The Sum Can Be Weaker Than Each Part*

Gaëtan Leurent[1] and Lei Wang[2]

[1] Inria, France, gaetan.leurent@inria.fr
[2] Nanyang Technological University, Singapore, wang.lei@ntu.edu.sg

Abstract. In this paper we study the security of summing the outputs of two *independent* hash functions, in an effort to increase the security of the resulting design, or to hedge against the failure of one of the hash functions. The exclusive-or (XOR) combiner $H_1(M) \oplus H_2(M)$ is one of the two most classical combiners, together with the concatenation combiner $H_1(M) \parallel H_2(M)$. While the security of the concatenation of two hash functions is well understood since Joux's seminal work on multicollisions, the security of the sum of two hash functions has been much less studied. The XOR combiner is well known as a good PRF and MAC combiner, and is used in practice in TLS versions 1.0 and 1.1. In a hash function setting, Hoch and Shamir have shown that if the compression functions are modeled as random oracles, or even *weak* random oracles (*i.e.* they can easily be inverted – in particular H_1 and H_2 offer no security), $H_1 \oplus H_2$ is indifferentiable from a random oracle up to the birthday bound.

In this work, we focus on the preimage resistance of the sum of two narrow-pipe n-bit hash functions, following the Merkle-Damgård or HAIFA structure (the internal state size and the output size are both n bits). We show a rather surprising result: the sum of two such hash functions, e.g. SHA-512 $\oplus$ Whirlpool, can never provide n-bit security for preimage resistance. More precisely, we present a generic preimage attack with a complexity of $\tilde{O}(2^{5n/6})$. While it is already known that the XOR combiner is not preserving for preimage resistance (*i.e.* there might be *some* instantiations where the hash functions are secure but the sum is not), our result is much stronger: for *any* narrow-pipe functions, the sum is not preimage resistant.

Besides, we also provide concrete preimage attacks on the XOR combiner (and the concatenation combiner) when one or both of the compression functions are weak; this complements Hoch and Shamir's proof by showing its tightness for preimage resistance.

Of independent interests, one of our main technical contributions is a novel structure to control *simultaneously* the behavior of independent hash computations which share the same input message. We hope that breaking the pairwise relationship between their internal states will have applications in related settings.

Keywords: Hash functions, combiners, XOR combiner, preimage attack.

1 Introduction

Hash functions are a very important class of primitive in modern cryptography, used in almost every secure system. A hash function $H : \{0,1\}^* \mapsto \{0,1\}^n$ takes an arbitrary length input and produces an n-bit output or digest. Hash functions are used in many setting with various security requirements; the general expectation is that a hash function should behave like a random function from $\{0,1\}^*$ to $\{0,1\}^n$. More concretely, the main security notions expected from a hash function are:

Collision resistance. It should be hard to find two messages $M \neq M'$ with $H(M) = H(M')$.

Second-preimage resistance. Given a message M, it should be hard to find $M' \neq M$ with $H(M) = H(M')$.

Preimage resistance. Given a target hash value $\overline{H}$, it should be hard to find M with $H(M) = \overline{H}$.

Since generic collision attacks require $2^{n/2}$ work, and generic preimage attacks require 2^n work, a secure hash function should have the same level of resistance.

In order to build more secure hash functions, or to protect oneself against future cryptanalysis advances, such as the devastating attacks of Wang *et al.* against the SHA family [42,41], a practical countermeasure might be to combine two different hash functions. The goal is that the combined hash function can only be broken when both components are weak. In particular, this reasoning was used by the designers of SSL [14] and TLS [8], who combined MD5 and SHA-1 in various ways. More precisely, the Key Derivation Function of TLS v1.0/v1.1 uses a sum of HMAC-MD5 and HMAC-SHA-1.[3] The designers explain [8]: "In order to make the PRF as secure as possible, it uses two hash algorithms in a way which should guarantee its security if either algorithm remains secure."

There are two classical hash function combiners: the concatenation combiner $H_1(M) \,\|\, H_2(M)$ and the XOR combiner $H_1(M) \oplus H_2(M)$. In a seminal work [20], Joux showed that the concatenation combiner with narrow-pipe hash functions offers much less security than could be expected: it has $2n$ bits of output, but essentially offers the same security as an n-bit hash function. In this work, we carry a similar analysis for the XOR combiner. Previous work has shown that it is indifferentiable from a random oracle up to the birthday bound [17], even if the initial functions are weak; in particular, it has optimal collision resistance of $n/2$ bits. However, we show that the preimage security is much less than one might expect, with a generic preimage attack with complexity $\tilde{O}(2^{5n/6})$.

Since the goal of a combiner is to keep some security even if one of the functions is found to be weak, it is natural that the two hash functions H_1 and H_2 are independent in practice. Throughout this paper the two hash functions

[3] We note that this MD5/SHA-1 combiner has been replaced by primitives based on single hash function (e.g., SHA-256) since TLS v1.2 [9].

used in a combiner are always assumed to be independent without specifying it explicitly[4].

Iterated hash function. In this paper we consider iterated hash functions, following the Merkle-Damgård construction [30,7] or the more general HAIFA construction [3], as shown in Figure 1. We focus on narrow-pipe designs, *i.e.* we assume that the internal state size is the same as the output size n. The message M is first split into blocks $m_0, \dots m_\ell$, and the hash function iterates a series of compression functions h_i over an internal state x, with the initial value denoted as IV. Finally, the hash value is computed with a finalization function g:

$$x_0 = IV \qquad x_{i+1} = h_i(x_i, m_i) \qquad H(M) = g(x_{\ell+1}, |M|)$$

In the following, we assume that the compression function is the same at every step ($\forall i, h_i = h$) in order to simplify the notations, but it is straightforward to apply the attack with the corresponding function at each step.

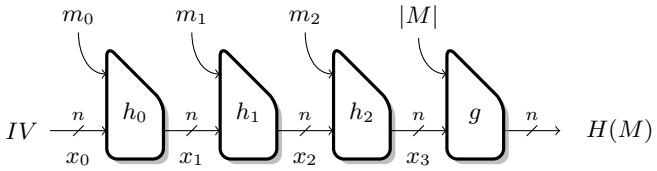

Fig. 1. Iterated hash function.

1.1 Related works

Combiners have been studied in several different settings. For generic attacks, the compression functions are modeled as random functions, in order to devise attacks that don't use any weakness of the compression functions. On the other hand, some work assumes that the compression function is a *weak* random oracle (that can easily be inverted), and prove that some constructions are still secure in this model. Finally, more theoretical work focus on the notion of *robustness*, *i.e.* the existence of a reduction from attacks on the hash functions to attacks on the combiner.

[4] If the two hash functions can be related, it is trivial that the XOR combiner is not security-preserving. For instance, let $H_2(M) := H_1(M) \oplus \texttt{const}$, where `const` is a constant. If H_1 is ideally secure, then H_2 is also ideally secure. However, the XOR combiner $H_1(M) \oplus H_2(M) = \texttt{const}$ for any message M.

Analysis of the concatenation combiner. The concatenation combiner $H_1(M) \| H_2(M)$ is probably the most studied one. In 2004, Joux [20] described surprising attacks on the concatenation of two narrow-pipe hash functions using multicollisions: while the output size is $2n$ bits, the concatenation can at most provide $n/2$-bit security for collision resistance and n-bit security for preimage resistance[5]. In particular, the concatenation is not security-amplifying. On the other hand, the concatenation combiner is robust for preimages and collisions, which gives a matching lower bound for generic attacks.

Later, Hoch and Shamir [17] evaluated the security of the concatenation combiner with two weak hash functions. More precisely, the two hash functions are narrow-pipe Merkle-Damgård, and the compression functions are modeled as weak random oracles (as defined by Liskov [24]), *i.e.*, the adversary is given additional interfaces to receive (random) preimages of the compression functions. They have proven that in this model, the concatenation combiner is still indifferentiable from a random oracle with $n/2$-bit security, implying (at least) the same security bound for collision resistance and preimage resistance. The bound is matched by Joux's attack for collisions, but there is a gap with Joux's attack for preimages, with complexity 2^n, which might be interesting to investigate further.

Mendel *et al.* analyzed some dedicated instantiations of the concatenation combiner [28], in particular using the hash function MD5. We omit the details and refer interested readers to [28].

Analysis of the XOR combiner. The XOR combiner has received less analysis. The work of Hoch and Shamir [17] actually proves the security of the XOR combiner as an intermediate result: it is also indifferentiable from a random oracle up to $2^{n/2}$ queries in the weak random oracle model. In particular, this proves that there are no generic attacks with complexity smaller than $2^{n/2}$. For collision resistance, the bound is tight, since it is matched with the generic birthday attack bound. On the other hand, for preimage resistance, there exists a gap between the $n/2$-bit proven bound and the n-bit expected ideal security bound.

To the best of our knowledge, no preimage attacks have been shown against the XOR combiner. Therefore, the preimage security of the XOR combiner against generic attacks is still an open problem, and will be the main topic of our work. We will also consider the preimage security of the XOR combiner with weak hash functions, and study the tightness of Hoch and Shamir's bound.

Robust combiners. In the last years, the general problem of combining two (or more) hash functions H_1 and H_2 has been extensively studied from a theoretical point of view. These works focus on the notion of a *robust* combiner: a robust combiner is secure with respect to property α as long as one of the underlying hash functions is secure for α. It can be shown that the concatenation combiner is a robust combiner for collision-resistant hash functions and for MACs, while the

[5] The attacks actually require only one of the functions to be narrow-pipe.

XOR combiner is robust for PRFs and for MACs [23]. More advanced combiners have been constructed in order to be robust for multiple properties simultaneously [11,12,13]. The notion was mostly studied via the black-box reduction model. A series of results have showed that robust combiners for collision resistance and preimage resistance cannot have an output length significantly shorter than the sum of the output length of the underlying hash functions [4,35,36,31]. Since the XOR combiner is length preserving, this shows that it is not robust for collision resistance and preimage resistance.

Actually, the impossibility results are in part due to the stringent requirement from the black-box reduction model. In order to overcome this limitation, Mittelbach introduced the *idealized* random oracle model [32]. He gives a construction of a short output combiner with optimal collision and preimage security in this model[6] (assuming that one of the functions is ideal): Cryptophia's short combiner uses the sum of two hash functions with some pre-processing of the messages (to allow non-independent functions).

More generally, we point out that a combiner being non-robust does not necessarily mean there is an attack. The non-robustness results only show that the security of the combiner cannot be proved with a *reduction* from the security of the hash functions. In particular, the XOR combiner is not robust for collision-resistance, or even collision-resistance preserving. However, Hoch and Shamir's work proves that there are no generic collision attacks on this construction, either with ideal compression function, or even with weak compression functions. This arguably makes the XOR a useful combiner for collision resistance. Regarding preimage security, the non-robustness result does not imply that the XOR of two concrete hash functions is weak, and the simplicity and short output of this construction still make it quite attractive.

1.2 Our results

In this work, we study the preimage security of the XOR combiner, the main remaining open problem for classical combiners. We show that, surprisingly, the sum of two narrow-pipe n-bit hash functions can never achieve n-bit security for preimage resistance. More precisely, we find a generic preimage attack with a complexity of $\tilde{O}(2^{5n/6})$. It does not exploit any structural weakness of the compression functions and hence is applicable even if the compression functions are two ideal random oracles. Thus, even if the two hash functions are n-bit secure for preimage resistance, the XOR combiner is at most $5n/6$-bit secure for primage resistance. In other words, *the sum can be weaker than each part.*

The attack is based on a novel technique to break the pairwise relationship between the internal states of the two hash functions. More precisely, the two hash functions H_1 and H_2 share the same input message, and hence the internal states of their iterative compression function computations are related. We control the computation chains of H_1 and H_2 simultaneously by constructing a new message

[6] A mistake in the initial proof and construction was later fixed by Mennink and Preneel [29].

structure $\mathcal{M}$, and two sets of internal states $\mathcal{A}$ for H_1 and $\mathcal{B}$ for H_2 such that: for any value A from $\mathcal{A}$ and any value B from $\mathcal{B}$, we can derive a message $M_{A,B}$ from $\mathcal{M}$ such that $H_1(M_{A,B})$ produces A and $H_2(M_{A,B})$ produces B. Hence we can select states from $\mathcal{A}$ and $\mathcal{B}$ independently. After that, we use a birthday match to find a message block m, a value A from $\mathcal{A}$ and a value B from $\mathcal{B}$ such that $h_1(A, m) \oplus h_2(B, m)$ is equal to the target hash digest, where h_1 and h_2 are the compression functions of H_1 and H_2 respectively. Finally we derive the message $M_{A,B}$ from $\mathcal{M}$, and output $M_{A,B} \parallel m$ as a preimage of the target hash digest.

Our preimage attack is also applicable to Cryptophia's short combiner [32,29]. This construction has been proven to provide optimal collision and preimage resistance, assuming that at least one of the initial functions is a monolithic random oracle, but our attack does not violate the security proof, because we use the fact that both functions have an iterative structures with an n-bit internal state. Still, this shows that with many practical hash functions, the combiner will be weaker that the initial functions. Our results also show that the XOR combiner and Cryptophia's combiner are not robust in the semi-black-box model introduced by Mittelbach [32][7], even with independant hash functions H_1 and H_2.

Our analysis on the XOR combiner is also interesting for dedicated hash function design. The hash function family RIPEMD [10] is based on a compression function with two parallel lanes, added together at the end of each compression function. Interestingly, RIPEMD-160 has been quite resilient to cryptanalysis [27,38,25,26], and are still considered secure. Several more recent designs use parallel lanes in a similar way (combining them at the end of each compression function call), such as HAS-V [34], FORK [18] and LANE [19]. It might be tempting to use parallel lanes during the full iteration, and to combine them only at the end. Indeed, the designers of SHA-V [15] used this approach: the 160-bit version of SHA-V has two parallel lanes, combined at the end with a modular sum. This is equivalent to summing two different hash functions, and hence our attack can be applied to SHA-V.

Another contribution of this paper is to present concrete preimage attacks on the XOR combiner with one or both weak hash functions (defined in [24]). The complexity of our attacks is $\tilde{O}(2^{n/2})$. Furthermore, the attack can be extended to the concatenation combiner with two weak hash functions under the same complexity. It can be seen that these attacks match the bound of Hoch and Shamir's security proof [17], and hence fulfill the gaps pointed out in Section 1.1. It implies the tightness of Hoch and Shamir's proof on the classical combiners with weak hash functions for preimage resistance.

Finally, we would like to highlight the technical interests of this paper. We devise a novel structure named interchange structure to simultaneously control

[7] Loosely speaking, a combiner is robust with respect to property α if it is (at least) as secure as the stronger underlying hash function for α.

two (or more) hash lanes with the same input message, and succeed in further relaxing the pairwise relation between the internal states of lanes. It is indeed a step of technical advance compared with previous extensive studies on this topic, and hence will hopefully have applications or lead to new technical development in related settings. We refer to the open discussions in Section 7 for more details.

1.3 Notations and roadmap in the rest of paper

We use the following notations:

H_1, H_2	: hash functions
IV_1, IV_2	: initial values for H_1 and H_2, respectively
h_1, h_2	: compression functions of H_1 and H_2, respectively
h_1^*, h_2^*	: compression functions iterated over several blocks (in particular, $H_i(\boldsymbol{M}) = h_i^*(IV_i, \boldsymbol{M})$)
m	: message block
M	: message chunk ($n/2$ blocks)
$\boldsymbol{M}$	: long message (several chunks)
a_j, b_k	: chains for H_1 and H_2, respectively; a_j denotes a generic chain, while a_{j_0} denotes a particular chain
A_j, B_k	: end points of the chains
n	: hash function output size

Roadmap. Section 2 provides an overview of our generic preimage attack on the XOR combiner. Sections 3, 4, and 5 elaborate the attack procedure step by step in details. Section 6 presents the applications and extensions of the attack. Finally we conclude the paper and discuss future directions in Section 7.

2 Overview of the attack

We first give an overview of the techniques and the structures used in the attack, while more detailed descriptions will be given in the following sections.

The main idea is to consider several chains of internal states reached by processing a common message $\boldsymbol{M}$ from different starting points (note that the message $\boldsymbol{M}$ is not fixed in advance, but will be determined when building the structure). More precisely, the message $\boldsymbol{M}$ is denoted as the *primary* message, and divided in several chunks: $\boldsymbol{M} = M_0 \,\|\, M_1 \,\|\, \ldots$ (as discussed later, a chunk will consist of several message blocks). We denote chains of internal states for H_1 as a_j, and the individual states of the chain as a_j^i, with $h_1^*(a_j^i, M_i) = a_j^{i+1}$. Similarly, we denote chains for H_2 as b_k, with $h_2^*(b_k^i, M_i) = b_k^{i+1}$. When considering both hash functions, message block M_i leads from the pair of states (a_j^i, b_k^i) to (a_j^{i+1}, b_k^{i+1}), which is denoted:

$$(a_j^i, b_k^i) \overset{M_i}{\rightsquigarrow} (a_j^{i+1}, b_k^{i+1}).$$

Switch structure. Next we build special structures called *switches* in order to jump between chains in a controlled way. A switch allows to jump from a specific pair of chains (a_{j_0}, b_{k_0}) to a different pair of chains (a_{j_0}, b_{k_1}) using a secondary message chunk M'_i, in addition to the normal transitions using chunk M_i of the primary message $\boldsymbol{M}$:

$$(a^i_j, b^i_k) \overset{M_i}{\rightsquigarrow} (a^{i+1}_j, b^{i+1}_k): \qquad \text{normal transition for each chain}$$
$$(a^i_{j_0}, b^i_{k_0}) \overset{M'_i}{\rightsquigarrow} (a^{i+1}_{j_0}, b^{i+1}_{k_1}): \qquad \text{jump from chains } (a_{j_0}, b_{k_0}) \text{ to } (a_{j_0}, b_{k_1})$$

In order to simplify the notations, we often omit the chunk index, in order to show only the chains that are affected by the switch.

The main message chunk M_i and the secondary message chunk M'_i are determined when building the switch, and the main message defines the next state of all the chains. We note that the secondary message chunk M'_i should only be used when the state is $(a^i_{j_0}, b^i_{k_0})$. A simple example is depicted in Figure 2.

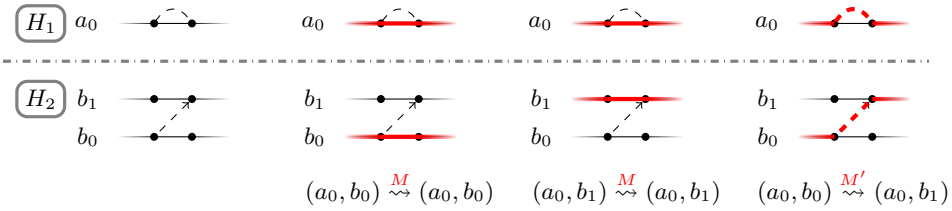

Fig. 2. A single switch: jump from (a_0, b_0) to (a_0, b_1) by using M' (dashed lines) instead of M (solid lines).

Alternatively, a switch can be designed to jump from (a_{j_0}, b_{k_0}) to (a_{j_1}, b_{k_0}). We defer the details of the construction to Section 3; it can be built with a complexity of $\tilde{O}(2^{n/2})$.

Interchange structure. By combining several simple switches, we can build an interchange structure with starting points IV_1 and IV_2 and ending points $\{A_j, j = 0 \dots 2^t - 1\}$ and $\{B_k, k = 0 \dots 2^t - 1\}$, so that we can select a message ending in any state (A_j, B_k). Figure 3 shows one possible way to build such a structure, and Figure 4 shows how to select a given message in the structure. An interchange structure with 2^t chains for each function requires about 2^{2t} switches. Since we can build a switch for a cost of $\tilde{O}(2^{n/2})$, the total structure is built with $\tilde{O}(2^{2t+n/2})$ operations.

Preimage search. Finally, we can use an interchange structure with ending points $\{A_j, j = 0 \dots 2^t - 1\}$ and $\{B_k, k = 0 \dots 2^t - 1\}$, to build a preimage

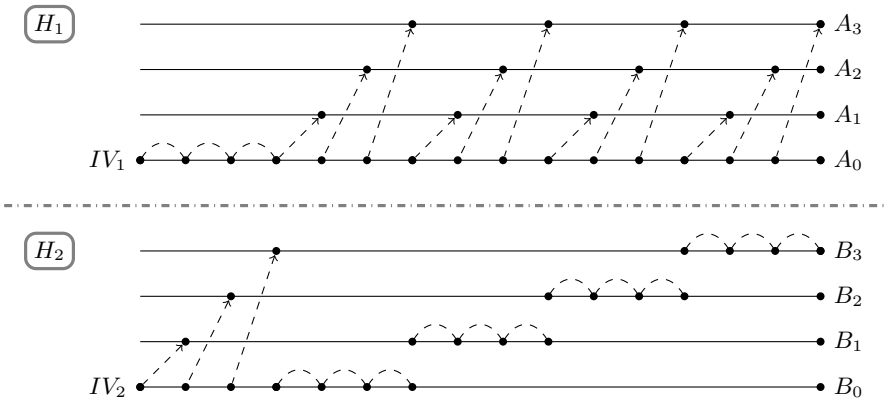

Fig. 3. Overview of an interchange structure

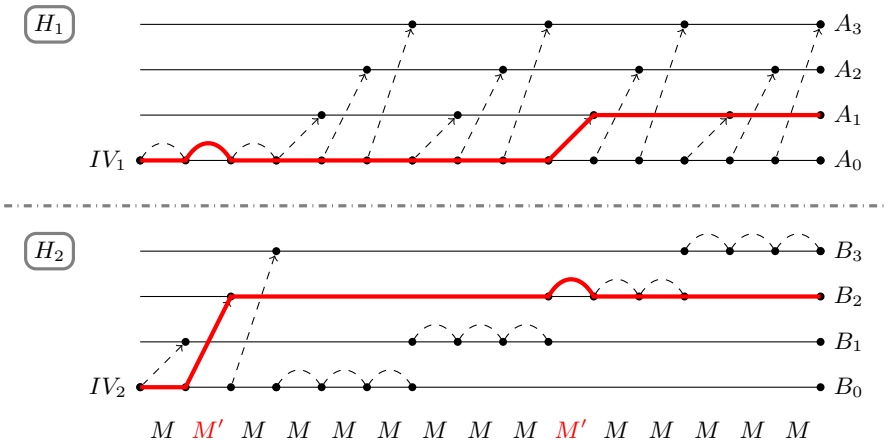

Fig. 4. Using of the interchange structure to reach output (A_1, B_2)

attack as follows. Let $\overline{H}$ denote the target value. We select a random message block m, and we compute two lists by evaluating the compression functions after the interchange structure: $\{A'_j = h_1(A_j, m), j = 0 \dots 2^t - 1\}$ and $\{B'_k = \overline{H} \oplus h_2(B_k, m), k = 0 \dots 2^t - 1\}$. We expect a match between the lists with probability 2^{2t-n}. After about 2^{n-2t} random choices of m, we get a match (j^*, k^*):

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

Therefore, we can construct a preimage of $\overline{H}$ by concatenating the message leading to (A_{j^*}, B_{k^*}) in the interchange structure, and the block m (we ignore the finalization function in this section).

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

Complexity analysis. Building the interchange structures requires about $2^{2t+n/2}$ evaluations of the compression function, while the preimage search requires about 2^{n-t}. The optimal complexity is reached when both steps take the same time, *i.e.* $t = n/6$. This gives a complexity of $\tilde{O}(2^{5n/6})$.

3 The switch structure

We now explain how to build the switch structure at the core of our attack. This construction is strongly based on the multicollision technique of Joux [20].

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

$$(a^i_{j_0}, b^i_{k_0}) \overset{M_i}{\rightsquigarrow} (a^{i+1}_{j_0}, b^{i+1}_{k_0}) \quad (a^i_{j_0}, b^i_{k_1}) \overset{M_i}{\rightsquigarrow} (a^{i+1}_{j_0}, b^{i+1}_{k_1}) \quad (a^i_{j_0}, b^i_{k_0}) \overset{M'_i}{\rightsquigarrow} (a^{i+1}_{j_0}, b^{i+1}_{k_1}).$$

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

$$\begin{aligned} a^{i+1}_{j_0} &= h^*_1(a^i_{j_0}, M_i) = h^*_1(a^i_{j_0}, M'_i) \\ b^{i+1}_{k_1} &= h^*_2(b^i_{k_1}, M_i) = h^*_2(b^i_{k_0}, M'_i) \\ b^{i+1}_{k_0} &= h^*_2(b^i_{k_0}, M_i) \neq b^{i+1}_{k_1} \end{aligned}$$

We first build a multicollision for h^*_1, starting from state $a^i_{j_0}$, *i.e.* a large set $\mathcal{M}$ of $2^{n/2}$ messages that all reach the same state $a^{i+1}_{j_0}$ ($\forall M \in \mathcal{M}$, $h^*_1(a^i_{j_0}, M) = a^{i+1}_{j_0}$). As shown by Joux, this can be done efficiently by sequentially building $n/2$ collisions.

Next, we evaluate $h^*_2(b^i_{k_0}, M)$ and $h^*_2(b^i_{k_1}, M)$ for all the messages M in the set $\mathcal{M}$. With high probability there is match between the sets of values[8]. We denote the colliding messages as M_i and M'_i, so that we have $h^*_2(b^i_{k_0}, M'_i) = h^*_2(b^i_{k_1}, M_i)$.

Finally we compute the missing chains using the message m_i: $a^{i+1}_j = h^*_1(a^i_j, m_i)$, $b^{i+1}_k = h^*_2(b^i_k, m_i)$. With high probability all the chains reach distinct values; if this is not the case, we restart the construction with a new multicollision. The full algorithm is shown as Algorithm 1, and illustrated by Figure 5; it requires about $n/2 \cdot 2^{n/2}$ evaluations of the compression functions.

[8] If this is not the case, we build a new multicollision.

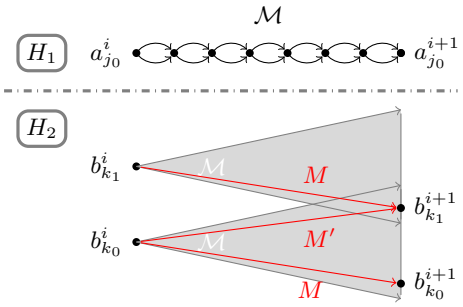

Fig. 5. Building a switch structure. First, M and M' are selected from $\mathcal{M}$ to generate a collision (defining the new b_{k_1}), then b_{k_0} is evaluated using M.

4 The interchange structure

Let us now describe the combination of switch structures into an interchange structure. The goal of this structure is to select the final value of the H_1 computation and the H_2 computation independently. More precisely, the structure defines two sets of final values A_j and B_k, and a set of messages $\boldsymbol{M}_{jk}$ such that:

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

In order to build this structure, we initialize the first chains with $a^0_0 = IV_1$, $b^0_0 = IV_2$, and set the other starting points randomly. Then, we use switches to jump for an already reachable pair (a_{j_0}, b_{k_0}) to a different pair (a_{j_0}, b_{k_1}) (or to (a_{j_1}, b_{k_0}), respectively). By using $2^{2t} - 1$ switches, we can make all pairs reachable. There are many way to combine the switches; a simple one can be described as follow:

1. first, build switches from (a_0, b_0) to each of the (a_0, b_k)'s;
2. then for each k, build a series of switches from (a_0, b_k) to all the (a_j, b_k)'s.

In order to reach the chains (a_j, b_k), one would activate the k-th switch in the first part to jump from (a_0, b_0) to (a_0, b_k), and then the j-th switch in the k-th series of the second part to jump from (a_0, b_k) to (a_j, b_k). This structure is shown in Figure 3 and a pseudo-code description is given by Algorithm 2, where the Interchange functions builds the structure, and the SelectMessage function extracts the message reaching (a_j, b_k).

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

We believe that a 2^t-interchange structure based on switches will need at least $\Theta(2^{2t})$ switches, because every switch can only increase the number of reachable

Algorithm 1 Building a single switch

```
function SWITCH(h_1, h_2, a, b, b')
    x ← a
    M ← ∅
    for 0 ≤ i < n/2 do
        (m, m') ← COLLISION(h_1, x)
        M ← (M ‖ m) ∪ (M ‖ m')
        x ← h_1(x, m)
    end for
    H ← {}
    for M ∈ M do
        y ← h_2^*(b, M)
        H[y] ← M
    end for
    for M ∈ M do
        y ← h_2^*(b', M)
        if H[y] exists then
            return M, H[y]
        end if
    end for
end function

function COLLISION(h, x)
    H ← {}
    loop
        m ← $
        y ← h_1(x, m)
        if H[y] exists then
            return m, H[y]
        else
            H[y] ← m
        end if
    end loop
end function
```

pairs (a_j, b_k) by one. As shown in Appendix A some switches can be saved in the beginning but it seems that new ideas would be needed to reduce the total complexity below $\Theta(2^{2t+n/2})$.

5 Preimage Attack

Finally, we describe the full preimage attack. We first build an interchange structure with 2^t chains for each of H_1 and H_2. We denote the ending points as $\{A_j, j = 0 \dots 2^t - 1\}$ and $\{B_k, k = 0 \dots 2^t - 1\}$, and we know how to select a message $\boldsymbol{M}_{jk}$ to reach any state (A_j, B_k). When adding a message block m to one of the messages $\boldsymbol{M}_{jk}$ in the interchange structure, the output of the combiner

Algorithm 2 Building and using a T-interchange structure

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

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

can be written as:

$$H_1(\boldsymbol{M}_{jk} \parallel m) \oplus H_2(\boldsymbol{M}_{jk} \parallel m) = g_1(h_1(A_j, m), \ell + 1) \oplus g_2(h_2(B_k, m), \ell + 1),$$

where g_1 and g_2 are the finalization functions of H_1 and H_2, respectively, and ℓ is the length of the messages in the structure.

In order to reach a target value $\overline{H}$, we select a random block m, and we evaluate $\left\{A'_j = g_1(h_1(A_j, m), \ell + 1), j = 0 \ldots 2^t - 1\right\}$ and $\left\{B'_k = \overline{H} \oplus g_2(h_2(B_k, m), \ell +\right.$

$1), k = 0 \dots 2^t - 1\big\}$. If there is a match (j^*, k^*) between the two lists, we have:

$$\begin{aligned} A'_{j^*} = B'_{k^*} &\Leftrightarrow g_1(h_1(A_{j^*}, m), \ell + 1) = \overline{H} \oplus g_2(h_2(B_{k^*}, m), \ell + 1) \\ &\Leftrightarrow H_1(\boldsymbol{M}_{jk} \parallel m) \oplus H_2(\boldsymbol{M}_{jk} \parallel m) = \overline{H}. \end{aligned}$$

For a random choice of m, we expect that a match exists with probability 2^{2t-n}, and testing it requires about 2^t operations[9]. We will have to repeat this procedure 2^{n-2t} times on average, therefore the total cost of the preimage search is about 2^{n-t} evaluations of h_1 and h_2.

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

5.1 Message length and memory complexity

The attack uses messages of length $n/2 \cdot 2^{2t}$, and the memory complexity of the attack[10] is also $n/2 \cdot 2^{2t}$. The optimal choice $t = n/6$ gives messages of length $n/2 \cdot 2^{n/3}$. The memory requirement is probably not an issue[11] for an attacker that can spend time $2^{5n/6}$, but the message length can be a problem with some hash functions that don't accept long inputs. For instance SHA-256 is only defined for message with less than 2^{64} bits (*i.e.* 2^{55} blocks).

In this case, one can apply the attack with a smaller value of t: this reduces the length of the messages, at the cost of more time spent in the preimage search step. For instance, we can mount a preimage attack against SHA-256 $\oplus$ BLAKE-256 with complexity 2^{232} using $t = 24$, while the optimal attack with $n = 256$ would cost only $2^{220.3}$. Similarly, our attack applied to SHA-512 $\oplus$ Whirlpool has a complexity of 2^{461}, rather than $2^{434.7}$.

6 Applications and Extensions

The attack works identically if the hash functions use the HAIFA mode rather than the plain Merkle-Damgård iteration. Also it can easily be extended to $H_1(M) \odot H_2(M)$ where $\odot$ denotes an easy to invert group operation (for instance, a modular addition rather than the exclusive or). The attack can also be extended to hash functions H_1 and/or H_2 using an internal check-sum, such as the GOST family of hash functions, using pairs of blocks with a constant sum.

[9] It takes $O(t \cdot 2^t)$ operations by sorting the lists, but only $2 \cdot 2^t$ using a hash table.
[10] We only need to store the messages $\boldsymbol{M}$ and $\boldsymbol{M'}$
[11] For instance, the attack is on the verge of practicality with $n = 64$; the time complexity is $2^{58.3}$ and the memory complexity is $2^{26.3}$.

6.1 Application to the sum of wide-pipe hash functions

The attack can also be used when the internal state size ℓ is larger than the output size n. The complexity of building a 2^t-interchange structure is related to ℓ as $\ell/2 \cdot 2^{2t+\ell/2}$. On the other hand, the complexity of the meet-in-the-middle preimage search is related to n as 2^{n-t}. The optimal complexity is $\ell/2 \cdot 2^{2n/3+\ell/6}$ by matching the two complexities with $t = n/3 - \ell/6$. Therefore our attack can be applied as long as $\ell + 6\log(\ell) \leq 2n$ holds. For instance, we can compute preimages of SHA-224 $\oplus$ BLAKE-224 with complexity roughly 2^{199}.

6.2 Application to Cryptophia's short combiner

Our attack can also be applied to Cryptophia's short combiner, as proposed by Mittelbach [32], and to the revised version of Mennink and Preneel [29]. This combiner computes the sum of two hash functions with some pre-processing of the message, to allow non-independent functions:

$$\begin{aligned} C(M) &= H_1\left(\tilde{m}_1^1 \parallel \ldots \parallel \tilde{m}_\ell^1\right) \oplus H_2\left(\tilde{m}_1^2 \parallel \ldots \parallel \tilde{m}_\ell^2\right) \\ \tilde{m}_j^1 &= H_1(0 \parallel l_1 \parallel m_j \oplus k_1) \oplus H_2(0 \parallel l_2 \parallel m_j \oplus k_2) \\ \tilde{m}_j^2 &= H_1(1 \parallel l_1 \parallel m_j \oplus k_1) \oplus H_2(1 \parallel l_2 \parallel m_j \oplus k_2) \end{aligned}$$

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

However, if both H_1 and H_2 are narrow-pipe, we can apply our preimage attack with complexity $\tilde{O}(2^{5n/6})$. This does not violate the security proof because we need both functions to be narrow-pipe, hence not n-bit ideal[12]. From a practical point of view, though, it shows that in many cases (*e.g.* using SHA-512 and Whirlpool) the combiner is *weaker* than the initial functions.

6.3 Improvements using weaknesses of the hash functions

If H_1 or H_2 has known cryptographic weaknesses, more efficient attacks are possible. More precisely, if the compression function of one of the hash functions can be inverted[13] in time 2^t, then we can find a preimage of $H_1 \oplus H_2$ with complexity only $\tilde{O}(2^{(n+t)/2})$.

The attack is presented using the case, where (at least) one compression function is modeled as a weak compression function defined in [24], as an example. Without loss of the generality, we assume the compression function of H_2, $h_2(x, y) = z$, is such a weak compression function, which is a random oracle with two additional interfaces as below.

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

[13] finding an input chaining value from the output chaining value and the message

- *Backward interface.* On a query $(?, y, z)$, it returns either a value x uniformly chosen from all the values satisfying $h_2(x, y) = z$, or $\perp$ if no such x exists.

- *Bridging interface.* On a query $(x, ?, z)$, it return either a value y uniformly chosen from all the values satisfying $h_2(x, y) = z$, or $\perp$ if no such y exists.

Note that the inversion of compression function h_2 takes unit time and hence the attack against $H_1 \oplus H_2$ takes time $\tilde{O}(2^{n/2})$. The procedure is detailed as follows, which is also illustrated in Figure 6.

Let the target hash digest be denoted as $\overline{H}$. We firstly build an n-block long multicollision on H_1 following Joux's approach [20]. Let the final output be denoted as Z. It contains a set of up to 2^n messages that link IV_1 to Z on H_1. We split every n-block long multicolliding message into halves, and collect the first half $n/2$-block long messages as a set $\mathcal{M}_1$ and the second half as another set $\mathcal{M}_2$. Hence for any message $M_1 \in \mathcal{M}_1$ and any $M_2 \in \mathcal{M}_2$, the concatenated message $M_1 \parallel M_2$ links IV_1 to Z on H_1. Secondly, for the messages $M_2 \in \mathcal{M}_2$, we use the additional backward interface of h_2 to carry out backward computations from $\overline{H} \oplus Z$, and get values X such that $h_2^*(X, M_2) = \overline{H} \oplus Z$. We store (X, M_2) in a table T_X. On average T_X contains $2^{n/2}$ elements.[14] Finally, for each message $M_1 \in \mathcal{M}_1$, we iteratively compute h_2 forward from IV_2, and get an internal state Y. We match Y to the elements in T_X. A match implies that concatenated $M_1 \parallel M_2$ links IV_2 to $\overline{H} \oplus Z$, and in turn is a preimage of $H_1(M_1 \parallel M_2) \oplus H_2(M_1 \parallel M_2) = Z \oplus \overline{H} \oplus Z = \overline{H}$. The success probability of finding such a match is not negligible since there are $2^{n/2}$ X's and Y's. The complexity is around $n \cdot 2^{n/2}$ that is $\tilde{O}(2^{n/2})$ by ignoring the polynomial factors.

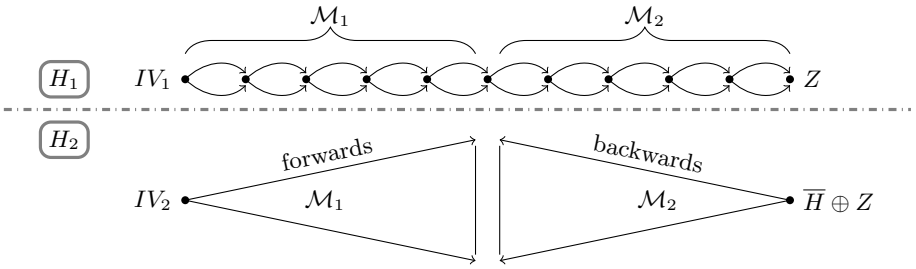

Fig. 6. Preimage attack on the XOR combiner with a weak H_2

Moreover, the above preimage attack can be extended to the concatenation combiner $H_1 \parallel H_2$ if both H_1 and H_2 are weak by a minor modification. This

[14] The backward interface may output $\perp$ for some message block. To compensate it, for the other message blocks, we make multiple queries, since they may have more than one preimages. On average, the backward interface should produce one preimage for each query.

shows that the proof of Hoch and Shamir [17] is tight for preimage resistance. Here we mainly highlight the modifications. Let the target hash digest be $\overline{H}_1 \| \overline{H}_2$, where $\overline{H}_1$ is from H_1 and $\overline{H}_2$ from H_2. After we build the multicollision on H_1 and let the output internal state be denoted as Z, we link Z to $\overline{H}_1$ by using the bridging interface of h_1 to receive a message m such that $h_1(Z, m) = \overline{H}_1$. This gives us a set of messages linking IV_1 to $\overline{H}_1$ on H_1. Also note that for the backward computations on H_2, the starting value should be $\overline{H}_2$. The rest of the attack procedure remains the same. Hence it is easy to get that the complexity is also $\tilde{O}(2^{n/2})$.

6.4 Extension to the sum of three or more hash functions

The attack can be extended to the sum of three or more hash functions. In order to attack the sum of k functions, two different strategies are possible: either we use a simpler structure that only gives two degrees of freedom, and fixes $k-2$ functions to a constant value, or we build an interchange structure to control all the k functions independently.

Controlling only two functions. The easiest way to extend the attack is to use a single chain in the $k-2$ extra hash functions. The procedure to build a switch is modified in order to use multicollisions for $k-1$ functions instead a simple multicollisions for one function; this costs $O(n^{k-1} \cdot 2^{n/2})$ using Joux's method [20].

As in the basic attack, we need $O(t^2)$ switches to generate a 2^t-interchange for two functions, and the preimage search costs $O(2^{n-t})$; the optimal complexity is therefore $O(n^{k-1} \cdot 2^{5n/6})$ with $t = n/6$.

Controlling all the functions. Alternatively, we can build a more complex interchange structure in order to control all the functions independently. When attacking three functions, we will use the switch structure to jump from chains $(a_{j_0}, b_{k_0}, c_{l_0})$ to $(a_{j_0}, b_{k_0}, c_{l_1})$ (or $(a_{j_0}, b_{k_1}, c_{l_0})$ or $(a_{j_1}, b_{k_0}, c_{l_0})$, respectively). We need $2^{3t} - 1$ switches in the interchange structure to reach all the 2^{3t} triplets of chains (a switch makes only one new triplet reachable). Each switch is built using a $2^{n/2}$-multicollision on two functions, which can be built for a cost of $O(n^2 \cdot 2^{n/2})$ following Joux's technique [20]. Therefore we can build a 2^t-interchange for a cost of $O(n^2 \cdot 2^{3t+n/2})$. More generally, for the sum of k hash functions, we can build an interchange structure for k functions for a cost of $O(n^{k-1} \cdot 2^{kt+n/2})$.

In the preimage search phase, we generate k lists of size 2^t, and we need to detect efficiently whether we can combine then to generate a zero sum. This problem can be solved using a algorithm similar to Wagner's generalized birthday algorithm [40]. If $k = 2^\kappa$, we find a solution with probability $O(2^{n-(\kappa+1)\cdot t})$ for a cost of $O(k \cdot 2^t)$. Therefore the preimage search costs $O(k \cdot 2^{n-\kappa t})$. With $k = 4$ (i.e. $\kappa = 2$), this yields a complexity of $O(n^3 \cdot 2^{5n/6})$. However, this approach is less efficient than the previous one for $k = 3$ and for $k > 4$.

To summarize, attacking the sum of k hash functions ($k \geq 2$) costs $O(n^{k-1} \cdot 2^{5n/6})$. Controlling chains independently in more than two hash function might be useful for further work, but it doesn't improve the preimage attack on the sum of k hash functions.

7 Conclusion and open discussions

In this work, we gave the first generic attack on the XOR combiner. Our result is rather surprising: the sum of two ideal narrow-pipe hash functions only has about $5n/6$ bits of security against preimage attacks. In particular, the *sum is easier to break* than the initial functions. Since most practical hash functions are narrow-pipe (*e.g.* SHA-1, SHA-256, SHA-512, Whirlpool, RIPEMD, GOST, BLAKE, Skein...), the XOR combiner will usually provide a weaker security than the component hash functions.

Moreover, we would like to discuss a few directions for future work.

On controlling multiple hash lanes. Since 2004, several generic attacks have been found against narrow-pipe hash functions, such as the multicollision attack[20], the long-message second preimage attack[22] and the herding attack[21]. There has been extensive work to extend these attacks to more complex constructions with several computation chains, such as the concatenation $H_1(M) \parallel H_2(M)$ and the cascade $H_2(H_1(M), M)$.

As in our present work, the essential difficulty in those attacks comes from the fact that several lanes of computation share the same input message, and hence their outputs are related. If an adversary considers a naïve set of messages, the set of outputs gives random pairs of n-bit values $\{(A_i, B_i) : i \in \mathcal{I}\}$: selecting a value for the first entry of the pair gives a single candidate for the second entry, and the adversary is essentially working with a $2n$-bit state. Previous works [33,16,2] have developed various message structures (mostly based on multicollision and diamond structures), in order to relax this relation. They mainly result in a set of messages $\mathcal{M}$ such that the corresponding outputs are in a more structured set $\{(A, B_i) : i \in \mathcal{I}\}$: the first entry is a constant value A, but several options B_i can be selected for the second entry. For any value (A, B_i), it is then possible to select a message in $\mathcal{M}$ so that the first lane reaches A, while the second lane reaches B_i. This allows to modify the value of the second lane without affecting the first lane.

Our result is quite stronger: with the interchange structure we have a set of message such that the corresponding outputs are a set $\{(A_i, B_j) : i \in \mathcal{I}, j \in \mathcal{J}\}$, where *both* lanes have several options that can be selected independently. We hope that our technique will have applications or lead to new technical development in related settings, e.g., the open problem of generic second preimage attacks (with long messages) on the concatenation hash or on the Zipper hash [24].

On extending to practical hash function. Several practical hash functions such as RIPEMD [10] and HAS-V [34] are based on a compression function with more than one independent lanes, which interacts with each other at the end of each compression function call. It is very interesting to investigate if our attack can be further modified to attack these hash functions in future. Again the obstacle comes from the relations between internal states of lanes. Particularly, the internal states of the lanes interact with each other, which makes the relation even tighter and in turn harder to attack.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments. Lei Wang is supported by the Singapore National Research Foundation Fellowship 2012 (NRF-NRFF2012-06).

References

1. Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.): Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations, Lecture Notes in Computer Science, vol. 5126. Springer (2008)
2. Andreeva, E., Bouillaguet, C., Dunkelman, O., Kelsey, J.: Herding, Second Preimage and Trojan Message Attacks beyond Merkle-Damgård. In: Jr., M.J.J., Rijmen, V., Safavi-Naini, R. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 5867, pp. 393–414. Springer (2009)
3. Biham, E., Dunkelman, O.: A Framework for Iterative Hash Functions - HAIFA. IACR Cryptology ePrint Archive, Report 2007/278 (2007)
4. Boneh, D., Boyen, X.: On the Impossibility of Efficiently Combining Collision Resistant Hash Functions. In: Dwork, C. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 4117, pp. 570–583. Springer (2006)
5. Brassard, G. (ed.): Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, Lecture Notes in Computer Science, vol. 435. Springer (1990)
6. Cramer, R. (ed.): Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3494. Springer (2005)
7. Damgård, I.: A Design Principle for Hash Functions. In: Brassard [5], pp. 416–427
8. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard) (Jan 1999), `http://www.ietf.org/rfc/rfc2246.txt`, obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176
9. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard) (Aug 2008), `http://www.ietf.org/rfc/rfc5246.txt`, updated by RFCs 5746, 5878, 6176

10. Dobbertin, H., Bosselaers, A., Preneel, B.: RIPEMD-160: A Strengthened Version of RIPEMD. In: Gollmann, D. (ed.) FSE. Lecture Notes in Computer Science, vol. 1039, pp. 71–82. Springer (1996)
11. Fischlin, M., Lehmann, A.: Multi-property Preserving Combiners for Hash Functions. In: Canetti, R. (ed.) TCC. Lecture Notes in Computer Science, vol. 4948, pp. 375–392. Springer (2008)
12. Fischlin, M., Lehmann, A., Pietrzak, K.: Robust Multi-property Combiners for Hash Functions Revisited. In: Aceto et al. [1], pp. 655–666
13. Fischlin, M., Lehmann, A., Pietrzak, K.: Robust Multi-Property Combiners for Hash Functions. J. Cryptology 27(3), 397–428 (2014)
14. Freier, A., Karlton, P., Kocher, P.: The Secure Sockets Layer (SSL) Protocol Version 3.0. RFC 6101 (Historic) (Aug 2011), http://www.ietf.org/rfc/rfc6101.txt
15. Her, Y.S., Sakurai, K.: A Design of Cryptographic Hash Function Group with Variable Output-Length Based on SHA-1. Technical report of IEICE. ISEC 102(212), 69–76 (July 2002), http://ci.nii.ac.jp/naid/110003298501/en/
16. Hoch, J.J., Shamir, A.: Breaking the ICE - Finding Multicollisions in Iterated Concatenated and Expanded (ICE) Hash Functions. In: Robshaw [37], pp. 179–194
17. Hoch, J.J., Shamir, A.: On the Strength of the Concatenated Hash Combiner When All the Hash Functions Are Weak. In: Aceto et al. [1], pp. 616–630
18. Hong, D., Chang, D., Sung, J., Lee, S., Hong, S., Lee, J., Moon, D., Chee, S.: A New Dedicated 256-Bit Hash Function: FORK-256. In: Robshaw [37], pp. 195–209
19. Indesteege, S.: The lane hash function. Submission to NIST (2008), http://www.cosic.esat.kuleuven.be/publications/article-1181.pdf
20. Joux, A.: Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In: Franklin, M.K. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 3152, pp. 306–316. Springer (2004)
21. Kelsey, J., Kohno, T.: Herding Hash Functions and the Nostradamus Attack. In: Vaudenay, S. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 183–200. Springer (2006)
22. Kelsey, J., Schneier, B.: Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In: Cramer [6], pp. 474–490
23. Lehmann, A.: On the Security of Hash Function Combiners. Ph.D. thesis, TU Darmstadt (2010)
24. Liskov, M.: Constructing an Ideal Hash Function from Weak Ideal Compression Functions. In: Biham, E., Youssef, A.M. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 4356, pp. 358–375. Springer (2006)
25. Mendel, F., Nad, T., Scherz, S., Schläffer, M.: Differential attacks on reduced RIPEMD-160. In: Gollmann, D., Freiling, F.C. (eds.) ISC 2012. Lecture Notes in Computer Science, vol. 7483, pp. 23–38. Springer (2012)
26. Mendel, F., Peyrin, T., Schläffer, M., Wang, L., Wu, S.: Improved Cryptanalysis of Reduced RIPEMD-160. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT (2). Lecture Notes in Computer Science, vol. 8270, pp. 484–503. Springer (2013)
27. Mendel, F., Pramstaller, N., Rechberger, C., Rijmen, V.: On the collision resistance of RIPEMD-160. In: Katsikas, S.K., Lopez, J., Backes, M., Gritzalis, S., Preneel, B. (eds.) ISC 2006. Lecture Notes in Computer Science, vol. 4176, pp. 101–116. Springer (2006)
28. Mendel, F., Rechberger, C., Schläffer, M.: MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners. In: Matsui, M. (ed.) ASIACRYPT. Lecture Notes in Computer Science, vol. 5912, pp. 144–161. Springer (2009)
29. Mennink, B., Preneel, B.: Breaking and Fixing Cryptophia's Short Combiner. In: Gritzalis, D., Kiayias, A. (eds.) CANS. Lecture Notes in Computer Science (2014)

30. Merkle, R.C.: One Way Hash Functions and DES. In: Brassard [5], pp. 428–446
31. Mittelbach, A.: Hash Combiners for Second Pre-image Resistance, Target Collision Resistance and Pre-image Resistance Have Long Output. In: Visconti, I., Prisco, R.D. (eds.) SCN. Lecture Notes in Computer Science, vol. 7485, pp. 522–539. Springer (2012)
32. Mittelbach, A.: Cryptophia's Short Combiner for Collision-Resistant Hash Functions. In: Jr., M.J.J., Locasto, M.E., Mohassel, P., Safavi-Naini, R. (eds.) ACNS. Lecture Notes in Computer Science, vol. 7954, pp. 136–153. Springer (2013)
33. Nandi, M., Stinson, D.R.: Multicollision Attacks on Some Generalized Sequential Hash Functions. IEEE Transactions on Information Theory 53(2), 759–767 (2007)
34. Park, N.K., Hwang, J.H., Lee, P.J.: HAS-V: A New Hash Function with Variable Output Length. In: Stinson, D.R., Tavares, S.E. (eds.) Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 2012, pp. 202–216. Springer (2000)
35. Pietrzak, K.: Non-trivial Black-Box Combiners for Collision-Resistant Hash-Functions Don't Exist. In: Naor, M. (ed.) EUROCRYPT. Lecture Notes in Computer Science, vol. 4515, pp. 23–33. Springer (2007)
36. Rjasko, M.: On Existence of Robust Combiners for Cryptographic Hash Functions. In: Vojtás, P. (ed.) ITAT. CEUR Workshop Proceedings, vol. 584, pp. 71–76. CEUR-WS.org (2009)
37. Robshaw, M.J.B. (ed.): Fast Software Encryption, 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers, Lecture Notes in Computer Science, vol. 4047. Springer (2006)
38. Sasaki, Y., Wang, L.: Distinguishers beyond Three Rounds of the RIPEMD-128/-160 Compression Functions. In: Bao, F., Samarati, P., Zhou, J. (eds.) ACNS. Lecture Notes in Computer Science, vol. 7341, pp. 275–292. Springer (2012)
39. Shoup, V. (ed.): Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, Lecture Notes in Computer Science, vol. 3621. Springer (2005)
40. Wagner, D.: A Generalized Birthday Problem. In: Yung, M. (ed.) CRYPTO. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer (2002)
41. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup [39], pp. 17–36
42. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer [6], pp. 19–35

A Optimized Interchange Structure

We now describe an optimized attack using only $(2^t-1)(2^t-1)$ switches rather than $2^{2t}-1$. The attack also requires multicollision structures, as introduced by Joux[20].

We replace the first 2^t-1 switches with a 2^t-multicollision in H_1, and we use those messages to initialize all the b_k chains in H_2. We can also optimize the first series of switches in H_2 in the same way: we build a 2^t-multicollision in H_2 starting from b_0, and we use those messages to initialize the a_j chains in H_1. This is illustrated by Figure 7, and the detailed attack is given as Algorithm 3.

Algorithm 3 Optimized T-interchange structure

function INTERCHANGE(h_1, h_2, IV_1, IV_2, T)
  $a_0 \leftarrow IV_1$, $b_0 \leftarrow IV_2$
  $\mathcal{M}_0 \leftarrow$ MULTICOLLISION(h_1, a_0)
  for $0 \leq k < T$ **do**
    $b_k \leftarrow h_2^*(b_0, \mathcal{M}_0[k])$
  end for
  $a_0 \leftarrow h_1^*(a_0, \mathcal{M}_0[0])$
  $\mathcal{M}_1 \leftarrow$ MULTICOLLISION(h_2, b_0)
  for $1 \leq k < T$ **do**
    $a_k \leftarrow h_2^*(a_0, \mathcal{M}_1[k])$
  end for
  $a_0 \leftarrow h_2^*(a_0, \mathcal{M}_1[0])$
  $b_0 \leftarrow h_1^*(b_0, \mathcal{M}_1[0])$
  for $2 \leq j < T$ **do**
    for $1 \leq i < T$ **do**
      $(M, M') \leftarrow$ SWITCH(h_2, h_1, b_j, a_0, a_i)
      $\boldsymbol{M} \leftarrow \boldsymbol{M} \parallel M; \boldsymbol{M'} \leftarrow \boldsymbol{M'} \parallel M'$
      for $0 \leq k < T$ **do**
        $a_k \leftarrow h_1^*(a_k, M)$
        $b_k \leftarrow h_2^*(b_k, M)$
      end for
    end for
  end for
  return $(\mathcal{M}_0, \mathcal{M}_1, \boldsymbol{M}, \boldsymbol{M'})$
end function

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

function MULTICOLLISION(h, x)
  $\mathcal{M} \leftarrow \{\}$
  for $0 \leq i < n/2$ **do**
    $(m, m') \leftarrow$ COLLISION(h, x)
    $x \leftarrow h(x, m)$
    $\mathcal{M} \leftarrow (\mathcal{M} \parallel m) \cup (\mathcal{M} \parallel m')$
  end for
  return $\mathcal{M}$
end function

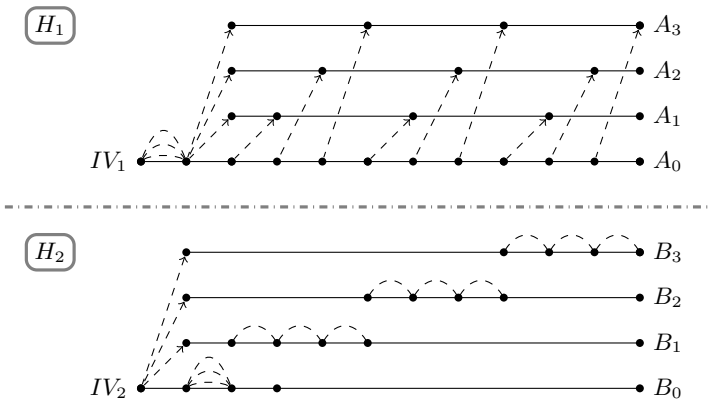

Fig. 7. Optimized interchange structure