlying compression functions are different within each computation chain, the same-offset collision between chains still propagates to the end. Thus, collisions detected in attacks on MACs using different compression functions (e.g. the HAIFA construction [BD07]) are also same-offset collisions. While, in this paper, we mainly focus on computations on identical mapping and see relationship between its property and the properties of its functional graph.

When computing 2^t chains of length 2^s, the expected number of same-offset collisions is about 2^{2t+s-l}. The total number of distinct nodes on those chains should be a constant fraction of the total developed nodes, that is $\Theta(2^{s+t})$. When the iteration function are all the same, the condition reduces to $2^s \cdot 2^t \cdot 2^s \leq 2^l \Rightarrow 2s+t \leq l$. When the iteration functions are distinct, in order to collect the 2^{2t+s-l} same-offset collisions, we require that a constant fraction of the chains do not collide with any other chain, implying that $2t+s-l < t \Rightarrow s+t \leq l$.

To collect 2^c same-offset collisions using chains of length 2^s, the time complexity is determined by parameter l, c and s, i.e., $2^{l/2+s/2+c/2}$. The required number of chains is $2^{l/2-s/2+c/2}$ which is also determined by parameters l, s and c. Thus, if one wants to get more collisions with a limited computational complexity, he has to use shorter chains and large number of chains.

Free-offset collision search. Collecting free-offset collisions is much more efficient than collecting same-offset collisions. However, that procedure is applicable only for off-line computation and for identical underlying compression functions.

When computing 2^t chains of length 2^s and $2^s \cdot 2^{t+s} \leq 2^l$, the expected number of free-offset collisions is about $2^{2(t+s)-l}$. To collect 2^c free-offset collisions using chains of length 2^s, the time complexity is determined by parameter l, c, i.e., $2^{l/2+c/2}$, which is a constant for different s. The required number of chains is $2^{l/2+c/2-s}$ which is determined by parameters l, s and c.

Properties of the chain evaluation and the collision searching algorithms [LPW13, DL14, DL17]

Lemma 1 ([DL17], Lemma 1)**.** *Let $s \leq l/2$ be a non-negative integer. Let $f_1, f_2, \ldots, f_{2^s}$ be a fixed sequence of random functions over the set of 2^l elements, and $g_{2^s} \triangleq f_{2^s} \circ f_{2^{s-1}} \circ \cdots \circ f_2 \circ f_1$ (with the f_i being either all identical, or independently distributed). Then, the images of two arbitrary inputs to g_{2^s} collide with probability of about 2^{s-l}, i.e., $\Pr_{x,y}[g_{2^s}(x) = g_{2^s}(y)] = \Theta(2^{s-l})$.*

Lemma 2 ([DL17], Lemma 2)**.** *Let $s \leq l/2$ be a non-negative integer. Let f be a random function over the set of 2^l elements. Then, for some $\alpha > 1$ the probability that the image size is more than $2\alpha \cdot 2^l/i$ is at most $1/\alpha$, i.e., with high probability the image size of the function f^i is $O(2^l/i)$.*

Lemma 3 ([DL17], Lemma 3)**.** *Let $\hat{x}$ and $\hat{y}$ be two random collisions (same-offset or free-offset) found by a collision search algorithm using 2^t chains of length 2^s, with a fixed l-bit random function f such that $2s+t \leq l$. Then $\Pr[\hat{x} = \hat{y}] = \Theta(2^{2s-l})$.*

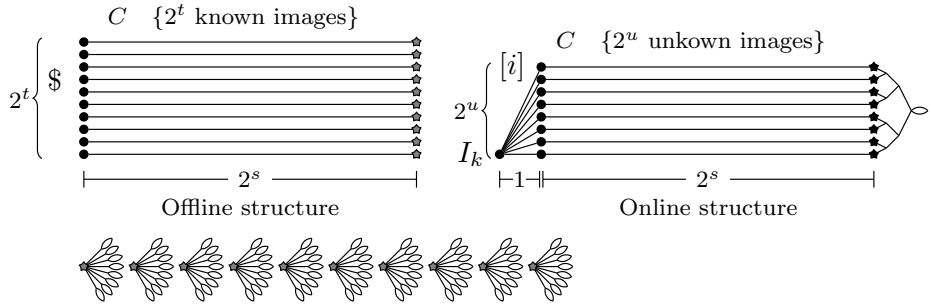
In [DL17], authors also provide similar lemmas on the same-offset collision search with fixed sequence of independently distributed random functions (applicable for attacks on the HAIFA construction). While, this paper mainly targets on presenting the relationship

between properties of the collision search and the functional graph formed by iterating on a single function. Thus, we refer [DL17] to interested readers for detailed descriptions on the collision search with fixed sequence of independently distributed random functions.

4.3.2 State Recovery Attacks — With Short Messages [LPW13, DL14, DL17]

State recovery attack based on reduction of image-set size [DL17]. Based on the entropy loss in the output of the chain evaluation algorithm indicated by Lemmas 1 and 2, one can launch a state recovery attack on hash-based MACs (applicable for both MD and HAIFA hash constructions, see Fig.18).

- **Step 1 (off-line).** Compute a set X of 2^t off-line images by developing 2^t chains of length 2^s using the compression function with fixed message C.
- **Step 2 (on-line).** Collect a set Y of 2^u unknown states (obtained after M_i) by querying the oracle with 2^u messages $M_i = [i]\|C$. Build on-line diamond filter for states in Y.
- **Step 3 (off-line).** Match between set X of 2^t off-line states and set Y of 2^u on-line states using the on-line filters.



We detect (off-line) a match between 2^t off-line known states (⭑) with 2^u on-line unknown states (⬟) using the diamond filter built on-line.

Figure 18: Image-set size reduction-based state recovery for hash-base MAC [DL17]

According to Lemmas 1 and 2, a pair of states in X and Y collides with probability 2^{s-l}. There are in total 2^{t+u} pairs of states. Thus, to find at least one match for the attack to succeed, parameters t, u and s must satisfy $t+u \geq l-s$. Thus, we first set $t = l-s-u$. Complexity of each step is:

Step 1: $2^{t+s} = 2^{l-u}$ **Step 2:** $2^{u+s} + u \cdot 2^{s+u/2+l/2}$ **Step 3:** $2^{t+u} \cdot u = 2^{l-s} \cdot u$

Note that $u < l/2$, thus $u+s < s+u/2+l/2$. We balance the complexity of Steps 1 and 2 by setting $l-u = s+u/2+l/2$, which gives $u = l/3 - 2s/3$. The sum of dominating terms is then $2^{2l/3+2s/3} + 2^{l-s}$. To summarize, the attack complexity is $\tilde{O}(2^{l-s})$ for $s \leq l/5$, and is optimally $4l/5$ achieved by setting $2l/3 + 2s/3 = l-s$, i.e. $s = l/5$.

State recovery attack based on free-offset collisions [DL14, DL17]. Based on the entropy loss in the output of collision search algorithms indicated by Lemma 3, one can launch more efficient state recovery attacks on MD hash-based MACs than the previous one (see Fig.19).

- **Step 1 (off-line).** Collect 2^c off-line free-offset collisions by developing 2^{l-2s} chains of length 2^s. Build an off-line diamond filter for those collisions.
- **Step 2 (on-line).** Find 1 on-line same-offset collision by querying the oracle with 2^t messages $M_i = [i]\|[0]^{2^s}$. Locate the unknown collision point using a binary search.
- **Step 3 (on-line).** Match the unknown on-line same-offset collision with the set of 2^c off-line free-offset collisions using the off-line diamond filter.

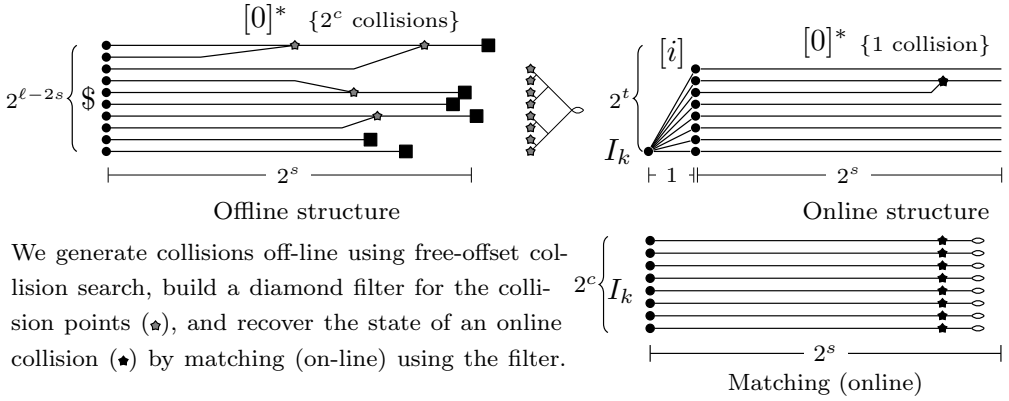


Figure 19: State recovery attack based on free-offset collisions and using off-line diamond filters on hash-base MAC [DL17]

According to the analysis on the free-offset collision search algorithm, the number of free-offset collisions detected off-line is $2^c \approx 2^{l-2s}$. Thus, from Lemma 3, the set of off-line collisions contains the collision found on-line with high probability. To find one same-offset collision on-line, we set $2t + s - l = 0 \Rightarrow t = (l - s)/2$. The complexity of each step is:

Step 1: $2^{l-s} + 2^{(c+l)/2} \approx 2^{l-s}$ **Step 2:** $2^{t+s} + s \cdot 2^s = 2^{(l+s)/2} + s \cdot 2^s$

Step 3: $2^{c+s} = 2^{l-s}$

Note that, there is a restriction on parameters t and s, that is $t + 2s < l$ due to the assumption required to analyze the collision search algorithm. Thus, this complexity analysis is valid only for $(l - s)/2 + 2s < l$, i.e., $s \leq l/3$.

The inherent lower bound on the complexity of this attack is $\max\{2^{l/2+s/2}, 2^{l-s}\}$, which creates the two lines forming the curve for Attack 4 in Fig.26a, where $2^{l/2+s/2}$ is the complexity of finding a single same-offset collision, and 2^{l-s} is the complexity of matching (on-line) between off-line free-offset collisions and on-line same-offset collisions. By setting $s = l/3$, the two terms can get a balance, which gives an optimal complexity $2^{2l/3}$. To summarize, the total complexity of this attack is $O(2^{l-s})$ if $s \leq l/3$.

Note that, for smaller message (using queries of length $2^s \leq 2^{l/8}$), one can optimize this free-offset collision-based state recovery attack by collecting more on-line same-offset collisions, building *on-line diamond filter* for their endpoints, and matching them off-line (refer to Sect. 2.4.6 with free-offset collisions found off-line. Then, the attack complexity turns to be $\tilde{O}(2^{l-2s})$ if $s \leq l/8$, and is optimally $\tilde{O}(3l/4)$ obtained by setting $s = l/8$.

4.3.3 Universal Forgery Attacks With Short Messages [DL14, DL17]

Dinur and Leurent further provide two methods to improve the first phase of previous universal forgery attacks with short messages.

Previous height-based attacks exploit the observation that height provides a way to classify nodes into mutually disjoint subsets. Thus, a two-phase matching process is more efficient than directly matching. Actually, distances from any special node can also provide a measure to divide the matching into two-phase process which is more efficient than directly matching those off-line states and those on-line states. In previous height-based attacks, the α-node in the functional graph plays a role as an "origin" of a coordinate system. This "origin" is detectable both on-line and off-line. Distances from this α-node (height) for on-line nodes and off-line nodes are also detectable. By detecting distance of a node from this "origin", one can get direct information of this node. Actually, one can use other kinds of special nodes as "origin" of a coordinate system. In this case, the original coordinate system are now divided into several subsystems (without overlap because $f \triangleq h_{[m]}$ is a mapping). To get information of a node, we first determine their "origin" and then find more information on it by exploiting its distance from the "origin" in the subsystem. That is actually how the following two improved universal forgery attack processed. An advantage when using other kinds of special nodes is that one can launch attacks with short message.

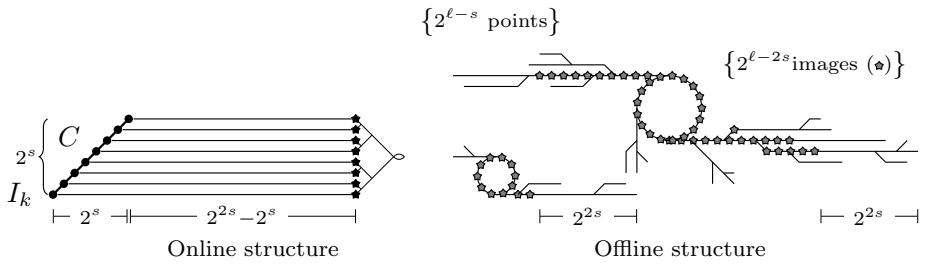
The first improved attack bases on the reduction of the image-set size as shown in Lemma 2. The second improved attack bases on the previous mentioned collision-based state recovery attack in Sect. 4.3.2 (which bases on the entropy loss in the output of collisions search algorithms as indicated in Lemma 3 in which the message is completely chosen by the adversary).

Universal forgery attack based on the reduction of the image-set size. Denote the challenge message by C. Details about the state recovery phase in the first improved universal forgery attack are as follows (see Fig.20) [3]:

- **Step 1 (on-line).** Generate a set X of unknown final states of 2^s chains of 2^{2s} length by querying the oracle with messages $M_i = C_{|i}\|[0]^{2s-i}$. Build an on-line diamond filter for X.
- **Step 2 (off-line).** Compute a set Y of 2^{2s}-iterates nodes (namely images of $h_{[0]}^{2^{2s}}$) by running Algorithm 1 with parameter $l - s$ and with the following modification: record the iteration times of $h_{[0]}$ to get each node instead of recording their heights.
- **Step 3 (off-line).** Match each node $y \in Y$ with the 2^s states in X using the on-line diamond filter built for X.
- **Step 4 (off-line).** For each match between an off-line node $y \in Y$ and an on-line state $x \in X$ obtained using M_i, launch the second matching phase to match the unknown state after processing $C_{|i}$, i.e. a predecessor which is $2^{2s} - i$ away from x, with all predecessors which are $2^{2s} - i$ away from y. This matching is performed using a binary search matching algorithm operated on a tree rooted at y in the functional graph (refer to [DL17] for details). Initial inputs to the binary search matching algorithm includes the message M_i, the tree rooted at y and distance $2^{2s} - i$. With high probability, the returned matched state y' equals the internal state after processing $C_{|i}$.

This attack will succeed with high probability. Reasons are as follows: The MAC oracle computes 2^s internal states on prefix of the challenge C. These 2^s on-line states must be predecessors of states in X in $\mathcal{FG}_{h_{[0]}}$. There are 2^{l-s} distinct nodes in $\mathcal{FG}_{h_{[0]}}$ developed off-line in Step 2. According to the birthday paradox, between these on-line states and

[3] Note that, in the original description of this attack in [DL17], notation on the set of on-line states and notation on the set of off-line states is respectively Y and X, which is not consistent with the provided figure.



We efficiently detect a match between the challenge points (•) and the offline structure, by first matching X (★) and Y (★).

Figure 20: Phase 1 of the universal forgery based on the reduction of image-set size [DL17]

these off-line nodes, there is an intersection point with high probability. As long as there exists an intersection point, the attack will always find it using the two follow-up matching steps 3 and 4.

The analysis of the computational complexity of this attack is as follows: In Step 1, generating X requires 2^{3s} computations and building the on-line diamond filter requires $2^{2s+s/2+l/2}$ computations. In Step 2, it requires 2^{l-s} computations to develop 2^{l-s} off-line nodes. There are about 2^{l-2s} distinct nodes in Y according to Lemma 2. The complexity of matching an element of Y with 2^s states in X can be estimated as $s \cdot 2^s$. Thus, the complexity of matching 2^{l-2s} elements of Y with 2^s elements of X in Step 3 can be approximated by $2^{l-2s} \cdot s \cdot 2^s = s \cdot 2^{l-s}$. In Step 4, the binary search matching algorithm recursively calls itself with a tree parameter cut by at least half in every recursive call. Thus, for a single call in Step 4, there are at most $l-s$ recursive calls of the binary search matching considering the initial tree has at most 2^{l-s} nodes. Within each recursive call, it traverses at most $d \leq 2^{2s}$ nodes, builds and tests a small constant number of collision filters. The complexity of those procedures within each recursive call are then respectively estimated as $d < 2^{l/2}$, $2^{l/2}$ and $2^{2s} < 2^{l/2}$. Considering there are at most 2^s matches in Step 3, the number of initial calls in Step 4 is at most 2^s. In conclusion, the time complexity of Step 4 is at most $2^s \cdot (l-s) \cdot 2^{l/2} \approx (l-s) \cdot 2^{l/2+s}$. To sum up, the complexity of each step in this attack is approximately:

$$\textbf{Step 1:}\quad 2^{l/2+5s/2} \qquad \textbf{Step 2:}\quad 2^{l-s} \qquad \textbf{Step 3:}\quad s \cdot 2^{l-s} \qquad \textbf{Step 4:}\quad (l-s) \cdot 2^{l/2+s}$$

The total complexity of this attack is $\tilde{O}(2^{l-s})$ if $s \leq l/7$ in which case $l/2 + 5s/2 \leq 6l/7 \leq l-s$ (the second phase of the universal forgery attack has similar complexity). The optimal complexity is $2^{6l/7}$ obtained by setting $s = l/7$ with queries of length $2^{2s} = 2^{2l/7}$.

Universal forgery attack based on the collision-based state recovery attack and the reduction of the image-set size. The first improved attack recovers the final states of on-line chains by building on-line diamond filters for them. The complexity analysis shows that when the length of queries exceeds $2^{2l/7}$, generating those on-line final states and building on-line filters for them will be the most expensive step. To improve the complexity when queries can be relatively longer, instead of building filters, the second improved attack recovers the final states of on-line chains by directly matching the tags with tags of carefully selected messages. Collisions among tags imply collisions among those final states. Those carefully selected messages are crafted based on the result of launching a collision-based state recovery attack shown in Sect. 4.3.2. Thus, the internal states for those carefully selected messages are known. Therefore, those colliding final states of on-line chains are also known. The state recovery phase of the second improved collision-based attack is as follows, again let C be the challenge message:

- **Step 1 (on-line).** Get a set S of 2^s tags by querying the oracle with messages $M_i = C_{|i} \| [0]^{2^{s+1}-i}$.

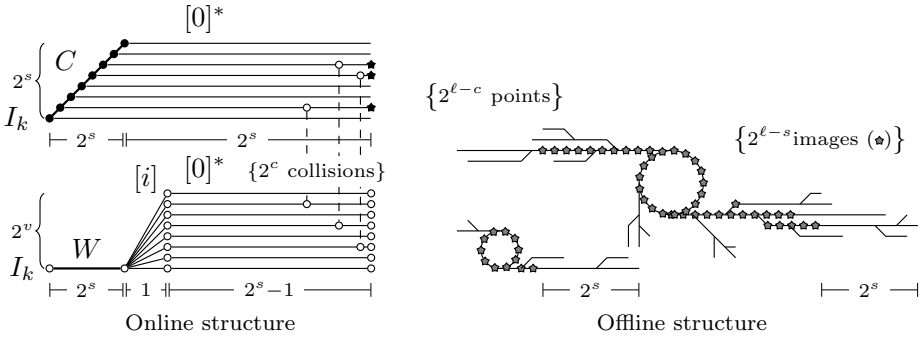
- **Step 2 (on-line).** Get a set T of 2^v tags by querying the oracle with messages $W_j = W \| [j] \| 0^{2^s-1}$. In each W_j, W is a message of length 2^s whose last computed state is recovered by executing the free-offset collision-based state recovery attack in Sect. 4.3.2 with messages of length $\min(2^s, 2^{l/3})$.

- **Step 3 (on-line).** Generate a set X of states reached after some M_i's by matching tags in S with tags in T. Suppose we get 2^c collisions between the tags. For each collision of tags between M_i and W_j, the state reached after M_i is known because the state reached after W_j is already known in Step 2.

- **Step 4 (off-line).** Generate a set Y of 2^s-iterates nodes (namely images of $h_{[0]}^{2^s}$) by running Algorithm 1 with parameter $l - c$ and with the following modification: record the iteration times of $h_{[0]}$ to get each node instead of recording their heights.

- **Step 5 (off-line).** Match each node $y \in Y$ with the 2^c recovered states in X directly.

- **Step 6 (on-line).** For each match between an off-line node $y \in Y$ and an on-line state $x \in X$ obtained using M_i, launch the second matching phase that is almost identical to Step 4 in the first improved attack. The only difference lies in one of the initial input parameters to the binary search matching algorithm, that is the distance is $2^{s+1} - i$ (instead of $2^{2s} - i$).



We match the known points in X (⋆) and Y (⋆) in order to detect a match between the challenge points (•) and the offline structure.

Figure 21: Phase 1 of the universal forgery attack using collisions [DL17]

The attack will succeed with high probability. Reasons are similar to that of the previous attack: There are 2^c internal states computed on-line on the prefix of the challenge C. They must be predecessors of states in X in $\mathcal{FG}_{h_{[0]}}$. There are 2^{l-c} distinct nodes in $\mathcal{FG}_{h_{[0]}}$ developed off-line in Step 4. The intersection point between the on-line states and the off-line nodes can be found using the follow-up matching steps.

The expected number of states in X is $2^c = 2^{2s+v-l}$ (recovered using Steps 1 - 3). Reasons are based on the reduction of the image-set size of $h_{[0]}^{2^s}$ and the assumption on independence between all evaluated on-line chains (there are 2^{s+v} pairs of tags and each pair colliding with probability 2^{s-l} from Lemma 1). The computational complexity of

each step in this attack is approximately:

Step 1:	2^{2s}	**Step 2:**	$\max(2^{l-s}, 2^{2l/3}) + 2^{v+s}$
Step 3:	$\max(2^s, 2^v)$	**Step 4:**	$2^{l-c} = 2^{2l-2s-v}$
Step 5:	$2^{l-s} \cdot 2^c = 2^{v+s}$	**Step 6:**	$(l-c) \cdot 2^{l/2+c} = (2l-2s-v) \cdot 2^{2s+v-l/2}$

Set $v + s = 2l - 2s - v \Rightarrow v = l - 3s/2$ to balance Steps 2, 4 and 5. The total complexity of this attack is $\tilde{O}(2^{l-s/2})$ for any $s \leq 2l/5$. The optimal complexity is $2^{4l/5}$ obtained by setting $s = 2l/5$, and with queries of length $2^s = 2^{2l/5}$.

5 Attacks on Hash Combiners Based on Functional Graph

In this section, we survey generic attacks against various hash combiners (refer to Fig. 1b for a list of main surveyed papers and their technical relations). Like in Sect. 4, we look into attacks involving iteratively evaluating a mapping many times, i.e., attacks exploiting properties of the functional graph of random mappings. And we mainly focus on attacks against combiners of two narrow-pipe MD hash functions (in each of the hash functions, both of the internal states and the output are n-bit). Note that, we reassign l (which formerly represented the width of internal states in Sect. 4) to represent the size of message, i.e. suppose the message is of length $L = 2^l$.

5.1 Attacks Based on Deep Iterates (FGDI [Din16])

Properties of the random functional graph have been largely exploited in generic attacks against hash-based MACs as shown in Sect. 4. Dinur discovers that some properties can also be exploited to launch efficient generic attacks on MD hash combiners. In the sequel, we denote by $f_1 = h_{1_{[m]}}$ and $f_2 = h_{2_{[m]}}$ the two n to n-bit random mappings derived from the two underlying compression functions h_1 and h_2 in the way shown in Sect. 3.3, and denote the corresponding functional graphs by $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$.

By exploiting special nodes in the functional graph, Dinur presents the first second-preimage attack on the concatenation combiner and an improved preimage attack on the XOR combiner of MD hash functions. The exploited special nodes are explicitly named *deep iterates*. Actually, deep iterates have already been exploited in attacks on hash-based MACs shown in Sect. 4.3 (the attacks based on the reduction of image-set size that occurs when iterating a random mapping many times). They are named deep iterates because they locate "deep" in the functional graph, i.e., there exists a terminal node reaching a deep iterate after large number of iterations of the random mapping. In other words, they are k-th iterate image nodes and k is relatively large. Two observations on deep iterates make them helpful in the proposed attacks:

- **Observation 1.** It is easy to get a large set of deep iterates. Specifically, by running Algorithm 1 with input parameter t (and do not record height), one can get a set of 2^t nodes, among which a constant fraction ($\Theta(2^t)$) are 2^{n-t}-th iterates.

- **Observation 2.** A deep iterate has a relatively high probability to be reached from an arbitrary starting node (refer Lemma 1). Thus, in a pair of functional graphs ($\mathcal{FG}_{f_1}$, $\mathcal{FG}_{f_2}$), the probability for a random pair (x_0, y_0) encountering d-th iterate image pair $(\bar{x}, \bar{y})$ at a common distance is estimated as $d^3/2^{2n}$. The expected number of trials to find such a pair (x_0, y_0) is then reasonably conjectured to be $2^{2n}/d^3$.

5.1.1 Second-Preimage Attack on Concatenation Combiner Based on Deep Iterates [Din16]

A primary second-preimage attack modified from Kelsey and Schneier's attack on a single MD-hash function [KS05] (refer to Sect. 2.4.2) is not efficient on concatenation combiner of two MD-hash functions mainly because of two problems: One should construct a set of messages that is simultaneously expandable and colliding with respect to both of the two underlying hash functions; One should efficiently find a common message fragment mapping the two output states of the expandable messages to two internal states at the same offset under two hash functions. There is actually $2n$-bit to be hit. To solve the first problem, Dinur invents the simultaneous expandable message (refer to Sect. 2.4.5). To solve the second problem, Dinur exploits those deep iterates in functional graphs according to the above observations.

In Dinur's second-preimage attack on concatenation combiner, after constructing a simultaneous expandable message starting from the two initialization vectors, instead of directly mapping the final states to a pair of internal states at the same offset in the two hash computations, it utilizes a pair of deep iterates as bridge to make the connection efficient. More explicitly, the first observation on deep iterates makes it efficient to independently find two sets of deep iterates for the two hash computations, which provides extra freedom to launch a meet-in-the-middle procedure when finding a message block mapping a pair of deep iterates to a pair of internal states at the same offset in the original two hash computations. The second observation on deep iterates makes it efficient to find a pair of random starting nodes connecting the pair of final states of the simultaneous expandable message to the pair of target deep iterates with common distance. In other words, make it efficient to find a common message fragment mapping the pair of final states of the simultaneous expandable message to the target pair of deep iterates.

Given a target message $M = m_1 \| m_2 \| \dots \| m_L$, the goal of the second-preimage attack on concatenation combiner is to find another message M' such that $\mathcal{H}_1(M') \| \mathcal{H}_2(M') = \mathcal{H}_1(M) \| \mathcal{H}_2(M)$. Suppose the internal state chains in computations of the original message M under the two hash functions are respectively $(a_1, a_2, \dots, a_L)$ and $(b_1, b_2, \dots, b_L)$. The attack procedure is sketched as follows (See Fig. 22 for more detailed steps):

- **Phase 1:** Build a simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ starting from the pair of initialization vectors (IV_1, IV_2) and ending at a pair of final state $(\hat{x}, \hat{y})$, such that its length could extend up to L.

- **Phase 2:** Find a pair of states $(\bar{x}, \bar{y})$ and a single message block $\bar{m}$ such that there exist $q \leq L$ and $(h_1(\bar{x}, \bar{m}), h_2(\bar{y}, \bar{m})) = (a_p, b_p)$. That is done as follows: First, generate two independent sets (of size 2^{n-g}) of 2^g-th deep iterates in the two functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$. Then, from the two sets of deep iterates, launch a meet-in-the-middle procedure to find a pair of state $(\bar{x}, \bar{y})$ and the single-block message $\bar{m}$ fulfilling the requirement. Refer $(\bar{x}, \bar{y})$ as target node pair.

- **Phase 3:** Find a message fragment $\hat{M} = \hat{m} \| [m]^d$ such that it maps $(\hat{x}, \hat{y})$ to $(\bar{x}, \bar{y})$. That is done as follows: First, launch a look-ahead procedure by developing more nodes (2^t) and storing with their distances from the target nodes in the functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ independently. Then, start from state pair $(\hat{x}, \hat{y})$, enumerate (2^{2n-3g} trials) a message block $\hat{m}$ to find a pair of starting nodes $(x_0, y_0) = (h_1(\hat{x}, \hat{m}), h_2(\hat{y}, \hat{m}))$ such that they reach the pair of target deep iterate $(\bar{x}, \bar{y})$ at a common distance d in the two functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$. Note that, this phase is more efficient than 2^n computations only when using results of the aforementioned look-ahead procedure to accelerate the deduction of distances between the starting nodes and the target nodes.

At the end, select a message prefix $M_{\|L-d-2}$ with $L-d-2$ blocks from the simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$, and construct a second-preimage M':

$$M' = M_{\|L-d-2}\|\hat{m}\|[m]^d\|\bar{m}\|m_{p+1}\|\ldots\|m_L.$$

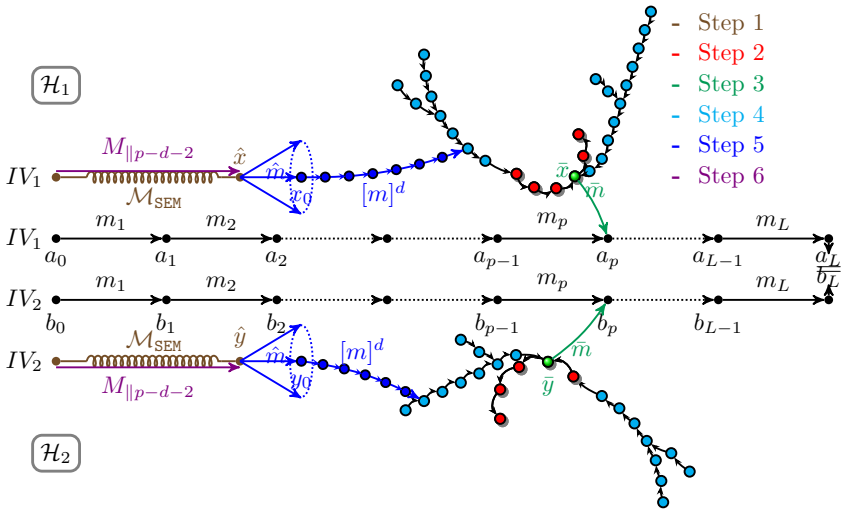


Figure 22: Second-preimage attack on concatenation combiner based on deep iterates

Denote the message length $L = 2^l$. Restriction on attack parameter g is $g \leq \min(n/2, l)$. Complexity of this attack is dominated by Phases 2 and 3 as long as message length is not extremely large. For Phase 2, the time complexity of the generation of the two sets of deep iterates is 2^{n-g}, and the time complexity of the meet-in-the-middle procedure is $2^{n-g} \cdot 2^{2g-l} = 2^{n+g-l}$ which is dominated in Phase 2. For Phase 3, mutual restraint factors are the time complexity of the look-ahead procedure and complexity of finding a pair of starting nodes fulfilling the requirement, i.e., between 2^t and $2^{2n-3g} \cdot 2^{n-t} = 2^{3n-3g-t}$. Here, we can directly set $t = 3n - 3g - t$ or $t = 3n/2 - 3g/2$ to make a balance because t is a local parameter in Phase 3. Thus, complexities of each phase are as follows:

Phase 1: $2^l + n^2 \cdot 2^{n/2}$ **Phase 2:** 2^{n+g-l} **Phase 3:** $2^{3n/2-3g/2}$

From the complexity formula of Phase 3, the deeper the two target nodes are located (the larger the value of g), the more efficient it is to find a pair of starting nodes reaching them at a common distance. A more fundamental reason is that the deeper a target node is located, the more preimages it has, the higher the probability is for a randomly selected node being one of its preimages. On the other hand, from the complexity formula of Phase 2, the deeper the target nodes are located (the larger the value of g), the more difficult it is to find a pair of them mapping by a message block $\bar{m}$ to a pair of internal states in the computation on the original message. A more fundamental reason is that the deeper the target nodes are located, the less their population is. Note that, the inherent lower bound on the complexity of this attack is determined by Phase 3. Because to fulfill restriction $g \leq \min(n/2, l)$, the complexity of Phase 3 is lower bounded by $\max(2^{3n/4}, 2^{3n/2-3l/2})$.

We make a balance between Phases 2 and 3 by setting $n + g - l = 3n/2 - 3g/2$, which gives $g = n/5 + 2l/5$. Thus, the overall complexity is $2^{6n/5-3l/5}$ if $l \leq 3n/4$, and is optimally $2^{3n/4}$ obtained for $l = 3n/4$. Otherwise, it is dominated by Phase 1 and equals 2^l.

5.1.2 Preimage Attack on XOR Combiner Based on Deep Iterates [Din16]

Similar to the second-preimage attack on concatenation combiner, deep iterates can also be exploited to improve the preimage attack on XOR combiner.

Given a target hash digest V, the goal of the preimage attack on XOR combiner is to find a message M such that $\mathcal{H}_1(M) \oplus \mathcal{H}_2(M) = V$. The attack procedure is quite similar to the second-preimage attack on concatenation combiner and is also composed of three main phases. Differences mainly lie in the second phase because the target is now of n-bit instead of $(2n-l)$-bit. The freedom from $(n-l)$-bit restriction allows us efficiently finding more target pairs $\{(\bar{x}_1, \bar{y}_1), (\bar{x}_2, \bar{y}_2), \dots\}$ instead of a single target pair $(\bar{x}, \bar{y})$. The attack procedure is sketched as follows (describe only the different phases with previous second-preimage attack in Sect. 5.1.1, see Fig. 23 for detailed steps):

- **Phase 2:** Generate a set (of size 2^s) of tuples $\{(\bar{x}_1, \bar{y}_1, \bar{m}_1), (\bar{x}_2, \bar{y}_2, \bar{m}_2), \dots\}$ such that for each i, $h_1^*(\bar{x}_i, \bar{m}_i \| pad) \oplus h_2^*(\bar{y}_i, \bar{m}_i \| pad) = V$, where pad is the final block of the (padded) preimage message of a predefined length L. This is done similarly to Phase 2 in the second-preimage attack in Sect. 5.1.1, including generating 2^g-th iterates and launching a meet-in-the-middle procedure.

- **Phase 3:** Find a message fragment $\hat{M} = \hat{m} \| [m]^d$ such that it maps $(\hat{x}, \hat{y})$ to one of the target pair $(\bar{x}_i, \bar{y}_i)$. That is done as follows: First, launch a look-ahead procedure by developing more nodes (2^t) and storing with their distances from all the target nodes in the functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ independently. Note that, this procedure additionally requires the distinguishing point technique because one has to record the distance for each node from 2^s target nodes instead of a single target as in the previous attack. Here, the distinguishing point technique provides trade-off between time and memory. Then, start from the state pair $(\hat{x}, \hat{y})$, enumerate ($2^{2n-3g-s}$ trials) a message block $\hat{m}$ to find a pair of starting nodes $(x_0, y_0) = (h_1(\hat{x}, \hat{m}), h_2(\hat{y}, \hat{m}))$ such that they reach any one of the pairs of target deep iterate $(\bar{x}_i, \bar{y}_i)$ at a common distance (denoted by d) in the two functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$. Again, this phase is more efficient than 2^n computations only when using results of previous look-ahead procedure.

At the end, select a message prefix $M_{\|L-d-3}$ with a block length $L-d-3$ from the simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$, and construct a preimage M: $M = M_{\|L-d-3} \| \hat{m} \| [m]^d \| \bar{m}_i \| pad$.

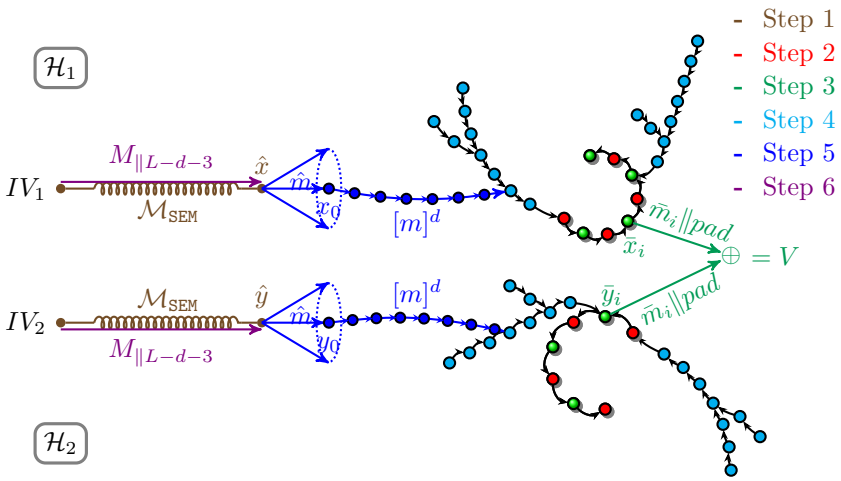


Figure 23: Preimage attack on XOR combiner based on deep iterates

The complexity analysis of this attack is similar to that of the second-preimage attack on concatenation combiner. Again, denote the message length $L = 2^l$. Restriction on attack parameter g is also $g \leq \min(n/2, l)$. Differences mainly lie in Phase 2. That is, there is only n-bit restriction on the choice of $\bar{x}_i$, $\bar{y}_i$ and $\bar{m}_i$. That is because the equation $h_1^*(\bar{x}_i, \bar{m}_i \| pad) \oplus h_2^*(\bar{y}_i, \bar{m}_i \| pad) = V$ only imposes n-bit restriction, compared with the $(2n-l)$-bit restriction imposed by equation $(h_1(\bar{x}, \bar{m}), h_2(\bar{y}, \bar{m})) = (a_p, b_p)$ (for any $p \leq 2^l$) in the second-preimage attack on concatenation combiner. This $(n-l)$-bit freedom allows collecting more target pairs (2^s). Thus, the complexity of this phase is reduced by a factor of 2^{n-l-s} from 2^{n+g-l}, and turns to be 2^{g+s}. This freedom is further transferred to the next phase to relieve from workload to find the pair of starting pairs meeting the requirement (reduced by a factor of 2^s from 2^{2n-3g}). The only inconvenience caused by utilizing this freedom is that: when developing nodes during the look-ahead procedure, one has to store with each node its distance from lots of target nodes. Thus, memory consumption becomes severe. To keep memory requirement reasonable, one has to trade the time for memory by adopting the distinguishing point technique. The specific time traded off is at most $2^{l+t-n} + 2^{l+s-(n-g)}$ per trial of starting node pair. Thus, the time complexity of Phase 3 turns to be $2^t + 2^{2n-3g-s} \cdot (2^{n-t} + 2^{l+t-n} + 2^{l+s-(n-g)}) = 2^t + 2^{3n-3g-s-t} + 2^{n-3g-s+l+t} + 2^{n-2g+l}$. Again, because t is a local parameter in Phase 3, we can first make a balance within Phase 3 by setting $t = 3n - 3g - s - t$, which gives $t = 3n/2 - 3g/2 - s/2$. Then, the time complexity of each phase can be summarized as follows:

Phase 1: $2^l + n^2 \cdot 2^{n/2}$ **Phase 2:** 2^{g+s}

Phase 3: $2^{3n/2-3g/2-s/2} + 2^{5n/2-9g/2-3s/2+l} + 2^{n-2g+l}$

From the complexity formula of Phase 3, the deeper the target nodes (the larger the value of g) and the more the target pairs (the larger the value of s), the more efficient this phase is. On the other hand, from the complexity formula of Phase 2, the effect of increasing the depth and increasing the numbers of target nodes is reverse. To make a balance, we first set $g+s = 3n/2-3g/2-s/2$, which results in $s = n-5g/3$. After plugging s back, the complexity of Phase 2 and Phase 3 turns to be about $2^{n-2g/3} + 2^{n-2g+l}$.

In case of $l \leq n/2$, we set $g = l$ due to the restriction $g \leq \min(n/2, l)$. The overall time complexity is about $2^{n-2l/3}$. In case of $l > n/2$, we set $g = n/2$, the overall time complexity is about $2^l + 2^{2n/3}$. The optimal complexity is $2^{2n/3}$ obtained for $l = n/2$.

5.2 Attacks Based on Multi-Cycles (FGMC [BWGG17])

Note that, one of the key point in Dinur's attacks on hash combiners is to efficiently find a pair of starting nodes (x_0, y_0), which reaches to a pair of target deep iterates $(\bar{x}, \bar{y})$ at a common distance. In [BWGG17], cyclic nodes in functional graph are exploited to make it more efficient to find such pair of starting nodes (x_0, y_0). Main observation is that, it is effortless to loop around the cycles to correct bias between their distances using difference of the two cycle lengths. More specifically, suppose one is starting from a random node x_0 (resp. y_0), after d_1 (resp. d_2) iterations of f_1 (resp. f_2), it reaches a cyclic node $\bar{x}$ (resp. $\bar{y}$) in $\mathcal{FG}_{f_1}$ (resp. $\mathcal{FG}_{f_2}$) for the first time. Further, suppose the cycle length is L_1 (resp. L_2). Without loss of generality, assume $L_1 \leq L_2$ and let $\Delta L = L_2 \mod L_1$. We know if $\exists\ (i,\ j),\ 0 \leq i, j \leq t,$ s.t. $d_1 + i \cdot L_1 = d_2 + j \cdot L_2$ then $(d_1 - d_2) \mod L_1 = j \cdot \Delta L \mod L_1$. Let

$$\mathcal{S} = \{j \cdot \Delta L \mod L_1 \mid j = 0, 1, \ldots, t\},$$

where t is the maximum number of cycles limited to be used. Since $d_1 = O(2^{n/2})$, $d_2 = O(2^{n/2})$ and $L_1 = \Theta(2^{n/2})$, it has $|d_1 - d_2| = O(L_1)$. We assume $|d_1 - d_2| < L_1$ by

ignoring the constant factor. Then, for the randomly sampled pair (x_0, y_0) that encounters $(\bar{x}, \bar{y})$, as long as their distance bias $(d_1 - d_2) \mod L_1$ is in the set $\mathcal{S}$, we are able to derive a pair of integers (i, j) such that $d_1 + i \cdot L_1 = d_2 + j \cdot L_2$. That is to say, their distance bias is correctable by differences of multi-cycle lengths. Hence, the probability of reaching $(\bar{x}, \bar{y})$ from a random pair (x_0, y_0) at the same distance is amplified by roughly t times. We refer $\mathcal{S}$ as the *set of correctable distance bias*.

5.2.1 Preimage Attack on XOR Combiner Based on Multi-Cycles [BWGG17]

Using cyclic nodes as targets and exploiting multi-cycles to correct distance bias, Dinur's preimage attack on XOR combiner can be optimized for $l \geq n/2$ as shown in [BWGG17]. The attack procedure is sketched as follows (describing only the differences with previous preimage attack in Sect. 5.1.2, see Fig. 24 for detailed steps):

- **Phase 2:** Generate a set (of size 2^s) of tuples $\{(\bar{x}_1, \bar{y}_1, \bar{m}_1), (\bar{x}_2, \bar{y}_2, \bar{m}_2), \dots\}$ such that for each i, $h_1^*(\bar{x}_i, \bar{m}_i \| pad) \oplus h_2^*(\bar{y}_i, \bar{m}_i \| pad) = V$, where pad is the final block of the (padded) preimage message of a predefined length L. Besides, those $\bar{x}_i$'s and $\bar{y}_i$ are cyclic nodes within the largest components in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$. That is done as follows: First, repeat the cycle search a few times (refer Sect. 6.1). Thus, one can get two independent sets (of size about $2^{n/2}$) of cyclic nodes as well as the cycle lengths L_1 and L_2. Then, from the two sets of cyclic nodes, launch a meet-in-the-middle procedure to find the set of tuples $(\bar{x}_i, \bar{y}_i, \bar{m}_i)$ fulfilling the requirement.

  In addition, compute the set of correctable distance bias $\mathcal{S} = \{j \cdot \Delta L \mod L_1 \mid j = 0, 1, \dots, \#C\}$ with parameters L_1, L_2 and $\#C$ (where $\#C$ is the maximum number of cycles that can use which is essentially $\lfloor L/L_1 \rfloor$, and $\Delta L = L_2 \mod L_1$.)

- **Phase 3:** Find a message fragment $\hat{M} = \hat{m} \| [m]^d$ such that it maps $(\hat{x}, \hat{y})$ to one of the target pair $(\bar{x}_i, \bar{y}_i)$. That is done as follows: First, launch a look-ahead procedure by developing more nodes (2^t) similar to that in previous preimage attack. The difference is that, it does not require the distinguishing point technique to trade time for memory, because the distance for each node from all target nodes can be derived from the distance value for this node from any one target node, and the distance from this particular target node to other targets nodes. Then, start from the state pair $(\hat{x}, \hat{y})$, enumerate (2^{n-s-l} trials) a message block $\hat{m}$ to find a pair of starting nodes $(x_0, y_0) = (h_1(\hat{x}, \hat{m}), h_2(\hat{y}, \hat{m}))$ such that they reach any one of the pairs of target deep iterate $(\bar{x}_i, \bar{y}_i)$ with distance bias $(d_1 - d_2) \mod L_1 \in \mathcal{S}$ in the two functional graphs $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$. Retrieve the common distance and denote it by $d \triangleq d_1 + i \cdot L_1 = d_2 + j \cdot L_2$ according to d_1, d_2, L_1 and L_2. Again, this phase is more efficient than 2^n computations only when using results of previous look-ahead procedure.

At the end, select a message prefix $M_{\|L-d-3}$ with a block length $L-d-3$ from the simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$, and construct a preimage M: $M = M_{\|L-d-3} \| \hat{m} \| [m]^d \| \bar{m}_i$.

The complexity analysis of this attack can be briefly deduced based on that of the deep-iterate-based preimage attack in Sect. 5.1.2. Denote the message length $L = 2^l$. In Phase 2, considered deep iterates are cyclic nodes whose depth is $2^{n/2}$, and whose population is about $2^{n/2}$. We can simply plug $g = n/2$ into the previous complexity formula 2^{g+s} of Phase 2 in the deep-iterate-based preimage attack, which gives $2^{n/2+s}$. For Phase 3, complexity analysis also can be made based on the previous complexity formula $2^t + 2^{2n-3g-s} \cdot 2^{n-t}$ of Phase 3 in the previous preimage attack (note that there is no additional time consumption caused by the distinguishing point technique as discussed before). The difference is that, number of required trials on starting node pairs can be reduced by a factor of $\#C = L \cdot 2^{-n/2} = 2^{l-n/2}$ from $2^{2n-3g-s}$. Again, after plugging

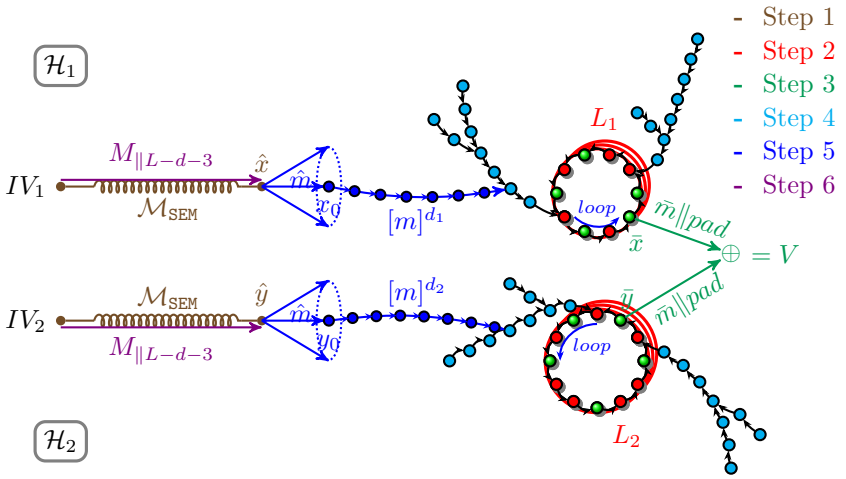


Figure 24: Preimage attack on XOR combiner based on multi-cycles

$g = n/2$ into the reduced formula, we get the complexity of this updated Phase 3, i.e., $2^t + 2^{2n-3n/2-s-l+n/2} \cdot 2^{n-t} = 2^t + 2^{2n-s-l-t}$. To make a balance within Phase 3, we set $t = 2n - s - l - t$, which results in $t = n - s/2 - l/2$. Thus, the computational complexity for each phase is as follows:

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

From the complexity formula of Phase 3, the more the target pairs (the larger the value of s) and the larger the maximum number of cycles limited to be used (which depended on l, and is $2^{l-n/2}$), the more efficient this phase is. On the other hand, from the complexity formula of Phase 2, the more required target pairs, the more time-consuming this phase is. To make a balance, we set $n/2+s = n-s/2-l/2$ which gives $s = n/3-l/3$. Consequently, the final complexity is $2^l + 2^{5n/6-l/3}$. Note that, this is only applicable for $l \geq n/2$. The optimized complexity of this attack is $2^{5n/8}$, obtained when $l = 5n/6 - l/3$, i.e. $l = 5n/8$.

5.2.2 Second-Preimage Attack on Zipper Hash Based on Multi-Cycles [BWGG17]

Previous techniques used in generic attack on parallel combiners are also applicable to carry out a second-preimage attack on the cascade combiner Zipper hash. Authors in [BWGG17] present the first generic second-preimage attack on Zipper hash, which is applicable for idealized compression function. The unique specification of Zipper hash makes it a main victim of the proposed second-preimage attack. Since the message length is fed in the middle of the computation instead of at the end, it enables to propagate an internal collision of messages of different length to the final output. That brings extra freedom to optimize the computational complexity by choosing a length for the second preimage message. Moreover, processing different message blocks enables to break the dependency between the two computational passes by building two independent standard multi-collsions, and thus enables to launch a meet-in-the middle procedure when searching for a pair of starting nodes reaching a pair of targets nodes $(\bar{x}, \bar{y})$ at a common distance.

The second-preimage attack on Zipper hash goes as follows (See Fig. 25 for more detailed steps): Given a target message $M = m_1\|m_2\|\ldots\|m_L$, the goal of the second-preimage attack on Zipper hash is to find another message M' such that $\mathcal{H}_2(\mathcal{H}_1(IV, M'), \overleftarrow{M'}) = \mathcal{H}_2(\mathcal{H}_1(IV, M), \overleftarrow{M})$. Suppose the internal state chains in computations of the original message M under the two hash functions are respectively $(a_1, a_2, \ldots, a_L)$ and $(b_1, b_2, \ldots, b_L)$.

- **Phase 1:** Generate a pair of target nodes $(\bar{x}, \bar{y})$ which are the root of the largest trees in the largest cycles in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ (refer to Sect. 6.1).

  Besides, generate two independent 2^r-multi-collisions $\mathcal{M}_{\texttt{MC1}}$ and $\mathcal{M}_{\texttt{MC2}}$ starting from these two target nodes $\bar{x}$ and $\bar{y}$ and ending at $\hat{x}$ and $\hat{y}$ respectively.

- **Phase 2:** Build a cascade simultaneous expandable message $\mathcal{M}_{\texttt{SEM}}$ starting from $\hat{x}$ and ending at $\tilde{y}$ across the two passes, such that its length could extend up to L' (a chosen length for the second-preimage message, denote $L' = 2^{l'}$) (refer to Sect. 2.4.5).

- **Phase 3:** Find a message block $\bar{m}$ mapping the final state $\hat{y}$ of the second multi-collision to one of the chaining values b_p in the second pass of the original message, then compute the corresponding state $\tilde{x}$ from a_p with $\bar{m}$ in the first pass.

- **Phase 4:** Find a pair of starting nodes $(\check{x}, \check{y})$ in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ such that they reach the pair of target nodes $(\bar{x}, \bar{y})$ at a common distance d (or at distances of which the bias is correctable by difference of the two cycle lengths). That is done as follows: First, launch a look-ahead procedure to develop more (2^t) nodes in $\mathcal{FG}_{f_1}$ and $\mathcal{FG}_{f_2}$ and record the distance information similar to the previous attack. Then, exploit messages in $\mathcal{M}_{\texttt{MC1}}$ and $\mathcal{M}_{\texttt{MC2}}$ to get two independent sets (of size 2^r) of starting nodes, compute and store with their distances from the targets $\bar{x}$ and $\bar{y}$ in two tables $\mathcal{T}_1$ and $\mathcal{T}_2$ independently. Finally, launch a meet-in-the-middle procedure to find a match on the distance d between $\mathcal{T}_1$ and $\mathcal{T}_2$. Retrieve the corresponding message $M_2 \in \mathcal{M}_{\texttt{MC2}}$ and $M_1 \in \mathcal{M}_{\texttt{MC1}}$. This phase is accelerated by using results of the look-ahead procedure.

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 M': $m_1 \| \ldots \| m_p \| M_2 \| [m]^d \| M_1 \| M_{\|L'-p-2r-d}$.

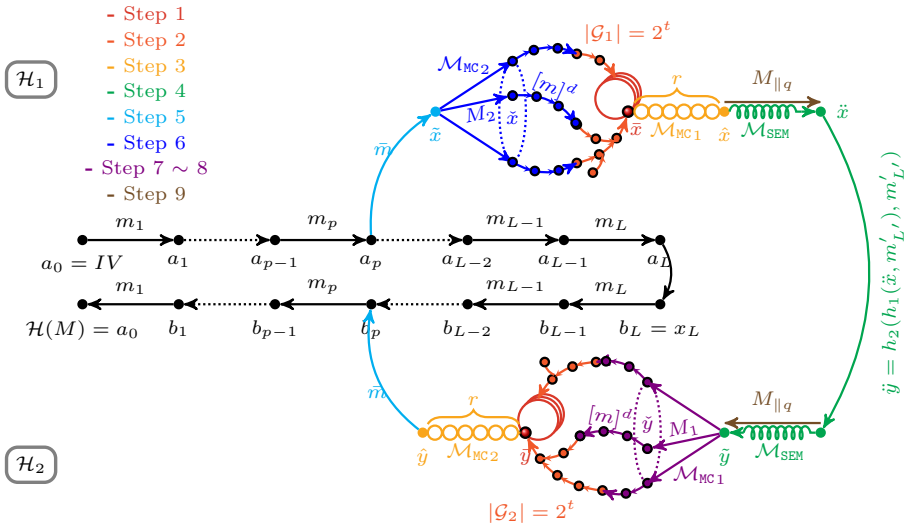


Figure 25: Second-preimage attack on Zipper hash based on multi-cycles

The time complexity of this attack is dominated by that of Phases 2, 3 and 4. In Phase 2, time complexities of building the simultaneous expandable message is about $2^{l'}$. In Phase 3, time complexities of connecting $\bar{y}$ with b_p is about 2^{n-l}. In Phase 4, time complexities of the look-ahead procedure is about 2^t, and that of the meet-in-the-middle procedure to find a pair of $(\check{x}, \check{y})$ connecting to the target nodes $(\bar{x}, \bar{y})$ at a common distance

(or with distances of which the bias is correctable by difference of the two cycle lengths) is about $2^r \cdot 2^{n-t} = 2^{r+n-t}$ (where value of parameter r is discussed below). Within Phase 4, we can directly balance the time complexities of the look-ahead procedure with that of the meet-in-the-middle procedure by setting $t = r + n - t$, which gives $t = r/2 + n/2$.

If there is nolimitation on the maximum message length, one can use multi-cycles technique to improve the complexity of Phase 4. Required number of trials to find $(\check{x}, \check{y})$ is $2^{2r} = 2^{2n-3n/2-(l'-n/2)}$. Thus, $r = n/2 - l'/2$. In this case, the complexity of each phase is:

Phase 1: $2^{n/2}$ **Phase 2:** $2^{l'}$ **Phase 3:** 2^{n-l} **Phase 4:** $2^{3n/4-l'/4}$

- If $l \geq 2n/5$, we set $l' = 3n/5$ to balance Phases 2 and 4. The total complexity is $2^{3n/5}$.
- If $3n/8 \leq l < 2n/5$, we set $l' = n - l$ to balance Phases 2 and 3. The total complexity is 2^{n-l}.
- If $l < 3n/8$, we set $l' = 3n/8$. The total complexity is dominated by Phase 3, which is 2^{n-l}.

If the maximum message length is limited to be not longer than $2^{n/2}$, we pick $2^{n/2}$ to be the length of the second-preimage to optimize the complexity. Required number of trials to find $(\check{x}, \check{y})$ is $2^{2r} = 2^{2n-3n/2}$. Thus, $r = n/4$. The complexity of each phase is as follows:

Phase 1: $2^{n/2}$ **Phase 2:** $2^{n/2}$ **Phase 3:** 2^{n-l} **Phase 4:** $2^{5n/8}$

In this case, the total complexity is $2^{5n/8}$ for $3n/8 \leq l < n/2$, and 2^{n-l} for $0 < l < 3n/8$.

6 Discussions, Summary and Open Problems

6.1 Relations Between Properties Utilized in Various Attacks and Properties of Functional Graphs

Almost all presented attacks consist of some probabilistic algorithms restoring part of the functional graph. They exploit the entropy loss phenomenon in the output of the corresponding probabilistic algorithm. Here, we characterize these probabilistic algorithms from their external behaviors (like black-boxes), in order to highlight that they all lose an amount of entropy in their output (compare with the entropy in their uniformly random auxiliary input). The amount of entropy they lost is closely related to the properties of the functional graph, and is a key factor affecting the efficiencies of the corresponding attacks:

- Cycle search algorithm: A probabilistic algorithm with a uniformly random variable $x \xleftarrow{\$} \{0, 1, \cdots, 2^l - 1\}$, outputs $f^i(x)$ and $j - i$ where $j \geq i$ and j is the minimum index such that $f^j(x) = f^i(x)$. Entropy loss in the output of this algorithm is about l bits (two outputs collide with constant probability). We explain more in Sect. 6.1.1.
- Chain evaluation algorithm: A probabilistic algorithm with a uniformly random variable $x \xleftarrow{\$} \{0, 1, \cdots, 2^l - 1\}$, outputs $f^{2^s}(x)$. Entropy loss in the output of this algorithm is about s bits (two outputs collide with probability 2^{s-l}). We explain more in Sect. 6.1.2.
- Collision search algorithm: A probabilistic algorithm with independent uniformly random variables $x,\ y \xleftarrow{\$} \{0, 1, \cdots, 2^l - 1\}$, outputs $f^i(x)$ if j is the minimum index

such that $f^i(x) = f^j(y)$ and $c \cdot 2^s \leq i,\ j \leq 2^s$ (where c is a constant factor, say 0.25). Entropy loss in the output of this algorithm is about $2s$ bits (two outputs collide with probability 2^{2s-l}). We explain more in Sect. 6.1.3.

For brute force algorithm, the entropy is full (2^l). Loosely speaking, for shortcut approaches, the more entropy lost in a utilized probability event, the higher efficiency the attack achieves. An intuitive explanation is that the loss of entropy reduces the real value of the birthday bound to find a match. For example, for cycle-based attacks, the probability for a certain output to occur is close to be constant and the entropy almost reduces to 0. Cycle-based attacks achieve the best efficiency. The computation complexities of presented attacks based on the reduction of image-set size and attacks based on collisions are directly related to the amount of entropy lost in the output of the chain evaluation algorithm or the collision search algorithm.

On the other hand, all output values of the discussed probabilistic algorithms are essentially special nodes in the functional graph. Special nodes exploited by the presented attacks can be classified as follows:

- **Cyclic nodes** Located in cycles of the functional graph, these nodes provide efficiency for both generic attacks on hash-based MACs and generic attacks on hash combiners. Detecting cycle and collecting cyclic nodes require computations of birthday bound, which is relatively fast. However, message length is also of birthday bound. They are essentially the most special nodes. Lots of them are both deep iterate nodes and collision nodes.

- **Deep iterate nodes** Located (deep) in the low stratum of the functional graph, these nodes provide trade-offs between the time complexity and the message length of presented attacks. Generally speaking, the deeper these iterate nodes locate, the higher efficiency the attacks achieve, at the same time, the longer the message length is required.

- **Deep collision nodes** The collision-based attacks presented in this paper essentially use the collisions located (deep) in the low stratum of the functional graph. We discuss this in depth in Sect. 6.1.3. Like deep iterate nodes, deep collision nodes can also provide trade-offs between the time complexity and the message length for presented attacks. These deep collision nodes are images of two deep iterates. Entropy on k-th deep collision nodes (see Definition 2) is strictly less than entropy on k-th deep iterate nodes. Additionally, entropy reduces more rapidly with the increasing of k for k-th deep collision nodes than the entropy reducing speed for k-th deep iterate nodes. That brings advantage for collision-based attacks when the maximum length of messages is limited to be small.

We note that the essential reason for entropy loss in the output of the cycle search algorithm, the reason for entropy loss in the output of the chain evaluation algorithm, and the reason for entropy loss in the output of the collision search algorithm are closely related. All of them can be uniformly explained using properties of functional graph of random mappings.

6.1.1 Relation Between Entropy Loss of the Cycle Search Algorithm and Properties of the Functional Graph

The cycle search algorithm (also known as cycle detection, cycle finding [Jou09, Chapter 7]) has been widely used in cryptanalysis. The relation between entropy loss in the output of the cycle search algorithm and properties of the functional graph has also been noticed.

On the one hand, from probabilistic argument and the birthday paradox, one knows that after about $2^{n/2}$ iterations on a random mapping, the computation chain enters a

cycle. On the other hand, from the known properties of the functional graph, we can get further support for this statement. Additionally, properties of the functional graph indicate that by running the cycle search algorithm several times, we can get the cycle in the giant component in the functional graph of the mapping. Besides, we can also locate the root of the largest tree. That is because, in the functional graph of a random mapping, the giant component is expected to occupy a fraction of 0.75782 of the total nodes, the largest tree 0.48 (refer to Theorem 5). All nodes in the giant component can reach the cycle, and they are expected to reach the cycle after iterating the function about $2^{n/2}$ times because the expected tail length and the expected rho length are respectively $0.62 \cdot 2^{n/2}$ and $1.2 \cdot 2^{n/2}$ (refer to Theorem 2). Besides, nodes in the largest tree reach the cycle through the root of the largest tree. Thus, if we run the cycle search algorithm starting from a randomly selected node, with a probability about 0.75782 the obtained cycle is the cycle in the largest component. Note that $0.75782 + 0.48 > 1$, therefore the largest tree is in the largest component with asymptotic probability one. Assume the cycle we get is indeed the one in the largest component, then with probability about $0.48/0.75782 \approx 0.63$ the α-node we get is the root of the largest tree [LPW13].

6.1.2 Relation Between Entropy Loss of the Chain Evaluation Algorithm and the Number of k-th Iterates In the Functional Graph

The chain evaluation is simply starting from a random point and iteratively evaluating the mapping f up to a certain number k of times, i.e., evaluation of f^k (more generally, the chain evaluation can be done on various mappings, i.e., $g_k = f_k \circ \cdots f_1$, which is applicable for HAIFA mode). The reduction of the image-set size of f^k with increasing of k can be illustrated by statistical properties of the functional graph of f. The image-set size of f^k is directly related to the expected number of k-th iterates in the functional graph of f.

As stated in Theorem 1, the expectation of the number of k-th iterate image nodes in a functional graph of a random mapping of size N has the asymptotic form $(1 - \tau_k)N$, as $N \to \infty$, where the τ_k satisfies the recurrence $\tau_0 = 0$ and $\tau_{k+1} = e^{-1+\tau_k}$. We refer to [RS67] in which one can find a precise estimate of the ratio τ_k. Accordingly,

Lemma 4. *Let f be a random mapping in $\mathcal{F}_N$. Denote $N = 2^n$. For $k \leq 2^{n/2}$, the expectation of number of k-th iterate image nodes in the functional graph of f is*

$$(1 - \tau_k) \cdot N \approx (\frac{2}{k} - \frac{2}{3}\frac{\log k}{k^2} - \frac{c}{k^2} - \cdots) \cdot N.$$

It suggests that $\lim_{k\to\infty} k \cdot (1 - \tau_k) = 2$. *Thus,*

$$\lim_{N\to\infty, k\to\infty, k\leq\sqrt{N}} (1 - \tau_k) \cdot N \approx 2^{n-\log_2(k)+1},$$

where τ_k satisfies the recurrence $\tau_0 = 0$, $\tau_{k+1} = e^{-1+\tau_k}$, and c is a certain constant.

We note that Lemma 4 is essentially consistent with Lemma 1 and 2, which also appears in different forms in many previous literature [GK11, GJMN15, DL17, Din16]. Here, Lemma 4 provides a more precise estimation on the entropy loss of the chain evaluation algorithm based on statistical properties of the functional graph of random mappings, which originate from analytic combinatorics (the use of the symbolic method, generating functions and singularity analysis) [FO89, FS09].

6.1.3 Relation Between Entropy Loss of the Collision Search Algorithm and Properties of the Functional Graph

The presented collision-based generic attacks on MD hash-based MACs essentially exploit entropy loss in the output of the collision search algorithm (Algorithm 2).

Authors in [DL17] provided a proof for the probability of two outputs of the collision search algorithm being equal (Lemma 3). Suppose the setting of the collision search algorithm in Lemma 3 is to compute chains of length 2^s to search for collisions (same-offset or free-offset) with a fixed n-bit random function f. In the proof provided in [DL17], an assumption is required (the assumption is thought to be fulfilled with constant probability), i.e., the offset of one collision $\hat{x}$ is a constant fraction of 2^s. That implicitly requires that this considered collision is a collision of images of $\Theta(2^s)$-iterate image nodes (this is the reason why we imposed a restriction $c \cdot 2^s \leq i, j \leq 2^s$ on the offset i, j of the collision at the beginning of Sect. 6.1). In the following, we name the collisions fulfilling this assumption explicitly. In that, we investigate them from another point of view, more specifically, from the properties of the functional graph of random mappings. Thereby, we show the relation between the entropy loss in the output of the collision search algorithm and the properties of the functional graph.

Here, we explicitly propose the concept of k-th iterate collision node to represent collisions fulfilling the assumption in the proof of Lemma 3 in [DL17].

Definition 2 (k-th iterate collision node)**.** A k-th iterate collision node in the functional graph of a random mapping $f \in \mathcal{F}_N$, is an r-node (a node of in-degree r), where $r \geq 2$ and at least two of its pre-images are k-th iterate image nodes.

Denote the set of k-th iterate collision node by A_k. We have, all nodes in A_k are also in A_{k-1}. And clearly, a k-th iterate collision node is also a $(k+1)$-th iterate image node. The expected number of k-th iterate collision nodes is determined by the expected number of k-th iterate image nodes. Before explicitly presenting this, an additional lemma is required.

Lemma 5. *The expected total number of collision nodes (0-th iterate collision nodes) in the functional graph of a random mapping $f \in \mathcal{F}_N$ is $(1 - 2 \cdot e^{-1}) \cdot N = 0.2642 \cdot N$.*

Proof. Collision nodes are r-nodes (nodes of in-degree r) where $r \geq 2$ in the functional graph of f. From Theorem 3 in Sect. 3, numbers of 0-nodes and 1-nodes are both $e^{-1} \cdot N$, which directly gives the lemma. □

Lemma 6. *Denote $N = 2^n$. For $N \to \infty$, $k \to \infty$ and $k \leq 2^{n/2}$, the expected number of k-th iterate collision nodes in the functional graph of a random mapping $f \in \mathcal{F}_N$ is $\Theta(k^{-2} \cdot N)$.*

Proof. Let x be a node selected uniformly at random. Denote the set of k-th iterate image nodes by B_k. Note that, according to the definition of k-th iterate image node (Def.1), all nodes in B_k are also in B_{k-1}. We have $|B_0| = 2^n$ and $\Pr[x \in B_0] = 1$. According to Lemma 4, $\Pr[x \in B_k] \approx 2/k$ for large k. Denote the set of k-th iterate collision node by A_k. Then, according to the definition of k-th iterate collision node and Lemma 5, $\Pr[x \in A_0] = \Pr[x \text{ is a collision node}] = 1 - 2 \cdot e^{-1}$. If x is a collision node, we denote two of it preimages by x_{p_1} and x_{p_2}. Then, we have

$$
\begin{aligned}
&\Pr[x \in A_k] = \Pr[x_{p_1} \in B_k \ \&\ x_{p_2} \in B_k \mid x \in A_0] \cdot \Pr[x \in A_0] \\
&\Pr[x \in A_k] = \Pr[x_{p_1} \in B_k \ \&\ x_{p_2} \in B_k \mid x_{p_1} \in B_0 \ \&\ x_{p_2} \in B_0] \cdot \Pr[x \in A_0] \\
&\Pr[x \in A_k] = \Pr[x_{p_1} \in B_k \ \&\ x_{p_2} \in B_k] \cdot \Pr[x \in A_0] \\
&\Pr[x \in A_k] \approx \Pr[x_{p_1} \in B_k] \cdot \Pr[x_{p_2} \in B_k] \cdot \Pr[x \in A_0] \\
&\Pr[x \in A_k] \approx (2/k)^2 \cdot (1 - 2 \cdot e^{-1}) \approx k^{-2}
\end{aligned}
$$

Accordingly, the expected number of k-th iterate collision nodes is $\Theta(k^{-2} \cdot N)$. □

Considering an extreme case to look into this lemma: When $k = 2^s$ and $s \to n/2$, according to Lemma 4, the number of k-th iterate image nodes keeps stable in $2^{n-s} \to 2^{n/2}$

which is the expected number of cyclic nodes. And the number of k-th iterate collision nodes is $2^{n-2s} \to \Theta(1)$. That is reasonable, because collision implies entropy loss. When there is no collision, entropy is conserved.

From Lemma 6, if two k-th iterate collisions are found independently with a random n-bit mapping f, then they are equal with probability $\Theta(k^2 \cdot 2^{-n})$.

Next, we discuss the applicability of Lemma 6 in the setting of collision-based attacks to provide ground for complexity analysis. Both of the same-offset collisions and the free-offset collisions found by corresponding collision search algorithms can be k-th iterate collision nodes, as long as their offsets concerning the two colliding chains are both larger than k. Note that, when analyzing the complexity of the collision-based attacks according to Lemma 6, one will find limitations on the applicability of Lemma 6. The first limitation is that it is only applicable when the target hash-based MACs being of Merkle-Damgård construction (for HAIFA construction, refer to [DL17] for analysis on entropy loss in the output of the same-offset collision search). The second limitation is that it is only applicable when among all collisions found by the collision searching algorithm using chains of length 2^s, there is a constant fraction of them being $\Theta(2^s)$-th iterate collision node (for both same-offset collisions and free-offset collisions). Recall the assumption in the original proof which states that the offset of a collision is uniformly distributed in the interval $[0, 2^s]$. This assumption implies the fulfillment of the requirement. However, this assumption is not true in general. The number of collisions at the beginning of those chain evaluations are more than that at the end. That is not hard to be understood considering that images at the beginning of those chains are more than that at the end due to entropy loss. The more nodes, the more likely to occur collisions in the next iteration. Thus, to meet the assumption, a restriction on the number of chains (denoted by 2^t) and the length of the chains (denoted by 2^s) is necessary. To make the observation on this phenomenon clearer, we have also performed experiments to evaluate the distribution of offset of same-offset collisions. Results show that when t and s are large, the number of collisions at each offset decreases more obviously with the increase of the value of offset (an extreme case is when $t = n$, number of collisions at small offsets are significantly larger than that at large offsets). When t and s are relatively small (e.g., when $t = 5n/8$ and $s = n/8$), this decreasing trend is not so obvious, and offset of the collisions can be roughly viewed as uniformly distributed in the interval $[0, 2^s]$ [4]. Actually, the primary restriction on t and s in the collision search algorithm stated in Lemma 3, i.e. $t + 2s < n$, is sufficient to meet the assumption. Under this restriction, each evaluated chain is not expected to collide with more than one different chain, thus collisions can be seen to be independent. As noted in the proof of Lemma 3 in [DL17], Lemma 1 shows that increasing the length of the chains increases the collision probability (at a common offset) by the same multiplicative factor. When collisions can be seen to be independent ($t + 2s < n$), the offset of the collision can be seen to be uniformly distributed in the interval $[0, 2^s]$. Thus, a constant fraction of the collisions obtained by running the collision search algorithm are $\Theta(2^s)$-th iterate collision nodes, and thus Lemma 6 is applicable in the complexity analysis of collision-based attacks.

6.1.4 Remarks on Approaches from Analytic Combinatorics

Lots of statistic results on functional graphs of random mappings have been deduced using approaches from analytic combinatorics (refer to Sect. 3 and [FO89, FS09]). Meanwhile, in previous generic cryptanalysis, properties of probabilistic algorithms related to the random mappings are usually derived using probabilistic arguments. Conclusions from these two independent approaches are usually consistent. However, analytic combinatorics (the symbolic method, generating functions and asymptotic analysis) is more powerful than probabilistic arguments when the corresponding amount is beyond the birthday

[4] Refer to https://github.com/FreeDisciplina/CollisionOffset for the source code of this experiment and more detailed experimental results.

bound. The applicability of probabilistic arguments is sometimes based on assumptions on independence between analyzed probability events (otherwise, complex correlations will have to be concerned). Whereas, using probabilistic argument is more convenient. As has been shown in the previous paragraph, conclusions obtained from probabilistic arguments can be uniformly obtained from known statistical results on functional graphs.

A natural question is: "Is it possible to use approaches from analytic combinatorics to directly get a recursive asymptotic formula for the expected number of k-th iterate collision nodes, or for more generalized concepts, such as collision nodes that at least one of their preimages is k-th iterates, or cyclic nodes which are also collision nodes and deep iterates?" Another question is: "Is it possible to build discrete combinatorial models for other concerned objects (e.g., the functional graph of constrained mappings, the particle functional graph restored by some probabilistic algorithm, iterations on distinct random functions) and then apply approaches from analytic combinatorics to obtain a quantitative estimation for the concerned parameters, such that they can be used in cryptanalysis on those objects?" A particular example is: "Is it possible to build a combinatorial model for the data structure constructed using Algorithm 1 to obtain a theoretical proof for Conjecture 2 (on height distribution)?" If these are possible, it will be great that more seemingly scattered enumerating problems in generic cryptanalysis can be uniformly solved systematically based on a solid theoretical foundation. We leave these to be a future work.

Experimental Verification on Statistical Properties of Functional Graph A practical question is: "For real world pseudo-random mappings designed by cryptographers, how the properties of their functional graph diverse from those statistical properties of the functional graph of random mappings, which is deduced using approaches from analytic combinatorics?" To answer this question, we performed experiments by simulating a few of n-bit random mappings with chopped AES-128 (obtained by fixing an arbitrary key and $128 - n$ bits of the input and taking n bits as the output, $n \in \{12, \ldots, 28\}$). For each $n < 26$, we sample hundreds of random mappings[5], and examined the average value, the maximum value, the minimum value with respect to parameters considered in Theorem.1, e.g., their number of cyclic nodes and their number of k-th iterates. We saw consistency between the experimental results and the theoretical ones. For example, for 24-bit chopped AES-128, deviation between the experimental average number of image nodes (denoted by $\mathbf{E}_{\text{exp}}\{\text{images}\}$) and the theoretical one (denoted by $\mathbf{E}_{\text{thm}}\{\text{images}\}$) is $|\mathbf{E}_{\text{exp}}\{\text{images}\} - \mathbf{E}_{\text{thm}}\{\text{images}\}| \approx 2^{-20.31}\mathbf{E}_{\text{thm}}\{\text{images}\}$ [6]. That shows the power of the approaches from analytic combinatorics.

6.2 Summary on Generic Attacks against Hash-based MACs

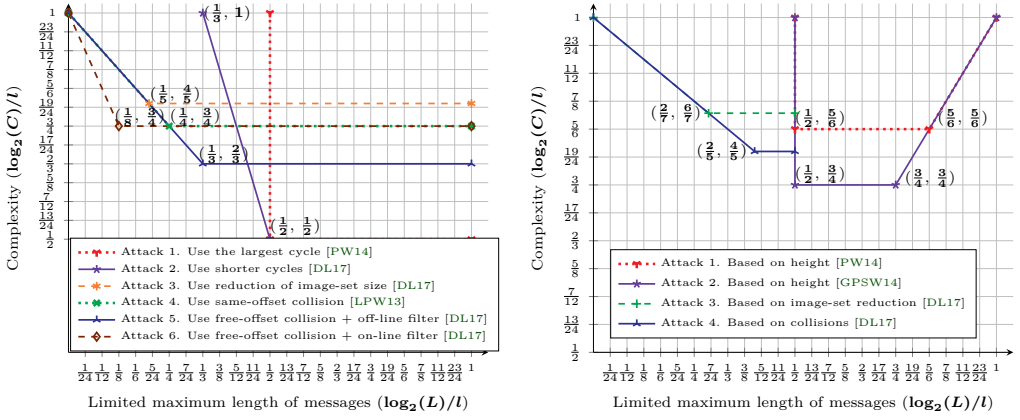
As noted above, the presented attacks on hash-based MACs are based on attributes of three types of probabilistic algorithms restoring part of the functional graph. These attacks simulate the off-line probabilistic algorithm by querying the oracle. The advantages of the adversaries depend on how efficient the probabilistic algorithms reduce the uncertainty on their output, such that the on-line simulation could also capture the same certainty. We use entropy to represent the amount of uncertainty. By detecting the overlap between the output of off-line probabilistic algorithm and that of the on-line simulation, one can get information on the on-line simulation. The loss of entropy reduces the real value of the birthday bound to detect an overlap. The more entropy loss in the output of the used probabilistic algorithm, the more efficiency the attacks achieve.

Here we summarize results of those presented generic attacks against hash-based MACs by drawing trade-off curves between the complexities and the maximum limited length of

[5] For $n >= 26$, we can only sample several random mappings due to limited computing resource

[6] Refer to https://github.com/FreeDisciplina/FunctionalGraphStatistics for the source code of the experiment and more detailed experimental results.

queries (see Fig.26a and Fig.26b). From these figures, we can see that attacks based on the reduction of image-set size do not have an advantage over attacks based on entropy loss of collisions. Notably, the universal forgery attack based on the reduction of image-set size was originally thought to have some advantage compared with the universal forgery attack based on collisions when queries are shorter than $2^{2l/7}$. However, from Fig.26b (curve of Attack 3 and curve of Attack 4) we can see, this is a wrong impression. The induction that leads this wrong impression, is possibly because in the description of the Attack 3, length of queries is denoted by 2^{2s} instead of by 2^s like that in the description of Attack 4. Roughly speaking, the reason those attacks based on the reduction of image-set size are less efficient than attacks based on entropy loss of collisions, is that, the former has to either build more filters or test more filters to detect overlap between off-line nodes and on-line nodes (note that number of matching pairs is 2^{l-s} for the former and 2^{l-2s} for the latter). Comparing between the cycle-based attacks and the attacks based on collisions, we can find that: the cycle-based attacks are generally more efficient. Reasons include that, there is no need to build a filter to make a match, the cycle length is a good filter, and the α-node can be efficiently detected both off-line and on-line. Additionally, we can use multi-cycles to correct difference of length of two distinct messages. On the other hand, disadvantage of the cycle-based attacks is also notable. That is, they are inapplicable for short messages. For short messages, the most efficient attacks are those collision-based attacks. However, there are also inherent limitations on the efficiency of those collision-based attacks. Collision-based attacks always evaluate a large number of internal states to collect enough off-line collisions and on-line collisions such that there exists an overlap; they also have to build filters and match using these filters for a large set of states to detect the overlap between on-line collisions and off-line collisions; These are expensive and are the performance bottlenecks of these attacks.



(a) State recovery attacks **(b)** Universal forgery attack

Figure 26: Relation curves between the limited maximum length of messages and the complexity of attacks on MACs

6.3 Summary on Generic Attacks against Hash Combiners

Here, we summarize results of presented generic attacks on hash combiners by drawing trade-off curves between the complexity and the length of the message (see Fig.27). From this figure, we can see that for MD XOR combiner and MD Zipper hash, security upper bounds provided by these generic attacks are quite close to the security lower bounds ($2^{n/2}$) regarding (second) preimage resistance. However, when limiting the length of message, security upper bounds remain high enough. That is mainly due to the limitation of

the functional-graph-based attacks which generally iterate the underlying compression function many times thus resulting long messages. Is it possible, for short message, to launch efficient attacks on hash combiners by exploiting other special nodes besides those deep-iterates in the functional graph? Is it possible, for long messages, to completely eliminate the gap between the security upper bound and the lower bound of these hash combiners by further exploiting the properties of random functional graph? We leave these as open problems.

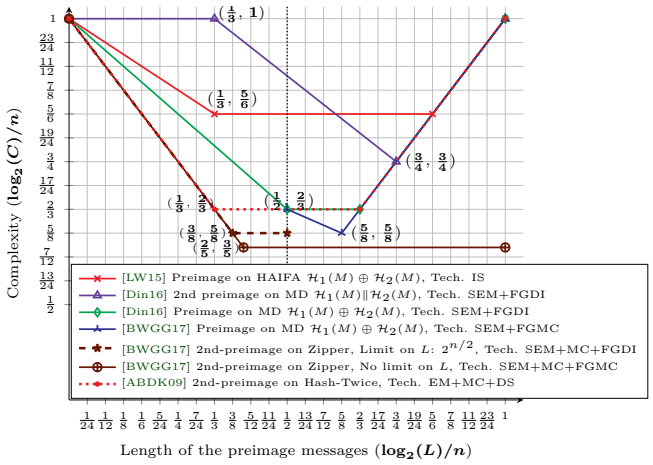


Figure 27: Trade-offs between message length and attack complexity for hash combiners

Acknowledgments

The authors would like to thank the reviewers of FSE 2018 for their detailed comments and valuable suggestions. Special thanks go to Pierre Karpman and Maria Eichlseder for their careful reading and thorough comments which helped us improve the manuscript significantly. Lei Wang is sponsored 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 *LNCS*, 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 *LNCS*, pages 270–288. Springer, 2008.

[AD99] Christopher Allen and Tim Dierks. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[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 *LNCS*, pages 570–583. Springer, 2006.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *LNCS*, pages 1–15. Springer, 1996.

[BD07] Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. Cryptology ePrint Archive, Report 2007/278, 2007. http://eprint.iacr.org/2007/278.

[Bra90] Gilles Brassard, editor. *Advances in Cryptology - CRYPTO' 89*, volume 435 of *LNCS*. 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 *LNCS*, pages 404–427. Springer, 2017.

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

[Dea99] Richard Drews Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University Princeton, 1999.

[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 *LNCS*, 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.

[DL17] Itai Dinur and Gaëtan Leurent. Improved Generic Attacks Against Hash-Based MACs and HAIFA. *Algorithmica*, 79(4):1161–1195, 2017.

[DR06] Tim Dierks and Eric Rescorla. The transport layer security (TLS) protocol version 1.1. RFC 4346, April 2006.

[FKK11] Alan O. Freier, Philip Karlton, and Paul C. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101, August 2011.

[FL07] Marc Fischlin and Anja Lehmann. Security-Amplifying Combiners for Collision-Resistant Hash Functions. In 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 *LNCS*, pages 224–243. Springer, 2007.

[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 *LNCS*, pages 375–392. Springer, 2008.

[FLP08] Marc Fischlin, Anja Lehmann, and Krzysztof Pietrzak. Robust Multi-property Combiners for Hash Functions Revisited. In 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 *LNCS*, pages 655–666. Springer, 2008.

[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 *LNCS*, pages 329–354. Springer, 1989.

[FS09] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. cambridge University press, 2009.

[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 *LNCS*. Springer, 2014.

[GJMN15] Jian Guo, Jérémy Jean, Nicky Mouha, and Ivica Nikolić. More Rounds, Less Security? Cryptology ePrint Archive, Report 2015/484, 2015. http://eprint.iacr.org/2015/484.

[GK11] Danilo Gligoroski and Vlastimil Klima. Practical Consequences of the Aberration of Narrow-Pipe Hash Designs from Ideal Random Functions. In Marjan Gusev and Pece Mitrevski, editors, *ICT Innovations 2010: Second International Conference, ICT Innovations 2010, Ohrid Macedonia, September 12-15, 2010. Revised Selected Papers*, volume 83 of *CCIS*, pages 81–93, Berlin, Heidelberg, 2011. Springer.

[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.

[JN15] Ashwin Jha and Mridul Nandi. Some Cryptanalytic Results on Zipper Hash and Concatenated Hash. Cryptology ePrint Archive, Report 2015/973, 2015. http://eprint.iacr.org/2015/973.

[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 *LNCS*, pages 306–316. Springer, 2004.

[Jou09] Antoine Joux. *Algorithmic cryptanalysis*. CRC Press, 2009.

[KBPH06a] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1. Cryptology ePrint Archive, Report 2006/187, 2006. http://eprint.iacr.org/2006/187.

[KBPH06b] Jongsung Kim, Alex Biryukov, Bart Preneel, and Seokhie Hong. On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1 (Extended Abstract). In Roberto De Prisco and Moti Yung, editors, *Security and Cryptography for Networks, 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006, Proceedings*, volume 4116 of *LNCS*, pages 242–256. Springer, 2006.

[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 *LNCS*, pages 183–200. Springer, 2006.

[KK13] Tuomas Kortelainen and Juha Kortelainen. On Diamond Structures and Trojan Message Attacks. In Sako and Sarkar [SS13], pages 524–539.

[KR95] Burt Kaliski and Matt Robshaw. Message authentication with MD5. *CryptoBytes, Sping*, 1995.

[KRS88] Burton S. Kaliski, Ronald L. Rivest, and Alan T. Sherman. Is the Data Encryption Standard a group? (Results of cycling experiments on DES). *Journal of Cryptology*, 1(1):3–36, Jan 1988.

[KS05] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In 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 *LNCS*, pages 474–490. Springer, 2005.

[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 *LNCS*, pages 358–375. Springer, 2006.

[LPW13] Gaëtan Leurent, Thomas Peyrin, and Lei Wang. New Generic Attacks against Hash-Based MACs. In Sako and Sarkar [SS13], pages 1–20.

[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 *LNCS*, pages 345–367. Springer, 2015.

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

[Mut88] Ljuben R Mutafchiev. The limit distribution of the number of nodes in low strata of a random mapping. *Statistics & Probability Letters*, 7(3):247 – 251, 1988.

[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 *LNCS*, 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 *LNCS*, pages 413–432. Springer, 2008.

[Pro74] G. V. Proskurin. On the Distribution of the Number of Vertices in Strata of a Random Mapping. *Theory of Probability & Its Applications*, 18(4):803–808, 1974.

[PSW12] Thomas Peyrin, Yu Sasaki, and Lei Wang. Generic Related-Key Attacks for HMAC. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, volume 7658 of *LNCS*, pages 580–597. Springer, 2012.

[PvO95] Bart Preneel and Paul C. van Oorschot. MDx-MAC and Building Fast MACs from Hash Functions. In Don Coppersmith, editor, *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *LNCS*, pages 1–14. Springer, 1995.

[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 *LNCS*, pages 147–164. Springer, 2014.

[RS67] A. Rényi and G. Szekeres. On the height of trees. *Journal of the Australian Mathematical Society*, 7(4):497–507, 11 1967.

[SS13] 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 *LNCS*. Springer, 2013.

[Tsu92] Gene Tsudik. Message authentication with one-way hash functions. In *Proceedings IEEE INFOCOM '92, The Conference on Computer Communications, Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies, One World through Communications, Florence, Italy, May 4-8, 1992*, pages 2055–2059. IEEE, 1992.

[Yas07] Kan Yasuda. "Sandwich" Is Indeed Secure: How to Authenticate a Message with Just One Hashing. In Josef Pieprzyk, Hossein Ghodosi, and Ed Dawson, editors, *Information Security and Privacy, 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, volume 4586 of *LNCS*, pages 355–369. Springer, 2007.