

On the Strength of the Concatenated Hash Combiner when All the Hash Functions are Weak

Jonathan J. Hoch and Adi Shamir

Department of Computer Science and Applied Mathematics,
The Weizmann Institute of Science, Israel
{yaakov.hoch,adi.shamir}@weizmann.ac.il

Abstract. At Crypto 2004 Joux showed a novel attack against the concatenated hash combiner instantiated with Merkle-Damgård iterated hash functions. His method of producing multicollisions in the Merkle-Damgård design was the first in a recent line of generic attacks against the Merkle-Damgård construction. In the same paper, Joux raised an open question concerning the strength of the concatenated hash combiner and asked whether his attack can be improved when the attacker can efficiently find collisions in both underlying compression functions. We solve this open problem by showing that even in the powerful adversarial scenario first introduced by Liskov (SAC 2006) in which the underlying compression functions can be fully inverted (which implies that collisions can be easily generated), collisions in the concatenated hash cannot be created using fewer than $2^{n/2}$ queries. We then expand this result to include the double pipe hash construction of Lucks from Asiacrypt 2005. One of the intermediate results is of interest on its own and provides the first streamable construction provably indistinguishable from a random oracle in this model.

Key words: hash functions, cryptographic combiners, indistinguishability.

1 Introduction

Cryptanalysis of hash functions has been a very active area of research in the past few years. A flurry of attacks have been found against various hash functions including SHA-1 and the MD variants (see [10, 16–19]). Besides these attacks on specific hash functions, a number of novel generic attacks against the Merkle-Damgård [5, 14] iterated construction have been published as well. These include among others Joux’s multicollision attack [7], Kelsey and Schneier’s expandable message attack [9] and Kelsey and Kohno’s herding attack [8]. Joux’s multicollision attack demonstrates how to find collisions in a concatenated hash construction $H(M) = F(M)||G(M)$ when at least one of the underlying hash functions is iterated.

In the classic combiner scenario we have two instantiations, I_1 and I_2 , of some cryptographic primitive, e.g., two encryption schemes or two hash functions. The goal is to build a new combined instantiation I of the primitive, which remains secure even when one of the underlying primitives is broken, as long as the other remains secure. In contrast to this classical approach, we will show that certain hash combiners retain a provable level of security even if all of the underlying hash functions are compromised, provided that the two primitives are sufficiently random and sufficiently different in a sense which will be made precise later.

1.1 Related Work

Joux's innovative attack focused attention on the security properties of hash combiners as his attack shows that the trivial combiner does not improve over the security of the underlying hash functions. A line of research concerning hash combiners has followed, demonstrating that security amplifying combiners exist [6] and on the other hand proving that any provably secure black-box combiner must preserve the total length of the underlying hash functions [1, 15]. Other responses to Joux's paper include Lucks' [12] proposal of the wide/double piped constructions whose aim was to overcome the multicollision attack by using a larger internal state. Lucks' proposal is provably secure in the random oracle model against multicollisions. Maurer *et al.* [13] introduced the notion of indistinguishability. Similar to the concept of indistinguishability, this notion describes a situation in which two systems are indistinguishable despite having extra access to the internal structure of the systems. Inspired by the generic attacks against the Merkle-Damgård iterated construction, Coron *et al.* [3] operated within the indistinguishability framework to show how iterated hash functions can be proved indistinguishable from random oracles in the ideal cipher model.¹ Liskov further pursued this approach in [11] by introducing *weak compression functions*. A weak compression function behaves like a random oracle except that the adversary is given access to corresponding inversion oracles. Liskov presented a new hash construction, the *zipper hash*, composed of a pair of weak compression functions and using the framework of Coron *et al.* proved it indistinguishable from a random oracle. In Joux's attack he did not assume that the attacker can find collisions in the underlying compression functions faster than the birthday paradox bound. Joux then posed the question whether the ability to find collisions efficiently in both the underlying compressions functions can help the attacker improve the complexity of his attack.

1.2 Our Results

In this paper we prove that even in a very strong attack scenario in which the attacker can find not only collisions but even invert in unit time *all* the compression functions on inputs of his choice, the best attack against the concatenated construction is Joux's multicollision attack with complexity $\mathcal{O}(2^{n/2})$. Furthermore, as an intermediate result we show a streamable² hash construction, provably indistinguishable from a random oracle in the model of weak compression functions, which has the same rate as the non-streamable zipper hash of Liskov [11]. This result is then extended to prove that the double pipe hash construction of Lucks [12] is also indistinguishable from a random oracle in the same model. We stress that the model of weak compression functions captures all black-box generic attacks arising from collision or preimage finding attacks against the underlying compression functions.

1.3 Paper Organization

Section 2 describes the model of weak compression functions and gives our notation for the rest of the paper. Section 3 proves the main result of the paper, namely that in the model of weak compression functions, finding collisions in the concatenated hash combiner requires $\mathcal{O}(2^{n/2})$ operations. Finally, Section 4 proves the indistinguishability of Lucks' double pipe hash construction.

¹ The underlying compression function is modelled as an ideal cipher.

² A hash construction in which each block of the message can be processed once and then be forgotten. This is an essential requirement in applications where the hash is computed on the fly from a data stream.

2 The Model

We first give a short description of the iterated hash construction. An iterated hash function $F^f : \{0, 1\}^* \rightarrow \{0, 1\}^n$ is built by iterating a basic compression function $f : \{0, 1\}^m \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ as follows:

- Split a message M into k , m -bit blocks x_1, \dots, x_k .
- Set $h_0 = IV$ where IV is the initialization vector.
- For each message block x_i compute $h_i = f(h_{i-1}, x_i)$.
- Output $F^f(M) = h_k$.

The classical Merkle-Damgård construction also contains padding and length encoding which we will ignore for the sake of simplicity since they do not affect our results.

Following Joux's open question, we will try to model a situation in which the attacker can efficiently find collisions in either compression function, but do not assume any other special properties of these colliding pairs. In fact we will give our adversary even stronger oracle access and allow him to find in unit time random preimages of two different types as well. Formally, let f and g be compression functions from $m + n$ bits to n bits, and let F and G be the corresponding hash functions built by instantiating the Merkle-Damgård paradigm with f and g respectively. We will model f and g as random functions provided as black box oracles with additional respective *inversion* oracles.

We define the following oracles:

- $f^*(x, ?, z) \rightarrow (x, y, z)$ where y is chosen uniformly such that $f(x, y) = z$, or \perp if no such y exists.
- $f^{-1}(?, y, z) \rightarrow (x, y, z)$ where x is chosen uniformly such that $f(x, y) = z$, or \perp if no such x exists.
- $g^*(x, ?, z) \rightarrow (x, y, z)$ where y is chosen uniformly such that $g(x, y) = z$, or \perp if no such y exists.
- $g^{-1}(?, y, z) \rightarrow (x, y, z)$ where x is chosen uniformly such that $g(x, y) = z$, or \perp if no such x exists.

f and g queries will be called *forward* queries, g^{-1} and f^{-1} queries will be called *backward* queries and f^* and g^* queries will be called *bridging* queries.³ The slightly more complicated case in which these inverses are not uniformly distributed will be discussed at the end of this section. One should notice that while weak compression functions are indeed weak in the sense that they allow trivial collision and preimage attacks, there are some operations in which they do not assist at all. For example, given two chaining values x_1 and x_2 finding a message block y such that $f(x_1, y, ?) = f(x_2, y, ?)$ still requires $\mathcal{O}(2^{n/2})$ queries.

We now introduce a slight modification due to Liskov [11] of the framework of Coron *et al.* [3] and Maurer *et al.* [13]. This framework will enable us to prove that certain hash functions based on weak compression functions are indifferentiable from random oracles. Let Γ be an oracle encapsulating $f, f^{-1}, f^*, g, g^{-1}$ and g^* .

Definition 1 (indifferentiability). *A construction C is (q, ϵ) indifferentiable in the presence of Γ from a random oracle RO if there exists a polynomial time simulator S , such that for every distinguisher D which uses at most q oracle queries (to either of the oracles),*

$$|Pr[D^{C, \Gamma} = 1] - Pr[D^{RO, S^{RO}} = 1]| < \epsilon$$

Notice that this definition is slightly different from the usual notion of indistinguishability in that the simulator, besides simulating the behavior of Γ , must also remain consistent with the random oracle RO . The following example illustrates the problem. Let C be an iterated hash function built from a compression function f and assume that f is a random oracle. The pair (C, f) is differentiable from (RO, S^{RO}) for any simulator S . The

³ Liskov in [11] used the term *squeezing* queries.

distinguisher D , when presented with a pair (A, B) , performs the following queries $h_1 = A(m_1)$, $h_2 = B(h_1, m_2)$, $h = A(m_1 m_2)$. If $h = h_2$ the distinguisher returns 1 and otherwise 0. When D is presented with the pair (C, f) , the equality will always hold and $\Pr[D^{C,f} = 1] = 1$. On the other hand, for any simulator S , the probability over the random coins of S and the random oracle that $S^{RO}(m_2) = RO(m_1 m_2)$ is negligible. In this example, the distinguisher worked since the simulator could not maintain the required consistency with RO . So we see that S does not only need to simulate Γ *per se* but also needs to maintain the relation of S relative to the RO , simulating the relationship between Γ and C as well. Maurer *et al.* [13] proved that this definition of indistinguishability will allow us to use the construction C in place of a random oracle in any cryptography protocol and retain the same level of provable security.

Another subtle issue is the fact that in our case Γ includes inversion oracles. Notice that when f is a random function, a fixed fraction of the queries $f^{-1}(?, y, z)$ do not have answers, while other queries might have multiple possible answers. We have defined f^{-1} and f^* to return an answer uniformly distributed the possible answers, and thus the simulator S must reproduce the same distribution of the number of inverses which is known to be Poisson.⁴ If we would like to model inversion oracles with a non-uniform distribution, the simulator will need to model this distribution as well.

3 A Lower Bound

Using techniques similar to those introduced by Coron et al. we will show that the construction $C(M) = F(M) \oplus G(M)$ is indistinguishable from a random oracle RO when less than $\mathcal{O}(2^{n/2})$ queries are performed. Since finding collisions in $H(M) = F(M) \parallel G(M)$ implies finding collisions in $C(M)$ as well, the indistinguishability of $C(M)$ will give us a lower bound on the number of queries required to find a collision in $H(M)$ with non-negligible probability. Notice that the same proof can be used for any construction of the form $H(M) = \alpha(F(M), G(M))$ for any n -bit function α which is uniquely invertible when its output and any one of its input parameters is known.

Let Γ be an oracle implementing $f, g, f^{-1}, f^*, g^{-1}$ and g^* . Let RO be a random oracle and let S^{RO} be an oracle Turing machine with the same black-box interface as Γ . In order to prove the indistinguishability result, we will give a hybrid argument and show that any distinguisher D cannot differentiate between interacting with the pair (C, Γ) and the pair (RO, S^{RO}) .

3.1 The Simulator S

We want the simulator S^{RO} to simulate Γ such that for any distinguisher D , which performs $q \ll 2^{n/2}$ queries⁵, $|\Pr[D^{C,\Gamma} = 1] - \Pr[D^{RO,S} = 1]|$ is negligible. Obviously we would like the simulator S to produce random responses to the simulated queries while maintaining consistency. The naive approach would be to keep a list of all answers given so far and each time S receives a new query, it will return a random value consistent with the values returned so far. Notice that there are two types of consistency involved: self consistency

⁴ Note that Liskov in [11] neglected to handle this problem, and therefore his simulator suffers from the fact that a distinguisher can query f^{-1} on a large number of random inputs and the simulator will always return an inverse whereas a true random function will only have inverses for $1 - 1/e$ fraction of the inputs.

⁵ We will charge queries to C or RO differently than queries to Γ or S . An l block message query to C or RO will cost l queries. The reason for this different cost will become clear in the remainder of the proof.

and consistency with the random oracle RO . Handling the self consistency can be done efficiently with the list of answers, however consistency with the random oracle is a bit more tricky. The following definition will capture the essence of maintaining consistency with the random oracle.

Definition 2 (Chains). A chain is a triplet (M, h_f, h_g) , where M is a k block message and h_f, h_g are hash values. In addition we require that

$$f(f(\dots f(IV, m_1), m_2), \dots), m_k) = h_f$$

$$g(g(\dots g(IV, m_1), m_2), \dots), m_k) = h_g$$

and all the intermediate links are defined in the list of known values (i.e., have been queried previously).

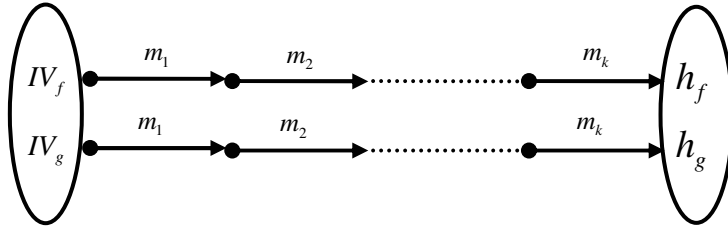


Fig. 1. Chains in the concatenated hash combiner

The chains create a tree structure, with the triplet (\perp, IV_f, IV_g) at the root. An edge between (M, h_f, h_g) and $(M \| m_{k+1}, h'_f, h'_g)$ corresponds to a pair of queries, linking h_f to h'_f and h_g to h'_g with the same message block m_{k+1} . Each node/chain in the tree corresponds to a constraint $h_f \oplus h_g = RO(M)$. The fact that with overwhelming probability the chains form a tree rather than a general graph structure will be proven later. To maintain consistency with the random oracle RO , our naive S will examine each new query and check if answering it will create a chain. If the response creates a chain, S will return a value consistent with RO . As stated, however, this task may require exponential time. Let us assume that the adversary uses a small number of calls to f and f^* in order to create an exponential size multicollision in F . When receiving a new g query, S must check exponentially many possible messages for G as there are that many messages with known chaining values for F . To overcome this problem the simulator will maintain three data structures in order to perform its operation. The first two structures T_f and T_g will contain explicit lists of the triplet answers given by S so far. The third structure will hold the tree of chains created so far. Notice that while the chain tree is implied from the first two lists, keeping it explicitly allows the simulator to run in polynomial time.

We will show how S updates these structures after each query and uses them in order to give consistent answers. For each forward query to f or g , S checks whether the value is already defined in the corresponding data structure of triplets, and if so returns the same value; if not, it returns a random value. To check if the value is defined, S checks if the query appears in its list of responses and additionally checks if the query completes a chain, i.e., extends the chain tree. If the query completes a chain with message M , S queries $RO(M)$ and uses the answer to give a consistent answer to the query. Notice that although chains might be created by bridging or backward queries, we will show that this will only happen with negligible probability and thus we can ignore these possibilities. In fact, we will show that with very high probability the chain tree does not contain any hash value more than once. I.e., the combined list of all x 's and z 's in the chain tree does

not contain duplicates. Our main lemma will show that with high probability, the above holds and chains are only created through forward queries. This in turn will imply that the answers S gives are consistent with the random oracle RO .

For backward and bridging queries, S also needs to reproduce the preimage distribution of Γ . In normal practice, m is significantly larger than n and therefore, returning a random value for bridging queries will reproduce the expected preimage distribution with respect to bridging queries. However, for backward queries⁶, we need to reproduce a Poisson distribution on the number of preimages. To this effect, S will keep together with each triplet, an integer j that represents the number of answers to the query $(?, y, z)$. Whenever a triplet containing the pair (y, z) is created for the first time, S generates j according to a Poisson distribution. If on a backward query $j = 0$, S returns the triplet (\perp, y, z) . For forward and bridging queries, j is generated according to a Poisson distribution conditioned on the output being non-zero. In future backward queries, S will return a uniform answer from the j possible answers. If one of the j possible answers is not defined yet, S will simply return a random value.

The simulator S formally acts as follows:

Forward queries

On input $(x, y, ?)$:

1. Check if there exists a triplet (x', y', z') in the same⁷ list and return that triplet if it exists.
2. If no such triplet exists, generate an integer j with Poisson distribution conditioned on being non-zero.
3. Check whether the query extends the chain tree.
4. If it does, query $RO(M)$ where M is the message corresponding to the new chain, and return the answer compatible with $RO(M)$.
5. Update the chain tree.
6. If no such chain is found, return a uniformly distributed answer.
7. In any case update the list of triplets with the answer and memorize the generated j .

Backward queries

On input $(?, y, z)$:

1. Check if there exists a triplet (x', y', z') in the same list with $(y, z) = (y', z')$.
2. If no such triplet exists, generate an integer j with Poisson distribution.
3. Choose uniformly from the j possible answers (some may not be defined yet).
4. If the chosen answer is not defined, generate a uniform answer x .
5. If $j = 0$, set $x = \perp$.
6. In any case (even if $j = 0$) update the list of triplets with the answer and memorize the generated j .

Bridging queries

On input $(x, ?, z)$:

1. Generate a random y .
2. Generate an integer j with Poisson distribution conditioned on being non-zero.
3. Update the list of triplets with the answer and memorize the generated j .

3.2 The Indifferentiability Proof

Our hybrid argument will have five settings. In the first setting, we simply have the pair (RO, S^{RO}) . In the second setting, we have the pair (R^{RO}, S^{RO}) where R simply relays the

⁶ The same special treatment given to backward queries can be given to bridging queries as well when m is not significantly larger than n .

⁷ I.e., T_f for f queries and T_G for g queries.

queries it receives to RO and answers with the responses it gets from RO . Since the view of any distinguisher D is identical with both pairs, we clearly have that

$$\Pr[D^{RO, S^{RO}} = 1] = \Pr[D^{R^{RO}}, S^{RO} = 1]$$

In the third setting, we have the pair (R^{RO}, S_1^{RO}) in which we slightly change the simulator S to S_1 such that when certain *unexpected events* occur, S_1 explicitly fails. Whenever an *unexpected event* occurs, S_1 fails explicitly, otherwise S_1 behaves exactly as S does.

Definition 3 (Unexpected events). *Let an unexpected event be the event that during an S query one of the following occurs:*

- U_1 *During a forward query the answer triplet (x, y, z) is such that there exists a triplet (x', y', z') ⁸ in one of the lists, $(x, y) \neq (x', y')$ and either $z = z'$, $z = x'$ or $z = IV$.*
- U_2 *During a backward query the answer triplet (x, y, z) is such that there exists a triplet (x', y', z') in one of the lists, $(y, z) \neq (y', z')$ and either $x = x'$, or $z = IV$.*
- U_3 *During a bridging query the answer triplet (x, y, z) is such that there exists a triplet (x', y', z') in one of the lists and $y = y'$.*

Lemma 1. *For any distinguisher D , the probability over the random coins of S and the random oracle RO that one of the unexpected events occurs is $\mathcal{O}\left(\frac{q^2}{2^n}\right)$.*

Proof. We will prove that the probability of an unexpected event occurring at query number i , conditioned on the event that so far no unexpected events have occurred is $\mathcal{O}\left(\frac{q}{2^n}\right)$. Using the union bound over all q queries we then get that the probability of the unexpected event is $\mathcal{O}\left(\frac{q^2}{2^n}\right)$.

We will examine each of the three possible unexpected events and bound their probability. We first analyze what happens if during the query, no chain is completed. In this case, for forward queries the answer triplet has a uniformly distributed z and therefore the probability of $z = z'$ or $z = x'$ for any of the existing x', z' in the respective list is bounded by $\frac{2q}{2^n}$. For bridging queries, the answer triplet has a uniformly distributed y and therefore the probability of $y = y'$ for any of the existing y' in the co-respecting list is bounded by $\frac{2q}{2^n}$. For backward queries the answer triplet has a uniformly distributed x and therefore the probability of $x = x'$ or $x = z'$ for any of the existing x', z' in the respective list is bounded by $\frac{q}{2^n}$.

We now examine the case in which the query completes a chain. For a backward and bridging queries the simulator's answer does not depend on the fact that a chain has been completed and therefore the probability of an unexpected event is the same as before. For forward queries, the response of the simulator is fully determined by $RO(M)$. However, the value of $RO(M)$ is uniformly distributed and hence so is the simulator's answer. Therefore, also in this case the probability of U_1 occurring is at most $\frac{2q}{2^n}$. Concluding, we have that the probability of an unexpected event occurring conditioned that no such events have happened so far is $\mathcal{O}\left(\frac{q}{2^n}\right)$. A union bound over all the queries gives us the bound $\mathcal{O}\left(\frac{q^2}{2^n}\right)$ as required.⁹ \square

Lemma 2.

1. *If we condition on the event that no unexpected events occur, then for every distinguisher the view when interacting with the pair (R^{RO}, S^{RO}) is identical to view when interacting with the pair (R^{RO}, S_1^{RO})*

⁸ Throughout this definition, answer triplets of the form (\perp, y', z') are also considered.

⁹ Note that even though using the union bound is usually not tight, in this case we get the birthday bound which is indeed tight.

$$2. |Pr[D^{R^{RO}, S^{RO}} = 1] - Pr[D^{R^{RO}, S_1^{RO}} = 1]| = \mathcal{O}\left(\frac{g^2}{2^n}\right)$$

Proof. Unless an unexpected event occurs, S_1 behaves exactly the same as S . This proves the first part of the lemma. Putting this result together with the fact that the probability of an unexpected event is bounded by $\mathcal{O}\left(\frac{g^2}{2^n}\right)$, proves the second part as well. \square

Now we turn our attention to the fourth setting, in which we examine the pair $(R_1^{S_1}, S_1^{RO})$, where R_1 answers its RO queries on input M by using S_1 to calculate $C(M)$. I.e., R_1 queries S on all the required f and g queries. Notice that a single query to R_1 with an l -block message will result in l queries to S_1 , for this reason R_1 queries cost l times more than a S_1 query.

Lemma 3. *If no unexpected events occur, then chains are only created by forward queries.*

Proof. Notice that when a chain is created, the message M is already determined. With out loss of generality, let the query which completes the chain be a f, f^{-1} or f^* query. In this case, all the g triplets in the chain have already been made and in particular, M is defined. Now, if a chain were created using a bridging query f^* , then the answer triplet (x, y, z) is such that $y \in M$ (as it completes a chain) and in particular y appears in a triplet in T_g , implying that the unexpected event U_2 occurred. If the chain were created using a backward query f^{-1} , then as the answer query (x, y, z) completed a chain, we know that x appears in a triplet in the T_f list or $x = IV$. Since (x, y, z) did not appear in T_f prior to the query (otherwise the chain would have been completed before) this implies that the unexpected event U_3 has occurred. Therefore, if no unexpected events occur all chains are created by forward queries. \square

Corollary 1. *If no unexpected events occur, the chain data structure is a tree containing all chains.*

Proof. If a forward call creates a cycle in the chain data structure, then unexpected event U_1 occurs. Hence, the chain data structure is a tree. Notice that if more than one chain is created during a forward call, then unexpected event U_1 has occurred previously (as there are two identical nodes in the chain tree). Therefore, at most a single chain is created during each forward call and the simulator tracks them correctly. \square

Lemma 4. *Unless an unexpected event occurs, then for every distinguisher the view when interacting with the pair (R^{RO}, S_1^{RO}) is indistinguishable from the view when interacting with the pair $(R_1^{S_1}, S_1^{RO})$.*

Proof. The proof will demonstrate the following three points:

1. Unless an unexpected event occurs when interacting with the pair (R^{RO}, S_1^{RO}) , the answers given by S_1 are consistent with those given by R^{RO} .
2. Unless an unexpected event occurs when interacting with the pair $(R_1^{S_1}, S_1^{RO})$, the answers given by S_1 are consistent with those given by $R_1^{S_1}$.
3. Unless an unexpected event occurs when interacting either with the pair (R^{RO}, S_1^{RO}) or with the pair $(R_1^{S_1}, S_1^{RO})$, the answers given by R^{RO} are exactly the same as those given by $R_1^{S_1}$.

Proof of point 1 Notice that from Lemma 3 we know that chains are only completed by forward queries. This implies that the simulator's answers are consistent with the value $RO(M)$ for any message M . Since $R^{RO}(M)$ simply replies with $RO(M)$, the answers given by both oracles are consistent.

Proof of point 2 The proof is similar to the proof of the previous point. The simulator's answers are always consistent with the value $RO(M)$ for any message M and $R_1^{S_1}(M) = RO(M)$ since the behavior of S_1 ensures this result.

Proof of point 3 This point is obvious since $R^{RO}(M) = RO(M)$ and also $R_1^{S_1}(M) = RO(M)$.

It now follows that unless an unexpected event occurs, the views generated by any distinguisher's interaction with the pairs (R^{RO}, S_1^{RO}) and $(R_1^{S_1}, S_1^{RO})$ are indistinguishable. \square

We are now ready for the proof of our main theorem:

Theorem 1. *The construction $C(M) = F(M) \oplus G(M)$, where F, G are iterated hash functions based on the compression function f and g respectively, is indistinguishable in $q \ll 2^{n/2}$ queries from a random oracle even in the presence of f^{-1}, f^*, g^{-1} and g^* oracles.*

Proof. Let S be the simulator defined above and let Γ be an oracle encapsulating $f, g, f^{-1}, f^*, g^{-1}$ and g^* . We will prove that for any distinguisher D

$$|Pr[D^{C,\Gamma} = 1] - Pr[D^{RO,S^{RO}} = 1]| = \mathcal{O}\left(\frac{q^2}{2^n}\right)$$

The lemmas so far have shown that $|Pr[D^{R_1^{S_1}, S_1} = 1] - Pr[D^{RO,S^{RO}} = 1]| = \mathcal{O}\left(\frac{q^2}{2^n}\right)$ and that S can be implemented in time polynomial in the number of queries q . It remains to show that for any possible distinguisher the pairs $(R_1^{S_1}, S_1^{RO})$ and (C, Γ) are indistinguishable. Notice however that unless an unexpected event occurs, S exactly simulates Γ and R_1^S exactly computes C . This completes the proof. \square

We have shown that the construction $C(M) = F(M) \oplus G(M)$ (or any n -bit function of F and G which is uniquely invertible when its output and any one of its input parameters is known) is indistinguishable in $q \ll 2^{n/2}$ queries from a random oracle even in the presence of f^{-1}, f^*, g^{-1} and g^* oracles and hence finding collisions in $H(M) = F(M) \parallel G(M)$ requires $\mathcal{O}\left(2^{n/2}\right)$ queries, matching the known upper bound of Joux. Notice that the construction $C(M) = F(M) \oplus G(M)$ requires the same amount of underlying function calls as the zipper hash of Liskov, albeit having a larger internal state, while having the advantage of being streamable.

3.3 Comments

Note that even though we have proved a lower bound on the number of calls to the compression functions and hence on the running time of a collision finding attack, this does not give a corresponding lower bound on the amount of memory required for the attack. In fact we can use Pollard's rho algorithm to find such a collision using only a linear amount of memory. Let $M_1^{0,1}M_2^{0,1}\dots M_n^{0,1}$ and $N_1^{0,1}N_2^{0,1}\dots N_n^{0,1}$ be Joux multicollisions for F and G respectively. We define two functions r_1, r_2 s.t. $r_1(x) = F(N_1^{x_1}N_2^{x_2}\dots N_n^{x_n})$ and $r_2(x) = G(M_1^{x_1}M_2^{x_2}\dots M_n^{x_n})$. We now use the rho algorithm to find a cycle in the path generated by iteratively alternating between applications of r_1 and r_2 . The memory complexity is $\mathcal{O}(n)$ while the time complexity is $\mathcal{O}(n2^{\frac{n}{2}})$.

4 Application to Lucks' Double Pipe Proposal

The same proof framework can be used to prove other indistinguishability results. For example, the double pipe hash from [12] can also be proved indistinguishable from a random oracle in the model of weak compression functions. Given a compression function f , the double pipe hash has a $2n$ bit internal state (r, s) and is defined as follows:

- Split a message M into k blocks each of size $(m - n)$ bits, x_1, \dots, x_k .
- Set $r_0 = IV_1$, $s_0 = IV_2$ where IV_1 and IV_2 are the initialization vectors.
- For each message block x_i compute $r_i = f(r_{i-1}, s_{i-1} \| x_i)$ and $s_i = f(s_{i-1}, r_{i-1} \| x_i)$.
- Output $DP^f(M) = f(IV_3, r_k \| s_k \| 0^{m-2n})$.

The double pipe hash is schematically described in Figure 2.

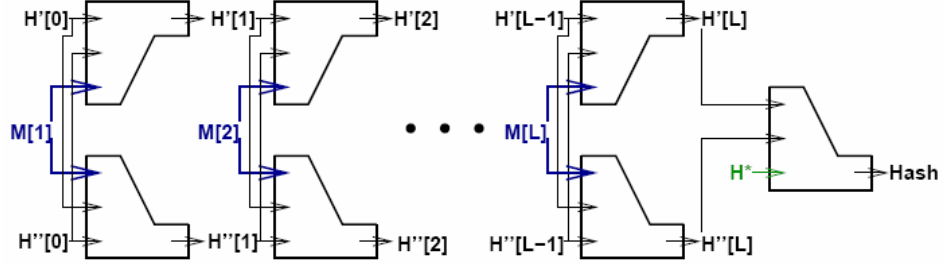


Fig. 2. Lucks' double pipe hash (taken from [12])

Note that Lucks proved that the double pipe hash is not vulnerable to multicollision or multi-(second)-preimage attacks when the underlying compression function is modeled as a random oracle (or ideal cipher) which has no weaknesses, while Liskov [11] claimed (without proof) that the construction is indistinguishable from a random oracle if the two pipes use two unrelated weak compression functions f and g . We will prove that the original construction is indistinguishable from a random oracle even when the same function is used in both pipes, and it is weak in the sense that the attacker is given both inversion and bridging oracles. Our proof will also hold if the final hash is replaced by a xor operation, or any function which is uniquely invertible when its output and any one of its input parameters are known.

The proof outline is identical to the one presented in Section 3; we will therefore only give the main lemmas required. We start by giving an adequate definition of *chains* in the double pipe hash, that following the example in section 3 captures the essence of consistency between the simulator S and the random oracle RO .

Definition 4 (Double pipe hash chains). A (double pipe hash) chain is a triplet M, h_1, h_2 , where M is a k block message and h_1 and h_2 are hash values. In addition we require that

$$f(f(\dots f(IV_1, s_1 \| m_1), s_2 \| m_2), \dots), s_k \| m_k) = h_1$$

$$f(f(\dots f(IV_2, r_1 \| m_1), r_2 \| m_2), \dots), r_k \| m_k) = h_2$$

where r_i is the chaining value of the upper pipe after the first i blocks and s_i is the chaining value of the lower pipe after the same i blocks. We additionally require that all the intermediate links are defined in the list of known values (i.e., have been queried previously).

The simulator will be identical to the one introduced in Section 3 with the following changes: We will change the simulator's behavior when a chain is completed with message M . Without loss of generality, assume that the query which completed the chain is in the lower pipe. The simulator computes the value $z = RO(M)$, generates a random value d , sets the triplet $(IV_3, r_k \| d \| 0^{m-2n}, z)$ and returns d as the response to the query. The unexpected events will now become:

Definition 5 (Double pipe unexpected events). Let an unexpected event be the event that during a S query one of the following occurs:

- V_1 During a forward query the answer triplet (x, y, z) is such that there exists a triplet (x', y', z') , $(x, y) \neq (x', y')$ and either $z = z'$, $z = x'$ or $z = IV$.
- V_1^* During a forward query a chain is completed and the random value d generated is such that there exists a triplet (x', y', z') , and $y = r_k \| d \| 0^{m-2n}, z$.
- V_2 During a backward query the answer triplet (x, y, z) is such that there exists a triplet (x', y', z') , $(y, z) \neq (y', z')$ and either $x = x'$, or $z = IV$.
- V_3 During a bridging query the answer triplet (x, y, z) is such that there exists a triplet (x', y', z') and $y = y'$.

Lemma 5. *The probability over the random coins of S and the random oracle RO , that an unexpected event occurs is $\mathcal{O}\left(\frac{q^2}{2^n}\right)$.*

Proof. As before, we will prove that the probability of an unexpected event occurring, conditioned on the event that so far no unexpected events have occurred is $\mathcal{O}\left(\frac{q}{2^n}\right)$. The proof for events V_1, V_2 and V_3 is identical to the proof for the corresponding events U_1, U_2 and U_3 . It remains to bound the probability of V_1^* . However, since d is completely uniform, we have that the probability is at most $\frac{q}{2^n}$. Using a union bound over all queries gives us the required bound of $\mathcal{O}\left(\frac{q^2}{2^n}\right)$. \square

As in the proof in Section 3, the main lemma will show that unless an unexpected event occurs, chains are only created during forward queries.

Lemma 6. *If no unexpected events occur, then chains are only created by forward queries.*

Proof. As before, notice that when a chain is created, the message M is already determined. Now, if a chain were created using a bridging query f^* , then the answer triplet (x, y, z) is such that $y \in M$ (as it completes a chain) and in particular y appears in an existing triplet, implying that the unexpected event V_2 occurred. If the chain were created using a backward query f^{-1} , then as the answer query (x, y, z) completed a chain, we know that x appears in an existing triplet or $x = IV$. Since (x, y, z) did not appear in the list of triplets prior to the query (otherwise the chain would have been completed before) this implies that the unexpected event V_3 has occurred. Therefore, if no unexpected events occur all chains are created by forward queries. \square

Theorem 2. *The double pipe hash construction is indifferentiable from a random oracle in the model of weak compression functions.*

Proof. The sequence of hybrids is the same as in the proof in Section 3 and culminates with the required result. \square

5 Conclusion

While the results of Joux [7], Kelsey and Schneier [9] and Kelsey and Kohno [8] have shown that there are a number of surprising attacks when the attacker is allowed more than $2^{n/2}$ time, we have shown that there is a surprising amount of ‘life’ below the $2^{n/2}$ barrier: Even an adversary with the power to invert compression functions on inputs of his choice in unit time is still unable to differentiate between a variety of hash constructions and a random oracle. It seems that there are two main issues at the heart of our results. The first is the assumed randomness of the compression function, which implies that with less than $2^{n/2}$ queries it is not feasible to use in an effective way the given inversion oracles. The second issue is the fact that during the simulation the simulator needs to maintain consistency with the random oracle. In order to do this, the simulator must somehow ‘know’ when the queries given so far define some final hash value. In all the examples we gave as well as in the zipper hash[11] of Liskov, the construction of the combined hash function is such that with overwhelming probability the simulator can always tell when a query determines the output of the hash.

References

1. D. Boneh and X. Boyen. On the impossibility of efficiently combining collision resistant hash functions. In *Advances in Cryptology—CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 570–583. Berlin: Springer-Verlag, 2006. Available at <http://www.cs.stanford.edu/~xb/crypto06b/>.
2. G. Brassard, editor. *CRYPTO '89, Santa Barbara, California, USA, August 0-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.
3. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-damgård revisited: How to construct a hash function. In *CRYPTO'05*, pages 430–448, 2005.
4. R. Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
5. I. Damgård. A Design Principle for Hash Functions. In Brassard [2], pages 416–427.
6. M. Fischlin and A. Lehmann. Security-amplifying combiners for collision-resistant hash functions. In A. Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 224–243. Springer, 2007.
7. A. Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In M. K. Franklin, editor, *CRYPTO'04*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.
8. J. Kelsey and T. Kohno. Herding Hash Functions and the Nostradamus Attack. In S. Vaudenay, editor, *EUROCRYPT'06*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.
9. J. Kelsey and B. Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In Cramer [4], pages 474–490.
10. V. Klima. Tunnels in Hash Functions: MD5 Collisions Within a Minute. Cryptology ePrint Archive, Report 2006/105, 2006. <http://eprint.iacr.org/>.
11. M. Liskov. Constructing an ideal hash function from weak ideal compression functions. In *Selected Areas in Cryptography*, pages 358–375, 2006.
12. S. Lucks. A Failure-Friendly Design Principle for Hash Functions. In B. K. Roy, editor, *ASIACRYPT'05*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005.
13. U. M. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In M. Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.
14. R. C. Merkle. One Way Hash Functions and DES. In Brassard [2], pages 428–446.
15. K. Pietrzak. Non-trivial black-box combiners for collision-resistant hash-functions don't exist. In *EUROCRYPT*, pages 23–33, 2007.
16. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [4], pages 1–18.
17. X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In Cramer [4], pages 19–35.
18. X. Wang, H. Yu, and Y. L. Yin. Efficient Collision Search Attacks on SHA-0. In V. Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.
19. H. Yu, G. Wang, G. Zhang, and X. Wang. The Second-Preimage Attack on MD4. In Y. Desmedt, H. Wang, Y. Mu, and Y. Li, editors, *CANS*, volume 3810 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2005.