The Rebound Attack and Subspace Distinguishers: Application to Whirlpool

Mario Lamberger[1], Florian Mendel[1], Christian Rechberger[2],
Vincent Rijmen[1,2], and Martin Schläffer[1]

[1] Institute for Applied Information Processing and Communications
Graz University of Technology, Inffeldgasse 16a, A–8010 Graz, Austria.
[2] Dept. of Electrical Engineering ESAT/COSIC, K.U.Leuven,
and Interdisciplinary Institute for BroadBand Technology (IBBT),
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium.

Abstract. We introduce the rebound attack as a variant of differential cryptanalysis on hash functions and apply it to the hash function Whirlpool, standardized by ISO/IEC. We give attacks on reduced variants of the Whirlpool hash function and the Whirlpool compression function. Next, we introduce the subspace problems as generalizations of near-collision resistance. Finally, we present distinguishers based on the rebound attack, that apply to the full compression function of Whirlpool and the underlying block cipher W.

Keywords: hash functions, cryptanalysis, near-collision, distinguisher

1 Introduction

A cryptographic hash function H maps a message m of arbitrary length to a fixed-length hash value h. Informally, a cryptographic hash function has to fulfill the following three classical security requirements: preimage resistance, second preimage resistance and collision resistance. The resistance of a hash function to these attacks depends in the first place on the length N of the hash value. Regardless of how a hash function is designed, an adversary will always be able to find preimages or second preimages after trying out about 2^N different messages. Finding collisions requires a much smaller number of trials: about $2^{N/2}$ due to the birthday paradox. A function is said to achieve *ideal security* if these bounds are guaranteed.

Although a satisfying formal definition of the collision resistance requirement is apparently still lacking, some recent work on the hash functions MD4, MD5 and SHA-1 has convinced many cryptographers that at least these hash functions can no longer be considered secure against collision attacks. [6,7,13,43,44]. As a consequence, people are evaluating alternative hash functions, e.g. in the SHA-3 initiative organized by NIST. During this ongoing evaluation, not only the three classical security requirements are considered. Researchers look at (semi-)free-start collisions, near-collisions, etc. Every demonstration of a 'behavior different from that expected of a random oracle,' is considered suspect, and so are weaknesses that are demonstrated only for the compression function and not for the full hash function.

In this paper, we provide a detailed analysis of the hash function Whirlpool. This hash function is based on a dedicated block cipher W, which was designed according to the Wide Trail design strategy. It is the only hash function standardized by ISO/IEC (since 2000) [1] that does not follow the MD4 design strategy.

One contribution of this paper is to give an in-depth account of the *rebound attack*, as a variant of differential cryptanalysis heavily optimized to the cryptanalysis of hash functions, and at the same time as a high-level model for hash function cryptanalysis. The rebound attack can be used to construct various types of collisions. Thus far, it has been very successful on designs that copy the simple byte-oriented structure of AES.

Our second contribution is the introduction and definition of the *subspace problems*, as a natural extension and formalization of the near-collision requirement. We give bounds for the difficulty of the subspace problems in the generic (or ideal) case. Finally, we show subspace distinguishers for the full compression function of Whirlpool, thereby demonstrating the first deviation from the ideal model of this function.

Parts of this work appeared before in abridged form in [26,31]. New in this paper are the extended descriptions of the rebound attacks, proper definitions for the two subspace problems and the proofs on the lower bounds of the query complexities of these problems in the generic case.

Organization. In Section 2, we describe the rebound attack. We introduce two subspace problems in Section 3 and give bounds for their difficulty in the generic case. We describe the hash function Whirlpool in Section 4. We start by discussing classical attacks on reduced variants of the Whirlpool hash function in Section 5. Next we discuss classical attacks on reduced variants of the Whirlpool compression function in Section 6. Finally, we give subspace distinguishers for the compression function in Section 7, and for the underlying block cipher W in Section 8. We conclude in Section 9.

2 The Rebound Attack

The rebound attack was proposed by Mendel et al. in [31] for the cryptanalysis of AES-based hash functions. It is a differential attack, using several new techniques to improve upon existing results.

2.1 Differential Cryptanalysis of Block Ciphers

Differential cryptanalysis is a general tool in the cryptanalysis of symmetric primitives. Originally devised to cryptanalyze DES [4], it has later been applied to other block ciphers, stream ciphers and hash functions. A differential attack exploits predictable propagation of the difference between a *pair* of inputs of a cryptographic primitive, to the corresponding outputs. The description of the difference patterns at the input, the intermediate values and the output of the cryptographic primitive, is called a *characteristic*, or sometimes *differential path* or *trail*. A pair that exhibits the differences of the characteristic, is called a *right pair*. The fraction of right pairs over all input pairs, possibly averaged over all keys (when the primitive is keyed), is called the *probability* of the trail.

Truncated differentials were proposed by Knudsen as a tool in block cipher cryptanalysis [22]. While in a standard differential attack, the full difference between two inputs/outputs is considered, in the case of truncated differentials, the differences are only partially determined, e.g. for every byte, one only checks if there is a difference or not. Truncated differentials have been applied by Peyrin [39] in the cryptanalysis of the hash function Grindahl [24].

2.2 Differential Cryptanalysis of Hash Functions

Rijmen and Preneel described differential attacks on hash functions based on a reduced variant of DES, with 15 instead of 16 rounds [40]. Also the attacks on MD4 by Dobbertin [13], on SHA by Chabaud and Joux [8], and on MD4, RIPEMD and SHA-1 by Wang et al. [43,44] are differential attacks. Most recently, Khovratovich et al. analyzed the security of AES-based hash functions with respect to collision resistance [5,21].

If we apply differential cryptanalysis to a hash function, a collision for the hash function corresponds to a right pair for a trail through that hash function, with output difference zero. Similarly, a near-collision corresponds to a right pair for a trail with an output difference of low Hamming weight. It follows that differential cryptanalysis of hash functions is intuitively very similar to differential cryptanalysis of block ciphers. However, there are also important differences between these two cases, as can be observed with the rebound attack in the next section.

In the case of block ciphers, an adversary that wants to find a right pair can usually do little better than simply trying out pairs. The needed effort is proportional to the inverse of the probability of the trail. Since hash functions do not have a secret key, an adversary can do better than that. In principle, an adversary could simply write out the equations that determine whether a pair is a right pair and solve them. In practice, these equations are highly nonlinear and difficult to solve. However, it is often possible to determine some of the message bits, thereby increasing the probability that a random guess for the remaining part of the solution will be correct. Typically, the equations arising from the first steps of the hash function are easier to solve, because they do not yet depend on all message words. These techniques are known in the literature under the name *message modification techniques* [44].

Hence, a (near-)collision attack on a hash function, that is based on differential cryptanalysis, can be described as follows.

1. Find a trail with a high probability.
2. Determine some message bits by applying message modification techniques.
3. Find the remaining message bits by guess-and-verify.

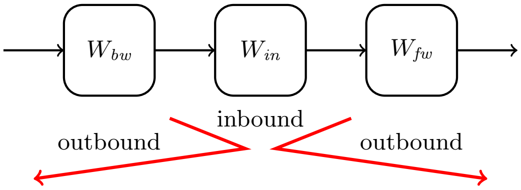

Fig. 1. A schematic view of the rebound attack. The attack consists of an inbound and two outbound phases.

2.3 The Rebound Attack

The rebound attack consists of two phases, called inbound and outbound phase, as shown in Figure 1. According to these phases, the compression function, internal block cipher or permutation of a hash function is split into three subparts. Let W be a block cipher, then we get $W = W_{fw} \circ W_{in} \circ W_{bw}$. Hence, the part of the inbound phase is placed in the middle of the cipher and the two parts of the outbound phase are placed next to the inbound part. In the outbound phase, two high-probability (truncated) differential trails are constructed, which are then connected in the inbound phase. Similar to message modification, the freedom in the message, key-inputs or (internal) state variables is used to efficiently fulfill many conditions of a differential trail.

The idea of placing the most expensive part of the differential trail in the middle was previously used in the cryptanalysis of the compression function of MD5 [12] and the hash function Tiger [20,30,33]. Also, *inside-out* techniques as used in the rebound attack, were invented by Wagner as an application of second order differentials in the cryptanalysis of block ciphers in the Boomerang attack [42].

Constructing a Trail. As in all differential attacks we first need to construct a "good" (truncated) differential trail. A good trail used for a rebound attack should have a high probability in the outbound phases and can have a rather low probability in the inbound phase. Two properties are important here: First, the system of equations that determine whether a pair follows the differential trail in the inbound phase, should be under-determined. Then, many solutions (starting points for the outbound phase) can be found efficiently by using guess-and-determine strategies. Second, the outbound phases need to have high probability in the outward direction.

Inbound Phase. The inbound part of a trail is defined such that the corresponding system of equations is under-determined. When searching for solutions, we first guess some variables such that the remaining system is easier to solve. Hence, the inbound phase of the attack is similar to message modification in an attack on the hash function. The available freedom in terms of the actual values of the internal variables is used to find a solution deterministically or with a high probability. Hence, also a differential trail with a high Hamming weight (and hence a low probability) can be used in the inbound phase.

Outbound Phase. In the outbound phase, we verify whether the solutions of the inbound phase also follow the differential trail in the outbound parts. Note that in the outbound phase, there are usually only a few or no free variables left. Hence, a solution of the inbound phase will lead to a solution of the outbound phase with a probability significantly smaller than 1. Therefore, we aim for narrow (truncated) differential trails in the outbound parts, which can be fulfilled with a probability as high as possible (in the outward directions). The advantage of using an inbound phase in the middle and two outbound phases at the beginning and end is that one can construct differential trails with a higher probability in the outbound phase.

Using more Inbound Phases. Sometimes, not all available freedom is used in the rebound attack. This is usually the case if some parts of the (internal) state or the input of the key schedule is not needed to find a solution in the inbound phase [26,28]. In these cases, the attack can often be extended to more rounds by having one or more independent inbound phases and then connect the solutions of the inbound phases. Note that this is usually not a trivial task. However, it is possible in the compression function attacks on Whirlpool using the freedom of the round keys as shown in Section 6.

3 The Subspace Problem

In [37], NIST requires that a good hash function should fulfill several properties. Along with the well known security notions of collision resistance and (second) preimage resistance, NIST also requires that any K-bit hash function specified by taking a fixed subset of the N output bits should possess the same security assertions as the original function. Of course, an attacker can choose the K-bit subset specifically to allow a limited number of precomputed message digests to collide, but once the subset has been chosen, finding additional violations of the above notions should again have the generic complexity.

From a practical application point of view, this requirement makes a lot of sense when we want to guarantee security in cases where the output space of the hash function is reduced by means of a simple truncation. However, instead of simply truncating the hash function output, the application developer might also choose to split the output string in two halves and `xor` them together [18,19]. This method is almost as simple as truncation, but the security requirement on the hash function becomes now that it should be difficult to construct two messages m, m^* such that

$$H(m) \oplus H(m^*) = z, \tag{1}$$

where z can be any vector with two equal halves. Since the output space of a hash function could be reduced by an arbitrary linear compression step L, we could formulate as generalized requirement that for any linear transformation L, it should be hard to find two inputs m, m^* such that

$$L(H(m^*) \oplus H(m)) = 0. \tag{2}$$

Clearly, the adversary should not be able to choose L, because for all m, m^*, it is trivial to find an L satisfying 2. On the other hand, if we require that the adversary can find suitable m, m^* for any arbitrarily selected L, or for a large subset of them, then, it may become too difficult to find an adversary for many intuitively bad hash function designs. In order to get out of this dilemma, we propose to generalize a bit further by defining the following problem.

Subspace Problem 1 (Subspace Problem for One-Way Functions) *When given a one-way function f mapping to $\mathbb{F}_2^N$, try to find t input pairs (a_i, a_i^*) such that the corresponding output differences $f(a_i) \oplus f(a_i^*)$ belong to a subspace $V_{out} \subset \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$ for some $n \leq N$.*

Here $\mathbb{F}_2 = GF(2)$ denotes the finite field of order 2.

If f is a hash or compression function, then solving Subspace Problem 1 should be *hard*, when n is significantly smaller than N, say $n \leq \frac{N}{2}$. Otherwise, the hash function has a certificational weakness. We show in Section 6 how Subspace Problem 1 can be solved when f is the compression function of Whirlpool, but first we discuss what we mean when we state that Subspace Problem 1 should be *hard*.

3.1 On the Hardness of Subspace Problem 1

In this section, we investigate how difficult it is to solve Subspace Problem 1 without using knowledge of the internals of the function f. We measure the difficulty by counting the number of queries that need to be made to the oracle. We bound the *query complexity* and ignore all other computations, memory accesses etc.

Let us now assume that an adversary is making $Q \ll 2^{N/2}$ queries to the function f. We thus get $K \leq \binom{Q}{2}$ differences $(\in \mathbb{F}_2^N)$ coming from these Q queries. For given n and $t > n$, we now want to calculate the probability that among the K corresponding output differences $f(a_i) \oplus f(a_i^*)$, we have t vectors (output differences) that belong to a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$.

We will need the following fact about matrices over finite fields. Let $E(t, N, d)$ denote the number of $t \times N$ matrices over $\mathbb{F}_2$ that have rank equal to d. Then, it is well known [14,27] that

$$E(t, N, d) = \prod_{i=0}^{d-1} \frac{(2^N - 2^i) \cdot (2^t - 2^i)}{2^d - 2^i} = \prod_{i=0}^{d-1} (2^N - 2^i) \cdot \binom{t}{d}_2, \tag{3}$$

where $\binom{t}{d}_2$ denotes the q-binomial coefficient with $q = 2$.

Proposition 1 *Let $n, t, N \in \mathbb{N}$ be given such that $t \geq N > n$. We assume a set of K vectors (output differences) chosen uniformly at random from $\mathbb{F}_2^N$. Let $\Pr(K, t, N, n)$ denote the probability that t of these K vectors span a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$. Then, we have*

$$\Pr(K, t, N, n) \leq \binom{K}{t} 2^{-t \cdot N} \sum_{d=0}^{n} E(t, N, d). \tag{4}$$

This probability is upper bounded by

$$\Pr(K, t, N, n) \leq \frac{1}{\sqrt{2\pi t}} \left(\frac{Ke}{t}\right)^t 2^{-(N-n)(t-n)+(n+1)}. \tag{5}$$

For the proof of Proposition 1, we will first need two lemmas.

Lemma 1. *Let $t, N, n \in \mathbb{N}$ be such that $t \geq N > n$. Then,*

$$E(t, N, n) \leq \sum_{d=0}^{n} E(t, N, d) \leq 2 \cdot E(t, N, n).$$

Proof. The first inequality is trivial. The second one is equivalent to

$$\sum_{d=0}^{n-1} E(t, N, d) \leq E(t, N, n).$$

and can be proven by induction over n. For $n = 1$, $E(t, N, 0) \leq E(t, N, 1)$ which is easily seen to be true. So let us assume that

$$\sum_{d=0}^{n-2} E(t, N, d) \leq E(t, N, n-1)$$

holds. To prove the statement, we add $E(t, N, n-1)$ to both sides. If we can show that $2E(t, N, n-1) \leq E(t, N, n)$, we are done. From (3) we derive

$$2E(t, N, n-1) = 2 \prod_{i=0}^{n-2} (2^N - 2^i) \cdot \binom{t}{n-1}_2,$$
$$E(t, N, n) = \prod_{i=0}^{n-1} (2^N - 2^i) \cdot \binom{t}{n}_2.$$

Since $t \geq N > n$, we have

$$2\binom{t}{n-1}_2 \leq (2^N - 2^{n-1})\binom{t}{n}_2.$$

The proof follows from the fact that

$$\binom{t}{n}_2 = \frac{2^{t-n+1} - 1}{2^n - 1}\binom{t}{n-1}_2.$$

□

Lemma 2. *Let $t, N, n \in \mathbb{N}$ be such that $t \geq N > n$. Then,*

$$\frac{(2^t - 2^i) \cdot (2^N - 2^i)}{2^n - 2^i} \leq \frac{(2^t - 2^j) \cdot (2^N - 2^j)}{2^n - 2^j}$$

holds for all $0 \leq i < j \leq n - 1$.

Proof. We show this by proving that for given $A > B > C > 0$ the function

$$f(x) = \frac{(A - x)(B - x)}{C - x}$$

has always a positive derivative $f'(x)$ on the interval $x \in [0, C/2]$. Elementary calculus shows that the derivative of $f(x)$ is

$$f'(x) = \frac{(A - C)(B - C)}{(C - x)^2} - 1,$$

from which we easily see that the condition $f'(x) > 0$ is satisfied if

$$(A - C)(B - C) > (C - x)^2$$

holds. The right side is smaller than C^2 which means that the statement is equal to

$$AB > C(A + B)$$

If we substitute $A = 2^t, B = 2^N, C = 2^n$ we see that the last inequality holds in our setting and we are done. □

Now, we are in the position to prove Proposition 1.

Proof (of Proposition 1). Remember that $E(t, N, d)$ was defined as the number of $t \times N$ matrices over $\mathbb{F}_2$ that have rank equal to d. Computing $\Pr(K, t, N, n)$ exactly would require the application of the inclusion-exclusion principle since the ranks of the $\binom{K}{t}$ considered subspaces are not independent. Therefore, we take (4) as an upper bound for the probability $\Pr(K, t, N, n)$.

Simplifying the upper bound consists of two steps. Bounding the binomial coefficient and bounding the rest. Based on Lemma 1 and 2 we can estimate the second part of the probability $\Pr(K, t, N, n)$ by

$$\begin{aligned} 2^{-t \cdot N} \sum_{d=1}^{n} E(t, N, d) &\leq 2^{-t \cdot N} \cdot 2 \cdot E(t, N, n) \\ &\leq 2^{-t \cdot N + 1} \left(\frac{(2^t - 2^{n-1}) \cdot (2^N - 2^{n-1})}{2^n - 2^{n-1}} \right)^n \qquad (6) \\ &\leq 2^{-t \cdot N + 1} \left(2^{n-1} \cdot 2^{t-(n-1)} \cdot 2^{N-(n-1)} \right)^n \\ &= 2^{-(t-n)(N-n)+(n+1)} \end{aligned}$$

For the binomial coefficient $\binom{K}{t}$ we combine the simple estimate $\binom{K}{t} \leq K^t/t!$ with the following inequality based on Stirling's formula [41]:

$$\sqrt{2\pi} t^{t+\frac{1}{2}} e^{-t+\frac{1}{12t+1}} < t! < \sqrt{2\pi} t^{t+\frac{1}{2}} e^{-t+\frac{1}{12t}} \tag{7}$$

From this we get

$$\binom{K}{t} \leq \frac{1}{\sqrt{2\pi t}} \left(\frac{K \cdot e}{t}\right)^t. \tag{8}$$

Putting together (6) and (8) proves the proposition. □

As a corollary, we can give a lower bound for the number of random vectors needed to fulfill the conditions of the proposition with a certain probability.

Corollary 1 *For a given probability p and given N, n, t as in Proposition 1, the number K of random vectors needed to contain t vectors that span a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$ with probability p is lower bounded by*

$$K \geq \frac{t}{e} \left(p\sqrt{2\pi t}\right)^{\frac{1}{t}} 2^{\frac{(N-n)(t-n)-(n+1)}{t}}. \tag{9}$$

Proof. Equation (9) follows immediately from (5). □

Corollary 2 *For a given probability p and given N, n, t as in Proposition 1, the and the number of queries Q to f needed to produce t vectors that span a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$ with probability p is lower bounded by*

$$Q \geq \sqrt{\frac{2t}{e}} \left(p\sqrt{2\pi t}\right)^{\frac{1}{2t}} 2^{\frac{(N-n)(t-n)-(n+1)}{2t}}. \tag{10}$$

Proof. (10) follows from setting $K \leq \binom{Q}{2} = Q(Q-1)/2$ in (9). □

3.2 The Permutation Case

This section is devoted to the study of the Subspace Problem in the case where the function f is replaced by a permutation π. In the case of a permutation, one can define adversaries that are allowed to make forward queries (i.e. to π) and backward queries (i.e. to π^{-1}). Clearly, backward queries render Subspace Problem 1 trivial, since the adversary can fix pairs with output differences in V_{out} and simply ask the backward queries. Therefore, if we want to define a meaningful subspace problem, we have to formulate additionally constraints on the inputs.

Subspace Problem 2 (Subspace Problem for Permutations)
When given a permutation π mapping from $\mathbb{F}_2^N$ to $\mathbb{F}_2^N$, try to find t input pairs (a_i, a_i^*) such that $a_i \oplus a_i^*$ belong to a subspace $V_{in} \subseteq \mathbb{F}_2^N$ with $\dim(V_{in}) \leq m$ and the corresponding output differences $\pi(a_i) \oplus \pi(a_i^*)$ belong to a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$ for some $m \leq M$ and $n \leq N$.

One can distinguish between two types of adversaries: the non-adaptive adversary and the adaptive adversary. While in the adaptive setting, the adversary can use the results of previous queries to select subsequent inputs for the next queries, the adversary has to decide on the inputs for the queries on beforehand in the non-adaptive setting. In all of the following, we will only consider non-adaptive adversaries.

Now we want to give a bound on the success probability of the adversary for solving Subspace Problem 2 when it is given oracle access to a permutation π and its inverse π^{-1}. The adversary is allowed to make at most Q queries in total to π and π^{-1}. Denote by Q_1 the number of queries that the adversary makes to π, and by $Q_2 = Q - Q_1$ the number of queries made to π^{-1}.

Let us now start with the Q_1 queries to π. It is easy to see that by choosing the inputs in a sub-vector space of dimension m we get $K_1 \leq \binom{Q_1}{2}$ input pairs (a_i, a_i^*) and hence input differences $a_i \oplus a_i^*$ belonging to a subspace $V_{in} \subseteq \mathbb{F}_2^N$ with $\dim(V_{in}) \leq m$. This approach obviously allows a maximum of 2^m queries.

In order to be able to make more queries we take the subsequent inputs to be in a translate of V_{in}, that is, we take $a_i, a_i^* \in u + V_{in} = \{u + v \,|\, v \in V_{in}\}$ where $u \notin V_{in}$. We can repeat this several times for different $u \notin V_{in}$. So if we set $Q_1 = q_1 \cdot 2^m + r_1$ and $r_1 < 2^m$, $q_1 \geq 0$, by making Q_1 queries to π we get

$$K_1 = q_1 \cdot \binom{2^m}{2} + \binom{r_1}{2} \tag{11}$$

input differences $a_i \oplus a_i^*$ belonging to a vector space V_{in} with $\dim(V_{in}) \leq m$.

Analogously to Proposition 1 we will first consider the case of differences. Note that the Q_1 queries to π are chosen such that the resulting K_1 input differences lie in a subspace V_{in} whereas the corresponding output differences can be assumed uniformly distributed in $\mathbb{F}_2^N$. In a similar way, the Q_2 queries to π^{-1} result in K_2 output differences in a space V_{out} where again the corresponding input differences are uniformly distributed. So in total we have $K_1 + K_2$ pairs of input and output differences.

Proposition 2 *Let $n, m, t, N \in \mathbb{N}$ be given such that $t \geq N > 2n$ and $m \leq n$. We assume a set of $K := K_1 + K_2$ difference pairs $\{(a_1, b_1), \ldots, (a_K, b_K)\}$ where b_i is uniformly distributed in $\mathbb{F}_2^N$ and a_i is taken from some subspace $V_{in} \subseteq \mathbb{F}_2^N$ for $i = 1, \ldots, K_1$ and where a_i is uniformly distributed in $\mathbb{F}_2^N$ and b_i is taken from some subspace $V_{out} \subseteq \mathbb{F}_2^N$ for $i = 1, \ldots, K_2$.*

Let $\Pr(K, t, N, m, n)$ denote the probability that t of these K difference pairs are such that the input differences span a subspace $V'_{in} \subseteq \mathbb{F}_2^N$ with $\dim(V'_{in}) \leq m$ and the output differences span a subspace $V'_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V'_{out}) \leq n$, simultaneously. Then, we have

$$\Pr(K, t, N, m, n) \leq \sum_{t_1=0}^{t} \binom{K_1}{t_1} \binom{K_2}{t - t_1} 2^{-t \cdot N} \sum_{i=0}^{m} E(t - t_1, N, i) \sum_{j=0}^{n} E(t_1, N, j). \tag{12}$$

This probability can be upper bounded by

$$\Pr(K, t, N, m, n) \leq \frac{1}{\sqrt{2\pi t}} \left(\frac{Ke}{t}\right)^t 2^{-(N-n)(t-2n)+2(n+1)}. \tag{13}$$

Proof. The $K = K_1 + K_2$ difference pairs described in the proposition can be seen as elements $(a_i, b_i) \in \mathbb{F}_2^N \times \mathbb{F}_2^N$, where in the first K_1 pairs, the a_i's can be chosen, and in the last K_2 the b_i's can be chosen by an adversary. In order to have the highest possible probability for the event in the proposition these values would always be chosen to be a fixed difference $a \neq 0$ and $b \neq 0$. The 0 difference is impossible when keeping in mind that they come from queries, so choosing identical differences leads to the smallest dimension for the difference vectors that can be controlled. So whenever t of the K difference pairs are selected, and say t_1 are taken from the first K_1 pairs, and $t - t_1$ from the second K_2 pairs, we can start to upper bound the sought probability by (12). This is because the probability that t of these input differences span a space of dimension $\leq m$ is upper bounded by

$$2^{-(t-t_1)N} \sum_{i=0}^{m} E(t - t_1, N, i). \tag{14}$$

Here we use that t_1 input differences are identical and we apply Proposition 1 to the remaining $t - t_1$ input differences, where we count the $\mathbb{F}_2$-matrices of rank $\leq m$. The sum in (14) is an overestimation since when the fixed input difference a is not in the span of the remaining $t - t_1$ differences, we would only be allowed to take the matrices of rank $\leq m - 1$ into account. Analogously, we get for the output differences

$$2^{-t_1 N} \sum_{i=0}^{n} E(t_1, N, i),$$

and since both conditions have to be satisfied simultaneously, we end up with (12).

To further bound (12) we proceed as follows. Without loss of generality, we assume that $m \leq n$ and obtain

$$\sum_{t_1=0}^{t} \binom{K_1}{t_1} \binom{K_2}{t - t_1} 2^{-tN} \sum_{i=0}^{n} E(t - t_1, N, i) \sum_{j=0}^{n} E(t_1, N, j)$$

as an upper bound for the probability. Using Lemma 3 we can simplify the last sum to

$$\binom{K}{t} 2^{-tN+2} E(\tfrac{t}{2}, N, n)^2$$

where we used

$$\sum_{t_1=0}^{t} \binom{K_1}{t_1} \binom{K_2}{t - t_1} = \binom{K_1 + K_2}{t}.$$

Then, we can prove (13) along the same lines as in (6). □

Lemma 3. *Let $t, N, n \in \mathbb{N}$ be such that $t \geq N > 2n$ and $t_1 \in \{0, 1, \dots, t\}$. Then,*

$$\sum_{i=0}^{n} E(t - t_1, N, i) \sum_{j=0}^{n} E(t_1, N, j) \leq 4E(\tfrac{t}{2}, N, n)^2. \tag{15}$$

Proof. We first consider the case where $t_1 \in \{n, n+1, \dots, t-n\}$. This implies that both t_1 and $t - t_1$ are greater or equal than n. In this case, we can use Lemma 1 to estimate both sums and we get

$$\sum_{i=0}^{n} E(t - t_1, N, i) \sum_{j=0}^{n} E(t_1, N, j) \leq 4E(t - t_1, N, n)E(t_1, N, n).$$

The product $E(t - t_1, N, n)E(t_1, N, n)$ can be written as

$$\prod_{i=0}^{n-1} \frac{(2^N - 2^i)^2 \cdot (2^{t-t_1} - 2^i) \cdot (2^{t_1} - 2^i)}{(2^n - 2^i)^2}.$$

From this and the fact that $(2^{t-t_1} - 2^i) \cdot (2^{t_1} - 2^i) \leq (2^{t/2} - 2^i)^2$ holds for $i \in \{0, \dots, n-1\}$ follows the statement of the lemma for $t_1 \in \{n, n+1, \dots, t-n\}$.

The case $t_1 \in \{0, 1, \dots, n-1\}$, respectively $t_1 \in \{t-n+1, \dots, t\}$, is symmetric, so without loss of generality, we only consider the first case. Then, the estimate of Lemma 1 applied to (15) results in

$$\sum_{i=0}^{n} E(t - t_1, N, i) \sum_{j=0}^{t_1} E(t_1, N, j) \leq 4E(t - t_1, N, n)E(t_1, N, t_1).$$

We can show

$$E(t - t_1, N, n)E(t_1, N, t_1) \leq E(\tfrac{t}{2}, N, n)^2$$

by splitting the statement into two inequalities:

$$E(t - t_1, N, n)E(t_1, N, t_1) \leq E(t - n, N, n)E(n, N, n) \tag{16}$$

$$E(t - n, N, n)E(n, N, n) \leq E(\tfrac{t}{2}, N, n)^2 \tag{17}$$

Here, (16) can be deduced with similar arguments as Lemma 2. To show (17) we look at $E(t - n, N, n)E(n, N, n)E(t/2, N, n)^{-2}$ and observe that

$$\prod_{i=0}^{n-1} \frac{(2^{t-n} - 2^i)(2^n - 2^i)}{(2^{t/2} - 2^i)^2} = \prod_{i=0}^{n-1} \frac{2^t - 2^{t-n+i} - 2^{n+i} + 2^{2i}}{2^t - 2^{t/2+i+1} + 2^{2i}} \leq 1,$$

since because of $t > 2n$, every term in the product is smaller or equal than 1. This proves the lemma in the case $t_1 \in \{0, 1, \dots, n-1\}$, respectively $t_1 \in \{t - n + 1, \dots, t\}$. □

Now we round up the whole discussion to derive some lower bounds for the number of differences and the query complexity.

Corollary 3 *Under the preliminaries stated in Proposition 2, the number K of difference pairs such that simultaneously, the input differences span a subspace $V'_{in} \subseteq \mathbb{F}_2^N$ with $\dim(V'_{in}) \leq m$ and the output differences span a subspace $V'_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V'_{out}) \leq n$ with probability p is lower bounded by*

$$K \geq \frac{t}{e}\left(p\sqrt{2\pi t}\right)^{\frac{1}{t}} 2^{\frac{(N-n)(t-2n)-2(n+1)}{t}}. \tag{18}$$

Proof. Equation (18) follows immediately from (13). □

Corollary 4 *Under the preliminaries stated in Proposition 2, let Q be the number of queries to π and π^{-1} needed to find t difference pairs such that simultaneously, the input differences span a subspace $V'_{in} \subseteq \mathbb{F}_2^N$ with $\dim(V'_{in}) \leq m$ and the output differences span a subspace $V'_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V'_{out}) \leq n$ with probability p. Let*

$$\hat{K} = \frac{t}{e}\left(p\sqrt{2\pi t}\right)^{\frac{1}{t}} 2^{\frac{(N-n)(t-2n)-2(n+1)}{t}}.$$

Then, Q is lower bounded by

$$Q \geq \begin{cases} \sqrt{2\hat{K}} & \text{if } \hat{K} < 2^{2n-1}, \\ \hat{K}2^{-n} & \text{if } \hat{K} \geq 2^{2n-1}. \end{cases} \tag{19}$$

Proof. We see that (11) suggests that an adversary would favor to take the dimension of the space V_{in}, respectively, V_{out}, as large as possible (that is, m, respectively n) in order to produce as many differences as possible from a given number of queries. Equation (11) gives rise to the easy estimates $K_1 \leq 2^{m-1}Q_1$ and $K_2 \leq 2^{n-1}Q_2$. Together with $m \leq n$, we use (18) to end up with (19) depending on the size of $\hat{K}$. Note that this rough bound combines the best possible cases for an adversary in terms of differences (by using (18)) and in terms of queries. □

Looking back at Corollary 2, we see that in the case of one-way functions the connection between differences and queries was much more obvious than it is here. This is caused by the fact that there we had only one type of queries. In the permutation case, we saw that the strategy of choosing differences/queries on both sides of π lead to a higher bound for the success probability of an adversary. This can be seen as evidence for preferring this strategy over the one-sided approach.

3.3 Related Work

Biryukov et al. introduce *differential q-collisions* as a means to construct distinguishers for a block cipher [5]. In our terminology, a differential q-collision corresponds to a set of q input pairs that have input differences in an affine subspace $v_i + \{0\} \subseteq \mathbb{F}_2^N$ of dimension 0 ($v_i \neq 0$) and output differences in an affine subspace $v_o + \{0\} \subseteq \mathbb{F}_2^N$ of dimension 0. An important difference with our approach is that they allow the adversary to specify the input difference and the output difference such that they optimally fit the block cipher under attack. Since we characterize subspaces only by their dimension, we impose less constraints on the adversary. The consequence is that for the same distinguisher, they compute a higher advantage than we do.

Also Gilbert and Peyrin discuss distinguishers for AES, Grøstl [15] and ECHO [3] in [17], which have some similarities with the subspace distinguishers. Note however that, like Biryukov et al., they allow the adversary to specify which of the coordinates have to be constant. Secondly, [17] ignores the invertible linear transformation in the last round of Grøstl and ECHO. We note also that [17] upper bounds the attack complexity for the generic case, while a lower bound is needed in order to prove that the distinguishers given for AES, Grøstl and ECHO are indeed valid distinguishers. Finally, [17] defines new families of AES-like constructions by considering keyed linear or non-linear building blocks, e.g. keyed S-boxes. Since neither AES, nor Grøstl uses keyed S-boxes or other similar randomization techniques, this construction can be seen as somewhat counter-intuitive.

4 The Hash Function Whirlpool

The Whirlpool hash function is a cryptographic hash function designed by Barreto and Rijmen in 2000 [2]. It has been evaluated and approved by NESSIE [38] and is standardized by ISO/IEC [1]. The hash function is commonly considered to be a conservative block cipher based design with a very conservative key schedule. The design follows the wide trail design strategy. In this section, we will give a detailed account of the Whirlpool hash function. It includes a discussion of its core design principle, the wide trail design strategy, and the properties of the employed round transformations with respect to differential and truncated differential cryptanalysis.

4.1 The Wide Trail Design Strategy

The wide trail design strategy has been proposed by Daemen and Rijmen in [9,10] and is a method to counter differential (and linear) attacks. The strategy allows to easily construct upper bounds for the probability of trails through the primitive. To obtain these bounds, we split up a design in a linear and a nonlinear part, each with its own functionality.

We assume here that the nonlinear part is implemented by means of a *bricklayer* of S-boxes [10]. The S-boxes $\mathcal{S}_i$ are selected such that for any differential $(a, b) \neq (0, 0)$, the fraction of inputs x for which

$$\mathcal{S}_i(x) \oplus \mathcal{S}_i(x \oplus a) = b,$$

is small. Let $p_{\mathcal{S}}$ denote an upper bound for this fraction.

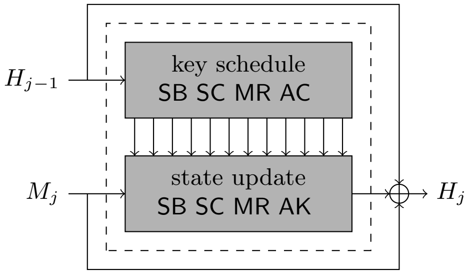

Fig. 2. An overview of the Whirlpool compression function. The 10-round block cipher W with key schedule and state update is used in Miyaguchi-Preneel mode.

The functionality of the linear part of the primitive is to make sure that there are no *narrow* trails, i.e. trails where only a small number of S-boxes has a non-zero input difference. An S-box with a non-zero input difference is called *active*. Let z denote a lower bound for the number of active S-boxes in a trail. Then it follows easily that $(p_{\mathcal{S}})^z$ upper bounds the probability of a trail.

4.2 Whirlpool

Whirlpool is an iterative hash function based on the Merkle-Damgård design principle [11,35]. It processes 512-bit message blocks and produces a 512-bit hash value. An unambiguous padding method is applied to ensure that the message length is a multiple of 512 bits [2]. Let $m = M_1 \| M_2 \| \cdots \| M_t$ be a t-block message (after padding). The hash value $h = H(m)$ is computed as follows (see Figure 2):

$$\begin{aligned} H_0 &= IV, \\ H_j &= W(H_{j-1}, M_j) \oplus H_{j-1} \oplus M_j, \quad \text{for } 0 < j \le t, \\ h &= H_t, \end{aligned}$$

where IV is a predefined initial value and W is a 512 bit block cipher used in the Miyaguchi-Preneel mode [34].

4.3 The Block Cipher *W*

The block cipher W is designed according to the wide trail strategy and its structure is very similar to the Advanced Encryption Standard (AES) [36]. The state update transformation and the key schedule update an 8×8 state S, respectively K, of 64 bytes in 10 rounds. In one round, the round transformation updates the state by means of the sequence of transformations

$$\mathsf{AK} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB},$$

while the key schedule applies

$$\mathsf{AC} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB}$$

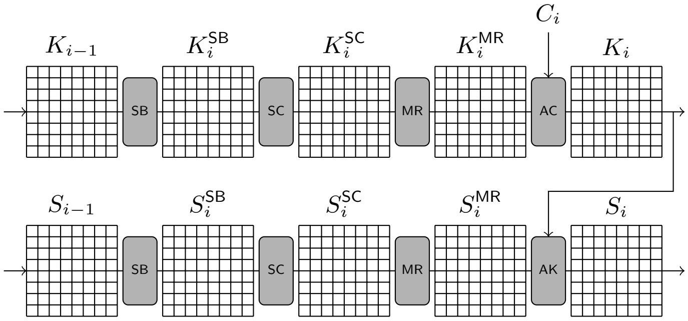

Fig. 3. One round of the block cipher W, used in the Whirlpool compression function.

to the round key. In the remainder of this paper, we will use the outline of Figure 3 for one round. We denote the resulting state after round i by S_i and the intermediate states after SubBytes (SB) by S_i^{SB}, after ShiftColumns (SC) by S_i^{SC} and after MixRows (MR) by S_i^{MR}. The initial state prior to the first round is denoted by $S_0 = M_j \oplus H_{j-1}$. The same notation is used for the key schedule with round keys K_i with $K_0 = H_{j-1}$. Note that we changed the names of some steps of the round transformation of the original description [2] in order to be more similar to the AES nomenclature [10].

4.4 The Round Transformations of *W*

In the following, we briefly describe the round transformations of the block cipher W used in the Whirlpool compression function.

SubBytes (SB). The SubBytes step is the only non-linear transformation of the cipher. It is a permutation consisting of an S-box applied to each byte of the state. The 8-bit S-box is composed of 3 smaller 4-bit mini-boxes (the exponential E-box, its inverse, and the pseudo-randomly generated R-box). For a detailed description of the S-box we refer to [2].

ShiftColumns (SC). The ShiftColumns step is a byte transposition that cyclically shifts the columns of the state over different offsets. Column j is shifted downwards by j positions.

MixRows (MR). The MixRows step is a permutation operating on the state row by row. To be more precise, it is a right-multiplication by an 8×8 matrix over $\mathbb{F}_{2^8}$. The coefficients of the matrix are determined in such a way that the *branch number* of MixRows (the smallest nonzero sum of active input and output bytes of each row) is 9, which is the maximum possible for a transformation with these dimensions.

Table 1. The number of differentials and possible pairs (a, b) for the Whirlpool S-box. The first row shows the number of *impossible differentials* and the last row corresponds to the zero differential.

solutions	frequency
0	39655
2	20018
4	5043
6	740
8	79
256	1

AddRoundKey (AK) and AddRoundConstant (AC). The key addition in the state update transformation is denoted by AddRoundKey and in the key schedule by AddRoundConstant, respectively. In this transformation the state is modified by combining it with a round key with a bitwise `xor` operation. While the round key in the state update transformation is generated by the key schedule, it is a predefined constant in the key schedule.

4.5 Differential Properties of Round Transformations

In this section, we describe the differential properties of the round transformations of Whirlpool.

SubBytes (SB). SubBytes has the following differential properties. Let $a, b \in \{0,1\}^8$. Exhaustively counting over all 2^{16} differentials shows that the number of solutions to the following equation

$$\mathcal{S}_i(x) \oplus \mathcal{S}_i(x \oplus a) = b, \tag{20}$$

can only be 0, 2, 4, 6, 8 and 256, which occur with frequency 39655, 20018, 5043, 740, 79 and 1, see Table 1. The task to return all solutions x to (20) for a given differential (a, b) is best solved by setting up a precomputed table of size 256×256 which stores the solutions (if there are any) for each (a, b).

However, it is easy to see that for any permutation $\mathcal{S}_i$ (to be more precise, for any injective map) the expected number of solutions to (20) is always one:

$$2^{-16} \sum_a \sum_b \#\{x \,|\, \mathcal{S}_i(x \oplus a) \oplus \mathcal{S}_i(x) = b\} = 2^{-16} \sum_a 2^8 = 1,$$

because for a fixed a, every solution x belongs to a unique b. Since all the S-boxes are independent, the same reasoning is valid for the full SubBytes transformation.

ShiftColumns (SC). The ShiftColumns transformation moves bytes and thus, differences to different positions of a column but does not change their value. Due to the good diffusion property of ShiftColumns, 8 active bytes of a full active

Table 2. Approximate probabilities (as base 2 logarithms) for the propagation of truncated differences through MixRows with predefined positions. a denotes the number of active bytes at the input and b the number of active bytes at the output of MixRows.

$a \setminus b$	0	1	2	3	4	5	6	7	8
0	0	×	×	×	×	×	×	×	×
1	×	×	×	×	×	×	×	×	0
2	×	×	×	×	×	×	×	-8	-0.0017
3	×	×	×	×	×	×	-16	-8	-0.0017
4	×	×	×	×	×	-24	-16	-8	-0.0017
5	×	×	×	×	-32	-24	-16	-8	-0.0017
6	×	×	×	-40	-32	-24	-16	-8	-0.0017
7	×	×	-48	-40	-32	-24	-16	-8	-0.0017
8	×	-56	-48	-40	-32	-24	-16	-8	-0.0017

row are moved to 8 different rows of the state. Hence, ShiftColumns ensures that the 8 bytes of one row of a state are processed independently in the subsequent MixRows transformation.

MixRows (MR). Since the MixRows operation is a linear transformation, standard differences propagate through MixRows in a deterministic way. The propagation only depends on the values of the differences and is independent of the actual value of the state. In case of truncated differences only the position, but not the value of the difference is determined. Therefore, the propagation of truncated differences through MixRows is probabilistic.

Since the branch number of MixRows is 9, a truncated difference with exactly one active byte will propagate to a truncated difference with 8 active bytes with a probability of 1. On the other hand, a truncated difference with 8 active bytes can result in a truncated difference with 1 to 8 active bytes after MixRows. The probability of an 8 to 1 transition is only $2^{-7 \cdot 8} = 2^{-56}$, since we need 7 out of 8 truncated differences to be zero. In general, the probability of any a to b transition with $1 \leq a, b \leq 8$ satisfying $a + b \geq 9$ is approximately $2^{(b-8) \cdot 8}$. Note that the probability depends on the *direction* of the propagation of truncated differences, see Table 2.

AddRoundKey (AK) and AddRoundConstant (AC). Since AddRoundKey and AddRoundConstant are simple `xor` operations with a round key or a constant. Therefore, both standard differences and truncated differences propagate through AddRoundKey and AddRoundConstant in a deterministic way.

4.6 Good Differential Trails

Due to the design of the Whirlpool hash function, constructing good truncated differential trails is rather simple, as long as there are no differences inserted from the key schedule. Therefore, we restrict ourselves to trails with no differences in the key schedule and hence chaining value of Whirlpool. This allows us to

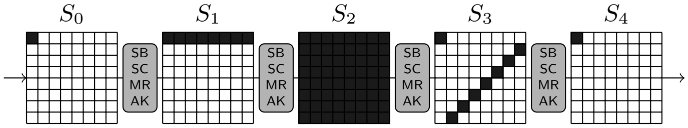

Fig. 4. A 4-round differential trail with the minimum number (81) of active S-boxes.

construct good differential trails by hand as shown in this section. We will use the following notation to specify the number of active bytes in two subsequent states in the state update:

$$a \xrightarrow{r_i} b,$$

with a the number of active bytes in the first state, b the number of active bytes in the second state and r_i the i-th round of Whirlpool. As an example, for one round r_i of Whirlpool, we either get $a + b \geq 9$ or $a = b = 0$, due to the design of the MixRows transformation. Hence, for $a = 1$ we always get:

$$1 \xrightarrow{r_i} 8.$$

It follows from the properties of the ShiftColumns and MixRows transformations, that any 4-round (truncated) differential trail has at least $9^2 = 81$ active S-boxes. Hence, $(p_{\mathcal{S}})^z = (2^{-5})^{81}$ upper bounds the probability of any 4-round differential trail (see Section 4.1). An example differential trail with 81 active S-boxes is given in Figure 4. Note that the active byte in state S_0 and state S_4 can be placed at any position (state S_1 and S_3 change accordingly). The number of active S-boxes in each state for these trails are as follows:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8 \xrightarrow{r_4} 1$$

This 4-round trail will be used to explain the principles of the rebound attack in Section 5.1. Note that this trail can be extended in a simple and straightforward way in the forward and in the backward direction. We will use the following trail to show a near-collision attack for the Whirlpool hash function in Section 5.4:

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 8 \xrightarrow{r_5} 1 \xrightarrow{r_6} 8$$

Another possibility is to extend the trail by adding rounds in the middle. If we add a second full active state in the middle, then we still get a valid trail. This trail will be used to extend the rebound attacks on the hash function by one round (see Section 5.3 and 5.4):

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 64 \xrightarrow{r_4} 8 \xrightarrow{r_5} 1$$

Moreover, two full active states allow us to place one or two states with 8 active bytes in between them, such that all properties of the round transformations are still fulfilled:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8 \xrightarrow{r_4} 8 \xrightarrow{r_5} 64 \xrightarrow{r_6} 8 \xrightarrow{r_7} 1$$

This trail will be used as the core for the compression function attacks on Whirlpool in Section 6.

5 Attacks on the Hash Function

In this section, we describe the application of the rebound attack to reduced variants of the Whirlpool hash function. First, we describe the basic idea of the attack for Whirlpool reduced to 4.5 rounds. By improving the inbound phase of the attack, the complexity can be significantly reduced to about 2^{64} compression function evaluations and negligible memory requirements. Furthermore, we show how the attack can be extended to 5.5 rounds by adding another full active state in the inbound phase. The resulting attack has a complexity of about 2^{120} and memory requirements of 2^{64}.

Second, we present near-collision attacks for the Whirlpool hash function reduced to 6.5 and 7.5 rounds. These attacks are straight forward extensions of the collision attacks on 4.5 and 5.5 rounds, respectively. By adding 2 rounds in the outbound phase, we get a near-collision for the Whirlpool hash function reduced to 6.5 and 7.5 rounds.

5.1 Collision Attack on 4.5 Rounds

The rebound attack on 4.5 rounds of Whirlpool uses a differential trail with the minimum number of active S-boxes according to the wide trail design strategy. For this attack, the full active state is placed in the middle of the trail (see Figure 5):

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8 \xrightarrow{r_4} 1 \xrightarrow{r_{4.5}} 1$$

To find a message pair following this 4.5-round differential trail, we first split the block cipher W into three sub-ciphers $W = W_{fw} \circ W_{in} \circ W_{bw}$, such that the full active state of the differential trail is covered by the inbound phase W_{in}.

$$\begin{aligned} W_{bw} &= \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \\ W_{in} &= \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \\ W_{fw} &= \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \end{aligned}$$

In the inbound phase, the actual values of the state are chosen to guarantee that the differential trail in W_{in} holds. The differential trail in the outbound phase (W_{fw}, W_{bw}) is supposed to have a relatively high probability. While standard `xor` differences are used in the inbound phase, truncated differentials are used in the outbound phase of the attack. In the following, we describe the inbound and outbound phase of the attack in detail.

Inbound Phase. In the inbound phase of the attack we have to find inputs to W_{in} such that the differential trail in W_{in} holds. It can be summarized as follows (see Figure 6).

1. We start at the output of MixRows of round r_3 (S_3^{MR}) with arbitrary nonzero differences at the 8 byte positions indicated on Figure 6. We propagate the difference backward. Since we have one active byte in each row of the state, we obtain a full active state at the output of SubBytes of round r_3 (S_3^{SB}).

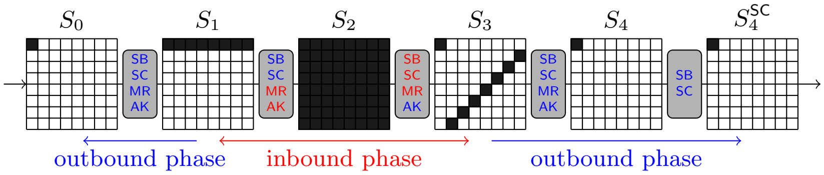

Fig. 5. Differential trail for the collision attack on 4.5 rounds of Whirlpool. Black state bytes are active.

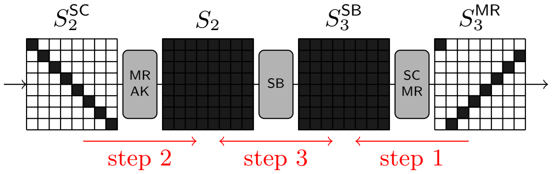

Fig. 6. Inbound phase of the attack on 4.5 rounds of Whirlpool. Black state bytes are active.

2. We choose a difference for the active byte in each row at the input of MixRows in round r_2 (S_2^{SC}) and compute forward to the input of SubBytes of round r_3 (S_2). Note that this can be done for all 255 ($\sim 2^8$) values (nonzero difference) of the active byte for each row independently, which facilitates the attack.
3. In the next step of the inbound phase, the *match-in-the-middle step*, we look for a matching input/output difference of the SubBytes layer of round r_3. This is done as described in Section 4.4 with a precomputed 256×256 S-box lookup table. As explained in Section 4.4, the expected number of solutions is one per trial. Note that we can search for S-box matches for each row of S_2 and S_3^{SB} independently. Since we have 2^8 candidates for each row of S_2 (and 1 for each row of S_3^{SB}) the expected number of solutions for each row is 2^8 (i.e. 2 solutions for each S-box). Hence, the expected number of solutions for the whole SubBytes layer (8 rows) equals 2^{64}. In other words, we can find 2^{64} actual values that follow the differential trail in the inbound phase with a complexity of about 2^8 round transformations.

Since we can repeat these 3 steps 2^{64} times, we can find 2^{128} actual values that follow the differential trail in the inbound phase.

Outbound Phase. In contrast to the inbound phase, we use truncated differentials in the outbound phase of the attack. By propagating the matching differences and state values through the next SubBytes layers outwards, we get a truncated differential in 8 active bytes in both backward and forward direction.

In order to get a collision after 4.5 rounds we require that the truncated differentials in the outbound phase propagate from 8 to 1 active byte through the MixRows transformation, both in the backward and forward direction (see Figure 5). The propagation of truncated differentials through the MixRows transformation can be modeled in a probabilistic way, see Section 4.4. Since we need to fulfill one 8 to 1 transition in the backward and forward direction, the probability of this part of the outbound phase is $2^{-2\cdot 56} = 2^{-112}$. Furthermore, to construct a collision at the output (after the feed-forward), we need that the differences at the input and output cancel out. Since only one byte is active, this has a probability of approximately 2^{-8}. Hence, the probability of the outbound phase of the attack is $2^{-112} \cdot 2^{-8} = 2^{-120}$. In other words, we need to generate 2^{120} starting points for the outbound phase to find one collision.

Since we can find one of these starting points in the inbound phase with an average complexity of 1, we can find a collision for the Whirlpool hash function reduced to 4.5 rounds with a complexity of about 2^{120} and negligible memory.

5.2 Improving the Collision Attack on 4.5 Rounds

In this section, we show how the complexity of the collision attack presented in the previous section can be improved significantly. The main idea is to extend the inbound phase of the attack by 1 round such that one 8 to 1 transition of the outbound phase is covered in the inbound phase of the attack. This improves the probability of the outbound phase significantly from 2^{-120} to $2^{-56-8} = 2^{-64}$. In other words, we need to construct only 2^{64} instead of 2^{120} starting points for the outbound phase of the attack in the inbound phase. In the following, we show how to find inputs that follow the differential trail in the inbound phase of the attack with the following sequence of active bytes:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8$$

Note that the attack is very similar to the attack on the hash function Grøstl in [29]. It can be summarized as follows.

1. Similar to the previous section, we first choose a difference for the 8 active bytes at the output of MixRows of round r_3 (S_3^{MR}) and propagate backward to get the differences of the full active state at the output of SubBytes of round r_3 (S_3^{SB}).
2. In the second step we choose a difference for the active byte in each row at the input of MixRows of round r_2 (S_2^{SC}) and compute forward to the input of SubBytes of round r_3 (S_2). Again, we can choose 2^8 differences for each row and compute each row independently.
3. Next, we look for a matching input/output difference of the SubBytes layer of round r_3 for each row of S_2 and S_3^{SB} independently. This is done with a precomputed 256×256 lookup table as described in Section 4.4. Since the expected number of solutions per trial is one and we have 2^8 candidates for each row of S_2 the expected number of solutions for each row equals 2^8, i.e. 2 solutions for each S-box.
4. For all 2^8 solution of each row of S_2, we compute backward to S_1. Since MixRows works independently on each row and since SubBytes, ShiftColumns,

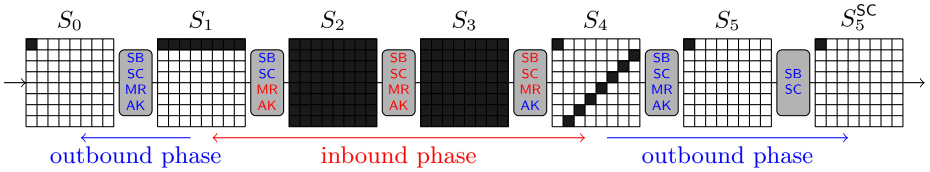

Fig. 7. Differential trail for the collision attack on 5.5 rounds of Whirlpool.

and AddRoundKey are byte-wise operations, this determines only 8 bytes of S_1 and the according differences (active bytes). In detail, we get 2^8 candidates for each active byte in S_1 after testing all 2^8 solutions for each row of S_2 independently. Hence, we get 2^{64} candidates for the 8 active bytes in row 1 of S_1 after this step of the attack with a complexity of about 2^8 round transformations.

5. In order to follow the differential trail in the inbound phase of the attack, we have to guarantee that the differences in S_1 propagate from 8 to 1 active byte through the MixRows transformation in the backward direction. Therefore, we compute all 2^8 differences of the single active byte at the input of MixRows in round r_1 (S_1^{SC}) forward to the input of SubBytes in round r_2 (S_1) and check for a match. Since we have 2^{64} candidates for the active bytes in S_2, i.e. 2^8 for each active byte, the expected number of solutions is 2^8 after testing all 2^8 candidates for the one active byte in S_1^{SC}. In other words we get 2^8 solutions (actual values) that follow the differential trail in the inbound phase of the attack with a complexity of about 2^8 round transformations.

Since the probability of the outbound phase of the attack is 2^{-64}, we need to repeat steps 1-5 about 2^{56} times to generate 2^{64} starting points for the outbound phase of the attack. Since we can find 2^8 starting point for the outbound phase with a complexity of 2^8, we can construct a collision for the Whirlpool hash function reduced to 4.5 rounds with a complexity of about 2^{64}.

5.3 Collision Attack on 5.5 Rounds

In this section, we present a collision attack for the Whirlpool hash function reduced to 5.5 rounds with a complexity of about 2^{184-s} and memory requirements of 2^s, with $0 \leq s \leq 64$. The attack is a straightforward extension of the collision attack on 4.5 rounds of Whirlpool described in Section 5.1. By adding one round in the inbound phase of the attack we can extend the attack to 5.5 rounds (see Figure 7). This idea was introduced in [26], applied to the SHA-3 candidate Grøstl in [32], and called *super-sbox cryptanalysis* in [17]. In the 5.5 round collision attack, we use the following sequence of active bytes:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 64 \xrightarrow{r_4} 8 \xrightarrow{r_5} 1 \xrightarrow{r_{5.5}} 1$$

Again, we split the block cipher W into three sub-ciphers $W = W_{fw} \circ W_{in} \circ W_{bw}$, such that the full active states of the trail are covered by the inbound phase

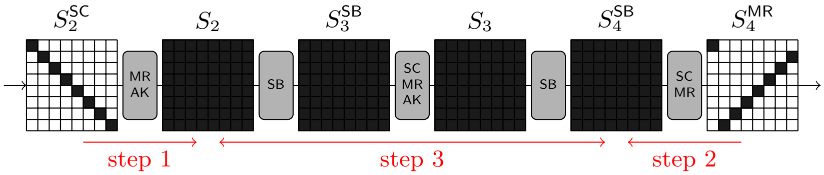

Fig. 8. The inbound phase of the collision attack on 5.5 rounds of Whirlpool.

W_{in}, while the trail in the outbound phase (W_{fw}, W_{bw}) can be fulfilled with a relatively high probability.

$$\begin{aligned}
W_{bw} &= \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \\
W_{in} &= \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \\
W_{fw} &= \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK} \circ \mathsf{MR} \circ \mathsf{SC} \circ \mathsf{SB} \circ \mathsf{AK}
\end{aligned}$$

Since the outbound phase is identical to the attack on 4.5 rounds, we only discuss the inbound phase of the attack here (see Figure 8).

Since the outbound phase of the attack has a probability of 2^{-120}, we have to generate 2^{120} starting points in the inbound phase of the attack. This can be summarized as follows.

1. Start at the input of MixRows in round r_2 (S_2^{SC}) with arbitrary nonzero differences in the 8 byte positions indicated on Figure 8. Propagate the difference forward to the input of SubBytes in round r_3 (S_2). Since we have one active byte in each row of the state this results in a full active state S_2.
2. Start with an arbitrary difference in the 8 active bytes at the output of MixRows in round r_4 (S_4^{MR}) and compute backward to the output of SubBytes in round r_4 (S_4^{SB}). Again, since we start with one active byte in each row, we get a full active state in S_4^{SB}.
3. Next we have to connect the states S_2 and S_4^{SB} such that the differential trail holds. Note that this can be done for each row of S_4^{SB} independently, which facilitates the attack. It can be summarized as follows.
   (a) For all 2^{64} actual values of the first row of S_4^{SB} compute backward to S_2 and check if the differential trail holds. Since MixRows works on each row independently and ShiftColumns and SubBytes are byte-wise operations, this determines 8 bytes of S_2 and the according differences. Hence, after testing all 2^{64} candidates, the expected number of inputs such that the differential trail holds, is one.
   (b) Do the same for row 2-8 of S_4^{SB}.

   After testing each row independently, the expected number of solutions is 1. Hence, we expect to get one actual value for state S_4^{SB} (and S_2) such that the differential trail holds. This step has a total complexity of about 2^{64} round computations.

To summarize, we can compute one starting point for the outbound phase of the attack with a complexity of about 2^{64}. Since we need 2^{120} starting points in the inbound phase, the collision attack has a complexity of about 2^{184}.

Note that the complexity of the inbound phase can be significantly reduced at the cost of higher memory requirements. By saving 2^s candidates for the differences (active bytes) in S_2, we can do a standard time/memory tradeoff with a complexity of about 2^{184-s} and memory requirements of 2^s with $0 \leq s \leq 64$. Hence, by setting $s = 64$ we can find a collision for the Whirlpool hash function reduced to 5.5 rounds with a complexity of about 2^{120} and memory requirements of 2^{64}.

5.4 Near-Collision for Whirlpool

The collision attacks on 4.5 and 5.5 rounds can be further extended by adding one round at the beginning and one round at the end of the trail. The result is a near-collision attack on 6.5 and 7.5 rounds of the hash function Whirlpool. We use the following sequence of active bytes

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 8 \xrightarrow{r_5} 1 \xrightarrow{r_6} 8 \xrightarrow{r_{6.5}} 8$$

for the near-collision attack on 6.5 rounds, and

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 64 \xrightarrow{r_5} 8 \xrightarrow{r_6} 1 \xrightarrow{r_7} 8 \xrightarrow{r_{7.5}} 8$$

for the near-collision attack on 7.5 rounds. In the following, we summarize the attack for 7.5 rounds. Note that the attack on 6.5 rounds works similar. Since the inbound phase is identical to the collision attack on 5.5 rounds, we only discuss the outbound phase here.

First, note that the 1-byte difference at the beginning and end of the 5.5 round trail will always result in 8 active bytes after one MixRows transformation. Thus, we can go both backward and forward 1 round with no additional costs. After the feed-forward, the position of two active bytes match and cancel each other with a probability of 2^{-16}. In other words, the outbound phase of attack has a probability of about 2^{-112} to construct a near-collision in 50 bytes and 2^{-128} to construct a near-collision in 52 bytes. Hence, we have to construct 2^{112} and 2^{128} starting points in the inbound phase of the attack to find a near-collision in 50 and 52 bytes, respectively. Since in the collision attack on 5.5 rounds one can construct 2^s starting points in the inbound phase of the attack with a complexity of about 2^{64} and memory requirements of 2^s with $0 \leq s \leq 64$ (see Section 5.3), the attack has a complexity of about 2^{176-s} and 2^{192-s}, respectively. Both attacks have memory requirements of 2^s.

Note that the attack on 6.5 rounds works similarly, except for the inbound phase of the attack. Since one can find a solution for the inbound phase with an average complexity of 1 (see Section 5.1), we can construct a near-collision in 50 and 52 bytes with a complexity of about 2^{112} and 2^{128}, respectively. Similar to the collision attack on 4.5 rounds one can improve the complexity of the attack by a factor of 2^{56}. Again, we extend the inbound phase of the attack by one round

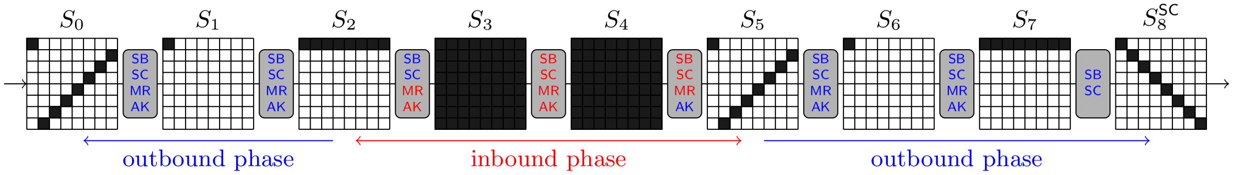

Fig. 9. Differential trail for the near-collision attack on 7.5 rounds of Whirlpool, constructed by extending the 5.5-round trail with one round at the beginning and one round at the end of the outbound phase.

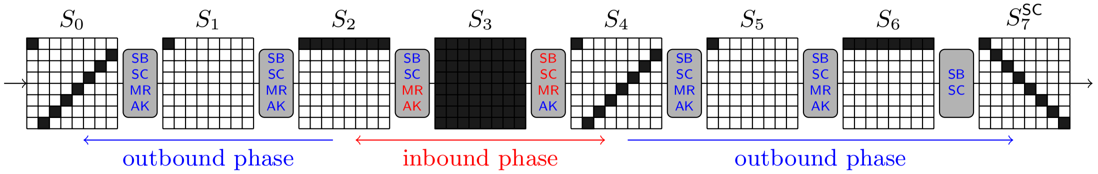

Fig. 10. Differential trail for the near-collision attack on 6.5 rounds of Whirlpool, constructed by extending the 4.5-round trail with one round at the beginning and one round at the end of the outbound phase.

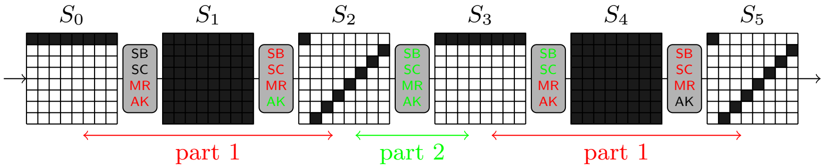

Fig. 11. The extended inbound phase of the attack on the compression function of Whirlpool.

such that one 8 to 1 transition of the outbound phase is covered by the inbound phase of the attack (see Section 5.2). Hence, we can construct a near-collision in 50 and 52 bytes for the Whirlpool hash function reduced to 6.5 rounds with a complexity of about 2^{56} and 2^{72}, respectively.

6 Attacks on the Compression Function

In this section, we will present attacks on the Whirlpool compression function. Since in an attack on the compression function, the attacker has full control over the chaining variable input, this can be used to extend the previous attacks to more rounds. In detail, we can show a semi-free-start collision for the Whirlpool compression function reduced to 7.5 rounds and a semi-free-start near-collision for 9.5 rounds. The basic idea is to have an extended inbound phase consisting of two instead of one inbound phase and connect them by choosing the subkeys accordingly. The outbound phase of the attacks are identical to the previous attacks on the Whirlpool hash function on 5.5 and 7.5 rounds (see Section 5). In the following, we describe both, the extended inbound phase and the outbound phase of the attack in detail.

6.1 The Extended Inbound Phase

In this section, we describe the extended inbound phase consisting of 2 independent inbound phases in detail. We use the following sequence of active bytes for the attack:

$$8 \xrightarrow{r_1} 64 \xrightarrow{r_2} 8 \xrightarrow{r_3} 8 \xrightarrow{r_4} 64 \xrightarrow{r_5} 8$$

In order to find inputs following the differential trail, we split the attack into two parts. In the first part, we have two inbound phases: one in round 1-2 and one in 4-5, with active bytes 8 → 64 → 8 each. In the second part, we need to find values for the subkeys such that the resulting differences in the 8 active bytes and the 64 (byte) values of the state between round 2 and 4 can be connected.

Part 1 (The 2 Independent Inbound Phases). This part of the attack consists of two inbound phases in rounds 1-2 and 4-5. It can be summarized as follows:

1. Inbound Phase 1 (round 1-2):
   (a) Start with 8 active bytes at the output of $\mathsf{AddRoundKey}$ in round r_2 (S_2) and propagate backward to the output of $\mathsf{SubBytes}$ in round r_2 (S_2^{SB}).
   (b) Start with 8 active bytes (1 in each row) at the input of $\mathsf{MixRows}$ in round r_1 (S_1^{SC}) and propagate forward to the input of $\mathsf{SubBytes}$ in round r_2 (S_1). Again, this can be done for all 2^8 differences (value of the active byte) and for each row independently.
   (c) Next, we look for a matching input/output difference of the $\mathsf{SubBytes}$ layer of round r_2 for each row of S_1 and S_2^{SB} independently. This can be implemented with a precomputed 256×256 lookup table as described in Section 4.4. Since, on average, we get one solution per trial and we have 2^8 candidates for each row of S_1, the expected number of solutions for each row is 2^8, i.e. 2 solutions for each S-box. After finishing this step we have 2^{64} inputs (2 for each S-box of S_1) that follow the differential trail in round 1-2.
2. Inbound Phase 2 (round 4-5): Do the same as in step 1 for rounds 4-5.

Note that after this part of the attack, we get 2^{64} candidates for S_2^{SB} and 2^{64} candidates for S_4 with a complexity of about 2^9 round transformations.

Part 2 (Connecting the 2 Inbound Phases). In the second part of the attack, we have to connect the results of the two inbound phases. In detail, we have to ensure that the differences in the 8 active bytes (a 64-bit condition) as well as the actual values of S_2^{SB} and S_4 (a 512-bit condition) match by choosing the subkeys K_2, K_3 and K_4 accordingly. In other words, we have to solve the following equation:

$$\mathsf{MR}(\mathsf{SC}(\mathsf{SB}(\mathsf{MR}(\mathsf{SC}(\mathsf{SB}(\mathsf{MR}(\mathsf{SC}(S_2^{\mathsf{SB}})) \oplus K_2))) \oplus K_3))) \oplus K_4 = S_4 \tag{21}$$

with

$$\begin{aligned} K_3 &= \mathsf{MR}(\mathsf{SC}(\mathsf{SB}(K_2))) \oplus C_3 \\ K_4 &= \mathsf{MR}(\mathsf{SC}(\mathsf{SB}(K_3))) \oplus C_4. \end{aligned} \tag{22}$$

Since we have 2^{64} candidates for S_2^{SB}, 2^{64} candidates for S_4 and can choose from 2^{512} values for the subkeys (K_2, K_3 or K_4 because of (22)), the expected number of solutions is 2^{64}.

Since $S_2^{\mathsf{MR}} = \mathsf{MR}(\mathsf{SC}(S_2^{\mathsf{SB}}))$, we can rewrite (21) as follows:

$$\mathsf{MR}(\mathsf{SC}(\mathsf{SB}(\mathsf{MR}(\mathsf{SC}(\mathsf{SB}(S_2^{\mathsf{MR}} \oplus K_2))) \oplus K_3))) \oplus K_4 = S_4 \tag{23}$$

Note that in the Whirlpool block cipher the order of $\mathsf{ShiftColumns}$ and $\mathsf{SubBytes}$ can always be changed without affecting the output of one round. In order to make the subsequent description of the attack easier, we do this here and get the following equation:

$$\mathsf{MR}(\mathsf{SC}(\mathsf{SB}(\mathsf{MR}(\mathsf{SB}(\mathsf{SC}(S_2^{\mathsf{MR}} \oplus K_2))) \oplus K_3))) \oplus K_4 = S_4 \tag{24}$$

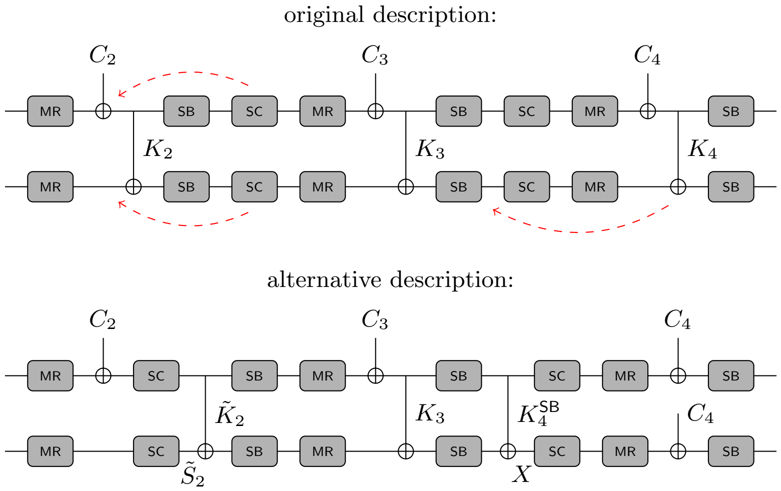

Fig. 12. The sequence of operations is changed to get an equivalent description of the W.

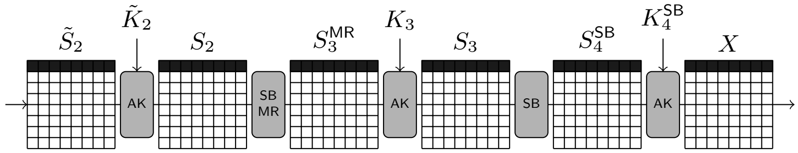

Fig. 13. The second part of the extended inbound phase of the attack on the compression function of Whirlpool by using the alternative description.

Furthermore, MixRows and ShiftColumns are linear transformations and hence we can rewrite the above equation as follows:

$$\mathsf{SB}(\mathsf{MR}(\mathsf{SB}(\tilde{S}_2 \oplus \tilde{K}_2)) \oplus K_3) \oplus K_4^{\mathsf{SB}} = X \tag{25}$$

with $\tilde{S}_2 = \mathsf{SC}(S_2^{\mathsf{MR}})$, $\tilde{K}_2 = \mathsf{SC}(K_2)$, $K_4^{\mathsf{SB}} = \mathsf{SB}(K_3)$, $X = \mathsf{SC}^{-1}(\mathsf{MR}^{-1}(S_4 \oplus C_4))$.

Figure 12 shows how the sequence of operations between state S_2^{MR} and S_4 of the Whirlpool state update and key schedule are changed. In the following, this equivalent description is used to connect the values and differences of the two states $\tilde{S}_2$ and X.

Remember that the differences of S_2^{SB} and S_4 have already been fixed in part 1 of the attack. Since ShiftColumns, MixRows and AddRoundKey are linear transformations, also the differences of $\tilde{S}_2$ and X are fixed. However, we can still choose from 2^{64} candidates for each of the states $\tilde{S}_2$ and X, since we found 2^{64} candidates for S_2^{SB} and 2^{64} candidates for S_4 in part 1 of the attack. Note that we can compute and store the candidates of $\tilde{S}_2$ (from S_2^{SB}) and X (from

S_4) row-by-row and independently. Hence, both the complexity and memory requirements for this step are 2^8 instead of 2^{64}.

Now, we use (25) to determine the subkey $\tilde{K}_2$ such that we get a solution for the extended inbound phase and hence, a starting point for the outbound phase of the attack. Note that we can solve (25) for each row of the states independently. It can be summarized as follows. (see Figure 13).

1. Since $\mathsf{AddRoundKey}$ is a linear transformation, we can compute the 8-byte difference in S_2 (form $\tilde{S}_2$) and S_4^{SB} (from X). We want to stress that these differences are the same for all 2^{64} candidates of the state $\tilde{S}_2$ and all 2^{64} candidates of the state X, respectively.
2. Choose arbitrary values for the first row of S_2 and compute forward to obtain the differences and values of the first row of S_3^{MR}. Again, since $\mathsf{AddRoundKey}$ is a linear transformation, this also determines the difference of S_3.
3. Next, we choose the first row of the key K_3 such that the differential of the S-box between S_3 and S_4^{SB} holds. This can be done similar as in the inbound phase with a precomputed 256×256 lookup table as described in Section 4.4.
4. Once the first row of K_3 is fixed we can also compute the first row of $\tilde{K}_2$ and K_4^{SB}. This also determines the first row (64 bits) of $\tilde{S}_2$ and the first row (64 bits) of X. Remember that we have 2^{64} candidates for state $\tilde{S}_2$ and 2^{64} candidates for state X due to step 1. Hence, the expected number of compatible candidates for both $\tilde{S}_2$ and X equals 1. In other words, we can connect the values and differences of the first row of $\tilde{S}_2$ and X with an average complexity of one.
5. Next, we have to connect the values of $\tilde{S}_2$ and X for rows 2-8. Note that this can be done independently for each row by a simple brute-force search over all 2^{64} values of the corresponding row of $\tilde{K}_2$. Since we have to connect 64 bits and we test 2^{64} values for each row of $\tilde{K}_2$ the expected number of solutions is one.

Since we can repeat the above procedure 2^{64} times with different values for the first row of S_2, we get in total 2^{64} solutions (matches) connecting state $\tilde{S}_2$ to state X with a complexity of 2^{128} and memory requirements of 2^8. In other words, we get 2^{64} starting points for the outbound phase of the attack. Hence, the average complexity to find one starting point for the outbound phase is 2^{64}.

Note that Step 5 can be implemented using a precomputed lookup table of size 2^{128}. In this lookup table each row of the key K_2 (64 bits) is saved for the corresponding two rows of $\tilde{S}_2$ and X (64 bits each). Using this lookup table, we can find one starting point for the outbound phase with an average complexity of 1. However, the complexity to generate this lookup table is 2^{128}. It is important to note that one can construct a total of 2^{192} starting points in the extended inbound phase to be used in the outbound phase of the attack.

6.2 Outbound Phase

In the outbound phase of the attack, we further extend the differential trail backward and forward. By propagating the matching differences and state values through the adjacent SubBytes layers, we get a truncated differential in 8 active bytes in each direction. These truncated differentials need to follow a specific active byte pattern to result in a semi-free-start collision for 7.5 rounds and a semi-free-start near-collision for 9.5 rounds, respectively. In the following, we describe the outbound phase of the two attacks in detail.

Semi-Free-Start Collision for 7.5 Rounds. By adding 1 round in the beginning and 1.5 rounds at the end of the trail, we get a semi-free-start collision for 7.5 rounds for the compression function of Whirlpool with the following sequence of active bytes:

$$1 \xrightarrow{r_1} 8 \xrightarrow{r_2} 64 \xrightarrow{r_3} 8 \xrightarrow{r_4} 8 \xrightarrow{r_5} 64 \xrightarrow{r_6} 8 \xrightarrow{r_7} 1 \xrightarrow{r_{7.5}} 1$$

For the differential trail to hold, we require that the truncated differentials in the outbound phase first propagate from 8 to 1 active byte through the MixRows transformation, both in the backward and forward direction (see Figure 14). Since the transition from 8 active bytes to 1 active byte through the MixRows transformation has a probability of about 2^{-56}, and the exact value of the input and output difference in one byte has to match after the feed-forward to get a semi-free-start collision, the outbound phase has a probability of $2^{-2\cdot 56-8} = 2^{-120}$. In other words, we have to generate 2^{120} starting points (for the outbound phase) in the extended inbound phase of the attack.

Since we can find one starting point with an average complexity of about 2^{64} and memory requirements of 2^{8}, we can find a semi-free-start collision with a complexity of about $2^{120+64} = 2^{184}$. The complexity of the attack can be reduced to 2^{120} by using a precomputed lookup table of size 2^{128} in the extended inbound phase of the attack.

Semi-Free-Start Near-Collision for 9.5 Rounds. As in the attack on the Whirlpool hash function, the semi-free-start collision attack on 7.5 rounds can be further extended by adding one round at the beginning and one round at the end of the trail in the outbound phase. The result is a semi-free-start near-collision for 9.5 rounds of the compression function with the following sequence of active bytes (see Figure 15):

$$8 \xrightarrow{r_1} 1 \xrightarrow{r_2} 8 \xrightarrow{r_3} 64 \xrightarrow{r_4} 8 \xrightarrow{r_5} 8 \xrightarrow{r_6} 64 \xrightarrow{r_7} 8 \xrightarrow{r_8} 1 \xrightarrow{r_9} 8 \xrightarrow{r_{9.5}} 8$$

Since the 1-byte difference at the beginning and end of the 7.5 round trail will always result in 8 active bytes after one MixRows transformation, we can go backward 1 round and forward 1 round with no additional cost. Using the feed-forward, the positions of two active S-boxes match and cancel one another with a probability of 2^{-16}. Hence, we get a semi-free-start near-collision in 50 and 52 bytes for the compression function of Whirlpool with a complexity of about 2^{176} and $2^{176+16} = 2^{192}$, respectively. Again, by using a precomputed lookup table (size 2^{128}) in the extended inbound phase the complexity of the attack can be reduced significantly. The result is a semi-free-start near-collision for 9.5 rounds of Whirlpool with a complexity of about 2^{112} and 2^{128}, respectively.

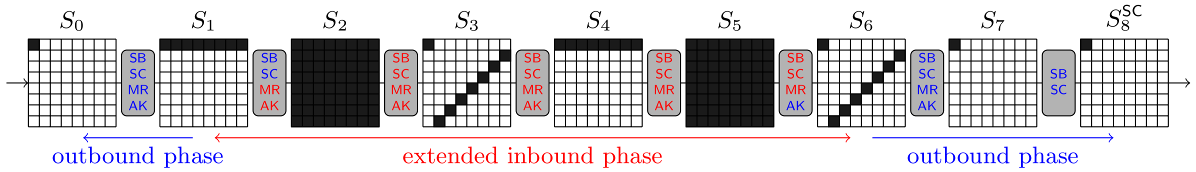

Fig. 14. Differential trail for the semi-free-start near-collision attack on 7.5 rounds of the compression function of Whirlpool, constructed by extending the 5-round trail with one round at the beginning and 1.5 rounds at the end.

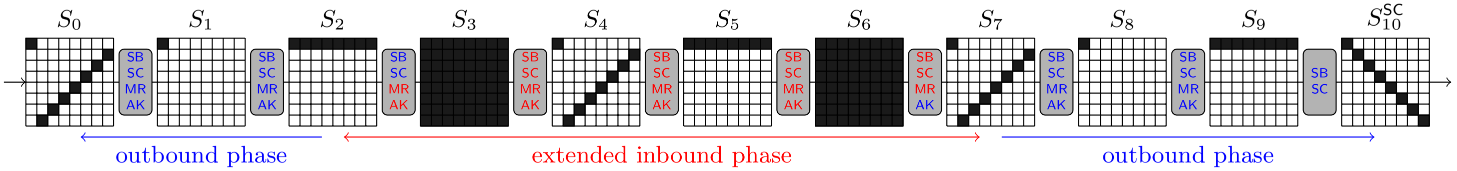

Fig. 15. Differential trail for the semi-free-start near-collision attack on 9.5 rounds of the compression function of Whirlpool, constructed by extending the 7.5-round trail with one round at the beginning and one round at the end of the outbound phase.

7 Distinguisher for the Whirlpool Compression Function

Now we will show how the compression function attack described in Section 6 can be used to show a certificational weakness in the full Whirlpool compression function. To be more precise, we will show how to distinguish the Whirlpool compression function from a random (that is, randomly selected) one-way function using the results described in Section 3.

Obviously, the difference between two Whirlpool states can be seen as a vector in the vector space of dimension $N = 512$ over $\mathbb{F}_2$. The crucial observation is that the attack of Section 6 can be interpreted as an algorithm that can find t difference vectors in $\mathbb{F}_2^{512}$ (output differences of the compression function) that form a subspace $V_{out} \subseteq \mathbb{F}_2^{512}$ with $\dim(V_{out}) \leq 128$. To see this, observe that by extending the differential trail from 9.5 to 10 rounds, the 8 active bytes in S_{10}^{SC} will always result in a full active state S_{10} due to the properties of the MixRows transformation. Thus the near-collision is destroyed. However, even though after the application of MixRows and AddRoundKey the state S_{10} is fully active in terms of truncated differences, the `xor` differences in S_{10} still belong to a subspace of $\mathbb{F}_2^{512}$ of dimension at most 64 due to the properties of MixRows. If we look again at Figure 15, both the differences in S_0 (respectively the message block M_j) and the differences in S_{10}^{SC} can be seen as (difference) vectors belonging to subspaces of $\mathbb{F}_2^{512}$ of dimension at most 64. Therefore, after the feed-forward, we can conclude that the differences at the output of the compression function form a subspace $V_{out} \subseteq \mathbb{F}_2^{512}$ with $\dim(V_{out}) \leq 128$

Hence, we can use the attack of Section 6 to find t input differences such that the corresponding output differences form a vector space V_{out} of dimension $n \leq 128$. This has a complexity of $t \cdot 2^{176}$ or $t \cdot 2^{112}$ using a precomputation step with complexity 2^{128}. Note that $t \leq 2^{192-112} = 2^{80}$, due to the restrictions in the extended inbound phase of the attack (see Section 6.1).

Now the main question is for which values of t our attack is more efficient than the generic attack. In other words, how do we have to choose t such that we can distinguish the compression function of Whirlpool from a random one-way function. Table 3 compares for several values of t the complexity of our dedicated approach to the query complexity in the generic case (cf. Section 3).

As can be seen in the table, the Subspace Problem for the full Whirlpool compression function is easier to solve than for a random one-way function when we take $t = 2^{12}$. The complexity of the attack is then about 2^{188}. The probability to solve the Subspace Problem when making $Q = 2^{188}$ queries to a random one-way function with the parameters $t = 2^{12}$, $N = 512$ and $n = 128$ is $\approx 2^{-30833}$. This follows from Proposition 1. Therefore, we get a distinguishing attack on the full Whirlpool compression function. Note, that by using a precomputation table as described in Section 6, the complexity reduces to 2^{121} with $t = 2^9$.

8 Distinguisher for the Block Cipher *W*

After our result on the compression function of Whirlpool, we now show how the subspace distinguisher for the compression function of Whirlpool can also

Table 3. Values for t, Q (query complexity), C (complexity of our attack), and C_p (complexity of our attack with precomputation) for $p = 1, N = 512, n = 128$

$\log_2(t)$	$\log_2(Q)$	$\log_2(C)$	$\log_2(C_p)$
9	148.16	185	121
10	172.72	186	122
11	185.25	187	123
12	191.76	188	124
13	195.27	189	125
14	197.28	190	126
15	198.53	191	127
16	199.40	192	128

be used to distinguish the block cipher W in the open-key setting [25]. To be more precise, we show a distinguisher for W reduced to 8 rounds in the known-key setting and a distinguisher for the full block cipher (all 10 rounds) in the chosen-key setting.

8.1 Known-Key Distinguisher for 8 Rounds

In this section, we present a known-key distinguisher for the block cipher W reduced to 8 rounds. In this setting, the adversary is given the key and the goal is to distinguish the given permutation from a randomly selected permutation on the plaintext space of the block cipher. This can be done by using the hardness of Subspace Problem 2 described in Section 3.2. That is, for a given permutation we have to find t plaintext pairs (p_i, p_i^*), such that the differences $p_i \oplus p_i^*$ form a subspace $V_{in} \subseteq \mathbb{F}_2^M$ with $\dim(V_{in}) \leq m$ and for the corresponding ciphertext pairs the differences $c_i \oplus c_i^*$ form a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$.

For the block cipher W reduced to 8 rounds and for a given key, the attack of Section 5.4 can be interpreted as an algorithm to find t plaintext pairs (p_i, p_i^*) with $p_i \oplus p_i^*$ belonging to a vector space of dimension $m \leq 64$, such that for the corresponding ciphertext pairs the differences $c_i \oplus c_i^*$ form a vector space of dimension $n \leq 64$. The resulting complexity is about $t \cdot 2^{176-s}$ with memory requirements of 2^s and $0 \leq s \leq 64$. So in other words, we have found a solution for Subspace Problem 2 in the case of W reduced to 8 rounds.

The discussion in Section 3.2 culminates in lower bounds for the query complexity of a non-adaptive adversary when trying to solve Subspace Problem 2 for a random permutation to which only black-box access is admissible. Table 4 compares the complexity of the generic approach of Section 3.2 and our dedicated approach for several values of t.

As can be seen in the table, the Subspace Problem for the Whirlpool block cipher, reduced to 8 rounds in the known key setting, is easier to solve than for a random permutation when we take $t = 2^{10}$. The complexity of the attack is then about 2^{186-s}. The probability for a non-adaptive adversary to solve the

Table 4. Values for t, Q (query complexity), C_s (complexity of our attack with $0 \leq s \leq 64$), for $p = 1, N = 512, n = 64, m = 64$

$\log_2(t)$	$\log_2(Q)$	$\log_2(C_s)$
10	336.44	186-s
11	365.50	187-s
12	380.53	188-s
13	388.54	189-s
14	393.05	190-s
15	395.80	191-s
16	397.68	192-s

Subspace Problem 2 when making $Q = 2^{186-s}$ queries to a random permutation with the parameters $t = 2^{10}$, $N = 512$, $n = m = 64$ and $s = 64$ is $\approx 2^{-350655}$. This follows from Proposition 2.

8.2 Chosen-Key Distinguisher for 10 Rounds

In the chosen-key setting, an adversary is also given control over the key-input. The goal of a distinguishing attack in this setting is to be able to distinguish the block cipher W from an ideal cipher. Again, we want to use something similar as Subspace Problem 2 for this task.

For the block cipher W we want to find t triples (p_i, p_i^*, k_i) such that the plaintext differences $p_i \oplus p_i^*$ form a subspace $V_{in} \subseteq \mathbb{F}_2^N$ with $\dim(V_{in}) \leq m$ and for the corresponding ciphertext pairs the differences $c_i \oplus c_i^*$ form a subspace $V_{out} \subseteq \mathbb{F}_2^N$ with $\dim(V_{out}) \leq n$.

The above task can be solved along the same lines as it was done in Section 7. Namely, we use the 9.5-round near-collision attack on the Whirlpool compression function of Section 6 to solve the above described problem. The only difference to Section 7 is that in the block cipher case, we omit the feed-forward. Thus, our attacks finds t triplets (p_i, p_i^*, k_i) confining to the above conditions with a complexity of $t \cdot 2^{176}$ (respectively $t \cdot 2^{112}$ with precomputation). Note that in our setting, a non-adaptive adversary is not able to exploit the fact that he can choose the keys, since he has to decide upon his queries on beforehand to solve the Subspace Problem 2. This is the reason why we can again use Proposition 2 with $Q = 2^{122}$, $t = 2^{10}$, $N = 512$ and $n = m = 64$ to show that the success probability for such an adversary is $\approx 2^{-285119}$ for a randomly selected cipher.

8.3 Related Work

For the block cipher W that is used in the Whirlpool compression function, Knudsen described a distinguisher for 6 (out of 10) rounds [23]. It needs 2^{120} inputs and has a complexity of 2^{120}. In [25] similar techniques were used to obtain known-key distinguishers for 7 rounds of the AES. Furthermore, the designers of Whirlpool describe in [2] a key recovery attack against W reduced to 7 rounds

Table 5. Values for t, Q (query complexity), C (complexity of our attack), and C_p (complexity of our attack with precomputation) for $p = 1, N = 512, n = 64, m = 64$

$\log_2(t)$	$\log_2(Q)$	$\log_2(C)$	$\log_2(C_p)$
10	336.44	186	122
11	365.50	187	123
12	380.53	188	124
13	388.54	189	125
14	393.05	190	126
15	395.80	191	127
16	397.68	192	128

with a complexity of about 2^{245}. It is an extension of the attack by Gilbert and Minier on AES [16].

9 Concluding Remarks

In this paper, we have explained the rebound attack on hash functions and applied it to the hash function Whirlpool. We presented a detailed security analysis of the Whirlpool hash function and the Whirlpool compression function with respect to collision resistance.

We have also introduced two subspace problems as natural generalizations of near-collision resistance for the cases of (non-invertible) functions and permutations. We have used rebound attacks to show that the compression function of Whirlpool does not have the resistance of ideal primitives against distinguishers based on one of these subspace problems.

An interesting property of both subspace problems is that the associated distinguishers are not affected by the presence of an invertible linear transformation at the end of the compression function or its underlying block cipher. This property corresponds to the common intuition that invertible linear transformation at the start or the end do not affect the security of a primitive, but it is not satisfied by the usual definition of near-collision resistance.

Thus far, the rebound attack has been applied mostly to hash functions that are based on or inspired by the AES design. This can be interpreted as a weakness of the AES design. However, one can also argue that the simple structure of AES simply accelerates understanding of new designs, and thereby the development of attacks. In that case, more results can be expected on other types of hash functions.

Acknowledgements

The work in this paper has been supported in part by the Secure Information Technology Center - Austria (A-SIT), by the Austrian Science Fund (FWF), project P21936-N23, by the IAP Programme P6/26 (BCRYPT) of the Belgian State (Belgian Science Policy) and by the European Commission under contract ICT-2007-216646 (ECRYPT II).

References

1. Information technology – Security techniques – Hash-functions – Part 3: Dedicated hash-functions. ISO/IEC 10118-3:2004., 2004.
2. Paulo S. L. M. Barreto and Vincent Rijmen. The Whirlpool Hashing Function. Submitted to NESSIE, September 2000. Revised May 2003. Available online at `http://www.larc.usp.br/~pbarreto/WhirlpoolPage.html` (2008/12/11).
3. Ryad Benadjila, Olivier Billet, Henri Gilbert, Gilles Macario-Rat, Thomas Peyrin, Matt Robshaw, and Yannick Seurin. Sha-3 proposal: Echo. Submission to NIST, 2008.
4. Eli Biham and Adi Shamir. Differential Cryptanalysis of DES-like Cryptosystems. *J. Cryptology*, 4(1):3–72, 1991.
5. Alex Biryukov, Dmitry Khovratovich, and Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *LNCS*, pages 231–249. Springer, 2009.
6. Christophe De Cannière, Florian Mendel, and Christian Rechberger. Collisions for 70-Step SHA-1: On the Full Cost of Collision Search. In Carlisle M. Adams, Ali Miri, and Michael J. Wiener, editors, *Selected Areas in Cryptography*, volume 4876 of *LNCS*, pages 56–73. Springer, 2007.
7. Christophe De Cannière and Christian Rechberger. Finding SHA-1 Characteristics: General Results and Applications. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *LNCS*, pages 1–20. Springer, 2006.
8. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *LNCS*, pages 56–71. Springer, 1998.
9. Joan Daemen and Vincent Rijmen. The Wide Trail Design Strategy. In Bahram Honary, editor, *IMA Int. Conf.*, volume 2260 of *LNCS*, pages 222–238. Springer, 2001.
10. Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.
11. Ivan Damgård. A Design Principle for Hash Functions. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 416–427. Springer, 1989.
12. Hans Dobbertin. The status of MD5 after a recent attack. *CryptoBytes*, 2(2):3–6, 1996.
13. Hans Dobbertin. Cryptanalysis of MD4. *J. Cryptology*, 11(4):253–271, 1998.
14. S. D. Fisher. Classroom Notes: Matrices over a Finite Field. *Amer. Math. Monthly*, 73(6):639–641, 1966.
15. Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Grøstl – a SHA-3 candidate. Available online at `http://www.groestl.info`, 2008.
16. Henri Gilbert and Marine Minier. A Collision Attack on 7 Rounds of Rijndael. In *AES Candidate Conference*, pages 230–241, 2000.
17. Henri Gilbert and Thomas Peyrin. Super-Sbox Cryptanalysis: Improved Attacks for AES-like permutations. Cryptology ePrint Archive, Report 2009/531, 2009.
18. N. Haller. RFC1760: The S/KEY One-Time Password System, 1995.

19. N. Haller, C. Metz, P. Nesser, and M. Straw. RFC2289: A One-Time Password System, 1998.
20. John Kelsey and Stefan Lucks. Collisions and Near-Collisions for Reduced-Round Tiger. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *LNCS*, pages 111–125. Springer, 2006.
21. Dmitry Khovratovich, Alex Biryukov, and Ivica Nikolic. Speeding up Collision Search for Byte-Oriented Hash Functions. In Marc Fischlin, editor, *CT-RSA*, volume 5473 of *LNCS*, pages 164–181. Springer, 2009.
22. Lars R. Knudsen. Truncated and Higher Order Differentials. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 196–211. Springer, 1994.
23. Lars R. Knudsen. Non-random properties of reduced-round Whirlpool. NESSIE public report, NES/DOC/UIB/WP5/017/1, 2002.
24. Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Alex Biryukov, editor, *FSE*, volume 4593 of *LNCS*, pages 39–57. Springer, 2007.
25. Lars R. Knudsen and Vincent Rijmen. Known-Key Distinguishers for Some Block Ciphers. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 315–324. Springer, 2007.
26. Mario Lamberger, Florian Mendel, Christian Rechberger, Vincent Rijmen, and Martin Schläffer. Rebound Distinguishers: Results on the Full Whirlpool Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 126–143. Springer, 2009.
27. Rudolf Lidl and Harald Niederreiter. *Finite fields*, volume 20 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, second edition, 1997. With a foreword by P. M. Cohn.
28. Krystian Matusiewicz, María Naya-Plasencia, Ivica Nikolic, Yu Sasaki, and Martin Schläffer. Rebound Attack on the Full Lane Compression Function. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *LNCS*, pages 106–125. Springer, 2009.
29. Florian Mendel, Thomas Peyrin, Christian Rechberger, and Martin Schläffer. Improved Cryptanalysis of the Reduced Grøstl Compression Function, ECHO Permutation and AES Block Cipher. In Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *LNCS*, pages 16–35. Springer, 2009.
30. Florian Mendel, Bart Preneel, Vincent Rijmen, Hirotaka Yoshida, and Dai Watanabe. Update on Tiger. In Rana Barua and Tanja Lange, editors, *INDOCRYPT*, volume 4329 of *LNCS*, pages 63–79. Springer, 2006.
31. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *LNCS*, pages 260–276. Springer, 2009.
32. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. Rebound Attacks on the Reduced Grøstl Hash Function. In Josef Pieprzyk, editor, *CT-RSA*, volume 5985 of *LNCS*, pages 350–365. Springer, 2010.
33. Florian Mendel and Vincent Rijmen. Cryptanalysis of the Tiger Hash Function. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 536–550. Springer, 2007.
34. Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
35. Ralph C. Merkle. One Way Hash Functions and DES. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *LNCS*, pages 428–446. Springer, 1989.

36. 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.
37. 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. Available: `http://csrc.nist.gov/groups/ST/hash/documents/FR_Notice_Nov07.pdf` (2008/10/17).
38. NESSIE. New European Schemes for Signatures, Integrity, and Encryption. IST-1999-12324. Available online at `http://cryptonessie.org/`.
39. Thomas Peyrin. Cryptanalysis of Grindahl. In Kaoru Kurosawa, editor, *ASIACRYPT*, volume 4833 of *LNCS*, pages 551–567. Springer, 2007.
40. Vincent Rijmen and Bart Preneel. Improved Characteristics for Differential Cryptanalysis of Hash Functions Based on Block Ciphers. In Bart Preneel, editor, *FSE*, volume 1008 of *LNCS*, pages 242–248. Springer, 1994.
41. Herbert Robbins. A remark on Stirling's formula. *Amer. Math. Monthly*, 62:26–29, 1955.
42. David Wagner. The Boomerang Attack. In Lars R. Knudsen, editor, *FSE*, volume 1636 of *LNCS*, pages 156–170. Springer, 1999.
43. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *LNCS*, pages 17–36. Springer, 2005.
44. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *LNCS*, pages 19–35. Springer, 2005.