Indifferentiability Characterization of Hash Functions and Optimal Bounds of Popular Domain Extensions

Rishiraj Bhattacharyya[1], Avradip Mandal[2], and Mridul Nandi[3]

[1] Applied Statistics Unit, Indian Statistical Institute, Kolkata, India
rishi_r@isical.ac.in
[2] Université du Luxembourg, Luxembourg
avradip.mandal@uni.lu
[3] NIST, USA
mridul.nandi@gmail.com

Abstract. Understanding the principle behind designing a good hash function is important. Nowadays it is getting more importance due to the current SHA3 competition which intends to make a new standard for cryptogrpahic hash functions. Indifferentiability, introduced by Maurer *et al* in TCC'04, is an appropriate notion for modeling (pseudo)random oracles based on ideal primitives. It also gives a strong security notion for hash-designs. Since then, we know several results providing indifferentiability upper bounds for many hash-designs. Here, we introduce a unified framework for indifferentiability security analysis by providing an indifferentiability upper bound for a wide class of hash designs GDE or *generalized domain extension*. In our framework, we present an unified simulator and avoid the problem of defining different simulators for different constructions. We show, the probability of some bad event (based on interaction of the attacker with the GDE and the underlying ideal primitve) is actually an upper bound for indifferentiable security. As immediate applications of our result, we provide simple and improved (in fact optimal) indifferentiability upper bounds for HAIFA and tree (with counter) mode of operations. In particular, we show that n-bit HAIFA and tree-hashing with counter have optimal indifferentiability bounds $\Theta(q\sigma/2^n)$ and $\Theta(q^2 \log \ell/2^n)$ respectively, where ℓ is the maximum number of blocks in a single query and σ is the total number of blocks in all q queries made by the distinguisher.

Key-words: Indifferentiability, Merkle-Damgård , HAIFA, Tree mode of operations with counter.

1 Introduction

Random Oracle method, introduced by Bellare and Rogaway [1], is a very popular platform for proving security of cryptographic protocol. In this model all the participating parties, including the adversary, is given access to a truly random function R. Unfortunately, it is impossible to realize a truly random function in practice. So while implementing the protocol the most natural choice is to instantiate R by an *ideal* hash function H. The formal proofs in Random Oracle model indicate that there is no structural flaw in the designed protocol. But how can we make sure, that replacing the random function R with a *good* hash function H will not make the protocol insecure? In fact recent results [13, 16] show that theoretically it is possible to construct some pathological protocols that are secure in random oracle model but completely insecure in standard model. Fortunately those separation results do not imply an immediate serious threat to any widely used cryptosystem, proven to be secure in random oracle model. So one can hope that any attack, which fails when a protocol is instantiated with R but succeeds when the protocol is instantiated with H, will use some structural flaw of H itself. So the above question boils down to the following. *How can we guarantee the structural robustness of a hash function H?*

Indifferentiability of Hash Functions: Motivated by above question, Coron et al. studied *Indifferentiability* of some known iterated hash designs[5], based on Maurer's indifferentiability framework [15]. Informally speaking, to prove indifferentiability of an iterated hash function C (based on some ideal primitive f), one has to design a simulator S. The job of S is to simulate the behavior of f while maintaining consistency with R. Now if no distinguisher D can distinguish the output distribution of the pair (C^f, f) from that of (R, S^R), the construction C is said to be indifferentiable from an RO. In [5], the authors proved that the well known Merkle-Damgård Hash function is indifferentiable from a random oracle under some specific prefix free padding rule. Subsequently, authors of [2, 4, 9, 12] proved indifferentiability of different iterated hash function constructions. Today indifferentiability is considered to be an essential property of any cryptographic hash function.

Related Work: In [14], Maurer introduced a concept of random systems and showed some techniques of proving *indistinguishability* of two random systems which can be useful to prove indistinguishability or even indifferentiability. However, Maurer's methodology can only be applied once one can prove the conditional probability distribution of the view (input/output) given non-ocurrance of bad event, remain identical in the two worlds. So far there is no known generic technique for finding the bad event and proving the distributions are actually identical. In [11], the authors introduced the concept of preimage awareness to prove the indifferentiability of MD with post-processor (modeled as an independent random oracle). More precisely, it was shown that if H is preimage-aware (a weaker notion than random oracle model) and R is a post-processor modeled as a random oracle then $R(H(\cdot))$ is indifferentiable. In[10], a particular tree mode of operation (4-ary tree) with specific counter scheme is shown to be indifferentiable secure.

Our Motivation: Although many known hash function constructions have been shown to be indifferentiable from an RO, the proof of these results are usually complicated (many times, due to numerous game hopings and hybrid arguments). Also, they require different simulators for each individual hash design. There are no known sufficient conditions for hash functions to be indifferentiable from an RO. From a different perspective, the existing security bounds for different constructions are not always optimal. In fact, to the best of our knowledge none of the known bounds was proven to be tight. The results of [11, 14] do not directly imply to improve the indifferentiability bounds for general iterated hash functions based on a single random oracle. The methods of [10] does not give us any optimal bound either. So a natural question to ask is: *Can we characterize the minimal conditions of a cryptographic hash function to be indifferentiable from a Random Oracle and achieve optimal bound?*

Our Result: In this paper, we present a unified technique of proving indifferentiabile security for a major class of iterated hash functions, called Generalized Domain Extensions. We extend the technique of [14] to the indifferentiability framework. We identify a set of events (called BAD events) and show that any distinguisher, even with unbounded computational power, has to provoke the BAD events in order to distinguish the hash function C from a random function R. Moreover we prove that, to argue indifferentiability of a construction C^f, one has only to show that the probability that any distinguisher invokes those BAD events, while interacting with the pair (C^f, f), is negligible. **We avoid the cumbersome process of defining simulator for each construction separately by providing a unified simulator for a wide range of constructions. To prove indifferentiability one simply need to compute the probability of provoking the BAD event when interacting with (C^f, f).**

In the second part of this paper, we apply our technique to some popular domain extension algorithms to provide optimal indifferentiable bounds. In particular, we consider Merkle-Damgård with HAIFA and tree mode with specific counter scheme.Many of candidates of *SHA3* competition actually use these two modes of operations. So, our result can also be viewd as an **optimal** indifferentiability guarantee of these candidates. We briefly describe our results below:

1. **MD with counter** or HAIFA: Let C^f be MD with counter where the last block counter is zero (all other counters are non-zero). Many SHA3 candidates such as BLAKE, LANE, SHAvite-3 etc are in this category. In Theorem 3 and Theorem 5, we show that the (tight) indifferentiable bound for C is $\Theta(\sigma q/2^n)$ where q is the number of queries, n is the size of the hash output and σ is total number of blocks in all the queries. The so far best known bound for HAIFA mode is $\sigma^2/2^n$ [5].
2. **Tree-mode with counter**: Tree mode with counter (e.g. the mode used in MD6) is known to be indifferentiable secure with upper bound $q^2\ell^2/2^n$ [10]. In Theorem 4 and Theorem 6, we are provide an optimal indifferentiable bound $\Theta(q^2 \log \ell/2^n)$.

2 Notations and Preliminaries

Let us begin with recalling the notion of indifferentiability, introduced by Maurer in [15]. Loosely speaking, if an ideal primitive $\mathcal{G}$ is indifferentiable from a construction C based on another ideal primitive $\mathcal{F}$, then $\mathcal{G}$ can be safely replaced by $C^{\mathcal{F}}$ in any cryptographic construction. In other terms if a cryptographic construction is secure in $\mathcal{G}$ model then it is secure in $\mathcal{F}$ model.

Definition 1. Indifferentiability [15]
A Turing machine C with oracle access to an ideal primitive $\mathcal{F}$ is said to be $(t, q_C, q_{\mathcal{F}}, \varepsilon)$ indifferentiable from an ideal primitive $\mathcal{G}$ if there exists a simulator S with an oracle access to $\mathcal{G}$ and running time at most t, such that for any distinguisher D, $|\Pr[D^{C^{\mathcal{F}},\mathcal{F}} = 1] - \Pr[D^{\mathcal{G},S^{\mathcal{G}}} = 1]| < \varepsilon$. The distinguisher makes at most q_C queries to C or $\mathcal{G}$ and

at most $q_{\mathcal{F}}$ queries to $\mathcal{F}$ or S. Similarly, $C^{\mathcal{F}}$ is said to be (computationally) indifferentiable from $\mathcal{G}$ if running time of D is bounded by some polynomial in the security parameter k and ε is a negligible function of k.

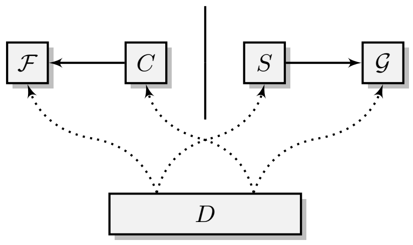

Fig. 1. The indifferentiability notion

We stress that in the above definition $\mathcal{G}$ and $\mathcal{F}$ can be two completely different primitives. As shown in Fig 1 the role of the simulator is to not only simulate the behavior of $\mathcal{F}$ but also remain consistent with the behavior of $\mathcal{G}$. Note that, the simulator does not know the queries made directly to $\mathcal{G}$, although it can query $\mathcal{G}$ whenever it needs.

For the rest of the paper C represents the domain extension algorithm of an iterated hash function. We consider $\mathcal{G}$ and $\mathcal{F}$ to be the same primitive; a random oracle. The only difference is $\mathcal{F}$ is a fixed length random oracle whereas $\mathcal{G}$ is a variable length random oracle. Intuitively a random function (oracle) is a function $f : X \to Y$ chosen uniformly at random from the set of all functions from X to Y.

Definition 2. *$f : X \to Y$ is said to be a* random oracle *if for each $x \in X$ the value of $f(x)$ is chosen uniformly at random from Y. More precisely, for $x \notin \{x_1, \ldots, x_q\}$ and $y, y_1, \cdots, y_q \in Y$ we have*

$$\Pr[f(x) = y \mid f(x_1) = y_1, f(x_2) = y_2, \cdots, f(x_q) = y_q] = \frac{1}{|Y|}$$

Most of the hash functions used in practice are iterated hash functions. The construction of an iterated hash function starts with a length compressing function $f : \{0,1\}^{m'} \to \{0,1\}^n$. Then we apply a domain extension technique, like the well known Merkle-Damgård , to realize a hash function $C^f : \{0,1\}^* \to \{0,1\}^n$. Intuitively, any practical domain extension technique applies the underlying compression function f in a sequence, where inputs of f are determined by previous outputs and the message $M \in \{0,1\}^*$ (for parallel constructions, inputs only depend on the message). Finally the output $C^f(M)$ is a function of all the previous intermediate outputs and the message M. The *Generalized Domain Extension* (GDE) are the domain extension techniques where u_ℓ is the input to final invocation of f and $C^f(M) = f(u_\ell)$. A domain extension algorithm from the class GDE is completely characterized by the following two functions:

1. **Length function**: $\ell : \{0,1\}^* \to \mathbb{N}$ is called *length function*, which actually measures the number of invocation of f. More precisely, given a message $M \in \{0,1\}^*$, $\ell = \ell(M)$ denotes the number of times f is applied while computing $C^f(M)$.
2. **Input function**: For each $j \in \mathbb{N}$, $U_j : \{0,1\}^* \times (\{0,1\}^n)^j \to \{0,1\}^{m'}$, called j^{th} *input function*. It computes the input of j^{th} invocation of f. This is computed from the message M and all $(j-1)$ previous outputs of f. In other words, $U_j(M, v_0, v_1, \cdots, v_{j-1})$ is the input of j^{th} invocation of f while computing $C^f(M)$, where $v_1, \cdots, v_{j-1}$ denote the first $(j-1)$ outputs of f and v_0 is a constant depending on the construction. The input function usually depend on message block, instead of whole message and hence we may not need to wait to get the complete message to start invoking f.

The above functions are independent of the underlying function f. Note that the padding rule of a domain extension is implicitly defined by the input functions defined above. At first sight, it may seem that GDE does not capture the constructions with independent post processor. But we argue that, when the underlying primitive is modeled like a random oracle, then queries to the post processor can be viewed as queries to same oracle (as in the intermediate queries) but with different padding. Namely in case of NMAC like constructions, we can consider a GDE construction where the inputs to the intermediate queries are padded with 1 and the final query is padded with 0. Similarly, one can incorporate domain extensions which use more than one random oracle.

Definition 3. (GDE: **Generalized Domain Extension**)
Let $\mathcal{S} = (\ell, \langle U_j \rangle_{j \geq 1})$ be tuple of deterministic functions as stated above. For any function $f : \{0,1\}^{m'} \to \{0,1\}^n$ and a message M, $\mathsf{GDE}^f_{\mathcal{S}}(M)$ is defined to be v_ℓ, where $\ell = \ell(M)$ and for $1 \leq j \leq \ell$,

$$v_j = f\big(U_j(M, v_0, v_1, \cdots, v_{j-1})\big).$$

The $u_j = U_j(M, v_0, v_1, \cdots, v_{j-1})$ is called the j^{th} intermediate input for the message M and the function f, $1 \leq j \leq \ell$. Similarly, $v_j = f(u_j)$ is called j^{th} intermediate output, $1 \leq j \leq \ell - 1$. The last intermediate input u_ℓ is also called final (intermediate) input. The tuple of functions $\mathcal{S}$ completely characterizes the domain extension and is called the **structure** *of the domain extension $\mathsf{GDE}_{\mathcal{S}}$.*

Note that we can safely assign $v_0 = IV$, the Initialization Vector, used in many domain extensions. In Fig 2 we describe the concept of GDE. Each G_i is an algorithm which computes the i^{th} intermediate input u_i, using the input-function U_i defined above. The wires between G_i and G_{i+1} is thick. In fact it contains all the previous input, output and the state information. In this paper we describe sufficient conditions to make a Generalized Domain Extension technique indifferentiable from a Random Oracle (RO). In the next section we show a hybrid technique to characterize the conditions and prove its correctness.

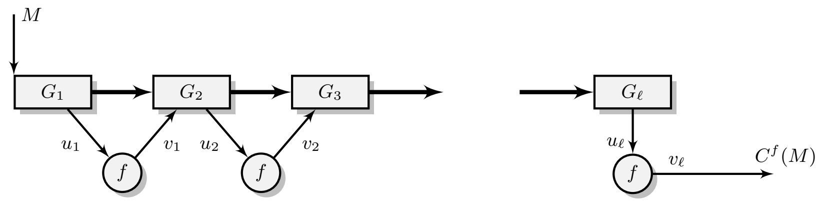

Fig. 2. The Generalized Domain Extension Circuit

3 Indifferentiability of GDE

In this section we discuss the sufficient condition for a domain extension algorithm C of the class GDE to be indifferentiable from a random oracle $\mathcal{R}$. Let C queries a fixed input length random oracle f. Recall that to prove the indifferentiability, for any distinguisher D running in time bounded by some polynomial of the security parameter κ, we need to define a simulator S such that

$$|\Pr[D^{C^f,f}=1]-\Pr[D^{\mathcal{R},S^{\mathcal{R}}}=1]|<\varepsilon(\kappa).$$

Here $\varepsilon(\kappa)$ is a negligible function and the probabilities are taken over random coin tosses of D and randomness of f and R. Let *right query* denote the queries to R/C^f and *left query* denote the queries to S^R/f. The simulator keeps a list L, initialized to empty. If u_i is the i^{th} query to the simulator and the response of the simulator was v_i then the i^{th} entry of L is the tuple (i, u_i, v_i).

Definition 4. *Let $C \in$ GDE. We say that $C^f(M)$ for a message M is computable from a list $L = \{(1, u_1, v_1), \cdots, (k, u_k, v_k)\}$ if there are $\ell = \ell(M)$ tuples $(i_1, u_{i_1}, v_{i_1}), \cdots,$ $(i_\ell, u_{i_\ell}, v_{i_\ell}) \in L$ such that for all $t \in \{1, 2, \cdots, \ell\}$,*

$$u_{i_t} = U_t(M, v_0, v_{i_1}, \cdots, v_{i_{t-1}}).$$

Intuitively for any simulator to work, C must have the following property:

Message Reconstruction: There should an efficient algorithm $\mathcal{P}$[4] such that given a set $L = \{(1, u_1, v_1), \cdots, (k, u_k, v_k)\}$, input-output of k many f queries and an input $u \in \{0,1\}^{m'}$ (in the domain of f); $\mathcal{P}(L, u)$ outputs M if $C^f(M)$ is computable from $L \cup \{(k+1, u, v)\}$ for all $v \in \{0,1\}^n$ where $u_\ell = u$ (as in Definition 4). If no such M exists $\mathcal{P}$ outputs $\perp$. If there are more than one such M, we assume $\mathcal{P}$ outputs any one of them.[5]

We argue that this is a very general property and is satisfied by all known secure domain extensions. In fact, the Message reconstruction algorithm $\mathcal{P}$ defined above is similar to the extractor of Preimage Awareness (PrA) of [11]. This is very natural as

[4] Note that the exact description of $\mathcal{P}$ depends on specific implementation.

[5] For example, $\mathcal{P}$ can choose a message randomly among all such messages. However, it will actually invoke BAD event.

the notion of PrA is much relaxed notion than that of PRO and every PRO is essentially PrA [11]. However existence of such an algorithm does not guarantee indifferentiability from a Random Oracle. For example, the traditional Merkle-Damgård construction is PrA but not PRO. In fact, The method of [11] is only applicable to prove indifferentiability when the final query is made to an independent post processor. On the other hand, Our contribution in this paper is to show a set of sufficient conditions along with the existence of extractor for a domain extension of the class GDE (where the final query can be made to that same function) to be a PRO.

Our simulator works as follows. Suppose the k^{th} query to the simulator is u. Then

- If $(i, u, v) \in L$ for some $i < k$ and some $v \in \{0, 1\}^n$, then $L = L \cup \{(k, u, v\}$ and return v.
- If $\mathcal{P}(L, u) = M$
  - $L = L \cup \{(k, u, R(M))\}$
  - return $R(M)$
- If $\mathcal{P}(L, u) = \perp$
  - Sample $h \in_R \{0, 1\}^n$
  - $L = L \cup \{(k, u, h)\}$
  - return h

Without loss of generality, we can assume adversary maintains two lists L_{right} and L_{left} to keep the query-responses made to R/C^f and S^R/f respectively.

3.1 Security Games

To prove the indifferentiability of GDE we shall use hybrid technique. We start with the scenario when the distinguisher D is interacting with C^f, f.

```
A left query S(u)

  1. return $COM\_RO(u)$.

COM_RO($u$)

  1. return $f(u)$.
```

```
A right query C(M)

  1. $v_0 = \lambda$.
  2. $\ell = \ell(M)$.
  3. for $i = 1$ to $\ell$
     (a) $u_i = U_i(M, v_0, v_1, \cdots, v_{i-1})$.
     (b) $v_i = COM\_RO(u_i)$.
  4. return $v_\ell$.
```

Fig. 3. Procedures of Game 0

Game 0: In this game the distinguisher is given access to an oracle S for the left queries. Additionally, both C and S is given access to another oracle COM_RO which can make f queries. Note that C or S do not have direct access to f. S on an input (u), queries $COM\_RO(u)$. COM_RO on input u returns $f(u)$. Formally, Game 0 can be viewed as Fig 3. Since the view of the distinguisher remains unchanged in this game we have

$$Pr[D^{C^f,f} = 1] = Pr[G_0 = 1]$$

where G_0 is the event when the distinguisher outputs 1 in Game 0.

Game 1 Now we change the description of the subroutine COM_RO and gives it an access to random oracle R as well. In this game COM_RO takes a 3-tuple (u, M, tag) as input where $u \in \{0,1\}^{m'}, M \in \{0,1\}^m$ and $tag \in \{0,1\}$. COM_RO returns $f(u)$ when $tag = 0$ and returns $R(M)$ otherwise. We also change the procedure to handle left and right query. In this game, the algorithm S maintains a list L containing the query number, input, output of previous left queries. While processing a right query M, the algorithm queries COM_RO with $tag = 1$ when querying with u_ℓ and makes $tag = 0$ for all other queries. Informally speaking, for a right query M, the algorithm C behaves almost similarly as game 0, except it returns $R(M)$ as the response. Similarly when a left query is a trivially derived from L and some message M, the algorithm sets $tag = 1$ before querying COM_RO and sets $tag = 0$ otherwise. Formally $Game1$ can be viewed as Figure 4.

```
A left query S(u)

1. If $(j, u, v) \in L$ for some $v, j$, return $v$.
2. If $\mathcal{P}(L, u) = M \neq \perp$
   (a) $v = COM\_RO(u, M, 1)$.
   (b) $index = index + 1$.
   (c) ADD $(index, u, v)$ to $L$
   (d) return $v$
3. else \\$\mathcal{P}(L, u) = \perp$
   (a) $v = COM\_RO(u, \lambda, 0)$.
   (b) $index = index + 1$.
   (c) ADD $(index, u, v)$ to $L$
   (d) return $v$
```

```
A right query C($M$)

1. $v_0 = IV$.
2. $\ell = \ell(M)$.
3. for $i = 1$ to $\ell - 1$
   (a) $u_i = U_i(M, v_0, v_1, \cdots, v_{i-1})$.
   (b) $v_i = COM\_RO(u_i, \lambda, 0)$.
4. $u_\ell = U_i(M, v_0, v_1, \cdots, v_{\ell-1})$.
5. $v_\ell = COM\_RO(u_\ell, M, 1)$.
6. return $v_\ell$.

COM_RO($u, M, tag$)

1. if $tag = 0$ return $f(u)$.
2. else return $R(M)$
```

Fig. 4. Procedures of Game 1. The variable $index$ represents the number of distinct queries made to S, so far; i. e. $index$ is the size of the list L. Initially index is set to 0. λ represent the empty string.

Definition 5. ***Trivial Query***
A left query u is said to be a trivially derived query (in short, trivial query) if there exist a $M \in L_{right}$ and k tuples $(i_1, u_{i_1}, v_{i_1}), \cdots, (i_k, u_{i_k}, v_{i_k}) \in L_{left}$ such that

– $u_{i_t} = U_t(M, v_0, v_{i_1}, \cdots, v_{i_{t-1}})$ *for all* $t \in \{1, 2, \cdots, k\}$
– $u = U_{k+1}(M, v_0, v_{i_1}, \cdots, v_{i_k})$

Similarly a right query M is said to be a trivial query if M is computable from L_{left}. Any other queries are said to be nontrivial queries.

Definition 6. BAD ***Events for Game 0 and Game 1***
Let D make q queries to a game (either Game 0 or Game 1). Let u_j be the j^{th} query when it is a left query and M_j be the j^{th} query when it is a right query. For i^{th} right query M_i, let u_i^f be the input to final COM_RO query and $u_{in,1}^i, u_{in,2}^i, \cdots$ be the inputs to the non-final intermediate COM_RO queries. The i^{th} query is said to set the BAD *event if one of the following happens*

- *for nontrivial right query $(M_i, right)$*
  - ***Collision in final input*** *The final input is same as final input of a previous right query. $u_i^f = u_j^f; i \neq j$ and $M_i \neq M_j$.*
  - ***Collision between final and non-final intermediate input***
    - *The final input is same as intermediate input of a previous right query, $u_i^f = u_{in,j}^k$ for some $k \leq i$ and $j < l(M_k)$.*
    - *One of the intermediate input is same as the final input of a previous right query. $u_{in,k}^i = u_j^f$ for some $j < i$ and $k \leq l(M_i)$*
  - ***Collision between final input and nontrivial left query*** *The final input is same as a non-trivial left query u_j; $u_i^f = u_j$ for some $j < i$ but u_j is not a trivial query for M_i.*
- *for left query $(u_i, left)$*
  - ***Collision between nontrivial left query and final input of a right query*** *$u_i = u_j^f$ for some $j < i$ but u_i is not trivially derived.*

Let us concentrate on how each of the event defined above can help the distinguisher. When nontrivial collision between the final input of two right (say M_i and M_j) query happens, the output of two queries will surely be a collision in Game 0. But in case of Game 1, the collision probability will be negligible. When final intermediate input of right query M_i collides with non-final intermediate input of another right query M_j, it may not be obvious how D can exploit this event. But we note that in that case output distribution of these two queries may not be independent in Game 0. The well known length extension attack can also be seen as exploiting this event. Finally if the final input of some right query M_j collides with input of some nontrivial left query u_i, the outputs of these two queries are same in Game 0. But it is easy to check that, in Game 1, they will be same with negligible probability. We stress that unless the nontrivial left query is same as the final input , adversary cannot gain anything. In fact in both of the games the output distribution remains same, even if the nontrivial left query collides with some non-final intermediate input of some right query.

Theorem 1. *Let $C \in$ GDE be a domain extension algorithm. Let* BAD *event be as defined in Definition 6. Then for any distinguisher D,*

$$|\Pr[D^{C^f,f} = 1] - \Pr[D^{R,S^R} = 1]| \leq \Pr[\text{BAD}^{C^f,f}]$$

where $\text{BAD}^{C^f,f}$ denotes the BAD *event when D is interacting with (C^f, f).*

Proof. To prove the theorem we will show the following relations. Let G_1 denote the event that the distinguisher outputs 1 in Game 1,

1. $|\Pr[G_0 = 1] - \Pr[G_1 = 1]| \leq \Pr[\text{BAD}^0]$.

2. $\Pr[G_1 = 1] = \Pr[D^{R,S^R} = 1]$

As $Pr[D^{C^f,f} = 1] = Pr[G_0 = 1]$ and $\Pr[\text{BAD}^0] = \Pr[\text{BAD}^{C^f,f}]$, the theorem will follow immediately. First we shall prove that if BAD events do not happen, then the input output distributions of Game 0 and Game 1 are identical. It is easy to check that $\neg$BAD is a monotone event as once BAD event happens (flag is set) it remains so for future queries. Now if the BAD events do not happen, then the final input of a right query is always "fresh" in both the games. So the output distribution remains same. On the other hand, if an input to a nontrivial left query is not same as the final input of a previous right query, then in both the cases the outputs are same and the output distribution of the left query is consistent with the previous outputs. Similar to [14], we view each input, output and internal states as random variables. We call the set of input, output and internal states as the transcript of the game. Let T_i^j denote the transcript of Game j after i^{th} query, $j = 0, 1$. Let BAD_i^0 and BAD_i^1 be the random variable of BAD event in i^{th} query in Game 0 and Game 1 respectively. The following lemma shows that the probability of BAD event occuring first in i^{th} query is same in both Game 0 and Game 1. Moreover if BAD does not happen in first i queries then the transcript after i^{th} query is identiaclly distributed in both the games.

Lemma 1. *1.* $\Pr[\text{BAD}_i^0 \wedge \neg(\cup_{k=1}^{i-1}\text{BAD}_k^0)] = \Pr[\text{BAD}_i^1 \wedge \neg(\cup_{k=1}^{i-1}\text{BAD}_k^1)]$
2. $\Pr[T_i^0 | \neg \cup_{k=1}^{i} \text{BAD}_k^1] = \Pr[T_i^1 | \neg \cup_{k=1}^{i} \text{BAD}_k^1]$

For a detail proof of the above Lemma, we refer the reader to Appendix A. As a direct application of this Lemma, we get the following results.

Corollary 1. *Let* BAD^j *denote the event that,* D *invokes* BAD *in Game* j*. Then we have,*

1. $\Pr[\text{BAD}^0] = \Pr[\text{BAD}^1]$
2. $\Pr[D^{G_0} \wedge \neg\text{BAD}^0] = \Pr[D^{G_1} \wedge \neg\text{BAD}^1]$

Using Corollary 1 one can get the following lemma.

Lemma 2. *Let* G_1 *denote the event that the distinguisher outputs* 1 *in Game* 1*.*

$$|\Pr[G_0 = 1] - \Pr[G_1 = 1]| \le \Pr[\text{BAD}^0]$$

For the proof of Lemma 2 we refer the reader to the full version of the paper.
Now we shall prove that $\Pr[G_1 = 1] = \Pr[D^{R,S^R} = 1]$. We prove it by hybrid arguments.
Game 2: In this game we change the description of C. Here we remove the lines $1-4$ in the description of C in Game 1 and change the query in line 5 to $COM\_RO(\lambda, M, 1)$ where λ is an empty string. So C does not anymore query COM_RO with $tag = 0$. Note that output of C is still $R(M)$. So the changes does not affect the input output distribution of the game. Hence

$$\Pr[G_2 = 1] = \Pr[G_1 = 1]$$

where G_2 is the event D outputs 1 in Game 2.

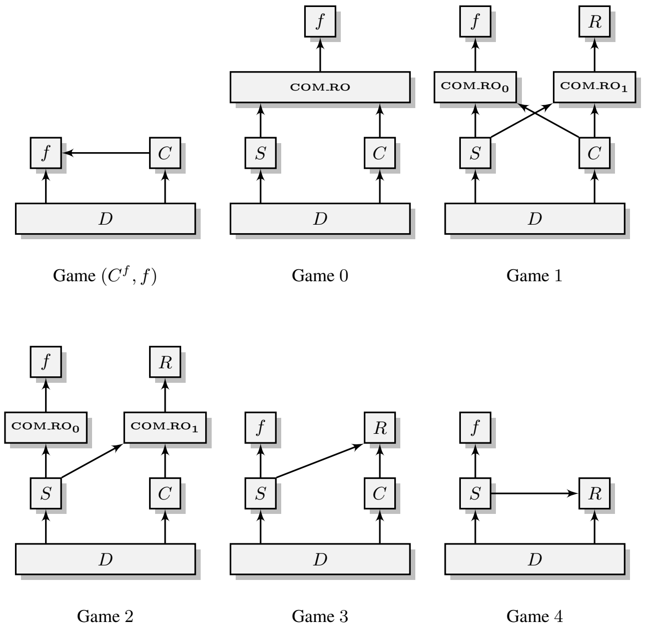

Fig. 5. Security Games

Game 3: Now we give S and C a direct access to f and R. So we replace the query $COM\_RO(u, M, 0)$ by $f(u)$. Similarly we write $R(M)$ in place of $COM\_RO(u, M, 1)$. As D did not have direct access to COM_RO and COM_RO did not modify any list, Game 3 is essentially same as Game 2. So

$$\Pr[G_3 = 1] = \Pr[G_2 = 1]$$

where G_3 is the event D outputs 1 in Game 3.
Game 4: In this game we remove the subroutine C. So the distinguisher D has direct access to R. Now as the simulator S had no access to internal variables of C, the input output distribution remains same after this change. So

$$\Pr[G_4 = 1] = \Pr[G_3 = 1]$$

where G_4 is the event D outputs 1 in Game 4.

The final observation we make is that S need not query f. Instead it can choose a uniform random value from $\{0,1\}^n$. Note that f is modeled as random function. So we changed a random variable of the game with another random variable of same distribution. Hence all the input, output, internal state distribution remains same. This makes S exactly the same simulator we defined.

$$\Pr[G_4 = 1] = \Pr[D^{R,S^R} = 1].$$

As the Game 0 is equivalent to the pair (C^f, f) we obtain our main result of the section (using triangle inequality):

$$|\Pr[D^{C^f,f} = 1] - \Pr[D^{R,S^R} = 1]| \leq \Pr[\text{BAD}^0] = \Pr[\text{BAD}^{C^f,f}]$$

□

4 Applications to popular mode of operations

In this section we show the indifferentiability of different popular mode of operations from a Random Oracle. We note that, according to Theorem 1 to upper bound distinguisher's advantage one needs to calculate the probability of BAD event defined in previous section. Moreover we can only concentrate on the specific mode of operation rather than the output of the simulator.

4.1 Merkle-Damgård with prefix free padding

It is well known that the usual Merkle-Damgård domain extension fails to satisfy indifferentiability property because of the length extension attacks. So we need to use some prefix free padding on the input message. Let g be the padding function. On input of message M and with oracle access to $f : \{0,1\}^{m'} \rightarrow \{0,1\}^n$, the MD domain extension computes the hash value using the following algorithm.
Merkle-Damgård $(MD^f(M))$

```
1. let $y_0 = 0^n$ (more generally, some fixed IV value can be used)
2. let $g(M) = (M_1, ..., M_l)$
3. for $i = 1$ to $l$
   – do $y_i = f(y_{i-1}, M_i)$
4. return $y_l$.
```

In [6], Coron et. al. proved indifferentiability of Merkle-Damgård Construction for prefix free padding. We reprove the result using Theorem 1 in a simpler way.

Theorem 2. *The prefix free Merkle-Damgård construction is $(t_S, q_C, q_{\mathcal{F}}, \varepsilon)$ - indifferentiable from a random oracle, with $t_S = \ell \cdot \mathcal{O}(q^2)$ and $\varepsilon = \mathcal{O}(\frac{\sigma^2}{2^n})$, where ℓ is the maximum length of a query made by the distinguisher D, σ is the sum of the lengths of the queries made by the distinguisher and $q = q_C + q_{\mathcal{F}}$.*

Note that for prefix free Merkle-Damgård constructions our simulator defined in Section 3 is similar to that of [12]. As shown in that paper, the simulator's running time is $\ell \cdot \mathcal{O}(q^2)$. For the proof of the above Theorem, the reader is referred to the full version of the paper. In this paper we concentrate on MD with a special padding rule, HAIFA.

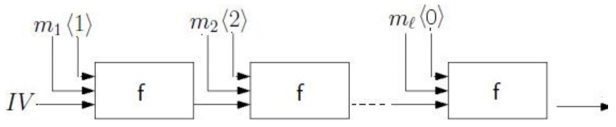

Fig. 6. Merkle-Damgård with padding rule HAIFA

4.2 Merkle-Damgård with HAIFA

Now we consider Merkle-Damgård mode of operation another variant of prefix free padding; HAIFA. In this padding we append a counter (indicating the block number) with each but last block of the message. The last block is padded with 0 (see Fig 6). It is easy to check that Merkle-Damgård with HAIFA belongs to GDE. In this case the reconstruction algorithm works as follows. Let t denote the length of the padding. On input of a f query u; check whether the last t bit of u is 0. If not return $\perp$. Otherwise parse u as $h_0||m_0$ where h_0 is of n bits. Find, whether h_0 is in the output column of a query in the list L. If no return $\perp$. If such a query exists select corresponding input u_i. Now last t bit of u_i will be $\ell - 1$, where ℓ is the number of blocks in possible message. We call such an u_i as $u_{\ell-1}$. Now for $j = \ell - 1$ to 2; parse u_j as $h_{j-1}||m_j$. find whether h_{j-1} exist in the output column of L where the corresponding input has padding $j - 1$. If no return $\perp$. Else select the input and call it u_{j-1}. Repeat the above three steps until we find a u_j with padding 1. If we can find such u_is, then construct the message $M = m_1||m_2|| \cdots m_\ell||m_0$ and return M. Check that for i^{th} query the algorithm P runs in time $\mathcal{O}(i\ell)$ where ℓ is the maximum block length of a query. Hence the total running time of P and hence of the simulator is $\mathcal{O}(q^2\ell)$.

For finding the probability of BAD events, the HAIFA padding rule gives us the following advantage. While computing $C^f(M)$ for any message M, all the intermediate inputs are unique. In fact the final input is always different from any intermediate input. So if no f query with same counter padding has collision in the output, the output of the penultimate f queries do not have collision in output and no nontrivial left query input is same as the final input of some right query, BAD event does not happen. If BAD event does not happen in i^{th} query, the output of i^{th} query is uniformly distributed over $Y = \{0,1\}^n$. Without loss of generality, we assume that D does not make any trivial query as trivial queries do not raise a BAD event. Moreover we can consider only a deterministic (albeit adaptive) distinguisher as the general case can easily be reduced to this case [17]. So input to the i^{th} query is uniquely determined by previous $i-1$ outputs. We represent the output of the nontrivial queries as the view (V) of the distinguisher. Let $f : \{0,1\}^{m'} \to \{0,1\}^n$ be a fixed input length random oracle. If D makes q nontrivial queries and $\mathcal{V}$ is the set of all possible views then $|\mathcal{V}| = |Y|^q$. We write V as $\cap_{i=1}^{q} V_i$, where V_i is the output corresponding to i^{th} query. Now for any $V \in \mathcal{V}$, we define an event BAD'^V which occurs whenever there is a collision between intermediate inputs, final inputs and left query inputs. In fact, $\neg\text{BAD}'^V \cap V \subseteq \neg\text{BAD} \cap V$. We split, BAD'^V as $\cup_{i=1}^{q} \text{BAD}_i'^V$. $\text{BAD}_i'^V$ occurs whenever any intermediate input (final or non-final) of i^{th} right query collides with any intermediate inputs of any other distinct right query or with input of any nontrivial left query. Although we are working with an adaptive attacker, future query inputs are fixed by V. Note that, if i^{th} query is left query $\text{BAD}_i'^V$ never occurs. Suppose ℓ_i is the number of blocks in i^{th} query.

Suppose the i^{th} query made by the distinguisher is a right query. For $\neg\text{BAD}_i'^V$ to happen, any intermediate input (final or non-final) has to be different from previous intermediate/final inputs. Because of HAIFA padding, no final input will be same with any intermediate input. So if $\neg\text{BAD}_i'^V$ has to be true, every intermediate input of i^{th} has to be different from the intermediate inputs with same counter of previous $i-1$ queries. Also any intermediate input can not be same as future left query inputs or future right query intermediate inputs fixed by the view. There only q many such candidates. So for any intermediate(final) input there are at most $i-1+q < 2q$ bad values. Hence,

$$\Pr[\neg\text{BAD}_i'^V \cap V_i | \cap_{j=1}^{i-1} (\neg\text{BAD}_j'^V \cap V_j)] \geq \left(\frac{|Y|-2q}{|Y|}\right)^{\ell_i - 1} \cdot \frac{1}{|Y|}.$$

If the i^{th} query is nontrivial left query,

$$\Pr[\neg\text{BAD}_i'^V \cap V_i | \cap_{j=1}^{i-1} (\neg\text{BAD}_j'^V \cap V_j)] = \frac{1}{|Y|}.$$

So one can calculate the probability of $\neg\text{BAD}$ as

$$\begin{aligned}
\Pr[\neg\text{BAD}] &= \sum_{V\in\mathcal{V}} \Pr[\neg\text{BAD} \cap V] \geq \sum_{V\in\mathcal{V}} \Pr[\neg\text{BAD}'^V \cap V] \\
&= \sum_{V\in\mathcal{V}} \Pr[\cap_{i=1}^{q}(\neg\text{BAD}_i'^V \cap V_i)] \\
&= \sum_{V\in\mathcal{V}} \Pr[\neg\text{BAD}_1'^V \cap V_1] \prod_{i=2}^{q} \Pr[\neg\text{BAD}_i'^V \cap V_i | \cap_{j=1}^{i-1} (\neg\text{BAD}_j'^V \cap V_j)] \\
&\geq \sum_{V\in\mathcal{V}} \prod_{i=1}^{q} \left(\frac{|Y|-2q}{|Y|}\right)^{\ell_i - 1} \cdot \frac{1}{|Y|} \\
&\geq \sum_{V\in\mathcal{V}} \left(1 - \mathcal{O}(\frac{\sigma q}{|Y|})\right) \cdot \frac{1}{|Y|^q} = 1 - \mathcal{O}(\frac{\sigma q}{|Y|})
\end{aligned}$$

Here $Y = \{0,1\}^n$ and $\sigma = \sum_{i=1}^{q} \ell_i$. So $\Pr[\text{BAD}] \leq \mathcal{O}(\frac{\sigma q}{2^n})$.

Theorem 3. *The Merkle-Damgård construction with HAIFA padding rule based on a FIL-RO is $(t_S, q_C, q_{\mathcal{F}}, \varepsilon)$ - indifferentiable from a random oracle, with $t_S = \ell \cdot \mathcal{O}(q^2)$ and $\varepsilon = \mathcal{O}(\frac{\sigma q}{2^n})$, where ℓ is the maximum length of a query made by the distinguisher D, σ is the sum of the lengths of the queries made by the distinguisher and $q = q_C + q_{\mathcal{F}}$.*

In [5], Coron et al. considered a specific prefix-free padding rule which is similar to HAIFA. There they proved indifferentiability bound as $\mathcal{O}(\frac{\sigma^2}{2^n})$. So Theorem 3 can be seen as improving that bound as well. In Section 5.1 we show that the bound we prove in Theorem 3 is tight.

4.3 Tree Mode of Operation with counter

Tree mode of operation is another popular mode of operation. *MD6*, a SHA3 candidate uses this mode of operation. Let $f : \{0,1\}^{m'} \to \{0,1\}^n$. The input message is divided in blocks and can be viewed as the leaf nodes. The edges are the function f. Any internal node can be viewed as the concatenation of the outputs of f on its child nodes. The output of the hash function is the output of f applied on the root. Now with each

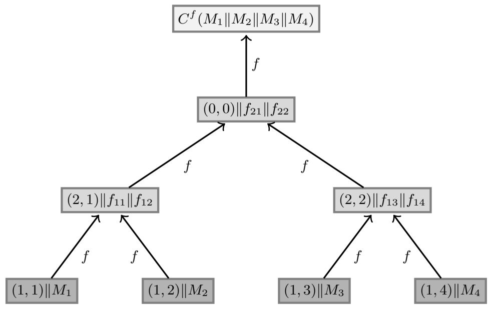

Fig. 7. Tree Mode of Operation with Sequential Padding where $\frac{m'}{n} = 2$

node we associate a tag $\langle height, index\rangle$ where $height$ denotes the height of the node in the tree and $index$ represents the index of the node in the level it is in (see Figure 7). Each node is padded with the tag. This padding makes, like HAIFA, each input unique in the evaluation tree of $C^f(M)$ for any fixed message M. One can easily construct the computable algorithm $\mathcal{P}$ using the same method as in HAIFA. Due to space constraint we don't describe the it here. Let M_i and M_j be two distinct right queries (for simplicity, both of length ℓ) made by distinguisher. Let k be an index such that k^{th} block of M_i and M_j is different. Consider the path from node $(1,k)$ to the root. It is easy to check that if no collision happens in this path, the final input of f query does not collide while computing $C^f(M_i)$ and $C^f(M_j)$. Length of this path is $\log \ell$ (height of the tree). On the other hand a nontrivial left query input can collide with at most one intermediate input of a right query. Hence, using a method similar to proof of Theorem 3, one can prove the following theorem

Theorem 4. *Let $\mathcal{F}$ be a FIL-RO and C be the tree mode of operation with the counter padding. $C^{\mathcal{F}}$ is $(t_S, q_C,\ q_{\mathcal{F}}, \varepsilon)$ - indifferentiable from a random oracle, with $t_S = \ell \cdot \mathcal{O}(q^2)$ and $\varepsilon = \mathcal{O}(\frac{q^2 \log \ell}{2^n})$, where ℓ is the maximum length of a query made by the distinguisher D and $q = q_C + q_{\mathcal{F}}$.*

We refer the reader to the full version of the paper for a proof of the above theorem.

5 Indistinguishability attacks on popular mode of operations

In this section we show a lower bound for the advantage of a distinguishing attacker against Merkle-Damgård constructions with HAIFA padding and Tree mode of operations with counter padding scheme. The bound we achieve actually reaches the corresponding upper bound shown before. Note, if all the queries are of length ℓ, then $q^2\ell = q\sigma$.

5.1 Distinguishing Attacks on Merkle-Damgård Constructions

Consider q messages $M_1, \cdots, M_q$ such that,

$$\begin{aligned}
\textsc{Pad}(M_1) &= M_1^1||M^2||\cdots||M^\ell \\
\textsc{Pad}(M_2) &= M_2^1||M^2||\cdots||M^\ell \\
&\vdots \\
\textsc{Pad}(M_q) &= M_q^1||M^2||\cdots||M^\ell
\end{aligned}$$

Let $\textsc{Coll}$ be the event denoting collision among $C^f(M_1), \cdots, C^f(M_q)$. We shall prove that, $\Pr[\textsc{Coll}] = \Omega(\frac{q^2\ell}{2^n})$ Let $\textsc{Coll}_{ij}$ be the event denoting the collision between $C^f(M_i)$ and $C^f(M_j)$. Hence,

$$\Pr[\textsc{Coll}] = \Pr[\bigcup_{1\le i<j\le q} \textsc{Coll}_{ij}].$$

Using principle of inclusion-exclusion we get,

$$\begin{aligned}
&\Pr[\bigcup_{1\le i<j\le q} \textsc{Coll}_{ij}] \ge \sum_{1\le i<j\le q} \Pr[\textsc{Coll}_{ij}] - \sum_{1\le i<j<k\le q} \big(\Pr[\textsc{Coll}_{ij} \cap \textsc{Coll}_{jk}] \\
&+ \Pr[\textsc{Coll}_{ij} \cap \textsc{Coll}_{ik}] + \Pr[\textsc{Coll}_{ik} \cap \textsc{Coll}_{jk}]\big) \\
&- \sum_{1\le i<j<k<r\le q} \big(\Pr[\textsc{Coll}_{ij} \cap \textsc{Coll}_{kr}] + \Pr[\textsc{Coll}_{ik} \cap \textsc{Coll}_{jr}] \\
&+ \Pr[\textsc{Coll}_{ir} \cap \textsc{Coll}_{jk}]\big) \qquad (1)
\end{aligned}$$

In the full version of the paper, we prove the following Lemma.

Lemma 3. *Let $Y = \{0,1\}^n$ and $1 \le i < j < k < r \le q$. If $\ell - 1 \le 2^n$, then*

1. $\Pr[\textsc{Coll}_{ij}] \ge \frac{\ell}{2|Y|}$
2. $\Pr[\textsc{Coll}_{ij} \cap \textsc{Coll}_{jk}] \le \frac{2\ell^2}{|Y|^2}$
3. $\Pr[\textsc{Coll}_{ij} \cap \textsc{Coll}_{kr}] \le \frac{\ell^2}{|Y|^2} + \frac{6\ell^3}{|Y|^3}$

Using Equation 1 and Lemma 3 one can prove,

$$\Pr[\textsc{Coll}] \ge \binom{q}{2}\frac{\ell}{2|Y|} - 3\binom{q}{3}\frac{2\ell^2}{|Y|^2} - 3\binom{q}{4}(\frac{\ell^2}{|Y|^2} + \frac{6\ell^3}{|Y|^3}) \approx \frac{\alpha}{4} - \frac{\alpha^2}{8} \ge \frac{\alpha}{8}$$

where $\alpha = \frac{q^2\ell}{2^n} < 1$. By Birthday Bound, for a random function R, the collision probability for q different messages is $\Theta(\frac{q^2}{2^n})$. Hence for a distinguisher D which queries messages $M_1, \cdots, M_q$, the advantage of the distinguisher is $\Omega(\frac{q^2\ell}{2^n})$. Also we can easily construct such q messages for any prefix free Merkle-Damgård scheme, specifically HAIFA.

Theorem 5. *Let C be the Merkle-Damgård domain extension with a prefix free padding. There exist a distinguisher D, such that*

$$|\Pr[D^{C^f,f} = 1] - \Pr[D^{S^R,R} = 1]| \geq \Omega(\frac{q^2\ell}{2^n})$$

where D makes q queries and length of each query is at most ℓ.

5.2 Distinguishing Attacks on Tree Mode

Similar to previous attack we choose q messages $M_1, \cdots, M_q$ such that after padding only first block of these messages are different. Formally

$$\textsc{Pad}(M_i) = M_i^1||M^2||\cdots||M^\ell.$$

Now for these massages the tree mode works like a Merkle-Damgård mode with messages $\overline{M}_1, \cdots, \overline{M}_q$ where

$$\textsc{Pad}(\overline{M_i}) = M_i^1||\overline{M}^2||\cdots||\overline{M}^h \; \forall i = 1, 2, \cdots, q$$

$h = \lceil \log \ell \rceil$ is the height of the tree. Hence using the similar method to previous section we get the following Theorem.

Theorem 6. *Let C be the Tree mode domain extension with the sequential counter padding. There exist a distinguisher D, such that*

$$|\Pr[D^{C^f,f} = 1] - \Pr[D^{S^R,R} = 1]| \geq \Omega(\frac{q^2 \log \ell}{2^n})$$

where D makes q queries and length of each query is atmost ℓ.

6 Conclusion and Future work

In this paper we proposed a unified method to prove indifferentiability of a wide class of iterated hash function, called GDE. Using our method we proved optimal indifferentiability bounds for Merkle-Damgård construction with counter (e.g. HAIFA) mode and for Tree Mode constructions with a similar sequential padding. This result shows tight indifferentiability bound (when the underlying compression functions are realized as random oracles) for many SHA3 candidates like BLAKE, LANE, SHAvite-3, MD6 etc. We strongly believe that tight indifferentiability bounds for MD constructions with independent post-processor [11] can also be proved using our method.

References

1. M. Bellare and P. Rogaway. Random Oracles Are Practical : A Paradigm for Designing Efficient Protocols. In *1st Conference on Computing and Communications Security*, ACM, pages 62–73. 1993.

2. M. Bellare and T. Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In *Advances in Cryptology - Asiacrypt 2006*, volume 4284 of *LNCS*, pages 299-314, Springer-Verlag, 2006.
3. R. Barke On the Security of Iterated MACs. Diploma Thesis '03. ETH Zurich
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. On the indifferentiability of the sponge construction. In *Advances in Cryptology- Eurocrypt 2008*, volume 4965 of LNCS, pages 181-197. Springer-Verlag, 2008.
5. J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya. Merkle-Damgard Revisited: How to Construct a Hash Function. In *Advances in Cryptology- Crypto 2005*, volume 3621 of *LNCS*, pages 430–448. Springer-Verlag, 2005.
6. J. S. Coron, Y. Dodis, C. Malinaud and P. Puniya. Merkle-Damgard Revisited: How to Construct a Hash Function (full version of [5]). `http://cs.nyu.edu/~dodis/ps/merkle.ps`
7. J. S. Coron, J. Patarin and Y Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In *Advances in Cryptology- Crypto 2008* volume 5157 of *LNCS*, Springer-Verlag, pages 1-20, 2008.
8. I. Damgård A Design Principles for hash functions. In *Advances in Cryptology-CRYPTO 1989*, volume 435 of *LNCS*, pages 416-427, Springer-Verlag, 1989.
9. Y. Dodis, K. Pietrzak, and P. Puniya. A new mode of operation for block ciphers and length-preserving MACs. In *Advances in Cryptology-EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 198-219. Springer-Verlag, 2008.
10. Y. Dodis, L. Reyzin, R. Rivest and E. Shen. Indifferentiability of Permutation-Based Compression Functions and Tree-Based Modes of Operation, with Applications to MD6. In FSE 2009.
11. Y. Dodis, T. Ristenpart and T. Shrimpton. Salvaging Merkle-Damgård for Practical Applications. In Advances in Cryptology, Eurocrypt 2009, volume 5479 of *LNCS*, pages 371-388,Springer-Verlag, 2009.
12. D. Chang, S. Lee, M. Nandi, and M. Yung. Indifferentiable security analysis of popular hash functions with prefix-free padding. In *Advances in Cryptology - Asiacrypt 2006*, volume 4284 of *LNCS*, pages 283-298, Springer-Verlag, 2006
13. R. Canetti, O. Goldreich and S. Halevi. The random oracle methodology, revisited. In *STOC' 1998*, ACM,1998.
14. U. Maurer. Indistinguishability of Random Systems. In Advances in Cryptology- EUROCRYPT 2002,volume 2332 of *LNCS* pages 110-132, Springer-Verlag, 2002.
15. U. Maurer, R. Renner and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC'2004*, volume 2951 of *LNCS*, pages 21–39. Springer-Verlag, 2004.
16. J. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *Advances in Cryptology-Crypto 2002*, volume 2442 of *LNCS*, Springer-Verlag, 2002.
17. M. Nandi A Simple and Unified Method of Proving Indistinguishability In *Progress in Cryptology - Indocrypt 2006*, volume 4329 of *LNCS*, pages 317-334, Springer-Verlag, 2002.
18. SHA 3 official website `http://csrc.nist.gov/groups/ST/hash/sha-3/Round1/submissions_rnd1.html`

Appendix A Proof of Lemma 1

Let $X_1^j, X_2^j, \cdots, X_q^j \in \mathcal{X}$ and $Y_1^j, Y_2^j, \cdots, Y_q^j \in \mathcal{Y}$ be input random variables and output random variables respectively of Game j; $j \in \{0,1\}$. Let $U_{1,i}^j, U_{2,i}^j, \cdots, U_{\ell_i,i}^j$ be the internal random variables (output of internal queries) of i^{th} query in Game j. As previously We call the set of input,output and internal states, the transcript of the game. Let T_i^j denote the transcript of Game j after i^{th} query.

In this proof,w.l.g., we assume that Distinguisher does not repeat queries. Let q be the number queries the adversary make. We shall prove the Lemma 1 by induction on i.

CASE $i = 1$: We start from the observation that

$$\Pr_D[X_1^0] = \Pr_D[X_1^1].$$

If X_1 is a right query $(M_1, right)$

$$\Pr_{D,f}[(U_1^0, \cdots, U_{\ell_1,1}^0)|X_1^0] = \Pr_{D,f}[(U_1^1, \cdots, U_{\ell_1,1}^1)|X_1^1].$$

Hence

$$\Pr_{D,f}[X_1^0, U_{1,1}^0, U_{2,1}^0, \cdots, U_{\ell_1,1}^0] = \Pr_{D,f,R}[X_1^1, U_{1,1}^1, U_{2,1}^1, \cdots, U_{\ell_1,1}^1].$$

It is easy to check that for the first query $\textsc{Bad}$ event can only be set by a right query. Also note that it happens when the final query is same with some non-final intermediate query. So

$$\Pr_{D,f}[\textsc{Bad}_1^0|X_1^0, U_{1,1}^0, U_{2,1}^0, \cdots, U_{\ell_1,1}^0] = \Pr_{D,R,f}[\textsc{Bad}_1^1|X_1^0, U_{1,1}^0, U_{2,1}^0, \cdots, U_{\ell_1,1}^0].$$

Hence

$$\Pr_{D,f}[\textsc{Bad}_1^0] = \Pr_{D,R,f}[\textsc{Bad}_1^1]$$

If $\neg\textsc{Bad}_1$ is true then $u_1^f \notin I_{M_1}$. As f and R are random oracles, we have

$$\Pr_f[f(u_1^f) = v] = \Pr_R[R(M_1) = v] \forall v \in \{0,1\}^n.$$

On the other hand if the first query is $(u, left)$ for any u, then $Y_1 = f(u)$ in both the games. So, $\forall v \in \{0,1\}^n$

$$\Pr_{D,f}[Y_1^0 = v|X_1^0, U_{1,1}^0, U_{2,1}^0, \cdots, U_{\ell_1,1}^0 \wedge \neg\textsc{Bad}_1^0]$$
$$= \Pr_{D,R,f}[Y_1^1 = v|X_1^1, U_{1,1}^1, U_{2,1}^1, \cdots, U_{\ell_1,1}^1 \wedge \neg\textsc{Bad}_1^1].$$

Hence,

$$\Pr_{D,f}[X_1^0, U_{1,1}^0, U_{2,1}^0, \cdots, U_{\ell_1,1}^0, Y_1^0|\neg\textsc{Bad}_1^0]$$
$$= \Pr_{D,f,R}[X_1^1, U_{1,1}^1, U_{2,1}^1, \cdots, U_{\ell_1,1}^1, Y_1^0|\neg\textsc{Bad}_1^1].$$

This Implies that the distribution of transcript after first query is identical in both the games if $\neg\textsc{Bad}_1$ is true. This finishes the proof of the case $i = 1$.
Suppose the lemma is true for all $i < t$.
$\textsc{Case}$ $i = t$:By Induction Hypothesis, we have,

$$\Pr_{D,f}[T^0_{t-1}|\neg(\cup_{k=1}^{t-1}\textsc{Bad}^0_k)] = \Pr_{D,f,R}[T^1_{t-1}|\neg(\cup_{k=1}^{t-1}\textsc{Bad}^1_k)].$$

As the input/output distribution of two games are same if $\neg(\cup_{k=1}^{t-1}\textsc{Bad}^1_k)$ is true, the distribution of t^{th} query must be same for both the games.

$$\Pr_{D,f}[X^0_t|T^0_{t-1} \wedge \neg(\cup_{k=1}^{t-1}\textsc{Bad}^0_k)] = \Pr_{D,f,R}[X^1_t|T^1_{t-1} \wedge \neg(\cup_{k=1}^{t-1}\textsc{Bad}^1_k)]$$

When $X_t = (u_t, left)$ is a non trivial left query then $Y_t = f(u_t)$ in both the games. Now if $\neg(\cup_{i=1}^{t-1}\textsc{Bad}_i)$ is true then, from induction hypothesis, the transcript distribution after t queries is same for both the games. The probability that u^f_t collides with some final input of any previous query is same for both the games. So for the left query

$$\Pr[\textsc{Bad}^0_t|\neg(\cup_{k=1}^{t-1}\textsc{Bad}^0_k)] = \Pr[\textsc{Bad}^0_t|\neg(\cup_{k=1}^{t-1}\textsc{Bad}^1_k)]$$

When $X_t = (M_t, right)$ then we have,

$$\Pr_{D,f}[X^0_t|T^0_{t-1} \wedge \neg\textsc{Bad}^0] = \Pr_{D,f,R}[X^1_t|T^1_{t-1} \wedge \neg\textsc{Bad}^1]$$

Notice that if the distribution of t^{th} query is same for both the games then the distribution of internal queries is also same for both the games. Hence

$$\Pr_{D,f}[X^0_t, U^0_{1,1}, \cdots, U^0_{\ell(M_t),(t)}|T^0_{t-1} \wedge \neg(\cup_{k=1}^{t-1}\textsc{Bad}^0_k)]$$
$$= \Pr_{D,f}[X^1_t, U^1_{1,1}, \cdots, U^1_{\ell(M_t),(t)}|T^1_{t-1} \wedge \neg(\cup_{k=1}^{t-1}\textsc{Bad}^1_k)].$$

Hence

$$\Pr_{D,f}[\textsc{Bad}^0_t|\neg(\cup_{k=1}^{t-1}\textsc{Bad}^0_k)] = \Pr_{R,D,f}[\textsc{Bad}^1_t|\neg(\cup_{k=1}^{t-1}\textsc{Bad}^1_k)]$$

For a non-trivial left query $(u_t, left)$, both the game queries $f(u_t)$. if $\neg(\cup_{k=1}^{t}\textsc{Bad}^1_k)$ is true then $u_t \neq u^f_j$ for all $j < t$. On the other hand , for a right query $(M_t, right)$, if $\neg(\cup_{k=1}^{t}\textsc{Bad}^1_k)$ is true then u^f_t has never been queried before. Then $\Pr_f[f(u^f_t) = v] = \Pr_R[R(M_t) = v]$ for all $v \in \{0,1\}^n$. So

$$\Pr_{D,f}[Y^0_t, X^0_t, U^0_{1,1}, \cdots, U^0_{\ell(M_t),(t)}|T^0_{t-1} \wedge \neg(\cup_{k=1}^{t}\textsc{Bad}^0_k)]$$
$$= \Pr_{D,R,f}[Y^1_t, X^1_t, U^1_{1,1}, \cdots, U^1_{\ell(M_t),(t)}|T^1_{t-1} \wedge \neg(\cup_{k=1}^{t}\textsc{Bad}^1_k)].$$

This implies

$$\Pr_{D,f}[T^0_t|\neg(\cup_{k=1}^{i}\textsc{Bad}^0_k)] = \Pr_{D,R,f}[T^1_t, \cdots, U^1_{\ell_i,i})|\neg(\cup_{k=1}^{i}\textsc{Bad}^1_k)]$$

□