Indifferentiability of Permutation-Based Compression Functions and Tree-Based Modes of Operation, with Applications to MD6

Yevgeniy Dodis[1], Leonid Reyzin[2], Ronald L. Rivest[3], and Emily Shen[3]

[1] New York University
dodis@cs.nyu.edu
[2] Boston University
reyzin@cs.bu.edu
[3] Massachusetts Institute of Technology
{rivest,eshen}@csail.mit.edu

Abstract. MD6 [16] is one of the earliest announced SHA-3 candidates, presented by Rivest at CRYPTO'08 [15]. Since then, MD6 has received a fair share of attention and has resisted several initial cryptanalytic attempts [1, 10].

Given the interest in MD6, it is important to formally verify the soundness of its design from a theoretical standpoint. In this paper, we do so in two ways: once for the MD6 compression function and once for the MD6 mode of operation. Both proofs are based on the indifferentiability framework of Maurer et al. [12] (also see [8]).

The first proof demonstrates that the "prepend/map/chop" manner in which the MD6 compression function is constructed yields a compression function that is indifferentiable from a fixed-input-length (FIL), fixed-output-length random oracle.

The second proof demonstrates that the tree-based manner in which the MD6 mode of operation is defined yields a hash function that is indifferentiable from a variable-input-length (VIL), fixed-output-length random oracle.

Both proofs are rather general and apply not only to MD6 but also to other sufficiently similar hash functions.

These results may be interpreted as saying that the MD6 design has no structural flaws that make its input/output behavior clearly distinguishable from that of a VIL random oracle, even for an adversary who has access to inner components of the hash function. It follows that, under plausible assumptions about those inner components, the MD6 hash function may be safely plugged into any application proven secure assuming a monolithic VIL random oracle.

1 Introduction

In light of recent devastating attacks on existing hash functions, such as MD4 [17, 21], MD5 [19], SHA-0 [20, 6], and SHA-1 [18], NIST recently announced a competition for a new hash function standard, to be called SHA-3 [13]. NIST received

64 submissions, one of which is MD6. The current status of the SHA-3 competition can be found on the NIST web site[1].

Crutchfield [9] showed that MD6 has many attractive properties required of a good hash function, such as preservation of collision-resistance, unpredictability, preimage-resistance, and pseudorandomness. As observed by Coron et al. [8], however, the above "traditional" properties of hash functions are often insufficient for many applications, which require that the hash function behaves "like a random oracle." Moreover, the initial NIST announcement for the SHA-3 competition [14] states that the candidate submissions will be judged in part by

> "The extent to which the algorithm output is indistinguishable from a random oracle."

Thus, it is important to show that the design of the hash function is "consistent" with the proofs of security in the random oracle (RO) model [3]. Such a notion of consistency with the random oracle model was recently defined by Coron et al. [8]; it is called *indifferentiability from a random oracle* and is based on the general indifferentiability framework of Maurer et al. [12].

Given the importance of the random oracle model in the design of practical cryptographic schemes and the increased popularity of the indifferentiability framework in the analysis of hash functions [8, 2, 7, 11, 4], we suggest that it is critical that the winner of the SHA-3 competition satisfy such an "indifferentiability from a random oracle" property.

The main result of this paper is a formal proof that the design of MD6, both at the compression function level and at the mode of operation level, provides indifferentiability from a random oracle. Thus, the MD6 mode of operation and compression function have no structural flaws that would allow them to be distinguished from (VIL or FIL, respectively) random oracles. It follows, from results due to Maurer et al. [12], that, given reasonable assumptions about the permutation inside the compression function, the MD6 hash function may be safely plugged into any higher-level application whose security is proven assuming the hash function is a VIL random oracle.

These results generalize to other hash functions built on permutation-based compression functions and tree-based modes of operations, if they are sufficiently similar to MD6 in structure to meet the conditions of our proofs.

To explain our results more precisely, we briefly recall the indifferentiability framework and the high level design of the MD6 hash function.

1.1 Indifferentiability

The notion of indifferentiability was first introduced by Maurer et al. in [12]. Informally, it gives sufficient conditions under which a primitive F can be "safely replaced" by some construction C^G (using an ideal primitive G). The formal definition is as follows.

[1] http://csrc.nist.gov/groups/ST/hash/sha-3/index.html

Definition 1. *A Turing machine C with oracle access to an ideal primitive G is* (t, q_F, q_S, ϵ)-indifferentiable *from an ideal primitive F if there exists a simulator S such that, for any distinguisher D, it holds that:*

$$\left|\Pr\left[D^{C,G} = 1\right] - \Pr\left[D^{F,S} = 1\right]\right| < \epsilon$$

The simulator S has oracle access to F (but does not see the queries of the distinguisher D to F) and runs in time at most t. The distinguisher makes at most q_F queries to C or F and at most q_S queries to G or S.

Indifferentiability is a powerful notion; Maurer et al. [12] show that if C^G is indifferentiable from F, then F may be replaced by C^G in any cryptosystem, and the resulting cryptosystem is at least as secure in the G model as in the F model.

In this paper, F will always be a random oracle — either fixed-input-length, fixed-output-length or variable-input-length, fixed-output-length. Thus, we will be showing that a certain construction C is indifferentiable from a random oracle, meaning that any cryptosystem proven secure in the RO model will still be secure in the G model, when the hash function is implemented using C^G.

1.2 The MD6 Hash Function: High-Level View

The MD6 function consists of the following two high-level steps. First, there is a *compression function* which operates on fixed-length inputs. Second, there is a *mode of operation*, which uses the compression functions as a black-box and to evaluate the hash function on arbitrary-length inputs. Naturally, our indifferentiability results will consist of two parts as well: (a) that the compression function, under some natural assumptions, is indifferentiable from a fixed-input-length RO, and (b) that, assuming the compression function is a fixed-input-length RO, the MD6 mode of operation yields a hash function that is indifferentiable from a variable-input-length RO.

The Compression Function. The MD6 compression function f maps an input N of length $n = 89$ words (consisting of 25 words of auxiliary input followed by 64 words of data) to an output of $c = 16$ words. Thus, the compression function reduces the length of its data input by a factor of 4. The compression function f is computed as a series of operations, which we view as the application of a random permutation π over the set of 89-word strings, followed by a truncation operation which returns the last 16 words of $\pi(N)$.

In each call to f in an MD6 computation, the first 15 words of auxiliary input are a constant Q (a representation of the fractional part of $\sqrt{6}$). Therefore, in our analysis, we consider the "reduced" compression function f_Q, where $f_Q(x) = f(Q||x)$. For the full specification of the MD6 compression function, we refer the reader to [16].

In Section 3, we prove that f_Q is indifferentiable from a random oracle F, assuming that the main operation of the compression function is the application of a fixed public random permutation π.

The Mode of Operation. The standard MD6 mode of operation is a hierarchical, tree-based construction to allow for parallelism. However, for devices with limited storage, the MD6 mode of operation can be iterative. There is an optional level parameter L which allows a smooth transition between the fully hierarchial mode of operation and the fully iterative mode of operation. At each node in the tree, the input to the compression function includes auxiliary information. The auxiliary information includes a unique identifier U for each node (consisting of a tree level and index). In addition, there is a bit z which equals 1 in the input to the root node (the final compression call) and 0 in the input to all other nodes. For the full specification of the MD6 mode of operation, we refer the reader to [16].

In Section 4, we prove that the MD6 mode of operation is indifferentiable from a random oracle F when the compression function f_Q is modeled as a random oracle. In fact, our proof is quite general and applies essentially to any tree-like construction, as long as the final computation node has a distinguishable input structure and compression function inputs are uniquely parsable into blocks that are either raw message bits, metadata, or outputs of the compression function on "child" nodes.

2 Notation

We first introduce some notation.

Let $\mathbf{W} = \{0,1\}^w$ denote the set of all $w = 64$-bit words. Let $\chi_a(X)$ denote a function that returns the last a bits of X. Let $\gamma_a(X)$ denote a function that truncates X by dropping its last a bits.

3 Indifferentiability from Random Oracle of MD6 Compression Function

In this section, we will prove the indifferentiability from a random oracle of the MD6 compression function construction, under certain assumptions. The compression function construction involves three steps:

- *Prepending* a constant value Q to the compression function input.
- *Mapping* the result by applying a fixed (pseudo)-random permutation π to it, and
- *Chopping* (removing) bits off the front of the result, so that what remains has the desired length as a compression function output.

Our proof applies in general to compression functions constructed in this manner. Our presentation of the proof will use notation shared with the formal specification of MD6, for convenience.

We view the compression function as based on a fixed public random permutation $\pi(\cdot)$, i.e., $f(N) = \chi_{cw}(\pi(N))$. (Recall that $\chi_{cw}(\cdot)$ returns the last cw bits of its input.) Since the permutation is public, an adversary can compute both π

and π^{-1} easily. Therefore, we need to consider an adversarial model where the adversary has these powers.

Note that in this model the adversary can both invert f and find collisions for f easily, if we do not do something additional. (This is because the adversary can take the c-word output C, prepend $n-c$ words of random junk, then apply π^{-1} to get a valid pre-image for C. He can do this twice, with different junk values, to get a collision.) However, MD6 does have an important additional feature: a valid compression function input must begin with a fixed constant Q. We now proceed to show that this yields a compression function that behaves like a random oracle when π is a random permutation.

Recall that $\mathbf{W} = \{0,1\}^w$ denotes the set of all w-bit words and that f takes n-word inputs. We let $f_Q(x) = f(Q||x)$ denotes the "reduced" compression function that takes $(n-q)$-word inputs, prepends the fixed prefix Q, and runs f. To make it explicit that in this section we are modeling f and f_Q in terms of a random permutation π on $\mathbf{W}^n$, we will write

$$f_Q^{\pi}(x) = \chi_{cw}(\pi(Q||x))\,, \tag{1}$$

where $\chi_{cw}(y)$ returns the last cw bits of y, and where x is in $\mathbf{W}^{n-q}$.

Let the ideal functionality be represented by $F : \mathbf{W}^{n-q} \to \mathbf{W}^c$, a random oracle with same signature as f_Q^{π}. We will show that f_Q^{π} is indifferentiable from F, as stated below.

Theorem 1. *If π is a random permutation and Q is arbitrary, the reduced MD6 compression function f_Q^{π} defined by equation (1) is (t, q_F, q_S, ϵ)-indifferentiable from a random oracle F, for any number of queries q_F and q_S, for distinguishing advantage*

$$\epsilon = \frac{(q_S+q_F)^2}{2^{nw}} + \frac{q_S}{2^{qw}} + \frac{q_S q_F}{2^{(n-c)w}}, \tag{2}$$

and for running time of the simulator $t = O(q_S nw)$.

Proof. We use the approach of Coron et al. [8], who showed that the indifferentiability framework can be successfully applied to the analysis of hash functions built from simpler primitives (such as block ciphers or compression functions). We note that related results have been obtained by Bertoni et al. [5] in their proof of indifferentiability of "sponge functions."

In our case, because π is a permutation, the oracle G contains both π and π^{-1}, and we need to simulate them both. Slightly abusing notation, we will write S for the simulator of π and S^{-1} for the simulator of π^{-1}. Thus, we need to construct simulator programs S and S^{-1} for π and π^{-1} such that no distinguisher D can distinguish (except with negligible probability) between the following two scenarios:

(A) The distinguisher has oracle access to f_Q^{π}, to π, and to π^{-1}.
(B) The distinguisher has oracle access to F, S, and S^{-1}.

We define the simulators S, S^{-1} for π, π^{-1} as follows:

1. S and S^{-1} always act consistently with each other and with previous calls, if possible. If not possible (i.e., there are multiple answers for a given query), they abort.
2. To evaluate $S(X)$ where $X = Q||x$, compute $y = F(x)$, then return $R||y$ where R is chosen randomly from in $\mathbf{W}^{n-c}$.
3. To evaluate $S(X)$ where X does not start with Q, return a value R chosen randomly from $\mathbf{W}^n$.
4. To evaluate $S^{-1}(Y)$: return a random N in $\mathbf{W}^n$ which does not start with Q (i.e., from $\mathbf{W}^n \backslash (Q||\mathbf{W}^{n-q})$).

The running time of the simulators is at most $t = O(q_S nw)$. Next, we argue the indifferentiability of our construction. To this end, consider any distinguisher D making at most q_S to queries to S/π and S^{-1}/π^{-1} and at most q_F queries to F/f_Q^π. To analyze the advantage of this distinguisher, we consider several games $G_0, G_1, \ldots, G_7$. For each game G_i below, let $p_i = \Pr(D \text{ outputs 1 in } G_i)$. Intuitively, G_0 will be the "real" game, G_7 will be the "ideal" game, and the intermediate game will slowly transform these games into each other.

Game G_0. This is the interaction of D with f_Q^π, π, π^{-1}.

Game G_1. The game is identical to G_0 except the permutation π is chosen in a "lazy" manner. Namely, we introduce a *controller* C_π which maintains a table T_π consisting of all currently defined values (X, Y) such that $\pi(X) = Y$. Initially, this table is empty. Then, whenever a value $\pi(X)$ or $\pi^{-1}(Y)$ is needed, C_π first checks in T_π whether the corresponding value is already defined. If yes, it supplies it consistently. Else, it chooses the corresponding value at random subject to the "permutation constraint". Namely, if $T_\pi = \{(X_i, Y_i)\}$, then $\pi(X)$ is drawn uniformly from $\mathbf{W}^n \backslash \{Y_i\}$ and $\pi^{-1}(Y)$ is drawn uniformly from $\mathbf{W}^n \backslash \{X_i\}$. It is clear that G_1 is simply a syntactic rewriting of G_0. Thus, $p_1 = p_0$.

Game G_2. This game is identical to G_1 except the controller C_π does not make an effort to respect the permutation constraint above. Instead, it simply chooses undefined values $\pi(X)$ and $\pi^{-1}(Y)$ completely at random from $\mathbf{W}^n$, but explicitly aborts the game in case the permutation constraint is not satisfied. It is clear that $|p_2 - p_1|$ is at most the probability of such an abort, which, in turn, is at most $(q_S + q_F)^2/2^{nw}$.

Game G_3. This game is identical to G_2 except the controller C_π does not choose values starting with Q when answering the new inverse queries $\pi^{-1}(Y)$. Namely, instead of choosing such queries at random from $\mathbf{W}^n$, it chooses them at random from $\mathbf{W}^n \backslash (Q||\mathbf{W}^{n-q})$. It is easy to see that $|p_3 - p_2|$ is at most the probability that C_π would choose an inverse starting with Q in the game G_2, which is at most $q_S/2^{qw}$.

Game G_4. This game is identical to G_3 except we modify the controller C_π as follows. Notice that there are three possible ways in which C_π would add an extra entry to the table T_π:

1. D makes a query $\pi(X)$ to π, in which case a new value (X, Y) might be added (for random Y). We call such additions *forward*.

2. D makes a query $\pi^{-1}(Y)$ to π^{-1}, in which case a new value (X, Y) is added (for random X not starting with Q). We call such additions *backward*.
3. D makes a query $f_Q^{\pi}(x) = \chi_{cw}(\pi(Q||x))$, in which case C_π needs to evaluate $\pi(Q||x)$ and add a value $(Q||x, Y)$ (for random Y). We call such additions *forced*.

We start by making a syntactic change. When a forced addition $(Q||x, Y)$ to T_π is made, C_π will mark it with a special symbol and will call this entry *marked*. C_π will keep it marked until D asks the usual forward query to $\pi(Q||x)$, in which case the entry will become *unmarked*, just like all the regular forward and backward additions to T_π. With this syntactic addition, we can now make a key semantic change in the behavior of the controller C_π.

– In game G_3, when a backward query $\pi^{-1}(Y)$ is made, C_π scans the entire table T_π to see if an entry of the form (X, Y) is present. In the new game G_4, C_π will only scan the *unmarked* entries in T_π, completely ignoring the currently marked entries.

We can see that the only way the distinguisher D will notice a difference between G_3 and G_4 is if D can produce a backward query $\pi^{-1}(Y)$ such that the current table T_π contains a *marked* entry of the form $(Q||x, Y)$. Let us call this event E, and let us upper-bound the probability of E. For each forced addition $(Q||x, Y)$, the value Y is chosen at random from $\mathbf{W}^n$, and the distinguisher D only learns the "chopped" value $y = \chi_{cw}(Y)$. In other words, D does not see $(n-c)w$ completely random bits of Y. Thus, for any particular forced addition, the probability that D ever "guesses" these missing bits is $2^{-(n-c)w}$. Since D gets at most q_S attempts, and there are at most q_F forced values to guess, we get that $\Pr(E) \leq q_S q_F / 2^{(n-c)w}$. Thus, $|p_4 - p_3| \leq \frac{q_S q_F}{2^{(n-c)w}}$.

Game G_5. We introduce a new controller C_F, which is simply imitating a random function $F : \mathbf{W}^{n-q} \to \mathbf{W}^c$. Namely, C_F keeps a table T_F, initially empty. When a query x is made, C_F checks if there is an entry (x, y) in T_F. If so, it outputs y. Else, it picks y at random from $\mathbf{W}^c$, adds (x, y) to T_F, and outputs y. Now, we modify the behaviors of the controller C_π for π/π^{-1} from the game G_4 as follows. In game G_4, when a new forward query $(Q||x)$ was made to π, or a new query x was made to f_Q^{π}, C_π chose a random Y from $\mathbf{W}^n$ and set $\pi(Q||x) = Y$. In game G_5, in either one of these cases, C_π will send a query x to the controller C_F, get the answer y, and then set $Y = R||y$, where R is chosen at random from $\mathbf{W}^{n-c}$.

We notice that the game G_5 is simply a syntactic rewriting of the game G_4, since choosing a random value in $\mathbf{W}^n$ is equivalent to concatenating two random values in $\mathbf{W}^{n-c}$ and $\mathbf{W}^c$. Thus, $p_5 = p_4$.

Game G_6. Before describing this game, we make the following observations about the game G_5. First, we claim that all the entries of the form $(Q||x, Y)$ in T_π, whether marked or unmarked, have come from the explicit interaction with the controller C_F. Indeed, because in game G_3 we restricted C_π to never answer a backward query so that the answer starts with Q, all such entries in T

have come either from a forward query $\pi(Q||x)$, or the f_Q^π-query $f_Q^\pi(x)$. In either case, in game G_5 the controller C_π "consulted" C_F before making the answer. In fact, we can say more about the f_Q^π-query $f_Q^\pi(x)$. The answer to this query was simply the value y which C_F returned to C_π on input x. Moreover, because of the rules introduced in game G_4, C_π immediately marked the entry $(Q||x, R||y)$ which it added to T_π, and completely ignored this entry when answering the future backward queries to π^{-1} (until a query $\pi(Q||x)$ to π was made).

Thus, we will make the following change in the new game G_6. When D asks a new query $f_Q^\pi(x)$, the value x no longer goes to C_π (which would then attempt to define $\pi(Q||x)$ by consulting C_F). Instead, this query goes directly to C_F, and D is given the answer y. In particular, C_π will no longer need to mark any of the entries in T_π, since all the f_Q^π queries are now handled directly by C_F. More precisely, C_π will only "directly" define the forward queries $\pi(X)$ and the backward queries $\pi^{-1}(Y)$ (in the same way it did in Game G_5), but no longer define $\pi(Q||x)$ as a result of D's call to $f_Q^\pi(x)$.

We claim that game G_6 is, once again, only a syntactic rewriting of game G_5. Indeed, the only change between the two games is that, in game G_5, T_π will contain some marked entries $(Q||x, R||y)$, which will be ignored anyway in answering all the inverse queries, while in Game G_6 such entries will be simply absent. There is only one very minor subtlety. In Game G_5, if D first asks $f_Q^\pi(x)$, and later asks $\pi(Q||x)$, the latter answer $R||y$ will already be stored in T_π at the time of the first question $f_Q^\pi(x)$. However, it will be marked and ignored until the second question $\pi(Q||x)$ is made. In contrast, in Game G_6 this answer will only be stored in T_π after the second question. However, since in both cases C_π would answer by choosing a random R and concatenating it with C_F's answer y to x, this minor difference results in the same view for D. To sum up, $p_6 = p_5$.

Game G_7. This is our "ideal" game where D interacts with S/S^{-1} and a true random oracle F. We claim this interaction is identical to the one in Game G_6. Indeed, C_F is simply a "lazy" evaluation of the random oracle F. Also, after all our changes, the controller C_π in Game G_6 is *precisely* equivalent to our simulators S and S^{-1}. Thus, $p_7 = p_6$.

Collecting all the pieces together, we get that the advantage of D in distinguishing Game G_0 and Game G_7 is at most the claimed value

$$\epsilon \leq \frac{(q_S + q_F)^2}{2^{nw}} + \frac{q_S}{2^{qw}} + \frac{q_S q_F}{2^{(n-c)w}}$$

□

(In practice, there are other inputs to consider, such as the key input K, the unique ID U, and the control word V. The above proof applies as given, assuming that these inputs are available for the distinguisher to control. This is the correct assumption to make from the viewpoint of the MD6 mode of operation or other applications using the MD6 compression function.)

Remark 1. We remark that our indifferentiability proof for the function $f_Q^\pi(x) = \chi_{cw}(\pi(Q||x))$ trivially generalizes to any compression function $f^\pi(x)$ of the form $f^\pi(x) = h_{cw}(\pi(g_{qw}(x)))$, where:

– $h_{cw} : \mathbf{W}^n \to \mathbf{W}^c$ is any *regular*[2] function. For the case of MD6, we use the "chop function" $h_{cw} = \chi_{cw}$.
– $g_{qw} : \mathbf{W}^{n-q} \to \mathbf{W}^n$ is any *injective* function which is (a) efficiently *invertible* and (b) efficiently *verifiable* (i.e., one can determine whether or not a point $y \in \mathbf{W}^n$ belongs to the range of g_{qw}). For the case of MD6, we use the "prepend function" $g_{qw}(x) = (Q||x)$, for some constant $Q \in \mathbf{W}^q$.

4 Indifferentiability of Tree-Based Modes of Operation

In this section, we prove that any tree-based mode of operation, with certain properties (defined below), is indifferentiable from a random oracle when the compression function is a random oracle. (In fact, our result applies to modes of operation that can be described as straight-line programs, which are more general than trees.) We then derive the indifferentiability of the MD6 mode of operation as a consequence of this result.

Consider a compression function $\phi : \mathbf{W}^\eta \to \mathbf{W}^c$. Let $\mu^\phi : \{0,1\}^* \to \{0,1\}^d$ denote the mode of operation μ applied to ϕ. We will prove that, if ϕ is a random oracle and μ satisfies the required properties, then μ^ϕ is indifferentiable from a random oracle.

4.1 Required Properties of Mode of Operation

To prove indifferentiability of a mode of operation μ, we will require μ to have the following properties. These are properties of the mode μ itself, independent of the particular compression function ϕ.

Unique Parsing. Every compression function input $x \in \mathbf{W}^\eta$ that occurs in the computation of $\mu^\phi(M)$ for some $M \in \{0,1\}^*$ must be efficiently and uniquely parsable into a sequence of blocks (not necessarily of the same size), each of which is a compression function output, raw message bits (from the message being hashed), or metadata. Note that there may be parsable inputs that do not actually occur in the computation of $\mu^\phi(M)$ for any M.

Parent Predicate. Given the unique parsing property, we define a predicate *parent*(x, y, i) which, given oracle access to a compression function ϕ, takes two inputs $x, y \in \mathbf{W}^\eta$, and an index i. It outputs true iff $\phi(y)$ is equal to the ith compression function output in the parsing of x. We say that x is a parent of y, or equivalently, y is a child of x if *parent*(x, y, i) is true for some i. We also say that y is the ith child of x if *parent*(x, y, i) is true. Note that actual parent-child pairs occurring during the execution of μ may satisfy additional conditions: for example, x and y may contain consecutive level numbers. Our definition of *parent* does not verify these conditions, and thus may create a parent relationship where none should exist. These conditions (among others) will be verified by a function ρ defined below.

[2] A function is regular if every value in the range has an equal number of preimages in the domain.

Leaf Predicate. Given the unique parsing property, we also define a predicate *leaf*(x) on compression function inputs x which returns true iff the parsing of x contains only message bits and metadata but no compression function outputs. We say that x is a leaf if *leaf*(x) is true.

Root Predicate. There must be a well-defined, efficiently testable predicate *root*(x) such that: for any $M \in \{0,1\}^*$, for every non-final compression function call x in the computation of $\mu^\phi(M)$, *root*(x) is false, and for the final compression function call y, *root*(y) is true. For strings x that are neither non-final nor final compression function calls in the computation of $\mu^\phi(M)$ for any M, *root*(x) can be either true or false.

Note that this condition implies that the set of strings $x \in \mathbf{W}^\eta$ that are the final compression call for some M must be disjoint from the set of strings $y \in \mathbf{W}^\eta$ that are a non-final compression call for some M.

Straight-Line Program Structure. The mode of operation μ must be a straight-line program in the following sense. It carries out a sequence of calls to the compression function, where input to call number i in the sequence is computed from the message itself, metadata, and outputs of some j calls numbered $i_1, i_2, \ldots, i_j < i$. This sequence (that is, the exact dependence of ith call on the previous calls and the message) must be deterministically computable from M alone, regardless of ϕ. For every call in the sequence except the last one, its output value must be used to compute some other input value. Moreover, for any ϕ, the output values of calls $i_1, i_2, \ldots, i_j$ must occur in the parsing of the input to the ith call. The last call x in that sequence must have *root*(x) = true; for all the others, the root predicate must be false. Denote by $\Sigma(M)$ the set of all calls to ϕ (input and output pairs) during the computation of $\mu^\phi(M)$.

Final Output Processing. It must be the case that $\mu^\phi(M) = \zeta(\phi(x))$ where x is the final compression input, where $\zeta : \mathbf{W}^c \to \{0,1\}^d$ is an efficiently computable, regular function. The set of all preimages $\zeta^{-1}(h)$ of a value h must be efficiently sampleable given h.

Message Reconstruction. There must be an efficiently computable function ρ that takes a set Π of compression function calls and returns a message M if $\Pi = \Sigma(M)$, and $\perp$ otherwise. Because μ is deterministic, it follows that if $\rho(\Pi_1) = M_1$ and $\rho(\Pi_2) = M_2$ and $\Pi_1 \neq \Pi_2$, then $M_1 \neq M_2$.

We let $\kappa(\ell)$ denote an upper bound on the running time of ρ on an input set Π containing at most ℓ compression function calls. We assume that the other efficiently computable operations defined above (compression function input parsing, computing $\zeta(C)$, sampling from $\zeta^{-1}(h)$, and evaluating *root*(x)) run in constant time.

4.2 The Simulator

For the proof of indifferentiability of μ, we will define a polynomial-time simulator S_0 for the compression function ϕ. S_0 works as follows.

S_0 maintains a set T, initially empty, of pairs $(x, C) \in \mathbf{W}^\eta \times \mathbf{W}^c$ such that it has responded with C to a query x.

Upon receiving a compression function query x^*, S_0 searches its set T for a pair (x^*, C^*). If it finds such a pair, S_0 returns C^*.

Otherwise, S_0 evaluates *root*(x^*). If false, S_0 chooses a fresh random string $C^* \in \mathbf{W}^c$, inserts (x^*, C^*) into T, and returns C^*.

If true, S_0 executes the following "reconstruction procedure" to determine whether x^* is the final compression function call in a computation of $\mu^\phi(M)$, all of whose non-final compression function calls have already been seen by S_0. In the reconstruction procedure, S_0 will build a set Π, initially empty, of pairs (x, C) of compression function inputs and outputs.

1. Parse x^* into an ordered sequence of message bits, metadata, and compression function outputs. Let j be the number of compression function outputs.
2. For $i = 1, \ldots, j$:
   For each pair (x, C) in T, evaluate *parent*(x^*, x, i). If no pair (x, C) satisfies *parent*(x^*, x, i), or if multiple pairs (x, C) satisfy *parent*(x^*, x, i), quit the reconstruction procedure; choose a fresh random string $C^* \in \mathbf{W}^c$, insert (x^*, C^*) into T, and return C^*. If a unique child pair (x, C) is found, add (x, C) to Π. If *leaf*(x) is false, execute steps 1 and 2 for x.

If S_0 completes the reconstruction procedure (without returning a fresh random string), S_0 now calls $\rho(\Pi)$. If ρ returns $\perp$, S_0 chooses a fresh random string $C^* \in \mathbf{W}^c$, inserts (x^*, C^*) into T, and returns C^*. If ρ returns a message M^*, S_0 calls F on M^* and samples a string $C^* \in \mathbf{W}^c$ randomly from $\zeta^{-1}(F(M^*))$. S_0 inserts (x^*, C^*) into T and returns C^*.

Running Time. Let q_t be the total number of compression function queries to the simulator. To answer each compression function query, the S_0 takes $O(q_t^2 + \kappa(q_t))$ time. Therefore, the total running time of the simulator is $O(q_t^3 + q_t \cdot \kappa(q_t))$.

Correctness. In our proof, we will use the following correctness properties of the simulator.

Property 1: Suppose that there are no collisions on the output of S_0 (equivalently, there are no Type 1 events, which we define later). If all of the non-final compression calls in the computation of $\mu^\phi(M^*)$ have been made to S_0, then on the final compression call x^*, S_0 will reconstruct M^* and consult F on M^*. This property can be easily proven by induction, starting at the final query and working backward; the straight-line program property ensures that all the compression function calls in $\Sigma(M^*)$ will be found, and the assumption of no collisions ensures that no extra calls will be found.

Property 2: Suppose S_0 reconstructs M in response to a query at time τ. Then all of the compression calls in the computation of the $\mu^\phi(M)$ have been made to S_0 by time τ (indeed, otherwise $\rho(\Pi)$ would not return M, by definition).

4.3 Games

To prove the indifferentiability of the mode of operation μ, we consider a distinguisher D making compression function queries and mode of operation queries. We define a sequence of games, G_0 through G_4. Game G_0 is the "ideal" game, where D has oracle access to the random oracle F and the simulator S_0. Game G_4 is the "real" game, where D has oracle access to μ^ϕ and the random oracle ϕ.

For each game G_i, let p_i denote the probability that D outputs 1 in game G_i. We will argue that the view of the distinguisher cannot differ between consecutive games with more than negligible probability.

The games are defined as follows.

Game G_0**.** In Game G_0, the distinguisher D interacts with the random oracle F and the polynomial-time simulator S_0 defined above.

Game G_1**.** In Game G_1, we modify the simulator. At the start of the game, the new simulator S_1 uses its random coins to specify a random oracle $\mathcal{O}_S : \mathbf{W}^\eta \to \mathbf{W}^c$. The subsequent behavior of S_1 is identical to that of S_0 except, whereas S_0 generates its random bits in a lazy manner, S_1 gets its random bits from $\mathcal{O}_S$. Specifically, wherever S_0 answers a query x^* with a fresh random string $C^* \in \mathbf{W}^c$, S_1 answers with $\mathcal{O}_S(x^*)$. Similarly, wherever S_0 answers a query x^* by sampling randomly from $\zeta^{-1}(F(M^*))$, S_1 samples randomly from $\zeta^{-1}(F(M^*))$ using $\mathcal{O}_S(x^*)$ as its source of random bits.

The view of the distinguisher is the same in Game G_1 and Game G_0, so we have $p_1 = p_0$.

Game G_2**.** In Game G_2, we introduce a relay algorithm R_0 between D and F. The relay algorithm R_0 has oracle access to F and simply relays D's mode of operation queries to F and relays F's responses back to D.

The view of the distinguisher is the same in Game G_2 and Game G_1, so we have $p_2 = p_1$.

Game G_3**.** In Game G_3, we modify the relay algorithm. Instead of querying F, the new relay algorithm R_1 computes the mode of operation on its input, querying S_1 on each compression function call.

Game G_4**.** This is the final game. In Game G_4, we modify the simulator so that it no longer consults F. The new simulator S_2 always responds to a new query x^* with $\mathcal{O}_S(x^*)$. Thus, in this game D interacts with the mode of operation $\mu^{\mathcal{O}_S}$ and the random oracle $\mathcal{O}_S$.

Bad Events. In order to argue that the view of D cannot differ between Game G_2 and Game G_3 with more than negligible probability, we first define three types of "bad events" that can occur in Game G_2 or Game G_3.

- Type 1: S_1 inserts a pair (x_2, C) into T when there is already a pair (x_1, C) such that $x_1 \neq x_2$.
- Type 2: S_1 inserts a pair (x_2, C_2) into T when there is already a pair (x_1, C_1) in T such that x_1 is a parent of x_2.

– Type 3: D makes a query x_2 to S_1 such that x_2 is a parent of an x_1 which was previously queried by R but never directly by D.

We now prove that if none of these bad events occur in Game G_3, then the view of D is identical in G_2 and G_3.

Lemma 1. *For fixed coins of F, S_1, and D, if no bad events occur in Game G_3, then the view of D is identical in Game G_2 and Game G_3.*

Proof. We fix the random coins of F, S_1, and D and assume no bad events occur. We show by induction that D's observable values (the responses of the relay algorithm and the simulator) are identical in Games G_2 and G_3.

Suppose that the observable values have been identical in Games G_2 and G_3 so far. Consider D's next query. It is either a mode of operation query or a compression function query.

Mode of Operation Query. Consider a query M^* to μ^ϕ. In Game G_2, R_0 always returns $F(M^*)$. In Game G_3, R_1 will return $F(M^*)$ if the response of S_1 on the final compression call x^* is sampled from $\zeta^{-1}(F(M^*))$. There are two cases two consider.

1. x^* is a new query to S_1. Since R_1 has made all of the non-final calls for M^* before x^*, assuming there are no Type 1 events, by Correctness Property 1 of the simulator, S_1 will reconstruct M^* and return a string sampled from $\zeta^{-1}(F(M^*))$.
2. x^* has already been queried to S_1 before. Suppose it was first queried at time τ. Consider x^*'s children in the current computation $\mu^\phi(x)$. All of these children must have also been seen before time τ. Otherwise, a Type 2 event occurred in Game G_3 during one of the calls by R_1 after time τ. By induction, all of R_1's calls in the computation of $\mu^\phi(x)$ must have been seen before time τ. Therefore, assuming there are no Type 1 events, by Correctness Property 1 of the simulator, S_1 reconstructed M^* and returned a string sampled from $\zeta^{-1}(F(M^))$ at time τ.

Therefore, given that the observables have been the same so far, if the next query is a mode of operation query, the next observable will be the same in games G_2 and G_3.

Compression Function Query. Consider a query x^* to ϕ.

We first make the following observation. Let T_2 be the set of queries S_1 has seen so far in Game G_2. Let T_3 be the set of queries S_1 has seen so far in Game G_3. Assuming the observables in the two games have been identical so far, T_2 must be a subset of T_3. This is because T_2 contains only the compression function queries made by D and T_3 contains these queries along with the queries made by R_1 in computing responses to D's mode of operation queries.

Now, suppose that x^* has already previously been queried to S_1 in Game G_2. Then S_1's response will be the same in both games, since by assumption it was the same in both games the first time x^* was queried.

So suppose that x^* is a new query in Game G_2. If in Game G_2, S_1 reconstructs an M^* and returns an element of $\zeta^{-1}(F(M^*))$ sampled using $\mathcal{O}_S(x^*)$ for its random bits, then by Correctness Property 2 of the simulator, all of M^*'s queries are in T_2, and therefore all of M^*'s queries are in T_3. Therefore, assuming no Type 1 events, by Correctness Property 1, S_1 must also reconstruct M^* in Game G_3 and return an element of $\zeta^{-1}(F(M^*))$ sampled using $\mathcal{O}_S(x^*)$ for its random bits.

If in Game G_2, S_1 instead returns $\mathcal{O}_S(x^*)$, then S_1's answer in Game G_3 is the same unless in Game G_3 S_1 reconstructs an M^*. If S_1 reconstructs an M^* in Game G_3 but not in Game G_2, then at least one of the queries used in the reconstruction must have come from R_1. But D made a query for the final compression function input x^*. Consider all of x^*'s children in G_3. All of these queries must have been asked by D at some time. Otherwise, a Type 3 event occurred in game G_3. By induction, all of the queries for M^* must have been asked by D before x^*, but then (assuming no Type 1 events) S_1 would have reconstructed M^* in Game G_2, contradicting the assumption that S_1 returns $\mathcal{O}_S(x^*)$ in Game G_2.

Therefore, given that the observables have been the same so far, if the next query is a mode of operation query, the next observable will be the same in games G_2 and G_3.

Therefore, conditioned on there being no occurrences of bad events in G_3, the view of D is identical in games G_2 and G_3.

We now bound the probability of bad events in Game G_3.

Lemma 2. *Suppose D makes q_S compression function queries and generates q_t compression function calls from its mode of operation queries and its compression function queries. Let* Pr[Bad] *denote the probability that a bad event occurs in Game G_3.* $\Pr[\text{Bad}] \le (2\eta/c+1)q_t^2/2^{cw}$.

Proof. We first make the observation that on a new query x, S_1 always responds with a fresh random string in $\mathbf{W}^c$. This is because S_1 responds with either (a) $\mathcal{O}_S(x) \in \mathbf{W}^c$ or (b) a random sample from $\zeta^{-1}(F(M))$ using $\mathcal{O}_S(x)$ as its source of randomness. In case (a), S_1's response is a fresh random string in $\mathbf{W}^c$. In case (b), S_1's is a fresh random string in $\mathbf{W}^c$ as long as S_1 has not queried F on M before. Since a message M has a unique final compression call and x is a new query, S_1 cannot have queried F on M before.

We now consider the three types of bad events.

Type 1 event: A Type 1 event corresponds to a collision between two random c-word strings among at most q_t compression function queries. We can bound the probability using the birthday bound over q_t random c-word strings: $\Pr[\text{Type 1}] \le q_t^2/2^{cw+1}$.

Type 2 event: A Type 2 event occurs when S_1's random c-word response C_2 to a query x_2 equals one of the compression function outputs in the parsing of a previous query x_1. There at most η/c compression function outputs in the parsing of any query. So, there are at most $q_t \cdot \eta/c$ compression function outputs for S_1 to "guess" and S_1 has at most q_t guesses. Therefore, $\Pr[\text{Type 2}] \le (\eta/c \cdot q_t^2)/2^{cw}$.

Type 3 event: A Type 3 event occurs when D makes a query which is a parent of a query that R made but D did not. The only output that D sees from R's queries is $\zeta(f(x))$ for the final compression call x, but it is not possible to make a query which is a parent of a final compression call. The probability of D "guessing" a non-final compression function output that it has not seen is $1/2^{cw}$. There are at most $(q_t - q_S)$ outputs to guess and D has at most $q_S \cdot \eta/c$ guesses. Therefore, $\Pr[\text{Type 3}] \leq (\eta/c \cdot q_S(q_t - q_S))/2^{cw}$.

Summing together, we get $\Pr[\text{Bad}] \leq q_t^2/2^{cw+1} + (\eta/c \cdot q_t^2)/2^{cw} + (\eta/c \cdot q_S(q_t - q_S))/2^{cw} \leq (2\eta/c + 1)q_t^2/2^{cw}$.

4.4 Indifferentiability Theorem

We now state our theorem of indifferentiability of μ.

Theorem 2. *If $\phi : \mathbf{W}^\eta \to \mathbf{W}^c$ is a random oracle, then μ^ϕ is (t, q_F, q_S, ϵ)-indifferentiable from a random oracle $F : \{0,1\}^* \to \{0,1\}^d$, with $\epsilon = (2\eta/c + 1)q_t^2/2^{cw}$ and $t = O(q_t^3 + q_t \cdot \kappa(q_t))$, for any q_F and q_S such that the total number of compression function calls from the mode of operation queries and the compression function queries of the distinguisher is at most q_t.*

Proof. Consider a distinguisher D that makes q_S compression function queries and generates q_t compression function calls from its mode of operation queries and its compression function queries.

It can easily be seen that the view of D is the same in Games G_0, G_1, and G_2, so $p_2 = p_1 = p_0$.

Combining Lemma 1 and Lemma 2, we have that $|p_3 - p_2| \leq (2\eta/c+1)q_t^2/2^{cw}$.

It is straightforward to see that $p_4 = p_3$. To see this, consider a query x^* (from either D or R_1) to S_2. If S_2 has seen the query x^* before, S_2 repeats the answer it gave the first time it was queried on x^*. If x^* is a new query, then in G_4, S_2 responds with the fresh random string $\mathcal{O}_S(x^*)$. As argued previously in the proof of Lemma 2, S_1 always responds to a new query x^* with a fresh random string in $\mathbf{W}^c$. Thus, the view of D is unchanged from G_3 to G_4, and $p_4 = p_3$.

Summing over all the games, the total advantage of D in distinguishing between G_0 and G_4 is at most the claimed $(2\eta/c + 1)q_t^2/2^{cw}$.

This completes the proof of the theorem.

It follows from our indifferentiability theorem that the MD6 mode of operation $\mathcal{M}^{f_Q}$ (specified in [16]) is indifferentiable from a random oracle when f_Q is a random oracle. This result is stated below.

Corollary 1. *If $f_Q : \mathbf{W}^{n-q} \to \mathbf{W}^c$ is a random oracle, the MD6 mode of operation $\mathcal{M}^{f_Q}$ is (t, q_F, q_S, ϵ)-indifferentiable from a random oracle $F : \{0,1\}^* \to \{0,1\}^d$, with $\epsilon = 9q_t^2/2^{cw}$ and $t = O(q_t^3)$, for any q_F and q_S such that the total number of compression function calls from the mode of operation queries and the compression function queries of the distinguisher is at most q_t.*

Proof. It can easily be seen that the MD6 mode of operation $\mathcal{M}$ satisfies the required properties. In particular, for MD6 ζ is the function χ_d which returns the last d bits of its input. The *root*(x) predicate corresponds to testing whether the z bit of x is 1. It is straightforward to define a message reconstruction algorithm ρ for MD6 that runs in time $\kappa(q_t) = O(q_t)$.

Remark 2. The distinguishing probability ϵ and simulator running time t stated above follow directly from Theorem 2.

However, for MD6 specifically, we can easily get a tighter bound on ϵ because each compression function input has a unique node identifier, consisting of a tree level and index. Therefore, for a given compression function input y, in any other compression function input x, there is at most one (not $\eta/c = 4$) compression function output that could cause y to be a child of x. Thus, in bounding the probability of bad events for MD6, we can replace all instances of η/c with 1 and get $\epsilon = \Pr[\mathrm{Bad}] \leq 2q_t^2/2^{cw}$.

Similarly, the presence of unique node identifiers in MD6 gives us a tighter bound on the running time t of the simulator. By maintaining its previously seen queries in sorted order (sorted by node identifier), the simulator can run in time $O(q_t^2)$ instead of $O(q_t^3)$.

5 Conclusion

We have shown that the (reduced) compression function $\chi_{cw}(\pi(Q\|x))$ of MD6 is indifferentiable from a fixed-input-length random oracle when π is a random permutation. We have also shown that any tree-based mode of operation with certain properties is indifferentiable from a variable-input-length random oracle when applied to a compression function that is a fixed-input-length random oracle. As a consequence of this result, the MD6 mode of operation is indifferentiable from a random oracle. Combined, these results imply that the design of MD6 has no stuctural weaknesses, such as "extension attacks", and that MD6 can be plugged into any application proven secure assuming a variable-input-length random oracle to obtain a scheme secure in a (fixed-length) random permutation model.

References

1. J.-P. Aumasson and W. Meier. Nonrandomness observed on a reduced version of the compression function with 18 rounds in about 2^{17} operations.
2. M. Bellare and T. Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In X. Lai and K. Chen, editors, *Advances in Cryptology – ASIACRYPT '06*, number 4284 in LNCS, pages 299–314. Springer, 2006.
3. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge functions, May 2007. http://www.csrc.nist.gov/pki/HashWorkshop/Public Comments/2007 May.html.

5. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In *Proc. Eurocrypt 2008*, volume 4965 of *LNCS*, pages 181–197. Springer, 2008.
6. F. Chabaud and A. Joux. Differential collisions of SHA-0 . In *Advances in Cryptology – CRYPTO '98*, number 1462 in LNCS, pages 56–71. Springer, 1998.
7. D. Chang, S. Lee, M. Nandi, and M. Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In *ASIACRYPT*, pages 283–298, 2006.
8. J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In *Advances in Cryptology – CRYPTO '05*, number 3621 in LNCS, pages 430–448. Springer, 2005.
9. C. Y. Crutchfield. Security proofs for the MD6 hash function mode of operation. Master's thesis, MIT EECS Department, 2008. http://groups.csail.mit.edu/cis/theses/crutchfield-masters-thesis.pdf.
10. I. Dinur and A. Shamir. Cube attack on a reduced version of the compression function with 15 rounds.
11. Y. Dodis, K. Pietrzak, and P. Puniya. A new mode of operation for block ciphers and length-preserving MACs. In N. P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 198–219. Springer, 2008.
12. U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *Proc. TCC '04*, volume 2951 of *LNCS*, pages 21–39. Springer, Feb. 2004.
13. N. I. of Standards and Technology. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (sha-3) family. *Federal Register Notices*, 72(212):62212–62220, Nov. 2 2007. http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf.
14. N. I. of Standards and Technology. Announcing the development of new hash algorithm(s) for the revision of federal information processing standard (fips) 1802, secure hash standard. *Federal Register Notices*, 72(14):2861–2863, Jan. 23 2007. http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Jan07.pdf.
15. R. L. Rivest. Slides from invited talk at Crypto '08, 2008.
16. R. L. Rivest, B. Agre, D. V. Bailey, C. Crutchfield, Y. Dodis, K. E. Fleming, A. Khan, J. Krishnamurthy, Y. Lin, L. Reyzin, E. Shen, J. Sukha, D. Sutherland, E. Tromer, and Y. L. Yin. The MD6 hash function: A proposal to NIST for SHA-3, 2008.
17. X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In *Proc. Eurocrypt 2005*, number 3494 in LNCS, pages 1– 18. Springer, 2005.
18. X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In *Proc. CRYPTO 2005*, number 3621 in LNCS, pages 17–36. Springer, 2005.
19. X. Wang and H. Yu. How to break MD5 and other hash functions. In *Proc. EUROCRYPT 2005*, number 3494 in LNCS, pages 19–35. Springer, 2005.
20. X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In *Proc. CRYPTO 2005*, number 3621 in LNCS, pages 1–16. Springer, 2005.
21. H. Yu and X. Wang. Multicollision attack on the compression functions of MD4 and 3-pass HAVAL. IACR eprint, 2007. http://eprint.iacr.org/2007/085/.