Merkle-Damgård Revisited : how to Construct a Hash Function

Jean-Sébastien Coron[1], Yevgeniy Dodis[⋆2], Cécile Malinaud[3], and Prashant Puniya[⋆⋆2]

[1] University of Luxembourg, coron@clipper.ens.fr
[2] New-York University, {dodis,puniya}@cs.nyu.edu
[3] Gemplus Card International, cecile.malinaud@normalesup.org

Abstract. The most common way of constructing a hash function (*e.g.*, SHA-1) is to iterate a compression function on the input message. The compression function is usually designed from scratch or made out of a block-cipher. In this paper, we introduce a new security notion for hash-functions, stronger than collision-resistance. Under this notion, the arbitrary length hash function H must behave as a random oracle when the fixed-length building block is viewed as a random oracle or an ideal block-cipher. The key property is that if a particular construction meets this definition, then any cryptosystem proven secure assuming H is a random oracle remains secure if one plugs in this construction (still assuming that the underlying fixed-length primitive is ideal). In this paper, we show that the current design principle behind hash functions such as SHA-1 and MD5 — the (strengthened) Merkle-Damgård transformation — does not satisfy this security notion. We provide several constructions that provably satisfy this notion; those new constructions introduce minimal changes to the plain Merkle-Damgård construction and are easily implementable in practice.

1 Introduction

Random Oracle Methodology. The random oracle model has been introduced by Bellare and Rogaway as a "paradigm for designing efficient protocols" [4]. It assumes that all parties, including the adversary, have access to a public, truly random hash function H. This model has been proven extremely useful for designing simple, efficient and highly practical solutions for many problems. From a theoretical perspective, 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, such as SHA-1 [16] or MD5 [18]. In fact, many recent "separation" results [11, 26, 19, 2, 12, 15] illustrated various cryptographic systems secure in the random oracle model but completely insecure for *any* concrete instantiation of the random oracle (even by a *family* of

⋆ Supported by NSF CAREER Award CCR-0133806 and TC Grant No. CCR-0311095.
⋆⋆ Supported by NSF Cybertrust/DARPA Grant No. CNS-0430425.

hash functions). Nevertheless, these important separation results do not seem to directly attack any of the concrete, widely used cryptosystems (such as OAEP [6] and PSS [5] as used in the PKCS #1 v2.1 standard [27]) which rely on "secure hash functions". Moreover, we hope that such particular systems are in fact *secure when instantiated with a "good" hash function.* In the random oracle model, instead of making a highly non-standard (and possibly unsubstantiated) assumption that "my system is secure with this H" (e.g., H being SHA-1), one proves that the system is at least secure with an "ideal" hash function H (under standard assumptions). Such formal proof in the random oracle model is believed to indicate that there are no structural flaws in the design of the system, and thus one can heuristically hope that no such flaws will suddenly appear with a particular, "well designed" function H. *But can we say anything about the lack of structural flaws in the design of H itself?*

Building Random Oracles. On the first glance, it appears that nothing theoretically meaningful can be said about this question. Namely, we know that mathematically a concrete function H is not a random oracle, so to prove that H is "good" we need to directly argue the security of our system with this given H. And the latter task is usually unmanageable given our current tools (e.g., "realizable" properties of H such as collision-resistance, pseudorandomness or one-wayness are usually not enough to prove the security of the system). However, we argue that there is a significant gap in this reasoning. Indeed, most systems abstractly model H as a function from $\{0,1\}^*$ to $\{0,1\}^n$ (where n is proportional to the security parameter), so that H can be used on some arbitrary input domain. On the other hand, in practice such *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. For example, SHA-1, MD5, as well as all the other hash function we know of, are constructed by applying some variant of the Merkle-Damgård construction to an underlying compression function $f: \{0,1\}^{n+\kappa} \to \{0,1\}^n$ (see Figure 5):

Function $H(m_1, \ldots, m_\ell)$:
  let $y_0 = 0^n$ (more generally, some fixed IV value can be used)
  for $i = 1$ to ℓ do $y_i \leftarrow f(y_{i-1}, m_i)$
  return y_ℓ

When the number of κ-bit message blocks ℓ is not fixed, one essentially appends an extra block $m_{\ell+1}$ containing the binary representation $\langle |m| \rangle$ of the length of the message (prepended by 1 and a string of 0's in order to make everything a multiple of κ; the exact details will not matter for our discussion). The fixed-length compression function f can either be constructed from scratch or made out of a block-cipher E via the Davies-Meyer construction (see [31] and Figure 9): $f(x, y) = E_y(x) \oplus x$. For example, the SHA-1 compression function was designed specifically for hashing, but a block-cipher can nevertheless be derived from it, as illustrated in [20].

Our Main Question. Given such particular and "structured" design of our hash function H,— which is actually the design used in practice,— we argue that there exists a missing link in the claim that no structural flaws exist in the design of our system. Indeed, we only know that no such flaws exist when H was modeled as a "monolithic" random oracle, and not as an iterated hash function built from some smaller building block. As since the real implementation of H as an iterated hash function has much more structure than a random monolithic hash function would have, maybe this structure could somehow invalidate the security proof in the random oracle model? To put this into a different perspective, all the ad-hoc (and hopefully "secure") design effort for widely used hash functions, such as SHA-1 and MD5, has been placed into the design of the fixed-length building block f (or E). On the other hand, even if f (or E) were assumed to be ideal, the current proofs in the random oracle model do not guarantee the security of the resulting system when such iterated hash function H is used!

Let us illustrate our point on a well known example. A common suggestion to construct a MAC algorithm is to simply include a secret key k as part of the input of the hash function, and take for example $\mathrm{MAC}(k, m) = H(k\|m)$. It is easy to see that this construction is secure when H is modeled as a random oracle [4], as no adversary can output a MAC forgery except with negligible probability. However, this MAC scheme is completely insecure for any Merkle-Damgård construction considered so far (including Merkle-Damgård strengthening used in current hash functions such as SHA-1, and any of the 64 block-cipher based variants of iterative hash-functions considered in [29, 9]), no matter which (ideal) compression function f (or a block cipher E) is used. Namely, given $\mathrm{MAC}(k, m) = H(k\|m)$, one can extend the message m with any single arbitrary block y and deduce $\mathrm{MAC}(k, m\|y) = H(k\|m\|y)$ without knowing the secret key k (even with Merkle-Damgård strengthening, one could still forge the MAC by more or less setting $y = \langle |m| \rangle$, where the actual block depends on the exact details of the strengthening). This (well known) example illustrates that the construction of a MAC from an iterated hash function requires a specific analysis, and cannot be derived from the security of this MAC with a monolithic hash function H. On the other hand, while the Merkle-Damgård transformation and its variants have been intensively studied for many "realizable" properties such as collision-resistance [13, 25, 29, 9], pseudorandomness [8], unforgeability [1, 24] and randomness extraction [14], it is clear that these analyses are insufficient to argue its applicability for the purposes of building a hash function which can be modeled as a random oracle, since the latter is a considerably stronger security notion (in fact unrealizable in the standard model). For a simple concrete example, the Merkle-Damgård strengthening is easily seen to preserve collision-resistance when instantiated with a collision-resistant compression function, while we just saw that it does not work to yield a random oracle or even just a variable-length MAC, and this holds even if the underlying compression function is modeled as a random oracle.

Our Goals. Summarizing the above discussion, our goal is two-fold. First, we would like to give a formal *definition* of what it means to implement an arbitrary-length random oracle H from a fixed-length building block f or E. The key property of this definition should be the fact that if a particular construction of H from f (or E) meets this definition, then *any application* proven secure assuming H is a random oracle would remain secure if we plug in our construction (although still assuming that the underlying fixed-length primitive f or E was ideal). In other words, we can safely use our implementation of H as if we were using a monolithic random oracle H. We remark that this means that our definition should not just preserve the pseudorandomness properties of H, but also all the other "tricks" present in the random oracle model, such as "programmability" and "extractability". For example, we could try to set $H(x) = f(h(x))$, where f is a fixed-length random oracle and h is a collision-resistant hash function (not viewed as a random oracle). While pseudorandom, this simple implementation is clearly not "extractable": for example, given output $z = f(h(x))$ for some unknown x, we can only "extract" the value $h(x)$ (by observing the random oracle queries made to f), but then have no way of extracting x itself from $h(x)$ (indeed, we will show a direct attack on this implementation in Section 3.1). This shows that the security definition we need is an interesting and non-trivial task of its own, especially if we also want it to be simple, natural and easy to use.

Second, while the definition we seek should not be too specific to some variant of the Merkle-Damgård transformation, we would like to give secure constructions which resemble what is done in practice as much as possible. Unfortunately, we already argued that the current design principle behind hash functions such as SHA-1 and MD5 — the (strengthened) Merkle-Damgård transformation — will *not* be secure for our ambitious goal. Therefore, instead of giving new and practically unmotivated constructions, our secondary goal is to come up with *minimal* and *easily implementable in practice* changes to the plain Merkle-Damgård construction, which would satisfy our security definition.

Our results. First, we give a satisfactory definition of what it means to implement an arbitrary-length random oracle H from a fixed length primitive g (where g is either an ideal compression function f, or a an ideal block cipher E). Our definition is based on the indifferentiability framework of Maurer et al. [23]. This framework enjoys the desired closure property we seek, and is very intuitive and easy to state.

Having a good security definition, we provide several provable constructions. We start by giving three modifications to the (insecure) plain Merkle-Damgård construction which yield a secure random oracle H taking *arbitrary-length* input, from a compression function viewed as a random oracle taking *fixed-length* input. This result can be viewed as a secure *domain extender* for the random oracle, which is an interesting result of independent interest. We remark that domain extenders are well studied for such primitives as collision-resistant hash functions [13, 25], pseudorandom functions [8], MACs [1, 24] and universal one-way hash functions [7, 30]. Although the above works also showed that some variants of Merkle-Damgård yield secure domain extenders for the corresponding primitive in question, these results are not sufficient to claim a domain extender for the random oracle.

Our secure modifications to the plain Merkle-Damgård construction are the following. (1) *Prefix-Free Encoding* : we show that if the inputs to the plain MD construction are guaranteed to be *prefix-free*, then the plain MD construction is secure. (2) *Dropping Some Output Bits* : we show that by dropping a non-trivial number of output bits from the plain MD chaining, we get a secure random oracle H even if the input is not encoded in the prefix-free manner. (3) *Using NMAC construction* (see Figure 8a): we show that by applying an independent hash function g to the output of the plain MD chaining (as in the NMAC construction [8]), then once again we get a secure construction of an arbitrary-length random oracle H, in the random oracle model for f and g. (4) *Using HMAC Construction* (see Figure 8b): we show a slightly modified variant of the NMAC construction allowing us to conveniently build the function g from the compression function f itself (as in [8] when going from NMAC to HMAC)! In this latter variant, one implements a secure hash function H by making two *black-box calls to the plain Merkle-Damgård construction* (with the same fixed IV and a given compression function f): first on $(\ell+1)$-block input $0^\kappa m_1 \ldots m_\ell$, getting an n-bit output y, and then on one-block κ-bit input y' (obtained by either truncating or padding y depending on whether or not $\kappa > n$), getting the final output.

However, in practice most hash-function constructions are block-cipher based, either explicitly as in [29] or implicitly as for SHA-1. Therefore, we consider the question of designing an arbitrary-length random oracle H from an ideal block cipher E, specifically concentrating on using the Merkle-Damgård construction with the Davies-Meyer compression function $f(x, y) = E_y(x) \oplus x$, since this is the most practically relevant construction. We show that all of the four fixes to the plain MD chaining which worked when f was a fixed-length random oracle, are still secure (in the ideal cipher model) when we plug in $f(x, y) = E_y(x) \oplus x$ instead. Specifically, we can either use a prefix-free encoding, or drop a non-trivial number of output bits (when possible), or apply an independent random oracle g to the output of plain MD chaining, or use the optimized HMAC construction which allows us to build this function g from the ideal cipher itself.

2 Definitions

In this section, we introduce the main notations and definitions used throughout the paper. Our security notion for secure hash-function is based on the notion of indifferentiability of systems, introduced by Maurer *et al.* in [23]. This is an extension of the classical notion of indistinguishability, when one or more oracles are publicly available, such as random oracles or ideal ciphers. This notion is based on ideas from the Universal Composition framework introduced by Canetti in [10] and on the model of Pfitzmann and Waidner [28]. The indifferentiability notion in [23] is given in the framework of random systems providing interfaces to other systems, but equivalently we use this notion in the framework of Interactive Turing Machines (as in [10]).

We define an *ideal primitive* as an algorithmic entity which receives inputs from one of the parties and deliver its output immediately to the querying party. The ideal primitives that we consider in this paper are random oracles and ideal ciphers. A *random oracle* [4] is an ideal primitive which provides a random output for each new query. Identical input queries are given the same answer. An *ideal cipher* is an ideal primitive that models a random block-cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$. Each key $k \in \{0,1\}^\kappa$ defines a random permutation $E_k = E(k, \cdot)$ on $\{0,1\}^n$. The ideal primitive provides oracle access to E and E^{-1}; that is, on query $(0, k, m)$, the primitive answers $c = E_k(m)$, and on query $(1, k, c)$, the primitive answers m such that $c = E_k(m)$.

We now proceed to the definition of indifferentiability [23] :

Definition 1. *A Turing machine C with oracle access to an ideal primitive $\mathcal{G}$ is said to be $(t_D, t_S, q, \varepsilon)$ indifferentiable from an ideal primitive $\mathcal{F}$ if there exists a simulator S, such that for any distinguisher D it holds that :*

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

The simulator has oracle access to $\mathcal{F}$ and runs in time at most t_S. The distinguisher runs in time at most t_D and makes at most q queries. Similarly, $C^{\mathcal{G}}$ is said to be (computationally) indifferentiable from $\mathcal{F}$ if ε is a negligible function of the security parameter k (for polynomially bounded t_D and t_S).

As illustrated in Figure 1, the role of the simulator is to simulate the ideal primitive $\mathcal{G}$ so that no distinguisher can tell whether it is interacting with C and $\mathcal{G}$, or with $\mathcal{F}$ and S; in other words, the output of S should look "consistent" with what the distinguisher can obtain from $\mathcal{F}$. Note that the simulator does not see the distinguisher's queries to $\mathcal{F}$; however, it can call $\mathcal{F}$ directly when needed for the simulation.

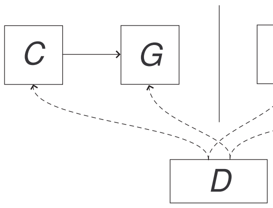

Fig. 1. The indifferentiability notion: the distinguisher D either interacts with algorithm C and ideal primitive $\mathcal{G}$, or with ideal primitive $\mathcal{F}$ and simulator S. Algorithm C has oracle access to $\mathcal{G}$, while simulator S has oracle access to $\mathcal{F}$.

In the rest of the paper, the algorithm C will represent the construction of an iterative hash-function (such as the Merkle-Damgård construction recalled in the introduction). The ideal primitive $\mathcal{G}$ will represent the underlying primitive used to build the hash-function. $\mathcal{G}$ will be either a random oracle (when the compression function is modelled as a random oracle), or an ideal block-cipher (when the compression function is based on a block-cipher). The ideal primitive $\mathcal{F}$ will represent the random oracle that the construction C should emulate. Therefore, one obtains the following setting : the distinguisher has oracle access to both the block-cipher and the hash-function, and these oracles are implemented in one of the following two ways: either the block-cipher E is chosen at random and the hash-function C is constructed from it, or the hash-function H is chosen at random and the block-cipher is implemented by a simulator S with oracle access to H. Those two cases should be indistinguishable, that is the distinguisher should not be able to tell whether the block-cipher was chosen at random and the iterated hash-function constructed from it, or the hash-function was chosen at random and the block-cipher then "tailored" to match that hash-function.

It is shown in [23] that if $C^{\mathcal{G}}$ is indifferentiable from $\mathcal{F}$, then $C^{\mathcal{G}}$ can replace $\mathcal{F}$ in any cryptosystem, and the resulting cryptosystem is at least as secure in the $\mathcal{G}$ model as in the $\mathcal{F}$ model. For example, if a block-cipher based iterative hash function is indifferentiable from a random oracle in the ideal cipher model, then the iterative hash-function can replace the random oracle in any cryptosystem, and the resulting cryptosystem remains secure in the ideal cipher model if the original scheme was secure in the random oracle model.

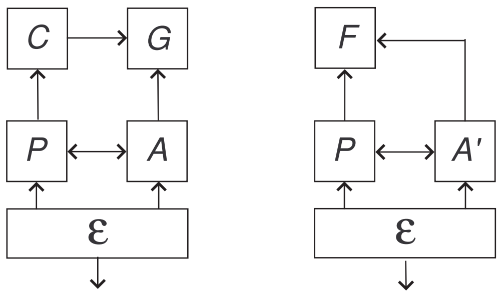

Fig. 2. The environment $\mathcal{E}$ interacts with cryptosystem $\mathcal{P}$ and attacker $\mathcal{A}$. In the $\mathcal{G}$ model (left), $\mathcal{P}$ has oracle access to C whereas $\mathcal{A}$ has oracle access to $\mathcal{G}$. In the $\mathcal{F}$ model, both $\mathcal{P}$ and $\mathcal{A}'$ have oracle access to $\mathcal{F}$

We use the definition of [23] to specify what it means for a cryptosystem to be at least as secure in the $\mathcal{G}$ model as in the $\mathcal{F}$ model. A cryptosystem is modelled as an Interactive Turing Machine with an interface to an adversary $\mathcal{A}$ and to a public oracle. The cryptosystem is run by an *environment* $\mathcal{E}$ which provides a binary output and also runs the adversary. In the $\mathcal{G}$ model, cryptosystem $\mathcal{P}$ has oracle access to C whereas attacker $\mathcal{A}$ has oracle access to $\mathcal{G}$. In the $\mathcal{F}$ model, both $\mathcal{P}$ and $\mathcal{A}$ have oracle access to $\mathcal{F}$. The definition is illustrated in Figure 2.

Definition 2. *A cryptosystem is said to be at least as secure in the $\mathcal{G}$ model with algorithm C as in the $\mathcal{F}$ model, if for any environment $\mathcal{E}$ and any attacker $\mathcal{A}$ in the $\mathcal{G}$ model, there exists an attacker $\mathcal{A}'$ in the $\mathcal{F}$ model, such that*

$$\left|\Pr\left[\mathcal{E}(\mathcal{P}^{\mathcal{C}}, \mathcal{A}^{\mathcal{G}}) = 1\right] - \Pr\left[\mathcal{E}(\mathcal{P}^{\mathcal{F}}, \mathcal{A}'^{\mathcal{F}}) = 1\right]\right|$$

is a negligible function of the security parameter k. Similarly, a cryptosystem is said to be computationally at least as secure, etc., if $\mathcal{E}$, $\mathcal{A}$ and $\mathcal{A}'$ are polynomial-time in k.

The following theorem from [23] shows that security is preserved when replacing an ideal primitive by an indifferentiable one :

Theorem 1. *Let $\mathcal{P}$ be a cryptosystem with oracle access to an ideal primitive $\mathcal{F}$. Let C be an algorithm such that $C^{\mathcal{G}}$ is indifferentiable from $\mathcal{F}$. Then cryptosystem $\mathcal{P}$ is at least as secure in the $\mathcal{G}$ model with algorithm C as in the $\mathcal{F}$ model.*

Proof. We only provide a proof sketch; see [23] for a full proof. Let $\mathcal{P}$ be any cryptosystem, modelled as an Interactive Turing Machine. Let $\mathcal{E}$ be any environment, and $\mathcal{A}$ be any attacker in the $\mathcal{G}$ model. In the $\mathcal{G}$ model, $\mathcal{P}$ has oracle access to C whereas $\mathcal{A}$ has oracle access to ideal primitive $\mathcal{G}$; moreover environment $\mathcal{E}$ interacts with both $\mathcal{P}$ and $\mathcal{A}$. This is illustrated in Figure 3 (left part).

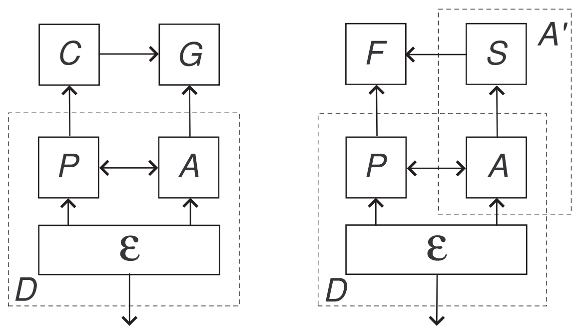

Fig. 3. Construction of attacker $\mathcal{A}'$ from attacker $\mathcal{A}$ and simulator $\mathcal{S}$.

Since $C^{\mathcal{G}}$ is indifferentiable from $\mathcal{F}$ (see Figure 1), one can replace $(C, \mathcal{G})$ by $(\mathcal{F}, S)$ with only a negligible modification of the environment's output distribution. As illustrated in Figure 3, by merging attacker $\mathcal{A}$ and simulator $\mathcal{S}$, one obtains an attacker $\mathcal{A}'$ in the $\mathcal{F}$ model, and the difference in $\mathcal{E}$'s output distribution is negligible. □

3 Domain Extension for Random Oracles

In this section, we show how to construct an iterative hash-function indifferentiable from a random oracle, from a compression function viewed as a random oracle. We start with two simple and intuitive constructions that do not work.

3.1 $H(x) = f(h(x))$ for Random Oracle f and Collision-Resistant One-way Hash-function h

One could hope to emulate a random oracle (with arbitrary-length input) by taking :

$$C^f(x) = f(h(x))$$

where $f : \{0,1\}^n \to \{0,1\}^n$ is modelled as a random oracle and $h : \{0,1\}^* \to \{0,1\}^n$ is any collision-resistant one-way hash-function (not modelled as a random oracle). However, we show that such C^f is not indifferentiable from a random oracle; namely, we construct a distinguisher that can fool any simulator.

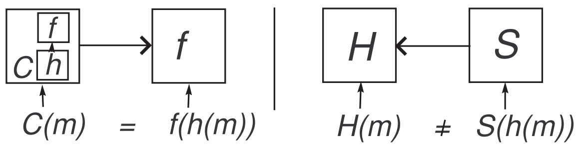

Fig. 4. The simulator cannot output $H(m)$ since it only receives $h(m)$ and cannot recover m from $h(m)$.

As illustrated in Figure 4, the distinguisher first generates an arbitrary m and computes $u = h(m)$. Then it queries $v = f(u)$ to random oracle f and queries $z = C^f(m)$ to C^f. It then checks that $z = v$ and outputs 1 in this case, and 0 otherwise. It is easy to see that the distinguisher always output 1 when interacting with C^f and f, but outputs 0 with overwhelming probability when interacting with H and any simulator S. Namely, when the distinguisher interacts with H and S, the simulator only receives $u = h(m)$; therefore, in order to output v such that $v = H(m)$, the simulator must either recover m from $h(m)$ (and then query $H(m)$) or guess the value of $H(m)$, which can be done with only negligible probability.

3.2 Plain Merkle-Damgård Construction

We show that the plain Merkle-Damgård construction (see Figure 5) fails to emulate a random oracle (taking arbitrary-length input) when the compression function f is viewed as a random oracle (taking fixed-length input). For simplicity, we only consider the usual Merkle-Damgård variant, although the discussion easily extends to the strengthened variant which appends the message length $\langle |m| \rangle$ at the last block :

Function $\mathrm{MD}^f(m_1, \ldots, m_\ell)$:
let $y_0 = 0^n$ (more generally, some fixed IV value can be used)
for $i = 1$ to ℓ do $y_i \leftarrow f(y_{i-1}, m_i)$
return $y_\ell \in \{0,1\}^n$.

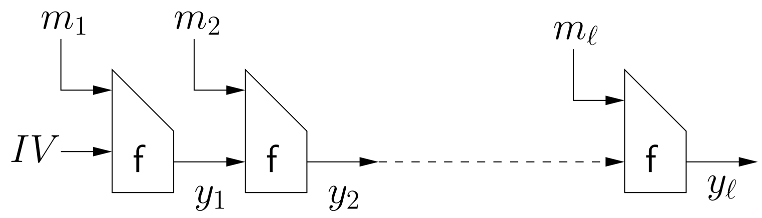

Fig. 5. The plain Merkle-Damgård Construction

where for all i, $|m_i| = \kappa$ and $f : \{0,1\}^{n+\kappa} \rightarrow \{0,1\}^n$.

We have already mentioned in introduction a counter-example based on MAC. Namely, we showed that $\mathrm{MAC}(k, m) = H(k\|m)$ provides a secure MAC in the random oracle model for H, but is completely insecure when H is replaced by the previous Merkle-Damgård construction MD^f, because of the message extension attack. In the following, we give a more direct refutation based on the definition of indifferentiability, using again the message extension attack.

We consider only one-block messages or two-block messages. For such messages, we have that $\mathrm{MD}^f(m_1) = f(0, m_1)$ and $\mathrm{MD}^f(m_1, m_2) = f(f(0, m_1), m_2)$. We build a distinguisher that can fool any simulator as follows. The distinguisher first makes a MD^f-query for m_1 and receives $u = \mathrm{MD}^f(m_1)$. Then it makes a query for $v = f(u, m_2)$ to random oracle f. The distinguisher then makes a MD^f-query for (m_1, m_2) and eventually checks that $v = \mathrm{MD}^f(m_1, m_2)$; in this case it outputs 1, and 0 otherwise. It is easy to see that the distinguisher always outputs 1 when interacting with MD^f and f. However, when the distinguisher interacts with H and S (who must simulate f), we observe that S has no information about m_1 (because S does not see the distinguisher's H-queries). Therefore, the simulator cannot answer v such that $v = H(m_1, m_2)$, except with negligible probability.

3.3 Prefix-free Merkle-Damgård

In this section, we show that if the inputs to the plain MD construction are guaranteed to be prefix-free, then the plain MD construction is secure. Namely, prefix-free encoding enables to eliminate the message expansion attack described previously. This "fix" is similar to the fix for the CBC-MAC [3], which is also insecure in its plain form. Thus, the plain MD construction can be safely used for any application of the random oracle H where the length of the inputs is fixed or where one uses domain separation (e.g., prepending $0, 1, \ldots$ to differentiate between inputs from different domains). For other applications, one must specifically ensure that prefix-freeness is satisfied.

A prefix-free code over the alphabet $\{0,1\}^\kappa$ is an efficiently computable injective function $g : \{0,1\}^* \rightarrow (\{0,1\}^\kappa)^*$ such that for all $x \neq y$, $g(x)$ is not a prefix of $g(y)$. Moreover, it must be easy to recover x given only $g(x)$. We provide two examples of prefix-free encodings. The first one consists in prepending the message size in bits as the first block. The last block is then padded with the bit one followed by zeroes.

Function $g_1(m)$:
let N be the message length of m in bits.
write m as $(m_1, \ldots, m_\ell)$ where for all i, $|m_i| = \kappa$
and with the last block m_ℓ padded with 10^r.
let $g_1(m) = (\langle N \rangle, m_1, \ldots, m_\ell)$ where $\langle N \rangle$ is a κ-bit binary encoding of N.

An important drawback of this encoding is that the message length must be known in advance; this can be a problem for streaming applications in which a large message must be processed on the fly. Our second encoding g_2 does not suffer from this drawback, but requires to waste one bit per block of the message :

Function $g_2(m)$:
write m as $(m_1, \ldots, m_\ell)$ where for all i, $|m_i| = \kappa - 1$
and with the last block m_ℓ padded with 10^r.
let $g_2(m) = (0|m_1, \ldots, 0|m_{\ell-1}, 1|m_\ell)$.

Given any prefix-free encoding g, we consider the following construction of the iterative hash-function pf-$\mathrm{MD}_g^f : \{0,1\}^* \to \{0,1\}^n$, using the Merkle-Damgård hash-function $\mathrm{MD}^f : (\{0,1\}^\kappa)^* \to \{0,1\}^n$ defined previously.

Function pf-$\mathrm{MD}_g^f(m)$:
let $g(m) = (m_1, \ldots, m_\ell)$
$y \leftarrow \mathrm{MD}^f(m_1, \ldots, m_\ell)$
return y

Theorem 2. *The previous construction is (t_D, t_S, q, ϵ)-indifferentiable from a random oracle, in the random oracle model for the compression function, for any t_D, with $t_S = \ell \cdot \mathcal{O}(q^2)$ and $\epsilon = 2^{-n} \cdot \ell^2 \cdot \mathcal{O}(q^2)$, where ℓ is the maximum length of a query made by the distinguisher D.*

Proof. Due to lack of space, we only provide a proof sketch for a particular prefix-free encoding which has a simpler proof; the proof for any prefix-free encoding will be provided in the full version of this paper.

The particular prefix-free encoding that we consider consists in adding the message-length as part of the input of f; moreover, the index of the current block is also included as part of the input of f, so that f can be viewed as an independent random oracle for each block m_i. Specifically, we construct an iterative hash-function $C^f : (\{0,1\}^\kappa)^* \to \{0,1\}^n$ from a compression function $f : \{0,1\}^{n+\kappa+2 \cdot t} \to \{0,1\}^n$ as follows :

Function $C^f(m_1, \ldots, m_\ell)$:
let $y_0 = 0^n$
for $i = 1$ to ℓ do $y_i \leftarrow f(y_{i-1}, m_i, \langle \ell \rangle, \langle i \rangle)$
return y_ℓ

where for all i, $|m_i| = \kappa$. The string $\langle \ell \rangle$ is a t-bit binary encoding of the message length ℓ, and $\langle i \rangle$ is a t-bit encoding of the block index. The construction is shown in Figure 6.

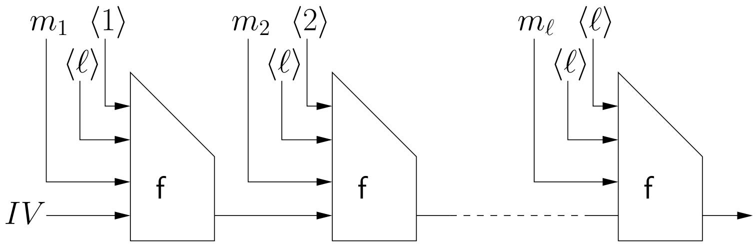

Fig. 6. Merkle-Damgård with a particular prefix-free encoding.

In the following, we show that C^f is indifferentiable from a random oracle, in the random oracle model for f. Since the block-length ℓ is part of the input of the compression function f, we have that C^f behaves independently for messages of different length. Therefore, we can restrict ourselves to messages of fixed length ℓ, *i.e.* it suffices to show that for all ℓ, the construction C^f with message length ℓ is indifferentiable from random oracle $H_\ell : (\{0,1\}^\kappa)^\ell \to \{0,1\}^n$.

We consider for all $1 \le j \le \ell$ the function $C_j^f : (\{0,1\}^\kappa)^j \to \{0,1\}^n$ outputting the intermediate value y_j in C^f. From the definition of C^f, we have for all $2 \le j \le \ell$:

$$C_j^f(m_1, \ldots, m_j) = f(C_{j-1}^f(m_1, \ldots, m_{j-1}), m_j, \langle \ell \rangle, \langle j \rangle) \tag{1}$$

We provide a recursive proof that for all j, the construction C_j^f is indifferentiable from a random oracle. The result for C^f will follow for $j = \ell$. The property clearly holds for $j = 1$. Assuming now that it holds for $j - 1$, we show that it holds for j. We use the following lemma :

Lemma 1. *Let $h_1 : \{0,1\}^a \to \{0,1\}^n$ and $h_2 : \{0,1\}^{n+\kappa} \to \{0,1\}^n$. The construction $R^{h_1,h_2} = h_2(h_1(x), y)$ is indifferentiable from a random oracle, in the random oracle model for h_1 and h_2.*

Replacing C_{j-1}^f by h_1 and $f(\cdot, \langle \ell \rangle, \langle j \rangle)$ by h_2 in equation (1), one then obtains that C_j^f is indifferentiable from a random oracle (see Figure 7 for an illustration).

We now proceed to the proof of lemma 1; due to lack of space, we only provide a proof sketch. One must construct a simulator S such that interacting with $(R, (h_1, h_2))$ is indistinguishable from interacting with (H, S), where H is a random oracle. Our simulator is defined as follows :

Simulator S :
On h_1-query x, return a random $v \in \{0,1\}^n$.
On h_2-query (v', y), check if $v' = h_2(x')$ for some previously queried x'.
In this case, query (x', y) to H and output $H(x', y)$.
Otherwise return a random output.

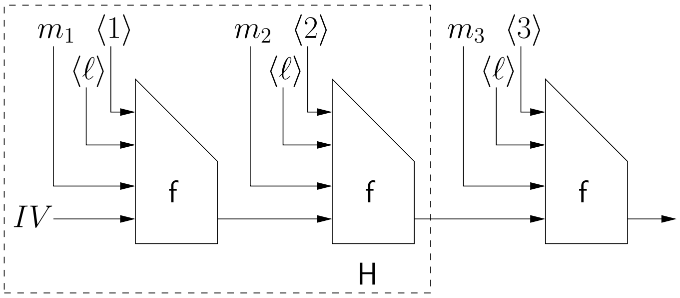

Fig. 7. The output of first two blocks is replaced by a random oracle using Lemma 1.

The distinguisher either interacts with $(R, (h_1, h_2))$ or with (H, S). We denote by F the event that a collision occurs for h_1, that is $h_1(x) = h_1(x')$ for some distinct queries x, x'. We denote by F' the event that the distinguisher makes a h_2-query (v', y) such that $v' = h_1(x)$ and (x, y) was previously queried to R, but x was never queried directly to h_1 by the distinguisher. We claim that conditioned on the complement of $F \vee F'$, the simulation of S is perfect (see the full paper for a complete justification). The distinguishing probability is then at most $\Pr[F \vee F']$; for a distinguisher making at most q queries, this gives:

$$\Pr[F \vee F'] \leq \frac{2q^2}{2^n}$$

which shows a negligible distinguishing probability. □

3.4 The Chop Solution

In this section, we show that by removing a fraction of the output of the plain Merkle-Damgård construction MD^f, one obtains a construction indifferentiable from a random oracle. This "fix" is similar to the method used by Dodis et al. [14] to overcome the problem of using plain MD chaining for randomness extraction from high-entropy distributions, and to the suggestion of Lucks [22] to increase the resilience of plain MD chaining to multi-collision attacks. It is also already used in practice in the design of hash functions SHA-348 and SHA-224 [17] (both obtained by dropping some output bits from SHA-512 and SHA-256). Here we show that by dropping a non-trivial number of output bits from the plain MD chaining, one gets a secure random oracle H even if the input is not encoded in the prefix-free manner. For example, such dropping prevents the "extension" attacks we saw in the MAC application, since the attacker cannot guess the value of the dropped bits, and cannot extend the output of the MAC to a valid MAC of a longer message.

Formally, given a compression function $f : \{0,1\}^{n+\kappa} \rightarrow \{0,1\}^n$, the new construction chop-MD^f_s is defined as follows :

Function chop-$\text{MD}_s^f(m)$:
  let $m = (m_1, \ldots, m_\ell)$
  $y \leftarrow \text{MD}^f(m_1, \ldots, m_\ell)$
  return the first $n - s$ bits of y.

Theorem 3. *The chop-MD_s^f construction is (t_D, t_S, q, ϵ) indifferentiable from a random oracle, for any t_D, with $t_S = \ell \cdot \mathcal{O}(q^2)$ and $\epsilon = 2^{-s} \cdot \ell^2 \cdot \mathcal{O}(q^2)$. Here ℓ is the maximum length of a query made by the distinguisher D.*

While really simple, the drawback of this method is that its exact security is proportional to $q^2 2^{-s}$, where s is the number of chopped bits and q is the number of oracle queries. Thus, to achieve adequate security level the value of s has to be relatively high, which means that short-output hash functions such as SHA-1 and MD5 cannot be fixed using this method. However, functions such as SHA-512 can naturally be fixed (say, by setting $s = 256$).

3.5 The NMAC and HMAC constructions

The NMAC construction [8], which is the basis of the popular HMAC construction, applies an *independent* hash function g to the output of the plain MD chaining. It has been shown very valuable in the design of MACs [8], and recently also randomness extractors [14]. Here we show that if g is modelled as another fixed-length random oracle *independent* from the random oracle f (used for the compression function), then once again one gets a secure construction of an arbitrary-length random oracle H, even if plain MD chaining is applied without prefix-free encoding. Intuitively, applying g gives another way to hide the output of the plain MD chaining, and thus prevent the "extension" attack described earlier.

Formally, given $f : \{0,1\}^{n+\kappa} \to \{0,1\}^n$ and $g : \{0,1\}^n \to \{0,1\}^{n'}$, the function $\text{NMAC}^{f,g}$ is defined as (see Figure 8a):

Function $\text{NMAC}^{f,g}(m)$:
  let $m = (m_1, \ldots, m_\ell)$
  $y \leftarrow MD^f(m_1, \ldots, m_\ell)$
  $Y \leftarrow g(y)$
  return Y

Theorem 4. *The construction $NMAC^{f,g}$ is (t_D, t_S, q, ϵ) indifferentiable from a random oracle for any t_D, $t_S = \ell \cdot \mathcal{O}(q^2)$ and $\epsilon = 2^{-\min(n,n')}\ell^2\mathcal{O}(q^2)$, in the random oracle model for f and g, where ℓ is the maximum message length queried by the distinguisher.*

To practically instantiate this suggestion, we would like to implement f and g from a single compression function. This problem is analogous to the problem in going from NMAC to HMAC in [8], although our solution is slightly different. One simple way for achieving this is to use domain separation: e.g., by prepending 0 for calls to f and 1 — for calls to g. However, with this modeling we are effectively using the prefix-free encoding mapping $m_1 m_2 \ldots m_\ell$ to $0m_1 0m_2 \ldots 0m_\ell 10^\kappa$, which appears slightly wasteful. Additionally, this also forces us to go into the lower-level implementation details for the compression function, which we would like to avoid. Instead, our solution consists in applying two *black-box calls to the plain Merkle-Damgård construction* MD^f (with the same f and IV) : first to the input $0^\kappa m_1 \ldots m_\ell$, getting an n-bit output y, and again to κ-bit y', where y' is defined from y as follows (see Figure 8b):

Function $\mathrm{HMAC}^f(m)$:
  let $m = (m_1, \ldots, m_\ell)$
  let $m_0 = 0^\kappa$
  $y \leftarrow \mathrm{MD}^f(m_0, m_1, \ldots, m_\ell)$
  if $n < \kappa$ then $y' \leftarrow y \,\|\, 0^{\kappa-n}$
  else $y' \leftarrow y|_\kappa$
  $Y \leftarrow \mathrm{MD}^f(y')$
  return Y

Intuitively, we are almost using the NMAC construction with $g(y) = f(IV, y')$ (where y' is obtained from y as above), except we prepend a fixed block $m_0 = 0^\kappa$ to our message. This latter tweak is done to ensure that there are no inter-dependencies between using the same IV on y' and the first message block (which would have been under adversarial control had we not prepended m_0). Indeed, it is very unlikely that "high-entropy" y' will ever be equal to $m_0 = 0^\kappa$, so the analysis for NMAC can be easily extended for this optimization.

Theorem 5. *The construction* $HMAC^f$ *is* (t_D, t_S, q, ϵ) *indifferentiable from a random oracle for any* t_D, $t_S = \ell \cdot \mathcal{O}(q^2)$ *and* $\epsilon = 2^{-\min(n,\kappa)} \cdot \ell^2 \cdot \mathcal{O}(q^2)$, *in the random oracle model for* f, *where* ℓ *is the maximum message length queried by the distinguisher.*

4 Constructions using Ideal Cipher

In practice, most hash-function constructions are block-cipher based, either explicitly as in [29] or implicitly as for SHA-1. Therefore, we consider the question of designing an arbitrary-length random oracle H from an ideal block cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \rightarrow \{0,1\}^n$, specifically concentrating on using the Merkle-Damgård construction with the Davies-Meyer compression function $f(x, y) = E_y(x) \oplus x$ (see Figure 9), since this is the most practically relevant construction. We notice that the question of designing a *collision-resistant* hash function H from an ideal block cipher was explicitly considered by Preneel, Govaerts and Vandewalle in [29], and latter formalized and extended by Black, Rogaway and Shrimpton [9]. Specifically, the authors of [9] actually considered 64 block-cipher variants of the Merkle-Damgård transform (which included the Davies-Meyer variant among them), and formally showed that exactly 20 of these variations (including the Davies-Meyer variant) are collision-resistant when the block cipher E is modeled as an ideal cipher. However, while our work will also model E

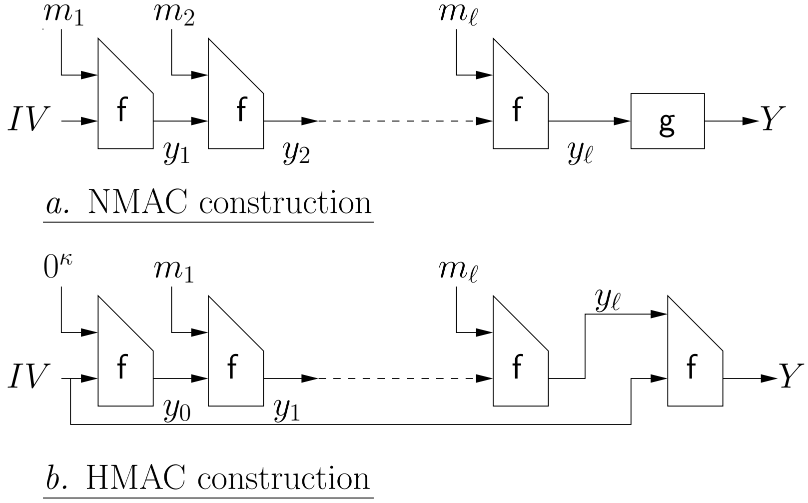

Fig. 8. The NMAC and HMAC constructions

as an ideal cipher, our security goal is considerably stronger than mere collision-resistance. Indeed, we already pointed out that none of the 64 variants above can withstand the "extension" attack on the MAC application, even with the Merkle-Damgård strengthening. And even when restricting to a fixed number of blocks ℓ (which invalidates the "extension" attack), collision-resistance is completely insufficient for our purposes. For example, the authors of [9] show the collision-resistance when using the plain MD chaining with fixed IV and compression function $f(x, y) = E_y(x)$. On the other hand, it is easy to see that this method does not provide a secure random oracle H according to our definition.

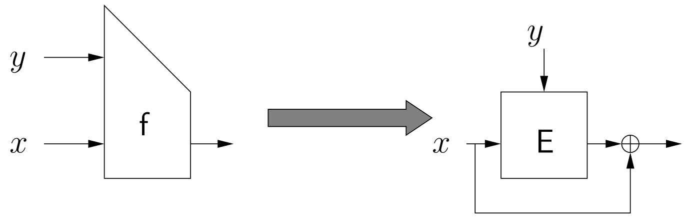

Fig. 9. The Davies-Meyer Compression function

From a different direction, if we could show that the Davies-Meyer compression function $f(x, y) = E_y(x) \oplus x$ is a secure random oracle when E is an ideal block-cipher, then we could directly apply any of the three fixes discussed above.

Unfortunately, this is again not the case: intuitively, the above construction allows anybody to compute x from $f(x, y) \oplus x$ and y (since $x = E_y^{-1}(f(x, y) \oplus x)$), which should not be the case if f was a true random oracle. Thus, we need a direct proof to argue the security of the Davies-Meyer construction. Luckily, using such direct proofs we indeed argue that all of the fixes to the plain MD chaining which worked when f was a fixed-length random oracle, are still secure when $f(x, y) = E_y(x) \oplus x$ is used instead. Namely, we can either use a prefix-free encoding, or drop a non-trivial number of output bits, or apply an independent random oracle g to the output of plain MD chaining. With respect to this latter fix, we also show that we can implement this independent g using the ideal cipher itself, similarly to the case with an ideal compression function f.

Formally, given a block-cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$, the plain Merkle-Damgård hash-function with Davies-Meyer's compression function is defined as :

Function $\mathrm{MD}^E(m_1, \ldots, m_\ell)$:
let $y_0 = 0^n$ (more generally, some fixed IV value can be used)
for $i = 1$ to ℓ do $y_i \leftarrow E_{m_i}(y_{i-1}) \oplus y_{i-1}$
return $y_\ell \in \{0,1\}^n$.

where for all i, $|m_i| = \kappa$. The block-cipher based iterative hash-functions pf-MD_g^E, chop-MD_s^E, NMAC_g^E and HMAC^E are then defined as in section 3, using MD^E instead of MD^f. The proof of the following theorem is given in the full version of this paper.

Theorem 6. *The block-cipher based constructions pf-MD_g^E, chop-MD_s^E, $NMAC_g^E$ and $HMAC^E$ are (t_D, t_S, q, ϵ)-indifferentiable from a random oracle, in the ideal cipher model for E, for any t_D and $t_S = \ell \cdot \mathcal{O}(q^2)$, with $\epsilon = 2^{-n} \cdot \ell^2 \cdot \mathcal{O}(q^2)$ for pf-MD_g^E, $\epsilon = 2^{-s} \cdot \ell^2 \cdot \mathcal{O}(q^2)$ for chop-MD_s^E, $\epsilon = 2^{-\min(n,n')} \cdot \ell^2 \cdot \mathcal{O}(q^2)$ for $NMAC_g^E$ and $\epsilon = 2^{-\min(\kappa,n)} \cdot \ell^2 \cdot \mathcal{O}(q^2)$ for $HMAC^E$. Here ℓ is the maximum message length queried by the distinguisher.*

5 Conclusion

In this paper, we pointed the attention of the cryptographic community to the gap between assuming an arbitrary-length random oracle H and assuming a fixed-length ideal building block for H such as a fixed-length compression function or a block cipher. We then provided a formal definition which suffices to eliminate this gap, noticed that the current iterative hash functions like SHA-1 and MD5 do not satisfy our security notion, and showed several practically motivated, easily implementable and provably secure fixes to the plain Merkle-Damgård transformation. Specifically, one can either ensure that all the inputs appear in the prefix-free form, or drop a nontrivial number of the output bits (if the output of the hash function is long enough to allow it), or, — when the above methods are not applicable — apply an independent fixed-length hash function to the output, which, as we illustrated, can be conveniently implemented using the corresponding building block itself.

An interesting open problem is to provide a construction in the opposite direction, that is, a construction that securely realizes an ideal block-cipher (or a random permutation) from a random oracle. One could use the Luby-Rackoff construction of a pseudo-random permutation from a pseudo-random function [21], but the major difference is that here the adversary has oracle access to the inner functions. One can show that at least six rounds are required to securely realize a random permutation from a random oracle (which should be contrasted with the secret-key case where four rounds are necessary and sufficient [21]), but we were not able to find a proof that six or more rounds would be sufficient.

Acknowledgments: We would like to deeply thank Victor Shoup for his invaluable contribution to all aspects of this work. We also thank the anonymous referees for many useful comments.

References

1. J. H. An, M. Bellare, *Constructing VIL-MACs from FIL-MACs: Message Authentication under Weakened Assumptions*, CRYPTO 1999, pages 252-269.
2. Mihir Bellare, Alexandra Boldyreva and Adriana Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. Proccedings of Eurocrypt 2004.
3. M. Bellare, J. Kilian, and P. Rogaway. The Security of Cipher Block Chaining. In *Crypto '94*, pages 341–358, 1994. LNCS No. 839.
4. M. Bellare and P. Rogaway, *Random oracles are practical : a paradigm for designing efficient protocols.* Proceedings of the First Annual Conference on Computer and Commmunications Security, ACM, 1993.
5. M. Bellare and P. Rogaway, *The exact security of digital signatures - How to sign with RSA and Rabin.* Proceedings of Eurocrypt'96, LNCS vol. 1070, Springer-Verlag, 1996, pp. 399-416.
6. M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption*, Proceedings of Eurocrypt'94, LNCS vol. 950, Springer-Verlag, 1994, pp. 92–111.
7. M. Bellare and P. Rogaway, *Collision-Resistant Hashing: Towards Making UOWHFs Practical*, In *Crypto '97*, LNCS Vol. 1294.
8. M. Bellare, R. Canetti, and H. Krawczyk, *Pseudorandom Functions Re-visited: The Cascade Construction and Its Concrete Security*, In Proc. 37th FOCS, pages 514-523. IEEE, 1996.
9. J. Black, P. Rogaway, T. Shrimpton, *Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV*, in Advances in Cryptology - CRYPTO 2002, California, USA.
10. R. Canetti, *Universally Composable Security: A New Paradigm for Cryptographic Protocols*, proceedings of the 42nd Symposium on Foundations of Computer Science (FOCS), 2001. Cryptology ePrint Archive, Report 2000/067, `http://eprint.iacr.org/`.
11. R. Canetti, O. Goldreich and S. Halevi, *The random oracle methodology, revisited*, STOC' 98, ACM, 1998.

12. Ran Canetti, Oded Goldreich and Shai Halevi. On the random oracle methodology as applied to Length-Restricted Signature Schemes. In *Proceedings of Theory of Cryptology Conference*, pp. 40–57, 2004.
13. I. Damgård, *A Design Principle for Hash Functions*, In Crypto '89, pages 416-427, 1989. LNCS No. 435.
14. Y. Dodis, R. Gennaro, J. Håstad, H. Krawczyk, and T. Rabin, *Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes*, Advances in Cryptology - CRYPTO, August 2004.
15. Y. Dodis, R. Oliveira, K. Pietrzak, *On the Generic Insecurity of the Full Domain Hash*, Advances in Cryptology - CRYPTO, August 2005.
16. FIPS 180-1, *Secure hash standard*, Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/N.I.S.T., National Technical Information Service, Springfield, Virginia, April 17 1995 (supersedes FIPS PUB 180).
17. National Institute of Standards and Technology (NIST). Secure hash standard. FIPS 180-2. August 2002.
18. RFC 1321, *The MD5 message-digest algorithm*, Internet Request for Comments 1321, R.L. Rivest, April 1992.
19. Shafi Goldwasser and Yael Tauman. On the (In)security of the Fiat-Shamir Paradigm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (2003), 102-114.
20. H. Handschuh and D. Naccache, *SHACAL*, In B. Preneel, Ed., First Open NESSIE Workshop, Leuven, Belgium, November 13-14, 2000
21. M. Luby and C. Rackoff, *How to construct pseudo-random permutations from pseudo-random functions*, SIAM J. Comput., Vol. 17, No. 2, April 1988.
22. Stefan Lucks. *Design Principles for Iterated Hash Functions*, available at E-Print Archive, `http://eprint.iacr.org/2004/253`.
23. U. Maurer, R. Renner, and C. Holenstein, *Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology*, Theory of Cryptography - TCC 2004, Lecture Notes in Computer Science, Springer-Verlag, vol. 2951, pp. 21-39, Feb 2004.
24. Ueli Maurer and Johan Sjodin. *Single-key AIL-MACs from any FIL-MAC*, In *ICALP 2005*, July 2005.
25. R. Merkle, *One way hash functions and DES*, Advances in Cryptology, Proc. Crypto'89, LNCS 435, G. Brassard, Ed., Springer-Verlag, 1990, pp. 428-446.
26. Jesper Buus Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-Committing Encryption Case. In *Advances in Cryptology - Crypto 2002 Proceedings* (2002), 111 -126
27. PKCS #1 v2.1, *RSA Cryptography Standard (draft)*, document available at `www.rsa security.com/rsalabs/pkcs`.
28. B. Pfitzmann and M. Waidner, *A model for asynchronous reactive systems and its application to secure message transmission*. In IEEE Symposium on Security and Privacy, pages 184-200. IEEE Computer Society Press, 2001.
29. B. Preneel, R. Govaerts and J. Vandewalle, *Hash Functions Based on Block Ciphers: A Synthetic Approach*, in Advances in Cryptology - CRYPTO '93,, Santa Barbara, California, USA.
30. V. Shoup, *A composition theorem for universal one-way hash functions*, In *Eurocrypt '00*, pp. 445–452, LNCS Vol. 1807.
31. R. Winternitz, *A secure one-way hash function built from DES*, in Proceedings of the IEEE Symposium on Information Security and Privacy, pages 88-90. IEEE Press, 1984.