Some Observations on Indifferentiability

Full version of the paper, an extended abstract appeared in the proceedings of ACISP'10

Ewan Fleischmann, Michael Gorski, and Stefan Lucks

Bauhaus-University Weimar, Germany
{ewan.fleischmann,stefan.lucks,michael.gorski}@uni-weimar.de

Abstract. At Crypto 2005, Coron et al. introduced a formalism to study the presence or absence of structural flaws in iterated hash functions: If one cannot differentiate a hash function using ideal primitives from a random oracle, it is considered structurally sound, while the ability to differentiate it from a random oracle indicates a structural weakness. This model was devised as a tool to see subtle real world weaknesses while in the random oracle world. In this paper we take in a practical point of view. We show, using well known examples like NMAC and the Mix-Compress-Mix (MCM) construction, how we can prove a hash construction secure and insecure at the same time in the indifferentiability setting. These constructions do not differ in their implementation but only on an abstract level. Naturally, this gives rise to the question what to conclude for the implemented hash function.
Our results cast doubts about the notion of "indifferentiability from a random oracle" to be a mandatory, practically relevant criterion (as *e.g.*, proposed by Knudsen [16] for the SHA-3 competition) to separate good hash structures from bad ones.

Keywords. hash function, provably secure, indifferentiability framework, ideal world models

1 Introduction

Random Oracle Methodology. A *hash function* $H : \{0,1\}^* \rightarrow \{0,1\}^n$ is used to compute an n-bit fingerprint from an arbitrarily-sized input. Established security requirements for cryptographic hash functions are collision resistance, preimage and 2nd preimage resistance. But, in an ideal world, most cryptographers expect a good hash function to somehow behave like a *random oracle* [4].

A random oracle is a mathematical abstraction used in cryptographic proofs, hiding away virtually all real world and implementational details. They are typically used when no known implementable function provides the mathematical properties required for the proof – or when it gets too tedious to formalize these properties. From a theoretical point of view, it is clear, that a security proof in the random oracle model is only a heuristic indication of the security of the system when instantiated with a particular hash function. In fact, many recent separation results [2, 7, 10, 12, 19, 21] illustrated various cryptographic systems secure in the random oracle model but completely insecure for *any* concrete instantiation of the random oracle. Nevertheless, these results do not seem to directly attack any concrete cryptosystem. In the random oracle model, one proves that the system is at least secure with and "ideal" hash function H. Such formal proof is believed to indicate that there are no structural flaws in the design of the system.
Building a Random Oracle. In practice, arbitrary length hash functions are built by first heuristically constructing a fixed-length building block, such as a fixed-length *compression function* or a block cipher, and then *iterating* this building block in some manner to extend the input domain arbitrarily.

Current *practical* hash functions, as *e.g.*, SHA-1, SHA-2 or MD5 [22, 23, 26], are all iterated hash functions using a compression function with a fixed-length input, $h : \{0,1\}^{n+l} \rightarrow \{0,1\}^n$, and the Merkle-Damgård transformation [9, 20] for the full hash function H with arbitrary input sizes. The core idea is to split the message M into l-bit blocks $M_1, \ldots, M_m \in \{0,1\}^l$ (with some padding to ensure that all the blocks are of size l-bit), to define an initial value H_0 and to apply the recurrence $H_i = h(H_{i-1}, M_i)$. The final *chaining variable* H_m is used as the hash output, *i.e.*, $H(M) := H_m$. The main benefit of the MD-transformation is that it preserves collision resistance: if the compression function h is collision resistant, then so is the hash function H.

Strctural Flaws in the Hash Function. Recent results on the security of the Merkle-Damgård construction [1, 13–15] indicate that there are some structural weaknesses in the design of the iteration process itself. They can be exploited even if the compression function is ideal – *i.e.*, a (fixed input length) random oracle. Motivated by the practical need to *"say anything about structural flaws in the design of H itself"*, Coron et al. [8] presented a new notion of security for cryptographic hash functions which is called *indifferentiability.*

In short, if one models the compression function(s) as random oracles with fixed-size inputs, then the iterated hash function composed from these compression functions should be indifferentiable from a random oracle with variably-sized inputs. They propose these as a practically relevant criterion, *e.g.*, to separate practical hash functions with a good structure from those which might suffer from structural flaws, especially in the context of the search for new hash function standard SHA-3 [16, 24]. The current paper discusses this issue.

Preliminaries

In this paper, we use notions such as "efficient", "significant" and "negligible" as usual in theoretical cryptography [29], e.g., an algorithm is efficient, if its running time is bounded from above by a polynomial in the security parameters. In the following we will call a hash function *secure* if it is indifferentiable from a random oracle (a formal definition will be given in Section 2) *i.e.*, there exists a simulator so that any efficient distinguisher has negligible advantage in distinguishing the hash function from a random oracle. A hash function is called *insecure* if there exists an efficient distinguisher that can distinguish the hash function for any simulator from a random oracle.

Remark: One purpose of this paper is to inspire a discussion about the practical relevance of the notions *secure* and *insecure*. More precisely: Does *insecure* actually indicate a structural flaw in a hash function whereas *secure* means the absence of them?

Our Contribution

Taking in a practical point of view, we will examine to what extent a structure of a hash function, proven secure using the indifferentiability framework, relates with instantiations satisfying this structure. This perspective is justified by the objective of Coron et. al. [8] to deliver a criteria for the design of practical hash functions that can distinguish between good hash structures and bad hash structures. On a merely abstract level – *i.e.*, if one views a hash function as a sole random oracle – the hash structure is trivially secure. Instantiated as one single collision resistant hash function, it is trivially insecure. We will examine what happens in between these two poles. We will show that one is able to prove one and the same practical hash function secure and insecure at the same time. These hash functions *do not* differ in their implementations but only on an abstract modeling level. Also, we will show how a slight modification to a secure hash function, *e.g.*, concatenation a one way function, can drive it insecure whereas concatenating an easily invertible function apparently preserves its security. We are able to derive some weird features that a secure hash function must offer. Moreover, as we can prove different structures that correspond to one and the same instantiated hash function secure and insecure, we are faced with an open problem what to conclude for the security of the practical hash function.

Section 2 gives an detailed paper outline and further motivates this discussion. Taking the practical point of view as a start, we show how one and the same implementation can be proven secure and insecure in the indifferentiability model. Section 3 introduces the random oracle model and the concept of "indifferentiability from a random oracle" as a security notion for hash functions and compares it to other ideal world security models for hash functions. Sections 4, 5, and Appendix C will give proofs for this.
In Section 6 we will derive some mandatory design principles for hash functions being secure in the indifferentiability framework. In Section 7 we discuss and conclude.

2 (In)Security in the Indifferentiability World

In the following sections we will examine various constructions that are secure in the indifferentiability framework (details on indifferentiability will follow in Section 3) involving one or more random oracles and show how slight modifications to them (or partial instantiations) drive them insecure (at least in this framework).

In this section we will motivate our research and summarize some of our results in Table 1. Furthermore, we will give a short example in which way our results correlate and with the design of practical hash functions.

Section	**Secure**	**Insecure** (partial instantiation or modification)	**Insecure** (extension)	**Secure**
4	$\mathcal{RO}$	X	$\mathcal{RO} \circ X$ $X \circ \mathcal{RO}$	$\mathcal{RO} \circ W$ $W \circ \mathcal{RO}$
4	$\mathcal{RO} \circ \mathcal{RO}$	$\mathcal{RO} \circ X$ $X \circ \mathcal{RO}$	$\mathcal{RO} \circ \mathcal{RO} \circ X$ $X \circ \mathcal{RO} \circ \mathcal{RO}$	$\mathcal{RO} \circ \mathcal{RO} \circ W$ $W \circ \mathcal{RO} \circ \mathcal{RO}$
5	$\mathcal{RO} \circ MD_{\mathcal{RO}}$ (NMAC)	$\mathcal{RO} \circ MD_Z$ $X \circ MD_{\mathcal{RO}}$	$\mathcal{RO} \circ MD_{\mathcal{RO}} \circ X$ $X \circ \mathcal{RO} \circ MD_{\mathcal{RO}}$	$\mathcal{RO} \circ MD_{\mathcal{RO}} \circ W$ $W \circ \mathcal{RO} \circ MD_{\mathcal{RO}}$
App. C	$\mathcal{RO}_i \circ X \circ \mathcal{RO}_i$ (MCM)	$\mathcal{RO}_x \circ X \circ Y$ $Y \circ X \circ \mathcal{RO}_x$ $X \circ \mathcal{RO}_x \circ Y$	$\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y$ $Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i$	$\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ W$ $W \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i$

Table 1. $\mathcal{RO}$ denotes a random oracle (with fixed or variable length input), $\mathcal{RO}_i$ an injective random 'oracle', $\mathcal{RO}_x$ a random oracle ($\mathcal{RO}_x$ is a fixed or variable input length, injective or not, random oracle), X,Y and Z collision resistant one-way functions (CROWF), W is an easily invertible function.

Our results are even stronger than indicated by Table 1.

Motivational, informal Example. Say we want to design a secure hash function and come up with the idea to design our hash function as a concatenation of a preprocessing function modeled as a random oracle $\mathcal{RO}$ and a collision resistant one way function (CROWF) X. Consequently, our hash function $\mathcal{H}$ for a message M is

$$\mathcal{H}(M) := (\mathcal{RO} \circ X)(M).$$

So we try to proof its security in the indifferentiability framework and come to the conclusion that this hash function is in fact insecure (refer to Theorem 1 (iii)). In the indifferentiability framework we have at least three straightforward approaches to get $\mathcal{H}$ secure:

1. Remove the CROWF X: $\mathcal{H}_1(M) = (\mathcal{RO})(M)$.
2. "Strengthen" X and make it a random oracle: $\mathcal{H}_2(M) = (\mathcal{RO} \circ \mathcal{RO})(M)$.
3. "Weaken" X and make it an easily invertible function W: $\mathcal{H}_3(M) = (\mathcal{RO} \circ W)(M)$.

The hash functions $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3$ can be proven secure in the indifferentiability framework (see Theorem 1 (i) and (ii)).

Indifferentiability was devised as a tool to see subtle real world weaknesses while in the random oracle world. But we can prove $\mathcal{H}$ insecure and $\mathcal{H}_3$ secure. In the real world (*i.e.*, comparing the instantiated hash functions) $\mathcal{H}_3$ is (almost) sure to be substantially weaker than $\mathcal{H}$. Additionally, the hash functions $\mathcal{H}$ and $\mathcal{H}_2$ could be implemented using exact the same lines of code but one is proved to be insecure, the other one seems to be secure. What shall we conclude for the security of our instantiated hash function in the real world?
How can we conclude that $\mathcal{H}$ has some real world weaknesses that $\mathcal{H}_2$ has not. Note that mixing complexity-theoretic and ideal building blocks is common and can *e.g.*, be found in [25].

3 Indifferentiability from a Random Oracle

For hash functions a random oracle serves as a reference model. It offers all the properties a hash function should have. This section gives an overview on all 'known methods' for comparing a hash function with a random oracle: indifferentiability and three weaker models: preimage awarenes, indifferentiability from a public-use random oracle and indistinguishability.

A random oracle, denoted $\mathcal{RO}$, takes as input binary strings of any length and returns for each input a random infinite string, *i.e.*, it is a map $\mathcal{RO} : \mathbb{Z}_2^* \to \mathbb{Z}_2^\infty$, chosen by selecting each bit of $\mathcal{RO}(x)$ uniformly and independently, for every x. As in [8] we will only consider random oracles $\mathcal{RO}$ truncated to a fixed output length $\mathcal{RO} : \mathbb{Z}_2^* \to \mathbb{Z}_2^n$.

Indifferentiability from a Random Oracle. The indifferentiability framework was introduced by Maurer et al. in [19] and is an extension to the classical notion of indistinguishability. Coron et al. [8] applied it to iterated hash function constructions and demonstrated for several iterated hash function constructions that they are indifferentiable from a random oracle if the compression function is a fixed input length (FIL) random oracle. Here, we give a brief introduction on these topics. For a more in-depth treatment, we refer to the original papers. In the context of iterated hashing, the adversary – called distinguisher D – shall distinguish between two systems as illustrated in Figure 1.

The system at the left (Case ALGORITHM) is the hash algorithm $\mathcal{H}_{Alg}$ using some ideal components (*i.e.*, FIL random oracles) contained in the set $\mathcal{G}$. The adversary can make queries to $\mathcal{H}_{Alg}$ as well as to the functions contained in $\mathcal{G}$. The system at the right consists of a random oracle

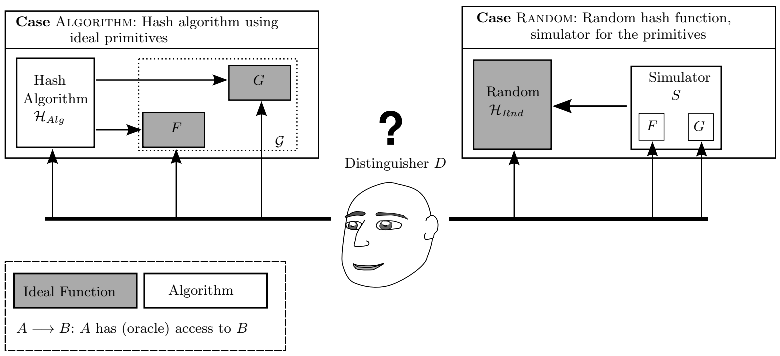

Figure 1. Defining $\mathcal{H}_{Alg}$ being indifferentiable from a random oracle $\mathcal{H}_{Rnd} := \mathcal{RO}$

(with truncated output) $\mathcal{H}_{Rnd} := \mathcal{RO}$ providing the same interface as the system on the left. To be indifferentiable to the system at the left, the system at the right (Case RANDOM) also needs a subsystem offering the same interface to the adversary as the ideal compression functions contained in $\mathcal{G}$. A simulator S is needed and its task is to simulate the ideal compression functions so that no distinguisher can tell whether it is interacting with the system at the left or with the one at the right. The output of S should look consistent with what the distinguisher can obtain from the random oracle $\mathcal{H}_{Rnd}$. In order to achieve that, the simulator can query the random oracle $\mathcal{H}_{Rnd}$. Note that the simulator does not see the distinguisher's queries to the random oracle. Formally, the indifferentiability of $\mathcal{H}_{Alg}$ from a random oracle $\mathcal{H}_{Rnd}$ is satisfied if:

Definition 1. *[8] A Turing machine $\mathcal{H}_{Alg}$ with oracle access to a set of ideal primitives contained in the set $\mathcal{G}$ is said to be (t_D, t_S, q, ϵ) indifferentiable from an ideal primitive $\mathcal{H}_{Rnd}$, if there exists a simulator S, such that for any distinguisher D it holds that:*

$$|Pr[D^{\mathcal{H}_{Alg},\mathcal{G}} = 1] - Pr[D^{\mathcal{H}_{Rnd},S} = 1]| < \epsilon.$$

The simulator has oracle access to $\mathcal{H}_{Rnd}$ and runs in time at most t_S. The distinguisher runs in time at most t_D and makes at most q queries. Similarly, $\mathcal{H}_{Alg}$ is said to be indifferentiable from $\mathcal{H}_{Rnd}$ if ϵ is a negligible function of the security parameter k.

Now, it is shown in [19] that if $\mathcal{H}_{Alg}$ is indifferentiable from a random oracle, then $\mathcal{H}_{Alg}$ can replace the random oracle in any cryptosystem, and the resulting cryptosystem is at least as secure in the ideal compression function model (*i.e.*, case ALGORITHM) as in the random oracle model (*i.e.*, case RANDOM).

'Non-Optimal' Ideal World Models. At EUROCRYPT'09 Dodis et. al. [11] have presented two ideal world security models that are strictly weaker than indifferentiability: *preimage awareness* and *indifferentiability from a public-use random oracle*. But both model a hash function fairly inadequate. A function that is preimage aware is not guaranteed to be secure against such trivial attacks as, *e.g.*, the Merkle-Damgård extension attack. And a function that is indifferentiability from a public-use random oracle has to 'publish' any oracle query and might be only of limited use in the context of some signature schemes.

If a hash function is indistinguishable from a random oracle, an attacker that can query $\mathcal{H}_{Alg}$ – but has no access to the compression functions contained in $\mathcal{G}$ – cannot distinguish it from a random oracle. For hash function constructions, indistinguishability makes little sense as, for any concrete hash function, the compression functions in $\mathcal{G}$ are public and hence accessible to the adversary. As opposed to block cipher constructions, there is no secret key or any other information the attacker has not. For them, indistinguishable from a random permutation seems to suffice (at least in the ideal cipher model).

Therefore, we will focus in this work on 'indifferentiability from a random oracle' since this seems to be the only security model known that is applicable in all contexts of cryptographic hash functions. The (open) challenge is to find an ideal world security model that is strong enough to defeat all known attacks but it should not be so strong that it leads to real world ambiguities. As we will show in this paper, the notion of indifferentiability has such ambiguities, namely we can prove one and the same real world hash function secure and insecure at the same time.

Security definitions that are based on a random oracle. Note that by assuming *ideal primitives* even in the ALGORITHM case, this definition is inherently based on the random oracle model. In the standard model we cannot assume ideal primitives (at least not without allowing an exponentially-sized memory to store a description of the function), so this notion of security only makes sense in the random oracle model.

Nevertheless, as we understand [8], a part of their motivation was to introduce a formalism for aiding the design of practical hash functions. Showing the above kind of "security" in the random oracle model ought to indicate the absence of structural flaws in the hash function.

On the other hand, if one can efficiently differentiate a hash function (using ideal primitives) from a random oracle, this appears to indicate a weakness in the hash function structure. With this reasoning, we again follow the example of Coron et al., who debunk certain hash function structures as insecure by pointing out efficient differentiation attacks [8, Sections 3.1 and 3.2].

4 Concatenation of Random Oracles: $\mathcal{RO} - \mathcal{RO}$

We start by investigating a fairly simple construction on what to conclude for the security of a hash function where a pre/post-processing function is available.

Definition 2. *Let*

$$F^{(*\to n)}, G^{(*\to n)} : \{0,1\}^* \to \{0,1\}^n \text{ and}$$
$$F^{(n\to n)}, G^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be random oracles. A Subindex 'i' denotes an injective random oracle, a subindex 'x' denotes a random oracle where we explicitly don't care whether it is injective or not. Let

$$P^{(*\to n)}, Q^{(*\to n)} : \{0,1\}^* \to \{0,1\}^n \text{ and}$$
$$P^{(n\to n)}, Q^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be collision resistant one way functions. Let

$$W^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be a function that is easily invertible.

(i) The hash function $H_{\mathcal{RO}\circ\mathcal{RO}} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ\mathcal{RO}}(M) := G^{(n\to n)}(F^{(*\to n)}(M)).$$

(ii) Modification/Partial instantiation I: The hash function $H_{\mathcal{RO}\circ X} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ X}(M) := F^{(n\to n)}(P^{(*\to n)}(M)).$$

(iii) Modification/Partial instantiation II: The hash function $H_{X\circ\mathcal{RO}} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{X\circ\mathcal{RO}}(M) := P^{(n\to n)}(F^{(*\to n)}(M)).$$

(iv) Extension I: The hash function $H_{\mathcal{RO}\circ\mathcal{RO}\circ X} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ\mathcal{RO}\circ X}(M) := F^{(n\to n)}(G^{(n\to n)}(P^{(*\to n)}(M))).$$

(v) Extension II: The hash function $H_{X\circ\mathcal{RO}\circ\mathcal{RO}} : \{0,1\}^* \to \{0,1\}^n$ *for a message* $M \in \{0,1\}^*$ *is defined by*

$$H_{X\circ\mathcal{RO}\circ\mathcal{RO}}(M) := P^{(n\to n)}(F^{(n\to n)}(G^{(*\to n)}(M))).$$

Theorem 1. *In the indifferentiability framework the following statements must hold:*

(i) $H_{\mathcal{RO}\circ\mathcal{RO}}$ *is secure (*i.e.*, indifferentiable from a random oracle),*
(ii) $H_{\mathcal{RO}\circ X}$ *is insecure (*i.e.*, differentiable from a random oracle),*
(iii) $H_{X\circ\mathcal{RO}}$ *is insecure,*
(iv) $H_{\mathcal{RO}\circ\mathcal{RO}\circ X}$ *is insecure,*
(v) $H_{X\circ\mathcal{RO}\circ\mathcal{RO}}$ *is insecure.*

Recall that for proving a hash function insecure we have to describe an efficient distinguisher which can decide with non-negligible probability if the hash function is an algorithm utilizing random oracles (the ALGORITHM case) or is a random oracle by itself (the RANDOM case).

Proof. Let H denote the hash oracle.

(i) The proof is easy and will be skipped here. It can be found in Appendix A.
Remark: The proof can be easily generalized to all functions $H_{\mathcal{RO}-\ldots-\mathcal{RO}}(M)$.

(ii) This result is essentially equivalent to the Coron et al. insecurity result regarding the composition of a CROWF with a random oracle [8], but we state a version of it here for completeness. We describe a distinguisher D to win this game, regardless of the simulator S:
1. Choose a random message $M \in \{0,1\}^*$.
2. Compute $u = P(M)$.
3. Ask the F-oracle for $v = F(u)$.
4. Ask the hash-oracle for $z = H(M)$.
5. If $z = v$ output ALGORITHM, else output RANDOM.

Analysis: Clearly, D is efficient. In the ALGORITHM world we always have

$$z = H(M) = F(P(M)) = F(u) = v.$$

so it always outputs ALGORITHM if it interacts with the algorithm and the ideal primitive F. A simulator trying to fool the distinguisher D does not know M as he receives the $F(u)$-oracle call. In order to answer correctly he has to come up with $M = P^{-1}(u)$ to ask the hash oracle for $H(M)$. As P is a CROWF this is not possible. Any such simulator can be used to invert P. Furthermore it is information-theoretically impossible to recover $M \in \{0,1\}^*$ from u.

(iii) Again, we describe a distinguisher D to win this game, regardless of the simulator S:
1. Choose a random message $M \in \{0,1\}^*$.
2. Ask the hash-oracle for $z = H(M)$.
3. Ask the F-oracle for $u = F(M)$.
4. Compute $v = P(u)$.
5. If $z = v$ output ALGORITHM, else output RANDOM.

Analysis: Again, D is obviously efficient and always outputs ALGORITHM in the algorithm world as we have

$$z = H(M) = P(F(M)) = P(u) = v.$$

In the random world, D learns a random target z and needs to find u with $z = P(u)$ whereas $u = F(M)$. So any simulator able to answer the F-oracle correctly can be used to invert P which is a CROWF.

(iv) The proof is essentially the same as in (ii).

(v) The proof is essentially the same as in (iii).

□

Paradox. As we have proven in (ii) and (iii) the hash functions $H_{\mathcal{RO}\circ X}$ and $H_{X\circ\mathcal{RO}}$ are in fact insecure, if X is a collision resistant one way function. What happens if we substitute that CROWF with an easily invertible function? It turns out that both hash functions get secure again. Taking preimage resistance as an example, we are not able to append an 'additional line of defense' (namely the function X) for preimage attacks. without losing the property of being 'indifferentiabity from a random oracle'. Note that we do *not* point out any paradox in the indifferentiability framework itself. But, indeed, we do show several ambiguities we inevitably have to face if we try to apply this framework as a guide for designing secure and practical hash functions. This situations occur if we try to decrease the size of the gap between a practical hash function and its ideal world mapping.

Theorem 2. *Using the same notations as in Definition 2 it must hold in the indifferentiability framework:*

(i) The hash function

$$H_{\mathcal{RO}\circ W}(M) := F^{(n\to n)}(W^{(*\to n)}(M)).$$

is secure if W is an invertible function.

(ii) The hash function

$$H_{W\circ\mathcal{RO}}(M) := W^{(n\to n)}(F^{(*\to n)}(M)).$$

is secure if W is an invertible function.

Proof. Let H denote the hash oracle.

(i) Note, that the function $W : \{0,1\}^* \to \{0,1\}^n$ is unlikely to be uniquely invertible in practice (normally, it will be information-theoretically impossible). But if we assume W's invertibility we can easily give a simulator S thus proving the hash function secure.
Description of Simulator S:
  - F-oracle queries (parameter A): As we can invert W we can easily calculate $M = W^{-1}(A)$ and ask the hash oracle $v = H(M)$ and return v to the caller.

(ii) Now we have the function $W : \{0,1\}^n \to \{0,1\}^n$. Again, we can give a simple simulator S.
Description of Simulator S:
  - F-oracle queries (parameter M): Ask the hash oracle $z = H(M)$ and return $W^{-1}(z)$ to the caller.

□

Recall the example of Section 2. Let us start with the hash function $H_{\mathcal{RO}\circ X}$. For X being a CROWF we have shown it to be insecure. If we *strengthen* X and let it be a random oracle our hash function gets secure. If we weaken X, *i.e.*, let X be an invertible function, our hash function gets secure again. The same is true if we begin our discussion with $H_{X\circ\mathcal{RO}}$.

Furthermore, all the theoretical hash functions $H_{\mathcal{RO}\circ\mathcal{RO}}$, $H_{X\circ\mathcal{RO}}$ and $H_{\mathcal{RO}\circ X}$ are a valid model for the same practical hash function, employing two functions F and G and defined by $H(x) = F(G(x))$. Should we conclude that this construction is secure, since we can prove their security if we model both F and G as random oracles? Or should we conclude that this construction is insecure, as we can disprove security in two other cases?

Insecurity by (partial) Instantiation. Another point of view of the problem is given here. We start with the proven-to-be-secure hash function $H_{\mathcal{RO}\circ\mathcal{RO}}$. So one should assume that there are no structural weaknesses found in our construction. In order to get a practical hash function we have to instantiate the random oracle by efficient collision resistant one way functions. Instead of instantiating both random oracles at the same time we choose to instantiate them one after the other. Our intermediate result is either $H_{X\circ\mathcal{RO}}$ or $H_{\mathcal{RO}\circ X}$. Both of them were proved the be insecure in Theorem 1 – but formally we still have not left the random oracle world. Informally, we start now with an insecure hash function and instantiate the other random oracle. Thus, regarding the structural soundness of our construction we get entirely contradicting messages again.

5 NMAC: $MD_{\mathcal{RO}} - \mathcal{RO}$

Definition 3 (NMAC-Hash). *Let*

$$C^{(n+m\to n)} : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n \text{ and}$$
$$D^{(n\to n)} : \{0,1\}^n \to \{0,1\}^n$$

be oracles, $M = (M_1, M_2 \ldots, M_L) \in (\{0,1\}^m)^L$ be a (padded) message and $H_0 \in \{0,1\}^n$ an arbitrary initial value. The Merkle-Damgård hash function $MD_C : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ is defined by

$$H_1 = C(H_0, M_1),\ \ldots,\ H_L = C(H_{L-1}, M_L),$$

$$MD_C(H_0, M_1, \dots, M_L) = H_L$$

The hash function $NMAC_{C,D} : \{0,1\}^n \times (\{0,1\}^m)^+ \to \{0,1\}^n$ *is defined for fixed* $H_0 \in \{0,1\}^n$ *as follows:*

$$NMAC_{C,D}(M_1, \dots, M_L) = D(MD_C(H_0, M_1, \dots, M_L)).$$

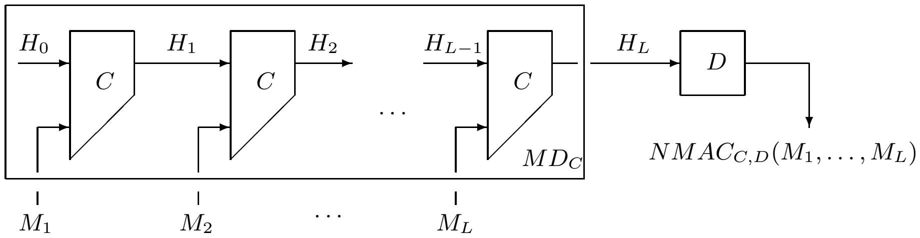

Figure 2. The NMAC-Hash $NMAC_{C,D}$ – an extension to the plain Merkle-Damgård hash function MD_C.

Definition 4. *Let* F, G, P, Q *be as in Definition 2. Additionally, let*

$$F^{(n+m\to n)} : \{0,1\}^{n+m} \to \{0,1\}^n$$

be a random oracle.

(i) The hash function $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}} : \{0,1\}^{m\cdot L} \to \{0,1\}^n$ *for a padded message* $M \in \{0,1\}^{m\cdot L}$ *is defined by*

$$\begin{aligned} H_{\mathcal{RO}\circ MD_{\mathcal{RO}}}(M) &= NMAC_{F^{(n+m\to n)},G^{(n\to n)}}(M) \\ &= G^{(n\to n)}(MD_{F^{(n+m\to n)}}(M)). \end{aligned}$$

(ii) Modification/Partial instantiation I: The hash function $H_{\mathcal{RO}\circ MD_X} : \{0,1\}^{m\cdot L} \to \{0,1\}^n$ *for a padded message* $M \in \{0,1\}^{m\cdot L}$ *is defined by*

$$\begin{aligned} H_{\mathcal{RO}\circ MD_X}(M) &= NMAC_{P^{(n+m\to n)},F^{(n\to n)}}(M) \\ &= F^{(n\to n)}(MD_{P^{(n+m\to n)}}(M)). \end{aligned}$$

(iii) Modification/Partial instantiation II: The hash function $H_{X\circ MD_{\mathcal{RO}}} : \{0,1\}^{m\cdot L} \to \{0,1\}^n$ *for a padded message* $M \in \{0,1\}^{m\cdot L}$ *is defined by*

$$\begin{aligned} H_{X\circ MD_{\mathcal{RO}}}(M) &= NMAC_{F^{(n+m\to n)},P^{(n\to n)}}(M) \\ &= P^{(n\to n)}(MD_{F^{(n+m\to n)}}(M)). \end{aligned}$$

(iv) Extension I: Let

$$R^{(*\to m\cdot L)} : \{0,1\}^* \to \{0,1\}^{m\cdot L}$$

be a padding function. The hash function $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}\circ R} : \{0,1\}^* \to \{0,1\}^n$ *for an (unpadded) message* $M \in \{0,1\}^*$ *is defined by*

$$H_{\mathcal{RO}\circ MD_{\mathcal{RO}}\circ X}(M) = NMAC_{F^{(n+m\to n)},G^{(n\to n)}}(R^{(*\to m\cdot L)}(M)).$$

(v) *Extension II: The hash function* $H_{X\circ\mathcal{RO}\circ MD_{\mathcal{RO}}} : \{0,1\}^{m\cdot L} \to \{0,1\}^n$ *for a padded message* $M \in \{0,1\}^{m\cdot L}$ *is defined by*

$$H_{X\circ\mathcal{RO}\circ MD_{\mathcal{RO}}}(M) = P^{(n\to n)}(NMAC_{F^{(n+m\to n)},G^{(n\to n)}}(M)).$$

Theorem 3. *Using the indifferentiability framework it must hold:*

(i) $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}}$ *is secure,*
(ii) $H_{\mathcal{RO}\circ MD_X}$ *is insecure,*
(iii) $H_{X\circ MD_{\mathcal{RO}}}$ *is insecure,*
(iv) $H_{\mathcal{RO}\circ MD_{\mathcal{RO}}\circ X}$ *is insecure.*
(v) $H_{X\circ\mathcal{RO}\circ MD_{\mathcal{RO}}}$ *is insecure,*

In [8], the plain Merkle-Damgård hash function with ideal compression functions $MD_{\mathcal{RO}}$ was shown to be insecure.

Proof. (i) Although this result was stated in [8], the proof seems to be missing in the published version of the paper. A proof is provided in Appendix B.

(ii) This proof is essentially the same as the proof of Theorem 1 (ii). We only have to take care for the Merkle-Damgård construction. Here, we only give the distinguisher D:
1. Choose a random message $M \in \{0,1\}^*$.
2. Compute $u = MD_P(M)$.
3. Ask the F-oracle for $v = F(u)$.
4. Ask the hash-oracle for $z = H(M)$.
5. If $z = v$ output ALGORITHM, else output RANDOM.

(iii) This proof is essentially the same as the proof of Theorem 1 (iii). We only have to take care for the Merkle-Damgård construction. Here, we only give the distinguisher D:
1. Choose a random message $M \in \{0,1\}^*$.
2. Ask the hash-oracle for $z = H(M)$.
3. Use the F-oracle to calculate $u = MD_F(M)$.
4. Compute $v = P(u)$.
5. If $z = v$ output ALGORITHM, else output RANDOM.

(iv) The proof is essentially the same as in 1 (iv). So if this hash function is secure, we could use our simulator to efficiently invert the padding function R.

(v) The proof is essentially the same as in 1 (iv). So if this hash function is secure, we could use our simulator to efficiently invert the CROWF G.

□

Paradox. Again, as discussed in Section 4, we get a secure hash function if we substitute the CROWF by an invertible function. For the hash functions (ii)-(v) similar results as were given in theorem 2 can easily be stated.

Part (iv) of the theorem might be (at least in this form) somewhat surprising. Most of the padding functions have the property of being easily invertible. But in the indifferentiability world this is a must-have feature for secure hash functions. If the padding function R is not efficiently invertible, $NMAC$ would be insecure.

In [8] the authors don't care with the padding function. But this turns out to be somewhat shortsighted in the case of indifferentiability secure hash functions. Even such a simple and (in the analysis phase) easily to be forgotten function can drive a hash function insecure if it is added.

We have also analyzed the Mix-Compress-Mix (MCM) construction given by Ristenpart et al. [25]. A short summary is given in the following theorem.

Theorem 4. *(i)* $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i}$ *is secure if* X *is a* $\Delta-$*regular function (*i.e., *every image of* H *has approximately the same number of preimages, for details see [25]).*

(ii) $H_{\mathcal{RO}_x \circ X \circ Y}$ *is insecure.*

(iii) $H_{Y \circ X \circ \mathcal{RO}_x}$ *is insecure.*

(iv) $H_{X \circ \mathcal{RO} \circ Y}$ *is insecure.*

(v) $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y}$ *is insecure.*

(vi) $H_{Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i}$ *is insecure.*

A proof and formal definitions are given in Appendix C.

6 Design Principles for Secure Hash Functions

Definition 5. *Let* $k \in \mathbb{N}$. *Let* $S_1 : \{0,1\}^* \to \{0,1\}^m$, $S_i : \{0,1\}^m \to \{0,1\}^m$, $(1 < i < k)$, $S_k : \{0,1\}^m \to \{0,1\}^n$, *be functions. The hash function* H *for a message* M *is defined by*

$$H(M) = (S_k \circ S_{k-1} \circ \ldots \circ S_1)(M)$$

Here, we generally don't care whether the functions S_i, $(1 \leq i \leq k)$ are ideal or not. Using the technique applied above it is easy to show

Theorem 5. *(Design Principles for Secure Hash Functions) Let* $\mathcal{H}$ *be defined as in Definition 5. Let* H *be a secure (indifferentiable) hash function. Then it must hold:*

(i) S_1 *is not a one way function.*

(ii) S_k *is not a one way function.*

Proof. (i) If S_1 is a one way function we could use the simulator (as H is secure) to invert S_1. The proof is essentially the same as the proof of Theorem 1 (ii).

(ii) If S_k is a one way function we could use the simulator (as H is secure) to invert S_k. The proof is essentially the same as the proof of Theorem 1 (iii).

This simple design principle is mandatory to all (indifferentiable) secure hash functions. Note that in the MCM construction, the one way function is in the middle of two random oracles. Applying Theorem 5 we can conclude: It is possible to prove a structure secure only if the 'first' and the 'last' functions are random oracles or easily invertible functions, but we might be able – under some circumstances – to choose some functions different to a random oracle. We do not see how this design principle for indifferentiable secure hash function will account for more security if constructing a practical hash function.

7 Discussion and Conclusion

The random oracle model. All ideal world notions and their definitions are inherently based on the random oracle model. Before going into details on indifferentiability itself in Section 7.1 let us recall some results from the literature on the random oracle model. As discussed in Section 1, there had been quite a few uninstantiability results, defining cryptosystems provably secure in the random oracle model, but insecure when instantiated by any efficient function. On can argue that all of these constructions are malicious. They are designed to be insecure. But either one relies on heuristics and intuitions, or one relies on proofs. If one puts proofs above all other aspects, then counter-examples do invalidate the proofs.

7.1 The Ambiguity of Indifferentiability in the Design of Practical Hash Functions

The ideas from Coron et al. [8] have been very influential and inspiring for a lot of researchers. Namely, there have been quite a few proposals for hash function structures provably "indifferentiable from a random oracle", often in addition to other security requirements as, *e.g.*, in [3, 5, 18].

But the current paper reveals a contradiction in the reasoning from [8]: The *same* formalism can be used to indicate the structural soundness of an implementation, and the presence of structural weaknesses.

The contradiction is not on the formal level – we do not claim any flaw in the theorems or proofs of [8]. If all components (*e.g.*, compression functions) of a hash function are ideal (*i.e.*, random oracles) we don't get ambiguous results. If all components are non-ideal we cannot use the indifferentiability framework to prove anything. But if some of the primitives are ideal and some are not (as for example in [3]) we can get ambiguous results for security proofs. Our research seems to indicate that the indifferentiability model is of limited use for proving the security of

- mixed-model hash functions (using complexity theoretic and ideal components at the same time) *and*
- practical hash functions (*e.g.* as the SHA-3 candidates).

One might conclude that if any possible description of a structure is insecure (as *e.g.*, is the case for Merkle-Damgård) in the indifferentiability framework then the hash structure is flawed. But it is not clear what we shall conclude for a concrete instantiation if one modeling is secure but another is not.
Taking a secure function (using only ideal components) we have shown in Section 4 and 5 how slight modifications (*i.e.*, adding a pre- or post-processing function) or partial instantiations (*i.e.*, starting our way towards an instantiated hash function) might possibly drive them insecure.

But, in addition to an inherent theoretical motivation, the notion of security in [8] has also been motivated by the need to decide if the structure of a hash function is sound or flawed. A criterion for good hash function structures is very valuable for hash function designers, indeed. On the strictly theoretical side, there is nothing wrong, and studying this kind of security remains an interesting topic for theoretical cryptography.

7.2 Conclusions

The random oracle model makes it possible to design cryptographic functions secure only in the ideal world. As discussed in Section 3, the notions of indistinguishability, preimage awareness and indifferentiability from a public-use random oracle seem to be too weak for designing a secure, practical and general purpose hash structure.

THE RIGHT LEVEL OF ABSTRACTION. If we state the discussion of Section 7.1 somewhat different we can come up with the following: For designing a hash function one might come up with a model/structure that describes the hash function on an abstract level. Then one might try to find a indifferentiability proof for this structure – given that some of the components are ideal. This process usually involves some sort of tweaking of the structure in order to 'find' the proof. Therefore we state that this structure is secure. But if we start with an implementation (*i.e.* a practical hash function) and want to assess its security in terms of indifferentiability, we are faced with the problem of the right level of abstraction/kind of modeling. If we abstract all the details and come up with a structure only consisting of a random oracle, *all* hash functions are trivially secure (again in terms of indifferentiability). If we abstract nothing, the indistinguishability framework does not have an answer to our question since we have no ideal components. But if we start abstracting some of the components we might be faced with the problem of finding some abstractions that are secure, and some that are not. And we might not know what to conclude for the security of the implementation.

OPEN PROBLEMS. It remains an open problem to derive an ideal world criterion to support the design of general purpose practical hash functions – telling us if the internal structure of a hash function is flawless or not. Certainly, a security proof (*i.e.*, a proof of a hash function being indifferentiable from a random oracle, when modeling some or all the internal functions as random oracles) is comforting. But pursuing this kind of security property requires great care since authors of a new hash function could be tempted to change, *e.g.*, some one-way final transform of their hash function into an easily invertible transformation. This could enable a theoretical security proof in the first place, while, at the same time, practically weaken the hash function.

Designers of practical hash functions, who accept the indifferentiability framework at face value, may be tempted to make poor design decisions. The indifferentiability framework suggests corrections to structures which sometimes make only sense in the ideal world but that have no real-word mapping. Even worse, the danger is that these very corrections drive the corresponding real-world hash function less secure.

Authors of new hash functions are well advised to prove other security properties, such as the established collision-, preimage-, and second-preimage-resistance under some reasonable standard-model assumptions, perhaps in addition to proving theoretical security properties, such as the indifferentiability from a random oracle.

References

1. Elena Andreeva, Charles Bouillaguet, Pierre-Alain Fouque, Jonathan J. Hoch, John Kelsey, Adi Shamir, and Sébastien Zimmer. Second preimage attacks on dithered hash functions. In Smart [28], pages 270–288.
2. Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2004.
3. Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the emd transform. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
5. Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Smart [28], pages 181–197.

6. Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.
7. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
8. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-damgård revisited: How to construct a hash function. In Shoup [27], pages 430–448.
9. Ivan Damgård. A design principle for hash functions. In Brassard [6], pages 416–427.
10. Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Shoup [27], pages 449–466.
11. Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging merkle-damgård for practical applications. In Antoine Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 371–388. Springer, 2009.
12. Sha Goldwasser and Yael Taumann. On the (in)security of the fiat-shamir paradigm. In *In FOCS 2003*, pages 102–115. IEEE Computer Society Press, 2003.
13. Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.
14. John Kelsey and Tadayoshi Kohno. Herding hash functions and the nostradamus attack. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.
15. John Kelsey and Bruce Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.
16. Lars R. Knudsen. Hash Functions and SHA-3. Invited Talk at FSE 2008, available at `http://fse2008.epfl.ch/docs/slides/day_1_sess_2/Knudsen-FSE2008.pdf`, 2008.
17. Anja Lehmann and Stefano Tessaro. A modular design for hash functions: Towards making the mix-compress-mix approach practical. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 364–381. Springer, 2009.
18. Moses Liskov. Constructing an ideal hash function from weak ideal compression functions. In Eli Biham and Amr M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 358–375. Springer, 2006.
19. Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2004.
20. Ralph C. Merkle. One way hash functions and des. In Brassard [6], pages 428–446.
21. Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.
22. NIST National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard. April 1995. See http://csrc.nist.gov.
23. NIST National Institute of Standards and Technology. FIPS 180-2: Secure Hash Standard. April 1995. See http://csrc.nist.gov.
24. NIST National Institute of Standards and Technology. Tentative Timeline of the Development of New Hash Functions. `http://csrc.nist.gov/pki/HashWorkshop/timeline.html`.
25. Thomas Ristenpart and Thomas Shrimpton. How to build a hash function from any collision-resistant function. In *ASIACRYPT*, pages 147–163, 2007.
26. Ron L. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm.* Internet Activities Board, April 1992.
27. Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.
28. Nigel P. Smart, editor. *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*. Springer, 2008.
29. Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC Press, 1995.

A Proof of Security for $H_{\mathcal{RO}-\mathcal{RO}}$

Proof of Theorem 1

Proof. Let $H := (\mathcal{RO}_2 \circ \mathcal{RO}_1)(M)$ be the definition of the hash function. We have to describe an efficient simulator S who is able to emulate the random oracles $\mathcal{RO}_1$ and $\mathcal{RO}_2$. The simulator has access to the hash oracle $H_{\mathcal{RO}}$.
Description of the Simulator S:

1. $\mathcal{RO}_1$ oracle queries: For all queries we perform record keeping. If we have answered the same query before we return the same value again. Else we choose a random value and add it to our database $DB \overset{add}{\longleftarrow} [query, random]$.
2. $\mathcal{RO}_2$ oracle queries: If $[?, query] \in DB$, then use the first entry to ask the hash oracle $H_{\mathcal{RO}}$ and return the answer. Else choose a random value and add it to our database $DB \overset{add}{\longleftarrow} [random, query]$. Use the new chosen random value to ask the hash oracle $H_{\mathcal{RO}}$ and return the answer.

Clearly, S is efficient and any distinguisher D cannot differentiate it from a random oracle.

B Proof of Security for the NMAC Construction

Proof. Our task is to describe an efficient simulator S which can emulate the oracles F and G in the RANDOM case. Namely, we we will

- give a description of S,
- show that S is efficient and
- show that the probability that an efficient distinguisher is successful (in differentiating this RANDOM case from the ALGORITHM case) is negligible.

Description of the Simulator S.

1. F **oracle queries:** In order to simulate $H' = F(H, M)$ we use the natural approach:
   - If we have answered the same query before we return the same value H' again.
   - Else we choose the answer H' to the queries randomly.

   But we have to take care that the randomly chosen answer must not be equal to an answer of any prior query. If this does happen, we call this event BAD:

   Definition 6. BAD *represents the event that a previously used value is generated as the random answer to a later query.*

   If BAD has happened, it follows that a message chain is indicated where, with overwhelming probability, should be none. The probability for BAD is negligible as will be shown later on.
   For all oracle queries we perform record keeping. There are two reasons for this: First to detect when BAD happens and, second, we need to know the values to answer the G oracle queries. As a result we have a record of numerous chains

   $$[H_0, M_{k_1}] \mapsto [H_{k_1}, M_{k_2}] \mapsto \ldots \mapsto [H_{k_l}, \perp].$$

   We call H_{k_j} accessible if there exists a chain starting with an initial value H_0, $H_{k_j} \neq H_{k_0}$, that contains H_{k_j}.
   Remarks: All H_{k_j} are different (or BAD would have occurred). Initially there does exist one chain for the initialization value: $[H_0, \perp]$.

2. G **oracle queries:** For simulating oracle queries for $H' = G(H)$ we consider two cases:
   (a) H is not accessible: choose a random value $H' \in \{0,1\}^n$. Add a new chain $[H', \perp]$ into the record and return H'.
   (b) H is accessible: There exists one specific chain that contains H. Therefore we can extract the related message $M = (M_{k_1}, \ldots M_{k_j})$ from the chain and forward it the the random oracle $\mathcal{H}_{\mathrm{Rnd}}$. We return $H' = \mathcal{H}_{\mathrm{Rnd}}(M)$.

Efficiency. The operations are clearly efficient. They all can be performed using standard techniques.

Success probability of the distinguisher. We will now show that, if the event BAD does not occur, our simulator fools the adversary by proving that the distribution of the adversary's output in the ALGORITHM case (without simulator S) is identical to the distribution of the adversary's output in the RANDOM case (with simulator S). Afterwards we will show that the probability of BAD is negligible. This concludes the proof.

Lemma 1. *The simulator S only aborts if* BAD *occurs.*
Proof: This is directly implied by the description of the simulator S. (□)

Lemma 2. *If the simulator does not abort the adversary's view of* ALGORITHM *and* RANDOM *is identical.*
Proof:

1. RANDOM: Only independent random values are returned. Either by a random oracle, the hash function, or random values chosen by the simulator (simulating F and G as being random oracles). If we have answered the same query before we return the same value again.
2. ALGORITHM: Here F and G are random oracles and, consequently, also the hash function.

In either case the distinguisher can only see random values. (□)

Lemma 3. *The probability that* BAD *occurs is negligible.*
Proof: If BAD has not happened after the first q queries to the simulator (either for F or G, then the probability that it happens on the $(q+1)$-st query is at most $(q+1) \cdot 2^{-n}$. This is because there are at most $(q+1)$ answers including the initial value H_0. Therefore, if the distinguisher makes a total of q queries, the probability of the event BAD is at most $(q^2+q) \cdot 2^{-n}$. (□)

This completes the proof. □

C Mix-Compress-Mix $\mathcal{RO}_i - X - \mathcal{RO}_i$

Note that there was given a more efficient instantiation at ASIACRYPT'09 by Lehmann and Tessaro [17].

Definition 7. *Fix numbers $\eta > 0$, $\tau \geq 0$, $L > 0$ and a hash key $K = (k_1, k, k_2)$. Let*

$$\begin{aligned} \varepsilon_1(k_1, \cdot) &: \{0,1\}^{\leq L} \to \{0,1\}^{\leq L+\tau}, \\ H(k, \cdot) &: \{0,1\}^{\leq L+\tau} \to \{0,1\}^{\eta}, \\ \varepsilon_2(k_2, \cdot) &: \{0,1\}^{\eta} \to \{0,1\}^{\eta+\tau}, \end{aligned}$$

whereas ε_1 has stretch τ, i.e., *$|\varepsilon_1(k_1, M)| = |M| + \tau$. The functions ϵ_1 and ϵ_2 are injective. The hash function $MCM_{\epsilon_1,H,\epsilon_2} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a fixed key $K = (k_1, k, k_2)$ is computed by*

$$MCM_{\epsilon_1,H,\epsilon_2}(M) = \epsilon_2(k_2, H(k, \epsilon_1(k_1, M))).$$

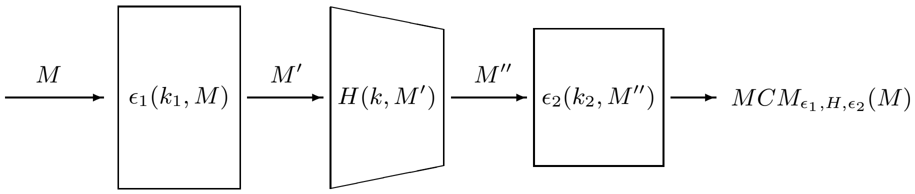

Figure 3. The MCM construction $MCM_{\epsilon_1,H,\epsilon_2}(M)$. H is a CROWF, ϵ_1, ϵ_2 are injective ("*mixing*") functions, $K = (k_1, k, k_2)$ is the previously fixed hash key.

Definition 8. *An index i denotes an injective function/oracle, an index x denotes that we explicitly don't care whether the function/oracle is injective or not.*

(i) The hash function $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a message $M \in \{0,1\}^{\leq L}$ is defined by

$$H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i}(M) = MCM_{F_i^{(\leq L \to \leq L+\tau)}, D^{(\leq L+\tau \to \eta)}, G_i^{(\eta \to \eta+\tau)}}(M)$$

whereas F and G both have stretch τ.

(ii) Modification/Partial Instantiation I: The hash function $H_{\mathcal{RO}_x \circ X \circ Y} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a message $M \in \{0,1\}^{\leq L}$ is defined by

$$H_{\mathcal{RO}_x \circ X \circ Y}(M) = MCM_{F_i^{(\leq L \to \leq L+\tau)}, D^{(\leq L+\tau \to \eta)}, E_i^{(\eta \to \eta+\tau)}}(M)$$

whereas F and E both have stretch τ.

(iii) Modification/Partial Instantiation II: The hash function $H_{Y \circ X \circ \mathcal{RO}_x} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a message $M \in \{0,1\}^{\leq L}$ is defined by

$$H_{Y \circ X \circ \mathcal{RO}_x}(M) = MCM_{E_i^{(\leq L \to \leq L+\tau)}, D^{(\leq L+\tau \to \eta)}, G_i^{(\eta \to \eta+\tau)}}(M)$$

whereas E and G both have stretch τ.

(iv) Modification III: The hash function $H_{X \circ \mathcal{RO} \circ Y} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a message $M \in \{0,1\}^{\leq L}$ is defined by

$$H_{X \circ \mathcal{RO} \circ Y}(M) = MCM_{D^{(\leq L \to \leq L+\tau)}, F^{(\leq L+\tau \to \eta)}, E^{(\eta \to \eta+\tau)}}(M)$$

whereas D and E both have stretch τ.

(v) Extension I: The hash function $H_{Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ for a message $M \in \{0,1\}^{\leq L}$ is defined by

$$H_{Y \circ \mathcal{RO}_i \circ X \circ \mathcal{RO}_i}(M) = E^{(\eta+\tau \to \eta+\tau)}(MCM_{F_i^{(\leq L \to \leq L+\tau)}, D^{(\leq L+\tau \to \eta)}, G_i^{(\eta \to \eta+\tau)}}(M))$$

whereas F and G both have stretch τ.

(vi) Extension II: The hash function $H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y} : \{0,1\}^{\leq L} \to \{0,1\}^{\eta+\tau}$ *for a message* $M \in \{0,1\}^{\leq L}$ *is defined by*

$$H_{\mathcal{RO}_i \circ X \circ \mathcal{RO}_i \circ Y}(M) = MCM_{F_i^{(\leq L \to \leq L+\tau)}, D^{(\leq L+\tau \to \eta)}, G_i^{(\eta \to \eta+\tau)}}(E^{(\leq L \to \leq L)}(M))$$

whereas F_i *and* G_i *both have stretch* τ*.*

Proof. (i) This proof is given in [25, Theorem 3.2]. Note that the condition of injective random oracles is only needed for their special conclusion (proving collision resistance in the standard model) – it is not compulsory to prove the MCM construction secure in the indifferentiability framework.
(ii) The proof is essentially the same as in Theorem 1 (ii).
(iii) The proof is essentially the same as in Theorem 1 (iii).
(iv) This can be proved either by the proof of Theorem 1 (ii) or (iii)
(v) The proof is essentially the same as in Theorem 1 (ii).
(vi) The proof is essentially the same as in Theorem 1 (iii).

D Uninstantiability

As we are in the random oracle model we could easily give a hash function that is provably secure in the random oracle world (*i.e.*, indistinguishable and indifferentiable).

Definition 9. *Let*

$$\mathcal{RO}_1 : \{0,1\}^* \to \{0,1\}^n, \tag{1}$$
$$\mathcal{RO}_2, \mathcal{RO}_3 : \{0,1\}^n \to \{0,1\}^n, \tag{2}$$

be random oracles and the function

$$Q : \{0,1\}^n \to \{0,1\}^n \tag{3}$$

be defined as follows:

Let $m \in \{0,1\}^n$ *be the parameter of the call. Check if* $\mathcal{RO}_2$ *is a random oracle (for details see [7]):*
yes: *return* m*,*
no: *return 1.*

The hash function H *is defined by*

$$H(M) = (\mathcal{RO}_3 \circ Q \circ \mathcal{RO}_1)(M) \tag{4}$$

Theorem 6. *Let* H *be as in Definition 9.* H *is a secure (indistinguishable and indifferentiable) hash function.*

Proof. In the random oracle model the hash function H simply reduces to the concatenation of two random oracles. By Theorem 1 (i) we can easily prove our hash function secure. If $\mathcal{RO}_2$ is no random oracle, our hash function always outputs the same hash value and is consequently completely insecure. □