# Some Cryptanalytic Results on Zipper Hash and Concatenated Hash

Ashwin Jha and Mridul Nandi

Indian Statistical Institute, Kolkata
ashwin.jha1991@gmail.com, mridul.nandi@gmail.com

**Abstract.** At SAC 2006, Liskov proposed the zipper hash, a technique for constructing secure (indifferentiable from random oracles) hash functions based on weak (invertible) compression functions. Zipper hash is a two pass scheme, which makes it unfit for practical consideration. But, from the theoretical point of view it seemed to be secure, as it had resisted standard attacks for long. Recently, Andreeva *et al.* gave a forced-suffix herding attack on the zipper hash, and Chen and Jin showed a second preimage attack provided $f_1$ is strong invertible. In this paper, we analyse the construction under the random oracle model as well as when the underlying compression functions have some weakness. We show (second) preimage, and herding attacks on an $n$-bit zipper hash and its relaxed variant with $f_1 = f_2$, all of which require less than $2^n$ online computations.
Hoch and Shamir have shown that the concatenated hash offers only $\frac{n}{2}$-bits security when both the underlying compression functions are strong invertible. We show that the bound is tight even when only one of the underlying compression functions is strong invertible.

**Keywords**. zipper hash, concatenated hash, time/memory trade-off, (second) preimage, herding attack.

## 1 Introduction

Cryptographic hash functions play a very important role in modern cryptography. These hash functions have many information security applications especially in digital signatures [7], encryption [6], authentication [5] and in many cryptographic protocols such as password hashing [39], bitcoins [35] etc. An $n$-bits hash function $H : \{0,1\}^* \to \{0,1\}^n$, takes an arbitrary length input (message) and produces an n-bits output (digest). Hash functions are used in many different settings and accordingly their security requirement changes. The three most general security notions are:

**Collision Resistance.** It should be hard to find two messages $M \neq M'$, such that $H(M) = H(M')$.
**Second Preimage Resistance.** Given a message $M$, it should be hard to find another message $M' \neq M$, such that $H(M') = H(M)$.
**Preimage Resistance.** Given a hash value $h$, it should be hard to find a message $M$, such that $H(M) = h$.

For a random oracle the collision attack requires $2^{n/2}$ computations, and the preimage and second preimage attacks require $2^n$ computations. A secure hash function is expected to offer the same level of security.

Among the different hash function designs, Merkle-Damgård [34, 16] or MD hash function has been the most popular and widely used design. Naturally, it is the most studied design as well. The cryptanalysis of MD hash has revealed several weaknesses in the design. Dean [17] first showed that the fixed points (i.e., $f(h, m) = h$) in the underlying compression function can be used for a long message second preimage attack in $O(2^{n/2})$ complexity. Later, the seminal work by Joux [25] suggested a new efficient way to construct **multicollision**[1] on the MD hash. Immediately after his work, Kelsey and Schneier [28] applied the ideas of Joux attack to Deans attack, and eliminated the requirement of fixed points in the compression function by building a new structure called $[a, b]$**-expandable message**.[2]

Another result in the same line of research is the herding attack by Kelsey and Kohno [27]. The attack is a chosen-target forced prefix (CTFP) attack, i.e., the adversary first commits to a digest value $h$ and is then presented with a challenge prefix $P$. Now, the adversary needs to efficiently compute a suffix $S$, such that $H(P\|S) = h$. Andreeva *et al.* [3] gave a modified version of herding attack on the zipper hash, called chosen-target forced-suffix (CTFS) attack in which the adversary is presented with a challenge suffix. The underlying technique used in these attacks, requires a precomputed structure called **diamond structure**, which is again a special form of multicollision and requires $O(2^{\frac{n}{2}+\frac{k}{2}})$ computations.

Inherent weeknesses (such as invertibility, collision and fixed point property) in specific hash function definitions can be used to construct efficient practical attacks. This is evident from the attacks by Wang *et al.* [41, 43, 42, 44], Biham *et al.* [8], Klima [29] and Joux *et al.* [26] on hash functions based on the MD4 design, such as MD5, RIPEMD, SHA-0 and SHA-1. Several popular compression functions such as Davis-Meyer, Miyaguchi-Preneel etc. are based on block ciphers. Preneel *et al.* [38] first studied the 64 most basic compression functions based on block ciphers. Black *et al.* [12] further proved that among the 64 choices 12 provide full collision and preimage security and 8 of the remaining 52 provide full collision security. Interestingly, many of these 20 secure constructions have one or more weaknesses (such as Davis-Meyer which offers efficient fixed point computation). These facts are compelling enough to analyse the security of a hash design in the presence of weaker compression functions [22].

To preclude the aforementioned attacks on the Merkle-Damgård design, many variants were proposed, such as the concatenated hash, HAIFA [9] construction by Biham *et al.* and the dithered Merkle-Damgård hash by Rivest [13]. It has been shown that these designs are not fully secure [25, 20, 1, 4], and their

[1] A set $\mathcal{M}$ of size $r$ is said to be a $r$-multicollision set for $H$ if $\forall\, M, M' \in \mathcal{M}$, $H(M) = H(M')$.

[2] A multicollision set $\mathcal{M}$ such that $\forall \ell \in [a, b]$, $\exists\, M \in \mathcal{M}$ with $\|M\| = \ell$, the number of message blocks in $M$.

security is implementation dependent. At SAC 2006, Liskov proposed the zipper hash [32] and argued that it is indifferentiable [15, 33] with random oracle even when the underlying compression functions have some weaknesses. To formulate the model for weakness, the adversary is given access to some oracles which solve some tasks. For example, a **strong invertible** oracle returns $h$ given $(h', m)$, such that $f(h, m) = h'$. Whereas, a **weak invertible** oracle returns $(h, m)$ for a given $h'$ such that $f(h, m) = h'$. In addition to these, in this paper, we consider one more oracle which returns a fixed point $h$ given $m$.

The analysis of zipper hash has seen relatively less interest due to the fact that it is a two pass scheme, and hence practically inefficient. But theoretically it is intriguing that the design withstands the standard attack techniques. The first significant attack on this design was given by Andreeva *et al.* [3], which talks about the chosen-target forced-suffix attack. Recently, Chen *et al.* [14] showed a second preimage attack with the assumption that the first pass compression function is strongly invertible. Further, the multicollision attacks by Nandi *et al.* [36] are also applicable on the zipper hash. In light of these recent developments, it would be interesting to investigate the (second) preimage and herding security of this design under various assumptions on the underlying compression functions.

### 1.1 Our Contribution

In this paper, we demonstrate (second) preimage and herding attacks on an $n$-bit zipper hash and its relaxed variant with $f_1 = f_2$, all of which require less than $2^n$ online computations.

**Second preimage attack.** Our second preimage attacks are applied in the following cases: (i) we precompute a specific structure of size $O(n)$ (which requires $O(2^n)$ computations for a general compression function and $O(2^{3n/4})$ computations when multiple fixed points are available) or (ii) relaxed variant of zipper hash (i.e., $f_1 = f_2$).

**Preimage attack.** We provide a preimage attack given that the $f_2$ is strong invertible. We also demonstrate preimage attacks under weaker assumption on $f_2$, namely $f_2$ is weak invertible (i.e., easy to invert for a target chaining value but the adversary has no control on choosing the message block), and one of the above cases for the second preimage holds.

**Herding attack.** Under the assumption of strong invertible property for $f_2$, we obtain a forced-prefix herding attack.

Apart from our attacks on the zipper hash we also give two attacks on the concatenated hash under some weakness assumptions on the underlying compression functions.

### 1.2 Organisation of the Paper

We start off with fixing the preliminary ideas and notations in section 2. In section 3, we briefly describe the various attack structures that are used for hash

function analysis and summarise the two known attacks on the zipper hash. In sections 4, 5, and 6 we present our attacks on the zipper hash, relaxed zipper hash, and the concatenated hash respectively. Section **??** concludes the paper, where we summarise our results.

## 2 Notations and Basic Definitions

### 2.1 Notations

For any $n \in \mathbb{N}$, $\{0,1\}^n$ represents the set of all $n$-bit binary strings. $\{0,1\}^+$ denotes the set of all binary strings of length $> 0$, and $\{0,1\}^*$ denotes $\{0,1\}^+ \cup \{\Lambda\}$, where $\Lambda$ represents the empty string. For any $x \in \{0,1\}^n$, $|x|$ denotes the length (in no. of blocks) of $x$, and $\langle x \rangle_k$ denotes the $k$-bit binary representation of $|x|$. For $x, y \in \{0,1\}^n$, $x \| y$ represents the concatenation of $x$ and $y$. We denote the padded version of a message $m \in \{0,1\}^*$ by $\overline{m} := (m_1, m_2, \cdots, m_\ell)$, where $m_i$ denotes the individual message block for $1 \leq i \leq \ell$. For a message block $m$, $m^i$ represents $i$ consecutive concatenations of $m$. Throughout this paper we will assume that the second preimage adversary is challenged with a message of length $2^t$ blocks.

### 2.2 Iterated Function

Given any compression function $f : \{0,1\}^n \times \{0,1\}^{n'} \to \{0,1\}^n$ and a predefined $n$-bit constant $iv$, referred as initial value, the *iterated hash* $f^+(iv;\ \cdot) : (\{0,1\}^{n'})^+ \to \{0,1\}^n$ is defined by iterating $f$ sequentially as follows:

$$f^+(iv;\ m_1, m_2, \ldots, m_\ell) = h_\ell := f(\cdots f(f(iv, m_1), m_2) \cdots).$$

We can also write $h_i = f(h_{i-1}, m_i)$ where $h_0 = iv$, $1 \leq i \leq \ell$.

A directed arc-labelled graph $G$ corresponding to a compression function $f$ has the set of vertices $V = \{0,1\}^n$, the set of labels $L = \{0,1\}^{n'}$ and whenever $f(u,m) = v$ we have a label $m$ for the arc $(u,v)$. We write $u \xrightarrow{m} v$. Thus, the following sub-graph (we also refer as structure) represents the computation of the iterated function $f^+$. We may have $h_i = h_j$ and so the above structure is a walk and not necessarily a path.


Padding Function. A block-parsing function $\xi : \{0,1\}^* \to (\{0,1\}^{n'})^+$ is an injective function. It is said to be a *padding function* if there exists a function $P : \{0,1\}^* \to \{0,1\}^*$ such that for all $m$, $\xi(m) = m \| P(m)$. The following are two popular examples of padding functions in which $d$ is chosen to be the smallest non-negative integer such that $n'$ divides $|P(m)| + |m|$: (i) $P_{\text{ozp}} = 10^d$ and (ii) $P_{\text{sMD}} = 10^d \| \langle m \rangle_{64}$ where $\langle m \rangle_{64}$ denotes the 64-bit binary representation of the

size $|m|$. Note that we can apply the later padding function to all messages of sizes at most $2^{64}$ which is more than sufficient for all practical messages. Unless mentioned explicitly, we fix an initial value iv and a padding rule $P$. We also denote $m\|P(m)$ by $\overline{m} := (m_1, \ldots, m_\ell)$ for some $\ell$. When $P = P_{\mathrm{sMD}}$, the composition function $f^+ \circ P$ is known as MD hash function, proposed independently by Merkle [34] and Damgård [16].

### 2.3 Zipper Hash

Conceptually, the Zipper hash [32] is a two-pass hash function that processes a message $m \in \{0,1\}^*$ in the first pass and its block-wise reverse in the second pass. Though zipper hash was originally defined to create secure hash functions from weak compression functions, we will be considering the zipper hash design on a general compression function not necessarily weak. Fig. 1 shows the graphical description of zipper hash function. Note that $h_\ell$ is the hash output for the single-pass and it is actually the MD hash output.

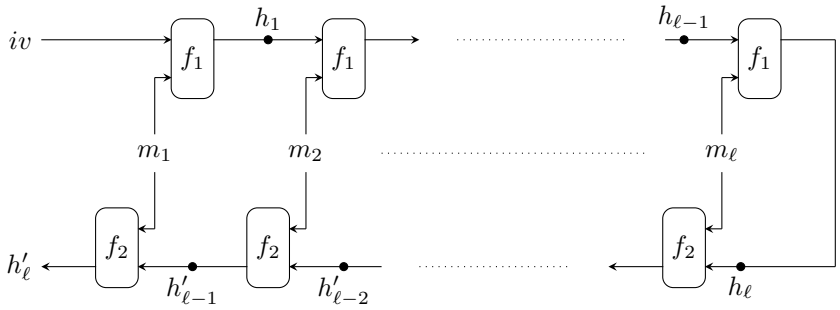


**Fig. 1:** Computation of the zipper hash $\mathcal{Z}(M)$: Here we apply some padding rule to obtain message blocks $m_1, \ldots, m_\ell$. $h'_\ell$ is the final output of the zipper hash.

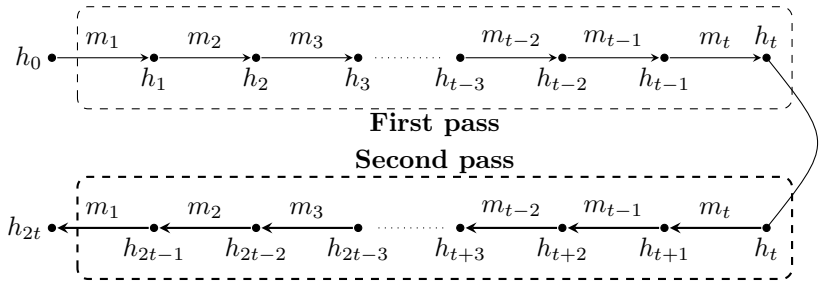


**Fig. 2:** Graphical Description of Zipper Hash Function where the first and second pass are computed through $f_1$ and $f_2$ respectively. When we choose $f_1 = f_2$, we call it the relaxed zipper hash.

Formally, an $n$-bit zipper hash function $\mathcal{Z}$ based on two compression functions $f_1$ and $f_2$ (and a fixed initial value $iv$) can be defined as

$$\mathcal{Z}(m) := f_2^+(f_1^+(iv;\ \overline{m}); \overline{m}^{\mathrm{rev}})$$

where $\overline{m} = (m_1, \ldots, m_\ell)$ and $\overline{m}^{\text{rev}} = (m_\ell, \ldots, m_1)$ is the block-wise reverse of $\overline{m}$. Fig. 2 shows graphical description of the zipper hash function. Throughout this paper, we will denote the compression function of first pass and second pass by $f_1$ and $f_2$ respectively, and distinguish the corresponding arc labels by using thick lines for the second pass, and denote the junction point between the two passes by an unfilled node (i.e. $\circ$).

### 2.4 A Note on TMTO Attacks

One-wayness of a function is the most fundamental problem in cryptography [18]. Hellman [21] first showed a *preimage attack* or in other words, broke the one-wayness of any function $H$ (i.e., *given $h$, to find $M$ such that $H(M) = h$*). In his attack, the online time $T$ and memory $M$ satisfy the relation $TM^2 = 2^{2n}$. Consequently, the attack is known as a time/memory trade-off (TMTO) algorithm. Note that in the TMTO set-up, the precomputation time is not considered, since this is a one-time offline activity which can be performed at the cryptanalyst's leisure. Several efforts have been made to improve the time/memory trade-off curve [11, 37, 23, 19, 10]. However, most of the improvement either assumes multiple targets or works for a specific structure of a one-way function, e.g., stream ciphers [19, 11].

### 2.5 A Note on Notions of Invertibility

**Weak invertible Function.** A compression function $f : \{0,1\}^n \times \{0,1\}^{n'} \to \{0,1\}^n$ is said to be weak invertible if given $z$ it is easy to compute a random $(x, y)$ pair, such that $f(x, y) = z$, i.e., the adversary can get random preimages at each invocation. Four of the eight group 2 compression functions ($f_{13}, f_{14}, f_{16}$ and $f_{18}$ in [12]) of PGV function family [38] are weak invertible.

**Strong invertible Function.** A compression function $f : \{0,1\}^n \times \{0,1\}^{n'} \to \{0,1\}^n$ is said to be strong invertible if,

- it has a backward interface [22, 31], i.e., given $y, z$ it is easy to compute $x$, such that $f(x, y) = z$. Or,
- it has a bridging interface [22, 31], i.e., given $x, z$ it is easy to compute $y$, such that $f(x, y) = z$.

In other words, the adversary has the additional power to set one of the value as per need. Four of the eight group 2 compression functions ($f_{15}, f_{17}, f_{19}$ and $f_{20}$ in [12]) of PGV function family [38] are strong invertible. Observe that a strong invertible function can simulate a weak invertible funtion, and thus, it is a stronger notion of invertibility.

## 3 Attack Structures and Known Results

### 3.1 Some Attack Structures

**Joux Multicollision.** Joux in his seminal paper [25], gave a novel method to get $2^k$-multicollision set of $f^+$ by successively applying birthday attack $k$

times (as illustrated in the figure below). Note that, the structure does not need space for all the $2^k$ elements, as it can be simply described by only $2k$ message blocks. The number of computations for this structure is $O(k \cdot 2^{n/2})$.

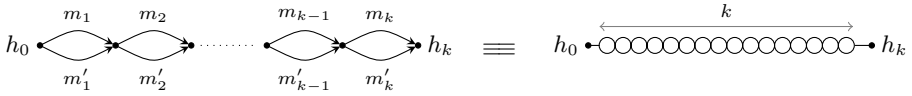


**Fig. 3:** Joux multicollision structure and its shorthand representation in R.H.S.

**Kelsey-Schneier Expandable Message.** Kelsey and Schneier extended the Joux multicollision idea to get an expandable message set (see the illustration below and [28] for details). Like Joux multicollision, we can also represent this set by about $2k + 1$ message blocks. The construction of this structure requires $O(2^k + k \cdot 2^{n/2})$ computations.

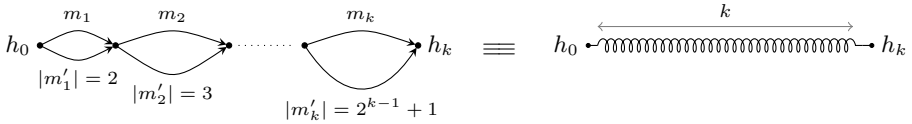


**Fig. 4:** Kelsey Schneier expandable message and its shorthand representation in R.H.S.

**Diamond Structure.** A $2^k$ Diamond $\mathcal{D}$ is a complete binary tree with $2^k$ leaf nodes, where each node represents a chaining value and the directed arc $(h, h')$ with label $m$ represents the transition $f(h, m) = h'$. The set of leaves and the root node of $\mathcal{D}$ will be denoted by $\mathcal{D}_l$ and $h_{\mathcal{D}}$, respectively. Based on the direction of edges, we can have two different types of diamond structure:

1. *Converging Diamond* [27], where the direction of arcs is from the leaves to the root node. It takes $O(2^{\frac{n}{2}+\frac{k}{2}})$ computations to build this structure using forward queries only [27, 30]. If the adversary has access to a strongly invertible interface, then this structure can be built in $O(2^k)$ backward computations. In this later case, this structure is also referred as *inverse diamond* [14].
2. *Diverging Diamond*, where the direction of edges is from the root node to the leaves. It takes $O(2^k)$ computations to build this structure using forward queries only.

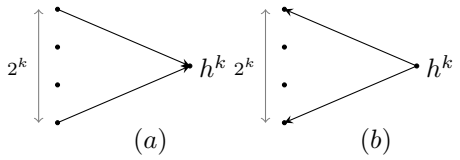


**Fig. 5:** Shorthand for (a) $2^k$-converging diamond, (b) $2^k$-diverging diamond.

**Collision between Lists**. Given two lists $L_1$ and $L_2$ of size $2^k$ and $2^{n-k}$ respectively, with high probability we can find a collision between these lists. Note that one of the lists can be generated run-time while finding collision. This

technique is called hitting and is very useful in finding a linking string[3] from a random chaining value to a chain. Meet-in-the-middle is a special case of the two list collision in which, one of the list is created by making forward queries and the other list by backward queries. This technique is extensively used for constructing attacks or structures.
Apart from these, several other examples of attack structures are available in the literature. Kite structure [4, 40] (which is nothing but a combination of diverging and converging diamond structures) has been used to construct second preimage attack on Dithered hash and Hash Twice. Similarly, Interchange structure [31] has been used by Leurent and Wang to construct a preimage attack in less than $2^n$ online computations on XOR combiners.

### 3.2 Known Attacks on Zipper Hash

**Chosen Target Forced Suffix Attack.** Andreeva *et al.* [3] proposed a modified version of herding attack which works on forced-suffix. The attack is illustrated in fig. 6. In the offline phase, the adversary first creates a $(k \cdot \frac{n}{2} + (n-k))$-cycles Joux multicollision $\mathcal{J}$ with endpoint $h_a$, using $f_1$ computations. The first $k \cdot \frac{n}{2}$ cycles of $\mathcal{J}$ are used in reverse, to build a $2^k$-converging diamond $\mathcal{D}$ using $f_2$ computations, and $h_{\mathcal{D}}$ is committed as the target hash value. In the online phase, after getting a suffix $S$ from the challenger, the adversary first computes $h_c = f_2^+(f_1^+(h_a, S), S^{rev})$, and then uses the last $(n-k)$ cycles of $\mathcal{J}$ in reverse, to get a linking message to $\mathcal{D}$. The path from $h_c$ to $h_{\mathcal{D}}$ gives the required prefix.

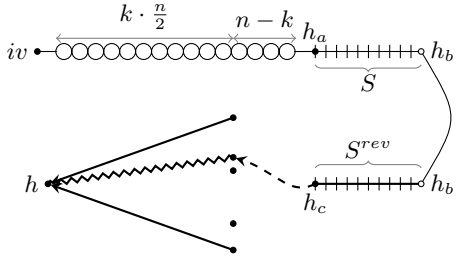


**Fig. 6:** Herding Attack on Zipper Hash [3]

**Second Preimage Attack with Strong Invertible $f_1$.** Chen *et al.* [14] proposed a second preimage attack on Zipper hash with strong invertible $f_1$. As illustrated in fig. 7, the attack works in two stages. In the offline phase, the adversary starts from a random chaining value $h_m$ and creates an $n$-cycles Joux multicollision $\mathcal{J}$ with endpoint $h_a$, using $f_2$ computations. The adversary then creates two lists $L_1$ and $L_2$ of $2^{n/2}$ messages each, by splitting $\mathcal{J}$ in the middle. Starting from $h_a$ an expandable message set $\mathcal{E}$ is created to counter the length padding. In the online phase, hitting technique is used to find a linking string $m$ between $h_b$ and the second pass chain. The index of hitting $i$(say) fixes the prefix of second preimage message. Now the adversary computes $h_c = f_1^+(iv, \overline{m}_0, i\|m\|*)$,

[3] A string is a concatenation of one or more message blocks.

where $*$ is the message of appropriate length drawn from $\mathcal{E}$. Meet-in-the-middle technique is used to connect $h_c$ and $h_m$ using block-wise reverse of messages in $L_1$ and $L_2$, respectively. The path from $iv$ to $h_m$ gives the required second preimage.

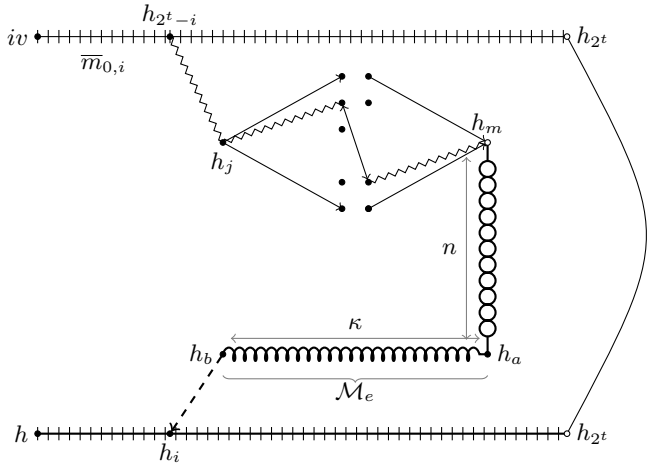


**Fig. 7:** Second preimage attack over Zipper hash with strongly invertible $f_1$ [14].

## 4 Attacks on Zipper Hash

In this section, we present (second) preimage and herding attacks on the zipper hash design. The second preimage attack requires $\tilde{O}(2^n)$ precomputations to construct a structure of $O(n)$ size. This structure is similar to the rho structure in the Pollard rho cycle detection algorithm, so we refer it as the *rho structure*. We start off with the construction of this structure, followed by our attacks on the zipper hash.

### 4.1 Rho Structure

A $2^k$-rho structure is a rho-shaped structure with a unit cycle length, i.e., a self loop and a tail of size $2^k$ denoted by $\mathcal{T}_\rho$. Moreover the labels for the tail as well as the self-loop are the same (which is $m$ in Fig. 8). We denote the *start of rho* by $h^s_\rho$; the *tip of rho* by $h^t_\rho$; and the *rho label* by $m_\rho$.

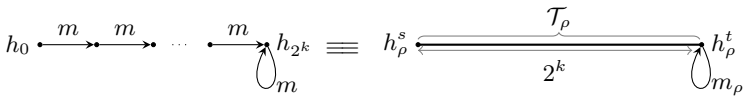


**Fig. 8:** Rho structure and its shorthand representation.

Based on the properties of the underlying compression functions, the rho structure can be constructed in two ways:

**Under Random Oracle Assumption.** When the underlying compression functions $f_1$ and $f_2$ behave as random oracle, the construction follows:

1. In the first phase, a fixed point chaining value $h$ and the corresponding message block $m$ is found for the given compression function.
2. In the second phase, the aim is to construct a $2^k$-blocks chain to the fixed point found above using the block $m$ repeatedly. This is done by choosing a random chaining value, and repeatedly applying $m$. In at most $2^n$ computations, it is expected that the chain will hit $h$.[4]

Both fixed point finding and the rho tail construction takes $\tilde{O}(2^n)$ computations. The construction requires $O(n)$ offline memory for book-keeping.

**Under Multiple Fixed Points Assumption.** A compression function is said to have multiple fixed points weakness, if computing multiple fixed points for a given message string is easy. Some of the insecure PGV schemes [38] have this property. Though we do not have knowledge of any practical scheme exhibiting this property, under this weak assumption the rho structure can be constructed in less than $2^n$ computations. The construction steps are as follows:

1. The fixed point algorithm queries a multiple fixed point oracle internally. It will return a list $\mathcal{L}$ of $2^{n/2}$ fixed points and corresponding message block $m$.
2. The rho construction algorithm applies the $2^k$ chain construction technique on each element of a list $\mathcal{C}_1$ of $2^{n/2}$ randomly selected chaining values. But here the process stops at $2^k$ steps. Suppose the set of endpoints of these chains be $\mathcal{C}_2$. The algorithm then applies list collision between $\mathcal{L}$ and $\mathcal{C}_2$ to find the required tip of rho from $\mathcal{L}$ and the tail of rho from $\mathcal{C}_1$.

The multiple fixed point finding step requires $\tilde{O}(2^{n/2})$ oracle calls and the rho tail construction requires $\tilde{O}(2^{k+\frac{n}{2}})$ computations. So, for $k < \frac{n}{2}$ the construction requires less than $2^n$ computations. The algorithm requires $O(2^{n/2})$ memory for storing $\mathcal{L}$, $\mathcal{C}_1$ and $\mathcal{C}_2$.
The rho structure can also be extended, in which case, a $2^k$ long chain is computed from the tip of rho.[5]

[4] Note that, we leave out the cases in which either $h$ is reached before $2^k$ or there is a cycle in the path. In these cases, one can start with a new node to reach already obtained nodes from which the tip node is reachable.

[5] Note that, this requires either a different message block (other than $m_\rho$) or a different compression function computation (say $f_2$).

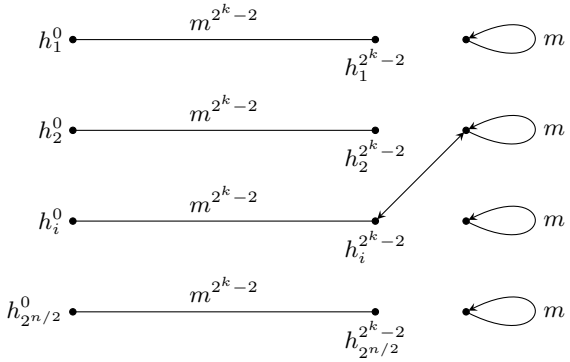


**Fig. 9:** Illustration of rho construction in multiple fixed points case.

### 4.2 Second Preimage Attack

The problem of a long-message second preimage attack on the zipper hash construction is to the best of our knowledge an open problem. Though, Chen and Jin showed a second preimage attack on the zipper hash design with weaker compression functions [14], but the general zipper hash design has seen fewer analysis. The forced-suffix attack by Andreeva *et al.* [3] is the only substantial attack on this design. The difficulty lies in three facts.

1. First, $f_1$ and $f_2$ are independent, i.e., the two pass scheme can use completely independent compression functions for the two passes.
2. Second, the message blocks which are processed last in the first pass are processed first in the second pass.
3. Third, and a bit easier problem is to accommodate the length padding operations.

These three facts compel the adversary to fix the meeting point of first and second pass operations in such a way that it does not violate the message block symmetry and incorporates the padding rule. This makes it hard to apply known attack techniques and structures[28, 27, 4, 2], which generally work for the iterated hash designs and its variants. We present here a second preimage attack that requires less than $2^n$ online computations.

The attack requires a one-time $\tilde{O}(2^n)$ precomputation. This is similar to the TMTO attack technique (as discussed in section 2). The attack can be summarised as follows:

1. In the precomputation phase, a $2^k$ rho structure is computed using $f_1$ computations.
2. In the online phase, the padding block is applied after the tip of rho and the structure is extended using $f_2$ computations.
3. From the endpoint $h_a$ of the extension, a $2^{n-k}$ Joux multicollision $\mathcal{J}$ is constructed using $f_2$ computations.

4. From $h_b$ a $[t, 2^t + t - 1]$ expandable message set $\mathcal{E}$ is computed. This takes care of the length padding.
5. From $h_c$ hitting technique is used to compute a one block connection to the second pass chain. This fixes the message length for $\mathcal{E}$ (set to $i - 2^k - (n - k) - 1$).
6. Suppose the walk from $h_b$ to $h$ is $\omega$ and $h_d = f_1^+(iv, \omega^{rev})$. At last, the messages from $\mathcal{J}$ are used in block-wise reverse fashion to get a link with $\mathcal{T}_\rho$.
7. The walk from $iv$ to $h_\rho^t$ gives the required second preimage message.

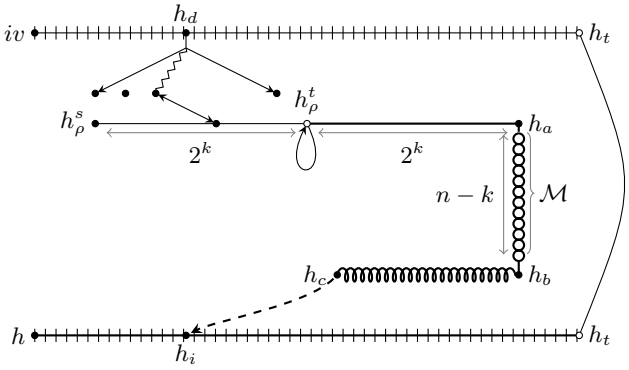


**Fig. 10:** Second preimage attack on Zipper hash.

Clearly, the precomputation phase costs, $\tilde{O}(2^n)$ in a random oracle model and $\tilde{O}(2^{n/2+k})$ in multiple fixed points oracle. Assuming $k \leq \frac{n}{2}$, and a $2^t$ blocks message (where $t = O(k)$), the overall online complexity is $\tilde{O}(2^{n-k})$. For $k = \frac{n}{2}$ this optimises to $\tilde{O}(2^{n/2})$. Note that, in the above algorithm as the message size increases beyond $O(2^{n/2})$, the complexity increases.

### 4.3 Preimage Attacks

The preimage security of zipper hash has not been studied up until now. With an assumption, that the finalisation function is identity, we are presenting two preimage attacks on the zipper hash. The first attack requires a weak invertible $f_2$, and the second attack requires a strong invertible $f_2$.

**Using Weak Invertible $f_2$** The preimage attack is similar in its philosophy to the second preimage attack discussed in 4.2. It uses a precomputed $2^{n/2}$ rho structure where the fixed point is computed on $f_2$. The complete attack can be summarised as follows:

1. In the precomputation phase, a $2^{n/2}$ rho structure is constructed where the tail is constructed using $f_1$ and the tip is computed using $f_2$. From the tip of rho, a $2^{n/2}$ Joux multicollision set $J$ is constructed.

2. In the online phase, meet-in-the-middle is used to find a two block linking message between $h_b$ and the target hash value $h$.[6] The two block string is $\omega$ (say) and $h_c = f_1^+(iv, \omega)$ (say). The last step is to use the messages from $\mathcal{J}$ in block-wise reverse fashion to get a matching with $\mathcal{T}_\rho$. The walk from $iv$ to $h_\rho^t$ gives the required preimage message.

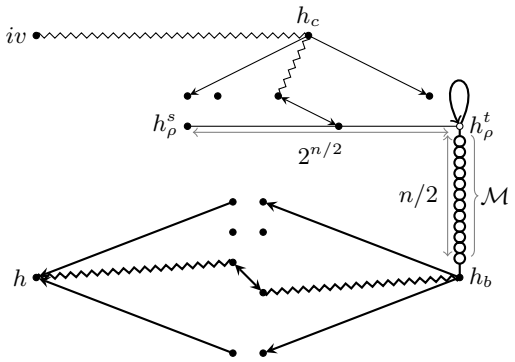


**Fig. 11:** Preimage attack on Zipper hash with weak invertible $f_2$.

All the online computations cost $\tilde{O}(2^{n/2})$ and hence the overall online complexity is $\tilde{O}(2^{n/2})$. The offline phase requires $\tilde{O}(2^n)$. The offline complexity becomes $\tilde{O}(2^{k+n/2})$ for multiple fixed points assumption, in which case, the complexity becomes $\tilde{O}(2^{3n/4})$ and memory complexity reduces to $\tilde{O}(2^{n/4})$ for $k = \frac{n}{4}$.

**Using Strong Invertible $f_2$.** If the underlying $f_2$ function is strong invertible then we can easily construct a preimage attack using Joux multicollision structure and meet-in-the-middle technique. The complete attack can be summarised as follows:

1. In the first phase, starting at $iv$ a $2^n$ Joux multicollision $\mathcal{J}$ is constructed using $f_1$. $\mathcal{J}$ is then divided into two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\frac{n}{2}$ cycles each.
2. In the second phase, messages from $\mathcal{M}_1$ and $\mathcal{M}_2$ are used in block-wise reverse to construct a meet-in-the-middle attack between $h_a$ and the target hash value $h$. The block-wise reverse of the linking message between $h_a$ and $h$ gives the required preimage.

Each step requires $\tilde{O}(2^{n/2})$ computations. So, the overall computational complexity is $\tilde{O}(2^{n/2})$ each.

*Application to Herding Attack on Zipper Hash* If the underlying $f_2$ function of the zipper hash is strongly invertible then, the adversary can use the preimage attack procedure, to herd any prefix or suffix to a target hash value.

[6] Note that, here we are assuming that $f_2$ produces random and distinct preimages at each invocations.

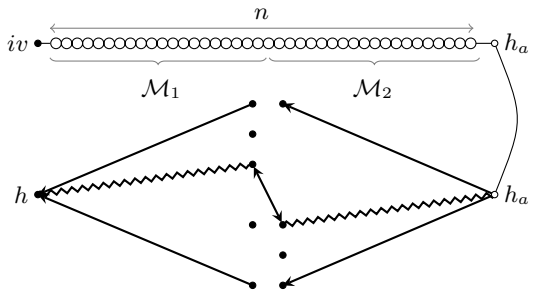


**Fig. 12:** Preimage attack on Zipper hash with strongly invertible $f_2$.

## 5 Attacks on Relaxed Zipper Hash

As noted in the last section, the independence of the underlying compression functions is an important requirement for the security of zipper hash. But, since the construction is two-pass, it might seem fair to use a single compression function in both the passes. We show here, that doing so will have catastrophic effects on the security of the construction. We present here (second) preimage attacks that exploit the $f_1 = f_2$ condition, to bring down the security from $n$ bits to $\frac{n}{2}$ bits. The second preimage attack requires a double-pipe expandable message set that can handle the length variations for two independent hash computation walks. We will begin with a discussion on the construction of this structure, followed by our attacks.

### 5.1 Double-Pipe Expandable Message Set

A double-pipe expandable message set is essentially an expandable message set but with an additional constraint that the expandable message set property should hold for two independent hash computation walks (chains). The Formally,

> For two compression functions $f_1$ and $f_2$ and a message $m$, a double-pipe is the pair of walks $(\omega_1, \omega_2)$ generated by independent hash computations on $f_1$ and $f_2$ starting from arbitrary chaining values $h_1$ and $h_2$ respectively.

With this definition in mind, it is easy to define the double-pipe expandable message set as,

> A $[a, b]$ double-pipe expandable message set is an $[a, b]$ expandable message set for each of the components of the double-pipe setting.

In other words, a double-pipe expandable message set can simultaneously provide us with expandable message set for two independent hash computation chains. This can be useful in constructing attacks on multi-pass and hash combiner schemes.

Before indulging ourselves in the construction of a double-pipe expandable message structure observe that similar notion exists for Joux multicollision. For

example, to get a single collision on double-pipe (see figure 13), we need $2^{n/2}$ multicollisions on the first pipe so that we get $2^{n/2}$ messages to get a single collision on the second-pipe, i.e., a single collision on double-pipe will have $\frac{n}{2}$ number of blocks in each arc.

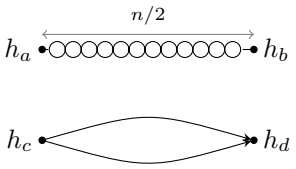


**Fig. 13:** Single collision in double-pipe.

Therefore, to get a single double-pipe collision, we need $O(\frac{n}{2} \cdot 2^{n/2})$ computations. It is easy to observe that finding $2^k$ double-pipe multicollisions require $O(\frac{nk}{2} \cdot 2^{n/2})$ computations ($k$ single double-pipe collisions).
Now, we will extend this idea to a double-pipe expandable message set. Suppose, while constructing a normal $[k, 2^k + k - 1]$ expandable message set $\mathcal{E}$, we inserted a $2^{n/2}$ Joux multicollision after each cycle (illustrated in figure 14) of the expandable message set. This gives us a $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ expandable message set for the first pipe. Now, for the second pipe, we construct the expandable message as follows:

1. For each cycle from $\mathcal{E}$, construct two separate walks $\alpha$ and $\beta$.
2. Use the intermediate $2^{n/2}$ Joux multicollision to collide $\alpha$ and $\beta$.

This will give a $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ double-pipe expandable message set $\mathcal{E}'$. Note that, if the second pipe uses block wise reverse of messages with respect to the first pipe, then the multicollision structure should be inserted before each cycle.

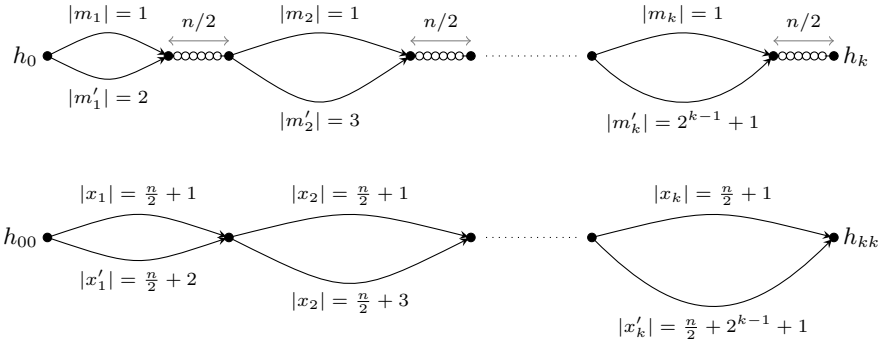


**Fig. 14:** A $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ double-pipe expandable message set.

Clearly it takes $O(2^k + (\frac{nk}{2} + k) \cdot 2^{n/2})$ computations to construct a $[\frac{nk}{2} + k, 2^k + \frac{nk}{2} + k - 1]$ double-pipe expandable message set. Observe that the double-pipe expandable message set can be generalised over multiple pipes. This has been discussed in [24].

### 5.2 Second Preimage Attack

This attack uses a variant of the chain structure [24] in which we only store the even indices. The attack is similar to the previous one with some critical changes in the handling of length padding. The complete attack (illustrated in fig. 15) can be summarised as:

1. First, a $2^{k+1}-1$ simple chain structure is computed using the padding block of $\overline{m}$, storing the even indices in $\mathcal{C}$.
2. From the endpoint $h_b$ of $\mathcal{C}$, a $2^{n-k}$ Joux multicollision $\mathcal{J}$ is constructed.
3. From the endpoint $h_c$ of $\mathcal{J}$ first pipe $\mathcal{DE}_1$ of a $[\frac{nk}{2}+k, 2^k+\frac{nk}{2}+k-1]$ double pipe expandable message $\mathcal{DE}$ is constructed and from its endpoint $h_d$ a $[t, 2^t+t-1]$ expandable message set $\mathcal{E}$ is constructed.
4. From the endpoint $h_e$ of $\mathcal{E}$ hitting technique is used to compute a one block connection to the second pass chain. This fixes the message length from $\mathcal{E}$ (set to $i-2^k-\frac{nk}{2}-n-1$).
5. Suppose the walk from $h$ to $h_e$ is $\omega$ and $h_f = f_1^+(iv, \omega^{rev})$. From $h_f$ the messages in $\mathcal{DE}_1$ are used to construct the second pipe $\mathcal{DE}_2$ of the double pipe expandable message set $\mathcal{DE}$. This will help in compensating for the length uncertainty produced by the chain collision in next step.
6. Now, the messages from $\mathcal{J}$ are used in block-wise reverse fashion to get a matching with $\mathcal{C}$. This fixes the lengths of messages in $\mathcal{DE}_1$ and $\mathcal{DE}_2$ (set to $\frac{nk}{2}+k+\frac{j}{2}$), for matching at index $j$ in $\mathcal{C}$). The midpoint from index $j$ to $2^{k+1}$ is shown as $h_m$.
7. The walk from $iv$ to index $h_m$ in the chain structure gives the required second preimage message.

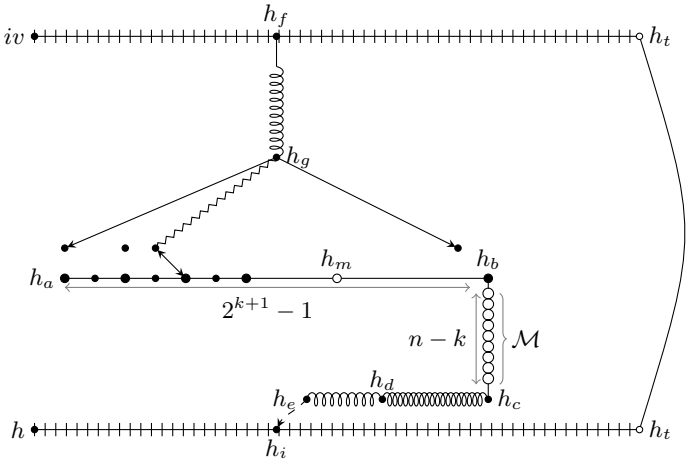


**Fig. 15:** Second preimage attack on Relaxed Zipper Hash.

Note that, the precomputation phase is not necessary for this algorithm. The computational cost can be analysed in four parts,

1. The chain structure takes $\tilde{O}(2^k)$ computations.
2. The double pipe expandable message set takes $O(2^k + (\frac{nk}{2} + k) \cdot 2^{n/2})$ computations.
3. The expandable message set $\mathcal{E}$ and the Joux multicollision cost $O(2^t + t \cdot 2^{n/2} + 2^{n-k})$.
4. The hitting process takes $O(2^{n-k})$ computations.

For $t = O(k)$ and $k = \frac{n}{2}$ the overall complexity becomes $O(2^{n/2})$. The memory complexity is $O(2^k)$. As in 4.2, the complexity increases as the message size increases beyond $O(2^{n/2})$.

### 5.3 Preimage Attack

As in 5.2, the preimage attack on the relaxed zipper hash uses a $2^{\frac{n}{2}+1} - 1$ chain instead of the rho structure and hence does not require any precomputation. The attack (illustrated in fig. 16) is similar to the previous attack on zipper hash. The complexity analysis is also similar to the previous attack.

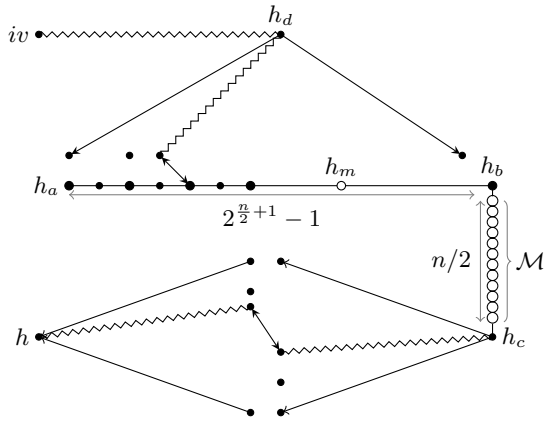


**Fig. 16:** Preimage attack on Relaxed Zipper Hash with weakly invertible $f_2$.

## 6 Attacks on Concatenated Hash

In this section, we present two attacks on concatenated hash. The first attack is a second preimage attack on concatenated hash with a strong invertible component and the second attack is a preimage attack on concatenated hash with one weak and one strong invertible component.

### 6.1 Preimage Attack on Concatenated Hash

Hoch and Shamir [22] proved that the concatenated hash combiner has $\frac{n}{2}$ bits security when both the underlying compression functions are strong invertible. Here we show that the bound is tight even when one of the compression function

is only weak invertible. The attack as illustrated in fig. 17 can be summarized as follows:

1. For the weakly invertible component, we start at $iv_1$ and compute a $2^n$ Joux multicollision set $\mathcal{J}$. Divide $\mathcal{J}$ into two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\frac{n}{2}$ blocks each.
2. From the target hash value compute $2^{n/2}$ random chaining values using the weakly invertible interface, and compute $2^{n/2}$ random chaining values from $h_a$ using forward queries. Use these two sets of $2^{n/2}$ chaining values to compute a 2-block message $x$ between $h_a$ and $h$.
3. Invert the strongly invertible component on $x$ from $g$ and then use $\mathcal{M}_1$ from $iv_2$ and $\mathcal{M}_2$ from $g_a$ to construct a meet-in-the-middle attack. The walk from $iv_2$ to $g$ will give a preimage in both the components, and hence in the concatenated hash.

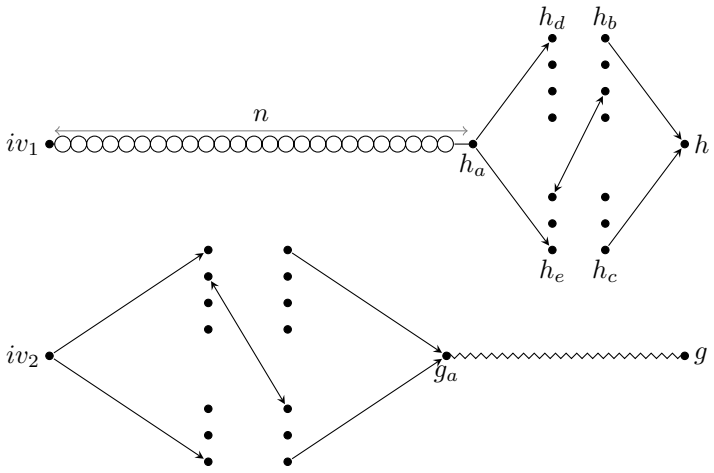


**Fig. 17:** Preimage attack on concatenated hash.

The attack requires construction of a Joux multicollision set and meet-in-the-middle attack on lists of $2^{n/2}$ size. So, the complexity is $\tilde{O}(2^{n/2})$. Note that, this attack is a minor improvement over the attack suggested by Leurent and Wang in [31]. The earlier attack works only when both the components are strong invertible, whereas, our attack works for a relaxed condition. This shows that the bound computed by Hoch and Shamir in [22] is tight when one of the components is only weak invertible.

### 6.2 Second Preimage Attack on Concatenated Hash

Here we assume that both the components are iterated hash and one of the component is strong invertible. Under these assumptions, we can construct a second preimage attack on the concatenated hash as follows:

1. For the non invertible component, start from $iv_1$ and compute a $2^n$ Joux multicollision set $\mathcal{J}$.

2. From the endpoint of $\mathcal{J}$ construct a Kelsey-Schneier long-message second preimage attack [28] on the rest of the chain. Together with the multicollision set, this step gives $2^n$ second preimages for the non invertible component.
3. For the strongly invertible component, divide $\mathcal{J}$ into two sets $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\frac{n}{2}$ blocks each. Invert the strongly invertible component on the second preimage string found in step 1 till $h_i$. Say this inversion ends at $g_a$.
4. Now, we construct a meet-in-the-middle attack between $iv_2$ and $g_a$ using $\mathcal{M}_1$ and $\mathcal{M}_2$ respectively to get the valid second preimage message.

The attack has similar complexity as in the Kelsey-Schneier attack [28], i.e., the complexity becomes $O(2^{n/2})$ in the optimal case and increases as the message size increases beyond $O(2^{n/2})$.

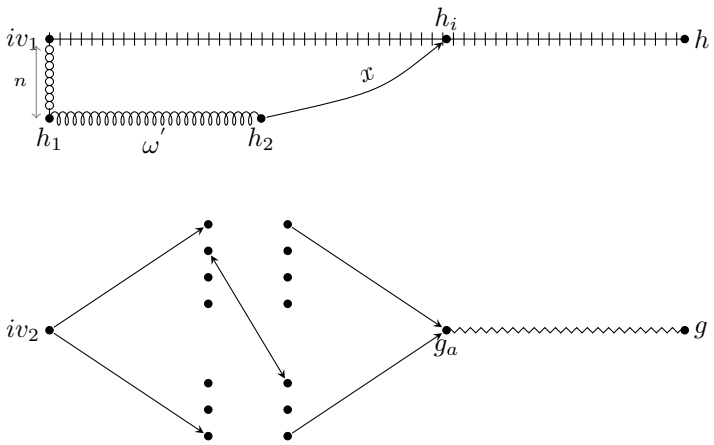


**Fig. 18:** Second preimage attack on concatenated hash.

## 7 Conclusion

In this paper, we have presented the first generic second preimage attack on zipper hash function and its relaxed variant ($f_1 = f_2$). We have also investigated the different attacks possible on zipper hash due to the weaknesses in the compression functions. Though the zipper hash is of lesser practical significance (due to its two pass nature), it is our belief that the results in this paper are important, as they show that even after an additional pass, this scheme has some security flaws. Particularly, our result on the relaxed variant of zipper hash is quite strong and surprising as it requires less than $O(2^n)$ offline as well as online computations. Our attacks on the general zipper hash design falls under the TMTO attack category [21], where the adversary is allowed a one time large precomputation of $O(2^n)$. The second preimage attack on the general zipper hash follows the time memory trade-off relation, $TM = 2^n$. In tables 1 and 2, we have summarised the attack complexity results for the zipper hash function and its relaxed variant.

| Attacks | Offline Complexity | Online Complexity | Remarks |
|---|---|---|---|
| Second Preimage Attack (see section 4) | $\tilde{O}(2^n)$ | $\tilde{O}(2^{n-k} + 2^k + 2^{n/2})$ | The online complexity optimises to $O(2^{n/2})$ for $k = \frac{n}{2}$. Note that the offline cost becomes $\tilde{O}(2^{k+n/2})$ for multiple fixed points oracle, i.e., the complexity reduces to $\tilde{O}(2^{3n/4})$ for $k = \frac{n}{4}$. |
| Preimage Attack (see section 4) | $\tilde{O}(2^n)$ | $\tilde{O}(2^{n/2})$ | This attack works for weak invertible $f_2$. The complexity becomes $\tilde{O}(2^{3n/4})$ for multiple fixed points oracle. |
| Preimage Attack (see section 4) | $\tilde{O}(2^{n/2})$ | $\tilde{O}(2^{n/2})$ | This attack works for strong invertible $f_2$. |
| Herding Attack (see section 4) | $\tilde{O}(2^{n/2})$ | $\tilde{O}(2^{n/2})$ | This attack works for strong invertible $f_2$, and utilises the preimage attack technique. |

**Table 1:** Complexity results for attacks on zipper hash.

| Attacks | Offline Complexity | Online Complexity | Remarks |
|---|---|---|---|
| Second Preimage Attack (see section 5) | | $\tilde{O}(2^{n/2} + 2^k + 2^{n-k})$ | The complexity optimises to $\tilde{O}(2^{n/2})$ for $k = \frac{n}{2}$. |
| Preimage Attack (see section 5) | $\tilde{O}(2^{n/2})$ | $\tilde{O}(2^{n/2})$ | This attack works for weak invertible $f_2$. |

**Table 2:** Complexity results for attacks on relaxed zipper hash.

We also give two attacks on the concatenated hash that show the tightness of bounds computed by Hoch and Shamir [22] for concatenated hash with weak compression functions. In table 3 we sumarise the attack complexity results for the concatenated hash function.

| Attacks | Offline Complexity | Online Complexity | Remarks |
|---|---|---|---|
| Second Preimage Attack (see section 6) | $\tilde{O}(2^{n/2} + 2^k)$ | $\tilde{O}(2^{n-k})$ | This attack works for one strongly invertible. The offline and online complexity optimises to $\tilde{O}(2^{n/2})$ for $k = \frac{n}{2}$. |
| Preimage Attack (see section 6) | | $\tilde{O}(2^{n/2})$ | This attack works for one weakly invertible and one strongly invertible component. |

**Table 3:** Complexity results for attacks on concatenated hash.

## References

1. Riham AlTawy and Amr M. Youssef. Watch your Constants: Malicious Streebog. *IACR Cryptology ePrint Archive*, 2014.
2. Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, Pierre-Alain Fouque, Jonathan Hoch, John Kelsey, Adi Shamir, and Sbastien Zimmer. New Second-Preimage Attacks on Hash Functions. *Journal of Cryptology*, 2010.
3. Elena Andreeva, Charles Bouillaguet, Orr Dunkelman, and John Kelsey. Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård. In *Selected Areas in Cryptography, 16th Annual International Workshop, SAC '09 Revised Selected Papers*, 2009.

4. Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second Preimage Attacks on Dithered Hash Functions. In *Advances in Cryptology - EUROCRYPT '08 Proceedings*, 2008.
5. Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology - CRYPTO '96 Proceedings*, 1996.
6. Mihir Bellare and Phillip Rogaway. Optimal Asymmetric Encryption. In *Advances in Cryptology - EUROCRYPT '94 Proceedings*, 1994.
7. Mihir Bellare and Phillip Rogaway. The Exact Security of Digital Signatures - How to Sign with RSA. In *Advances in Cryptology - EUROCRYPT '96 Proceeding*, 1996.
8. Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet, and William Jalby. Collisions of SHA-0 and Reduced SHA-1. In *Advances in Cryptology - EUROCRYPT '05 Proceedings*, 2005.
9. Eli Biham and Orr Dunkelman. A Framework for Iterative Hash Functions - HAIFA. *IACR Cryptology ePrint Archive*, 2007.
10. Alex Biryukov, Sourav Mukhopadhyay, and Palash Sarkar. Improved Time-Memory Trade-Offs with Multiple Data. In *Selected Areas in Cryptography, 12th International Workshop, SAC '05 Proceedings*, 2005.
11. Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *Advances in Cryptology - ASIACRYPT '00 Proceedings*, 2000.
12. John Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block Cipher Based Hash Function Constructions from PGV. In *Advances in Cryptology - CRYPTO '02 Proceedings*. 2002.
13. R. Canettia, R. L. Rivest, M. Sudan, L. Trevisan, S. P. Vadhan, and H. Wee. Amplifying Collision Resistance: A Complexity-Theoretic Treatment. In *Advances in Cryptology - CRYPTO '07 Proceedings*, 2007.
14. Shiwei Chen and Chenhui Jin. A Second Preimage Attack on Zipper Hash. *Security and Communication Networks*, 2015.
15. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In *Advances in Cryptology - CRYPTO '05 Proceedings*, 2005.
16. Ivan B. Damgård. A Design Principle for Hash Functions. In *Advances in Cryptology - CRYPTO '89 Proceedings*, 1989.
17. R.D. Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, 1999.
18. Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Cambridge University Press, 2000.
19. Jovan Dj. Golic. Cryptanalysis of Alleged A5 Stream Cipher. In *Advances in Cryptology - EUROCRYPT '97 Proceeding*, 1997.
20. Jian Guo, Jérémy Jean, Gaëtan Leurent, Thomas Peyrin, and Lei Wang. The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function. In *Selected Areas in Cryptography, 21st International Workshop, SAC '14 Proceedings*, 2014.
21. Martin E. Hellman. A Cryptanalytic Time-Memory Trade-Off. *IEEE Transactions on Information Theory*, 1980.
22. Jonathan J. Hoch and Adi Shamir. On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In *Automata, Languages and Programming, ICALP '08 Proceedings*, 2008.
23. Jin Hong and Palash Sarkar. Rediscovery of Time Memory Tradeoffs. *IACR Cryptology ePrint Archive*, 2005.

24. Ashwin Jha. Cryptanalysis of Iterated Hash and Its Variants. Master's thesis, Indian Statistical Institute, Kolkata, 2015.
25. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In *Advances in Cryptology - CRYPTO '04 Proceedings*, 2004.
26. Antoine Joux and Thomas Peyrin. Hash Functions and the (Amplified) Boomerang Attack. In *Advances in Cryptology - CRYPTO '07 Proceedings*, 2007.
27. John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In *Advances in Cryptology - EUROCRYPT '06 Proceedings*, 2006.
28. John Kelsey and Bruce Schneier. Second Preimages on n-bit Hash Functions for Much Less than $2^n$ Work. *IACR Cryptology ePrint Archive*, 2004.
29. Vlastimil Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. *IACR Cryptology ePrint Archive*, 2006.
30. Tuomas Kortelainen and Juha Kortelainen. On diamond structures and trojan message attacks. In *Advances in Cryptology - ASIACRYPT '13 Proceedings*, 2013.
31. Gaëtan Leurent and Lei Wang. The Sum Can Be Weaker Than Each Part. *IACR Cryptology ePrint Archive*, 2015.
32. Moses Liskov. Constructing an Ideal Hash function from Weak Ideal Compression Functions. In *Selected Areas in Cryptography, 13th International Workshop, SAC '06 Proceedings*, 2006.
33. Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. *IACR Cryptology ePrint Archive*, 2003.
34. Ralph C. Merkle. One way Hash Functions and DES. In *Advances in Cryptology - CRYPTO '89 Proceedings*. 1989.
35. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
36. Mridul Nandi and Douglas R. Stinson. Multicollision Attacks on Some Generalized Sequential Hash Functions. *IEEE Transactions on Information Theory*, 2007.
37. Philippe Oechslin. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *Advances in Cryptology - CRYPTO '03 Proceedings*, 2003.
38. Bart Preneel, René Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In *Advances in Cryptology - CRYPTO '93 Proceedings*, 1993.
39. Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Usenix Security*, 2005.
40. Jalaj Upadhyay. Generic Attacks on Hash Functions. Master's thesis, University of Waterloo, 2010.
41. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In *Advances in Cryptology - EUROCRYPT '05 Proceedings*, 2005.
42. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In *Advances in Cryptology - CRYPTO '05 Proceedings*, 2005.
43. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In *Advances in Cryptology - EUROCRYPT '05 Proceedings*, 2005.
44. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient Collision Search Attacks on SHA-0. In *Advances in Cryptology - CRYPTO '05 Proceedings*, 2005.