Merkle-Damgård Revisited : how to Construct a Hash Function

Jean-Sébastien Coron * Yevgeniy Dodis † Cécile Malinaud ‡ Prashant Puniya §

September 4, 2007

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.

*University of Luxembourg, Email: `coron@clipper.ens.fr`
†New York University, Email: `dodis@cs.nyu.edu`
‡Gemplus Card International, Email: `cecile.malinaud@normalesup.org`
§New York University, Email: `puniya@cs.nyu.edu`

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 [17] or MD5 [19]. In fact, many recent "separation" results [12, 27, 20, 2, 13, 16] 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 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 [28]) 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). This procedure is known as *Merkle-Damgård strengthening.* 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 [32] 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 [21].

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 [30, 10]), 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 [14, 26, 30, 10], pseudorandomness [8], unforgeability [1, 25] and randomness extraction [15], 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. [24]. This framework enjoys the desired closure property we seek, and is very intuitive and easy to state. However, we view adapting the indifferentiability framework of Maurer to our problem as one of the main contributions of our work.

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 [14, 26], pseudorandom functions [8], MACs [1, 25] and universal one-way hash functions [7, 31]. 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 $0^\kappa m_1 \dots m_\ell$, 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.

Note that we could also define the HMAC construction by using a different initialization vector in each part of the construction, instead of using the same IV but prepending 0^κ to the input. However, our purpose here is to present these constructions as black-box extensions of existing hash functions such as SHA-1 which have only one fixed IV, in which case our proposed construction can be viewed as making two black-box calls to SHA-1 to get $SHA-1(SHA-1(0^\kappa \parallel m_1 \dots m_\ell)$.

However, in practice most hash-function constructions are block-cipher based, either explicitly as in [30] 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.

Other Extensions. We also discuss other practical extensions of our results. We note that each of our proposed constructions allow really efficient *output expansion*. In particular, one can extend the output length of any of our constructions by a factor of L by using only one extra evaluation of the underlying fixed-length input primitive for each extra block. We also show that the same procedure can also be used for *domain separation of RO* to get multiple independent random oracles using a single one.

Bellare and Ristenpart [9] discuss the notion of a multi-property preserving construction. In particular, such a construction is an indifferentiable random oracle construction as well as a domain extender for pseudorandom functions and collision-resistant hash functions.

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 [24]. 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 [11] and on the model of Pfitzmann and Waidner [29]. The indifferentiability notion in [24] 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 [11]).

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 [24] :

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.

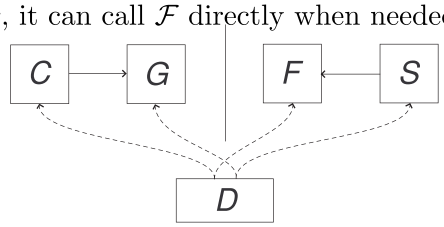

Figure 1: The indifferentiability notion: the distinguisher D either interacts with algorithm C and ideal primitive $\mathcal{G}$, or with ideal primitive $\mathcal{F}$ and simulator S. Algorithm C has oracle access to $\mathcal{G}$, while simulator S has oracle access to $\mathcal{F}$.

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 [24] 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.

We use the definition of [24] 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

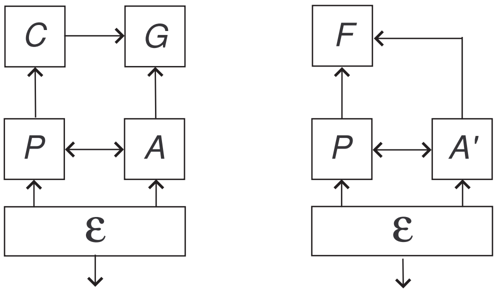

Figure 2: The environment $\mathcal{E}$ interacts with cryptosystem $\mathcal{P}$ and attacker $\mathcal{A}$. In the $\mathcal{G}$ model (left), $\mathcal{P}$ has oracle access to C whereas $\mathcal{A}$ has oracle access to $\mathcal{G}$. In the $\mathcal{F}$ model, both $\mathcal{P}$ and $\mathcal{A}'$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

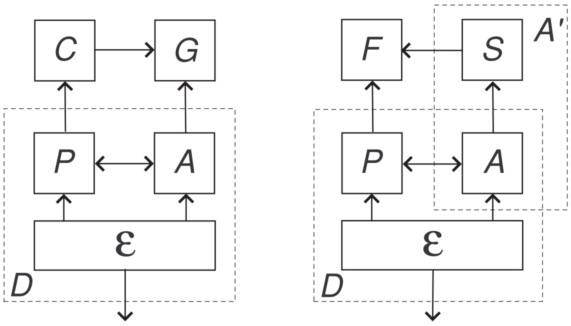

Figure 3: Construction of attacker $\mathcal{A}'$ from attacker $\mathcal{A}$ and simulator $\mathcal{S}$.

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 [24] shows that security is preserved when replacing an ideal primitive by an indifferentiable one :

Theorem 2.1 *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 [24] 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).

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. $\square$

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.

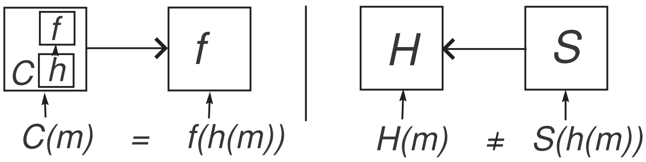

Figure 4: The simulator cannot output $H(m)$ since it only receives $h(m)$ and cannot recover m from $h(m)$.

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 $y_\ell \in \{0,1\}^n$.

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

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

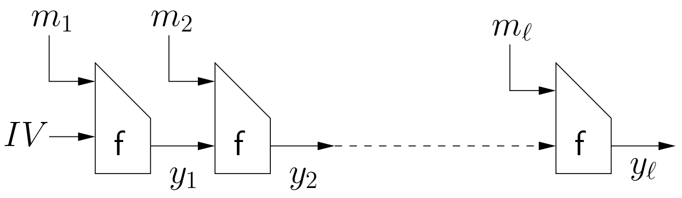

Figure 5: The plain Merkle-Damgård Construction

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-$\mathrm{MD}^f_g : \{0,1\}^* \to \{0,1\}^n$, using the Merkle-Damgård hash-function $\mathrm{MD}^f : (\{0,1\}^\kappa)^* \to \{0,1\}^n$ defined previously.

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

Theorem 3.1 *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 is given at the end of section 4 because we derive it as an implication of theorem 4.1. The simpler proof given here illustrates the overall structure of proofs of indifferentiability.

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.

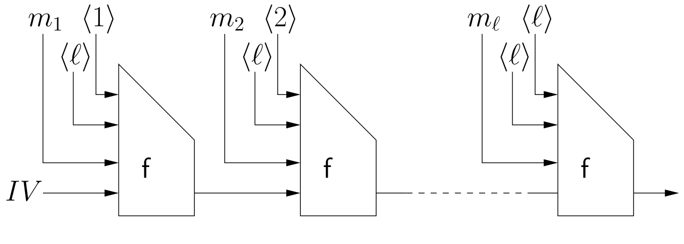

Figure 6: Merkle-Damgård with a particular prefix-free encoding.

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 3.2 *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).

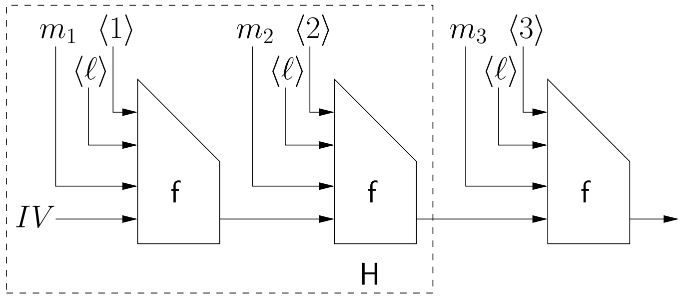

Figure 7: The output of the first two blocks is replaced by a random oracle using Lemma 3.2.

We now proceed to the proof of lemma 3.2; 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.

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:

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

which shows a negligible distinguishing probability. $\square$

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. [15] to overcome the problem of using plain MD chaining for randomness extraction from high-entropy distributions, and to the suggestion of Lucks [23] 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 [18] (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 $f : \{0,1\}^{n+\kappa} \to \{0,1\}^n$, the new construction chop-MD_s^f is defined as follows :

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

Theorem 3.3 *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 $s = 256$). A formal proof of theorem 3.3 is given in the next section.

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 [15]. 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 $\mathrm{NMAC}^{f,g}$ is defined as (see Figure 8a):

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

Theorem 3.4 *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 $m_1 m_2 \dots m_\ell$ to $0m_1 0m_2 \dots 0m_\ell 10^\kappa$, 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 $0^\kappa m_1 \dots m_\ell$, getting an n-bit output y, and again to κ-bit y', where y' is defined from y as follows (see Figure 8b):

Function HMAC$^f(m)$:
  let $m = (m_1, \dots, m_\ell)$
  let $m_0 = 0^\kappa$
  $y \leftarrow \mathrm{MD}^f(m_0, m_1, \dots, m_\ell)$
  if $n < \kappa$ then $y' \leftarrow y \parallel 0^{\kappa-n}$
  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 3.5 *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.*

The formal proofs for both theorems 3.4 and 3.5 are given in the next section.

4 Constructions using Ideal Cipher

In practice, most hash-function constructions are block-cipher based, either explicitly as in [30] 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 [30], and latter formalized and extended by Black, Rogaway and Shrimpton [10]. Specifically, the authors of [10] 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 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

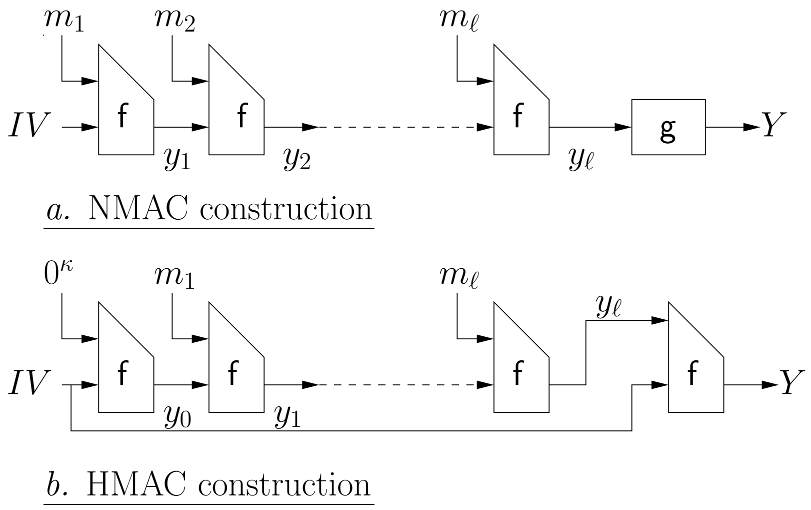

Figure 8: The NMAC and HMAC constructions

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 [10] 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.

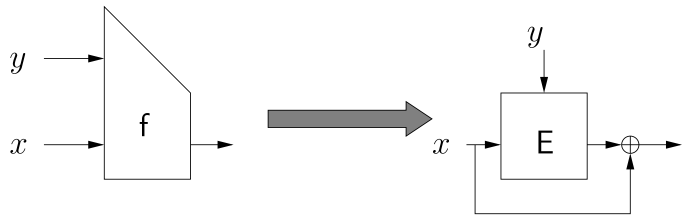

Figure 9: The Davies-Meyer Compression function

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 4.1 *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.*

Proof: We will prove that the Merkle-Damgård (MD) based constructions are indifferentiable constructions of a random oracle (RO), when applied to the Davies-Meyer (DM) compression function using an ideal block cipher (IC). The four constructions that we prove to be secure are:

1. **Prefix-free Merkle-Damgård construction pf-$\mathbf{MD}_g^E$:** In this construction, we apply the Davies-Meyer Merkle-Damgård (DMMD) construction to a prefix-free encoding of the input (using the prefix-free encoding scheme g).

2. **Merkle-Damgård construction with truncated output chop-$\mathbf{MD}_s^E$:** This is the plain DMMD construction applied directly to the input, with a non-trivial number, s, of the output bits chopped.

3. **NMAC construction $\mathbf{NMAC}^{E1,E2}$:** This construction uses two independent ideal block ciphers $E1 : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ and $E2 : \{0,1\}^{\kappa'} \times \{0,1\}^{n'} \to \{0,1\}^{n'}$. It first applies the DMMD construction using $E1$ to the input, getting a n bit output Y. Then it applies the Davies-Meyer compression function using $E2$ to Y to get the final output.

4. **HMAC construction $\mathbf{HMAC}^E$:** This is an instantiation of the NMAC construction using the same ideal cipher for both parts, but using different initialization vectors in each part (implemented by prepending 0^κ to the input).

The proof of indifferentiability in each of these cases essentially involves two steps. First, we propose a simulator that simulates the task of the ideal cipher in the random oracle model (ROM). Secondly, we show that the view of any distinguisher in the ROM, with oracle access to the actual random oracle and the ideal cipher simulator, does not differ from its view in the ideal cipher model (ICM), with oracle access to the RO construction and the ideal cipher, by more than a negligible amount. We have provided detailed proofs for each of the above constructions in the appendix 6. Here we will concentrate on providing an intuitive idea of the basic paradigm used in each of the proofs.

The Simulator. The task of the simulator in each of the cases is to simulate the ideal cipher in the ROM, in such a way that its relation with the random oracle is consistent with the relation between that actual ideal cipher and the RO construction in the ICM. Thus, in each case, the simulator essentially gives random responses to all forward block cipher queries except those that form the last application of the ideal cipher for some random oracle input (when processed using the RO construction). For example, in the Chop construction this will be the last block cipher call in the Davies-Meyer Merkle-Damgård computation.

If the query corresponds to a last block cipher call, then the simulator consults the random oracle and adjusts its response so as to remain consistent with the ICM scenario.

In the case of an inverse block cipher query, the simulator always gives random responses. In addition, the simulator also maintains a table $\mathcal{T}$ in which it records all previous query-response pairs (so as to maintain consistency among its responses).

Proof of Indifferentiability. Each of the proofs of indifferentiability consist of a hybrid argument that presents a sequence of mutually indistinguishable games starting in the random oracle model, with the RO F and the ideal cipher simulator S, leading up to the ideal cipher model, with the RO construction (which we call C^E) and the ideal cipher E. The overall structure of the hybrid argument is similar for each of the constructions, though the formal proof differs. We will describe the overall structure of the proof here.

Game 1. This is the random oracle model, where the distinguisher is given oracle access to the random oracle F and the ideal cipher simulator S.

Game 2. In this game, we introduce a *relay algorithm* R_0 that is simply a dummy algorithm between the distinguisher and the random oracle F. This relay algorithm simply relays the queries of the distinguisher to the RO and relays back the output of F.

Game 3. In this game, we modify the simulator by defining a few failure conditions for its query-response pairs. If any of these failure conditions is true, then the new simulator S_0 explicitly fails. These failure conditions capture certain collision conditions which, if they happen, could be exploited by the distinguisher to decide the scenario it is in. The failure conditions are different for each constructions and are described in the formal proof. Thus the distinguisher has oracle access to the new simulator S_0^F and the relay algorithm R_0^F in this game.

Game 4. Now we modify the relay algorithm so as to make its responses directly dependent on the simulator, instead of the RO F. The new relay algorithm R_1 essentially evaluates the construction C^E using the simulator S_0 instead of the ideal cipher E. The main idea here is to prove that unless one of the failure conditions described in game 3 is true for the query-response pairs of the simulator S_0 (in which case it would fail), the responses of R_1 are still consistent with the random oracle. Thus, games 3 and 4 form the heart of the proof in each case. In this game, the distinguisher has oracle access to the relay algorithm $R_1^{S_0^F}$ and the simulator S_0^F.

Game 5. In this game, we modify the simulator so that it chooses its responses independent of the random oracle (i.e. uniformly random by itself). In addition, the new simulator S_1 does not check for any of the failure conditions described above. This does not introduce any changes in the view of the distinguisher since the relay algorithm R_1 uses the simulator S_1 to construct its responses (which still look random). Thus, in this game the distinguisher has oracle access to the relay algorithm $R_1^{S_1}$ and the simulator S_1.

Game 6. Finally, we replace the simulator S_1 by the ideal block cipher E. Thus the relay algorithm R_1 now becomes identical to the RO construction C^E. Thus in this game the distinguisher has oracle access to the RO construction C^E and the ideal cipher E.

This completes the proof of indifferentiability in each of the cases. We have skipped most of the details here, which can be found in appendix 6. $\square$

4.1 Implications for the RO Domain Extenders

We saw above that the four modifications of the Merkle-Damgård construction, i.e. the prefix-free, chop, NMAC and HMAC constructions, applied to the Davies-Meyer compression function are indifferentiable from a variable-length input random oracle (VIL-RO) in the ideal cipher model. This fact was formally stated and proved in theorem 4.1. Now we will show that this result is stronger than the indifferentiability of domain extenders for the random oracle described in section 3. In particular, we show that theorems 3.1, 3.3, 3.4 and 3.5 from section 3 can be derived as a direct consequence of theorem 4.1.

To this purpose, say we are given a fixed-length input random function oracle (FIL-RO) $f : \{0,1\}^{\kappa+n} \to \{0,1\}^n$. Consider the following construction based on f:

$$\begin{array}{rcl} T^f : \{0,1\}^\kappa \times \{0,1\}^n & \to & \{0,1\}^n \\ (x, y) & \mapsto & f(x \parallel y) \oplus y \end{array}$$

Note that the construction T^f is essentially the same as the Davies-Meyer construction except that the latter is defined for an ideal block cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$. If we are able to show that T^f is indifferentiable from the ideal block cipher E, then it will complete the proof of all theorems from section 3 as an implication of theorem 4.1 and the composability property of indifferentiable constructions. This is because the Davies-Meyer construction applied to T^f is identical to the FIL-RO f. However, it is easily seen that T^f cannot be proven indifferentiable from the ideal cipher E[1].

To overcome this, we introduce a weaker ideal primitive than the ideal cipher, which we will call the *weak ideal block cipher*. A *weak ideal block cipher* E is essentially the same as an ideal cipher, except that it only responds to forward block cipher queries. In this case, we do not run into the problem of responding to inverse queries made to the construction T^f. Unfortunately, we cannot use theorem 4.1 in a "black-box manner" to get indifferentiable VIL-RO construction using a weak ideal cipher. However, none of the constructions proposed in theorem 4.1 make use of inverse queries to the underlying block cipher.

Corollary 4.2 *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* weak 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.*

In fact, the proof of this theorem is simpler than that for theorem 4.1 since the simulator need not respond to inverse ideal cipher queries. We now show that the construction T^f is an indifferentiable construction of a weak ideal cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ using the FIL-RO f.

Lemma 4.3 *The construction T^f (described above) is (t_D, t_S, q, ϵ)-indifferentiable from a weak ideal cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ for any t_D, $t_S = \mathcal{O}(q^2)$ and $\epsilon = 2^{-n} \cdot q^2$, in the random oracle model for f.*

Proof: In order to prove this theorem, we need to describe a random oracle simulator S such that no distinguisher can tell apart the random oracle model, where it has oracle access to the random function oracle f and the construction T_f, from the weak ideal cipher model, where it has oracle access to the simulator S and the weak ideal block cipher E. We will start by describing the simulator.

The Simulator. The simulator S gets random oracle queries of the form $x \parallel y \in \{0,1\}^{\kappa+n}$. The simulator makes the forward query (x, y) to block cipher E to get $E_x(y)$. Then S responds with $z = E_x(y) \oplus y$. In addition, the simulator S also maintains a table $\mathcal{T}$ of previous query-response pairs $(x \parallel y, z)$ which it

[1] In particular, the construction T^f cannot answer inverse ideal cipher queries.

checks each time to see if the current query matches a previous one.

Proof of Indifferentiability. The proof of indifferentiability involves a hybrid argument that starts in the ideal cipher model, where the distinguisher D has oracle access to E and S, which is game 1.

GAME 1. This is essentially the *weak ideal cipher model*, where the distinguisher D is given oracle access to the random oracle simulator S and the weak ideal cipher E. Let G_1 denote the event that D outputs 1 in this game. Thus, if λ denote the security parameter,

$$\Pr[G_1] = \Pr\left[D^{S^E,E}(1^\lambda) = 1\right]$$

GAME 2. In this game, we give the distinguisher D oracle access to a relay algorithm R_0, instead of the weak ideal cipher E. This relay algorithm R_0 has oracle access to the simulator S^E. On a forward block cipher query $(x, y) \in \{0,1\}^\kappa \times \{0,1\}^n$, the relay algorithm R_0 simply queries the simulator S_E on $x \parallel y$ to get its response z. Then R_0 responds to the block cipher query with $y \oplus z$.

Let G_2 denote the event that D outputs 1 in this game. Since the view of the distinguisher does not change in this game, we can deduce that

$$\Pr[G_1] = \Pr\left[D^{S^E,R^{S^E}}(1^\lambda) = 1\right] = \Pr[G_1]$$

GAME 3. In this game, we modify the simulator so that it does not consult the ideal block cipher for any of the queries made to it. Instead, the new simulator S_0 always chooses a uniformly random n-bit response z to every new query $x \parallel y$, and records it in its table $\mathcal{T}$ before sending over the response.

Let G_3 denote the event that the distinguisher D outputs 1 in this game. Since the relay algorithm only consults S_0 for any query, so that the view of the distinguisher in this game is entirely independent of the weak ideal cipher E. Thus the distinguisher D detects a difference between this game and game 2 only if the relay algorithm R_0 outputs a collision for two block cipher queries with the same key, and the probability of this event can be easily bounded using the birthday paradox. Thus, we can deduce that

$$|\Pr[G_3] - \Pr[G_2]| \leq \frac{q^2}{2^n}$$

Note that the simulator S_0 is essentially the same as the fixed-length input RO f, while the relay algorithm R_0 is defined in the same way as the construction T^f. Hence, we can also deduce that

$$\begin{aligned}\left|\Pr\left[D^{f,T^f}(1^\lambda) = 1\right] \Pr\left[D^{S^E,E}(1^\lambda) = 1\right]\right| &= |\Pr[G_3] - \Pr[G_1]| \\ &\leq \frac{q^2}{2^n}\end{aligned}$$

□

5 Practical Implications and Other Extensions

Increasing Output Length. All the random oracle constructions that we have discussed, permit really efficient output expansion. Given a random oracle $H : \{0,1\}^* \to \{0,1\}^n$, output expansion by a factor L can be achieved by appending an extra $\log(L)$-bit block to the input X and outputting the concatenation of the following blocks:

$$H(X \parallel \langle 1\rangle), H(X \parallel \langle 2\rangle), \ldots, H(X \parallel \langle L\rangle)$$

It can be easily seen that this construction is generically secure, including any of the indifferentiable constructions of VIL-RO that we have proposed. However, one would imagine that evaluating this construction would involve L evaluations of the VIL-RO H.

As it turns out, for the Prefix-free, Chop, NMAC and HMAC constructions of a VIL-RO using a FIL-RO or an ideal cipher, this procedure can be completed extremely efficiently using only one (or two) extra evaluation of the underlying fixed-length input primitive for each extra block of output. [2] This can be done by first computing the Merkle-Damgård construction on the input X, and evaluating only one last part of the construction for each of the output blocks. This reduces the running time for the procedure from $L \cdot (|X|/\kappa)$ to $L + (|X|/k)$ computations.

Domain Separation for Independent ROs. The same technique as above can also be used for domain separation of the random oracle, to get multiple independent random function oracles from a single one. This is useful in cryptographic constructions where one needs to use multiple independent random oracles in order to prove the security of the construction. In particular, if we have a single random oracle $H : \{0,1\}^* \rightarrow \{0,1\}^n$, and we need L independent random oracles in our constructions, then we can achieve this by defining these random oracles as:

$$\begin{aligned} H_1(X) &:= H(X \parallel \langle 1\rangle) \\ &\vdots \\ H_L(X) &:= H(X \parallel \langle L\rangle) \end{aligned}$$

We cannot use the same efficient processing technique that we used for output expansion, since one usually does not need to evaluate the independent random oracles on the same input.

6 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 [22], but the major difference is that here the adversary has oracle access to the inner functions. One can show that at

[2] For a prefix-free encoding g, this can be done by appending $\langle 1\rangle \ldots \langle L\rangle$ to $g(X)$ instead appending to X and then evaluating g.

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 [22]), 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] M. Bellare and T. Ristenpart, *Multi-Property-Preserving Hash Domain Extension and the EMD Transform*, In Advances in Cryptology - Asiacrypt 2006.

[10] 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.

[11] 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/`.

[12] R. Canetti, O. Goldreich and S. Halevi, *The random oracle methodology, revisited*, STOC' 98, ACM, 1998.

[13] 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.

[14] I. Damgård, *A Design Principle for Hash Functions*, In Crypto '89, pages 416-427, 1989. LNCS No. 435.

[15] 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.

[16] Y. Dodis, R. Oliveira, K. Pietrzak, *On the Generic Insecurity of the Full Domain Hash*, Advances in Cryptology - CRYPTO, August 2005.

[17] 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).

[18] National Institute of Standards and Technology (NIST). Secure hash standard. FIPS 180-2. August 2002.

[19] RFC 1321, *The MD5 message-digest algorithm*, Internet Request for Comments 1321, R.L. Rivest, April 1992.

[20] 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.

[21] H. Handschuh and D. Naccache, *SHACAL*, In B. Preneel, Ed., First Open NESSIE Workshop, Leuven, Belgium, November 13-14, 2000

[22] M. Luby and C. Rackoff, *How to construct pseudo-random permutations from pseudo-random functions*, SIAM J. Comput., Vol. 17, No. 2, April 1988.

[23] Stefan Lucks. *Design Principles for Iterated Hash Functions*, available at E-Print Archive, `http://eprint.iacr.org/2004/253`.

[24] 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.

[25] Ueli Maurer and Johan Sjodin. *Single-key AIL-MACs from any FIL-MAC*, In *ICALP 2005*, July 2005.

[26] 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.

[27] 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

[28] PKCS #1 v2.1, *RSA Cryptography Standard (draft)*, document available at `www.rsasecurity.com/rsalabs/pkcs`.

[29] 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.

[30] 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.

[31] V. Shoup, *A composition theorem for universal one-way hash functions*, In *Eurocrypt '00*, pp. 445–452, LNCS Vol. 1807.

[32] 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.

A Proof of Theorem 4.1

Theorem A.1 *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.*

Proof: We will prove that the Merkle-Damgård (MD) based constructions are indifferentiable constructions of a random oracle (RO), when applied to the Davies-Meyer (DM) compression function using an ideal block cipher (IC). The four constructions that we prove to be secure are:

1. **Prefix-free Merkle-Damgård construction pf-MD$_g^E$:** In this construction, we apply the Davies-Meyer Merkle-Damgård (DMMD) construction to a prefix-free encoding of the input (using the prefix-free encoding scheme g).

2. **Merkle-Damgård construction with truncated output chop-MD$_s^E$:** This is the plain DMMD construction applied directly to the input, with a non-trivial number, s, of the output bits chopped.

3. **NMAC construction NMAC^{E1,E2}:** This construction uses two independent ideal block ciphers $E1 : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ and $E2 : \{0,1\}^{\kappa'} \times \{0,1\}^{n'} \to \{0,1\}^{n'}$. It first applies the DMMD construction using $E1$ to the input, getting a n bit output Y. Then it applies the Davies-Meyer compression function using $E2$ to Y to get the final output.

4. **HMAC construction HMAC^E:** This is an instantiation of the NMAC construction using the same ideal cipher for both parts, but using different initialization vectors in each part (implemented by prepending 0^κ to the input).

The proof of indifferentiability in each of these cases essentially involves two steps. First, we propose a simulator that simulates the task of the ideal cipher in the random oracle model (ROM). Secondly, we show that the view of any distinguisher in the ROM, with oracle access to the actual random oracle and the ideal cipher simulator, does not differ from its view in the ideal cipher model (ICM), with oracle access to the RO construction and the ideal cipher, by more than a negligible amount. The proof of indifferentiability in each of the four cases involves a hybrid argument.

We start by proving the indifferentiability of the prefix-free MD construction pf-MD$_g^E$.

Lemma A.2 *The prefix-free Merkle-Damgård construction pf-MD$_g^E$ using an ideal cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ is (t_D, t_S, q, ϵ)-indifferentiable from a random oracle in the ideal cipher model for E, for any t_D and $t_S = \mathcal{O}(q \cdot R_g(q \cdot \kappa))$ (where $R_g(q \cdot \kappa)$ is the running time of the decoding algorithm of g on an input of length $q \cdot \kappa$), with $\epsilon = 2^{-n} \cdot \ell^2 \cdot \mathcal{O}(q^2)$.*

Proof:
The Simulator. The simulator S_E accepts either forward ideal cipher queries, $(+, x, y)$, or inverse ideal cipher queries, $(-, x, z)$, such that $x \in \{0,1\}^\kappa$ and $y, z \in \{0,1\}^n$. In either case, the simulator S responds with a n-bit string that is interpreted as $E_x(y)$ in case of a forward query $(+, x, y)$ and as $E_x^{-1}(z)$ in case of an inverse query. The simulator maintains a table $\mathcal{T}$ of triples $(x, y, z) \in \{0,1\}^\kappa \times \{0,1\}^n \times \{0,1\}^n$, such that it either responded with z to a forward query $(+, x, y)$ or with y to an inverse query $(-, x, z)$.

On getting a forward query $(+, x, y)$, the simulator searches its table $\mathcal{T}$ for a triple (x, y, z) for any z. If there exists such a triple, then it responds with z otherwise it needs to choose a new response to this query. It then searches its table $\mathcal{T}$ for a sequence of triples $(x_1, y_1, z_1) \ldots (x_i, y_i, z_i)$ such that:

- The bit string $x_1 \parallel \ldots \parallel x_i \parallel x$ decodes to a valid RO input under the prefix-free encoding g.
- It is the case that $y_1 = IV$, where IV denotes the initialization vector used in the construction pf-MD$_g^E$.
- For each $j = 2 \ldots i$, it is the case that $y_j = z_{j-1} \oplus y_{j-1}$.
- It is the case that $y = z_i \oplus y_i$, where y is the input message in the current forward query.

Note that for an empty sequence of triples, i.e. when just considering the κ-bit block x from the current query, only the first requirement makes sense. We additionally also require that $y = IV$ in this case.

If the simulator S finds such a sequence of triples, then it needs to give a response that is consistent with the random oracle output on $g^{-1}(x_1 \parallel \ldots \parallel x_i \parallel x)$. Thus, the simulator makes this RO query to get the output $Y = F(g^{-1}(x_1 \parallel \ldots \parallel x_i \parallel x))$, and responds with $z = Y \oplus y$. If the simulator does not find such a sequence of triples, it outputs a random response z. In either case, it stores the triple (x, y, z) in its table $\mathcal{T}$.

On receiving an inverse query $(-, x, z)$, the simulator S searches its table $\mathcal{T}$ for a triple (x, y, z) for any y. If it finds such a triple, then it outputs y as its response. If it does not find such a triple, it chooses a random n-bit string y and responds with y. It then stores the triple (x, y, z) into its table $\mathcal{T}$.

Proof of Indifferentiability. We need to prove that the distinguisher cannot tell apart the two scenarios, one where it has oracle access to the random oracle F and the simulator S and the other where it has access to the RO construction pf-MD$_g^E$ and the ideal block cipher E. As we mentioned above, the proof involves a hybrid argument starting in the random oracle scenario, and ending in the ideal cipher scenario through a sequence of mutually indistinguishable hybrid games.

Game 1. This is the *random oracle model*, where the distinguisher D has oracle access to the random oracle F and the simulator S described above. Let G_1 denote the event that D outputs 1 after interacting with F and S. Thus,

$$\Pr[G_1] = \Pr\left[D^{F,S^F}(1^\lambda) = 1\right]$$

Game 2. In this game, we give the distinguisher oracle access to a dummy *relay algorithm* R_0 instead of direct oracle access to the random oracle F. This relay algorithm R_0 is given oracle access to the random oracle F, and on getting a random oracle query from the distinguisher, it simply makes the same query to the RO F and forwards the RO output to the distinguisher as its response. Let G_2 denote the event that the distinguisher outputs 1 in this game. Since we have left the view of the distinguisher unchanged in this game, the distribution of its outputs also remains the same.

$$\Pr[G_2] = \Pr\left[D^{R_0^F,S^F}(1^\lambda) = 1\right] = \Pr[G_1]$$

Game 3. In this game, we modify the simulator S. In particular, we restrict the responses of the simulator such that they never satisfy certain specific failure conditions. If the simulator comes up with a response that results in its responses satisfying one of these conditions, then it fails explicitly instead of sending this response.

The failure conditions that the new simulator S_0 avoids essentially describe certain dependencies that could arise among its responses that could be exploited by the distinguisher. In response to a forward query $(+, x, y)$, the new simulator chooses a response $z \in \{0,1\}^n$ similar to the original simulator S and it checks for the following conditions:

1. *Condition B_1:* It is the case that $z \oplus y = IV$, where IV is the n-bit initialization vector used in the RO construction pf-MD_g^E.

2. *Condition B_2:* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', y') \neq (x, y)$, such that $y' \oplus z' = y \oplus z$.

3. *Condition B_3:* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', y') \neq (x, y)$, such that $y \oplus z = y'$.

If the response z is chosen by the simulator S_0 at random then the simulator S_0 checks for these conditions and explicitly fails if any of them holds. However, if the simulator is forced to choose a response in order to maintain consistency with the random oracle F, then it only checks for the conditions B_1 and B_2.

Let us briefly describe how the distinguisher can exploit each of these conditions to its advantage. If the condition B_1 holds then the distinguisher could possibly force two different RO query sequences to end in the same block, where one input is the suffix of the other. Hence the simulator can be consistent with at most one of these two RO inputs. If condition B_2 holds, then the distinguisher can again force two query sequences to end in the same block. However, in this case the two RO inputs have a common suffix and the simulator can be consistent with at most one of these inputs. If condition B_3 holds, then distinguisher can make a RO query sequence to the simulator such that the simulator is not consistent with the RO output because the query corresponding to the last block of the (encoding of the) RO input is not the last one that it makes.

Now we will estimate the occurrence probability for each of the above failure conditions. Let the number of random oracle queries made by the distinguisher be q_F, and let the number of ideal cipher queries be q_E. To start with, it is easy to see that the occurrence probability of condition B_1 is at most the probability that one of $q(= q_E + q_F)$ random n-bit strings are equal to IV.

To bound the occurrence probability of failure condition B_2, we will analyze three situations separately.

- Query $(+, x, y)$ does not correspond to the last block of (the prefix-free encoding of) a random oracle query. In this case, condition B_2 occurs only if the uniformly random n-bit string $y \oplus z$ (with z chosen by the simulator), collides with one of q_E n-bit strings corresponding to other queries.

- Both (x, y, z) and (x', y', z') form last blocks of random oracle queries. In this case, condition B_2 is exactly the event that two random oracle outputs collide.

- The triple (x, y, z) forms the last block of a random oracle query, but (x', y', z') does not. In this case, $y' \oplus z'$ is a random n-bit string chosen by the simulator. Hence, condition B_2 corresponds to the random oracle output $y \oplus z$ collides with a random n-bit string chosen by the simulator.

Hence, we can bound the occurrence probability of condition B_2 by the birthday bound over $(q_E + q_F)$ uniformly random n-bit strings.

The simulator checks for condition B_3 only if it chooses the response independently. In this case, the occurrence probability of this failure condition can be bounded by the $(q_E 2/2^n)$. We do not force the simulator to check for condition B_3 when it is forced to be consistent with the random oracle. This is because the distinguisher can force this condition using RO queries, but this does not help since we use a prefix-free encoding before applying the Merkle-Damgård construction.

If an inverse query $(-, x, z)$ is made to the simulator S_0, the it chooses a response $y \in \{0, 1\}^n$ to this query similar to the original simulator S and checks for the following failure conditions:

1. *Condition C_1.* It is the case that $y = IV$ or $y \oplus z = IV$, where IV is the n-bit initialization vector.

2. *Condition C_2.* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', z') \neq (x, z)$, such that $y' \oplus z' = y \oplus z$.

3. *Condition C_3.* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', z') \neq (x, z)$, such that $y \oplus z = y'$ or $y' \oplus z' = y$.

In the case of inverse queries, the simulator always independently chooses random responses to any new queries and fails if any of the conditions C_1, C_2 or C_3 holds, and hence estimating the occurrence probability of these failure conditions is straightforward. The reasons for avoiding the conditions C_1, C_2 and C_3 are similar to those given above for B_1, B_2 and B_3.

Let G_3 denote the event that the distinguisher outputs 1 in game 3, i.e. $\Pr[G_3] = \Pr\left[D^{R_0^F, S_0^F}(1^\lambda) = 1\right]$. The responses of the distinguisher in games 2 and 3 differ only in situations where the new simulator S_0 explicitly fails and the original simulator S does not. This event is identical with the event that any of the failure conditions hold for the responses of either simulator (both of which are identically distributed).

$$\begin{aligned} |\Pr[G_3] - \Pr[G_2]| &\leq 2 \cdot \Pr[B_1 \cup B_2 \cup B_3 \cup C_1 \cup C_2 \cup C_3 \text{ hold for a corresponding query.}] \\ &\leq \frac{2 \cdot (q_E + q_F) \cdot (2 \cdot (q_E + q_F) + 1)}{2^n} \\ &= \mathcal{O}\left(\frac{q^2}{2^n}\right) \end{aligned}$$

Game 4. In this game, we modify the relay algorithm and leave the ideal cipher simulator unchanged. The underlying idea is to make the responses of the relay algorithm directly dependent on the simulator. Thus, instead of giving the new relay algorithm R_1 an oracle access to the random oracle F, here it is given oracle access to the simulator S_0.

On a random oracle query X, the relay algorithm R_1 computes the prefix-free encoding of X, i.e. $g(X)$. It then applies the Davies-Meyer Merkle-Damgård construction to $g(X)$ by querying the simulator S_0. Thus the relay algorithm R_1 is essentially the same as the random oracle construction pf-MD_g^E, except that it is based on the simulator S_0 instead of the ideal cipher E.

Let G_4 denote the event that the distinguisher D outputs 1 when given oracle access to S_0 and R_1 in this game. Thus, we know that

$$\Pr[G_4] = \Pr[D^{R_1^{S_0}, S_0^F}(1^\lambda) = 1]$$

Now we will show that the view of the distinguisher D remains unchanged (upto a negligible additive factor) in the transformation from game 3 to game 4. We will assume that that maximum length of the prefix-free encoding $g(X)$ of a random oracle input X queried upon by the distinguisher is $\ell\kappa$. This claim is formally stated below:

Claim 3 *Let G_3 and G_4 denote the events that the distinguisher D outputs 1 in games 3 and 4, respectively. If q_E and q_F denote the number of ideal cipher and random oracle queries made by the distinguisher (respectively), then it is the case that*

$$|Pr[G_4] - Pr[G_3]| = \mathcal{O}\left(\frac{(q_E + \ell \cdot q_F)^2}{2^n}\right)$$

proof of claim 3: From the view of the distinguisher, the games 3 and 4 differ only if it detects any difference in the responses of the relay algorithm or the simulator in these two games. We will prove that such a difference in the responses is impossible unless the simulator S_0 fails in either game 3 or 4. We start by demonstrating a few useful properties of the responses of the simulator S_0.

Claim 4 *If the simulator S_0 does not explicitly fail, then there are no two different sequences of κ-bit blocks $x_1 \dots x_m$ and $x'_1 \dots x'_p$ with corresponding triples $(x_1, y_1, z_1) \dots (x_m, y_m, z_m)$ and $(x'_1, y'_1, z'_1) \dots (x'_p, y'_p, z'_p)$ in table $\mathcal{T}$ such that:*

- *Both* $x_1 \parallel \ldots \parallel x_m$ *and* $x'_1 \parallel \ldots \parallel x'_p$ *constitute valid prefix-free encodings of random oracle inputs.*

- *It is the case that* $y_1 = y'_1 = IV$*, and for each* $s = 1 \ldots m$ *and* $s' = 1 \ldots p$*,* $y_s = y_{s-1} \oplus z_{s-1}$ *and* $y'_{s'} = y'_{s'-1} \oplus z'_{s'-1}$*.*

- *There is a* $s \in \{1, m\}$ *such that* $(x_s, y_s, z_s) = (x'_p, y'_p, z'_p)$*.*

proof of claim 4: We will prove this claim by performing an induction on the number of queries made to the simulator S_0, and show that unless the simulator explicitly fails, such sequence of triples cannot exist in the table $\mathcal{T}$ maintained by it. When no queries have been made, then the claim is vacuously true. Assume that it holds when q queries have already been made to the simulator S_0.

Say there are two sequences of κ-bit blocks $x_1 \ldots x_m$ and $x'_1 \ldots x'_p$ that satisfy the properties mentioned in the statement of the claim after the $(q+1)^{th}$ query. We can deduce that there are two subsequences of κ-bit blocks $x_{j-r} \ldots x_j$ and $x'_{p-r} \ldots x'_p$ such that:

$$\forall s \in \{0, r\} \ : \ (x_{j-s}, y_{j-s}, z_{j-s}) = (x'_{p-s}, y'_{p-s}, z'_{p-s})$$

If $r < j - 1$ and $r < p - 1$, then consider the triples $(x_{j-r-1}, y_{j-r-1}, z_{j-r-1})$ and $(x'_{p-r-1}, y'_{p-r-1}, z'_{p-r-1})$. Since $y_{j-r} = y'_{p-r}$, we can deduce that $y_{j-r-1} \oplus z_{j-r-1}) = y'_{p-r-1} \oplus z'_{p-r-1}$. Without loss of generality, assume that the query corresponding to the triple $(x_{j-r-1}, y_{j-r-1}, z_{j-r-1})$ was made after the one corresponding to $(x'_{p-r-1}, y'_{p-r-1}, z'_{p-r-1})$. If this query was a forward query then the simulator S_0 would have explicitly failed because of failure condition B_2. If this was an inverse query then the simulator would have failed because of failure condition C_2.

Now consider the case that $r = p - 1$ but $r < j - 1$. In this case, if the triple $(x_{j-r-1}, y_{j-r-1}, z_{j-r-1})$ was generated as a result of a forward query, then the simulator S_0 would have explicitly failed because of failure condition B_1 since $z_{j-r-1} \oplus y_{j-r-1} = y_{j-r} = y'_1 = IV$. If this triple was generated due to an inverse query then the simulator will fail because of failure condition C_1. The case when $r = j - 1$, but $r < p - 1$ is similar.

Lastly, if $r = p - 1 = j - 1$ then we have that $\forall s \in \{1, p\} \ : \ (x_s, y_s, z_s) = (x'_s, y'_s, z'_s)$. But this implies that $x'_1 \parallel \ldots \parallel x'_p$ is a prefix of $x_1 \parallel \ldots \parallel x_m$, which is not possible since they are encodings of two different inputs using the prefix-free encoding g.

Hence, we can conclude that there can be no such sequence of κ-bit blocks $x_1 \parallel \ldots \parallel x_m$ and $x'_1 \parallel \ldots \parallel x'_p$ if the simulator S_0 does not explicitly fail. □

Next we show that if the distinguisher wishes to find the random oracle output for an input $X \in \{0,1\}^*$, such that $g(X) = x_1 \parallel \ldots \parallel x_s$, by making queries to the simulator S_0 to compute the Davies-Meyer Merkle-Damgård construction applied to $x_1 \parallel \ldots \parallel x_s$, then the only way it can do so is by making the ordered sequence of forward queries $(+, x_1, y_1) \ldots (+, x_s, y_s)$.

Claim 5 *Consider any sequence of* κ*-bit blocks* $x_1 \ldots x_s$*, with corresponding triples* $(x_1, y_1, z_1) \ldots (x_s, y_s, z_s)$ *in the table* $\mathcal{T}$ *maintained by the simulator* S_0*, such that:*

- $x_1 \parallel \ldots \parallel x_s$ *is a valid encoding of a random oracle input* X *under the prefix-free encoding* g*.*

- $y_1 = IV$*, and for all* $j \in \{2, s\}$ *it is the case that* $y_j = y_{j-1} \oplus z_{j-1}$*.*

If the simulator S_0 *does not explicitly fail then it must be the case that the triples* $(x_1, y_1, z_1) \ldots (x_s, y_s, z_s)$ *were stored as a result of the ordered sequence of queries* $(+, x_1, y_1) \ldots (+, x_s, y_s)$*.*

proof of claim 5: To the contrary, assume that the sequence of queries that resulted in the triples $(x_1, y_1, z_1) \dots (x_s, y_s, z_s)$ was not the sequence of forward queries given in the claim statement. We can then deduce that at least one of the following must be true regarding this sequence of queries:

1. For $j \in \{1, s-1\}$, a forward query $(+, x_j, y_j)$ was made when the triple $(x_{j+1}, y_{j+1}, z_{j+1})$ already existed in the table $\mathcal{T}$.

2. For $j \in \{2, s\}$, an inverse query was made $(-, x_j, z_j)$ when the triple $(x_{j-1}, y_{j-1}, z_{j-1})$ already existed in the table $\mathcal{T}$.

3. The triple (x_1, y_1, z_1) was generated as a result of an inverse query $(-, x_1, z_1)$.

In the first case, we know from claim 4 that the triple (x_j, y_j, z_j) cannot be the last block of the prefix-free encoding of another query if the simulator S_0 does not fail. Hence it must be the case that the response to the corresponding query was randomly chosen by the simulator itself (independent of the random oracle). But since the triple $(x_{j+1}, y_{j+1}, z_{j+1})$ already exists in table $\mathcal{T}$, the simulator will explicitly fail from condition B_3 since the equality $y_j \oplus z_j = y_{j+1}$ holds. In the second case, the simulator will explicitly fail due to failure condition C_3 since the equality $y_j = z_{j-1} \oplus y_{j-1}$ holds. In the last case the simulator fails due to failure condition C_1.

Thus the simulator S_0 explicitly fails in either of the above situations, and the only sequence of queries possible is the one mentioned in the statement of the claim. □

Next, we wish to show that the responses of the relay algorithm R_0 and the simulator S_0 are always consistent in game 3. Note that in game 4, the relay algorithm R_1 responds to all queries by computing the RO construction pf-MD$_g^{S_0}$, with the ideal cipher E replaced by the simulator S_0. On the other hand, the responses of the relay algorithm R_0 could be inconsistent with the simulator S_0 (i.e. the distinguisher may get a different output to a random oracle input depending on whether it uses the construction pf-MD$_g^{S_0}$ itself, or queries the relay algorithm R_0). We show that such a situation is impossible unless the simulator S_0 fails.

Claim 6 *In game 3, if the simulator S_0 never fails then there is no sequence of κ-bit blocks $x_1 \dots x_j$, with corresponding triples $(x_1, y_1, z_1) \dots (x_j, y_j, z_j)$ such that:*

- *The bit string $x_1 \parallel \dots \parallel x_j$ is a valid prefix-free encoding of a random oracle input.*
- *$y_1 = IV$ and for $l = 2 \dots j$ it is the case that $y_l = y_{l-1} \oplus z_{l-1}$.*
- *To the random oracle query $g^{-1}(x_1 \parallel \dots \parallel x_j)$, the response of the relay algorithm R_0 is different from $y_j \oplus z_j$.*

proof of claim 6: To any random oracle query X, the relay algorithm R_0 always responds with the random oracle output $F(X)$. Thus the situation described in the statement of the claim occurs if and only if the simulator responds to its queries (corresponding to the κ-bit blocks in $g(X) = x_1 \parallel \dots \parallel x_j$) in such a way that $y_j \oplus z_j \neq F(X)$.

From claim 5, we can deduce that if the distinguisher is to compute the Davies-Meyer Merkle Damgård output on $g(X) = x_1 \parallel \dots x_j$, then the only way to do this is to make the ordered sequence of queries $(+, x_1, y_1), \dots, (+, x_j, y_j)$ unless the simulator S_0 fails. Here $y_1 = IV$ and for each $i = 2 \dots j$ we have $y_i = y_{i-1} \oplus z_{i-1}$. Hence the simulator S_0 already has the triples $(x_1, y_1, z_1) \dots (x_{j-1}, y_{j-1}, z_{j-1})$ in its table $\mathcal{T}$ when the query $(+, x_j, y_j)$ is made.

If the response of the simulator S_0 to the query $(+, x_j, y_j)$ is different from $F(X) \oplus y_j$, then it must be the case that the simulator is unable to give this response because of some other constraint. But from claim 4, we can deduce that the block x_j cannot be part of any other valid Davies-Meyer Merkle-Damgård computation sequence unless the simulator S_0 fails. Thus there can be no other constraint of the response of S_0 if it has not explicitly failed.

Thus the responses of S_0 are always consistent with the relay algorithm R_0 in game 3, if it does not fail.

□

In fact, we can use the same argument as in proof of claim 6 to show that the responses of S_0 are consistent with the random oracle F in game 4 as well (that is, the result of applying Davies-Meyer Merkle-Damgård construction using S_0 to $g(X)$ is the same as $F(X)$).

From the above, we can deduce that if the simulator S_0 does not fail in game 4, then the responses of the relay algorithm R_1 are identical to the responses of the relay algorithm R_0. And since we are using the same simulator S_0 in both games, and have shown that the responses of the simulator and the two relay algorithms are consistent in the two games, we can also deduce that the view of the distinguisher D remains unchanged from game 3 to game 4 if the simulator S_0 does not fail in either of the two games.

Hence, we can finally complete the proof of claim 3 by observing that if the maximum length of the prefix-free encoding of a random oracle query made by D is $\ell \cdot \kappa$ then,

$$\begin{aligned}
|\Pr[G_4] - \Pr[G_3]| &\leq \Pr[S_0 \text{ fails in game 3}] + \Pr[S_0 \text{ fails in game 4}] \\
&= \mathcal{O}\left(\frac{(q_E + q_F\ell)^2}{2^n}\right) \\
&= \mathcal{O}\left(\frac{(q\ell)^2}{2^n}\right)
\end{aligned}$$

□

Game 5. In this game, we modify the simulator S_0 so as to make its responses independent of the random oracle F. For this purpose, we remove the random oracle F from this game entirely and the new simulator S_1 always chooses a random n-bit response, even in situations where S_0 would have consulted the RO F. We also remove all the failure conditions from the new simulator S_1.

Thus on a forward query $(+, x, y)$, the new simulator S_1 checks if there is a triple (x, y, z) in its table $\mathcal{T}$. If it finds such a triple then it responds with the n-bit string z. Otherwise it chooses a uniformly random n-bit string z and sends this as its response, while storing the triple (x, y, z) in $\mathcal{T}$. On an inverse query $(-, x, z)$, it similarly checks to see if there is a triple (x, y, z) in its table $\mathcal{T}$. If it finds such a triple, it responds with y, else it chooses a uniformly random n-bit response y.

Now we will show that the view of the distinguisher D does not change by a non-negligible amount from game 4 to game 5. In fact, if we can show that the responses of the simulators S_0 and S_1 seem almost identical to the distinguisher D, then we will be done. But the responses of these two simulators are identical apart from the failure conditions which are used by S_0 and not by S_1 (even when S_0 consults the random oracle, its response is still uniformly distributed). Thus, the distinguisher does not notice a difference between these games if:

- In game 4, the simulator S_0 does not fail.
- In game 5, the simulator S_1 does not respond to its queries in such a manner that its satisfy one of the failure conditions specified in the definition of S_0.

In fact, these two events are identical in terms of their probability of occurrence since the distribution of the responses of the two simulators is identical. Let G_5 denote the event that the distinguisher D outputs 1 in game 5, so that $\Pr[G_5] = \Pr[D^{R_1^{S_1}, S_1}(1^\lambda) = 1]$. Then we can deduce that,

$$\begin{aligned} |\Pr[G_5] - Pr[G_4]| &\leq \Pr[S_0 \text{ fails in game 4}] + \Pr[S_1 \text{ should have failed in game 5}] \\ &= \mathcal{O}\left(\frac{q^2\ell^2}{2^n}\right) \end{aligned}$$

Game 6. This is the final game of our argument. Here we finally replace the simulator S_1 with the ideal cipher E. Since the relay algorithm R_1 simply implemented the construction pf-MD$_g^{S_1}$, it will be the same as the RO construction pf-MD$_g^E$ in this game. Hence this game is same as the view of the distinguisher in the *ideal cipher model*.

The outputs of the ideal cipher E are not distributed uniformly like the responses of S_1. Hence the distinguisher may be able to differentiate between games 5 and 6 if it can detect this. However, this happens only if S_1 outputs an input/output collision for the same ideal cipher key. The probability of this event is easily seen to be at most the birthday bound. Let G_6 denote the probability that the distinguisher outputs 1 in game 6, so that $\Pr[G_6] = \Pr[D^{\text{pf-MD}_g^E, E}(1^\lambda) = 1]$. Then we can deduce that

$$|\Pr[G_6] - \Pr[G_5]| = \mathcal{O}\left(\frac{q^2\ell^2}{2^n}\right)$$

Now we can complete the proof of lemma A.2 by combining games 1 to 6, and observing that game 1 is same as the random oracle model while game 6 is same as the ideal cipher model. Hence we can deduce that

$$\left|\Pr\left[D^{F, S^F}(1^\lambda) = 1\right] - \Pr\left[D^{\text{pf-MD}_g^E, E}(1^\lambda) = 1\right]\right| = \mathcal{O}\left(\frac{q^2\ell^2}{2^n}\right)$$

$\square$

Now we will prove the indifferentiability of the second random oracle construction chop-MD$_s^E$. Recall that this construction essentially applies the plain Davies-Meyer Merkle-Damgård construction (using the ideal cipher E) to the input and then removes a non-trivial number s of the output bits.

Lemma A.3 *The Merkle-Damgård construction with truncated output chop-MD$_s^E$ based on the Davies-Meyer construction applied to an ideal cipher $E : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ is (t_D, t_S, q, ϵ)-indifferentiable from a random oracle $F : \{0,1\}^* \to \{0,1\}^{n-s}$ in the ideal cipher model for E, for any t_D and $t_S = \mathcal{O}(q^2 \cdot \kappa))$, with $\epsilon = 2^{-n} \cdot \ell^2 \cdot \mathcal{O}(q^2)$.*

Proof:
We will assume that the random oracle inputs provided to the construction chop-MD$_s^E$ are all of length, that is a multiple of the block length κ. In actual implementation, this can be achieved by applying an appropriate encoding scheme to the input, such as appending a 1 followed by a sufficient number of 0s to the input.

The Simulator. The simulator S accepts either forward ideal cipher queries, $(+, x, y)$, or inverse ideal cipher queries, $(-, x, z)$, such that $x \in \{0,1\}^\kappa$ and $y, z \in \{0,1\}^n$. In either case, the simulator responds with a n-bit string that is interpreted as $E_x(y)$ in case of a forward query $(+, x, y)$, and as $E_x^{-1}(z)$ in case of an inverse query $(-, x, z)$. The simulator maintains a table $\mathcal{T}$ consisting of triples $(x, y, z) \in \{0,1\}^\kappa \times \{0,1\}^n \times \{0,1\}^n$, such that it either responded with z to a forward query $(+, x, y)$ or with y to an inverse query $(-, x, z)$.

On getting a forward query $(+, x, y)$, the simulator searches its table $\mathcal{T}$ for a triple of the form (x, y, z). If it finds such a triple then it responds with the n-bit string z otherwise it needs to choose a fresh response to this query. It proceeds by searching its table $\mathcal{T}$ for a sequence of triples $(x_1, y_1, z_1) \dots (x_i, y_i, z_i)$ such that:

- It is the case that $y_1 = IV$, where IV denotes the initialization vector used in the construction chop-MD$_s^E$.
- For each $j = 2 \dots i$, it holds that $y_j = y_{j-1} \oplus z_{j-1}$.
- It is the case that $y = y_i \oplus z_i$, where y is the ideal cipher input from the current forward query.

Note that for an empty sequence of triples, i.e. when just considering the κ-bit block x from the current query, we only need to check if $y = IV$ and none of the above conditions make sense.

If the simulator finds such a sequence of triples, then it needs to give a response that is consistent with the random oracle output on $x_1 \parallel \dots \parallel x_i \parallel x$. Thus, the simulator makes this RO query to get the output $Y = F(x_1 \parallel \dots \parallel x_i \parallel x)$. It then samples a uniformly random s-bit string Y' and outputs the n-bit string $z = (Y \parallel Y') \oplus y$. If the simulator does not find any such sequence of triples in its table $\mathcal{T}$, then it samples a uniformly distributed random n-bit string z and sends z as its response. In either case, it inserts the triple (x, y, z) in its table $\mathcal{T}$.

On an inverse query $(-, x, z)$, the simulator S searches its table $\mathcal{T}$ for a triple (x, y, z) with arbitrary y. If it finds such a triple, then it responds with y. Otherwise, the simulator S chooses a uniformly distributed random n-bit string y and responds with y. It then inserts the triple (x, y, z) in its table $\mathcal{T}$.

Proof of Indifferentiability. We need to prove that the distinguisher cannot tell apart the two scenarios, one where it has oracle access to the random oracle F and the simulator S, and the other where it has oracle access to the RO construction chop-MD$_s^E$ and the ideal cipher E. As in the case of the prefix-free Merkle-Damgård construction, the proof involves a hybrid argument.

GAME 1. This is the *random oracle model*, and the distinguisher D is given oracle access to the random oracle F and the ideal cipher simulator S described above. Let G_1 denote the event that the distinguisher D outputs 1 in this game.

$$\Pr[G_1] = \Pr[D^{F,S^F}(1^\lambda) = 1]$$

GAME 2. In this game, the distinguisher is given oracle access to a *relay algorithm* R_0 instead of direct oracle access to F. The *relay algorithm*, in turn, has oracle access to the random oracle F. On a random oracle query X, the relay algorithm simply makes the same query to F and responds with the RO output $F(X)$. Let G_2 denote the event that D outputs 1 in game 2. Since the view of the distinguisher remains unchanged in this game, we can deduce that

$$\Pr[G_2] = \Pr[D^{R_0^F,S^F}(1^\lambda) = 1] = \Pr[G_1]$$

GAME 3. In this game, we modify the simulator S. In particular, we restrict the responses of the simulator such that they never satisfy certain specific failure conditions. If the simulator comes up with a response that results in its responses satisfying one of these conditions, then it explicitly fails instead of sending this response.

These failure conditions, that the new simulator S_0 checks for, describe certain dependencies among its responses that could be exploited by a distinguisher. In response to a forward query $(+, x, y)$, the new simulator S_0 starts by choosing a n-bit response $z \in \{0, 1\}^n$ in the same way as the original simulator S. It then checks if one of the following conditions is satisfied:

1. *Condition B_1:* It is the case that $z \oplus y = IV$, where IV is the initialization vector used in the RO construction chop-MD$_s^E$.
2. *Condition B_2:* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', y') \neq (x, y)$, such that $y' \oplus z' = y \oplus z$.
3. *Condition B_3:* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', y') \neq (x, y)$, such that $y \oplus z = y'$.

If the response z, whether S_0 chooses a uniformly random z or z is chosen to be consistent with the RO F on some query, is such that one of these conditions is satisfied, then the simulator S_0 explicitly fails.

On a new inverse query $(-, x, z)$, the simulator S_0 again chooses its response $y \in \{0,1\}^n$ in the same way as S. It then checks if the following conditions, and fails if any one of them is satisfied:

1. *Condition C_1:* It is the case that $y = IV$ or $y \oplus z = IV$, where IV is the initialization vector used in the RO construction chop-MD$_s^E$.
2. *Condition C_2:* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', z') \neq (x, z)$, such that $y' \oplus z' = y \oplus z$.
3. *Condition C_3:* There is a triple $(x', y', z') \in \mathcal{T}$, with $(x', z') \neq (x, z)$, such that either $y \oplus z = y'$ or $y' \oplus z' = y$.

Next we will estimate the occurrence probability for each of the above failure conditions. We start by noting that the probability that one of the conditions C_1, C_2 and C_3 holds can be readily estimated, since the simulator always chooses uniformly random responses to inverse queries.

In the case of a forward query, the simulator might be forced to choose its response so as to maintain consistency with the random oracle F. Hence the distinguisher could find out $(n - s)$ bits of the response of the simulator by making a random oracle query. Thus, it is not as straightforward to estimate the occurrence probabilities for the failure conditions B_1, B_2 and B_3. Let the number of random oracle queries made by D be q_F, and let the number of ideal cipher queries be q_E (hence the total number of queries $q = q_E + q_F$)

We can bound the occurrence probability of event B_1 easily, since it is the probability that at least one of $(q_E + q_F)$ uniformly random n-bit strings is IV. In order to estimate the occurrence probability of failure condition B_2, we will analyze three situations separately.

- Query $(+, x, y)$ does not correspond to the last block of a random oracle input. In this case, condition B_2 holds only if the uniformly random n-bit string $y \oplus z$ is equal to one of upto q_E n-bit strings corresponding to previous queries.
- Both (x, y, z) and (x', y', z') correspond to last blocks of random oracle inputs, and the simulator adjusted its response according to the RO output in each case. In this case, condition B_2 implies a collision among the two random oracle outputs as well as a collision among the remaining s uniformly random bits chosen by the simulator in each case.
- The triple (x, y, z) forms the last block of a random oracle input and the simulator adjusts its response z accordingly, but (x', y', z') does not. In this case, $y' \oplus z'$ is a random n-bit string chosen by the simulator. Here, the condition B_2 corresponds to a random oracle output along with the extra s random bits chosen by the simulator colliding with another randomly and independently chosen n-bit string chosen by the simulator.

From the above, we can deduce that the occurrence probability of failure condition B_2 can be bounded by the birthday bound over $(q_E + q_F)$ random n-bit strings.

In order to bound the occurrence probability of failure condition B_3, we note that the simulator S_0 chooses at least s random and independent bits in its response (even if it is forced to make the remaining $(n - s)$ bits consistent with the random oracle). Thus the occurrence probability of condition B_3 can be bounded by the birthday bound over $(q_E + q_F)$ independent and random s-bit random strings.

Let G_3 denote the event that the distinguisher D outputs 1 in this game, i.e. $\Pr[G_3] = \Pr\left[D^{R_0^F, S_0^F}(1^\lambda) = 1\right]$. The responses of the distinguisher in games 2 and 3 differ only if the simulator S_0 exits because of one of the failure conditions in game 3. This event is identical with the event that at least one of the failure conditions hold for the responses of either simulators (in which case S_0 exits while S does not).

$$\begin{aligned}
|\Pr[G_3] - \Pr[G_2]| &\leq \Pr[B_1 \cup B_2 \cup B_3 \cup C_1 \cup C_2 \cup C_3 \text{ hold for a corresponding query.}] \\
&= \mathcal{O}\left(\frac{q^2}{2^s}\right)
\end{aligned}$$

Game 4. In this game, we modify the relay algorithm but leave the ideal cipher simulator S_0 unchanged. The underlying idea is to make the responses of the relay algorithm directly dependent on the simulator. Thus, instead of giving the new relay algorithm R_1 oracle access to the random oracle F, here it is given oracle access to the simulator S_0. It responds to a random oracle query X by computing the Davies-Meyer Merkle-Damgård construction using input X and then chops the same s bits from the output as in the case of the RO construction chop-MD_s^E.

Let G_4 denote the event that the distinguisher D outputs 1 in game 4. Thus we know that

$$\Pr[G_4] = \Pr\left[D^{R_1^{S_0}, S_0^F}(1^\lambda) = 1\right]$$

We will assume that the maximum length of a random oracle query made by the adversary is $\ell \cdot \kappa$. Now we will show that the view of the distinguisher changes by at most a negligible amount in the transition from game 3 to game 4. This claim is formally stated below.

Claim A.4 *Let G_3 and G_4 denote the events that the distinguisher outputs* 1 *in game* 3 *and game* 4*, respectively. Let q_E and q_F denote the number of ideal cipher and random oracle queries made by the distinguisher, then it is the case that*

$$|\Pr[G_4] - \Pr[G_3]| = \mathcal{O}\left(\frac{(q_E + q_F \cdot \ell)^2}{2^s}\right)$$

proof of claim A.4: The view of the distinguisher differs in games 3 and 4 only if it finds a difference in responses of either the relay algorithm or the simulator among the two games. We will show that such a difference is impossible, unless the simulator S_0 fails in at least one of the two games. Let us start by proving a few important properties of the simulator S_0 that are valid in both games 3 and 4.

Claim A.5 *If the simulator S_0 does not explicitly fail, then there are no two different sequences of κ-bit blocks $x_1 \dots x_m$ and $x'_1 \dots x'_p$ with corresponding triples $(x_1, y_1, z_1) \dots (x_m, y_m, z_m)$ and $(x'_1, y'_1, z'_1) \dots (x'_p, y'_p, z'_p)$ in the table $\mathcal{T}$ such that:*

- *It is the case that $y_1 = y'_1 = IV$, and for each $b = 2 \dots m$ and $b' = 2 \dots p$, it holds that $y_b = y_{b-1} \oplus z_{b-1}$ and $y'_{b'} = y'_{b'-1} \oplus z'_{b'-1}$.*
- *It is the case that $(x_m, y_m, z_m) = (x'_p, y'_p, z'_p)$.*

proof of claim A.5: We will prove this claim by performing an induction on the number of queries made to the simulator and show that unless the simulator S_0 fails, such sequences of triples cannot exist. When no queries have been made as yet, this claim is vacuously true. Let us assume that the claim is also true when q queries have been made to the simulator S_0.

Now say there exist two sequences of triples be $(x_1, y_1, z_1) \dots (x_m, y_m, z_m)$ and $(x'_1, y'_1, z'_1) \dots (x'_p, y'_p, z'_p)$, that satisfy the properties stated in the claim, after the $(q+1)^{th}$ query. Since we know that $(x_m, y_m, z_m) = (x'_p, y'_p, z'_p)$, we can deduce that there are two subsequences of κ-bit blocks $x_{m-r} \dots x_m$ and $x_{p-r} \dots x_p$ such that

$$\forall b \in \{0, r\} \ : \ (x_{m-b}, y_{m-b}, z_{m-b}) = (x'_{p-b}, y'_{p-b}, z'_{p-b})$$

If $r < m-1$ and $r < p-1$, then consider the triples $(x_{m-r-1}, y_{m-r-1}, z_{m-r-1})$ and $(x'_{p-r-1}, y'_{p-r-1}, z'_{p-r-1})$. Since $y_{m-r} = y'_{p-r}$, we can deduce that $y_{m-r-1} \oplus z_{m-r-1} = y'_{p-r-1} \oplus z'_{p-r-1}$. Without loss of generality, assume that the query corresponding to the triple $(x_{m-r-1}, y_{m-r-1}, z_{m-r-1})$ was made earlier than the one corresponding to $(x'_{p-r-1}, y'_{p-r-1}, z'_{p-r-1})$. If this query is a forward query, then the simulator S_0 would fail because of failure condition B_2. On the other hand, if this were an inverse query, then the simulator would have failed due to failure condition C_2.

Now consider the case that $r = p-1$ but $r < m-1$. In this case, if the triple $(x_{m-r-1}, y_{m-r-1}, z_{m-r-1})$ was generated as a result of a forward query then the simulator S_0 would have failed due to failure condition B_1 because $y_{m-r-1} \oplus z_{m-r-1} = y_{m-r} = y'_1 = IV$. If this triple were generated as a result of an inverse query then the simulator would have failed as a result of failure condition C_1 being true. The case when $r = m-1$ but $r < p-1$ is symmetrical

Lastly, it cannot be the case that $r = p-1$ as well as $r = m-1$, since the two bit strings $x'_1 \parallel \dots \parallel x'_p$ and $x_1 \parallel \dots \parallel x_m$ are different.

Hence we can conclude that there can be no such sequences of κ-bit blocks $x_1, \dots, x_m$ and $x_1, \dots, x'_p$ if the simulator does not explicitly fail. □

Next we show that if the distinguisher wishes to find the random oracle output for an input $X = x_1 \parallel \dots \parallel x_s$ by making queries to the simulator S_0 and computing the Davies-Meyer Merkle-Damgård construction, then the only way it can do so is by making the ordered sequence of forward queries $(+, x_1, y_1) \dots (+, x_s, y_s)$.

Claim A.6 *Consider any sequence of κ-bit blocks $x_1 \dots x_s$, with corresponding triples $(x_1, y_1, z_1) \dots (x_s, y_s, z_s)$ in the table $\mathcal{T}$ maintained by the simulator S_0, such that $y_1 = IV$ and for each $j = 2 \dots s$ it holds that $y_j = y_{j-1} \oplus z_{j-1}$. If the simulator S_0 does not explicitly fail then it must be the case that the triples $(x_1, y_1, z_1) \dots (x_s, y_s, z_s)$ are generated as a result of the ordered sequence of forward queries $(+, x_1, y_1) \dots (+, x_s, y_s)$.*

proof of claim A.6: To the contrary, assume that the triples $(x_1, y_1, z_1) \dots (x_s, y_s, z_s)$ were not generated as a result of the sequence of forward queries mentioned in the claim. We can then deduce that one of the following must be true regarding the actual sequence of queries that resulted in these triples:

1. For $j = 1 \dots (s-1)$, a forward query $(+, x_j, y_j)$ was made when the triple $(x_{j+1}, y_{j+1}, z_{j+1})$ already existed in the table $\mathcal{T}$.

2. For $j = 2 \dots s$, an inverse query $(-, x_j, z_j)$ was made when the triple $(x_{j-1}, y_{j-1}, z_{j-1})$ already existed in the table $\mathcal{T}$.

3. The triple (x_1, y_1, z_1) was generated as a result of an inverse query $(-, x_1, y_1)$.

In the first case, the simulator S_0 would fail since the failure condition B_3 holds. Indeed, we can deduce that $y_j \oplus z_j = y_{j+1}$. In the second case, the simulator explicitly fails because of failure condition C_3 since we know that $y_j = y_{j-1} \oplus z_{j-1}$. In the third and final case, the simulator would explicitly fail since the failure condition C_1 holds. Thus the only possible sequence of queries that could result in these triples is the one mentioned in the claim. $\square$

Now we will show that the responses of the relay algorithm R_0 in game 3 are consistent with those of the simulator S_0. Note that in game 4, the relay algorithm R_1 is designed in such a way that its responses are always consistent with S_0 while the relay algorithm R_0 is given oracle access to the random oracle F and may not be consistent with S_0. We show that such inconsistency is impossible unless the simulator S_0 explicitly fails.

Claim A.7 *In game 3, if the simulator S_0 never fails then there is no sequence of κ-bit blocks $x_1 \ldots x_j$, with corresponding triples $(x_1, y_1, z_1) \ldots (x_j, y_j, z_j)$ such that:*

- *$y_1 = IV$ and for $l = 2 \ldots j$ it is the case that $y_l = y_{l-1} \oplus z_{l-1}$.*
- *To the random oracle query $X = x_1 \parallel \ldots \parallel x_j$, the response of the relay algorithm R_0 is different from the $(n-s)$ bits of $y_j \oplus z_j$ that are not chopped in the construction chop-MD_s^E.*

proof of claim A.7: To any random oracle query X, the relay algorithm R_0 always responds with the random oracle output $F(X)$. Thus the situation described in the statement of the claim occurs if and only if the simulator responds to its queries (corresponding to the κ-bit blocks in $X = x_1 \parallel \ldots \parallel x_j$) in such a way that $y_j \oplus z_j \neq F(X)$.

From claim A.6, we can deduce that if the distinguisher is to compute the RO output on $X = x_1 \parallel \ldots x_j$ by querying the simulator, then the only way to do this is to make the ordered sequence of queries $(+, x_1, y_1), \ldots, (+, x_j, y_j)$ unless the simulator S_0 fails. Here $y_1 = IV$ and for each $i = 2 \ldots j$ we have $y_i = y_{i-1} \oplus z_{i-1}$. Hence the simulator S_0 already has the triples $(x_1, y_1, z_1) \ldots (x_{j-1}, y_{j-1}, z_{j-1})$ in its table $\mathcal{T}$ when the query $(+, x_j, y_j)$ is made.

If the response of the simulator S_0 to the query $(+, x_j, y_j)$ is different from $F(X) \oplus y_j$, then it must be the case that the simulator is unable to give this response because of some other constraint. But from claim A.5, we can deduce that the block x_j cannot be the last block of any other valid Davies-Meyer Merkle-Damgård computation sequence unless the simulator S_0 fails. Thus there can be no other constraint of the response of S_0 if it has not explicitly failed. $\square$

Thus we have shown that, even though the relay algorithm R_0 simply forwards the random oracle outputs in game 3, its responses are still consistent with the responses of simulator S_0 in that game. Another way to look at this claim would be to note that the responses of the simulator S_0 are always consistent with the random oracle outputs, unless it explicitly fails.

Hence, it is easy to see that if the simulator S_0 does not fail in either of the games 3 or 4, the view of the distinguisher does not change in going from one game to the other. Now we can complete the proof of claim A.4 by observing that if the longest RO query made by the distinguisher D consists consists of at most ℓ κ-bit blocks then

$$\begin{aligned}
|\Pr[G_4] - \Pr[G_3]| &\leq \Pr[S_0 \text{ fails in game 3}] + \Pr[S_0 \text{ fails in game 4}] \\
&= \mathcal{O}\left(\frac{(q_E + q_F \cdot \ell)^2}{2^s}\right) \\
&= \mathcal{O}\left(\frac{(q \cdot \ell)^2}{2^s}\right)
\end{aligned}$$

□

Game 5. In this game, we modify the simulator S_0 so as to make the view of the distinguisher independent of the random oracle F. For this purpose, we introduce a new simulator S_1 that does not have oracle access to the random oracle F, and always outputs a n-bit random response to all new forward as well as inverse queries even in cases where S_0 would have maintained consistency with F. We also remove all failure conditions from the simulator S_1.

On a forward query $(+, x, y)$, the new simulator S_1 checks if there already exists a triple (x, y, z) in its table $\mathcal{T}$. If it finds such a triple, then it responds with the n-bit string z. If not, then it chooses a uniformly random n-bit string z and sends this as its response, while storing the triple (x, y, z) in $\mathcal{T}$. On an inverse query $(-, x, z)$, it similarly checks to see if there is a triple (x, y, z) in its table $\mathcal{T}$. If it finds such a triple, it responds with y otherwise it chooses a uniformly random n-bit response y.

Now we will show that the view of the distinguisher does not change by a non-negligible amount in going from game 4 to game 5. Note that if we can show that the responses of the simulators S_0 and S_1 are indistinguishable, then we will be done. But in the view of the distinguisher, these two simulators are identical apart from the failure conditions used by S_0 but not by S_1. Thus, we can deduce that the distinguisher does not notice a difference between games 4 and 5 unless:

- In game 4, simulator S_0 explicitly fails.
- In game 5, simulator S_1 responds with an output such that it satisfies one of the failure conditions (for which S_0 would have failed).

Since the simulator S_1 always chooses a uniformly random n-bit response to every query, we can easily bound the occurrence probability of any of the failure conditions using the birthday bound. Let G_5 denote the event that the distinguisher D outputs 1 in game 5, so that $\Pr[G_5] = \Pr[D^{R_1^{S_1}, S_1}(1^\lambda) = 1]$. Thus we can deduce that

$$\begin{aligned}
|\Pr[G_5] - \Pr[G_4]| &\leq \Pr[S_0 \text{ fails in game 4}] + \Pr[S_1 \text{ satisfies a failure condition in game 5}] \\
&= \mathcal{O}\left(\frac{(q \cdot \ell)^2}{2^s} + \frac{(q \cdot \ell)^2}{2^n}\right) \\
&= \mathcal{O}\left(\frac{q^2 \ell^2}{2^s}\right)
\end{aligned}$$

Game 6. This is the final game of our proof. In this game, we replace the simulator S_1 with the ideal cipher E. Since the relay algorithm R_1 essentially implements the RO construction chop-MD_s^E, the view of the distinguisher in this game is essentially its view in the *ideal cipher model.*

The outputs of the ideal cipher E are not uniformly distributed as are the responses of S_1. However, the distinguisher can differentiate between the two only if the simulator S_1 outputs a collision for the same ideal cipher key. The occurrence probability of this event can be easily bounded using the birthday bound. Thus let G_6 be the event that the distinguisher D outputs 1 in this game, so that $\Pr[G_6] = \Pr[D^{\text{chop-MD}_s^E, E}(1^\lambda) = 1]$ and we can deduce that

$$|\Pr[G_5] - \Pr[G_4]| \leq \mathcal{O}\left(\frac{q^2 \ell^2}{2^n}\right)$$

Now we can complete the proof of lemma A.3 by combining games 1 to 6, and observing that game 1 is same as the random oracle model while game 6 is the same as the ideal cipher model. Hence we can deduce that

$$\left|\Pr\left[D^{F,S^F}(1^\lambda)=1\right]-\Pr\left[D^{\text{chop-MD}_s^E,E}(1^\lambda)=1\right]\right|=\mathcal{O}\left(\frac{q^2\ell^2}{2^n}\right)$$

□

Lemma A.8 *The NMAC construction NMAC^{E1,E2} that uses two independent ideal block ciphers $E1:\{0,1\}^\kappa\times\{0,1\}^n\to\{0,1\}^n$ and $E2:\{0,1\}^{\kappa'}\times\{0,1\}^{n'}\to\{0,1\}^{n'}$ is (t_D,t_S,q,ϵ)-indifferentiable from a random oracle $F:\{0,1\}^*\to\{0,1\}^{n'}$ in the ideal block cipher model for $E1$ and $E2$, for any t_D and $t_S=\mathcal{O}(q^2)$, with $\epsilon=2^{-\min(n,n')}\cdot\ell^2\cdot\mathcal{O}(q^2)$ ($\ell\kappa$ is the maximum length of an RO query made by the distinguisher).*

Proof: Recall that the construction NMAC^{E1,E2} essentially applies the Davies-Meyer Merkle-Damgård construction using the block cipher $E1$ to the input $x_1\parallel\ldots\parallel x_\ell$ to get the final output Y. It then applies the Davies-Meyer compression function using $E2$ to this output Y. We will assume for simplicity that the output length n of $E1$ is the same as the key length κ' of $E2$[3]. We will use the initialization vector IV for the Davies-Meyer Merkle-Damgård construction applied to $E1$, and use initialization vector IV' for the Davies-Meyer construction with $E2$.

The Simulator. Let us start by describing the simulator for the ideal block ciphers $E1$ and $E2$ in the random oracle model with an actual random oracle F. The simulator gets forward/inverse queries for either of the block ciphers $E1$ and $E2$. Thus the queries that simulator S responds to are as follows:

1. $(1,+,x,y)$: A forward $E1$ query, where $(x,y)\in\{0,1\}^\kappa\times\{0,1\}^n$. The expected response is $E1_x(y)$.
2. $(1,-,x,z)$: An inverse $E1$ query, where $(x,z)\in\{0,1\}^\kappa\times\{0,1\}^n$. The expected response is $E1_x^{-1}(z)$.
3. $(2,+,x,y)$: A forward $E2$ query, where $(x,y)\in\{0,1\}^{\kappa'}\times\{0,1\}^{n'}$. The expected response is $E2_x(y)$.
4. $(2,-,x,z)$: An inverse $E2$ query, where $(x,z)\in\{0,1\}^{\kappa'}\times\{0,1\}^{n'}$. The expected response is $E2_x^{-1}(z)$.

The simulator S also maintains a table $\mathcal{T}$ in which it records all previous queries that were made to it, along with the responses it gave to each. Thus, it records an entry $(1,x,y,z)$ in $\mathcal{T}$ for every forward (resp. inverse) query of the form $(1,+,x,y)$ (resp. $(1,-,x,z)$) to which it responded with z (resp. y). On the other hand, it records an entry $(2,x,y,z)$ in $\mathcal{T}$ for every forward (resp. inverse) query of the form $(2,+,x,y)$ (resp. $(2,-,x,z)$) to which it responded with z (resp. y).

On getting a forward query $(1,+,x,y)$, the simulator first checks if there is a tuple $(1,x,y,z)$ in its table $\mathcal{T}$. If this is the case, then the simulator S responds with z, otherwise it chooses a uniformly random n-bit string z and sends this as its response. It then records $(1,x,y,z)$ in its table $\mathcal{T}$.

Similarly, on getting an inverse query $(1,-,x,z)$, it first searches its table $\mathcal{T}$ for a tuple $(1,x,y,z)$. If it finds such a tuple, then it responds with z, otherwise it sends a uniformly random n-bit string y as its response and stores $(1,x,y,z)$ in its table $\mathcal{T}$.

On a query $(2,+,x,y)$, the simulator S again checks if there is a tuple $(2,x,y,z)\in\mathcal{T}$. If this is the case then it responds with z. If it cannot find such a tuple, then the simulator checks if $y=IV'$, where IV' is the initialization vector used in the second part of the construction NMAC^{E1,E2}. If $y\neq IV'$, then the simulator simply sends back a random response $z\in\{0,1\}^{n'}$ and stores $(2,x,y,z)$ in $\mathcal{T}$. On the other hand, if $y=IV'$, then the simulator S searches its table $\mathcal{T}$ for a sequence of tuples $(1,x_1,y_1,z_1),\ldots,(1,x_i,y_i,z_i)$ such that the following conditions hold:

[3]one can use suitable padding techniques to expand Y from n bits to κ' bits

- It is the case that $y_1 = IV$, where IV denotes the initialization vector used in $\text{NMAC}^{E1,E2}$.
- For each $j = 2 \dots i$, it holds that $y_j = y_{j-1} \oplus z_{j-1}$.
- It is the case that $y_i \oplus z_i = x$, where x is the key provided in the current query $(2, +, x, y)$ (here we assume that $\kappa' = n$).

If the simulator S finds such a sequence of tuples, then it needs to send a response that is consistent with the random oracle F. Thus, it queries the random oracle F on the input $x_1 \parallel \dots \parallel x_\ell$ to get the output $Y = F(x_1$ *parallel* $\dots \parallel x_\ell)$. It then chooses its response as $z = Y \oplus y = Y \oplus IV'$ (since we know that $y = IV'$). It then sends this n'-bit string z as its response and store $(2, x, y, z)$ in its table $\mathcal{T}$. If S does not find such a tuple, then it sends a random response $z \in \{0,1\}^{n'}$ and stores $(2, x, y, z)$ in $\mathcal{T}$.

On getting an inverse query $(2, -, x, z)$, the simulator searches its table $\mathcal{T}$ for a tuple $(2, x, y, z)$ and responds with y if it finds such a tuple. If it does not find such a tuple, then it sends a uniformly random n'-bit response y and stores $(2, x, y, z)$ in its table $\mathcal{T}$.

Proof of Indifferentiability. We need to show that the distinguisher cannot tell apart the two scenarios, one where it has oracle access to the actual random oracle F and the simulator S described above, and the other where it has oracle access to RO construction $\text{NMAC}^{E1,E2}$ and the ideal block ciphers $E1$ and $E2$. We will use a hybrid argument to prove this result starting in the random oracle scenario, and ending in the ideal cipher scenario through a sequence of indistinguishable games.

GAME 1. This is the random oracle model, where the distinguisher D has oracle access to the random oracle F and the simulator S. Let G_1 denote the event that D outputs 1 after interacting with F and S. Thus,

$$\Pr[G_1] = \Pr\left[D^{F,S^F}(1^\lambda) = 1\right]$$

GAME 2. In this game, we give the distinguisher oracle access to a dummy *relay algorithm* R_0 instead of direct oracle access to the RO F. This relay algorithm, in turn, has oracle access to the RO F, and on getting a random oracle query from the distinguisher, it simply makes the same query to F and forwards the RO output to the distinguisher D as its response. The simulator S still has direct oracle access to F. Let G_2 denote the event that the distinguisher D outputs 1 in this game. Since the view of the distinguisher remains unchanged in this game, we can deduce that

$$\Pr[G_2] = \Pr\left[D^{R_0^F,S^F}(1^\lambda) = 1\right] = \Pr[G_1]$$

GAME 3. In this game, we will modify the simulator S by restricting its responses. In particular, the new simulator S_0 chooses its responses in the same fashion as the original simulator S, but after making its choice the simulator S_0 checks if it responses so far satisfy one of a few conditions that could aid the distinguisher in getting to know that it is in the random oracle scenario.

On a forward query $(1, +, x, y)$, the new simulator S_0 checks if there is a tuple $(1, x, y, z)$ in its table $\mathcal{T}$, and chooses its response z in the same way as the original simulator S. However, if the response chosen is a new one then it checks if the tuple (x, y, z) satisfies one of the following conditions before sending z.

1. *Condition B_1:* It is the case that $z \oplus y = IV$, where IV is the n-bit initialization vector used in the first Merkle-Damgård construction using $E1$.
2. *Condition B_2:* There is a tuple $(1, x', y', z') \in \mathcal{T}$, with $(x', y') \neq (x, y)$, such that $y' \oplus z' = y \oplus z$.
3. *Condition B_3:* There is a tuple $(1, x', y', z') \in \mathcal{T}$ such that $z \oplus y = y'$.

4. *Condition B_4:* There is a tuple $(2, x', y', z') \in \mathcal{T}$ such that $y \oplus z = x'$.

If the response z chosen by the simulator S_0 is such that at least one of these conditions is satisfied, then the simulator explicitly fails. Essentially, the idea is that conditions B_1 and B_2 could be used by the distinguisher to make two random oracle inputs collide after the Merkle-Damgård part using $E1$. On the other hand, conditions B_3 and B_4 could be used by the distinguisher to generate a random oracle input such that the simulator cannot adjust its output to match that of the random oracle. Since the simulator S_0 always chooses the response to any $E1$ query at random, we can bound the occurrence probabilities of each of these events using simple probability calculations.

On an inverse query $(1, -, x, z)$, the new simulator S_0 chooses its response y in the same fashion as the original simulator S. However, if the response is not chosen from the table $\mathcal{T}$, then S_0 checks if the tuple (x, y, z) satisfies any of the following conditions.

1. *Condition C_1:* It is the case that $y = IV$ or $y \oplus z = IV$, where IV is the initialization vector used in the Merkle-Damgård construction using $E1$.
2. *Condition C_2:* There is a tuple $(1, x', y', z') \in \mathcal{T}$, with $(x', z') \neq (x, z)$, such that $y' \oplus z' = y \oplus z$.
3. *Condition C_3:* There is a tuple $(1, x', y', z') \in \mathcal{T}$ such that $y \oplus z = y'$ or $y' \oplus z' = y$.
4. *Condition C_4:* There is a tuple $(2, x', y', z') \in \mathcal{T}$ such that $y \oplus z = x'$.

If the response y is such that at least one of these conditions is satisfied, then the simulator S_0 explicitly fails. We can estimate the occurrence probabilities for these failure condition similar to the case of a forward query $(1, +, x, y)$.

For queries made to the block cipher $E2$, we need to check for different failure conditions. In particular, the Merkle-Damgård construction using $E2$ will only be applied to one block inputs in the RO construction $\mathrm{NMAC}^{E1,E2}$. For forward queries $(2, +, x, y)$, the new simulator S_0 chooses $z \in \{0,1\}^{n'}$ in the same way as the original simulator S and sends z as its response without checking for any failure conditions. On the other hand, for inverse queries $(2, -, x, z)$, the simulator S_0 chooses $y \in \{0,1\}^{n'}$ similar to S, but then checks to see if the tuple (x, y, z) satisfies the following condition:

1. *Condition C'_1:* It is the case that $y = IV'$.

If the tuple (x, y, z) satisfies this condition and the response y was freshly chosen at random, then the simulator S_0 explicitly fails. The probability of occurrence of the failure condition C'_1 is a straightforward probability computation.

Let G_3 denote the event that the distinguisher D outputs 1 in game 3, i.e. $\Pr[G_3] = \Pr\left[D^{R_0^F, S_0^F}(1^\lambda) = 1\right]$. The response distribution of the distinguisher differs in games 2 and 3 if and only if the simulator S_0 fails in game 3. This event is identical to one of the failure conditions holding for the responses of the simulator S_0.

$$
\begin{aligned}
|\Pr[G_3] - \Pr[G_2]| &= \Pr[B_1 \vee B_2 \vee B_3 \vee B_4 \vee C_1 \vee C_2 \vee C_3 \vee C_4 \vee C'_1] \\
&\leq \frac{q^2}{2^{\min(n.n')}}
\end{aligned}
$$

GAME 4. In this game, we modify the relay algorithm, but leave the ideal cipher simulator S_0 unchanged. In particular, the new relay algorithm R_1 does not simply relay the outputs of the random oracle F. Instead, R_1 is given oracle access to the simulator S_0, and it responds to any random oracle queries made to it by honestly evaluating the RO construction $\mathrm{NMAC}^{E1,E2}$ by using the simulator S_0 in place of the ideal ciphers $E1$ and $E2$.

Let G_4 denote the event that the distinguisher D outputs 1 in game 4, so that

$$\Pr[G_4] = \Pr\left[D^{R_1^{S_0^F}, S_0^F}(1^\lambda) = 1\right]$$

We assume that the maximum length of a random oracle query made by the distinguisher is $\ell \cdot \kappa$. Now we will show that the view of the distinguisher D does not change by a non-negligible amount when we make this change to the relay algorithm. This is formally stated below.

Claim A.9 *Let G_3 and G_4 denote the events that the distinguisher outputs 1 in game 3 and 4, respectively. Let q_E and q_F denote the number of ideal cipher (including both $E1$ and $E2$ queries) and random oracle queries made by the distinguisher, then it is the case that*

$$|\Pr[G_4] - \Pr[G_3]| = \mathcal{O}\left(\frac{(q_E + q_F \cdot \ell)^2}{2^{\min(n,n')}}\right)$$

proof of claim A.9: The view of the distinguisher changes in the transition from game 3 to 4 only if there is a change in the response distributions of either the relay algorithm or the simulator between the two games. We will show that if the simulator S_0 does not fail in either of the two games, then such a change in the response distributions is impossible.

Let us start by analyzing the way the two relay algorithms, R_0 and R_1, choose their responses. The relay algorithm from game 3, R_0, simply forwards the random oracle output to any RO query X (i.e. responds with $F(X)$). On the other hand, the relay algorithm from game 4 uses the block ciphers simulated by S_0 to implement the RO construction $\mathrm{NMAC}^{E1,E2}$, and responds with the output of this "simulated construction". If the distinguisher detects a difference in the responses of the two relay algorithms, then it must be the case that the simulator S_0 did not adjust its responses consistently with the RO F in game 4, which resulted in the response of the relay algorithm R_1 not matching the RO output. We will show that unless the simulator S_0 explicitly fails, it is always able to adjust its responses consistent with the random oracle F.

The simulator S_0 is the same in both games 3 and 4. However, the simulator receives extra queries from the relay algorithm R_1 in game 4. Thus it may be the case that the simulator S_0 chooses its response to the same query differently, depending on whether it is in game 3 or game 4. This is the case only if the simulator chooses its response consistent with the RO F in one game, while independently at random in the other game. We will show that such a difference is impossible, unless the simulator S_0 explicitly fails in one of the games.

Below, for simplicity, we will denote by $\mathrm{NMAC}^{S_0}(X)$ the output of the "simulated RO construction" $\mathrm{NMAC}^{E1,E2}$ using the block ciphers simulated by S_0, while $F(X)$ is the actual random oracle output on X. We will start by proving a couple of useful properties of the responses of the simulator S_0 that hold in both games 3 and 4. The first property essentially says that if the simulator S_0 does not fail then it is not possible for the input to the Davies-Meyer function based on $E2$ to collide for two different RO inputs.

Claim A.10 *If the simulator S_0 does not explicitly fail, then there are no two different sequences of κ-bit blocks $x_1 \dots x_m$ and $x'_1 \dots x_p$ with corresponding tuples $(1, x_1, y_1, z_1) \dots (1, x_m, y_m, z_m)$ and $(1, x'_1, y'_1, z'_1) \dots (1, x'_p, y'_p, z'_p)$ in the table $\mathcal{T}$ of S_0 such that:*

- *It is the case that $y_1 = y'_1 = IV$. Moreover, for each $b = 2 \dots m$ and $b' = 2 \dots p$, it holds that $y_b = y_{b-1} \oplus z_{b-1}$ and $y'_{b'} = y'_{b'-1} \oplus z'_{b'-1}$.*
- *It is the case that $y_m \oplus z_m = y'_p \oplus z'_p$.*

proof of claim A.10: This is easy to see since there is a $r \in \{0 \dots (\min(m,p) - 1)\}$ such that,

$$\forall s \in \{0, (r+1)\} \ : \ (x_{m-s}, y_{m-s}, z_{m-s}) = (x'_{p-s}, y'_{p-s}, z'_{p-s}) \text{ and } (x_{m-r}, y_{m-r}, z_{m-r}) \neq (x'_{p-r}, y'_{p-r}, z'_{p-r})$$

Of the two tuples $(1, x_{m-r}, y_{m-r}, z_{m-r})$ and $(1, x'_{p-r}, y'_{p-r}, z'_{p-r})$, we consider the one whose corresponding query was made later. Without loss of generality, let this be $(1, x_{m-r}, y_{m-r}, z_{m-r})$. If this was a result of a forward query $(1, +, x_{m-r}, y_{m-r})$, then the simulator S_0 would have failed due to failure condition B_2. On the other hand if this were an inverse query, then S_0 would have failed as a result of the failure condition C_2. □

Next, we show that if the distinguisher wishes to find out the output $\text{NMAC}^{S_0}(X)$ for a random oracle query $X = x_1 \parallel \ldots \parallel x_m$, then the only way it can do so is by computing the RO construction honestly.

Claim A.11 *Consider any sequence of entries* $(1, x_1, y_1, z_1) \ldots (1, x_m, y_m, z_m), (2, x', y', z')$ *in the table* $\mathcal{T}$ *maintained by the simulator* S_0 *that satisfy the following properties:*

- *It is the case that* $y_1 = IV$ *and* $y' = IV'$.
- *For all* $i = 2 \ldots m$, *it is the case that* $y_i = y_{i-1} \oplus z_{i-1}$.
- *It also holds that* $x' = y_m \oplus z_m$.

If the simulator S_0 *does not explicitly fail, then it is necessarily the case that these entries were generated as a result of the ordered sequence of queries* $(1, +, x_1, y_1), \ldots, (1, +, x_m, y_m), (2, +, x', y')$.

proof of claim A.11: To the contrary, assume that the tuples $(1, x_1, y_1, z_1) \ldots (1, x_m, y_m, z_m), (2, x', y', z')$ were not generated as a result of the ordered sequence of forward queries $(1, +, x_1, y_1), \ldots, (1, +, x_m, y_m)$, $(2, +, x', y')$. In this case, one of the following must hold:

1. The tuple $(1, x_m, y_m, z_m)$ was stored in the table $\mathcal{T}$ after the tuple $(2, x', y', z')$, as a result of a forward/inverse query.
2. For some $j \in \{1 \ldots (m-1)\}$, a new forward query $(1, +, x_j, y_j)$ was made when the tuple $(1, x_{j+1}, y_{j+1}, z_{j+1})$ already existed in the table $\mathcal{T}$.
3. For some $j \in \{2 \ldots m\}$, a new inverse query $(1, -, x_j, z_j)$ was made when the tuple $(1, x_{j-1}, y_{j-1}, z_{j-1})$ already existed in the table $\mathcal{T}$.
4. The tuple $(1, x_1, y_1, z_1)$ was stored in $\mathcal{T}$ as a result of an inverse query $(1, -, x_1, z_1)$.
5. The tuple $(2, x', y', z')$ was stored in $\mathcal{T}$ as a result of the inverse query $(2, -, x', z')$.

We will show how any of these situations would have resulted in the simulator S_0 explicitly failing. In each of these cases, we can deduce that at least one of the failure conditions would have held.

- Case 1 : In this case, the failure condition B_4 (resp. C_4) would have been true for the query $(1, +, x_m, y_m)$ (resp. $(1, -, x_m, z_m)$).
- Case 2 : Failure condition B_3 would have been true for the query $(1, +, x_j, y_j)$.
- Case 3 : Failure condition C_3 would have been true for the query $(1, -, x_j, z_j)$.
- Case 4 : Failure condition C_1 would have been true for the query $(1, -, x_1, z_1)$.
- Case 5 : Failure condition C'_1 would have been true for the query $(2, -, x', z')$.

Thus if the simulator never fails, then the sequence of tuples $(1, x_1, y_1, z_1)\ldots(1, x_m, y_m, z_m), (2, x', y', z')$ could have been stored only as a result of the sequence of forward queries $(1, +, x_1, y_1), \ldots, (1, +, x_m, y_m), (2, +, x', y')$. □

As a consequence of claims A.10 and A.11, we can deduce that in both games 3 and 4 the simulator is always able to adjust its responses to be consistent with random oracle F if it does not explicitly fail. Thus the responses of the relay algorithm R_0 and R_1 are identical in the view of the distinguisher. Moreover, as a result of claim A.11, we can also deduce that the distinguisher D can only find the output $\mathrm{NMAC}^{S_0}(X)$ by making the sequence of forward queries given in claim A.11. In this case, the simulator adjusts its response accordingly so that $\mathrm{NMAC}^{S_0}(X) = F(X)$ for any X. Thus the view of the distinguisher D does not change in the transition between games 3 and 4 if the simulator S_0 does not explicitly fail in either game. Hence, we can deduce that

$$\begin{aligned} |\Pr[G_4] - \Pr[G_3]| &\leq \Pr[S_0 \text{ fails in either game}] \\ &= \mathcal{O}\left(\frac{(q_E + q_F \cdot \ell)^2}{2^{\min(n,n')}}\right) \end{aligned}$$

□

GAME 5. In this game, we modify the simulator so that it always selects its responses independent of the random oracle F. This does not induce any inconsistencies in the view of the distinguisher since the relay algorithm R_1 also uses the new simulator S_1 instead of directly using the random oracle F.

The new simulator S_1 always chooses a uniformly random response to any query made to it, including any forward query $(2, +, x, IV')$. Moreover, after it chooses a response it does not check for any of the failure conditions that the old simulator S_0 checked for in game 4. The view of the distinguisher does not change by more than a negligible amount in the transition from game 4 to 5. This is because the distinguisher only notices a difference between the two games if S_0 fails in game 4 (or equivalently, the new simulator S_1 responds with a z that satisfies one of the failure conditions checked by S_0). Since the new simulator S_1 always chooses a uniformly random response to any query, we can easily bound this difference.

$$\begin{aligned} |\Pr[G_5] - \Pr[G_4]| &\leq \Pr[S_0 \text{ fails in game 4}] + \Pr[S_1 \text{ satisfies one of the failure conditions}] \\ &= \mathcal{O}\left(\frac{(q \cdot \ell)^2}{2^{\min(n,n')}}\right) \end{aligned}$$

GAME 6. This is the final game of our proof. Here we replace the simulator S_1 by actual ideal block ciphers $E1 : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ and $E2 : \{0,1\}^{\kappa'} \times \{0,1\}^{n'} \to \{0,1\}^{n'}$. Since the relay algorithm R_1 essentially implements the RO construction $\mathrm{NMAC}^{E1,E2}$, the view of the distinguisher in this game is identical to its view in the ideal cipher model.

Let G_6 denote the event that the distinguisher D outputs 1 in this game. We can deduce that the view of the distinguisher does not change in the transition from game 5 to 6, unless the simulator S_1 outputs a collision in block cipher outputs for the same key. The probability of this event can be bounded by simply using the birthday paradox.

$$\begin{aligned} |\Pr[G_6] - \Pr[G_5]| &\leq \Pr[S_1 \text{ outputs a collision.}] \\ &= \mathcal{O}\left(\frac{(q \cdot \ell)^2}{2^{\min(n,n')}}\right) \end{aligned}$$

Now we can complete the proof of lemma A.8 by combining the above games. Hence, we deduce that

$$\left|\Pr\left[D^{\mathrm{NMAC}^{E1,E2},E1,E2}(1^\lambda)=1\right]-\Pr\left[D^{F,S}(1^\lambda)=1\right]\right|=\mathcal{O}\left(\frac{q^2\ell^2}{2^{\min(n,n')}}\right)$$

□

Lemma A.12 *The HMAC construction $HMAC^E$ using an ideal block cipher $E:\{0,1\}^\kappa\times\{0,1\}^n\rightarrow\{0,1\}^n$ is (t_D,t_S,q,ϵ)-indifferentiable from a random oracle $F:\{0,1\}^*\rightarrow\{0,1\}^n$ in the ideal block cipher model for E, for any t_D and $t_S=\mathcal{O}(q^2)$, with $\epsilon=2^{-n}\cdot\ell^2\cdot\mathcal{O}(q^2)$ ($\ell\kappa$ is the maximum length of an RO query made by the distinguisher).*

Proof: The proof of this lemma is almost identical to the proof of indifferentiability for the NMAC construction given in lemma A.8. This is because the HMAC construction essentially implements the NMAC using a single block cipher, by using different initialization vectors in each part of the construction. With slight modifications, the simulator described in lemma A.8 works in this case as well.

The proof of indifferentiability is also almost identical to that in lemma A.8. We do add a few extra "failure conditions" to handle the fact that we are using the same ideal cipher E in place of both E_1 and E_2. □

This completes the proofs of indifferentiability for each of our proposed constructions of random oracle in the ideal cipher model. □