Super-Sbox Cryptanalysis: Improved Attacks for AES-like Permutations

Henri Gilbert and Thomas Peyrin

Orange Labs, France
henri.gilbert@orange-ftgroup.fr

Ingenico, France
thomas.peyrin@ingenico.com

Abstract. In this paper, we improve the recent rebound and start-from-the-middle attacks on AES-like permutations. Our new cryptanalysis technique uses the fact that one can view two rounds of such permutations as a layer of big Sboxes preceded and followed by simple affine transformations. The big Sboxes encountered in this alternative representation are named Super-Sboxes. We apply this method to two second-round SHA-3 candidates Grøstl and ECHO, and obtain improvements over the previous cryptanalysis results for these two schemes. Moreover, we improve the best distinguisher for the AES block cipher in the known-key setting, reaching 8 rounds for the 128-bit version.

Key words: hash function, cryptanalysis, AES, Grøstl and ECHO.

1 Introduction

Hash functions are among the most important and widely spread primitives in cryptography. Informally a hash function H is a function that takes an arbitrarily long message as input and outputs a fixed-length hash value of size n bits. The classical security requirements for such a function are collision resistance and (second)-preimage resistance. Namely, it should be impossible for an adversary to find a collision (two different messages that lead to the same hash value) in less than $2^{n/2}$ hash computations, or a (second)-preimage (a message hashing to a given challenge) in less than 2^n hash computations. Recently, most of the standardized hash functions [29, 35] have suffered from major improvements in hash function cryptanalysis [38,39]. As a response, the NIST organized the SHA-3 competition [31] and 51 candidates were accepted to the first round. In July 2009, 14 of them have been selected to the second round. Among them, several hash proposals like Grøstl [14] or ECHO [2] use parts of the standardized block cipher AES [9, 30] as internal primitives or mimick the structure of this cipher.

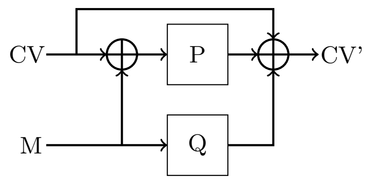
The separation between block ciphers and hash functions has always been blurry as many constructions [6, 34] are known that turn the former into the latter. For example, the Davies-Meyer mode converts a secure block cipher into a secure compression function and is incorporated in a large majority of the currently known hash functions. A major difference between the cryptanalysis of block ciphers and hash functions is that the attacker can fully control the inner behavior of a hash function. In other words, the attacker can use more efficiently the freedom degrees available on the input (i.e. the number of independent binary variables he has to determine). A new security model for block ciphers, the so-called *known-key model* [21], was recently proposed in order to fill the gap between those two situations. In this model, the secret key is known to the adversary and its goal is to distinguish the behavior of a random instance of the block cipher from the one of a random permutation by constructing a set of (plaintext, ciphertext) pairs satisfying an *evasive* property. Such a property is easy to check but impossible to achieve with the same complexity and a non-negligible probability using oracle accesses to a random permutation and its inverse. In particular, reduced versions of the `AES` have been studied in this setting [21, 28].[1] An even more demanding requirement for block ciphers, also introduced for filling the gap between block ciphers and hash functions, is to behave as an ideal cipher, *i.e.* a family of independent random permutations indexed by the key space, even when the key values can be chosen by an adversary. It has been recently shown that the full `AES`-256 does not behave as an ideal cipher due to the existence of a so-called *chosen key distinguisher* [4].

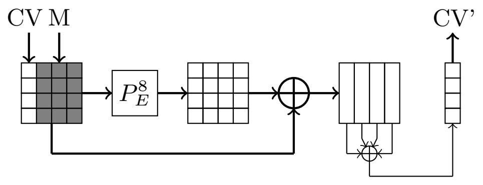
Cryptanalysis of `AES`-based hash functions began with the hash family proposal `Grindahl` [20] for which collision attacks have been found [19, 32]. This showed that truncated differentials [22] are useful when cryptanalyzing a byte-oriented primitive such as the `AES`. Later on, the rebound attack [27] was shown to lead to substantial efficiency improvements in the freedom degrees usage of the attacker [23,25,40]. The idea is to build a differential path and use the available freedom degrees in the "most expensive" part of the path. The "cheaper" parts are then covered in an inside-out manner in both forward and backward directions. More recently, improved variants of the initial rebound attack such as the so-called "start-from-the-middle" attack were also introduced [26].

Our contribution. In this paper, we further improve the rebound or start-from-the-middle attacks for `AES`-like permutations. The idea is to view two consecutive rounds of an `AES`-like permutation as the application of a so-called Super-Sbox [10, 11, 16]: this allows a more efficient use of the freedom degrees.[2] Instead of dealing with the classical 8-bit `AES` Sboxes, one will consider 32-bit Sboxes each composed of two `AES` Sbox layers surrounding one MixColumns and

[1] It was noticed in [7] that for any cipher which key space is smaller than the plaintext space, it is possible to construct a (plaintext, ciphertext) pair satisfying an evasive relation by encrypting the key under itself. However, known key distinguishers such as those of [21, 28] distinguish round-reduced versions of `AES` from a random permutation in a less contrived manner than such a generic evasive relation.

[2] Note that a similar technique, independently discovered by Lamberger et al. [23], has been applied by these authors to the Whirlpool hash function. However, this method uses the incoming round subkeys as additional freedom degrees and would not work as is for fixed-key `AES`-like permutations such as the ones studied in this article (for `ECHO` or `Grøstl`). Moreover, the known-key distinguishers would not apply anymore for `AES` as the key would have to be chosen by the attacker.





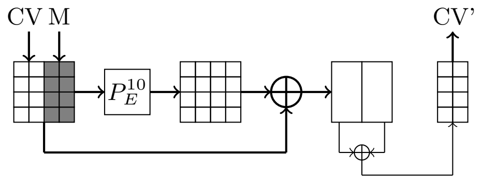


Fig. 1. The compression functions of Grøstl, ECHO-256, and ECHO-512 illustrated from the top down.

one AddConstant function. The resulting attack method, that we propose to name Super-Sbox cryptanalysis, is not only simpler than the previous ones, it also results in cryptanalysis performance improvements since we are now able to find deviations from the behavior of a random permutation up to 8 rounds. We also provide an analysis regarding the freedom degrees available during the attack. We apply this technique to (1) AES, (2) the internal permutations P and Q of Grøstl, and (3) the internal permutation P_E of ECHO. The link between the compression functions of Grøstl and ECHO and the underlying internal permutations is illustrated in Figure 1. Our results for AES and the internal permutations of Grøstl and ECHO are summarized in the first part of Table 1. We obtain an 8-round distinguisher in the known-key model for all versions of the AES block cipher. This is the first published "attack" against the 8-round version of AES-128, for which the full number of rounds is 10. We also present a distinguisher for the 8-round reduced internal permutations of Grøstl-256 (the full number of rounds is 10) and for the full number of rounds of the internal permutation of ECHO-256 (8 rounds). In the case of Grøstl-256, our distinguishers for round-reduced versions of the internal permutation can be immediately converted into distinguishers for reduced versions of the compression function with the same number of rounds, or even semi-free-start collisions in some cases.[3] We outline in the second part of Table 1 the results obtained for reduced versions of the Grøstl-256 compression function. In the case of ECHO and its reduced versions, our distinguishers for the internal permutation cannot be converted into distinguishers for the compression function due to the extra protection provided by the final shrinking stage of the compression function – namely its convolution effect on the output distribution of the permutation.

Table 1. The first table gives a summary of results for AES and the internal permutations used in Grøstl-256 and ECHO. The second table shows the results for the compression functions of Grøstl-256 and ECHO. Some structural observations [1,18] for Grøstl have not been included in the Tables.

target	rounds	computational complexity	memory requirements	type	source
AES	7	2^{24}	2^{16}	known-key-distinguisher	see [26]
	8	2^{48}	2^{32}	known-key-distinguisher	this paper
Grøstl-256 permutation	7	2^{55}		distinguisher	see [26]
	8	2^{112}	2^{64}	distinguisher	this paper
ECHO internal permutation	7	2^{384}	2^{64}	distinguisher	see [26]
	8	2^{768}	2^{512}	distinguisher	this paper

target	rounds	computational complexity	memory requirements	type	source
Grøstl-256 comp. function	6	2^{120}	2^{64}	semi-free-start collision	see [27]
	6	2^{64}	2^{64}	semi-free-start collision	see [26]
	7	2^{120}	2^{64}	semi-free-start collision	this paper
	7	2^{55}		distinguisher	see [26]
	8	2^{112}	2^{64}	distinguisher	this paper
ECHO comp. function	none	none		none	—

2 Description of the analyzed schemes

We give in this section a generic description of an AES-like permutation and we then provide the parameters in this generic model for AES, Grøstl and ECHO. We refer to the corresponding specifications [2,9,14,30] for a detailed description of these schemes.

A generic n-bit AES-like permutation has an internal state that can be viewed as a square matrix of c-bit cells with r columns and r rows. A cell will be denoted by $C_{i,j}$, where i is its row position and j its column position in the matrix, starting the counting from 0. The permutation is composed of R rounds and each round has four layers. The first layer (AddConstant or AC) is a constant/key

[3] Similar results were presented at CT-RSA 2010 [13] while the final version of this paper was in preparation.

addition function. More precisely, for each cell of the internal state, we XOR a c-bit constant. The second layer (SubBytes or SB) is a non-linear function defined by the application of an Sbox S: for each cell $C_{i,j}$ of the internal state, we compute $C'_{i,j} = S[C_{i,j}]$. The third layer (ShiftRows or ShR) permutes the position of each cell in its own row: for each cell $C_{i,j}$ of the internal state, we compute $C'_{i,j} = C_{i,Sub_i(j)}$ where $Sub_i(j)$ is parametrized by the row i. Finally, the last layer (MixColumns or MC) is a linear function that mixes all the columns of the internal state separately. The round function on an internal state C can thus be defined as:

$$MixColumns \circ ShiftRows \circ SubBytes \circ AddConstant(C).$$

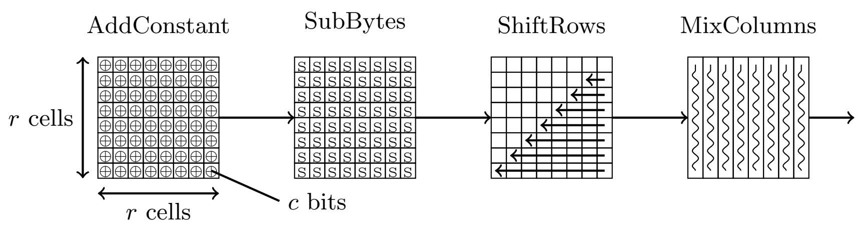


Fig. 2. The generic `AES`-like permutation

Even though the MixColumns functions used in `AES`, `Grøstl`, and `ECHO` are distinct (we do not provide here their detailed specification), we can stick to this generic description since we are only interested in the differential properties of this layer, which remain essentially the same.

Extended known key model for the considered schemes In what follows, we will introduce distinguishers for permutations and compression functions based upon specific examples of the above generic `AES`-like permutation. Therefore we have to explain what we mean by "distinguisher" in this context. Our aim is to capture structural properties that distinguish the behavior of permutations or functions belonging to a family of permutations (resp. functions) indexed by a parameter from the one of a random permutation (resp. a random function). In the case of permutations, these parameters can be viewed as a known key and our notion of distinguisher entirely coincides with the notion of known key distinguisher. In order to also cover the case of compression functions, we introduce a natural extension of the notion of known key distinguisher to a family $F = \{f_i\}$ of functions indexed by a parameter $i \in I$. We (informally) define a distinguisher for F as a procedure allowing an adversary to construct, when input with a randomly drawn parameter value i of a function of F, a tuple of (input, output) pairs for f_i satisfying (with a non negligible probability) an *evasive* property independent of i. An evasive property means in this context a property impossible to achieve with the same complexity and a non-negligible probability using oracle accesses to a random function.[4] We propose to view the permutations and compression functions based upon the generic AES construction as families of permutations (resp. functions) indexed by the parameter set $I = \mathcal{C} \times \mathcal{SB}$ equipped with the uniform probability distribution. For a given family, $\mathcal{C}$ is defined as the set of possible values for the constants involved in the various AddConstant layers, and $\mathcal{SB}$ represents the set of tuples of r^2R permutations involved in the various SubBytes layers (R represents as before the number of rounds of the considered instances of the generic AES construction, and r the number of rows and columns of the matrices representing their states). This allows to define structural distinguishers for these permutations and these compression functions, using the extended known key model introduced above.

2.1 AES

Following our generic description, AES [30] is a $n = 128$-bit block cipher that can handle 128, 192 or 256-bit keys and those variants have a different number of rounds, 10, 12 and 14 respectively. The internal state is viewed as a 4×4 matrix of bytes and SB is an 8-bit Sbox. The ShiftRows transformation is simply defined by $Sub_i(j) = (j - i) \bmod 4$. Finally, we note that in AES the MixColumns transformation of the last round is not applied and that the last round is composed with an extra AddConstant transformation.

Since we will analyze AES in the known-key attacker model, the key schedule and the key additions can be replaced by the AddConstant function. In the known key distinguishers for R-round versions of AES considered in the sequel, the set $\mathcal{C}$ consists of all $R+1$-tuples of 128-bit constants equiped with the uniform law, and the set $\mathcal{SB}$ can be either defined as the singleton containing the actual SubBytes layers or as the set of all $16R$-tuples of bijections over $\{0,1\}^8$. Our distinguishers are equally applicable in both settings, and can be immediately converted into known key distinguishers for R-round versions of AES-128, AES-192, and AES-256.

2.2 Grøstl

Grøstl [14] is a double-pipe hash function whose compression function is built upon two AES-like permutations P and Q (that only differ by the constants used

[4] Considering a family of functions rather than one single function allows us to express a structural property common to many individual functions. Moreover, it avoids the considerable difficulties one would encounter in the case of one single function f for expressing the requirement that the evasive property used to distinguish f must be "independent" of f as to exclude for instance the evasive property trivially provided by each (input,output) pair of f; in addition, if the size of I is larger than the input size of F, considering a family of functions instead of one single function allows us to avoid the already mentioned paradox of [7].

during the AddConstant layer). In the case of Grøstl-256, the internal state of those permutations can be viewed as a 8×8 matrix of bytes and their number of rounds is 10. The ShiftRows transformation is defined by $Sub_i(j) = (j-i) \bmod 8$. In the case of Grøstl-512, the internal state of those permutations can be viewed as a 8×16 matrix of bytes, thus not fitting in our generic model.

Finally, as already shown on Figure 1, the compression function takes a message input M and a chaining variable input CV and outputs a new chaining variable CV' with

$$CV' = P(CV \oplus M) \oplus Q(M) \oplus CV.$$

In the subsequent analysis of the security of R-round versions of the Grøstl-256 permutations and the associated compression function, we either define the set $\mathcal{C}$ as the singleton containing the actual constants used in the AddConstant layers or the set of all possible R-tuples of 512-bit constants (this will not make any difference for our distinguishers), and we define $\mathcal{SB}$ as the set of all the $64 \cdot R$-tuples of bijections over $\{0,1\}^8$.

2.3 ECHO

ECHO [2] is also a double-pipe hash function. It uses a compression function built upon a 2048-bit AES-like permutation whose i-round version is denoted by P_E^i. The internal state of this permutation can be viewed as a 4×4 matrix of 128-bit words. The Sbox layer on a 128-bit cell is composed of two AES rounds with a fixed key. The AddConstant layer is not present (or equivalently is present with constant values equal to zero) and in order to avoid trivial vulnerabilities that would result from an entirely symmetric round function, each Sbox in ECHO is distinct thanks to different key additions in each invocation of the 2-round AES. As for the AES, the ShiftRows transformation is simply defined by $Sub_i(j) = (j-i) \bmod 4$. In the case of the ECHO-256 compression function, 8 rounds of the permutation are applied and a shrinking transformation is performed after the final feedforward. This transformation (denoted here by shrink_{256}) consists of "XORing" all the four 512-bit columns together. Finally, as already shown on Figure 1, the compression function takes a message input M and a chaining variable input CV and outputs a new chaining variable CV' with

$$CV' = \mathsf{shrink}_{256}(P_E^8(CV||M) \oplus (CV||M)).$$

In the case of the ECHO-512 compression function, 10 rounds of the permutation are applied and a shrinking transformation is applied after the final feedforward. This transformation (denoted by shrink_{512}) consists of "XORing" the two first and the two last 512-bit columns together.

$$CV' = \mathsf{shrink}_{512}(P_E^{10}(CV||M) \oplus (CV||M)).$$

Since ECHO is a nested design of AES-like permutations, we will use the prefix "BIG" when referring to one of the three layers of the 2048-bit permutation.

When not using this prefix, we will refer to the layers of the 2-round AES permutation in the BIG-Sbox of ECHO.

In the subsequent analysis of the security of R-round versions of the ECHO permutation, we define the set $\mathcal{C}$ as the singleton containing the R-tuple of AddConstant layers associated with the constant zero, and $\mathcal{SB}$ as the set of all the $16 \cdot R$-tuples of bijections over $\{0,1\}^{128}$.

3 The Super-Sbox cryptanalysis technique

In this section, we introduce the Super-Sbox view for two rounds of an AES-like permutation [9–11, 16]. Based on this observation, we describe a new cryptanalysis technique in the generic framework of Section 2 and we will apply it to the specific cases of AES, Grøstl and ECHO in the following section.

3.1 The generic differential paths

In the following attacks, we will consider two distinct generic truncated differential paths for AES-like permutations. The only difference considered between two words A and A' is the XOR difference, that is $\delta = A \oplus A'$. The first path is 7-round long and the second one is 8-round long. Both are depicted in Figure 3. A white cell denotes a c-bit word without difference (inactive word) and a dark cell represents a truncated c-bit difference (active word), that is a non-zero difference whose actual value is not considered by the attacker.

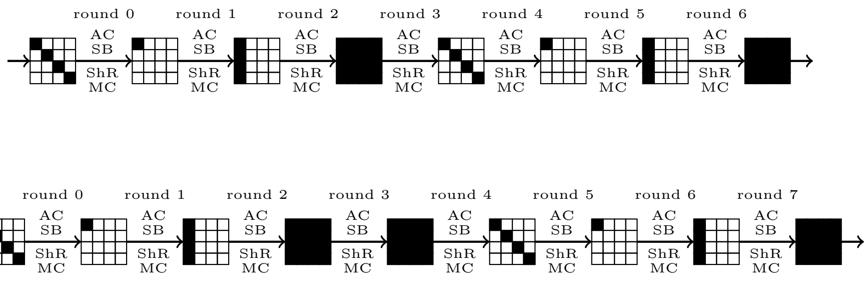


Fig. 3. 7-round and 8-round differential paths for AES-like permutations.

When dealing with truncated differentials for AES-like permutations, one can easily check that only the MixColumns transformations will not behave deterministically. Indeed, while AddConstant have no effect on the difference of a cell and ShiftRows just permutes the array of c-bit differences, the SubBytes transformation will impact the value of the difference, but it will not affect the truncated difference. The matrix multiplication underlying the MixColumns transformation presents the interesting property of being a Maximum-Distance Separable (MDS) mapping: the number of active input and output cells is always greater or equal to $r + 1$ (unless there is no active input and output cell at all). The probability of a truncated differential transition through the restriction of the MixColumns transformation to one column that meets the MDS constraints is determined by the number of active cells in the output column. More precisely, if such a differential transition contains $k > 0$ active cells in the output column, its probability of success is closely approximated by $2^{-c(r-k)}$. For example, a $4 \mapsto 1$ transition for one column of the `AES` MixColumns layer has a success probability of approximatively 2^{-24}. Note that the same reasoning applies when dealing with the inverse function of the MixColumns layer as well.

3.2 Previous start-from-the-middle attacks

By observing the two previous differential paths, one can easily be convinced that the most costly part is located in the middle rounds, where the full internal state is active. Therefore, the classical early-round use of the freedom degrees available to the attacker is not successful in this case. It is more efficient to actually utilize the freedom degrees during the middle rounds and then let the rest of the differential trail be verified backward and forward in a probabilistic way.

3.3 The Super-Sbox view

In [9–11, 16], Daemen and Rijmen introduced the super box representation for two rounds of the `AES` in order to study differential properties. The underlying idea is simple: by considering $(r \times c)$-bit permutations (named here Super-Sboxes) instead of the usual c-bit S-boxes, two rounds of an `AES`-like permutation can be represented using only one non-linear layer. More precisely, the application of two `AES`-like permutation rounds on a internal state C

$$\mathsf{MC} \circ \mathsf{ShR} \circ \mathsf{SB} \circ \mathsf{AC} \circ \mathsf{MC} \circ \mathsf{ShR} \circ \mathsf{SB} \circ \mathsf{AC}(C)$$

can be rewritten

$$\mathsf{MC} \circ \mathsf{ShR} \circ \mathsf{SB} \circ \mathsf{AC} \circ \mathsf{MC} \circ \mathsf{SB} \circ \mathsf{ShR} \circ \mathsf{AC}(C)$$

since two adjacent SubBytes and ShiftRows transformations commute. The middle part

$$\mathsf{Super\text{-}SB} = \mathsf{SB} \circ \mathsf{AC} \circ \mathsf{MC} \circ \mathsf{SB}$$

of the former composition represents a layer of column-wise applications of r $(r \times c)$-bit Super-Boxes. The transformation associated with two consecutive rounds can thus be rewritten

$$\mathsf{MC} \circ \mathsf{ShR} \circ \mathsf{Super\text{-}SB} \circ \mathsf{ShR} \circ \mathsf{AC}(C).$$

This is depicted in Figure 4.

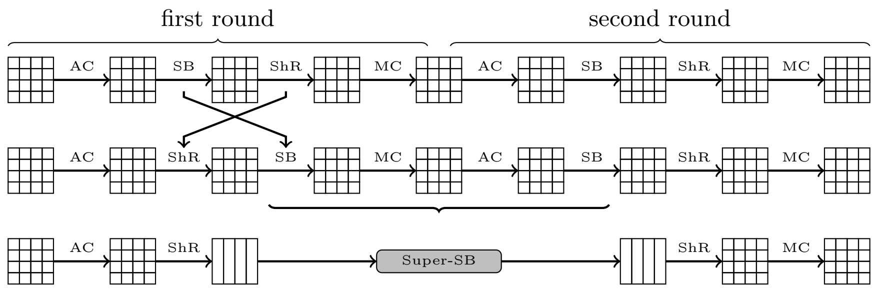


Fig. 4. Three equivalent views of 2 rounds of an `AES`-like permutation

3.4 The Super-Sbox cryptanalysis

Our overall strategy is the same as in the previous rebound or start-from-the-middle attacks: we will try to find a pair of internal state values in the middle of a well chosen truncated differential path (where the full internal state is active) such that the path is verified for as many possible rounds as possible backward and forward. We call this part the *controlled rounds* and the rest of the path in both directions will be fulfilled probabilistically.

In order to describe our attack, we use the 8-round path from Figure 3. With a restricted number of operations on average, we will find an internal state values pair such that the path is verified for three middle rounds: from the beginning of round 2 until the end of round 4. A more detailed description of the three controlled rounds is given in Figure 5.

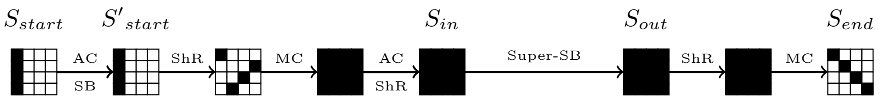


Fig. 5. A detailed Super-Sbox view (see Figure 4) of the controlled rounds (from round 2 to round 4) of the 8-round differential path from Figure 3

Before describing the attack, we have to make an assumption: we require that the number of potential distinct differences at the start of the controlled rounds (S_{start}) or at the end of the controlled rounds (S_{end}) be at least $(2^c - 1)^r$. In other words, we must have at least r active cells in S_{start} or in S_{end}. Note that this assumption holds for all the types of controlled rounds that will be considered in the two differential paths of Figure 3. Without loss of generality, we consider for the rest of the description that this assumption is fulfilled for the ending difference mask.

The controlled rounds. The initial state of the controlled round is S_{start} at the beginning of round 2 (see Figure 5). Since the AddConstant and SubBytes layers of this round have no effect on truncated differentials, we can directly begin at S'_{start}, *i.e.* just after those two transformations. Thus, we select a random difference value (we do not use truncated differences in the subsequent procedure) for all the active cells of the internal state S'_{start} at the output of the Sbox layer of round 2. Since the difference mask for the entire internal state is now specified, one can apply the ShiftRows and MixColumns transformations and enter round 3 with an updated difference mask. We then easily deduce the input difference mask $\Delta = (\Delta_1, \cdots, \Delta_r)$ in S_{in} for the r-tuple of Super-Sboxes of Super-SB.

Now we perform the following local precomputation: for each of the r Super-Sboxes, knowing its input difference mask Δ_i, we go through all the $2^{(rc)-1}$ pairs of input values differing by Δ_i and compute the Super-Sbox forward. This provides $2^{(rc)-1}$ output differences values (distinct or not). For each Super-Sbox output difference reached, the attacker stores the appropriate pair(s) of input that led to it. We name this storage by Tables T_i for each Super-Sbox i and note that this precomputation phase requires about 2^{rc} operations and memory.

We now go backward by starting from the end of round 4: we pick a random difference for all the active cells of the internal state S_{end} at the output of the MixColumns transformation of round 4. We can invert this MixColumns layer and get the differences on its input. By also inverting the ShiftRows function of round 4, we get the aimed output difference mask $\Delta' = (\Delta'_1, \cdots, \Delta'_r)$ in S_{out} for the r Super-Sboxes of Super-SB. We only have to check whether for all the r Super-Sboxes (numbered $i = 1$ to r), the output difference Δ'_i is present in tables T_i. If this is the case, we can efficiently enumerate all the pairs of input difference Δ in S_{in} leading to an output Δ' in S_{out}. It is easy to see that if $\Delta' = (\Delta'_1, \cdots, \Delta'_r)$ was a fully random r-tuple of output differences for the r Super-Sboxes, the average value over Δ' of the number $n(\Delta')$ of distinct pairs of input difference Δ resulting in an output difference Δ' would be exactly $1/2$ (this actually holds for any permutation, independently of the fact that this permutation is a r-tuple of Super-Sboxes).[5] We make the (natural) heuristic assumption that though Δ' is not selected from the whole set $\mathcal{D}$ of all 2^{r^2c} possible difference values, but from a smaller subset $\mathcal{D}'$ of $(2^c - 1)^r \simeq 2^{rc}$ difference values, the average value of $n(\Delta)$ over $\mathcal{D}'$ remains extremely close to $1/2$. This assumption is supported by the fact that $\mathcal{D}'$ consists of difference values with

[5] The exact distribution of the number $n(\Delta')$ of pairs is complex to derive, but one can at least notice that if $n(\Delta') \neq 0$, *i.e.* the numbers $n_1, \cdots, n_r$ of input pairs returned by tables $T_1, \cdots, T_r$ are all distinct from zero, then $n(\Delta') = n_1 \times \cdots \times n_r \times 2^{r-1}$ since each tuple of pairs provides 2^{r-1} pairs of complete blocks (as a matter of fact, the values of each pair of inputs to one Super-Sbox can be swapped, but each of the 2^r pairs of blocks one obtains this way are repeated two times). Though this is not essential for the estimate of the attack complexity, we can expect the two most probable values of $n(\Delta')$ to be $n(\Delta') = 0$ (with a probability about $1 - 2^{-r}$) and $n(\Delta') = 2^{r-1}$ (with a probability about 2^{-r}).

r^2 active cells, and this output difference pattern meets the MDS constraints of the r Super-Sboxes for any input difference pattern.

Thus, by going through all the potential difference candidates for S_{end}, we expect to get half of the amount of distinct solutions on average. Since we previously assumed that we have at least 2^{rc} potential difference candidates for S_{end}, the average complexity to find one solution is only two operations as soon as we wish to obtain at least 2^{rc-1} solutions. In other words, the 2^{rc} cost for building the tables T_i has been absorbed by the fact that this will allow us to test about 2^{rc} distinct output difference values at a time. Once all the possible output differences in S_{end} have been exhausted, we can pick a new input difference candidate in S_{start} and build new tables T_i.

To conclude, in order to find k distinct solutions for the controlled rounds, the overall complexity is $\max\{2^{rc}, k\}$ operations and 2^{rc} memory.

The uncontrolled rounds. The rest of the path (the uncontrolled rounds) is fulfilled probabilistically. More precisely, we managed so far to get valid candidates from round 2 to round 4, but we have no control on the difference values in S_{start} (since we selected random differences in S'_{start} and going through an Sbox layer impacts the difference values). Similarly, we know the differences in S_{end}, but the beginning of the next round is a SubBytes transformation that does not allow us to control the behavior of the MixColumns function on round 5. The study of the MixColumns differential properties indicates that the path will be fulfilled with probability about $2^{-c(r-1)}$ at round 1 and at round 5 since we are aiming for a $r \mapsto 1$ active cells differential transition through both of those two MixColumns layers. Note that for round 0, round 6 and round 7, the probability of success is equal to one. Finally, the attacker must find $2^{2c(r-1)}$ distinct solutions for the controlled rounds, providing a single valid pair for the whole 8-round differential path with a total complexity of $2^{2c(r-1)}$ operations and 2^{rc} memory.

Of course, the same technique applies to the 7-round differential characteristic of Figure 3 as well, up to the fact that since the condition on the number of distinct difference values is fulfilled at the input of the Super-Sboxes and not on the output, one has to fix a random value for the active cell of the ending difference (at the input of round 5) and work backward instead of forward. Since only one round of the uncontrolled part (namely round 1) has now to be fulfilled probabilistically, the attacker must find $2^{c(r-1)}$ distinct solutions for the controlled rounds, providing a valid pair for the whole 7-round differential path with a total complexity of about 2^{rc} operations and memory. If one wants to find k' solutions for the whole 7-round path, then the computational complexity is $\max\{2^{rc}, k' \times 2^{c(r-1)}\}$.

3.5 Considering freedom degrees

Before moving forward into the study of the various applications of the Super-Sbox cryptanalysis, we need to evaluate the freedom degrees available to the attacker. Indeed, we have to be sure that our attacks will find with a good probability a valid pair for the whole differential path considered. That is, we want to be sure that enough valid candidates for the controlled rounds exist so that we have a good probability that one of them will fulfill the entire characteristic. Moreover, in some of our attacks, our goal will not only be to find one valid pair, but to find many of them. In the case of some round reduced versions of `Grøstl`, we will even use a birthday paradox technique on the set of valid candidates in order to find a semi-free-start collision for the compression function. For this reason, we need to evaluate how many valid pairs one can find for a specified differential path, and how many distinct differential paths can be considered.

A simple counting argument shows that one can generate only about 2^{2c-1} pairs that verify the entire characteristic. Let us first consider the 8-round path of Figure 3: the controlled rounds allow to produce about 2^{2rc-1} valid pairs for rounds 2 to 4, out of which about $2^{2rc-1} \times 2^{-2(r-1)c} = 2^{2c-1}$ pairs fulfill the entire condition resulting from the differential transitions at round 1 and 5. In the case of the 7-round path from Figure 3, the controlled rounds allow to produce about $2^{(r+1)c-1}$ valid pairs from round 2 to 4, out of which about $2^{(r+1)c-1} \times 2^{-(r-1)c} = 2^{2c-1}$ fulfill the entire condition resulting from the differential transitions at round 1.

We also have to count how many differential paths such as the ones from Figure 3 can be generated. When the internal state contains only one active cell, there are clearly r^2 possible positions for the location of this cell in the matrix. Since this situation happens in the forward and in the backward direction, we get r^4 distinct differential paths. To conclude, we can generate r^4 distinct differential paths, each potentially producing 2^{2c-1} distinct valid pairs.

4 Applications

When trying to obtain distinguishing attacks for 7 rounds of an `AES`-like permutation, the Super-Sbox cryptanalysis will generally not provide any complexity improvement over existing techniques. However, our method allows the attacker to carry out an attack on a number of rounds that was unreachable before. For example, we provide a new known-key distinguisher attack for 8-round reduced `AES` and the first distinguishers for the 8-round versions of the reduced `Grøstl` internal permutation, the `ECHO` internal permutation, and the reduced `Grøstl` compression function.

4.1 Limited-birthday distinguishers

Before moving to applications of the Super-Sbox cryptanalysis, we have to describe the distinguishers we will build. One of our goals is to distinguish an `AES`-like permutation from an ideal permutation in the known-key setting. The kind of distinguishers we consider consist in deriving pairs of plaintext/ciphertext couples with a zero difference value at i prescribed input bit positions and a zero difference value at j prescribed output bit positions (and arbitrary difference values for the other $r^2c - i$ input bit positions and the other $r^2c - j$ output bit positions). What is the generic attack complexity in the case of an ideal (random) permutation ? More generally, we can study the problem of mapping a i-bit difference mask not necessarily equal to the all-zero word to a j-bit difference mask through an ideal permutation. A rough analysis might suggest that due to the the birthday paradox, a generic attack requiring $2^{\min\{i/2,j/2\}}$ exists. However, this is not always the case since we can find ourselves in the situation where not enough difference positions are available in order to take full advantage of the birthday attack. In other words we don't always have the $k/2$ unconstrained difference bits required to mount a $2^{k/2}$ collision attack on k bits.

Since we handle a permutation, the attacker can choose to study the function or its inverse. Without loss of generality, let's assume that $i \geq j$. Due to the birthday paradox, each structure of 2^{r^2c-i} input values obtained by fixing the value of those i bits where a zero input difference is required allows to achieve a zero output difference on up to $2(r^2c - i)$ prescribed output bit positions.

- if $j \leq 2(r^2c - i)$, then one can select $2^{j/2}$ input values from one single structure and this suffices to achieve a collision on the j target positions. The attack complexity is about $2^{j/2}$.
- if $j > 2(r^2c - i)$, then about $2^{j-2(r^2c-i)}$ structures have to be used to obtain a collision on the j prescribed positions. Overall, the complexity of the attack is about $2^{r^2c-i} \times 2^{j-2(r^2c-i)} = 2^{i+j-r^2c}$.

The same reasoning holds when applying the birthday paradox over the $r^2c - j$ free difference bits on the output and attacking the inverse function.

- if $i \leq 2(r^2c - j)$, then the attack complexity is about $2^{i/2}$.
- if $i > 2(r^2c - j)$, then the attack complexity is about $2^{r^2c-j} \times 2^{i-2(r^2c-j)} = 2^{i+j-r^2c}$.

It can be shown that overall, the attack complexity is $\max\{2^{j/2}, 2^{i+j-r^2c}\}$.

We want to be able to distinguish `AES`-like permutation-based compression functions as well. Studying the generic attack for an ideal compression function is almost the same as previously. The only difference is that we cannot consider the inverse function anymore, and we have to take into account both the message and the chaining variable as inputs. Thus, we study the problem of mapping a i-bit zero difference mask on the input chaining variable and the message (with t denoting the total number of input bits) to a j-bit zero difference mask on the output through an ideal compression function. By applying the birthday paradox, each structure of 2^{t-i} input values obtained by fixing the input values at the i positions of the input mask bits allows to achieve a collision on up to $2(t - i)$ prescribed output bit positions.

- if $j \leq 2(t - i)$, then the attack complexity is $2^{j/2}$.
- if $j > 2(t - i)$, then $2^{j-2(t-i)}$ structures have to be used to obtain a collision on the j prescribed positions. Overall, the complexity of the attack is $2^{t-i} \times 2^{j-2(t-i)} = 2^{i+j-t}$.

4.2 AES

Our first application is a known-key distinguishing attack against the AES block cipher. We will focus on the application of this attack to AES-128. Previously known attacks on round-reduced version of AES-128 allow to reach up to 7 rounds, and for 7 rounds we get no improvement, due to the minimal cost 2^{rc} of the Super-Sbox technique. However, we describe here the first known-key distinguishing attack against a 8-round reduced version of AES-128 (a recent announcement regarding an unpublished work [5] describes an 8-round chosen-key distinguisher for AES-128). We recall that in the case of AES, the last MixColumns transformation is not applied.

We will use the 8-round differential path from Figure 3 and we already showed that one can get a pair of input fulfilling this path with a computation complexity of $2^{2c(r-1)} = 2^{48}$ operations and 2^{32} in memory. The amount of freedom degrees is not an issue here since we only need to find one candidate verifying the whole differential path. This gives us a pair of plaintext/ciphertext with 4 active cells in the input and 4 active cells in the output, with undetermined non-zero differences. In the previous section, we gave some evidence that in the case of a perfect random permutation this should require 2^{64} operations, and we can conclude that 8-round reduced AES-128 can be distinguished from an ideal cipher in a known-key model with 2^{48} computations and 2^{32} memory. Note that our distinguishers work even if the last round MixColumns transformation is applied.

4.3 Grøstl

For Grøstl, finding a valid pair following the generic 8-round path from Figure 3 requires $2^{2c(r-1)} = 2^{112}$ computations and 2^{64} memory. The obtained pairs have the distinctive property that $i = 512 - 64 = 448$ predetermined bits of the input difference and $j = 512 - 64 = 448$ predetermined bits of the preimage of the output difference by the linear transformation MixColumns are equal to zero. We thus obtain a distinguisher for the 8-round reduced Grøstl-256 internal permutation since the ideal cipher case should require $2^{i+j-r^2c} = 2^{384}$ computations. This immediately provides a distinguisher for the 8-round reduced Grøstl-256 compression function as well, as can be seen on Figure 1, by using the differential path for the P permutation and inserting no difference in the message (no difference will occur in Q) or alternatively with the differential path for the permutation Q only (and no difference in P). In both cases, the input difference of the compression function belongs to a predetermined vector space of $\{0,1\}^{1024}$ of dimension $8 \times 8 = 64$ and the output difference belongs to the sum of two predetermined vector spaces of $\{0,1\}^{512}$ of dimension 64 each, *i.e.* a predetermined vector space of dimension at most 128 (by analogy $i = 1024 - 64 = 960$ and $j = 512 - 128 = 384$). In the ideal compression function case, this should require $2^{i+j-t} = 2^{320}$ computations. For completeness, we give in Figure 6 the differential paths for the Grøstl parameters.

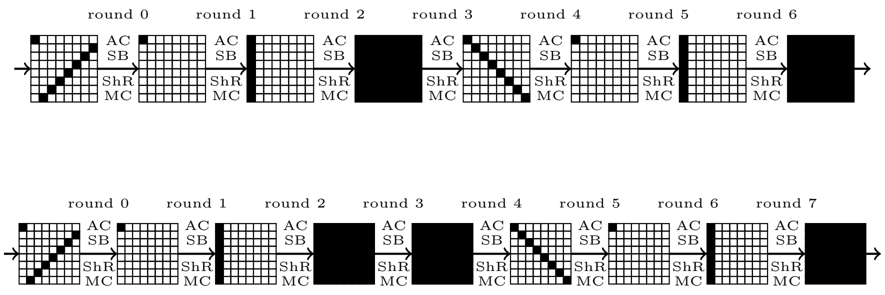


Fig. 6. 7-round and 8-round differential paths for `Grøstl`-256

Note that in both the 7-round and 8-round cases, we can generate $2^{2\times 8-1} = 2^{15}$ distinct valid pairs for each characteristic and one can build $8^4 = 2^{12}$ distinct differential paths. We can now try to compute semi-free-start collisions in the same manner as in [27]: we use the birthday paradox between the solutions found for the P and Q branches in order to find colliding difference values for the active cells of the input and the output. If one expects x active cells in the input and y in the output of the differential path, then one can find colliding values by computing $2^{(x+y)/2}$ valid candidates for both P and Q. Note that since it is linear, the very last MixColumns function can be ignored and only the number of active cells before this layer should be considered. Assuming that we have $2^{12} \times 2^{15} = 2^{27}$ freedom degrees available in order to apply this birthday attack would be incorrect: one can apply the birthday paradox only for the same differential path considered. Thus, we have 2^{15} freedom degrees for each birthday attack, and we can repeat this step 2^{12} times. Overall, we can make the input and output difference values collide for only $\log_2((2^{15})^2 \times 2^{12}) = 42$ bits.

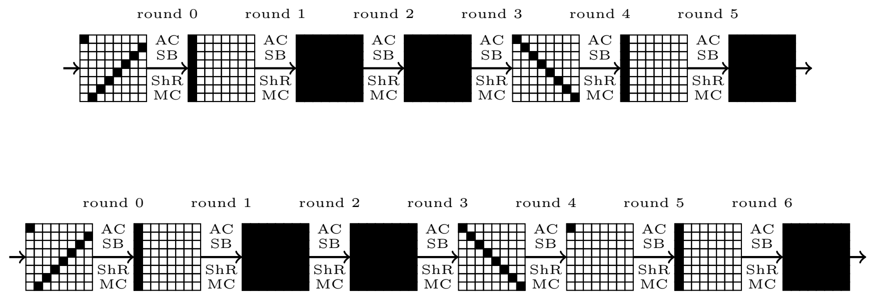


Fig. 7. 6-round and 7-round differential paths for `Grøstl`-256

For this reason, one can not generate collisions for the 7-round and 8-round reduced compression function with the paths from Figure 3. However, by using the slightly different paths depicted in Figure 7, one can find semi-free-start collisions for the 6-round reduced Grøstl-256 compression function with $2^{64} = 2^{64}$ operations and memory and for the 7-round reduced case with $2^{56+64} = 2^{120}$ operations and 2^{64} memory. The particularity of those paths is that enough freedom degrees are now left to the attacker in order to complete the final birthday attack. The drawback is that one more $r \mapsto 1$ transition is uncontrolled for the same total number of rounds. Thus, this type of paths is more costly than the one from Figure 6.

4.4 ECHO

Since its structure is mimicking the AES, our results regarding the internal permutation of ECHO are very similar, but the complexity has to adapted to the ECHO parameters. By using the 8-round path from Figure 3, we can distinguish 8 rounds of the ECHO internal permutation from an ideal 2048-bit permutation with 2^{768} computations and 2^{512} memory (the ideal permutation case would require 2^{1024} computations).

Note that this distinguisher does not apply to the ECHO compression function because of the shrink operation utilized after the internal permutation and the feedforward. As a matter of fact the convolution effect of this operation over the output distribution of the permutation makes it considerably more difficult to mount a distinguishing attack on the compression function of ECHO than on its underlying permutation. Moreover, since the Super-Sbox cryptanalysis of the ECHO permutation presented above requires at least 2^{512} computations and memory, it is not a well suited starting point for trying to mount a distinguisher or a collision search attack against one of the compression functions of ECHO-256 or ECHO-512 (or one of their single-pipe variants).

5 Conclusion

In this paper, we introduced the Super-Sbox cryptanalysis, which very often improves upon the classical rebound or start-from-the-middle attacks both in terms of efficiency and simplicity. This technique leads to improved cryptanalytic results for both Grøstl and ECHO, two SHA-3 candidates, and to the best known-key distinguisher so far for the AES-128 block cipher.

Acknowledgments

The authors would like to thank Matt Robshaw for insightful discussions and the anonymous referees for their helpful comments.

References

1. P.S.L.M. Barreto. An observation on Grøstl. Comment submitted to the NIST hash function mailing list, hash-forum@nist.gov. http://www.larc.usp.br/ pbarreto/Grizzly.pdf.
2. Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. SHA-3 Proposal: ECHO. Submission to NIST, 2008. Available online at http://crypto.rd.francetelecom.com/echo/.
3. Alex Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.
4. Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In Halevi [15], pages 231–249.
5. Alex Biryukov and Ivica Nikolic. A New Security Analysis of AES-128. CRYPTO 2009 rump session, 2009. http://rump2009.cr.yp.to/b6f3cb038135799a7ea398f99faf4a55.pdf.
6. John R. Black, Phillip Rogaway, and Thomas Shrimpton. Black-Box Analysis of the Block-Cipher-Based Hash-Function Constructions from PGV. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 320–335. Springer-Verlag, 2002.
7. Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. J. ACM, 51(4):557, 2004.
8. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.
9. Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Information Security and Cryptography. Springer, 2002. ISBN 3-540-42580-2.
10. Joan Daemen and Vincent Rijmen. Two-Round AES Differentials. Cryptology ePrint Archive, Report 2006/039, 2006. http://eprint.iacr.org/.
11. Joan Daemen and Vincent Rijmen. Understanding Two-Round Differentials in AES. In Roberto De Prisco and Moti Yung, editors, *SCN*, volume 4116 of *Lecture Notes in Computer Science*, pages 78–94. Springer, 2006.
12. Orr Dunkelman, editor. *Fast Software Encryption, 16th International Workshop, FSE 2009, Leuven, Belgium, February 22-25, 2009, Revised Selected Papers*, volume 5665 of *Lecture Notes in Computer Science*. Springer, 2009.
13. Florian Mendel and Christian Rechberger and Martin Schläffer and Søren Steffen Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In Josef Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010*, LNCS. Springer, 2010. to appear.
14. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Submission to NIST, 2008. Available online at http://www.groestl.info.
15. Shai Halevi, editor. *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*. Springer, 2009.
16. Joan Daemen and Vincent Rijmen. Plateau Characteristics. Information Security, IET, vol. 1, no. 1, pages 11–17, March 2007.
17. Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors. *Selected Areas in Cryptography, 16th Annual International Workshop, SAC 2009,*

Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers, volume 5867 of *Lecture Notes in Computer Science*. Springer, 2009.

18. J. Kelsey. Some notes on Grøstl. Comment submitted to the NIST hash function mailing list, hash-forum@nist.gov. http://ehash.iaik.tugraz.at/uploads/d/d0/Grostl-comment-april28.pdf.
19. Dmitry Khovratovich. Cryptanalysis of Hash Functions with Structures. In Jr. et al. [17], pages 108–125.
20. Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Biryukov [3], pages 39–57.
21. Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In K. Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 315–324. Springer, 2007.
22. L.R. Knudsen. Truncated and Higher Order Differentials. In B. Preneel, editor, Fast Software Encryption – FSE 1994, volume 1008 of *Lecture Notes in Computer Science*, pages 196–211. Springer, 1994.
23. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Matsui [24], pages 126–143.
24. Mitsuru Matsui, editor. *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, volume 5912 of *Lecture Notes in Computer Science*. Springer, 2009.
25. Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolic, Yu Sasaki, and Martin Schläffer. Rebound Attack on the Full Lane Compression Function. In Matsui [24], pages 106–125.
26. Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Jr. et al. [17], pages 16–35.
27. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman [12], pages 260–276.
28. Marine Minier, Raphael C.-W. Phan, and Benjamin Pousse. Distinguishers for Ciphers and Known Key Attack against Rijndael with Large Blocks. In Preneel [33], pages 60–76.
29. National Institute of Standards and Technology. FIPS 180-1: Secure Hash Standard. http://csrc.nist.gov, April 1995.
30. National Institute of Standards and Technology. FIPS PUB 197, Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, U.S. Department of Commerce, November 2001.
31. National Institute of Standards and Technology. Announcing Request for Candidate Algorithm Nominations for a New Cryptographic Hash Algorithm (SHA-3) Family. *Federal Register*, 27(212):62212–62220, November 2007.
32. Thomas Peyrin. Cryptanalysis of Grindahl. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 551–567. Springer, 2007.
33. Bart Preneel, editor. *Progress in Cryptology - AFRICACRYPT 2009, Second International Conference on Cryptology in Africa, Gammarth, Tunisia, June 21-25, 2009. Proceedings*, volume 5580 of *Lecture Notes in Computer Science*. Springer, 2009.

34. Bart Preneel, Ren Govaerts, and Joos Vandewalle. Hash Functions Based on Block Ciphers: A Synthetic Approach. In Douglas Robert Stinson, editor, *Advances in Cryptology – CRYPTO 1993*, volume 773 of *Lecture Notes in Computer Science*, pages 368–378. Springer-Verlag, 1993.
35. Ronald L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. http://www.ietf.org/rfc/rfc1321.txt, April 1992.
36. Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.
37. Vincent Rijmen. Cryptanalysis and design of iterated block ciphers. Ph.D. thesis, KULeuven 1997.
38. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Shoup [36], pages 17–36.
39. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [8], pages 19–35.
40. Shuang Wu, Dengguo Feng, and Wenling Wu. Cryptanalysis of the LANE Hash Function. In Jr. et al. [17], pages 126–140.