Improving Generic Attacks Using Exceptional Functions

Xavier Bonnetain[1], Rachelle Heim Boissier[2], Gaëtan Leurent[3], and André Schrottenloher[4]

[1] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
firstname.lastname@inria.fr
[2] Université Paris-Saclay, UVSQ, CNRS,
Laboratoire de mathématiques de Versailles, Versailles, France
firstname.lastname@uvsq.fr
[3] Inria, Paris, France
firstname.lastname@inria.fr
[4] Univ Rennes, Inria, CNRS, IRISA, Rennes, France
firstname.lastname@inria.fr

Abstract. Over the past ten years, there have been many attacks on symmetric constructions using the statistical properties of random functions. Initially, these attacks targeted iterated hash constructions and their combiners, developing a wide array of methods based on internal collisions and on the average behavior of iterated random functions. More recently, Gilbert et al. (EUROCRYPT 2023) introduced a forgery attack on so-called *duplex-based* Authenticated Encryption modes which was based on *exceptional* random functions, i.e., functions whose graph admits a large component with an exceptionally small cycle.
In this paper, we expand the use of such functions in generic cryptanalysis with several new attacks. First, we improve the attack of Gilbert et al. from $\mathcal{O}(2^{3c/4})$ to $\mathcal{O}(2^{2c/3})$, where c is the capacity. This new attack uses a nested pair of functions with exceptional behavior, where the second function is defined over the cycle of the first one. Next, we introduce several new generic attacks against hash combiners, notably using small cycles to improve the complexities of the best existing attacks on the XOR combiner, Zipper Hash and Hash-Twice.
Last but not least, we propose the first quantum second preimage attack against Hash-Twice, reaching a quantum complexity $\mathcal{O}(2^{3n/7})$.

Keywords: Cryptanalysis · Generic attack · Duplex-based modes · Hash Combiners · Random Functions

1 Introduction

The landscape of symmetric cryptanalysis is often divided into two categories: attacks that consider a specific primitive, and *generic attacks* which target the

way this primitive is used. In this paper, we study generic attacks on authenticated encryption and hashing modes based on a public function.

Authenticated encryption (AE) modes allow to both encrypt and authenticate a message. Often, such modes allow the option to authenticate some extra public data, the *associated data*. Such modes are then called *Authenticated encryption with associated data* (AEAD). Among these, we will consider a family of permutation-based modes based on the Duplex construction [8,14] which we refer to as *Duplex-based modes*. Such modes have an undisputed popularity: several candidates to the NIST lightweight cryptography standardization process used a Duplex-based mode and Ascon, the winner, is a variant of Duplex.

Recently, Gilbert et al. introduced a generic forgery attack on Duplex-based modes, which reduced an existing gap in the security proof [25]. The attack starts by rewriting the decryption of the Duplex as an iteration of a random function, obtained by truncating the underlying permutation of the mode. It then balances a precomputation phase in which an *exceptional* random function is found, and a computation phase in which the adversary produces a *forgery*, i.e. a valid ciphertext for a message that was never given to the encryption oracle.

Since then, [35] has shown that for certain parameter choices this attack matches the security bound. However, with those parameters, the online complexity of the attack is brought up to $\mathcal{O}(2^c)$, *i.e.* the attack is not more efficient than exhaustive search. As well explained in [35], '*the gap here lies in the way queries are counted, as the security proofs assume that a permutation evaluation done with one path is counted only once*'. This illustrates that provable security struggles to capture the actual cost of attacks, which is in practice the most crucial factor for security in real-life contexts. It is thus essential to understand the security of Duplex-based modes by the means of cryptanalysis.

In a seemingly different context, hashing modes transform a fixed-size compression function h into a full-fledged hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^d$ capable of turning a message of arbitrary length into a fixed-length digest. A hash function is expected to withstand collision, preimage and second preimage attacks, up to their respective generic complexities $\mathcal{O}(2^{d/2})$ (birthday attack) and $\mathcal{O}(2^d)$ (brute-force search).

In practice, many hash functions are built iteratively using the well-known Merkle-Damgård construction [15,39]. The message is first padded, then split into L blocks, $M = M_1||\dots||M_L$, and an internal state is initialized with the IV. For each block, the compression function h is applied to the block and the current internal state, and returns the next internal state. Finally, a finalization function g is applied. Hereafter, we consider that the padding, and in practice the function g itself, depends on the message length (*MD strengthening*)[5].

A seemingly innocuous way to increase the security of an iterated hash function is to combine two of them using a *hash combiner*. Several constructions

[5] To be more precise, the function g itself depends on the message length only when the last message block is processed before the finalization function, which is true for classical Merkle-Dåmgard and all combiners using iterated Hash functions except the Zipper Hash.

Table 1. Summary of attacks against duplex mode. k is the key size, t the tag size and c the capacity. Notation $\mathcal{O}$ is omitted.

Kind	Technique	Time	Source
Key-recovery	Brute-force	2^k	Folklore
Forgery	Brute-force	$2^{\min(t,c)}$	Folklore
	Small cycles	$2^{3c/4}$	[25]
	Nested cycles	$2^{2c/3}$	Section 3

were proposed: the concatenation combiner $\mathcal{H}(M) = \mathcal{H}_1(M)||\mathcal{H}_2(M)$, the XOR combiner $\mathcal{H}(M) = \mathcal{H}_1(M) \oplus \mathcal{H}_2(M)$ [18], cascade constructions such as Hash-Twice, $\mathcal{H}(M) = \mathcal{H}_2(\mathcal{H}_1(IV, M), M)$ and the Zipper Hash [38]. However, since Joux' collision attack on the concatenation combiner [30], it is known that such constructions do not necessarily increase security, and in fact, sometimes, they *decrease* it [37]. Typically, preimage and second preimage attacks on these combiners can be mounted using internal collisions between the two hash functions. These attacks have gradually improved over time [5,19,37], using more and more advanced techniques (a recent overview is given in [3]). These attacks are generic, and consider the compression functions of $\mathcal{H}_1$ and $\mathcal{H}_2$ as random. They make use of the average properties of the functional graphs of random functions.

Contributions. In this paper, we improve the complexity of several generic attacks based on random functions. Some of these attacks are based on a new construction which expands the *exceptional functions* already used in [25]: *nested* exceptional functions (Section 3.2). While an exceptional function is a random function with a small cycle, the nested case further considers random functions mapping the small cycle to itself and looks for an exceptional such function.

This new idea allows us to improve the forgery attack on Duplex-based modes (Section 3.3), reducing the complexity from $\mathcal{O}\left(2^{3c/4}\right)$ to $\mathcal{O}\left(2^{2c/3}\right)$, where c is the capacity. This is summarized in Table 1.

Next, we introduce several new attacks on hash combiners, which introduce for the first time exceptional functions in this context, and still improve over the best existing attacks after more than ten years of extensive analysis. We study preimage attacks on the XOR combiner (Section 5), second preimage attacks on the Zipper Hash (Section 6), second preimage attacks on Hash-Twice (Section 7). In all these cases, we start by describing simple attacks using small cycles in random functional graphs, to showcase the application of this technique. Then, we propose more intricate attacks which improve the best previous ones using small cycles. Our results are summarized in Table 2.

Finally, in Section 8, we give the first quantum second preimage attack against Hash-Twice (different attacks and combiners were considered in two previous works [4,22]). This attack reaches a quantum complexity of $\mathcal{O}\left(2^{3n/7}\right)$.

Table 2. Summary of attacks against hash combiners. n is the digest size. Notation $\mathcal{O}$ is omitted.

Target	Type	Time	Source
XOR	Preimage	$2^{5n/6} \approx 2^{0.833n}$	[37]
		$2^{2n/3} \approx 2^{0.667n}$	[19]
		$2^{5n/8} \approx 2^{0.625n}$	[5]
		$2^{11n/18} \approx 2^{0.611n}$	[3]
		$2^{3n/5} \approx 2^{0.6n}$	Section 5.2
Zipper	Second preimage, $L' \leq 2^{n/2}$	$2^{5n/8} \approx 2^{0.625n}$	[5]
	Second preimage	$2^{3n/5} \approx 2^{0.6n}$	[5]
	Second preimage	$2^{7n/12} \approx 2^{0.583n}$	Section 6.3
Hash-Twice	Second preimage	$2^{2n/3} \approx 2^{0.667n}$	[1]
		$2^{13n/22} \approx 2^{0.591n}$	[3]
		$2^{15n/26} \approx 2^{0.577n}$	Section 7
	Second preimage, quantum	$2^{n/2}$	Brute-force
		$2^{3n/7} \approx 2^{0.459n}$	Section 8

Organization of the Paper. We start in Section 2 with important preliminaries on the statistics of random functions, exceptional functions and how to find them.

In Section 3, after a detailed description of Duplex-based modes and the previous attack of [25], we describe nested exceptional functions and how to use them to mount a new attack against DuplexAEAD.

The rest of the paper deals with our new attacks on hash combiners. Section 4 is a reminder of important algorithmic techniques which are used ubiquitously in these generic attacks. Afterwards, Section 5, Section 6 and Section 7 are dedicated to the XOR combiner, Zipper Hash and Hash-Twice respectively. In Section 8 we give a (self-contained) description of our new quantum attack on Hash-Twice.

In this paper, for simplicity, we focus on the asymptotic complexity of attacks, and we aim only for constant probability of success, without explicitly computing it.

2 Preliminaries

In this paper, we use the notation $h : \mathbb{F}_2^n \times \mathbb{F}_2^u \to \mathbb{F}_2^n$ to denote a public compression function. In the constructions we study, the message is divided into u-bit blocks $M = m_0||\cdots||m_{L-1}$ and the internal state is initialized with a value x_0 which might (for Merkle-Damgård) or *might not* (for Duplex-based modes) be public. Then, letting x_i denote the value of the internal state after processing the block m_{i-1}, we define $x_{i+1} = h(x_i, m_i)$. We let h^* be the application of h to several concatenated message blocks, and when $h^*(x_0, M) = x_L$, we say that x_0 is mapped to x_L by the message M. Importantly, h^* does not include

the finalization function of Merkle-Damgård and its dependency on the message length (MD strengthening).

2.1 Iterating a Public Function

The attacks considered in this paper are generic attacks, that is, they do not rely on any weakness of the primitive. In the last years, many attacks of this type have exploited long messages that repeat a fixed message block $\beta \in \mathbb{F}_2^u$: $M = \beta^L = \underbrace{\beta||\cdots||\beta}_{L}$ with $\beta \in \mathbb{F}_2^u$.

Processing such a message can be viewed as the iteration of the function $x \mapsto h(x, \beta)$ with domain and co-domain $\mathbb{F}_2^n$. Generic attacks rely on the assumption that for β drawn in $\mathbb{F}_2^u$ uniformly and at random, $h(\cdot, \beta)$ behaves like a function drawn uniformly and at random. Under this assumption, we can exploit statistics on the behavior of random functions for cryptanalytic purposes. Statistics of random functions and properties of their functional graph have been widely studied in combinatorics [17,23,24,28,40]. As mentioned in the introduction, their properties have often been exploited in cryptanalysis, especially in the context of generic attacks on hash-based MACs [36,41] and hash combiners [3,19,37]. More recently, they have also been used to mount a generic attack against Duplex-based AEAD modes [25]. Whilst most cryptanalysis of MACs and hash combiners solely exploited average properties of random functions, this last work also relied on the ability of the attacker to find a function such that it has an exceptionally small cycle located on a large component (we refer to such function as an *exceptional function*). This paper builds upon this work and continues to explore how exceptional functions can be exploited to attack not only Duplex-based modes but also a wider range of targets.

2.2 Random Function Statistics

Define $\mathfrak{F}_{2^n}$ to be the set of functions that have the set $\{0, 1, \ldots, 2^n - 1\}$ as both domain and range. In the following, the term 'random function in $\mathfrak{F}_{2^n}$' refers to a function selected uniformly at random from this set. The functional graph of $f \in \mathfrak{F}_{2^n}$ is defined as the directed graph where each node is an element in $\{0, 1, \ldots, 2^n - 1\}$, and an edge goes from node i to node j if and only if $f(i) = j$ [24,40]. For any $x_0 \in \{0, 1, \ldots, 2^n - 1\}$, the set $\{x_i := f^i(x_0)\}_{i \in \mathbb{N}}$ has at most 2^n elements. Thus, there exist $i < j \leq 2^n$ such that $x_i = x_j$. Graphically, it comes that the path $x_0 \to \cdots \to x_i$ is connected to a *cycle* $x_i \to x_{i+1} \to \cdots \to x_{j-1} \to x_i$ [24]. In the following, we denote by *tail* of x_0 the path $x_0 \to \cdots \to x_i$, and by *cycle* of x_0 the path $x_i \to x_{i+1} \to \cdots \to x_{j-1} \to x_i$. The *tail length* $\lambda(x_0)$ (resp. *cycle length* $\mu(x_0)$) of x_0 is defined as the number of nodes located on its tail (resp. cycle). The set of all nodes such that their cycle is the same as the one of x_0 is called the *connected component* of x_0. The set of all nodes in the component of x_0 such that the first node on their path located on the cycle is x_i is called the *tree* of x_0. Considering all starting points in $\{0, 1, \ldots, 2^n - 1\}$, it comes that a functional graph can be viewed as a set of

connected components, where each component is a collection of trees linked by a cycle [23].

Theorem 1 ([23, Theorem 3]). *Seen from a random point x in a random function f of $\mathfrak{F}_{2^n}$, the expectations of parameters tail length, cycle length have the following asymptotic forms:*

- *Tail length $\lambda(x)$: $2^{n/2}\sqrt{\pi/8}$.*
- *Cycle length $\mu(x)$: $2^{n/2}\sqrt{\pi/8}$.*

Theorem 2 ([23, Theorem 8]). *Assuming a smoothness condition[6], the expected value of the size of the largest tree and of the largest connected component in a random mapping of $\mathfrak{F}_{2^n}$ are asymptotically:*

- *Largest tree: $\approx 0.48 \cdot 2^n$.*
- *Largest component: $\approx 0.76 \cdot 2^n$.*

These theorems show that for a random function f, most nodes tend to belong to a single giant component, and in fact to a single giant tree. In the following, we denote the cycle of this giant component by *main cycle* of f. This main cycle is expected to have an asymptotic size in the order of $\Theta(2^{n/2})$ (this is implied by Theorem 9 from [23]). However, as mentioned above, we will need functions that exhibit an exceptional functional graph in our attacks. In particular, we are interested in the probability for a random function to possess a large component with an exceptionally small cycle. We begin by defining more formally what is understood by 'large component' and 'exceptionally small cycle'. Our definitions are inspired by [17] and [25].

Definition 1. *μ-component. Let $0 < \mu < n/2$. A μ-component is a component that has a cycle of length at most 2^μ.*

Definition 2. *(s, μ)-component. Let $0 < \mu < n/2$, $0 < s < 1$. An (s, μ)-component is a μ-component of size greater than or equal to $s \cdot 2^n$.*

An (s, μ)-component is thus the formal denomination of what we loosely call a "large component with a small cycle". In [17], DeLaurentis provides the probability to find a function with such a component.

Theorem 3. *([17]). For a random $f \in \mathfrak{F}_{2^n}$, the probability $p_{s,\mu}$ that the functional graph of f has an (s, μ)-component is*

$$p_{s,\mu} = \sqrt{\frac{2(1-s)}{\pi s}} \cdot 2^{\mu - n/2} \cdot [1 + \mathcal{O}(r_n(s))]$$

where $r_n(s) = s^{-2}2^{3\mu-2n} + s^{\frac{-1}{2}}2^{\mu-n/2} + 2^{-n/3}$.

[6] This is a technical assumption in [23] that given the statistic of interest ξ, the quantity $\frac{1}{2^n}\mathbb{E}(\xi|\mathfrak{F}_{2^n})$ admits a limit when n goes to infinity.

In particular, this implies $p_{s,\mu} \approx \sqrt{\frac{2(1-s)}{\pi s}} \cdot 2^{\mu - n/2}$.

A heuristic way to understand this theorem is as follows. The functional graph of a random exceptional function resembles the graph of a random function in the sense that the distributions of the tree sizes and shapes as well as the distribution of the cycle sizes is the same. In particular, in both cases, there exist a tree of size $\Theta(2^n)$, some smaller trees and some cycles of length $\Theta(2^\mu)$ for all $0 < \mu \leq n/2$, as given by the distribution of the cycle lengths of a random function. However, in the case of an exceptional function, the tree of size $\Theta(2^n)$ is connected to a small cycle instead of a cycle of length $\Theta(2^{n/2})$ (as in the random case). Since there are $\Theta(2^{n/2})$ cyclic points, a random cyclic point is in a small cycle of size $\Theta(2^\mu)$ with probability $\Theta(2^{\mu - n/2})$. Thus, the probability that the main tree belongs to this cycle is about $2^{\mu - n/2}$, as described by Theorem 3 above.

2.3 Finding Exceptional Functions

Algorithm 1 Finding β such that $h(\cdot, \beta)$ has an (s, μ)-component.

Parameters: A function h.
Parameters: $\mu \leq n/2$, $0 < s < 1$.
Output: β.

loop ▷ $\mathcal{O}(2^{n/2-\mu})$ iterations
  $\beta \leftarrow \$$
  $x_0 \leftarrow \$$
  Compute cycle length 2^ν of $h(\cdot, \beta)$ starting from x_0 (Brent) ▷ Complexity $\mathcal{O}(2^{n/2})$
  if $\nu \leq \mu$ **then**
    Measure the size t of the component of x_0 by random sampling.
    if $t \geq s \cdot 2^n$ **then**
      return β.
    end if
  end if
end loop

As in [25], our generic attacks exploit functions that possess a large component with a small cycle. We must thus be able to find such functions. More precisely, we need to efficiently find β such that $h(\cdot, \beta)$ possesses an (s, μ)-component. To do so, we use an algorithm from [25], given here as Algorithm 1. This algorithm relies heavily on cycle-finding algorithms such as Floyd's or Brent's algorithm [31], which allow to efficiently recover the cycle length of a node using a negligible amount of memory, and constitute as such the primary tool to determine the main cycle length of a function. A brief description of Brent's algorithm can be found in Appendix A, and more details about cycle-finding algorithms can be found in Chapter 7 of [31].

The main idea in Algorithm 1 is to sample random values β from $\mathbb{F}_2^u$ and investigate whether or not the functional graph of $h(\cdot, \beta)$ has an (s, μ)-component.

To do so, for each sampled β, a random value x in $\mathbb{F}_2^n$ is sampled, and the algorithm then investigates whether or not the component on which x is located is an (s, μ)-component. First, Brent's algorithm outputs $\mu(x)$, the cycle length of the connected component on which x is located, and the value of a node on the cycle. This allows to determine whether or not the component on which x is located is a μ-component. However, it does not say much about its size. Thus, once a β and an element of its graph x such that the component on which x is located is a μ-component have been found, the algorithm investigates the size of the component by sampling several other points on the graph, applying Brent's algorithm, and then checking whether or not these elements belong to the μ-component to which x belongs. A probabilistic conclusion on the size of the component can then be drawn using the Central Limit Theorem.

The complexity analysis of this algorithm is detailed in Section 3.4 of [25], but can be summarized as follows. Since an (x, β) pair such that the component of $h(\cdot, \beta)$ on which x is located is a μ-component is found rarely, the main contribution to the complexity is the application of Brent's algorithm for each sampled x and for each sampled β. On average, a β such that its graph has a (s, μ)-component is drawn after $1/p_{s,\mu}$. The probability that x is located on this component is greater than s. Thus, the algorithm stops on average after $1/(p_{s,\mu}s) = \mathcal{O}(2^{n/2-\mu})$ applications of Brent's algorithm. Since Brent's algorithm has complexity $\mathcal{O}(2^{n/2})$, this gives a total complexity in $\mathcal{O}(2^{n-\mu})$ applications of the function h. Typically, for $s = 1/2$ and $\mu = \frac{n}{4}$ we obtain a complexity of $\mathcal{O}(2^{\frac{3n}{4}})$ applications of h.

3 Generic forgery attack against Duplex-based AEAD

The Duplex construction [14] can be viewed as an adaptation of the Sponge construction to the AEAD context. Since SpongeWrap in 2011 [8,9], many AEAD schemes have relied on this construction. While several variants of the mode exist (monkeyWrap, monkeyDuplex, Cyclist, Motorist,...), we describe here the common structure coined DuplexAEAD by the authors of [25].

The mode is instantiated with a public b-bit permutation P. The first r bits of internal state form the *outer part* and the last c bits form the *inner part*, where r is called the *rate* and c the *capacity*. During encryption (see Figure 1), the internal b-bit state is initialized with a function P_{init} that takes as input the key K, the nonce N and the associated data A. The injectively padded message is processed by r-bit blocks, which are XORed to the outer part between two calls to P. Lastly, a public finalization function P_{final} is applied. Note that we do not consider modes with a key-dependent finalization function such as the modes of Ascon [21] or of the third version of `NORX` for the CAESAR competition [2].

During decryption (see Figure 2), the initialization and finalization are unchanged. The ciphertext is also processed blockwise. This time, however, the outer state is *replaced* by the ciphertext blocks rather than XORed with them (this has been called *outerstate overwriting* by the authors of [25]).

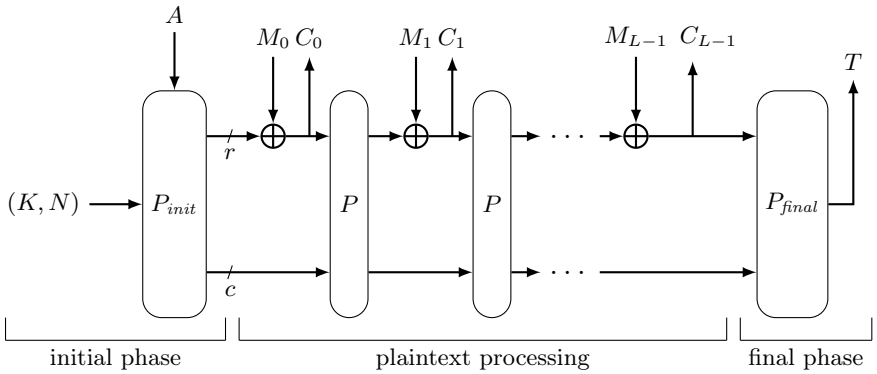

Fig. 1. Encryption using a Duplex-based mode.

More precisely, the decryption can be seen as the iteration of a compression function h, as illustrated in Figure 2. Letting $n = c$ and $u = r$, the public function h is defined as the restriction to its last n bits of the public permutation P:

$$\begin{aligned} h \quad : \quad & \mathbb{F}_2^{n+u} \longrightarrow \mathbb{F}_2^n \\ & (x, C) \longmapsto h(x, C) = \lfloor P(C||x) \rfloor_n \,. \end{aligned}$$

The initial value x_0 is secret, as it is the output of an initialization function that takes as input the key and the nonce.

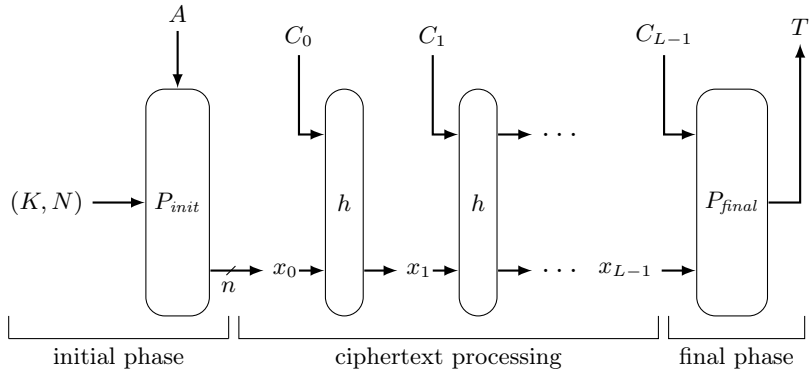

Fig. 2. Decryption using a Duplex-based mode.

Attack Scenario. The attacks on Duplex-based modes that we consider are forgery attacks. A *forgery* is a valid decryption query (N, A, C, T) such that

(C, T) was not outputted by the encryption oracle. We assume that in order to provide a forgery, the adversary has access to an encryption oracle, a decryption oracle (that does not release unverified plaintext) and a primitive oracle. The adversary is also assumed to be *nonce-respecting*, that is, she does not make two different *encryption* queries using the same nonce (this is however not required for decryption queries). For modes such that P_{init} is reversible for known nonce and associated data, forgery attacks allow to recover the secret key with $\mathcal{O}(2^{n/2})$ extra applications of the permutation, which is negligible compared to the complexity of the forgery (see Section 3.7 of [25]).

3.1 Generic attack against DuplexAEAD [25]

In [25], the authors describe a generic attack against Duplex-based AEAD modes. It is the first generic attack that exploits exceptional functions. This attack is a nonce-respecting forgery attack that does not assume the release of unverified plaintext, does not use encryption queries and is essentially memory free. The attack is described by Algorithm 2. It balances an *offline* phase in which the attacker makes calls to the primitive in order to find an exceptional function, and an *online* phase in which she exploits this exceptional function to make decryption queries that have an exceptionally high probability of success.

To find an exceptional function, the authors use Algorithm 1. This algorithm uses $\mathcal{O}(2^{n-\mu})$ calls to the primitive to find a β such that the function $h(\cdot, \beta)$ has a cycle of length smaller than $\mathcal{O}(2^{\mu})$[7]. In the online phase, the attacker makes calls to the decryption oracle using a single message consisting in the block β concatenated L times with L sufficiently large.

The nonce is either sampled randomly or takes arbitrary random values. The associated data is set to the empty string. Most importantly, the tags are produced by applying P_{final} to a state such that the outer state is set to β and such that the inner state is equal to an element in the small cycle of $h(\cdot, \beta)$'s main component.

Setting $L = cst \cdot 2^{n/2}$, this attack succeeds with constant probability. Indeed, since $h(\cdot, \beta)$ has a large component with a small cycle, the unknown inner state x_0 at the output of P_{init} for a (random) choice of nonce and an unknown key, belongs to this large component with constant probability. As $L = cst \cdot 2^{n/2}$, the inner state x_{L-1} obtained after processing β^{L-1} belongs to this cycle with constant probability (see [28] and Section 3.1 of [25]). If β^{L-1} belongs to the cycle, then the attack is successful as each cycle element is tried exhaustively.

The online phase has a total complexity equivalent to $\mathcal{O}(2^{n/2+\mu})$ primitive calls as the attacker makes $\mathcal{O}(2^{\mu})$ decryption queries with messages of length $\mathcal{O}(2^{n/2})$ blocks. Since the offline phase has complexity $\mathcal{O}(2^{n-\mu})$, the optimal total complexity is $\mathcal{O}(2^{3n/4})$ obtained for cycles of length $2^{\mu} = 2^{n/4}$.

[7] Although the set of possible choices β is a priori restricted by the rate of the sponge function, one can circumvent this limitation by using multiple blocks. The function $x \mapsto h(\beta_1, h(\beta_2, x))$ cannot be considered as a uniformly random function anymore, but the only difference is that it has polynomially more collisions, which bears no impact on the complexity exponents of our attacks.

Algorithm 2 Forgery attack against DuplexAEAD [25].

Parameters: μ, $L = 2^{n/2}$.
1: Find β such that $h(\cdot, \beta)$ has a $(1/2, \mu)$-component.
2: For an element y in the main cycle of $h(\cdot, \beta)$, compute $T = P_{final}(\beta \| y)$.
3: **for** 2^μ random nonces N **do**
4: Try (N, A, β^L, T) as a forgery (for an arbitrary A).
5: **end for**

3.2 Nesting Exceptional Functions

Long messages β^L have been used in many previous works to reduce the entropy of the internal state. Indeed, with $L \geq 2^{n/2}$, the state after processing the message β^L is within the cycle of the main component of $h(\cdot, \beta)$ with constant probability $p_\beta = \Pr[h^*(x, \beta^L) \in \mathcal{C} : x \in \mathbb{F}_2^u]$. We denote the main cycle of $h(\cdot, \beta)$ as $\mathcal{C}$ and its length as 2^μ ($2^\mu \leq 2^{n/2}$).

We propose a new method to further reduce the number of possible states. The main idea is to create a new function $\bar{g}$ mapping a point in $\mathcal{C}$ to another point in $\mathcal{C}$. Assuming that $\bar{g}$ behaves like a random function, we reach a cycle whose expected size is $\sqrt{2^\mu} = 2^{\mu/2}$, by iterating $\bar{g}$. Moreover, if we find an exceptional $\bar{g}$ such that its large component has a small cycle, we obtain an even smaller cycle of length 2^ν.

We first define the function $g_{\beta,\gamma}$ as follows, with γ a block in $\mathbb{F}_2^u$:

$$g_{\beta,\gamma} : x \mapsto h^*(x, \gamma \| \beta^L) \ .$$

The message block γ at the beginning randomizes the state, to make the function $g_{\beta,\gamma}$ independent from $h(\cdot, \beta)$. Moreover, $h^*(x, \beta^L)$ is in the cycle $\mathcal{C}$ with constant probability. In order to increase this probability, we define the function $\bar{g}_{\beta,\gamma}$ that iterates $g_{\beta,\gamma}$ until reaching a point in $\mathcal{C}$:

$$\bar{g}_{\beta,\gamma} : \mathcal{C} \to \mathcal{C}$$
$$x \mapsto \begin{cases} g_{\beta,\gamma}(x) = h^*(x, \gamma \| \beta^L) & \text{if } g_{\beta,\gamma}(x) \in \mathcal{C} \\ \bar{g}_{\beta,\gamma}(g_{\beta,\gamma}(x)) & \text{otherwise.} \end{cases}$$

The function $\bar{g}_{\beta,\gamma}$ is illustrated in Figure 3. In order to exploit its properties, we consider messages of the form

$$(\gamma \| \beta^L)^\Lambda = \underbrace{\gamma \| \beta^L \,\|\, \gamma \| \beta^L \,\|\, \cdots \,\|\, \gamma \| \beta^L}_{\Lambda} \ .$$

The state after processing this message is in the main cycle of $\bar{g}_{\beta,\gamma}$ with constant probability when $L \geq 2^{n/2}$ and $\Lambda \geq 2^{\mu/2}/p_\beta$. Indeed, each block $\gamma \| \beta^L$ corresponds to one application of $g_{\beta,\gamma}$; the final state is of the form $g_{\beta,\gamma}^\Lambda(x)$. With constant probability, the final call to $g_{\beta,\gamma}$ returns a state in $\mathcal{C}$. In this case, we can rewrite $g_{\beta,\gamma}^\Lambda(x)$ as $\bar{g}_{\beta,\gamma}^{\Lambda'}(x)$, with $\Lambda' \approx \Lambda \times p_\beta \geq 2^{\mu/2}$, by grouping calls to $g_{\beta,\gamma}$ corresponding to a call to $\bar{g}_{\beta,\gamma}$. Iteration of $g_{\beta,\gamma}$ is illustrated in Figure 3.

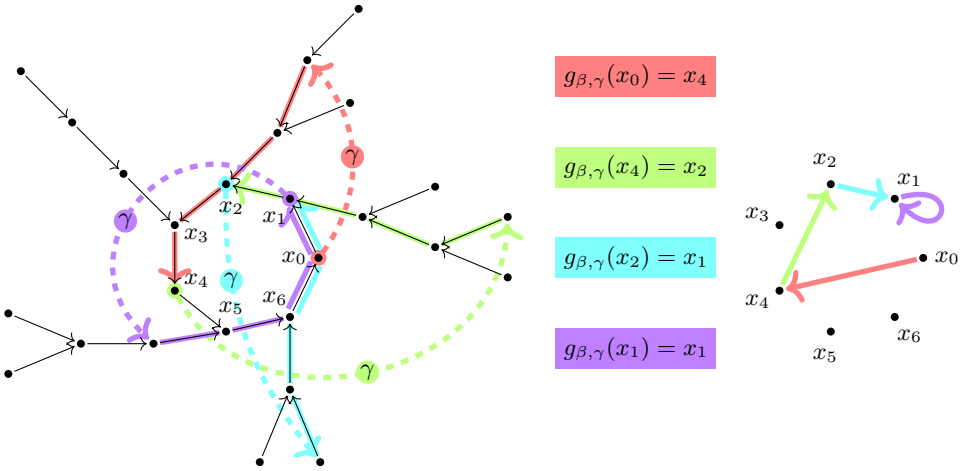

Fig. 3. Nesting functions. The black edges correspond to function $h(\cdot, \beta)$, while colored arrows correspond to function $g_{\beta,\gamma}$ (dashed colored edge correspond to $h(\cdot, \gamma)$).

This technique is related to the NestedRho method of Dinur, Dunkelman, Keller, and Shamir [20]. However the goal of the attacker and the context are different (in [20], the attacker has access to internal values), and this results in different strategies to build the nested function: in [20], the nested function is a map over collisions obtained for various *flavours* of the initial function, while our nested function is a map over cyclic points of the initial function.

Optimal parameters. We choose β and γ such that the corresponding functions have large components with small cycles. The algorithm is as follows, where the length of the cycle $\mathcal{C}$ in $h(\cdot, \beta)$ is denoted as 2^μ, and the length of the cycle of $\bar{g}_{\beta,\gamma}$ is denoted as 2^ν (we also use $L = 2^{n/2}$, $\Lambda = 2^{\mu/2}/p_\beta$):

1. Find β such that $h(\cdot, \beta)$ has a $(1/2, \mu)$-component.
2. Find γ such that $\bar{g}_{\beta,\gamma}$ has an $(1/2, \nu)$-component.
3. Return message $(\gamma \| \beta^{2^{n/2}})^{2^{\mu/2}/p_\beta}$.

Step 1 has complexity $\mathcal{O}(2^{n-\mu})$, and Step 3 has complexity $\mathcal{O}(2^{n/2+\mu/2})$. Therefore, the whole algorithm has complexity (up to constant factors) at least $2^{n-\mu} + 2^{n/2+\mu/2}$, which is at least $2^{2n/3}$ (achieved for $\mu = n/3$).

Step 2 requires $\mathcal{O}(2^{\mu-\nu})$ evaluations of $\bar{g}_{\beta,\gamma}$ in order to find an $(1/2, \nu)$-component in a function with domain and range of size 2^μ. We use precomputation in the functional graph of $h(\cdot, \beta)$ to reduce the cost of evaluating $h^*(\cdot, \beta^L)$ and $\bar{g}_{\beta,\gamma}$. After a precomputation that evaluates the function $h(\cdot, \beta)$ 2^t times, we can compute $h^*(\cdot, \beta^\Lambda)$ with complexity $\mathcal{O}(2^{n-t})$. Therefore step 2 has complexity $\mathcal{O}(2^{\mu-\nu} \times 2^{n-t})$. Finally the optimal trade-off uses $t = 2n/3$, $\mu = n/3$ and $\nu = 0$, with total complexity $\mathcal{O}(2^{2n/3})$.

Since this construction requires at least $2^{2n/3}$ operations, and we obtain a cycle of length 1 (a fixed point) with $2^{2n/3}$ operations, there is no reason to consider other trade-offs. For the same reason, nesting three or more functions would not improve the result.

This construction defines a message $\mathcal{M}$ of length $\mathcal{O}(2^{2n/3})$ and a state y, such that processing the message results in the state y with constant probability: $\Pr_x[h^*(x, \mathcal{M}) = y] = \Theta(1)$. This is a strong property that results in better trade-offs than the use of exceptional functions in some contexts: we obtain a fixed point rather than a cycle, but this requires longer messages.

Experimental Verification. We implemented this attack on a small-scale compression function h to check the behavior of random functions in practice, and verified that it matches the theoretical complexity bound.

3.3 New attack against DuplexAEAD

The nesting construction directly gives an improved attack against DuplexAEAD. Indeed, we know that $h^*(x, \mathcal{M}) = y$ with constant probability for a random x. Therefore, we can compute the tag from state y and obtain a forgery with constant probability. The new attack is described by Algorithm 3. Note that we add a final block after the message corresponding to the nesting construction, because this block enters the finalisation function P_{final} rather than the iteration h.

Algorithm 3 Improved forgery attack against DuplexAEAD.

Parameters: $\mu = n/3$, $\nu = 0$, $L = 2^{n/2}$, $\Lambda = 2^{\mu/2} = 2^{n/6}$.
1: Find β such that $h(\cdot, \beta)$ has an $(1/2, 2^{n/3})$-component.
2: Find γ such that $\bar{g}_{\beta,\gamma}$ has an $(1/2, 1)$-component.
3: For y the single element in the main cycle of $\bar{g}_{\beta,\gamma}$, compute $T = P_{final}(0\|y)$.
4: Try $(N, A, (\gamma\|\beta^L)^\Lambda\|0, T)$ as a forgery attempt (for an arbitrary N and A).

This attack has offline complexity $\mathcal{O}(2^{2n/3})$, and makes a single forgery attempt with a message of length $\mathcal{O}(2^{2n/3})$. It improves the previous attack with offline complexity $\mathcal{O}(2^{3n/4})$, making $\mathcal{O}(2^{n/4})$ forgery attempts with messages of length $\mathcal{O}(2^{n/2})$ [25].

Comparison with the lower bound. Interestingly, we propose an improvement over an attack that was described as tight in [35]. This is because the lower bound proof does not directly count time, but only the number of *distinct* calls to the inner permutation. This is indeed a lower bound on time, but in this setting the adversary does not have a direct control on the input of the permutation, meaning there can be a large gap between the two. In fact, the proof-tight variant of the [25] attack does $\mathcal{O}(2^n)$ permutation calls overall, but

only $\mathcal{O}(2^{n/2})$ distinct calls. The optimized attacks are more balanced, with the same asymptotics for calls and distinct calls, i.e. $\mathcal{O}(2^{3n/4})$ for [25] and $\mathcal{O}(2^{2n/3})$ for our attack.

4 Algorithmic Cryptanalysis Tools

The next sections study the use of exceptional functions to improve generic attacks on hash combiners. We start by reviewing several building blocks that have been used in previous generic attacks on hash functions and hash combiners and will be used often in the following sections.

These attacks target a hash function $\mathcal{H}$ following the Merkle-Damgård construction, using compression function h and a finalization function g:

$$\mathcal{H}(M) = g(h^*(\mathrm{IV}, M), |M|)$$

Precomputations in the Functional Graph. In this paper we will consider a scenario in which, having a large amount of precomputation available, we want to be able to find quickly the distance of a given point to the main cycle. This is done by expanding the functional graph of f [3,27,41] and building an appropriate data structure[8]. The algorithm takes as input a parameter $t \geq n/2$ which determines the amount of nodes in the expansion and returns a structure that contains 2^t nodes in the graph of f and their distance to a given common node x_0 on the main cycle (as well as their tail length). Given a new random node in the graph, we evaluate the function until we find one of the 2^t nodes in the data structure. This happens after $\mathcal{O}\left(2^{n-t}\right)$ evaluations.

The structure is built by computing chains of iterations of f until they reach a previously computed chain. As explained in [3] we can use distinguished points to store only a fraction 2^{t-n} of the points evaluated. This increases the online complexity by a constant factor, and reduces the memory complexity from 2^t to 2^{2t-n}. Instead of storing the distance to the main cycle in the structure, we can also store a link to the next distinguished point. This variant enables efficient computation of iterations of f, even when they don't reach the main cycle.

Joux's Multicollisions. Let $\mathcal{H}$ be a hash function. A 2^r-*multicollision* is a set $\mathcal{M}_{MC}$ of 2^r messages such that for any pair $M, M' \in \mathcal{M}_{MC}$, $\mathcal{H}(M) = \mathcal{H}(M')$. Joux [30] showed that when considering a hash function based on the Merkle-Damgård construction, it is possible to find a 2^r-multicollision in $r \cdot 2^{n/2}$ evaluations of the compression function h, using a series of collisions, following Algorithm 4.

The algorithm returns a structure of size $\mathcal{O}\left(r\right)$ representing a set of 2^r colliding messages. It is straightforward that for any $(u_1, \ldots, u_r) \in \mathbb{F}_2^r$, $m_1^{u_1} || \ldots || m_r^{u_r}$ is a preimage of x_f. As the complexity of Step 3 is in $\mathcal{O}(2^{n/2})$, the total complexity of this algorithm is in $\mathcal{O}(r \cdot 2^{n/2})$.

[8] See for example Step 4 in Attack 3 in [3].

Algorithm 4 Building a 2^r-multicollision. [30]

Parameters: An initial value x_0.
Output: A 2^r-multicollision $\mathcal{M}_{MC}$ and the collision value x_f.

1: Initialize a structure $\mathcal{M}_{MC}$ of pairs of message blocks.
2: **for** $i = 1, \ldots, s$ **do**
3: Find m_i^0, m_i^1 such that $h(x_{i-1}, m_i^0) = h(x_{i-1}, m_i^1) = x_i$.
4: Append (m_i^0, m_i^1) to $\mathcal{M}_{MC}$.
5: **end for**
6: **Return** $(x_f \leftarrow x_r, \mathcal{M}_{MC})$.

Expandable Message. An *expandable message* (EM) is a set of messages of different lengths such that they all map an initial state x_0 to the same final internal state x_f. They have been introduced by [16] in order to mount a second preimage attack against iterated hash functions when the message length is taken as input by a finalization function.

In [16], the authors built EM efficiently under the assumption that one could compute fixed points efficiently in the compression function. In [33], Kelsey and Schneier introduced a generic manner to build expandable message with message lengths in the range $[\ell, 2^\ell + \ell - 1]$ (such a set is called an $[\ell, 2^\ell + \ell - 1]$-expandable message) using Joux's multicollisions. This generic technique is described below.

Algorithm 5 Building an expandable message. [33]

Parameters: An initial value x_0, a length ℓ.
Output: An $[\ell, 2^\ell + \ell - 1]$-expandable message $\mathcal{M}_{EM}$, the output value x_f.

1: Initialize a structure $\mathcal{M}_{EM}$ of pairs of message blocks.
2: **for** $i = 1, \ldots, \ell$ **do**
3: Find m_i^0, m_i^1 such that $h(x_{i-1}, m_i^0) = h^*(x_{i-1}, [0]^{2^{i-1}-1} || m_i^1) = x_i$.
4: Append $(m_i^0, [0]^{2^{i-1}-1} || m_i^1)$ to $\mathcal{M}_{EM}$.
5: **end for**
6: Return $(x_f \leftarrow x_\ell, \mathcal{M}_{EM})$.

As in the case of multicollisions, it is straightforward that for any $(u_1, \ldots, u_\ell) \in \mathbb{F}_2^\ell$, $[0]^{u_1 \cdot (2^0 - 1)} m_1^{u_1} || \ldots || [0]^{u_\ell \cdot (2^{\ell-1} - 1)} m_\ell^{u_\ell}$ is a preimage of x_f. Further, for any $p \in [\ell, 2^\ell + \ell - 1]$, a message of length p can be built by considering the LSB of the binary representation of $p - \ell$. The i^{th} message block depends on the i^{th} least significant bit of $p - \ell$: if it is equal to 0, then the short message (of length 1) m_i^0, is selected; otherwise, the long message (of length 2^{i-1}) $[0]^{2^{i-1}-1} || m_i^1$ is selected. Step 3 of this algorithm has complexity $\mathcal{O}(2^{n/2})$. Thus, the total complexity of this procedure is $\mathcal{O}(\ell \cdot 2^{n/2} + 2^\ell)$.

Simultaneous Expandable Message. Simultaneous expandable messages [3,29] are an adaptation of the Expandable Message technique to contexts where two

compression functions h_1 and h_2 are iterated upon in parallel, as in the XOR Combiner or Hash-Twice. More precisely, a *simultaneous expandable message* (SEM) is a set of messages $\mathcal{M}_{SEM}$ such that for any length p in a certain range, there exist a message that maps some pre-fixed initial states (x_0, y_0) to two final states (x_f, y_f).

Similarly to the classical expandable message case, the idea is to use building blocks consisting of two messages, a shorter one and a longer one. For any starting state, the shorter one has fixed length $C = n/2 + \log(n)$, and the longer message has length i strictly greater than C. For a fixed parameter t, the authors of [3] build a $[C(C-1)+tC, C^2-1+C(2^t+t-1)]$-simultaneous expandable message using $C-1+t$ building blocks. The first $C-1$ building blocks use long messages with length $C+1 \leq i \leq 2C-1$, which give a $(C(C-1), C^2-1)$-expandable message by considering at most one longer message and using the shorter message in all other building blocks. The last t building blocks are built with parameter $i = C(2^{j-1}+1)$ with $1 \leq j \leq t$. Then, for a length $p \in [C(C-1)+tC, C^2-1+C(2^t+t-1)]$, one first computes $p \bmod C$, finds p' in $[C(C-1), C^2-1]$ such that $p' = p \bmod C$. This determines the first $C-1$ messages used. Last, one computes $(p-p')/C$ which is in $[t, 2^t+t-1]$, and selects the final t messages using the binary representation of $(p-p')/C$.

Section 2.6 of [3] details how to construct a building block given a starting state and a longer message length i with complexity about $i + n \cdot 2^{n/2}$. Constructing all building blocks required for a $[C(C-1)+tC, C^2-1+C(2^t+t-1)]$-simultaneous expandable message thus has complexity approximately $\sum_{i=C+1}^{2C-1} i + \sum_{j=1}^{t} C(2^{j-1}+1) + n \cdot 2^{n/2} \cdot (C-2+t) \approx n^2 \cdot 2^{n/2} + n \cdot 2^t$. To obtain a message that extends up to length 2^ℓ, we need $C \cdot 2^t \approx n \cdot 2^t \approx 2^\ell$. Ignoring the constant terms, this gives a total complexity of about

$$2^\ell + n^2 \cdot 2^{n/2}\,.$$

Case of the Zipper Hash (Cascade Expandable Message). For our second preimage attack, we will also need a cascading expandable message (CEM) for the middle part of the Zipper Hash. Section 7.3 of [3] describes how to build such a set of messages. As the procedure is very similar to the one used for building a SEM we do not provide more details here. Letting 2^ℓ be the maximum length reached by the cascading expandable message, the complexity of the construction of a CEM is equal to $2^\ell + n^2 \cdot 2^{n/2+1}$.

For an expandable message $\mathcal{M}$ of any type (EM, SEM or CEM) and for any integer q in the range covered by this expandable message, we denote by $M_{\|q}$ the message in $\mathcal{M}$ that has q blocks.

Interchange Structure. Given two compression functions h_1 and h_2 and two initial states (x_1, x_2), an *interchange structure* [37] (IS) consists in two sets $\mathcal{Z}_1$ and $\mathcal{Z}_2$ and an associated set of messages $\mathcal{M}$ such that for any pair (z_1, z_2) of final states in $\mathcal{Z}_1 \times \mathcal{Z}_2$, there exist a message of this set that maps (x_1, x_2) to

(z_1, z_2):

$$\forall (z_1, z_2) \in \mathcal{Z}_1 \times \mathcal{Z}_2, \exists m \in \mathcal{M}, h_1^*(x_1, m) = z_1, h_2^*(x_2, m) = z_2 \ . \quad (1)$$

The idea is to consider a primary message M made of a sequence of chunks $M_0, \ldots, M_i, \ldots$, and chains of internal states a_j for h_1 (resp. b_k for h_2). An individual state in a chain is denoted a_j^i so that $h_1(a_j^i, M_i) = a_j^{i+1}$. A message block M_i transforms the pair of states (a_j^i, b_k^i) into (a_j^{i+1}, b_k^{i+1}).

The IS is made of 2^{2t} *switches*, where a switch allows to jump between two chains of states in a controlled way. More precisely, by selecting a secondary message chunk M_i' instead of M_i, a switch will allow to jump from state $(a_{j_0}^i, b_{k_0}^j)$ to $(a_{j_0}^{i+1}, b_{k_1}^{j+1})$, i.e., entering a new chain of states where b has changed, but not a. The roles of a and b can be swapped. After building a series of 2^{2t} switches, we have obtained 2^t chains of internal states for h_1 and h_2 respectively, and the sets $\mathcal{Z}_1$ and $\mathcal{Z}_2$ are obtained as the respective ends of these chains. By selecting appropriate message chunks for each switch we can obtain any pair (z_1, z_2).

A single switch is constructed using a multicollision. Starting from pairs of states (a, b_1) and (a, b_2), we start by finding a $2^{n/2}$-collision of h_1^*, i.e., a set of $2^{n/2}$ messages $\mathcal{M}_c$ such that $\forall m \in \mathcal{M}_c, h_1^*(a, m) = a'$ where a' becomes a new value in the chain. Among the set of messages $\mathcal{M}_c$, we find a pair m, m' such that $h_2^*(b_1, m') = h_2^*(b_2, m)$ in order to change the value of the h_2-chain. The message chunk m (of $n/2$ blocks approximately) becomes the new "primary" message block and m' becomes the new secondary message block. All current chains are extended with m.

Multi-Cycles in Functional Graphs. The goal of this technique [5] is to find a pair of starting states (x_0, y_0) that will map to a pair of target states (x_1, y_1) after λ iterates of h_1 (resp. h_2) with a fixed message β. These targets are both cyclic nodes in their respective cycles, and the length λ has usually some limitation. In this paper, we use an alternative description of the multi-cycle technique using the Sun-Qin theorem[9], which is explained in detail in Section 5.1.

Diamond structure. The diamond structure was introduced by Kelsey and Kohno [32] in the context of herding attacks against Merkle-Damgård based hash functions. A diamond structure with 2^t leaves is a binary tree of messages mapping 2^t chosen starting states to the same output state. It can be constructed using about $n\sqrt{t}2^{(n+t)/2}$ computations [10], by finding colliding messages between pairs of states at each level in the tree. Intuitively, for the first level, at there are 2^t starting states, trying $2^{(n-t)/2}$ messages for each of them creates $2^{(n+t)/2}$ new states among which 2^t collisions can be expected to exist. After that, at each level the number of collisions to be found diminishes faster than the difficult of finding them, e.g., the final collision only requires about $2^{n/2}$ compression function queries.

[9] Usually referred to as the Chinese Remainder Theorem.

5 Preimage Attack on the XOR Combiner

We begin with the XOR combiner: $M \mapsto \mathcal{H}_1(M) \oplus \mathcal{H}_2(M)$, with $\mathcal{H}_1$ and $\mathcal{H}_2$ two iterated hash functions:

$$\mathcal{H}_1(M) = g_1(h_1^*(\mathrm{IV}_1, M), |M|) \qquad \mathcal{H}_2(M) = g_2(h_2^*(\mathrm{IV}_2, M), |M|)$$

The adversary is given a challenge $\bar{H}$, and she must construct a message M such that $\mathcal{H}_1(M) \oplus \mathcal{H}_2(M) = \bar{H}$. The first attack in the setting was proposed by Leurent and Wang, with complexity $2^{5n/6}$ using the interchange structure [37]. Later work has reduced the complexity to $2^{2n/3}$ [19], then to $2^{5n/8}$ [5], and the best known attack today has complexity $2^{11n/18}$ [3].

5.1 Simple Attack Using Multi-cycles

We first explain a basic cycle-based attack with complexity $\mathcal{O}(2^{3n/4})$, represented in Figure 4, and a variant using small cycles with complexity $\tilde{\mathcal{O}}(2^{7n/10})$. This attack is an alternative description of the multi-cycle idea introduced in [5].

When processing a long message β^λ, we iterate the function $h_1(\cdot, \beta)$ starting from IV_1, and $h_2(\cdot, \beta)$ starting from IV_2. We denote the cycle reached in $h_1(\cdot, \beta)$ (respectively in $h_2(\cdot, \beta)$) as $\{ A_j \}$ (resp. $\{ B_k \}$) and its length as 2^{μ_1} (resp. 2^{μ_2}). We assume that 2^{μ_1} and 2^{μ_2} are relatively prime[10] (μ_1 and μ_2 are not necessarily integers) with $2^{\mu_1} < 2^{\mu_2}$.

This simple structure provides a way to control independently the behavior of $\mathcal{H}_1$ and $\mathcal{H}_2$, by varying the length of the message $M = \beta^\lambda$. Indeed, with $\lambda > T$ with T the maximum tail length of IV_1 and IV_2, the final state is of the form (A_j, B_k); moreover, with a suitable choice of the indices of A_j and B_k we have:

$$h_1^*(IV_1, \beta^\lambda) = A_{\lambda \bmod 2^{\mu_1}} \qquad h_2^*(IV_2, \beta^\lambda) = B_{\lambda \bmod 2^{\mu_2}}$$

Therefore, we can reach any specific state (A_{j^*}, B_{k^*}) by solving a system of modular equations: $\lambda \equiv j^* \bmod 2^{\mu_1}$, $\lambda \equiv k^* \bmod 2^{\mu_2}$. Using the Sun-Qin theorem, we deduce a length $\lambda \geq T$ such that $h_1^*(\mathrm{IV}_1, \beta^\lambda) = A_{j^*}$ and $h_2^*(\mathrm{IV}_2, \beta^\lambda) = B_{k^*}$. For a random (j^*, k^*), the length λ is uniformly distributed between T and $T + 2^{\mu_1+\mu_2}$.

The full attack requires an expandable message (defined in Section 4) to bypass the MD strengthening. It uses the following steps (as shown in Figure 4) with a parameter $L = 2^\ell$ corresponding to the preimage length:

1. Build a simultaneous expandable message $\mathcal{M}$ with maximum length L, with final states $(\check{x}, \check{y})$.
2. Select a message block β. Find the cycle $\{ A_j \}$ of $h_1(\cdot, \beta)$ and the cycle $\{ B_k \}$ of $h_2(\cdot, \beta)$, starting from $\check{x}$ and $\check{y}$.

[10] This happens with probability $6/\pi^2 \approx 0.61$ for random integers.

3. For a random block w, match $\{\, g_1(h_1(A_j, w)) \,\}$ and $\{\, g_2(h_2(B_k, w)) \oplus \overline{H} \,\}$.
   If there is a match (j^*, k^*), find the length λ such that $h_1^*(\check{x}, \beta^\lambda) = A_{j^*}$ and $h_2^*(\check{y}, \beta^\lambda) = B_{k^*}$ using the Sun-Qin theorem.
   If $\lambda < L$, select the message $\bar{M}$ of length $L-\lambda-1$ in the expandable message.
   The preimage is $M = \bar{M} \| \beta^\lambda \| w$.
   Otherwise, repeat Step 3.

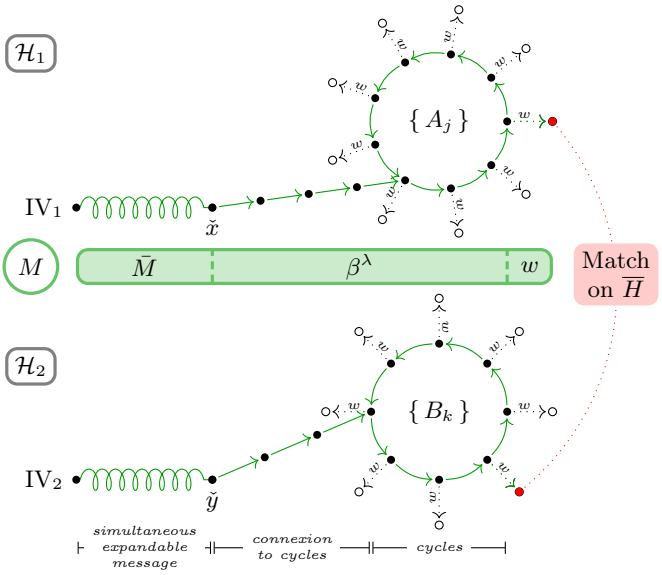

Fig. 4. Preimage attack on the XOR combiner based on cycles.

Complexity analysis.

- Step 1 has complexity $\mathcal{O}(2^\ell)$ (assuming $\ell > n/2$).
- Step 2 has complexity $\mathcal{O}(2^{n/2})$.
- Each iteration of Step 3 (for each try of w) has complexity $\mathcal{O}(2^\mu)$, with $\mu = \max[\mu_1, \mu_2]$. There is a match (j^*, k^*) with probability $2^{\mu_1+\mu_2-n}$, and the probability that $\lambda < L$ is $2^{\ell-\mu_1-\mu_2}$. Therefore, we expect $2^{n-\ell}$ iterations, which gives a total complexity of $\mathcal{O}(2^{n-\ell+\mu})$ for Step 3.

Trade-off with arbitrary β. With an arbitrary choice of β we have $\mu \approx n/2$. The optimal trade-off is achieved with $\ell = 3n/4$ with complexity $\mathcal{O}(2^{3n/4})$.

Trade-off with small cycles. The attack is improved by first searching β such that $h_1(\cdot, \beta)$ and $h_2(\cdot, \beta)$ have exceptionally small cycles in their main component, of length at most 2^μ. Because we need to find two small cycles simultaneously, the

complexity of this precomputation step is $\mathcal{O}(2^{3n/2-2\mu})$. Indeed, for a random β the probability that $h_1(\cdot,\beta)$ and $h_2(\cdot,\beta)$ both have small cycles is $(2^{\mu-n/2})^2$, and checking if a β works costs time $\mathcal{O}(2^{n/2})$.

In this case, the optimal trade-off is achieved with $\mu = 2n/5$ and $\ell = 7n/10$ with complexity $\mathcal{O}(2^{7n/10})$.

5.2 Advanced Attack using Multi-cycles

We now give an alternative description of the best known attack [3] (changing the order of some steps), with complexity $\tilde{\mathcal{O}}(2^{11n/18})$, and an improvement using exceptional functions with complexity $\tilde{\mathcal{O}}(2^{3n/5})$.

Instead of using a single connection from the IVs to the cycles, this attack considers several connections. In the final step, each target (A_j, B_k) can now be reached in many different ways, and this improves the probability that one of the corresponding messages β^λ is short. In order to build the connections efficiently, we precompute 2^t points in the functional graph, together with their distance to a fixed cyclic point. Therefore, starting from a random point, we only need 2^{n-t} iterations to reach a precomputed point, and to deduce the connection to the cycle. Moreover, this attack uses an interchange structure to provide independent choices of the connections in $\mathcal{H}_1$ and $\mathcal{H}_2$.

The full attack produces a message of length L with the following steps, as shown in Figure 5:

1. Build a simultaneous expandable message $\mathcal{M}$, with maximum length L, with final state $(\check{x}, \check{y})$.
2. Build an interchange structure $\mathcal{I}$ with 2^r endpoints $\{\,\bar{x}\,\}$ and $\{\,\bar{y}\,\}$, starting from $(\check{x}, \check{y})$.
3. Find the main cycle of $h_1(\cdot,\beta)$ denoted as $\{\,A_j\,\}$ and the main cycle of $h_2(\cdot,\beta)$ as $\{\,B_k\,\}$.
4. Precompute a set of 2^t points in the main component of $h_1(\cdot,\beta)$ and $h_2(\cdot,\beta)$, with known distances to the main tree root.
5. Choose 2^v random blocks ρ; for all interchange endpoints $\bar{x}$ and $\bar{y}$ compute the distance from $h_1(\bar{x},\rho)$ and $h_2(\bar{y},\rho)$ to the respective cycle.
6. For a random block w, match $\{\,g_1(h_1(A_j,w))\,\}$ and $\{\,g_2(h_2(B_k,w)) \oplus \overline{H}\,\}$. If there is a match (j^*,k^*), then for each $\bar{x},\bar{y},\rho$, find the length λ such that $h_1^*(\bar{x},\rho\|\beta^\lambda) = A_{j^*}$ and $h_2^*(\bar{y},\rho\|\beta^\lambda) = B_{k^*}$ using the Chinese remainder theorem.
   If $\lambda < L$, select the message $\hat{M}$ such that $h_1^*(\check{x},\hat{M}) = \bar{x}, h_2^*(\check{y},\hat{M}) = \bar{y}$ in the interchange structure and the message $\bar{M}$ of appropriate length in the expandable message. The preimage is $M = \bar{M}\|\hat{M}\|\rho\|\beta^\lambda\|w$.
   Otherwise, repeat Step 6.

Complexity analysis.

- The complexity of step 1 is $\mathcal{O}(2^\ell)$ (assuming $\ell > n/2$).
- The complexity of step 2 is $\tilde{\mathcal{O}}(2^{n/2} \times 2^{2r})$.

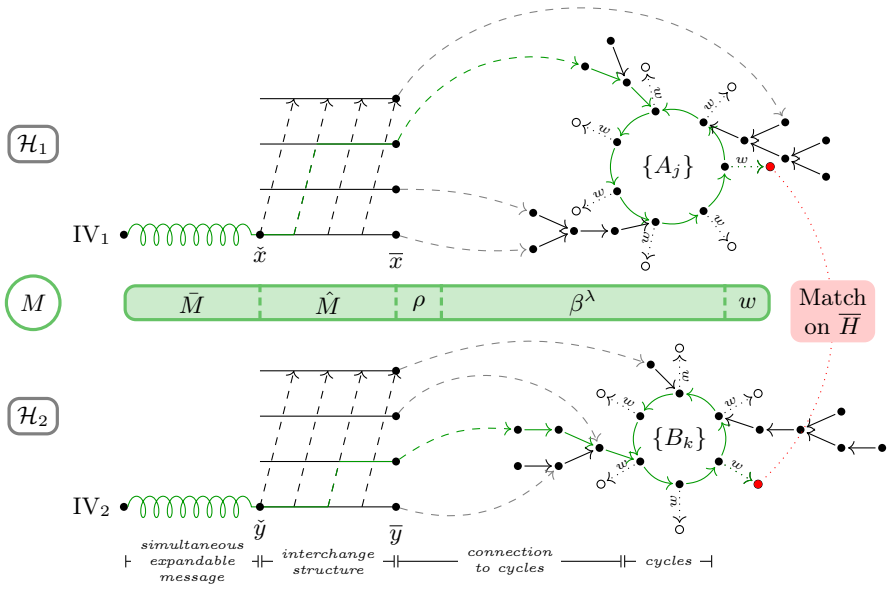

Fig. 5. Improved preimage attack on the XOR combiner based on cycles.

- The complexity of step 3 is $\mathcal{O}(2^{n/2})$.
- The complexity of step 4 is $\mathcal{O}(2^t)$ (assuming $t > n/2$).
- The complexity of step 5 is $\mathcal{O}(2^{n-t} \times 2^r \times 2^v)$, using the points precomputed at step 4. Since we use the main component of $h_1(\cdot, \beta)$ and $h_2(\cdot, \beta)$, each point $h_1(\bar{x}, \rho)$ or $h_2(\bar{y}, \rho)$ has a constant probability to reach the cycle.
- Each iteration of Step 6 (for each try of w) has complexity $\mathcal{O}(2^\mu)$, with $\mu = \max[\mu_1, \mu_2]$. There is a match (j^*, k^*) with probability $2^{\mu_1+\mu_2-n}$. In this case, we obtain 2^{2r+v} candidates λ; each one is smaller than L with probability $\Omega(2^{\ell-\mu_1-\mu_2})$. Therefore, we expect $\Theta(2^{n-\ell-2r-v})$ iterations, with a total complexity $\mathcal{O}(2^{n+\mu-\ell-2r-v})$ (assuming $2r + v < n - \mu$).

Trade-off with arbitrary β. With an arbitrary choice of β we have $\mu \approx n/2$. The optimal trade-off is achieved with $\ell = t = 11n/18$, $r = n/18$ and $v = 3n/18$, with complexity $\tilde{\mathcal{O}}(2^{11n/18})$, corresponding to the attack from [3].

Trade-off with small cycles. The attack is also improved by first searching β such that $h_1(\cdot, \beta)$ and $h_2(\cdot, \beta)$ have exceptionally small cycles in their main component, with a precomputation of complexity $\mathcal{O}(2^{3n/2-2\mu})$.

In this case, the optimal trade-off is achieved with $\mu = 9n/20$, $\ell = t = 3n/5$, $r = n/20$ and $v = 3n/20$, with complexity $\tilde{\mathcal{O}}(2^{3n/5})$, resulting in the best attack against the XOR combiner.

6 Second preimage attack on the Zipper Hash

We now consider the Zipper Hash construction [38], which processes the message through two iterated hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$, first in the normal order, and then in reverse order. Formally, the Zipper Hash is defined as[11]

$$\mathcal{H}(M) = \mathcal{H}_2(\mathcal{H}_1(\mathrm{IV}, M), \overleftarrow{M}) = g_2\big(h_2^*\big(g_1(h_1^*(\mathrm{IV}, M)), \overleftarrow{M}\big)\big)$$

with h_1, h_2 and g_1, g_2 the compression and finalization functions of $\mathcal{H}_1$ and $\mathcal{H}_2$, and $\overleftarrow{M}$ the message built by concatenating the blocks of M in reverse order.

In this section, we present second preimage attacks against this construction. The adversary is given a long challenge message $M = m_1 \| m_2 \| \dots \| m_L$, and her goal is to find $M' \neq M$ such that $\mathcal{H}(M') = \mathcal{H}(M) = H$. Following Kelsey and Schneier [33], our strategy is to build a new message with a path that collides with the path of the challenge. Note that in the Zipper Hash, it is assumed that the finalization functions g_1 and g_2 do not depend on the length of the message. The message length only influences the padding (MD strengthening). In particular, the block containing the padding is the last block of the message, and it only impacts the middle part of the absorption: it is the last block processed by h_1 and the first block processed by h_2. Therefore, the second preimage does not necessarily have the same length as the challenge.

In Section 7 of [3], the authors present the first generic second preimage attack on Zipper Hash. This attack uses a variety of techniques: Joux's multi-collisions, cascading expandable messages as well as deep iterates and multi-cycle techniques. They propose two variants. The first one considers second preimages of length L' at most $2^{n/2}$ and has complexity $\mathcal{O}(2^{5n/8})$. The other one does not consider a limit on the length L' of the second preimage and has complexity $\mathcal{O}(2^{3n/5})$. In this section, we present attacks on Zipper Hash that exploit exceptional functions. Our attacks contain a precomputation phase that finds a message block β such that $h_2(\cdot, \beta)$ (and then $h_1(\cdot, \beta)$ for the most advanced attacks) has (have) a large component with a small cycle. Our contribution is twofold.

First, we present two attacks that solely rely on exceptional functions. The first attack has complexity $\mathcal{O}(2^{3n/4})$ but can be improved easily to $\mathcal{O}(2^{2n/3})$ by the means of graph expansion techniques. The second attack also has complexity $\mathcal{O}(2^{2n/3})$, but relies on the technique of nested exceptional functions introduced in Section 3.2. Although these two attacks do not beat the current state of the art, we believe that they are an interesting contribution as they are based on techniques that are different to the previous existing attack of [3]. Next, we review the best known attack [3], with complexity $\mathcal{O}(2^{3n/5})$, and improve it using exceptional functions, reaching complexity $\mathcal{O}(2^{7n/12})$.

In the following, as in [3], we denote the sequence of internal states computed during the invocation of h_1 (resp. h_2) on M (resp. $\overleftarrow{M}$) by $a_0, a_1, \dots, a_L$ (resp. $b_0, b_1, \dots, b_L$) with $a_0 = \mathrm{IV}$ and $b_0 = \mathcal{H}_1(M)$. We let $\ell = \log_2(L)$ where L is the

[11] We slightly abuse the notations $\mathcal{H}_1$ and $\mathcal{H}_2$ to take the IV as a parameter.

length in blocks of the challenge, and $L' = 2^{\ell'}$ to be the length in blocks of the second preimage.

6.1 Simple Attack based on Exceptional Functions

Based on techniques for finding functions with an exceptionally small cycle, this first attack has complexity $\mathcal{O}(2^{3n/4})$ in its basic version, and $\mathcal{O}(2^{2n/3})$ using a simple improvement with graph expansion techniques. It is represented in Figure 6.

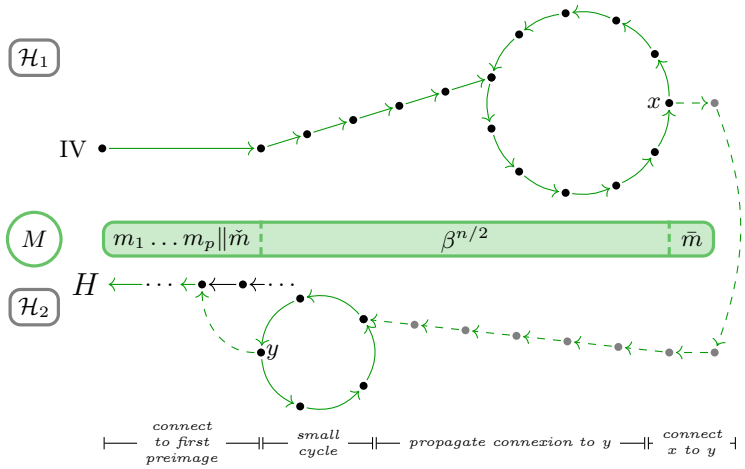

Fig. 6. Simple second preimage against Zipper.

1. Compute the sequence of internal states $b_1, b_2, \dots, b_L$.
2. Find a block β such that the graph of $h_2(\cdot, \beta)$ has a $(1/2, \mu)$-component with $\mu \leq 2^{n/2}$. Recover an element from the cycle y.
3. Compute $h_2(y, m)$ with $m = 0, 1, \dots$ until a collision with some b_p, $1 \leq p \leq L-1$, is found for a message block $\check{m}$.
4. Compute the digest of messages $m_1 \dots m_p \| \check{m} \| \beta^{2^{n/2}} \| m'$ with $m' = 0, 1, \dots$ until a second preimage is found using a message block $\bar{m}$.

The second preimage is $M' = m_1 \dots m_p \| \check{m} \| \beta^{2^{n/2}} \| \bar{m}$.

Complexity analysis. When computing the digest of messages

$$m_1 \dots m_p \| \check{m} \| \beta^{2^{n/2}} \| m' \,,$$

we focus on the state $b_{2^{n/2}+1} = h_2^*(\mathcal{H}_1(M), m' \| \beta^{2^{n/2}})$ obtained in the second pass, after processing the final block m' and the sequence of blocks β. With

high probability, $b_{2^{n/2}+1}$ is in the cycle of size 2^μ of $h_2(\cdot,\beta)$. Therefore, we have $b_{2^{n/2}+1} = y$ with probability $\Theta(2^{-\mu})$, and in this case we obtain a valid preimage. Therefore we expect $\Theta(2^\mu)$ iterations of Step 4.

- The complexity of Step 1 is $\mathcal{O}(2^\ell)$.
- The complexity of Step 2 is $\mathcal{O}(2^{n-\mu})$.
- The complexity of Step 3 is $\mathcal{O}(2^{n-\ell})$.
- The complexity of Step 4 is $\mathcal{O}(2^{\mu+n/2})$.

The optimal trade-off is $\mathcal{O}(2^{3n/4})$ using $\mu = n/4$. This complexity can be obtained for messages of length $L = 2^\ell$ such that $n/4 \leq \ell \leq 3n/4$. It builds second preimages of length at most $2^{n/2} + 2^\ell$.

This attack is interesting because the attacker does not need to control the behaviour of $\mathcal{H}_1$ in any way: the use of exceptional function sends the state of $\mathcal{H}_2$ into a small set of states, which is sufficient to mount an attack.

Improvement using graph expansion techniques. This attack can be improved by adding a step in between Steps 3 and 4 in which the attacker stores the distance from y of 2^t nodes, $t \geq n/2$, in the graph of $h_2(\cdot,\beta)$ in a table $\mathcal{T}$ (graph expansion). This step costs $\mathcal{O}(2^t)$, but reduces the cost of the last step to $\mathcal{O}(2^{\mu+n-t})$. The optimal trade-off is $\mathcal{O}(2^{2n/3})$ using $\mu = n/3, t = 2n/3, n/3 \leq \ell' \leq 2n/3$. This complexity is obtained for challenges of length L such that $2^{n/3} \leq L \leq 2^{2n/3}$ blocks, and builds second preimages of length at most $2^{n/2}+2^\ell$.

6.2 Simple Attack based on Nested Exceptional Functions

The nesting construction provides another way to improve the basic attack complexity. We use the same notations as in Section 3.2, with

$$g_{\beta,\gamma} : x \mapsto h_2^*(x, \gamma \| \beta^L)\,.$$

1. Compute the sequence of internal states $b_1, b_2, \ldots, b_L$.
2. Find β such that $h_2(\cdot,\beta)$ has an $(1/2, 2^\mu)$-component.
3. Find γ such that $\bar{g}_{\beta,\gamma}$ has an $(1/2, \nu)$-component.
4. Compute $h_2(y, m)$ with $m = 0, 1, \ldots$ until a collision with some b_p, $1 \leq p \leq L-1$, is found for a message block $\check{m}$.

The second preimage is $M' = m_1 \ldots m_p \| \check{m} \| (\beta^L \| \gamma)^\Lambda$.

Since the nested exceptional function forces the state to a single value, Step 4 only requires a constant number of iterations. This attack has complexity $\mathcal{O}(2^{2n/3})$ and provides a second preimage for target messages of length $2^{n/3} \leq L \leq 2^{2n/3}$. The second preimage obtained has length $\mathcal{O}(2^{2n/3})$.

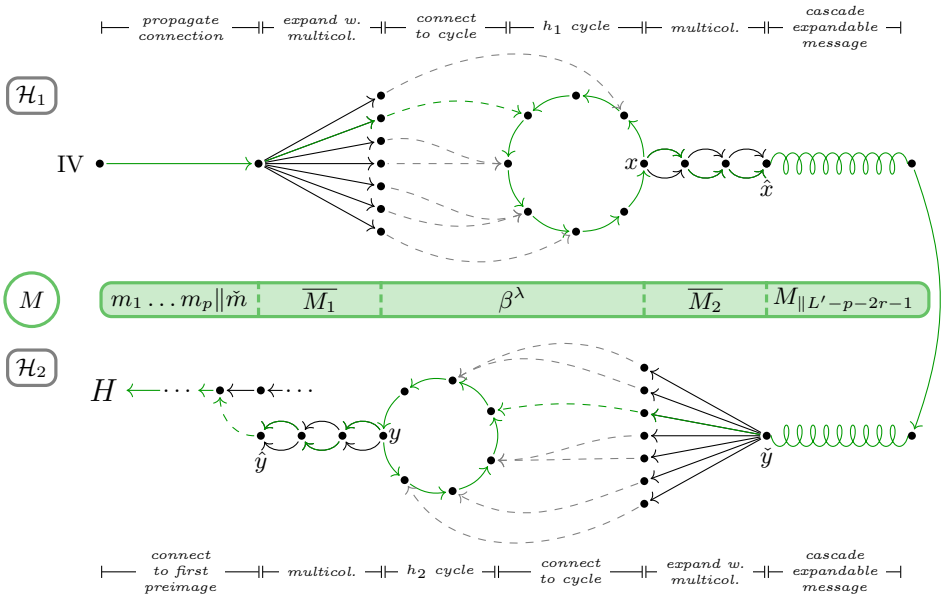

Fig. 7. Second preimage attack against the Zipper Hash construction.

6.3 Improving the attack from [3] using Exceptional Functions

We now give an alternative description of the best known attack [3] (changing the order of some steps), with complexity $\mathcal{O}(2^{3n/5})$, and an improvement using exceptional functions with complexity $\mathcal{O}(2^{7n/12})$. This attack is illustrated in Figure 7.

1. Compute the sequence of internal states $b_1, b_2, \dots, b_L$.
2. Select a message block β. Find the main cycle of $h_1(\cdot, \beta)$ of length 2^{μ_1} and choose an element x in the cycle. Find the main cycle of $h_2(\cdot, \beta)$ of length 2^{μ_2} and choose an element y in the cycle. Repeat this step if 2^{μ_1} and 2^{μ_2} are not relatively prime (μ_1 and μ_2 are not integers).
3. Precompute the distance from the main root of the main cycle of 2^t nodes in the graph of $h_1(\cdot, \beta)$ and in the graph of $h_2(\cdot, \beta)$ with $t \geq n/2$ (graph expansion).
4. Build a 2^r-multicollision $\mathcal{M}_{MC1}$ (resp. $\mathcal{M}_{MC2}$) on h_2 (resp. h_1) using y (resp. x) as initial value, denote by $\hat{y}$ (resp. $\hat{x}$) the final value of this multi-collision.
5. Set $L' = 2^{\ell'}$ to be the length of the second preimage. Build an L'-cascade expandable message $\mathcal{M}_{CEM}$ on h_1 and h_2 using $\hat{x}$ as initial value. Denote the final value by $\check{y}$.
6. Compute $h_2(\hat{y}, m)$ with $m = 0, 1, \dots$ until a collision with some b_p, $1 \leq p \leq L' - 1$, is found for a message block $\check{m}$.
7. For each $\overline{M_1} \in \mathcal{M}_{MC1}$,

- Process $m_1 \dots m_p \| \check{m} \| \overline{M_1}$ with h_1 starting from IV, denote by x_{M_1} the final state. Compute the distance d_1 from x_{M_1} to x.
- Store $(d_1, \overline{M_1})$ in a table $\mathcal{T}_1$.

8. For each $\overline{M_2} \in \mathcal{M}_{MC2}$,
   (a) Process $\overline{M_2}$ with h_2 starting from $\check{y}$, denote by y_{M_2} the final state. Compute the distance d_2 from y_{M_2} to y.
   (b) For all $(d_1, \overline{M_1}) \in \mathcal{T}_1$, compute λ such that $\lambda \equiv d_1 \bmod 2^{\mu_1}$ and $\lambda \equiv d_2 \bmod 2^{\mu_2}$.
   (c) If $1 + \lambda + 2r + p \leq L'$, a second preimage is

$$m_1 \dots m_p \| \check{m} \| \overline{M_1} \| \beta^\lambda \| \overline{M_2} \| M_{\| L' - p - 2r - 1} \, .$$

Complexity analysis:

- The complexity of Step 1 is $\mathcal{O}(2^\ell)$.
- The complexity of Step 2 is $\mathcal{O}(2^{n/2})$ because 2^{μ_1} and 2^{μ_2} are relatively prime with constant probability.
- The complexity of Step 3 is $\mathcal{O}(2^t)$.
- The complexity of Step 4 is $\tilde{\mathcal{O}}(2^{n/2})$.
- The complexity of Step 5 is $\mathcal{O}(2^{\ell'})$.
- The complexity of Step 6 is $\mathcal{O}(2^{n-\ell})$.
- The complexity of Step 7 is $\mathcal{O}(2^{r+n-t})$.
- The complexity of Step 8 is $\mathcal{O}(2^r \cdot (2^{n-t} + 2^r))$.

At Step 8, a total of 2^{2r} candidates λ are computed. Since each λ is computed using the Sun-Qin theorem with two integers of size $\Theta(2^{n/2})$, each λ can be seen as uniformly drawn in $[0, 2^{\mu_1+\mu_2}]$. Since p is of the same order as L', the probability that λ verifies the condition $1+\lambda+2r+p \leq L'$ can be approximated by the probability that λ is of the same order as L', which is $2^{\ell' - \mu_1 - \mu_2}$. For the attack to be successful with constant probability, it is thus necessary that $2^{\mu_1+\mu_2-\ell'} \leq 2^{2r}$.

Trade-off with arbitrary β. With an arbitrary choice of β, we have $2^{\mu_1+\mu_2} \approx 2^{n/2+n/2} = 2^n$. The optimal trade-off is achieved with $\ell' = t = 3n/5$ and $r = n/5$. This attack has a total complexity $\mathcal{O}(2^{3n/5})$ obtained for challenges of length $2^{2n/5} \leq L \leq 2^{3n/5}$. It builds second preimages of length $\mathcal{O}(2^{3n/5})$ blocks, corresponding to the attack from [3].

Trade-off with small cycles. Instead of selecting β at random, the attacker selects a block β such that the graphs of $h_i(\cdot, \beta)$, $i = 1, 2$, have a main component with cycle lengths $2^{\mu_1}, 2^{\mu_2}$ at most $2^\mu \leq 2^{n/2}$. This corresponds to an initial step with complexity $\mathcal{O}(2^{3n/2-2\mu})$.

The optimal trade-off is achieved with $\mu = 11n/24$, $\ell' = t = 7n/12$ and $r = n/6$. Our improved attack thus has a total complexity $\mathcal{O}(2^{7n/12})$ obtained for challenges of length $2^{5n/12} \leq L \leq 2^{7n/12}$. It builds second preimages of length $\mathcal{O}(2^{7n/12})$ blocks. This attack is the best generic second preimage attack on Zipper Hash to the best of our knowledge.

7 Second preimage attack on Hash-Twice

We now consider Hash-Twice, a folklore construction that processes the message iteratively through two iterated hash functions $\mathcal{H}_1$ and $\mathcal{H}_2$. Hash-Twice is defined similarly to the Zipper Hash, but doesn't reverse the message in the second pass:

$$\mathcal{H}(M) = \mathcal{H}_2(\mathcal{H}_1(\mathrm{IV}, M), M) = g_2\big(h_2^*\big(g_1(h_1^*(\mathrm{IV}, M)), M\big)\big)\,.$$

The first second preimage attack on Hash-Twice was published in 2009 by Andreeva *et al.* [1]. It is a herding attack exploiting techniques originally used by [32] to attack a single hash function. The complexity of this attack depends on the challenge length, and can be optimized to achieve a complexity $\mathcal{O}(2^{3n/4})$. A second attack was published in [3], it exploits a wide variety of techniques such as Joux's multi-collisions technique, the diamond structure, the interchange structure and simultaneous expandable messages. Depending on the second preimage length, it also exploits deep iterates or multi-cycles techniques. The best attack has complexity $\tilde{\mathcal{O}}(2^{13n/22}) \approx \tilde{\mathcal{O}}(2^{0.591n})$ and constructs second preimages of length $2^{13n/22}$. In this Section, we show how to improve this attack using exceptional functions. It reaches a complexity of $\tilde{\mathcal{O}}(2^{15n/26}) \approx \tilde{\mathcal{O}}(2^{0.577n})$ for challenges and second preimages of the same length. This attack is, to the best of our knowledge, the best generic second preimage attack on Hash-Twice.

As in [3], we denote the sequence of internal states computed during the invocation of h_1 (resp. h_2) on M by $a_0, a_1, \ldots, a_L$ (resp. $b_0, b_1, \ldots, b_L$) with $a_0 = \mathrm{IV}$ and $b_0 = \mathcal{H}_1(\mathrm{IV}, M)$. We let $\ell = \log_2(L)$ where L is the length in blocks of the challenge. In Hash-Twice, contrarily to the case of the Zipper Hash, the padding is processed last. Since our second preimage attacks are based on finding a path that collides with the path of the challenge, we build a second preimage of the same length as the challenge.

7.1 Simple Attacks based on Exceptional Functions

The attacks of Sections 6.1 and 6.2 can easily be adapted to the case of Hash-Twice simply by adding a simultaneous expandable message in order to bypass the MD-strengthening. We do not go into further details as the adaptation is straightforward.

7.2 Improving the attack from [3] using Exceptional Functions

We give an alternative description of the best known attack [3], with complexity $\mathcal{O}(2^{13n/22})$, and an improvement using exceptional functions with complexity $\mathcal{O}(2^{15n/26})$. This attack is illustrated in Figure 8.

1. Compute the sequence of internal states $b_1, b_2, \ldots, b_L$.
2. Select a message block β. Find the main cycle of $h_1(\cdot, \beta)$ of length 2^{μ_1} and choose 2^u elements x_i, $1 \le i \le 2^u$, in the cycle. Find the main cycle of $h_2(\cdot, \beta)$ of length 2^{μ_2} and choose an element y in the cycle. Repeat this step if 2^{μ_1} and 2^{μ_2} are not relatively prime (μ_1 and μ_2 are not integers).

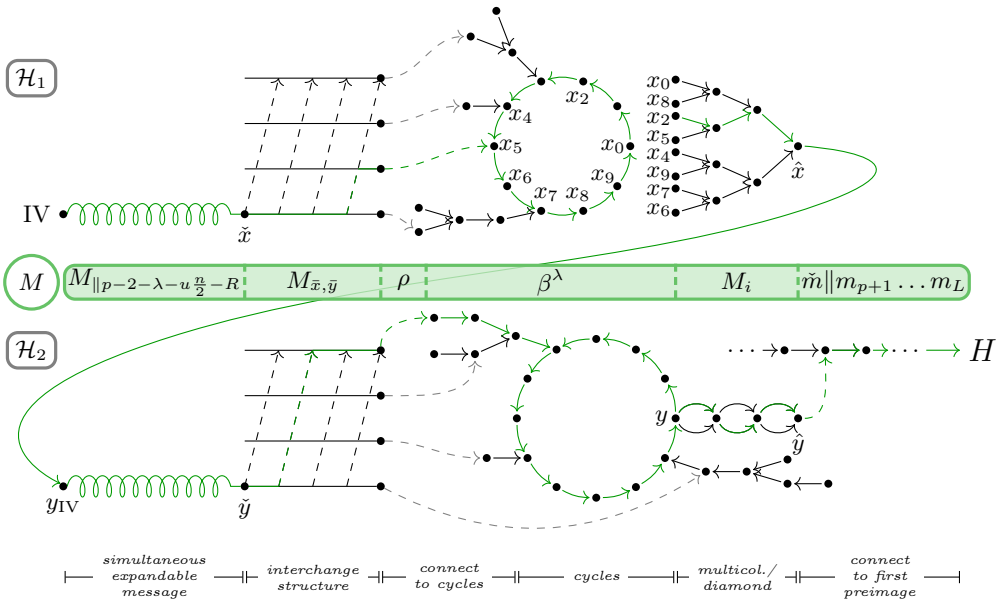

Fig. 8. Second preimage attack against Hash-twice from [3].

3. Precompute the distance from the main root of the main cycle of 2^t nodes in the graph of $h_1(\cdot,\beta)$ and in the graph of $h_2(\cdot,\beta)$ with $t \geq n/2$.
4. Build a $2^{u\cdot n/2}$-multicollision $\mathcal{M}_{MC}$ on h_2 using y as initial value. Note that all messages in $\mathcal{M}_{MC}$ have $u \cdot n/2$ blocks. Denote by $\hat{y}$ the final value of the multicollision. Using this multicollision, build a diamond structure $\mathcal{M}_{DS}$ using the cyclic nodes x_i, $1 \leq i \leq 2^u$ as starting points. Denote the end point by $\hat{x}$.
5. Compute $h_2(\hat{y}, m)$ with $m = 0, 1, \dots$ until a collision with some b_p, $1 \leq p \leq L-1$, is found for a message block $\check{m}$. Then, process $\check{m}||m_{p+1} \cdots m_L$ with h_1 using $\hat{x}$ as starting point to compute the initial value y_{IV} at the beginning of the h_2 processing.
6. Build an L-simultaneous expandable message $\mathcal{M}_{SEM}$ using as initial values $(\text{IV}, y_{\text{IV}})$. Denote by $(\check{x}, \check{y})$ the final states.
7. Build a 2^r-interchange structure $\mathcal{IS}$ starting from $(\check{x}, \check{y})$ and denote the set of end points by $\mathcal{X}$ and $\mathcal{Y}$.
8. Until a second preimage is found, select a message block ρ at random and do the following:
   (a) For each endpoint $\bar{x} \in \mathcal{X}$, compute the distance from $h(\bar{x}, \rho)$ to the main cycle of $h_1(\cdot, \beta)$. This provides directly the 2^u distances d_1^i, $1 \leq i \leq 2^u$ from $h(\bar{x}, \rho)$ to x_i. Store $(\bar{x}, d_{1,1}, \cdots, d_{1,2^u})$ in a table $\mathcal{T}_1$.
   (b) For each endpoint $\bar{y} \in \mathcal{Y}$:
      i. Compute the distance d_2 from $h(\bar{y}, \rho)$ to y.
      ii. For all $\bar{x}$ and for all $1 \leq i \leq 2^u$, using the Sun-Qin theorem, find λ such that $\lambda \equiv d_1 \bmod 2^{\mu_1}$ and $\lambda \equiv d_2 \bmod 2^{\mu_2}$. If $\lambda + 2 + u \cdot n/2 + R \leq$

p, with R the length of the message from the interchange structure, $M_{\|p-2-\lambda-u\frac{n}{2}-R}\|M_{\bar{x},\bar{y}}\|\rho\|\beta^{\lambda}\|M_i\|\check{m}\|m_{p+1}\dots m_L$ is a second preimage, where $M_{\bar{x},\bar{y}}$ is the message in the interchange structure that maps $(\check{x},\check{y})$ to $(\bar{x},\bar{y})$ and where M_i is the message corresponding to x_i in the diamond structure.

Complexity analysis: We first evaluate the number of iterations of Step 8. For each ρ, 2^{2r+u} candidates are computed. Since p is of the same order as L and R is of the order 2^{2r} (which we will show to be $\ll 2^{\ell}$), the probability that λ verifies the condition $\lambda + 2 + u \cdot n/2 + R \leq p$ is about $2^{\ell-\mu_1-\mu_2}$. Therefore we expect $\Theta(2^{\mu_1+\mu_2-\ell-2r-u})$ iterations.

- The complexity of Step 1 is $\mathcal{O}(2^{\ell})$.
- The complexity of Step 2 is $\mathcal{O}(2^{n/2})$ because 2^{μ_1} and 2^{μ_2} are relatively prime with constant probability.
- The complexity of Step 3 is $\mathcal{O}(2^{t})$.
- The complexity of Step 4 is $\tilde{\mathcal{O}}(2^{n/2+u/2})$.
- The complexity of Step 5 is $\mathcal{O}(2^{n-\ell})$.
- The complexity of Step 6 is $\max(\mathcal{O}(2^{\ell}), \tilde{\mathcal{O}}(2^{n/2}))$.
- The complexity of Step 7 is $\mathcal{O}(2^{n/2+2r})$.
- The complexity of Step 8 is $\mathcal{O}(2^{\mu_1+\mu_2-\ell-2r-u}(2^{r+n-t}+2^{2r+u}))$ $= \mathcal{O}(2^{\mu_1+\mu_2+n-\ell-r-u-t})$

Trade-off with arbitrary β. With an arbitrary choice of β, we have $2^{\mu_1+\mu_2} \approx 2^{n/2+n/2} = 2^n$. The optimal trade-off is achieved with $\ell = t = 13n/22$, $r = n/22$ and $u = 2n/11$. This results in an attack with complexity $\mathcal{O}(2^{13n/22}) \approx 2^{0.591n}$ assuming a challenge of the same length, corresponding to the attack from [3].

Trade-off with small cycles. Instead of selecting β at random, the attacker selects a block β such that the graphs of $h_i(\cdot,\beta)$, $i = 1, 2$, have a main component with cycle lengths $2^{\mu_1}, 2^{\mu_2}$ at most $2^{\mu} \leq 2^{n/2}$. This corresponds to an initial step with complexity $\mathcal{O}(2^{3n/2-2\mu})$.

The optimal trade-off is achieved with $\mu = 6n/13$, $\ell = t = 15n/26$, $r = n/26$ and $u = 2n/13$. This improved attack has a total complexity $\mathcal{O}(2^{15n/26}) \approx 2^{0.577n}$ which is obtained for challenges of length $\mathcal{O}(2^{15n/26})$ blocks. This attack is the best generic second preimage attack on Hash-Twice to the best of our knowledge.

8 Quantum Second Preimage against Hash-Twice

Finally, we consider generic attacks against combiners in the quantum setting. Several quantum attacks on hash combiners have been proposed in [4,22], but this does not include second preimage attacks on the Hash-Twice combiner.

Generically preimage search can be done in $\mathcal{O}\left(2^{n/2}\right)$ quantum time using Grover's search. Moreover, quantum cycle search costs at least $\mathcal{O}\left(2^{n/2}\right)$. Thus, we cannot apply the advanced techniques from this paper in a quantum setting.

The attack we propose is thus simpler. It is a quantization of the second preimage attack from [1] and is close to the herding attack from [22].

8.1 Preliminaries

We assume familiarity with the quantum computing model and refer to [42] for more details. In particular, our algorithms are ultimately written as quantum circuits, and "quantum time" will refer to the number of gates of these circuits.

Grover's algorithm [26] is well known to speed up exhaustive search problems quadratically. More generally, amplitude amplification [12] can be used to accelerate the recovery of any "good" output of a randomized algorithm, provided that we can run such algorithms in superposition. We only use these algorithms asymptotically, e.g., finding a preimage of an n-bit random function costs $\mathcal{O}\left(2^{n/2}\right)$ quantum evaluations of the function.

Any classical computation using T time and S space can be turned into a reversible computation using time $T^{1+\varepsilon}$ and space $\mathcal{O}\left(S \log T\right)$ [7,34] for any ε (where the constant in the $\mathcal{O}$ depends on ε). In particular, this allows to embed it as a quantum computation (since quantum circuits are, by definition, reversible). An important remark is that we can evaluate iterates of a random function f almost at the same time and space cost as classically.

Cycle Search. To the best of our knowledge, there is no quantum algorithm that finds a cycle of a random function f in time less than $\mathcal{O}\left(2^{n/2}\right)$, whatever the cycle size. The reason is that there is no quantum speedup on iterating the function f, and this cost compensates the speedup obtained through Grover's search. For example, looking for a fixed point costs $\mathcal{O}\left(2^{n/2}\right)$. More generally, looking for a cycle of length $\leq D$ requires to iterate f at least D times. Starting from a random point, the probability to fall on such a cycle after D iterates of f is about $\frac{D^2}{2^n}$, leading to $\mathcal{O}\left(2^{n/2}/D\right)$ iterates of quantum search – and $\mathcal{O}\left(2^{n/2}\right)$ evaluations of f in total, again.

Quantum collisions. Still, we can reuse part of the classical tools. One can compute 2^k collisions on the same function quantumly in time $\mathcal{O}\left(2^{2k/3+n/3}\right)$ instead of $\mathcal{O}\left(2^{k/2+n/2}\right)$ [11], which means we can construct multicollisions, expandable messages, diamond structures and interchange structures more efficiently, and in particular at a cost below the generic preimage quantum bound of $\mathcal{O}\left(2^{n/2}\right)$.

8.2 Quantum Attack

The attack follows the same steps as [1]. Recall that the adversary is given a challenge message $M = m_1 \dots m_L$ and seeks M' such that $\mathcal{H}_2(\mathcal{H}_1(IV, M), M) = \mathcal{H}_2(\mathcal{H}_1(\mathrm{IV}, M'), M')$. We note $\ell = \log_2(L)$. The attack is described in Figure 9.

1. Compute $b_1, \dots, b_L$, the sequence of internal states of $\mathcal{H}_2$ in the computation of $\mathcal{H}_2(\mathcal{H}_1(\mathrm{IV}, M), M)$.
2. Build an expandable message for h_1 starting from IV. Let $\check{x}$ be the end state.
3. Build a $2^{n-r+rn/2}$-multicollision on h_1 starting from $\check{x}$. Denote by $\hat{x}$ the end state.

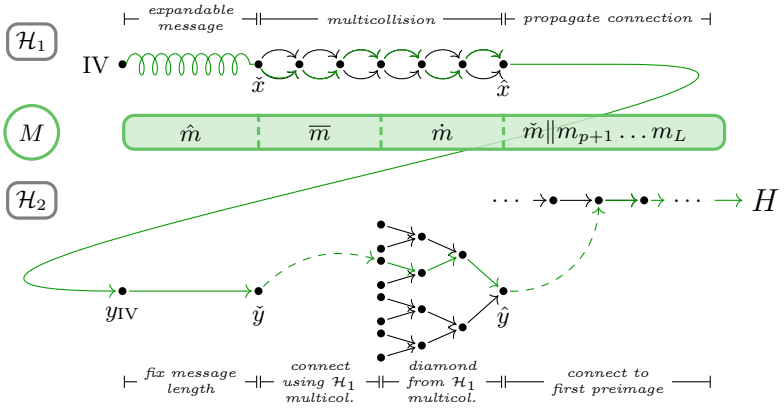

Fig. 9. Second-preimage attack against Hash-Twice from [1]

4. Build a 2^r-diamond structure on h_2 with random starting points that use the last $rn/2$ blocks of the previous multicollision. Let $\hat{y}$ be the end state.
5. From $\hat{y}$, find a message $\check{m}$ that connects it to one of the b_i. Denote by p the corresponding index.

5a. Compute $y_{\text{IV}} = h_1^*(\hat{x}, \check{m}\|m_{p+1}\dots m_L)$.

5b. Fix the expandable message to $\hat{m}$, such that the length of $\hat{m}$ plus the multicollision chain and $\check{m}$ is p.

5c. Compute $\check{y} = h_2^*(y_{\text{IV}}, \hat{m})$.

6. Find a message $\overline{m}$ built from the first $n-r$ blocks of the multicollision that connects $\check{y}$ to the diamond structure using h_2. Denote by $\dot{m}$ the message that connects the leaf of the diamond to $\hat{y}$.

6a. The second preimage is $\hat{m}||\overline{m}||\dot{m}||\check{m}||m_{p+1}\dots m_L$.

Quantum complexity:

- The complexity of Step 1 is $\mathcal{O}\left(2^\ell\right)$.
- The complexity of Step 2 is $\mathcal{O}\left(2^\ell\right)$ if $\ell > n/3$ (BHT collision-finding [13] finds one collision in time $\mathcal{O}\left(2^{n/3}\right)$).
- The complexity of Step 3 is $\tilde{\mathcal{O}}\left(2^{n/3}\right)$ (same reason).
- The complexity of Step 4 is $\tilde{\mathcal{O}}\left(2^{2r/3+n/3}\right)$ using [6].
- The complexity of Step 5 is $\mathcal{O}\left(2^{(n-\ell)/2}\right)$ using Grover search (2^ℓ solutions, which are the b_i, among 2^n).
- The complexity of Step 6 is $\mathcal{O}\left(2^{(n-r)/2}\right)$ using Grover search (2^r solutions, which are the leaves of the diamond structure, among 2^n).

Overall, with $r = \ell = n/7$, we reach a quantum cost of $\tilde{\mathcal{O}}\left(2^{3n/7}\right)$.

Acknowledgments. This work has been partially supported by the French Agence Nationale de la Recherche through the OREO project under Contract

ANR-22-CE39-0015, and through the France 2030 program under grant agreement No. ANR-22-PETQ-0007 EPiQ, ANR-22-PETQ-0008 PQ-TLS and ANR-22-PECY-0010 CRYPTANALYSE.

A Brief description of Brent's algorithm

Let f be a function in $\mathfrak{F}_{2^n}$, and x be a node in its functional graph. Cycle-finding algorithms allow to recover $\mu(x)$, the cycle length of x and an element e of this cycle, using iterated function values $x_i := f^i(x)$ for $i \geq 0$.

In more details, Brent's algorithm works as follows [31]. It uses two variables *tortoise* and *hare*. First, an integer variable i is set to 0, and *tortoise* is set to $x_0 = x$. Then, at each step j, *hare* successively takes the value of x_{2^i+j} for $j = 1, \ldots, 2^i$ and is compared to *tortoise*. If a value j is found such that *hare* = *tortoise*, or in other words $x_{2^i} = x_{2^i+j}$, then a node in the cycle has been found, and *hare* is returned. Otherwise, *tortoise* takes the value of *hare*, and i is incremented. At the end of this step, 2^i is the *smallest* power of two such that $2^i \geq \max(\mu(x), \lambda(x))$, and j is exactly $\mu(x)$. Thus, Brent's algorithm has a complexity smaller than $3\max(\mu(x), \lambda(x))$ applications of f. For a random node in a random function in $\mathfrak{F}_{2^n}$, both $\mu(x)$ and $\lambda(x)$ are expected to have length $\mathcal{O}(2^{\frac{n}{2}})$. Thus, the average complexity of Floyd is $\mathcal{O}(2^{\frac{n}{2}})$ applications of said random function.

References

1. Andreeva, E., Bouillaguet, C., Dunkelman, O., Kelsey, J.: Herding, second preimage and trojan message attacks beyond Merkle-Damgård. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009: 16th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5867, pp. 393–414. Springer, Heidelberg, Germany, Calgary, Alberta, Canada (Aug 13–14, 2009). https://doi.org/10.1007/978-3-642-05445-7_25
2. Aumasson, J.P., Jovanovic, P., Neves, S.: NORX v3. Submission to the Caesar competition (2016), https://competitions.cr.yp.to/round3/norxv30.pdf
3. Bao, Z., Dinur, I., Guo, J., Leurent, G., Wang, L.: Generic attacks on hash combiners. Journal of Cryptology **33**(3), 742–823 (Jul 2020). https://doi.org/10.1007/s00145-019-09328-w
4. Bao, Z., Guo, J., Li, S., Pham, P.: Evaluating the security of merkle-damgård hash functions and combiners in quantum settings. In: NSS. Lecture Notes in Computer Science, vol. 13787, pp. 687–711. Springer (2022). https://doi.org/10.1007/978-3-031-23020-2_39
5. Bao, Z., Wang, L., Guo, J., Gu, D.: Functional graph revisited: Updates on (second) preimage attacks on hash combiners. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 404–427. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63715-0_14
6. Benedikt, B.J., Fischlin, M., Huppert, M.: Nostradamus goes quantum. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part III.

Lecture Notes in Computer Science, vol. 13793, pp. 583–613. Springer, Heidelberg, Germany, Taipei, Taiwan (Dec 5–9, 2022). https://doi.org/10.1007/978-3-031-22969-5_20
7. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**(4), 766–776 (1989)
8. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011: 18th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 7118, pp. 320–337. Springer, Heidelberg, Germany, Toronto, Ontario, Canada (Aug 11–12, 2012). https://doi.org/10.1007/978-3-642-28496-0_19
9. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Cryptographic sponge functions (2011), https://keccak.team/files/CSF-0.1.pdf
10. Blackburn, S.R., Stinson, D.R., Upadhyay, J.: On the complexity of the herding attack and some related attacks on hash functions. Designs, Codes and Cryptography **64**, 171–193 (2012)
11. Bonnetain, X., Chailloux, A., Schrottenloher, A., Shen, Y.: Finding many collisions via reusable quantum walks: Application to lattice sieving. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 221–251. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30589-4_8
12. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemporary Mathematics **305**, 53–74 (2002)
13. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN '98: Theoretical Informatics, Third Latin American Symposium, Campinas, Brazil, April, 20-24, 1998, Proceedings. Lecture Notes in Computer Science, vol. 1380, pp. 163–169. Springer (1998). https://doi.org/10.1007/BFB0054319, https://doi.org/10.1007/BFb0054319
14. Daemen, J., Mennink, B., Van Assche, G.: Full-state keyed duplex with built-in multi-user support. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 606–637. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_21, https://doi.org/10.1007/978-3-319-70697-9_21
15. Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) Advances in Cryptology – CRYPTO'89. Lecture Notes in Computer Science, vol. 435, pp. 416–427. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990). https://doi.org/10.1007/0-387-34805-0_39
16. Dean, R.D.: Formal aspects of mobile code security. Ph.D. thesis (1999)
17. DeLaurentis, J.M.: Components and cycles of a random function. In: Pomerance, C. (ed.) Advances in Cryptology – CRYPTO'87. Lecture Notes in Computer Science, vol. 293, pp. 231–242. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 1988). https://doi.org/10.1007/3-540-48184-2_21
18. Dierks, T., Allen, C.: RFC 2246 - The TLS Protocol Version 1.0. Internet Activities Board (Jan 1999)
19. Dinur, I.: New attacks on the concatenation and XOR hash combiners. In: Fischlin, M., Coron, J.S. (eds.) Advances in Cryptology – EUROCRYPT 2016, Part I.

Lecture Notes in Computer Science, vol. 9665, pp. 484–508. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49890-3_19
20. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Memory-efficient algorithms for finding needles in haystacks. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology – CRYPTO 2016, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 185–206. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53008-5_7
21. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. Journal of Cryptology **34**(3), 33 (Jul 2021). https://doi.org/10.1007/s00145-021-09398-9
22. Dong, X., Li, S., Pham, P., Zhang, G.: Quantum attacks on hash constructions with low quantum random access memory. In: ASIACRYPT (3). Lecture Notes in Computer Science, vol. 14440, pp. 3–33. Springer (2023). https://doi.org/10.1007/978-981-99-8727-6_1
23. Flajolet, P., Odlyzko, A.M.: Random mapping statistics. In: Quisquater, J.J., Vandewalle, J. (eds.) Advances in Cryptology – EUROCRYPT'89. Lecture Notes in Computer Science, vol. 434, pp. 329–354. Springer, Heidelberg, Germany, Houthalen, Belgium (Apr 10–13, 1990). https://doi.org/10.1007/3-540-46885-4_34
24. Flajolet, P., Sedgewick, R.: Analytic Combinatorics. Cambridge University Press (2009), http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521898065
25. Gilbert, H., Boissier, R.H., Khati, L., Rotella, Y.: Generic attack on duplex-based AEAD modes using random function statistics. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part IV. Lecture Notes in Computer Science, vol. 14007, pp. 348–378. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30634-1_12
26. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: 28th Annual ACM Symposium on Theory of Computing. pp. 212–219. ACM Press, Philadephia, PA, USA (May 22–24, 1996). https://doi.org/10.1145/237814.237866
27. Guo, J., Peyrin, T., Sasaki, Y., Wang, L.: Updates on generic attacks against HMAC and NMAC. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 131–148. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2014). https://doi.org/10.1007/978-3-662-44371-2_8
28. Harris, B.: Probability Distributions Related to Random Mappings. The Annals of Mathematical Statistics **31**(4), 1045 – 1062 (1960). https://doi.org/10.1214/aoms/1177705677, https://doi.org/10.1214/aoms/1177705677
29. Jha, A., Nandi, M.: Some cryptanalytic results on zipper hash and concatenated hash. Cryptology ePrint Archive, Paper 2015/973 (2015), https://eprint.iacr.org/2015/973, https://eprint.iacr.org/2015/973
30. Joux, A.: Multicollisions in iterated hash functions. Application to cascaded constructions. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. Lecture Notes in Computer Science, vol. 3152, pp. 306–316. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 15–19, 2004). https://doi.org/10.1007/978-3-540-28628-8_19
31. Joux, A.: Algorithmic Cryptanalysis. Chapman and Hall/CRC (2009). https://doi.org/https://doi.org/10.1201/9781420070033

32. Kelsey, J., Kohno, T.: Herding hash functions and the Nostradamus attack. In: Vaudenay, S. (ed.) Advances in Cryptology – EUROCRYPT 2006. Lecture Notes in Computer Science, vol. 4004, pp. 183–200. Springer, Heidelberg, Germany, St. Petersburg, Russia (May 28 – Jun 1, 2006). https://doi.org/10.1007/11761679_12
33. Kelsey, J., Schneier, B.: Second preimages on n-bit hash functions for much less than 2n work. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. Lecture Notes in Computer Science, vol. 3494, pp. 474–490. Springer, Heidelberg, Germany, Aarhus, Denmark (May 22–26, 2005). https://doi.org/10.1007/11426639_28
34. Knill, E.: An analysis of bennett's pebble game. CoRR **abs/math/9508218** (1995)
35. Lefevre, C.: A note on adversarial online complexity in security proofs of duplex-based authenticated encryption modes. soon to appear on Eprint (2024)
36. Leurent, G., Peyrin, T., Wang, L.: New generic attacks against hash-based MACs. In: Sako, K., Sarkar, P. (eds.) Advances in Cryptology – ASIACRYPT 2013, Part II. Lecture Notes in Computer Science, vol. 8270, pp. 1–20. Springer, Heidelberg, Germany, Bengalore, India (Dec 1–5, 2013). https://doi.org/10.1007/978-3-642-42045-0_1
37. Leurent, G., Wang, L.: The sum can be weaker than each part. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 345–367. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46800-5_14
38. Liskov, M.: Constructing an ideal hash function from weak ideal compression functions. In: Biham, E., Youssef, A.M. (eds.) SAC 2006: 13th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 4356, pp. 358–375. Springer, Heidelberg, Germany, Montreal, Canada (Aug 17–18, 2007). https://doi.org/10.1007/978-3-540-74462-7_25
39. Merkle, R.C.: Fast software encryption functions. In: Menezes, A.J., Vanstone, S.A. (eds.) Advances in Cryptology – CRYPTO'90. Lecture Notes in Computer Science, vol. 537, pp. 476–501. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 11–15, 1991). https://doi.org/10.1007/3-540-38424-3_34
40. Moon, J.W.: Counting Labelled Trees. Canadian Mathematical Congress 1970, William Clowes and Sons (1970)
41. Peyrin, T., Wang, L.: Generic universal forgery attack on iterative hash-based MACs. In: Nguyen, P.Q., Oswald, E. (eds.) Advances in Cryptology – EUROCRYPT 2014. Lecture Notes in Computer Science, vol. 8441, pp. 147–164. Springer, Heidelberg, Germany, Copenhagen, Denmark (May 11–15, 2014). https://doi.org/10.1007/978-3-642-55220-5_9
42. de Wolf, R.: Quantum computing: Lecture notes (2019)