On the Indifferentiability of the Sponge Construction

Guido Bertoni[1], Joan Daemen[1], Michaël Peeters[2], and Gilles Van Assche[1]
sponge@noekeon.org

[1] STMicroelectronics
[2] NXP Semiconductors

Abstract. In this paper we prove that the sponge construction introduced in [4] is indifferentiable from a random oracle when being used with a random transformation or a random permutation and discuss its implications. To our knowledge, this is the first time indifferentiability has been shown for a construction calling a random permutation (instead of an ideal compression function or ideal block cipher) and for a construction generating outputs of any length (instead of a fixed length).

1 Introduction

All cryptographic hash functions of any significance known today, i.e., MD4, MD5, the SHA and RIPEMD [15] families and several others, share the same design paradigm. They all consist of the iterated application of a compression function. The iteration mechanism is known as Merkle-Damgård [8, 16] and guarantees that if the compression function is collision-resistant, the resulting hash function is collision-resistant. This is a very attractive property as collision-resistance appears to be one of the most important properties of cryptographic hash functions. The compression functions of the above mentioned hash functions were designed with collision-resistance in mind. During the last years, with the recent collision attacks on SHA-1 as culminating point, it has become clear that designing a compression function that is both collision-resistant and efficient is not an easy task. Moreover, weaknesses have been shown in the Merkle-Damgård construction itself. While it does guarantee certain properties such as collision-resistance on the condition that the underlying compression function has the same property, this is not the case for all properties that are expected from cryptographic hash functions. A well known example of such a weakness, discussed in [7], is the insecurity of the MAC function constructed from a Merkle-Damgård hash function by feeding the latter with the secret key followed by the message.

In [7] Coron et al. propose a number of variants of the Merkle-Damgård construction that do not have this and other weaknesses. For each of these constructions they provide theorems stating that if the compression function is constructed using an ideal component, i.e., a finite input length (FIL) random oracle or an ideal block cipher (used as Davies-Meyer compression function), the hash function behaves as a random oracle [3] with output truncated to a fixed length. They present their theorems in the indifferentiability framework that was introduced by Maurer et al. in [14]. As a (truncated) random oracle has all desired properties that may be expected from a cryptographic hash function, this provides a direction for the design of hash functions that do not only provide resistance against collision search, but are as strong as a truncated random oracle with respect to many criteria. Instead of constructing an efficient compression function that is collision-resistant, one shall now design an efficient function that behaves as a FIL random oracle, or in other words, a random $n+m$ to n bit compression function, or an ideal block cipher. In the meanwhile, several other hash function constructions have been shown to be indifferentiable from a random oracle, see for example [2, 6]. Note that indifferentiability is not the only approach to proving properties of hash function constructions: some authors analyze the properties of the compression function that can be preserved by the construction [1, 2].

Recently, we introduced a new iterative hash function construction, called a sponge [4]. It builds upon a fixed-length transformation (i.e., with codomain equal to domain) or permutation f instead of a compression function and can generate output strings of infinite length. In [4] we proved that when f is a random transformation or permutation, the resulting function is only distinguishable from a random oracle with probability below $N(N+1)/2^{c+1}$, where N is the number of calls to f (and f^{-1}) and c is a security parameter related to the size of the domain of f. At first sight, one may consider the indistinguishability proof as an argument that it behaves as a random oracle with probability $1-N(N+1)/2^{c+1}$. However, this is restricted to adversaries that can only query the sponge function and not f (and f^{-1}). In a concrete hash function, f is publicly specified and this is of limited interest. We also included computations of the complexity of a number of so-called critical operations and discussed how this impacts the classical properties expected from hash functions such as collision-resistance and (2nd)-preimage resistance. However, this does not result in lower bounds for the security of these properties but rather upper bounds to the reachable security level. In this paper we apply the approach of [7] to the sponge construction and demonstrate that the advantage of an adversary in differentiating the sponge construction from a random oracle is about $N(N+1)/2^{c+1}$ if the underlying f is a random transformation and an even smaller upper bound if it is a random permutation.

As discussed above, this implies that the sponge construction when calling a random transformation or permutation has all properties of a random oracle as long as c is large enough. Hence we are now able to provide the security bounds for collision-resistance and (2nd)-preimage resistance that are lacking in [4].

There are several iterative hash function constructions that have been shown to be indifferentiable from a random oracle. However, the sponge construction has two unique features. First, it can generate long outputs. While other constructions can only behave as a random oracle whose output has been truncated to a fixed length, a random sponge does not have this limitation and may also serve as a reference for stream ciphers. Second, it can be built using a permutation, where both f and f^{-1} can be queried by the adversary. Paradoxically, collision-resistance and (2nd)-preimage resistance can be realized by employing a function that is easy to invert.

In [4] our main goal was to define a reference for security properties of hash designs. With our indifferentiability result, we prove that the resistance of the sponge construction calling a random transformation or permutation is as good as that of a random oracle, lower bounded by about $N(N+1)/2^{c+1}$. This coincides with what is presented in [4] as the *flat sponge claim*. Despite our original intention in [4], we argue that the sponge construction can lead to practical hash function designs. First of all, as mentioned in [4], the support for long outputs is a useful feature for a hash function when being used as a mask generating function (MGF) or a key derivation function (KDF). Second, instead of a collision-resistant compression function (Merkle-Damgård) or a random-looking compression function or ideal block cipher (as in [7]), it takes the design of a random-looking permutation. As a good block cipher should behave as a set of (independent and) random-looking permutations, hash function design can now benefit from insights gained in block cipher design. However, as opposed to a block cipher, a permutation has no key schedule and has not the concerns that come with it such as its computational overhead and possible related-key weaknesses. This makes in our opinion the sponge construction a very interesting alternative to the constructions based on a compression function.

The remainder of this paper is organized as follows. Section 2 gives a short introduction to indifferentiability applied to hash function constructions and is followed by Section 3 that defines and discusses the sponge construction in the indifferentiability setting. Section 4 gives the actual proofs and finally Section 5 discusses its implications.

2 Indifferentiability from a random oracle

Indifferentiability deals with the interaction between systems where the objective is to show that two systems cannot be told apart by an adversary able to query both systems but not knowing a priori which system is which. For hash function constructions, a random oracle serves as an ideal system.

We use the definition of random oracle from [3]. A random oracle, denoted $\mathcal{RO}$, takes as input binary strings of any length and returns for each input a random infinite string, i.e., it is a map from $\mathbf{Z}_2^*$ to $\mathbf{Z}_2^\infty$, chosen by selecting each bit of $\mathcal{RO}(x)$ uniformly and independently, for every x. In [7] and other papers on the subject, one does not consider indifferentiability from a random oracle, but rather a random oracle with output truncated to a fixed number of bits.

The indifferentiability framework was introduced by Maurer et al. in [14] and is an extension of the classical notion of indistinguishability. Coron et al. applied it to iterated hash function constructions in [7] and demonstrated for a number of iterated hash function constructions that they are indifferentiable from a random oracle if the compression function is a random FIL oracle. In

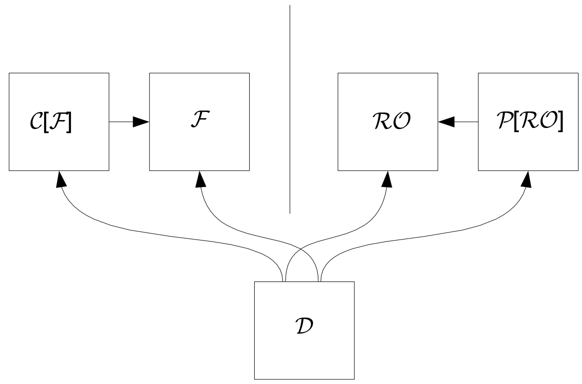

Fig. 1. The differentiability setup

this section we give a brief introduction to these subjects; for a more in-depth treatment, we refer to the original papers.

In the context of iterated hashing the adversary shall distinguish between two systems that each have two components, as illustrated in Figure 1. The system at the left is the combination of the ideal compression function $\mathcal{F}$ and the hash function construction $\mathcal{C}$. The adversary can make queries to both components separately, where the latter in turn calls the former to construct its responses. This is denoted by $\mathcal{C}[\mathcal{F}]$. These are the two different interfaces to the system to the left.

The system at the right consists of a random oracle (with truncated output) $\mathcal{RO}$ providing the same interface as $\mathcal{C}[\mathcal{F}]$. To be indifferentiable from the system at the left, the system at the right also needs a subsystem offering the same interface to the adversary as the ideal compression function $\mathcal{F}$. This is called a *simulator* $\mathcal{P}$ and its role is to simulate the ideal compression function $\mathcal{F}$ so that no distinguisher can tell whether it is interacting with the system at the left or with the one at the right. The output of $\mathcal{P}$ should look *consistent* with what the distinguisher can obtain from the random oracle $\mathcal{RO}$ as if $\mathcal{P}$ was $\mathcal{F}$ and $\mathcal{RO}$ was $\mathcal{C}[\mathcal{F}]$. To achieve that, the simulator can query the random oracle, denoted by $\mathcal{P}[\mathcal{RO}]$. Note that the simulator does not see the distinguisher's queries to the random oracle.

Indifferentiability of $\mathcal{C}[\mathcal{F}]$ from a random oracle $\mathcal{RO}$ is now satisfied if there exists a simulator $\mathcal{P}$ such that no distinguisher can tell the two systems apart with non-negligible probability, based on their responses to queries it may send. We repeat here the definition as given in [7] where the hash function construction is called Turing machine $\mathcal{C}$, the ideal compression function is called ideal primitive $\mathcal{F}$ and the random oracle is called ideal primitive $\mathcal{G}$.

Definition 1 ([7]). *A Turing machine $\mathcal{C}$ with oracle access to an ideal primitive $\mathcal{F}$ is said to be (t_D, t_S, q, ϵ) indifferentiable from an ideal primitive $\mathcal{G}$ if there*

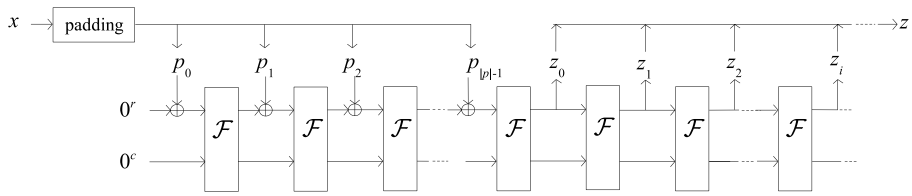

Fig. 2. The padded sponge construction

exists a simulator $\mathcal{P}[\mathcal{G}]$*, such that for any distinguisher* $\mathcal{D}$ *it holds that:*

$$|\Pr[\mathcal{D}[\mathcal{C}[\mathcal{F}], \mathcal{F}] = 1] - \Pr[\mathcal{D}[\mathcal{G}, \mathcal{P}[\mathcal{G}]] = 1]| < \epsilon. \tag{1}$$

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

Now, it is shown in [14] that if $\mathcal{C}[\mathcal{F}]$ is indifferentiable from a random oracle, then $\mathcal{C}[\mathcal{F}]$ can replace the random oracle in any cryptosystem, and the resulting cryptosystem is at least as secure in the ideal compression function model as in the random oracle model. This is much stronger than the indistinguishability of $\mathcal{C}[\mathcal{F}]$ from a random oracle, which just merely means that an attacker that can query $\mathcal{C}[\mathcal{F}]$, but has no direct access to $\mathcal{F}$, cannot distinguish it from a random oracle. As said, for hash function constructions indistinguishability makes little sense as, for any concrete hash function, the compression function $\mathcal{F}$ is public and hence accessible to the adversary.

3 The sponge construction

3.1 Definition

In this section we define the sponge construction. Our definition is a special case of the more general definition in [4]. To simplify the presentation, we restrict the input and output of the sponge to binary strings instead of a more general alphabet. Our indifferentiability result can however easily be extended to the generic definition. The (padded) sponge construction is illustrated in Figure 2.

In the sequel, we generally denote by x a message in $\mathbf{Z}_2^*$, and by p a sequence of blocks of r bits each (i.e., $p \in \mathbf{Z}_2^{r*}$), indexed from 0 to $|p| - 1$, with $|p|$ the number of r-bit blocks of p.

Definition 2. *For positive integers* r, c*, a* sponge function $\mathcal{S}[\mathcal{F}]$ *maps binary strings with length a multiple of* r *to binary strings of any requested length, i.e.,* $\mathbf{Z}_2^{r*}$ *to* $\mathbf{Z}_2^{\infty}$*. A sponge calls a transformation* $\mathcal{F}$ *operating on* $\mathbf{Z}_2^{r+c} = \mathbf{Z}_2^r \times \mathbf{Z}_2^c$ *as described in Algorithm 1. Here* c *is called the* capacity *and* r *is called the* bitrate[3] *of the sponge. The (group) operation* $\oplus$ *denotes the bitwise addition of* r*-bit blocks and* 0^r *is the all-zero block, the neutral element of this group. The input* p *to a sponge function must consist of one or more blocks and shall not have* 0^r *as last block, i.e.,* $|p| > 0$ *and* $p_{|p|-1} \neq 0^r$*.*

Algorithm 1 The sponge construction $\mathcal{S}[\mathcal{F}]$

  Input $p = p_0 p_1 \ldots p_{|p|-1}$ and requested length n
Require: $|p| \geq 1$ and $p_{|p|-1} \neq 0^r$
  Output $z \in \mathbf{Z}_2^n$
  $s = (s_\mathrm{a}, s_\mathrm{c}) = (0^r, 0^c)$
  for $i = 0$ to $|p| - 1$ **do**
    $(s_\mathrm{a}, s_\mathrm{c}) = \mathcal{F}(s_\mathrm{a} \oplus p_i, s_\mathrm{c})$
  end for
  for $i = 0$ to $\lceil \frac{n}{r} \rceil - 1$ **do**
    Append s_a to the output
    $(s_\mathrm{a}, s_\mathrm{c}) = \mathcal{F}(s_\mathrm{a}, s_\mathrm{c})$
  end for
  Discard the last $r\lceil \frac{n}{r} \rceil - n$ bits

3.2 Graph representation of sponge operation

In [4] we used a graph representation to prove bounds on success probability of generating collisions. We adopt this graph representation in the specification of our simulators. In our discussions on the graphs, we need to clearly distinguish between the first r bits and the last c bits of an $r+c$-bit variable s. For this, we again use the notation of [4]: $A = \mathbf{Z}_2^r$ and $C = \mathbf{Z}_2^c$ and we call the first r bits of s its A-part s_a, and the last c bits its C-part s_c.

We consider the transformation $\mathcal{F}$ as a directed graph whose vertex set (called *nodes*) is $A \times C$ and whose edges are $(s, \mathcal{F}(s))$. It has both 2^{r+c} nodes and edges. From the node graph, we derive the (directed) supernode graph, with vertex set (called *supernodes*) equal to C. In this graph, an edge $(s_\mathrm{c}, t_\mathrm{c})$ is in the edge set if and only if $\exists s_\mathrm{a}, t_\mathrm{a}$ such that $((s_\mathrm{a}, s_\mathrm{c}), (t_\mathrm{a}, t_\mathrm{c}))$ is an edge in the node graph. The set of supernodes is a partition of the nodes where a supernode contains the 2^r nodes with the same C-part.

The sponge construction operates on a chaining variable s and its operation can be seen as a walk through the node graph of the chaining variable. We denote the chaining variable before processing p_i by s_i. Its initial value is $s_0 = (0^r, 0^c)$. Then for each block p_i, it performs a two-step transition. First, it moves to the node s' within the same supernode with $s'_\mathrm{a} = s_{i,\mathrm{a}} \oplus p_i$, and then it follows the edge starting from s', arriving in s_{i+1}. After processing all blocks of p it is in

[3] The bitrate r is not to be confused with rate meaning the number of block cipher calls it takes to implement the compression function, as in, e.g., [12].

node $s_{|p|}$. Then it gives out the A-part of $s_{|p|}$ as z_0. For each additional block z_i produced, it follows the edge from $s_{|p|+i-1}$ arriving in $s_{|p|+i}$ and gives out the A-part of the latter as z_i. Note that this can be considered a special case of the above two-step transition if we extend p with blocks $p_{|p|+i} = 0^r$ for all $i \geq 0$. Clearly, the chaining variable s_i is completely determined by the first i blocks of p. We call this a *path* to s_i. Or more exactly:

Definition 3 ([4]). *First, the empty string is a path to the node* $(0^r, 0^c)$. *Then, if* p *is a path to node* $s = (s_a, s_c)$ *and there is an edge* $((s_a \oplus a, s_c), t)$ *in the node graph,* $p' = pa$ *is a path to node* t.

Note that although a path completely determines a node, there may be many paths to a node.

It follows from the above that z_j of $z = \mathcal{S}[\mathcal{F}](p)$ is the A-part of the node with path $p0^{rj}$. And so, given a path p (different from 0^{rj}) to a node s, one can find its A-part by a call to the sponge construction. We have $s_{\mathrm{a}} = z_j$ with $z = \mathcal{S}[\mathcal{F}](p')$ and p' and j given by $p = p'0^{rj}$ such that p' is a valid sponge input, i.e., $|p'| > 0$ and $p'_{|p'|-1} \neq 0^r$. For a path of form 0^{rj} there is no such p' and hence the sponge construction cannot be queried to obtain s_{a}.

3.3 The padded sponge construction

The sponge construction $\mathcal{S}[\mathcal{F}](p)$ only supports input strings $p \in \mathbf{Z}_2^{r*}$ where p is not the empty string and has last block different from 0^r. To allow the input to be any binary string in $\mathbf{Z}_2^*$, one needs to define an injective mapping $\mathrm{pad}(x)$ that converts any binary string x to a valid sponge input. The simplest such mapping $\mathrm{pad}(x)$ consists in padding the string with a single bit 1 and a number w of zeroes with $0 \leq w < r$ so that $\mathrm{pad}(x)$ contains a multiple of r bits. To indicate the sponge construction including the padding operation, we use the symbol $\mathcal{S}'$:

$$\mathcal{S}'[\mathcal{F}](x) \triangleq \mathcal{S}[\mathcal{F}](\mathrm{pad}(x)).$$

3.4 The distinguisher's setting

We give proofs of indifferentiability for the cases that $\mathcal{F}$ is a random transformation or a random permutation. A random transformation (permutation) operating on a certain domain is a transformation selected randomly and uniformly from all transformations (permutations) operating on that domain.

The adversary shall distinguish between two systems using their responses to sequences of queries. At the left is the system $(\mathcal{S}'[\mathcal{F}], \mathcal{F})$. The padded sponge construction $\mathcal{S}'[\mathcal{F}]$ provides one interface denoted by $\mathcal{H}$, taking a binary string $x \in \mathbf{Z}_2^*$ and an integer n and returning a binary string $y \in \mathbf{Z}_2^n$, the sponge output truncated to n bits. If $\mathcal{F}$ is a random transformation it has a single interface $\mathcal{F}^1$ which takes as input an element s of $\mathbf{Z}_2^{r+c}$ and returns $t = \mathcal{F}(s)$, an element of the same set. If $\mathcal{F}$ is a random permutation, it has an additional interface $\mathcal{F}^{-1}$ that implements the inverse of $\mathcal{F}$. Note that the sponge construction in Algorithm 1 only uses the interface $\mathcal{F}^1$.

At the right is the system $(\mathcal{RO}, \mathcal{P}[\mathcal{RO}])$. It offers the same interface as the left system, i.e., $\mathcal{RO}$ provides the interface $\mathcal{H}$ and returns an output truncated to the requested length. We define two simulators, one for the case of a random transformation and another one for the case of a random permutation. The transformation simulator provides a single interface $\mathcal{F}^1$. The permutation simulator provides both interfaces $\mathcal{F}^1$ and $\mathcal{F}^{-1}$.

Let $\mathcal{X}$ be either $(\mathcal{S}'[\mathcal{F}], \mathcal{F})$ or $(\mathcal{RO}, \mathcal{P}[\mathcal{RO}])$. The sequence of queries Q to $\mathcal{X}$ consist of a sequence of queries to the interface $\mathcal{H}$, denoted Q^0 and a sequence of queries to the interface $\mathcal{F}^1$ (and $\mathcal{F}^{-1}$), denoted Q^1. Q^0 is a sequence of couples (x, n), with $x \in \mathbf{Z}_2^*$ and n a positive integer. Q^1 is a sequence of couples (s, b) with $s \in \mathbf{Z}_2^{r+c}$ and b either 1 or -1, indicating whether the interface $\mathcal{F}^1$ or $\mathcal{F}^{-1}$ is addressed. In the case that $\mathcal{F}$ is a transformation, b is restricted to 1.

3.5 The cost of queries

Definition 1 suggests expressing an upper bound to the advantage of a distinguisher in terms of the number of queries q. The bounds provided in [7] however also make use of parameter ℓ, the maximum input length of the queries. In our bounds we use another measure for the query complexity which is more natural when applied to the sponge construction. We call this measure *cost* and denote it by N. The cost N of queries to a system $\mathcal{X}$ is the total number of calls to $\mathcal{F}$ or $\mathcal{F}^{-1}$ it would yield if $\mathcal{X} = (\mathcal{F}, \mathcal{S}'[\mathcal{F}])$, either directly due to queries Q^1, or indirectly via queries Q^0 to $\mathcal{S}'[\mathcal{F}]$. The cost of a sequence of queries is fully determined by their number and their input and output lengths. Each query to $\mathcal{F}^1$ or $\mathcal{F}^{-1}$ contributes 1 to the cost. A query to $\mathcal{H}$ with an ℓ-bit input contributes $\lfloor \frac{\ell}{r} \rfloor + \lceil \frac{n}{r} \rceil$ to the cost (assuming the simple padding of Section 3.3 is used). Our bounds in terms of cost are comparable to those of [7]: for a fixed output size, as considered in [7], N is an affine function of q and $q\ell$.

In the sequel, we consider the indifferentiability as in Definition 1 but with the cost N replacing the number of queries q and their maximum length ℓ.

4 Indifferentiability proofs

4.1 The simulators we use in our proofs

We define simulators for the case that $\mathcal{F}$ is a random transformation and for the case of a random permutation. In both cases, the simulator should behave as a deterministic function and give responses to queries Q^1 that in combination with the responses to queries Q^0 to the random oracle shall minimize the probability that the system $(\mathcal{RO}, \mathcal{P}[RO])$ can be distinguished from a system $(\mathcal{S}'[\mathcal{F}], \mathcal{F})$. In this section we informally explain how our simulators work.

A simulator keeps track of the queries it received and the responses it returned in a graph, very similar to the graphs discussed in Section 3.2. The only difference is that initially the simulator graph has no edges and for each new query $\mathcal{F}^1(s)$ (or $\mathcal{F}^{-1}(s)$) the simulator generates a response t and adds the edge (s, t) (or

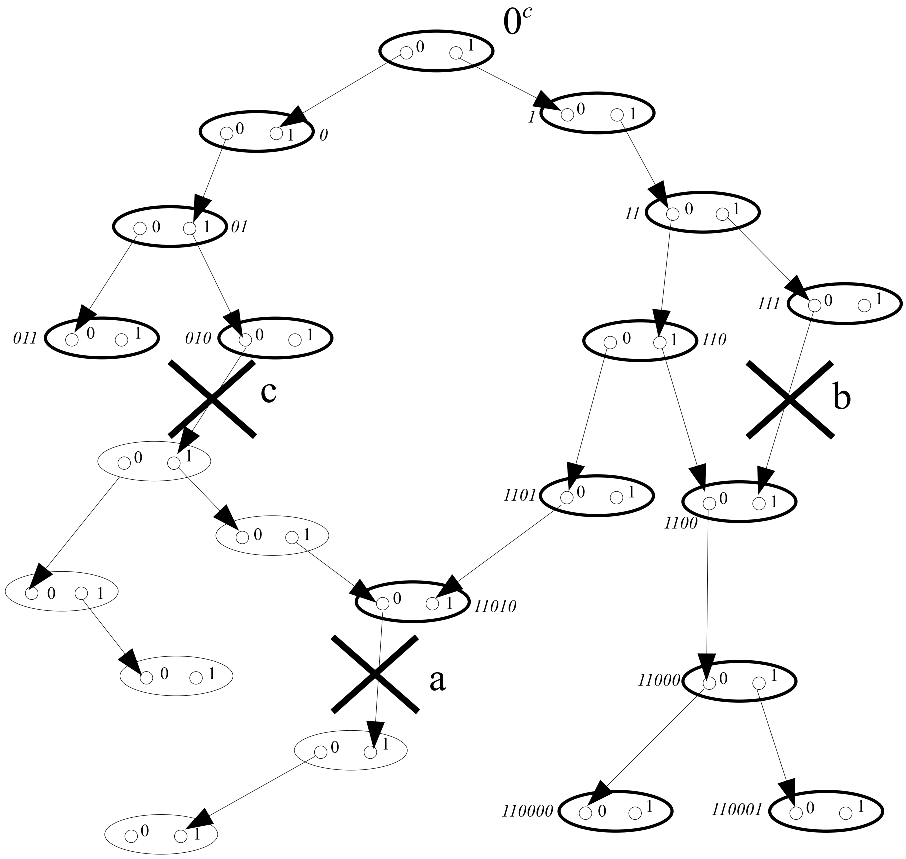

Fig. 3. Example of simulator graph. The rooted supernodes are in bold. Paths are indicated in italic next to the nodes having a path.

(t, s)). Note that using the responses of the simulator to its queries, the adversary can fully reconstruct the simulator graph.

In order to motivate the design of the simulators, we now discuss properties of this graph that it has at any moment during or after the queries, using an example depicted in Figure 3.

For a subset of the nodes in the simulator graph, the adversary knows a path. From Definition 3, it is clear that these are the nodes that have an incoming edge and are in a supernode that can be reached from supernode 0^c by following the directed edges from supernode to supernode. For this purpose, we define the set of *rooted* supernodes R as the subset of C containing 0^c and all the supernodes accessible from it through the supernode graph. By extension, we say that a node $s = (s_\mathrm{a}, s_\mathrm{c})$ is rooted if $s_\mathrm{c} \in R$. So the adversary knows paths to all rooted nodes that have an incoming edge from another rooted node, plus the empty path of the $(0^r, 0^c)$ node. For each of these rooted nodes it can query the interface $\mathcal{H}$ of the system hoping to reveal an inconsistency, which is evidence that it is not $(\mathcal{S}'[\mathcal{F}], \mathcal{F})$. We call *sponge-consistent* the responses to a sequence of queries Q that do not result in such inconsistency.

We will now explain why our simulators generate sponge-consistent responses (up to 2^c queries Q^1). Whenever a simulator receives a query $\mathcal{F}^1(s)$ with s rooted, it will result in an image t with known path. Therefore, the simulator constructs the A-part of t to be sponge-consistent by querying $\mathcal{RO}$ using the path to t (except for the all-zero path). When the simulator receives a query $\mathcal{F}^1(s)$ with s not rooted, no path to the image t is known and it chooses t randomly from all the nodes (with no incoming edge, if $\mathcal{F}$ is a random permutation).

The idea is that the simulators are designed so that a call to $\mathcal{F}^1(s)$ results only in the path of a single node becoming known, that of $t = \mathcal{F}(s)$ if s is rooted. For that, when selecting t_c for a rooted node s, they exclude the supernodes with outgoing edges (cases a and c in Figure 3). Additionally, they avoid the occurrence of nodes with multiple paths. For that, when selecting t_c for a rooted node s, they exclude the rooted supernodes (case b in Figure 3) and those with outgoing edges (case c in Figure 3). The permutation simulator avoids paths of nodes becoming known as a result of a call to $\mathcal{F}^{-1}(s)$ altogether by excluding rooted supernodes when selecting t_c.

Let O be the set of supernodes with an outgoing edge. When the simulator receives a query $\mathcal{F}^1(s)$ with s a rooted node and all supernodes are rooted or have an outgoing edge, i.e., if $R \cup O = C$, it can no longer ensure sponge-consistency and we call the simulator *saturated*. As every query to the simulator adds at most one edge and that hence $R \cup O$ can be extended by at most 1 per query, this cannot happen before 2^c queries.

4.2 When being used with a random transformation

The simulator for the case that $\mathcal{F}$ is a random transformation is given in Algorithm 2. We prove the indifferentiability by means of a series of lemmas and a final theorem.

Lemma 1. *To every node in the simulator graph there is at most one path, unless the simulator is saturated.*

Proof. First, we show that the rooted supernodes in the supernode graph form a tree. When no edges exist, this is indeed the case. The only way to create a new rooted node is by calling $\mathcal{F}^1(s)$ with s rooted. Assuming the simulator is not saturated, this happens only in first part of Algorithm 2 (lines 4–12), if s is rooted and has no outgoing edge. The new edge only adds a single supernode to R as the simulator selects it from the supernodes with no outgoing edges. Moreover, the new edge cannot arrive in a rooted supernode (because the simulator selects t_c from $C \setminus R$) or in a supernode from which a rooted supernode can be reached (because the simulator select t_c from the supernodes with no outgoing edges).

Algorithm 2 The transformation simulator $\mathcal{P}[\mathcal{RO}]$

Interface $\mathcal{F}^1$, taking node s as input
if node s has no outgoing edge **then**
  if node s is rooted AND $R \cup O \neq C$ (no saturation) **then**
    Construct path to t: find path to s, append s_a and call the result p
    Write p as $p = p'0^{rj}$ where p' does not end with 0^r
    if p' can be unpadded into x **then**
      Assign to t_a the value z_j with $z = \mathcal{RO}(x)$
    else
      Choose t_a randomly and uniformly
    end if
    Choose t_c randomly and uniformly from $C \setminus (R \cup O)$
    Let $t = (t_\mathrm{a}, t_\mathrm{c})$
  else
    Choose t randomly and uniformly from all nodes
  end if
  Add an edge from s to t
end if
return the node t at the end of the outgoing edge from s

Then, for two connected supernodes $(s_\mathrm{c}, t_\mathrm{c})$, there exists only one edge in the simulator graph of the form $((s_\mathrm{a}, s_\mathrm{c}), (t_\mathrm{a}, t_\mathrm{c}))$. This is because the simulator chooses a distinct C-part for each new rooted node (unless it is saturated).

Finally, since A is a group, each r-bit block of the path is uniquely determined by the transitions on the A-part of the nodes. □

For a given set of queries Q and their responses $\mathcal{X}(Q)$, we define the *sponge consistency* as the property that the responses to Q^0 are equal to those that one would obtain by applying the sponge construction from the responses to Q^1 (when the queries Q^1 suffice to perform this calculation), i.e., that $\mathcal{X}(Q^0) = \mathcal{S}'[\mathcal{X}(Q^1)](Q^0)$. By construction, the queries, and their responses, made to the system $(\mathcal{S}'[\mathcal{F}], \mathcal{F})$ are sponge-consistent. For the sponge-consistency of the queries, and their responses, made to $(\mathcal{RO}, \mathcal{P}[\mathcal{RO}])$, we refer to the following lemma.

Lemma 2. *Given queries to the simulator $\mathcal{P}[\mathcal{RO}]$ described in Algorithm 2 and to $\mathcal{RO}$, it returns sponge-consistent responses, unless the simulator is saturated.*

Proof. The adversary can check by querying the random oracle for sponge-consistency for every node s in the simulator graph to which it knows the path p. The all-zero path does not correspond to a block that can be output by the sponge construction, so without loss of generality we assume that $p \neq 0^{rj}$.

Given the path p to the node s, its A-part must be equal to z_j with $z = \mathcal{RO}(x)$, where $\mathrm{pad}(x) = p'$ and p' is a valid sponge input given by $p = p'0^{rj}$. As Lemma 1 says, there is only a single path to any rooted node in the simulator graph, and thus the simulator guarantees this equality for the response t to every query to $\mathcal{F}^1(s)$ with s a rooted node, as long as it is not saturated.

We also need to show that no path is assigned to a node unless its A-part is chosen by the lines 6–9 of Algorithm 2. Indeed, the supernode t_c (at line 11) is the only supernode that becomes rooted due to the query. This is because the simulator excludes supernodes with outgoing edges in the selection of t_c (as long as the simulator is not saturated).

It follows that the simulator guarantees sponge-consistency for all queries Q up to saturation. □

Lemma 3. *Any sequence of queries Q^0 up to cost 2^c can be converted to a sequence of queries Q^1 where Q^1 gives at least the same amount of information to the adversary and has no higher cost than Q^0.*

Proof. A query in Q^0 consists of an input x and a length n. Let $p = \mathrm{pad}(x)0^{r\lceil \frac{n}{r} \rceil}$ and we can now convert this query into $|p|$ queries to $\mathcal{F}^1$. Let $s_0 = (0^r, 0^c)$ and $s_{i+1} = \mathcal{F}^1(s_{i,\mathrm{a}} \oplus p_i, s_{i,\mathrm{a}})$ for $0 \leq i < |p|$ be the responses to the new queries. As Lemma 2 says that all queries up to cost 2^c are sponge-consistent, the output to the original (x, n) query consists of the concatenation of the A-parts of $s_{|p|}$ to $s_{|p|+\lceil \frac{n}{r} \rceil - 1}$ truncated to n bits. By the definition of the cost of queries, the original query in Q^0 has cost $|p|$ and it results in $|p|$ queries in Q^1, each one with cost 1.

This process can be repeated for all queries in Q^0 resulting in a sequence of queries Q^1 with the same cost. If there are queries in Q^0 with inputs having common prefixes, these can give rise to the same queries in Q^1 resulting in a reduction in cost. □

Lemma 4. *The advantage of an adversary in distinguishing between $\mathcal{F}$ and $\mathcal{P}[\mathcal{RO}]$ with the responses to a sequence of $N < 2^c$ queries Q^1 is upper bounded by:*

$$f_{\mathrm{T}}(N) = 1 - \prod_{i=1}^{N} \left(1 - \frac{i}{2^c}\right) .$$

Proof. The advantage is defined as

$$\mathrm{Adv}(\mathcal{A}) = |\Pr[\mathcal{A}[\mathcal{F}] = 1] - \Pr[\mathcal{A}[\mathcal{P}[\mathcal{RO}]] = 1]|$$

The response sequence x to a sequence of N different queries is a sequence of N values in $A \times C$. We can provide an upper bound of the advantage by computing the probability distributions of the outcomes of the queries to $\mathcal{F}$ on the one hand and to $\mathcal{P}[\mathcal{RO}]$ on the other. The optimal adversary gives back 1 for the response sequence x if $\Pr(x|\mathcal{F}) > \Pr(x|\mathcal{P}[\mathcal{RO}])$ and 0 otherwise, yielding the following upper bound:

$$\mathrm{Adv}(\mathcal{A}) \leq \frac{1}{2} \sum_x |\Pr(x|\mathcal{F}) - \Pr(x|\mathcal{P}[\mathcal{RO}])|, \tag{2}$$

where the righthand side of this equation is known as the variational distance. Since $\mathcal{F}$ is a transformation over $A \times C$ chosen randomly and uniformly, the responses to the different queries are independent and uniformally distributed over $A \times C$. It follows that all $(2^r 2^c)^N$ possible outcomes are all equiprobable.

By inspecting Algorithm 2, the simulator always returns uniform values for the A-part of the image. For the C-part, the simulator chooses it non-uniformly only if the pre-image s is rooted. To obtain the greatest possible variational distance, the optimum strategy consists in creating N rooted nodes. As a response to the first query, it may return all values but 0^r. At each subsequent query, one value of C is added to R, and thus for each query, the simulator returns a C-part value different from 0^r and all previous ones. Note that by restricting $N < 2^c$ the simulator will not be saturated. Using this strategy gives us an upper bound on the variational distance. So for the simulator, there are $(2^r)^N(2^c-1)_{(N)}$ (where $a_{(n)}$ denotes $a!/(a-n)!$) possible responses with different C-parts, each with equal probability $((2^r)^N(2^c-1)_{(N)})^{-1}$, and the $(2^r)^N((2^c)^N-(2^c-1)_{(N)})$ others have probability 0. This gives:

$$\mathrm{Adv}(\mathcal{A}) \leq 1 - \frac{(2^c-1)_{(N)}}{(2^c)^N} = 1 - \prod_{i=1}^{N}\left(1-\frac{i}{2^c}\right) . \tag{3}$$

□

We have now all ingredients to prove the following theorem.

Theorem 1. *A padded sponge construction calling a random transformation, $\mathcal{S}'[\mathcal{F}]$, is (t_D, t_S, N, ϵ)-indistinguishable from a random oracle, for any t_D, $t_S = O(N^2)$, $N < 2^c$ and any ϵ with $\epsilon > f_{\mathrm{T}}(N)$.*

Proof. As discussed in Lemma 3 we can construct from a set of query sequences Q^0, Q^1 an equivalent sequence of queries $Q^{1\prime} \circ Q^1$ with no higher cost and giving at least the same information. So, without loss of generality, we only need to consider adversaries using queries $\overline{Q}^1 = Q^{1\prime} \circ Q^1$ and their response $\mathcal{X}(\overline{Q}^1)$ and no queries Q^0.

For any fixed query $\overline{Q}^1$, we look at the problem of distinguishing the random variable $\mathcal{F}(\overline{Q}^1)$ from the random variable $\mathcal{P}[\mathcal{RO}](\overline{Q}^1)$. For a sequence of queries $\overline{Q}^1$ with cost N, Lemma 4 upper bounds the advantage of such an adversary to $f_{\mathrm{T}}(N)$.

We have $t_S = O(N^2)$ as for each query to the simulator with s rooted, it must find the path to s and send a query to the random oracle of cost equal to the length of the path to s. The length of the path to s it upper bounded by N, the total number of rooted supernodes in the simulator graph. □

If N is significantly smaller than 2^c, we can use the approximation $1-x \approx e^{-x}$ for $x \ll 1$ to simplify the expression for $f_{\mathrm{T}}(N)$:

$$f_{\mathrm{T}}(N) \approx 1 - e^{-\frac{N(N+1)}{2^{c+1}}} < \frac{N(N+1)}{2^{c+1}}. \tag{4}$$

4.3 When being used with a random permutation

The simulator for the case that $\mathcal{F}$ is a random permutation is given in Algorithm 3. We now can prove indifferentiability using a series of similar lemmas.

Algorithm 3 The permutation simulator $\mathcal{P}[\mathcal{RO}]$

Interface $\mathcal{F}^1$, taking node s as input
if node s has no outgoing edge **then**
  if node s is rooted AND $R \cup O \neq C$ (no saturation) **then**
    Construct path to t: find path to s, append s_a and call the result p
    Write p as $p = p'0^{rj}$ where p' does not end with 0^r
    if p' can be unpadded into x **then**
      Assign to t_a the value z_j with $z = \mathcal{RO}(x)$
    else
      Choose t_a randomly and uniformly
    end if
    Choose t_c randomly and uniformly from $C \setminus (R \cup O)$ and such that $(t_\mathrm{a}, t_\mathrm{c})$ has no incoming edge yet
    Let $t = (t_\mathrm{a}, t_\mathrm{c})$
  else
    Choose t randomly and uniformly from all nodes that have no incoming edge yet
  end if
  Add an edge from s to t
end if
return the node t at the end of the outgoing edge from s

Interface $\mathcal{F}^{-1}$, taking node s as input
if node s has no incoming edge **then**
  Choose t_a randomly and uniformly
  Choose t_c randomly and uniformly from $C \setminus R$ and such that $(t_\mathrm{a}, t_\mathrm{c})$ has no outgoing edge yet
  Let $t = (t_\mathrm{a}, t_\mathrm{c})$
  Add an edge from t to s
end if
return the node t at the beginning of the incoming edge into s

The proofs of Lemma 1 and Lemma 2 are valid for the permutation simulator with respect to all calls to $\mathcal{F}^1$ but do naturally not consider calls to $\mathcal{F}^{-1}$. The proofs can simply be extended to the permutation simulator case by noting that the $\mathcal{F}^{-1}$ interface of the simulator excludes rooted nodes in the selection of the response, implying that a call to $\mathcal{F}^{-1}$ cannot lead to new rooted nodes and hence also not to new paths. The proof of Lemma 3 is valid for the permutation simulator as it is. Finally, the output produced by the interfaces $\mathcal{F}^1$ and $\mathcal{F}^{-1}$ are consistent, i.e., if $\mathcal{F}^1(s) = t$ then $\mathcal{F}^{-1}(t) = s$ and vice-versa.

Instead of Lemma 4 we now have the following lemma.

Lemma 5. *The advantage of an adversary in distinguishing $\mathcal{F}$ and $\mathcal{P}[\mathcal{RO}]$ with the responses to a sequence of $N < 2^c$ queries Q^1 is upper bounded by:*

$$f_{\mathrm{P}}(N) = 1 - \prod_{i=0}^{N-1} \left(\frac{1 - \frac{i+1}{2^c}}{1 - \frac{i}{2^r 2^c}} \right) .$$

Proof. The proof is similar to that of Lemma 4. Since $\mathcal{F}$ is a permutation over $A \times C$ chosen randomly and uniformly, the only limitation is that for the i-th query, the image (or preimage) shall not be equal to any of the found images (or preimage), resulting in $(2^r 2^c) - i$ possibilities. This leads to $(2^r 2^c)_{(N)}$ possible outcomes each with probability $((2^r 2^c)_{(N)})^{-1}$ and $(2^r 2^c)^N - (2^r 2^c)_{(N)}$ outcomes with probability 0.

From inspecting Algorithm 3 if follows that the adversary obtains the greatest possible variational distance when he creates N rooted nodes. This leads to the same distribution as for the transformation simulator. The possible outcomes of the permutation simulator are a subset of the possible outcomes for $\mathcal{F}$. This gives:

$$\mathrm{Adv}(\mathcal{A}) \leq 1 - \frac{(2^r)^N (2^c - 1)_{(N)}}{(2^r 2^c)_{(N)}} = 1 - \prod_{i=0}^{N-1} \left(\frac{1 - \frac{i+1}{2^c}}{1 - \frac{i}{2^r 2^c}} \right) . \tag{5}$$

□

These lemmas and proofs result in the following theorem, where the proof is similar to that of Theorem 1.

Theorem 2. *A padded sponge construction calling a random permutation, $\mathcal{S}'[\mathcal{F}]$, is (t_D, t_S, N, ϵ)-indistinguishable from a random oracle, for any t_D, $t_S = O(N^2)$, $N < 2^c$ and and for any ϵ with $\epsilon > f_{\mathrm{P}}(N)$.*

If N is significantly smaller than 2^c, $f_P(N)$ can be approximated closely by:

$$f_P(N) \approx 1 - e^{-\frac{(1-2^{-r})N^2 + (1+2^{-r})N}{2^{c+1}}} < \frac{(1-2^{-r})N^2 + (1+2^{-r})N}{2^{c+1}}. \tag{6}$$

Note that using a random permutation results in a better bound than using a random transformation. By assigning distinct C-part values of rooted nodes, the simulators tend to generate an output distribution which is closer to that of a permutation than to that of a transformation.

5 Discussion and conclusions

We have proven that the sponge construction calling a random transformation or permutation is indifferentiable from a random oracle and obtained concrete bounds. Here, the security parameter is the capacity c and not the output length of the hash function. Note that other constructions also consider the size of the internal state as a security parameter, e.g., [13].

One may ask the question: what does this say about resistance to classical attacks such as collision-resistance, including multicollisions [9], (2nd) preimage resistance, including long-message attacks [10] and herding [11]? In general, it is expected that a hash function offers the same resistance as would a random oracle, truncated to the hash function's output length n. The success probability after q queries is about $q^2/2^{n+1}$ for generating collisions and $q/2^n$ for generating a (2nd) preimage. The sponge construction does not have a fixed output length. However, when a hash function with the sponge construction is used in an actual cryptographic scheme, its output will be truncated. Our indifferentiability bounds in terms of the capacity c permit to express up to which output length n such a hash function may offer the expected resistance. For example, it offers collision resistance (as a truncated random oracle would) for any output length smaller than the capacity and (2nd) preimage resistance for any output length smaller than half the capacity. In other words, when for instance $c = 512$, a random sponge offers the same resistance as a random oracle but with a maximum of 2^{256} in complexity.

A function with the sponge construction can be used to build a MAC function (by just pre-pending the key to the input) or, thanks to its long output, to build a synchronous stream cipher (by taking as input the concatenation of a key and an IV). Alternatively, the sponge construction can be used as a reference for expressing security claims when building new such designs.

Note that the bounds we have provided only hold when the sponge construction makes use of a random transformation or random permutation. When a concrete transformation or permutation is taken, no such bounds can be given. See for example [5] and also [14] for discussions on this subject. However, our bounds do say that using the sponge construction excludes generic attacks with a success probability higher than the maximum of our bound $\frac{N(N+1)}{2^{c+1}}$ and the success probability the attack would have for a random oracle. By generic attacks we mean here attacks such as those described in [9–11], that do not exploit specific properties of the transformation or permutation used but only properties of the construction.

References

1. E. Andreeva, G. Neven, B. Preneel, and T. Shrimpton, *Seven-property-preserving hashing: ROX*, Advances in Cryptology – Asiacrypt 2007 (K. Kurosawa, ed.), LNCS, no. 4833, Springer-Verlag, 2007, pp. 130–146.

2. M. Bellare and T. Ristenpart, *Multi-property-preserving hash domain extension and the EMD transform*, Advances in Cryptology – Asiacrypt 2006 (X. Lai and K. Chen, eds.), LNCS, no. 4284, Springer-Verlag, 2006, pp. 299–314.
3. M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM Conference on Computer and Communications Security 1993 (ACM, ed.), 1993, pp. 62–73.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, *Sponge functions*, Ecrypt Hash Workshop 2007, May 2007, also available as public comment to NIST from `http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html`.
5. R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, ACM Press, 1998, pp. 209–218.
6. D. Chang, S. Lee, M. Nandi, and M. Yung, *Indifferentiable security analysis of popular hash function with prefix-free padding*, Advances in Cryptology – Asiacrypt 2006 (X. Lai and K. Chen, eds.), LNCS, no. 4284, Springer-Verlag, 2006, pp. 283–298.
7. J. Coron, Y. Dodis, C. Malinaud, and P. Puniya, *Merkle-Damgård revisited: How to construct a hash function*, Advances in Cryptology – Crypto 2005 (V. Shoup, ed.), LNCS, no. 3621, Springer-Verlag, 2005, pp. 430–448.
8. I. Damgård, *A design principle for hash functions,*, Advances in Cryptology – Crypto '89 (G. Brassard, ed.), LNCS, no. 435, Springer-Verlag, 1989, pp. 416–427.
9. A. Joux, *Multicollisions in iterated hash functions. Application to cascaded constructions*, Advances in Cryptology – Crypto 2004 (M. Franklin, ed.), LNCS, no. 3152, Springer-Verlag, 2004, pp. 306–316.
10. J. Kelsey and B. Schneier, *Second preimages on n-bit hash functions for much less than 2^n work*, Advances in Cryptology – Eurocrypt 2005 (R. Cramer, ed.), LNCS, no. 3494, Springer-Verlag, 2005, pp. 474–490.
11. T. Kohno and J. Kelsey, *Herding hash functions and the Nostradamus attack*, Advances in Cryptology – Eurocrypt 2006 (S. Vaudenay, ed.), LNCS, no. 4004, Springer-Verlag, 2006, pp. 222–232.
12. H. Kuwakado and M. Morii, *Indifferentiability of single-block-length and rate-1 compression functions*, Cryptology ePrint Archive, Report 2006/485, 2006, http://eprint.iacr.org/.
13. S. Lucks, *A failure-friendly design principle for hash functions*, Advances in Cryptology – Asiacrypt 2005 (B. Roy, ed.), LNCS, no. 3788, Springer-Verlag, 2005, pp. 474–494.
14. U. Maurer, R. Renner, and C. Holenstein, *Inidifferentiability, impossibility results on reductions, and applications to the random oracle methodology*, Theory of Cryptography - TCC 2004 (Moni Naor, ed.), Lecture Notes in Computer Science, no. 2951, Springer-Verlag, 2004, pp. 21–39.
15. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.
16. R. Merkle, *One way hash functions and DES*, Advances in Cryptology – Crypto '89 (G. Brassard, ed.), LNCS, no. 435, Springer-Verlag, 1989, pp. 428–446.