Analysis of Differential Attacks in ARX Constructions

Gaëtan Leurent

University of Luxembourg – LACS

Abstract. In this paper, we study differential attacks against ARX schemes. We build upon the generalized characteristics of de Cannière and Rechberger; we introduce new multi-bit constraints to describe differential characteristics in ARX designs more accurately, and quartet constraints to analyze boomerang attacks. We also describe how to propagate those constraints; this can be used either to assist manual construction of a differential characteristic, or to extract more information from an already built characteristic. We show that our new constraints are more precise than what was used in previous works, and can detect more cases of incompatibility.

In particular, we show that several published attacks are in fact fact invalid because the differential characteristics cannot be satisfied. This highlights the importance of verifying differential attacks more thoroughly.

Keywords: Symmetric ciphers, Hash functions, ARX, Generalized characteristics, Differential attacks, Boomerang attacks.

1 Introduction

A popular way to construct cryptographic primitives is the so-called ARX design, where the construction only uses Additions ($a \boxplus b$), Rotations ($a \ggg i$), and Xors ($a \oplus b$). These operations are very simple and can be implemented efficiently in software or in hardware, but when mixed together, they interact in complex and non-linear ways. In particular, two of the SHA-3 finalists, Blake and Skein, follow this design strategy. More generally, functions of the MD/SHA family are built using Additions, Rotations, Xors, but also bitwise Boolean functions, and logical shifts; they are sometimes also referred to as ARX. This stategy as also been used for stream ciphers such as Salsa20 and ChaCha, and block ciphers, such as TEA, XTEA, HIGHT, or SHACAL (RC5 uses additions and data-dependant rotations, but we only consider construction with fixed rotations).

The ARX design philosophy is opposed to S-Box based designs such as the AES. Analysis of S-Box based designs usually happen at the word-level, and differential characteristic are relatively easy to build, but efficient attacks often need novel techniques, such as the rebound attack against hash functions [17]. For ARX designs, the analysis is done on a bit-level; finding good differential characteristics remains an important challenge. In particular, the seminal attacks on the MD/SHA-familiy by the team of X. Wang are based on differential characteristics built by hand [28,27,29], and an important effort has been devoted

X. Wang and K. Sako (Eds.): ASIACRYPT 2012, LNCS 7658, pp. 226–243, 2012.

to building tools to construct automatically such characteristics [6,23,8,15,24]. This effort has been quite successful for functions of the MD/SHA family, and it has allowed new attacks based on specially designed characteristics: attacks against HMAC [9], the construction of a rogue MD5 CA certificate [25], and attacks against combiners [16].

Another important problem is that the components of an ARX design can interact in complex and unexpected ways. Differential characteristics are usually built by looking at each operation individually, and multiplying the probabilities of each non-linear operation, but this approach can lead to very misleading results. For SHA-0 and SHA-1 differential characteristics, it has been shown that the hypothesis of independence between the local collisions is flawed, and some patterns of local collisions lead to impossible characteristics [4,21,14]. Problems have also been identified for differential attacks on SHACAL [26]. More recently, Mendel, Nad, and Schläffer have tackled the problem of building differential characteristics for SHA-2, and found that many of them are in fact incompatible [15].

A similar problem has been discussed in the context of boomerang attacks by Murphy [20]: the assumption that the differential characteristics are independent does not necessarily hold. Several recent works have found characteristics that turned out to be incompatible when analyzing ARX hash functions such as HAVAL [22], SHA-256 [2], or Skein [11].

Our Results. In this paper, we try to provide a framework to study these problems for ARX designs. In pure ARX functions, the modular addition is the only source of non-linearity (with respect to the xor difference). Therefore it is important to capture its behaviour as accurately as possible.

We extend the generalized characteristics of de Cannière and Rechberger [6] by introducing constraints involving several consecutive bits of a variable (*i.e.* $x^{[i]}$ and $x^{[i-1]}$), instead of considering bits one by one. We show that constraints on 2 consecutive bits can completely capture the modular difference, and we introduce reduced sets of constraints on 1.5 and 2.5 consecutive bits. This is motivated by the analysis of modular addition, but since these constraints are still local, they interact well with bitwise Boolean operations and rotations, and we can use them to study pure ARX as well as SHA-like constructions. We show that they capture more information than the single bit constraints of [6]. In particular, we describe cases of incompatibility in ARX characteristics due to interactions between consecutive bits, and we show that a proposed path for Skein is invalid [29]. This is detected automatically by our new constraints.

We also study boomerang attacks, and introduce constraints on *quartets* of variables, instead of considering each characteristic separately with constraints on *pairs* of variables. This allows to capture some extra information in the middle of the attack, when the top characteristic and the bottom characteristic meet. In particular, we can automatically detect incompatibilities in previously published attacks against Skein [1,5] and Blake [3].

As opposed to [15], our work is focused on local conditions, and we try to extract as much information as possible from a single operation. If needed, it

can be combined with more computing intensive techniques considering several operations simultaneously.

Additionally, we give a complete description of how to compute the probability of a characteristic using these constraints, and how to do constraints propagation. All our code will be available from our webpage[1] so that these tools can be used by the community to build or verify differential attacks. Our tools are quite generic and we hope that they can be used to study more primitives. We don't provide a complete solution to automatically find differential characteristics in ARX schemes, but we believe our work is an important step in this direction.

This paper is organized as follows: first, we explain the theory of S-systems and how to solve them efficiently in Section 2, and we show how to use S-systems to study differential attacks using the generalized characteristic of de Cannière and Rechberger in Section 3. In Section 4, we introduce multi-bit constraints and show how they improve over previous results. Finally, in Section 5, we describe quartet constraint to study boomerang attacks, and show that they can detect incompatibilities in several attacks.

2 Analysis of S-systems

Since ARX systems in general are hard to analyze, we first study systems without rotations. An important remark is that a system of Additions and Xors, can be seen as a T-function [10], or more precisely, as an S-function [19]. We use the following definitions:

T-function. A T-function on n-bit words with k inputs and l outputs is a function from $(\{0,1\}^n)^k$ to $(\{0,1\}^n)^l$ with the following property:
For all t, the t least significant bits of the outputs can be computed from the t least significant bits of the inputs.

S-function. An S-function on n-bit words is a function from $(\{0,1\}^n)^k$ to $(\{0,1\}^n)^l$, for which we can define a small set of *states* $\mathcal{S}$, and an initial state $S[-1] \in \mathcal{S}$ with the following property:
For all t, bit t of the outputs and the state $S[t] \in \mathcal{S}$ can be computed from bit t of the inputs, and the state $S[t-1]$.

In practice, our analysis will be linear in the number of states, and the number of states can be exponential in the size of the system. We can only study systems with a limited number of states.

For instance, the modular addition is an S-function, with a 1-bit state corresponding to the carry. An S-function can also include bitwise functions, shifts to the left by a fixed number of bits, or multiplications by constants. However, a shift to the left by i bits, or multiplication by constant of i bits, leads to an increase of the state by a factor of 2^i, so the analysis will only be practical for small values of i. Note that the multiplication of two variables, or a data-dependant shift to the left, are T-functions, but are not S-functions because the size of the state has to grow with n.

[1] `http://www.di.ens.fr/~leurent/arxtools.html`

In this work, we consider systems of the form $f(P, x) = 0$ where f is an S-function, P is a vector of p parameters, and x is a vector of v unknown variables. This defines a family of systems, and we are interested in properties of the set of solutions of the unknown x for a given P. We call such a system an S-system.

A simple and yet important example is the system

$$x \oplus \Delta = x \boxplus \delta \tag{1}$$

where the parameter are Δ, δ. Solving this system is equivalent to finding a pair of variables with a given modular difference and a given xor difference, and was an important part of a recent attack on BMW [12].

It is well-known that those systems are T-functions, and can be solved from the least significant bit to the most significant bit. However, the naive approach to solve such a system uses backtracking, and can lead to an exponential complexity in the worst case.[2]

2.1 Representation of S-systems Using Finite State Machines

A more efficient strategy is to use an approach based on Finite State Machines, or automata: any system of such equations can be represented by an automaton, and solving a particular instance take time proportional to the word length. This kind of approach has been used to study differential properties of S-functions in [19].

The first step to apply this technique is to build an automaton corresponding to the system of equations. The states of this automaton correspond to the states of the S-function in $\mathcal{S}$, *i. e.* the carry bits: a system with s modular additions gives an automaton with 2^s states. The alphabet of the automaton is $\{0, 1\}^{p+v}$; each transition reads one bit from each parameter and each variable, starting from the least significant bit. The automaton just accepts (P, x) if and only if $f(P, x) = 0$.

We can then count the number of solutions to the system by counting paths in the graph corresponding to the automaton. In this work we mainly use this technique to decide whether a system is solvable, but we can also compute a random solution, or enumerate the set of solutions.

If the S-system is given as an expression with additions and bitwise Boolean operations, the transition table of the automaton can easily be constructed by evaluating the expression for every possible state, every possible 1-bit parameters, and every possible 1-bit variable.

Decision Automaton. When we remove the information about the variables from the edges, we obtain a non-deterministic automaton which can decide whether a system is solvable or not, *i. e.* whether there *exists* a choice of the variable x so that $f(P, x) = 0$ for a given P. We can then optionally build an equivalent deterministic automaton using the powerset construction.

[2] *e.g.* to solve the system $x \oplus \texttt{0x80000000} = x$, the backtracking algorithm will try all possible values for the 31 lower bits of x before concluding that there is no solution.

Implementation. We have automated the construction of the FSM from a simple description of the S-system. Our tool can deal with any system of additions, and bitwise Boolean functions. For instance System (1) will be written as `V0^P0==V0+P1`, and System (2) will be written as `P0|V0|V1; P1|V0|~V1; P2|~V0|V1; P3|~V0|~V1`. The variables are denoted by `V`i and the parameters by `P`i, and the operations are written naturally with a C-like syntax. The tool outputs the transition table of the automaton, and we have a collection of function to compute properties of the system from this table. From the FSM representation of an S-system, we can automatically derive:

- Whether a given set of parameter leads to a compatible system
- A random solution when the system is compatible
- The number of solutions (and the probability that a random x is a solution)
- A description of the solution set, from which we can efficiently iterate over the solutions

3 Study of Differential Characteristics

The most basic approach to describe a differential characteristic is to choose a difference operation (usually the modular difference $\boxminus$ or the xor difference $\oplus$), and to specify the difference $x' - x$ for every internal variable of a cipher. One can compute the probability of reaching the specified output difference for each operation, and the probability of the full characteristic is computed by multiplying the probabilities of each operation, under the assumption that the probabilities are independent.

However, this approach is not very successful for ARX designs, because the assumption of independence is very often false. To overcome this, Wang *et al.* introduced the notion of a signed difference. For each bit, we now consider three different possibilities:

- $x^{[i]} = x'^{[i]}$, this is denoted as 0;
- $x^{[i]} = 0,\ x'^{[i]} = 1$, this is denoted as $+1$;
- $x^{[i]} = 1,\ x'^{[i]} = 0$, this is denoted as -1.

This gives much better results in the presence of modular addition, because it combines both the modular difference and the xor difference.

3.1 Generalized Constraints

More generally, de Cannière and Rechberger noted that we can define a difference characteristic by allowing certain subsets of the values of (x, x') for each bit of the cipher [6].

Table 1 shows the symbol they use to denote all the possible subsets of $\mathcal{P}(\{0,1\}^2)$. For a given internal state variable x, and a constraint Δ, we write $\delta(x, x') = \Delta$ — or $\delta x = \Delta$ if there is no ambiguity — to means that (x, x') is restricted to the subset defined by Δ.

Table 1. Constraints used in [6]

	(x, x'):	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
?	*anything*	✓	✓	✓	✓
-	$x = x'$	✓	-	-	✓
x	$x \neq x'$	-	✓	✓	-
0	$x = x' = 0$	✓	-	-	-
u	$(x, x') = (0, 1)$	-	✓	-	-
n	$(x, x') = (1, 0)$	-	-	✓	-
1	$x = x' = 1$	-	-	-	✓
#	*incompatible*	-	-	-	-
3	$x = 0$	✓	✓	-	-
5	$x' = 0$	✓	-	✓	-
7		✓	✓	✓	-
A	$x' = 1$	-	✓	-	✓
B		✓	✓	-	✓
C	$x = 1$	-	-	✓	✓
D		✓	-	✓	✓
E		-	✓	✓	✓

Table 2. Trivial encoding

	P_0	P_1	P_2	P_3
?	1	1	1	1
-	1	0	0	1
x	0	1	1	0
0	1	0	0	0
u	0	1	0	0
n	0	0	1	0
1	0	0	0	1
#	0	0	0	0
3	1	1	0	0
5	1	0	1	0
7	1	1	1	0
A	0	1	0	1
B	1	1	0	1
C	0	0	1	1
D	1	0	1	1
E	0	1	1	1

Since the definition of δ only involves bitwise operation, we can write it as an S-system, if we encode Δ as shown in Table 2:

$$P_0 = 0 \Rightarrow (x, x') \neq (0,0) \quad P_1 = 0 \Rightarrow (x, x') \neq (0,1)$$
$$P_2 = 0 \Rightarrow (x, x') \neq (1,0) \quad P_3 = 0 \Rightarrow (x, x') \neq (1,1)$$

or equivalently:

$$P_0 \vee x \vee x' \quad P_1 \vee x \vee \bar{x}' \quad P_2 \vee \bar{x} \vee x' \quad P_3 \vee \bar{x} \vee \bar{x}'. \tag{2}$$

3.2 Differential Characteristics

In order to describe a differential characteristics with this framework, we specify a difference for each internal variable of a cipher, and we consider the operations that connect the variables. For each operation $\odot$, we can write an S-system[3]:

$$\delta x = \Delta_x \qquad \delta y = \Delta_y \qquad \delta z = \Delta_z \qquad z = x \odot y \qquad z' = x' \odot y', \tag{3}$$

where x, y, z, x', y', z' are unknowns, and $\Delta_x, \Delta_y, \Delta_z$ are parameters. Using this S-system, we can verify if the differences specified input and output patterns for each operation are compatible. Moreover, we can compute the probability to reach the specified output pattern by counting the number of solutions. Assuming that the probabilities of each operations are independent, we can compute the probability of the full characteristic by multiplying the probabilities of each operations. We deal with the rotations $y = x \ggg i$ by just rotating the constraint pattern: if $\delta x = \Delta_x$ then we use $\delta y = \Delta_x \ggg i$.

[3] We assume that all the operations except the rotations are S-function, as is the case in ARX designs.

3.3 Propagation of Constraints

This approach can also be used to propagate the constraints associated with a differential characteristic. The main idea is to consider each bit constraint, and to split it into two disjoint subsets; if one of the subsets result in an incompatible system, we known that we can restrict the constraint to the other subset without reducing the number of solutions. More precisely, we use the following splits for the 1-bit constraints of [6]:

`?` $\rightarrow$ `-`/`x`, `3`/`C`, `5`/`A`, `0`/`E`, `1`/`7`, `u`/`D`, `n`/`B`	`-` $\rightarrow$ `0`/`1`		`x` $\rightarrow$ `u`/`n`
`3` $\rightarrow$ `0`/`u`	`C` $\rightarrow$ `1`/`n`	`5` $\rightarrow$ `0`/`n`	`A` $\rightarrow$ `1`/`u`
`7` $\rightarrow$ `0`/`x`, `u`/`5`, `n`/`3`	`B` $\rightarrow$ `0`/`A`, `u`/`-`, `1`/`3`	`D` $\rightarrow$ `0`/`C`, `n`/`-`, `1`/`5`	`E` $\rightarrow$ `u`/`C`, `n`/`A`, `1`/`x`

For instance, if a bit is specified as `?`, we test whether the system is still compatible when it is restricted to `-` and to `x`, respectively. If one of the systems becomes incompatible, we can turn the `?` constraint into `x` or `-`, accordingly. If both are still compatible, we then try to restrict the `?` bit to `3` and `C`, and try all the available splits.

This will be repeated with the S-systems corresponding to each operation in the cipher. We can not apply this strategy to bigger chunks because the resulting system would be too large. Still, the constraints found in one system will be given as input to other systems involving the same variable, and can generate new constraints. The technique will discover *necessary* constraints, and output a characteristic more precise than the input characteristic.

This can also be combined with more global techniques such as Section 2.3 of [7]. or the "Complete Condition Check" of [15]. When we a constraint is split into two subsets, we can look for contradictions by running the propagation algorithm on the full path, instead of running it on a single operation. However, this becomes very expensive for large systems and it can take hours to try to split each constraint. In this work, we focus on discovering local conditions efficiently, and we leave the analysis of less local techniques for future work.

All this can be implemented quite efficiently using automata to solve S-systems. If we build deterministic decision automata, we can test whether a system is compatible with only n table access. This approach is very similar to the technique used in [6], and explained in more details in [21] and [18]. The main difference is that we iterate over all possible choices for the variables only when building the automaton, not when using it. In previous work, a similar result is achieved by caching the results of the computations.

4 Multi-bit Constraints

In this work, we extend this framework by considering constraints on several consecutive bits, instead of strictly bitwise constraints. This allows to express some conditions that occur naturally when considering carry extension, such as $x^{[i]} = x^{[i-1]}$. Two-bit conditions have already been proposed in [15], but they are treated separately from the main characteristic. In particular, two-bit conditions

Table 3. New 1.5-bit constraints

$(x \oplus x', x \oplus 2x, x)$:		$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
`=`	$x' = x = 2x$	✓	✓	-	-	-	-	-	-
`!`	$x' = x \neq 2x$	-	-	✓	✓	-	-	-	-
`<`	$x' \neq x = 2x$	-	-	-	-	✓	✓	-	-
`>`	$x' \neq x \neq 2x$	-	-	-	-	-	-	✓	✓

are not used to deduce further constraints through the propagation algorithm. In our work, multi-bit constraints can only deal with consecutive bits of a variable, but they are part of the characteristic, and they can be propagated efficiently.

1.5-bit Constraints. First, we consider constraints on pairs of consecutive bits. Intuitively, this is used to capture the fact that in a carry chain, even if we don't known the sign of the modular difference, we know that the active bits all have the same sign, except the final one. For instance, if we have `---x`$\boxplus$`----` $\rightarrow$ `-xxx`, we know that output difference must be either `-nuu` (if the input difference is `---n`) or `-unn` (if the input difference is `---u`). We can capture this behaviour using constraints that link the sign of an active bit to the sign of the previous bit. In our implementation, we introduce a set of 16 constraints described in Table 3 and 1: `?`, `-`, `x`, `0`, `u`, `n`, `1`, `#`, `3`, `C`, `5`, `A`, `=`, `!`, `<`, `>`. For instance, the symbol `<` means that the current bit is active, and that bit i of x is equal to bit i of $2x$, *i. e.* to bit $i-1$ of x — this can be written as $x'^{[i]} \neq x^{[i]} = x^{[i-1]}$, and it appears in the middle of carry chain. The situation of a carry extension with an unknown sign as in `---x`$\boxplus$`----` $\rightarrow$ `-xxx` can now be written more accurately as `-><x`.

The constraints of Table 3 are written as subsets of $(x^{[i]}, x'^{[i]}, x^{[i-1]})$; we call them 1.5-bit constraints because we use $x^{[i-1]}$ but we do not use $x'^{[i-1]}$.

2-bit Constraints. The 1.5-bit constraints are quite efficient to capture information about the carries when the xor difference is known. However, when the xor difference is not known *a priori*, we still loose a lot of information. To overcome this problem, we considered the full set of 2^{16} possible constraints on $(x^{[i]}, x'^{[i]}, x^{[i-1]}, x'^{[i-1]})$, and we discovered an important property: they can restrict the pair (x, x') to *exactly* the set of values with any given modular difference. More precisely, this is achieved using the 10 constraints of Table 4. We found this set of constraints experimentally, by testing all 8-bit differences.

This is an important result because it allows to express the modular difference using only local constraints. Local constraints can easily go through rotations, and can be expressed as S-functions. Therefore we can compute the probability of a differential characteristic expressed in this way, and we can propagate these constraints automatically.

We denote the first four constraints as `U`, `V`, `N` and `M`; the remaining six can be obtained by combining previous constraints. The most important constraints are

Table 4. 2-bit constraints sufficient to describe exactly the modular difference

$(2x, 2x', x \oplus x')$:	$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
	`-0`	`x0`	`-u`	`xu`	`-n`	`xn`	`-1`	`x1`
`U` $\equiv$ {`--`, `-u`, `xn`}	✓	-	-	✓	✓	-	✓	-
`V` not `U`	-	✓	✓	-	-	✓	-	✓
`N` $\equiv$ {`--`, `-n`, `xu`}	✓	-	✓	-	-	✓	✓	-
`M` not `N`	-	✓	-	✓	✓	-	-	✓
$\equiv$ `x0`	-	✓	-	-	-	-	-	-
$\equiv$ `-0`	✓	-	-	-	-	-	-	-
$\equiv$ `x-`	-	✓	-	-	-	-	-	✓
$\equiv$ `--`	✓	-	-	-	-	-	✓	-
$\equiv$ `Ux`	-	-	-	✓	✓	-	-	-
$\equiv$ `Nx`	-	-	✓	-	-	✓	-	-

Table 5. New 2.5-bit constraints

$(x \oplus x', x \oplus 2x, x)$:	$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
$2x \oplus 2x', 4x \oplus 2x$:	$(0,0)/(0,1)/(1,0)/(1,1)$							
`X` carry chain	✓	✓	✓	✓	- / ✓	- / ✓	- / ✓	- / ✓
`U` `u` carry	✓ / ✓	✓ / -	✓ / -	✓ / ✓	- / -	- / ✓	- / ✓	- / -
`N` `n` carry	✓ / -	✓ / ✓	✓ / ✓	✓ / -	- / ✓	- / -	- / -	- / ✓
`/` `x` carry	✓/✓/-/✓	✓/✓/-/✓	✓/✓/-/✓	✓/✓/-/✓	-/-/✓/-	-/-/✓/-	-/-/✓/-	-/-/✓/-
`\` `X` minus `/`	-/-/✓/-	-/-/✓/-	-/-/✓/-	-/-/✓/-	-/-/-/✓	-/-/-/✓	-/-/-/✓	-/-/-/✓

the one denoted as `U` and `N`: they can capture the carry extension of a positive (resp. negative) modular difference. For instance a modular difference of $+1$ can be realized with 4-bit words as `---u`, `--un`, `-unn`, `unnn` or `nnnn`, depending on the carry extension. For each of the potential carry bits (1–3), we can see that the difference pattern of bit i and bit $i-1$ is always one of `--`, `-u`, `un`, or `nn`. Reciprocally, if bits 1–3 follow these patterns, then the full difference has to be one of the previous patterns, and the modular difference will be $+1$. The `U` constraint correspond to these patterns.

In our implementation, we only use the `U` and `N` constraints, which are sufficient to express sparse modular differences.

2.5-bit Constraints. To obtain an efficient technique to study differential characteristics in ARX constructions, we want to combine the results of the 1.5-bit constraints, and the 2-bit constraints. On the one hand, the 1.5-bit constraints are constructed from the 1-bit constraints in order to capture information about the carry when the sign of the difference is not known. On the other hand, the 2-bit constraints can capture exactly the modular difference, but we need to know the sign of the difference. We now introduce constraints to capture the modular difference when the sign is not known.

Table 6. Comparison of the constraint sets. We show how simple difference sets can be encoded with our constraints, and the number of pairs allowed by each constraint.

Diff, carry	1-bit cstr.	1.5-bit cstr.	2-bit cstr.	2.5-bit cstr.
$+1$, k-bit (2^{n-k})	`-unnn` (2^{n-k})	`-unnn` (2^{n-k})	`-unnn` (2^{n-k})	`-unnn` (2^{n-k})
± 1, k-bit (2^{n-k+1})	`-xxxx` (2^n)	`->><x` (2^{n-k+1})	`->><x` (2^{n-k+1})	`->><x` (2^{n-k+1})
$+1$, any (2^n)	`????x` (2^{2n-1})	`????x` (2^{2n-1})	`UUUUx` (2^n)	`UUUUx` (2^n)
± 1, any (2^{n+1})	`????x` (2^{2n-1})	`????x` (2^{2n-1})	`XXXXx` $(2^n \times n)$	`///Xx` (2^{n+1})

Following the analysis of the 2-bit constraints, we study the patterns created by a carry extension with an unknown sign. Using the 1.5-bit constraints, we can see that the constraints of bits i and $i-1$ are either `--`, `->`, or `x<`. Reciprocally, if all the bits follow these patterns, this result in a valid carry extension. We denote the corresponding set of possibilities for $(x^{[i]}, x'^{[i]}, x^{[i-1]}, x'^{[i-1]}, x^{[i-2]})$ as `/`.

As shown in Table 5, we introduce the following new constraints: `X` $\equiv$ {`--`, `-x`, `xx`}, `U` $\equiv$ {`--`, `-u`, `xn`}, `N` $\equiv$ {`--`, `-n`, `xu`}, `/` $\equiv$ {`--`, `->`, `x<`}, `\` $\equiv$ {`-<`, `x>`}. For efficiency reasons, we keep a set of only 16 constraints by removing the less useful ones: `?`, `-`, `x`, `0`, `u`, `n`, `1`, `=`, `!`, `<`, `>`, `X`, `U`, `N`, `/`, `\`.

4.1 Comparison

To compare the sets of constraints, we show how they can be used in simple situations in Table 6. We consider 4 situations, were we describe a set of pairs with a modular difference of ± 1:

- First, we assume that we know the sign of the difference, and the length of the carry (*e.g.* `-----u` $\boxplus$ `------` $\rightarrow$ `-xxxxx`). In this case all the constraints systems give an optimal characterization of the set of allowed pairs.
- Second, we assume that we don't known the sign of the difference, but we know the length of he carry (*e.g.* `-----x` $\boxplus$ `------` $\rightarrow$ `-xxxxx`). In the case, we need constraints on 1.5 bits to optimally capture the relations in the carry-extended bits.
- Third, we assume that we know the sign of the difference, but we don't know the length of the carry (*e.g.* `-----u` $\boxplus$ `------` $\rightarrow$ `??????`). In this situation, the 2-bit constraints can express precisely the modular difference.
- Finally, we assume that we don't know the sign of the difference, nor the length of the carry (*e.g.* `-----x` $\boxplus$ `------` $\rightarrow$ `??????`). Here, we need constraints on 2.5 bits to restrict the set of pairs optimally using relations between the bits.

4.2 Use as S-sytems

We also denote the new sets of constraint by δ. Since the definition of δ only involves bitwise operation and left-shift by a few bits, $\delta x = \Delta$ can by written as a S-system, similar to System (2). We can use the tools of Section 2 to compute the probability of a characteristic specified with the new constraints, and to propagate the new constraints, by build the automata associated with the systems

of each operation, as given in (3). These automata are quite large, because the state of the automata has to include the values of $x^{[i-1]}$, $x'^{[i-1]}$, and $x^{[i-2]}$.

In practice, we implemented the 1.5-bit constraints and the 2.5-bit constraints. With the 1.5-bit constraints, we have 5 bits of state for the S-system of the addition, but the transition automaton only reaches 16 different states. When using the powerset construction to build a deterministic decision, we obtain 12929 states, and the full table takes 102MB. With the 2.5-bit constraints, we have 11 bits of state, and the transition automaton reaches 160 different states (we cannot build a deterministic decision automaton in this case).

We could easily include more constraints in our framework, but this set of symbol is quite expressive, and a larger set of constraints would result in larger tables. We will see that those constraints give good results in practice. Moreover, we note that many cases can be expressed using the constraints of two consecutive bits. For instance, the constraint $x^{[i]} = x'^{[i]} = x^{[i-1]} = 0$ cannot be expressed in Table 3, but it will be coded with constraint `=` for bit i, and constraint `3` for bit $i-1$ (if some more information is known for bit $i-1$, it will become `0` or `u`).

When we deal with a rotation, we have to relax the constraints slightly if the multi-bit constraints are broken by the rotation. For a rotation of i bits to the right, if $\Delta_x^{[i]}$ is one of `=`, `!`, `<` or `>`, it will be relaxed to `-`, `-`, `x` and `x`, respectively.

4.3 Propagation of Constraints

To propagate the new constraints, we need to define how to split the new constraints. We use the following splits for the 1.5-bit constraints:

`?` → `-/x, 3/C, 5/A`	`-` → `0/1, =/!`	`x` → `u/n, </>`	
`3` → `0/u`	`C` → `1/n`	`5` → `0/n`	`A` → `1/u`
`=` → `0/1`	`!` → `0/1`	`>` → `u/n`	`<` → `u/n`

For the 2.5-bit constraints, some useful subsets are not included in the 16 constraints, but can be obtained by restricting both $\Delta_x^{[i]}$ and $\Delta_x^{[i-1]}$. We use the following splits:

`?` → `-/x, X/x-`	`X` → `U/Nx, N/Ux, -/xx, //\`	`N` → `-/xu`
`\` → `-</x>`	`/` → `U/Nx, N/Ux, -/x<`	`U` → `-/xn`

This approach is quite efficient. As an example, let us consider this system:

$\delta x =$ `x--x`	$\delta y =$ `----`	$z = x \boxplus y$	
$\delta u =$ `---x`	$\delta v =$ `----`	$z = u \boxplus v$	$\delta z =$ `-???`.

It is easy to see that this system is incompatible when considering modular differences: the difference in $x \boxplus y$ is $\pm 8 \pm 1$, while the difference in $u \boxplus v$ is ± 1. However, when using only the xor difference, or the constraints of [6], this system seems to be compatible, and constraint propagation gives $\delta z =$ `-xxx`. Using our new constraints, the algorithm can further deduces $\delta z =$ `-<<x` from the first

Table 7. Experiments with a few rounds of a 4-bit Skein. We give the number of input/output differences accepted by each technique, and the ratio of false positive.

	4 rounds (total: 2^{32})		6 rounds (sparse[1])	
Method	Accepted	Fp.	Accepted	Fp.
Exhaustive search	35960536 ($2^{25.1}$)	0	427667 ($2^{18.7}$)	0
2.5-bit constraints	40820032 ($2^{25.3}$)	0.13	746742 ($2^{19.5}$)	0.7
1.5-bit constraints	40820032 ($2^{25.3}$)	0.13	1372774 ($2^{20.4}$)	2.2
1-bit constraints	43564288 ($2^{25.4}$)	0.21	1762857 ($2^{20.7}$)	3.1
Checking additions independently	56484732 ($2^{25.8}$)	0.57		

[1] Weight 4 differences. The total number of input/output differences is $\binom{24}{4}\binom{24}{4} \approx 2^{26.75}$

addition and $\delta z =$ `-><x` from the second addition, and the incompatibility is detected. Moreover, the incompatibility can be detected without specifying the difference in z beforehand using the 2.5 bit constraints.

4.4 Comparison with Previous Works

To compare the efficiency of the constraints, we did some experiments with reduced versions of Skein. We test a set of input and output xor differences, and we compare several methods to detect if the differences are compatible. We use small versions so that we can find exact results with exhaustive search. We verify that no false-negative are found, and we compare how many false-positive are found with each technique.

First we use a reduced Skein with two rounds and 4 words of 4 bits each. We note that for a two-round Skein, all the intermediary xor difference can be computed from the input and output xor differences; therefore we have a full xor differential characteristic. As a reference point, we can check whether each non-linear operation has a non-zero probability. Our result in Table 7 show that the assumption of independence of the operations can be quite flawed: we found many paths where each operation has a non-zero probability, but no pair can satisfy the differential. This motivates the use of more advanced constraints in order to extract information from one operation and combine it with another operation. We also see that our 1.5-bit constraints can detect more problems that the 1-bit constraints of [6]. In this setting the 2.5-bit constraints are no better than the 1.5-bit constraints because the xor differences are all known.

We also did experiments with a reduced Skein with three rounds and 4 words of 6 bits each. We only use sparse differences (less than 4 active bits in the input and output), because the full space is too large to be exhausted in practice. Our results are given in Table 7, and show that in this setting, the 2.5-bit constraints reduces the number of false positives threefold over the 1.5-bit constraints. The 2.5-bit constraints provide much better results than previous works when the xor difference is not known beforehand.

4.5 Description of Some Case of Incompatibility

We have developed a graphical tool that can display such a characteristic, and allows the user to easily modify the characteristic by adding and removing constraints. The tool can automatically propagate the new constraints, and show incompatibilities if there are some.

We have studied published differential trails with this tool and we found problems in several of them. It seems that many characteristics following a natural construction, and seemingly valid when verified manually, are in fact incompatible. We will now describe some of the patterns that can lead to unexpected problem.

Problems with Modular Addition. A simple class of problems is related to the modular additions when using xor differences. Techniques to check the validity of these operations are well known [13,19], but in some cases the results are somewhat unexpected. In particular, the valid differences are quite constrained in the least significant bit, because the incoming carry is fixed to zero. For instance the following path is built with a simple linearization, but it is in fact incompatible:

$$\delta a = \texttt{---x} \qquad \delta b = \texttt{---x} \qquad \delta c = \texttt{---x}$$
$$x = a \boxplus b \boxplus c \qquad \delta x = \texttt{---x}.$$

More generally, some pattern which seem valid when studied with a signed difference are in fact incompatible. The characteristic used in a recent near-collision attack against Skein [29] contains a pattern similar to this one[4]:

$$\delta a = \texttt{--xxxxx-} \qquad \delta b = \texttt{---xx---}$$
$$x = a \boxplus b \qquad \delta x = \texttt{-xxxx-x-}.$$

This seems valid when considering signed differences: the difference should be ± 2 in a, ± 8 in b, and $\pm 2 \pm 8$ in x. In fact, this does not have any solution, and it does not seem easy to modify the characteristic of [29] to obtain a valid attack.

Problems with Carry Extensions. Carry extensions in modular additions generate constraints between consecutive bits which can be detected with our framework. For instance, let us consider the following simple path:

$$\delta a = \texttt{-xx---} \qquad c = a \boxplus b \qquad c' = c \ggg 2 \qquad u = c' \boxplus d$$
$$\delta b = \texttt{xxx---} \qquad \delta c = \texttt{------} \qquad \delta d = \texttt{---xx-} \qquad \delta u = \texttt{---xx-}.$$

The first addition generate a constraint $c^{[4]} \neq c^{[3]}$ (*i. e.* $\delta c = \texttt{-!----}$), and the second addition generate a constraints $c'^{[2]} = c'^{[1]}$ (*i. e.* $\delta c' = \texttt{---=--}$). Obviously these constraints are contradictory through the rotation. In this example the problem will be detected by our new constraints, but not when looking at each operation individually, or using the single-bit constraints of [6].

[4] This can be found at round 20, in the addition $c_{20} = c_{19} \boxplus d_{19}$, with the following xor-differences: $\Delta c_{19} = \texttt{0x020030a0000f80a0}$, $\Delta d_{19} = \texttt{0xf8f87ca007f7c7a7}$, $\Delta c_{20} = \texttt{0x7ef8f50001104501}$.

5 Constraints for the Analysis of Boomerang Attacks

We also study differential characteristics in the context of boomerang attacks. The traditional approach is to specify each characteristic separately, and to assume that they are all independent. In this work, we consider a boomerang characteristic mostly as collection of constraints for the top characteristics and the bottom characteristics.

Let x be some internal state variable, and $x^{(0)}$, $x^{(1)}$, $x^{(2)}$, $x^{(3)}$ be the corresponding variables in a boomerang quartet. A boomerang property is built by specifying a top trail for $(x^{(0)}, x^{(2)})$ and $(x^{(1)}, x^{(3)})$, and a bottom trail for $(x^{(0)}, x^{(1)})$ and $(x^{(2)}, x^{(3)})$. For more generality, we allow the two characteristics to be different in each case (*e.g.*, the signs might be different). The top trail will be mostly unconstrained for the bottom part of the cipher, while the bottom trail will be mostly unconstrained for the top part.

Unfortunately, the hypothesis of independence might be wrong in practice, and we can find paths that are impossible to satisfy simultaneously, as shown by Murphy [20]. In fact, this kind of problem seem to be quite common with ARX designs, as shown in the case of HAVAL [22], SHA-256 [2], or Skein [11]. To capture this kind of dependency, we use constraints on *quartets* of variables, instead of constraints on pairs of variables. We can not use the full set of 2^{16} constraints, because the resulting system is too large, but we use a set of 81 constraints given in Table 8 to specify the xor difference in each of the four sides of the quartet. For (i, j) in $\{(0,1),(2,3),(0,2),(1,3)\}$, we restrict $x^{(i)} \oplus x^{(j)}$ to 0 or 1, or leave it unrestricted. Note that some constraints are actually contradictory[5] or redundant[6], but this uniform set is much easier to work with than a reduced set without the extra constraints.

We use three different kinds of S-systems to propagate constraints in a boomerang characteristic:

1. systems with multi-bit constraints and non-linear operations in each individual path, following System (3) (for (i, j) in $\{(0,1),(2,3),(0,2),(1,3)\}$):
$$\delta(x^{(i)}, x^{(j)}) = \Delta_x^{i,j} \quad \delta(y^{(i)}, y^{(j)}) = \Delta_y^{i,j} \quad \delta(z^{(i)}, z^{(j)}) = \Delta_z^{i,j}$$
$$x^{(i)} \boxplus y^{(i)} = z^{(i)} \quad x^{(j)} \boxplus y^{(j)} = z^{(j)};$$
2. systems with quartet constraints and non-linear operations:
$$\delta(u^{(0)}, u^{(1)}, u^{(2)}, u^{(3)}) = \Delta_u^{0,1,2,3}, \qquad \text{for all } u \text{ in } \{x, y, z\}$$
$$x^{(i)} \boxplus y^{(i)} = z^{(i)}, \qquad \text{for all } i \text{ in } \{0, 1, 2, 3\}$$
3. systems with multi-bit constraints linking the four variables of a quartet:
$$\delta(x^{(0)}, x^{(2)}) = \Delta_x^{0,2} \qquad \delta(x^{(1)}, x^{(3)}) = \Delta_x^{1,3} \qquad \textit{(Top path)}$$
$$\delta(x^{(0)}, x^{(1)}) = \Delta_x^{0,1} \qquad \delta(x^{(2)}, x^{(3)}) = \Delta_x^{2,3}. \qquad \textit{(Bottom path)}$$

[5] *e.g.* `---x` means $x^{(0)} \oplus x^{(1)} = 0$, $x^{(2)} \oplus x^{(3)} = 0$, $x^{(0)} \oplus x^{(2)} = 0$, $x^{(1)} \oplus x^{(3)} = 1$, which is impossible.

[6] *e.g.* `---?` and `----` allow the same values for the $x^{(i)}$'s.

Table 8. New boomerang constraints

$(x^{(0)} \oplus x^{(1)}, x^{(2)} \oplus x^{(3)}, x^{(0)} \oplus x^{(2)}, x^{(1)} \oplus x^{(3)})$:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
`????`	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`x???`	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓
`-???`	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
`?x??`	-	-	-	-	✓	✓	✓	✓	-	-	-	-	✓	✓	✓	✓
`xx??`	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓
`----`	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

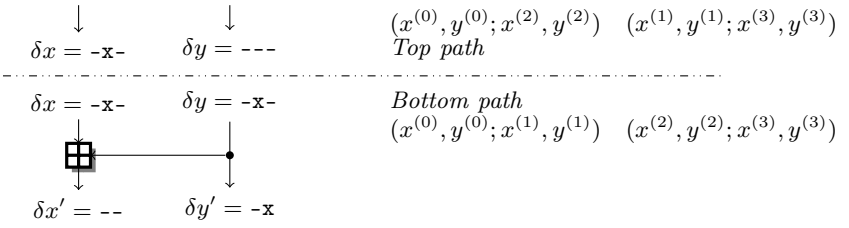

Fig. 1. Example of incompatible characteristics

5.1 Incompatibility in Boomerang Characteristics

We found that some very simple patterns can lead to incompatibilities. Figure 1 gives an example of a pattern that results in incompatible characteristics. If a quartet follows these characteristics, the middle bit of the variables has to satisfy:

$$x^{(0)} \oplus x^{(2)} = x^{(1)} \oplus x^{(3)} = \mathtt{1} \quad y^{(0)} \oplus y^{(2)} = y^{(1)} \oplus y^{(3)} = \mathtt{0} \quad \textit{(Top path)}$$
$$x^{(0)} \oplus x^{(1)} = x^{(2)} \oplus x^{(3)} = \mathtt{1} \quad y^{(0)} \oplus y^{(1)} = y^{(2)} \oplus y^{(3)} = \mathtt{1} \quad \textit{(Bottom path)}$$
$$x^{(0)} \boxplus y^{(0)} = x^{(1)} \boxplus y^{(1)} \qquad x^{(2)} \boxplus y^{(2)} = x^{(3)} \boxplus y^{(3)}$$

We can assume that $x^{(0)} = 0$, and deduce $x^{(1)} = 1$, $x^{(2)} = 1$, $x^{(3)} = 0$. Since the difference in $(y^{(0)}, y^{(1)})$ must cancel the difference in $(x^{(0)}, x^{(1)})$, we have $y^{(0)} = 1$, $y^{(1)} = 0$, and we can deduce $y^{(2)} = 1$, $y^{(3)} = 0$. But the difference in $(y^{(2)}, y^{(2)})$ can not cancel the difference in $(x^{(2)}, x^{(3)})$. A more detailed analysis shows that this pattern can lead to incompatibilities even if we allow some incoming carries.

This pattern seem to appear very frequently when using linearized characteristics in ARX designs.

5.2 Application

We used our tools to verify several boomerang attacks in the literature, and found some attack using incompatible paths.

Blake-256. First, we studied the boomerang attacks on Blake from Biryukov *et.al* in [3]. When looking at the paths used for the attacks on 7 and 8 round of the keyed permutation, our tool detects an incompatibility. More precisely, when starting from a middle quartet with the specified differences, and going backward through G_3, it is impossible to get the specified difference simultaneously in both paths. We verified experimentally that we could not find such quartets, even with significantly more trials than predicted under the assumption that the paths are independent.

With the help of the authors of [3], we found out an alternative path that give a valid boomerang attack. More precisely we modify the top path by using a difference on bit 25 instead of 31, and rotating all the difference patterns. We verified experimentally that this leads to a valid attack, but the cost of the attack becomes higher than reported in [3].

Similarly, for the compression function attacks, our tool detects that the path used for the 6.5 and 7-round attacks is invalid. We found that this can corrected by modifying the top path to use differences on bits 4 and 20 instead of 15 and 31.

Skein-512. We also used our tool to study the boomerang attacks on Skein. We start with only the linearized (or almost linearized) xor differential characteristics for rounds 12–16 and 16–20, with the key addition in between to provide extra freedom, and we use our tool to propagate the constraints. We found that the following paths lead to contradictions:

- The paths for the 32-round attack of [5];
- The paths for the 33- and 34-round attack of [5];
- The paths for the attack of[1], based on the old rotation constants, and inverse permutations; as well as a modified version using the correct permutations.

In each case, our tool detect the contradiction automatically. More recently, a new path has been proposed [30], and a middle quartet was given to show that the paths are compatible.

6 Conclusion

In this paper, we study differential characteristics in ARX constructions. We extend the framework of de Cannière and Rechberger with new constraints. First we introduce multi-bit constraints that can be propagated more accurately through modular addition. We show that a set of 2-bit constraints can express exactly the modular difference of a pair of variables, and describe a reduced set of 2.5-bit constraints that can express the modular difference in simple cases and can also capture the carry extensions of an unsigned difference. Second, we introduce new quartet constraints to work with boomerang attacks.

We provide experimental results showing that our constraints can automatically detect several cases of incompatibility in differential characteristics undetected by previous techniques; and we point out several published attacks that

turn out to be invalid. We show that some paths can in fact be incompatibile; this shows the importance of verifying differential attacks.

We hope that the tools will be useful to other cryptanalysts, and they are available at `http://www.di.ens.fr/~leurent/arxtools.html`.

Acknowledgement. We would like to thank the anonymous reviewers of Asiacrypt for their insightful comments. We would also like to thank the authors of [3] for helping us verify the problem with their attack, and finding an alternative path.

Gaëtan Leurent is supported by the AFR grant PDR-10-022 of the FNR Luxembourg.

References

1. Aumasson, J.-P., Çalık, Ç., Meier, W., Özen, O., Phan, R.C.-W., Varıcı, K.: Improved Cryptanalysis of Skein. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 542–559. Springer, Heidelberg (2009)
2. Biryukov, A., Lamberger, M., Mendel, F., Nikolić, I.: Second-Order Differential Collisions for Reduced SHA-256. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 270–287. Springer, Heidelberg (2011)
3. Biryukov, A., Nikolić, I., Roy, A.: Boomerang Attacks on BLAKE-32. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 218–237. Springer, Heidelberg (2011)
4. Chabaud, F., Joux, A.: Differential Collisions in SHA-0. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 56–71. Springer, Heidelberg (1998)
5. Chen, J., Jia, K.: Improved Related-Key Boomerang Attacks on Round-Reduced Threefish-512. In: Kwak, J., Deng, R.H., Won, Y., Wang, G. (eds.) ISPEC 2010. LNCS, vol. 6047, pp. 1–18. Springer, Heidelberg (2010)
6. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 1–20. Springer, Heidelberg (2006)
7. Grechnikov, E.A.: Collisions for 72-step and 73-step sha-1: Improvements in the method of characteristics. Cryptology ePrint Archive, Report 2010/413 (2010), `http://eprint.iacr.org/`
8. Fouque, P.A., Leurent, G., Nguyen, P.: Automatic Search of Differential Path in MD4. ECRYPT Hash Worshop – Cryptology ePrint Archive, Report 2007/206 (2007), `http://eprint.iacr.org/`
9. Fouque, P.-A., Leurent, G., Nguyen, P.Q.: Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 13–30. Springer, Heidelberg (2007)
10. Klimov, A., Shamir, A.: A New Class of Invertible Mappings. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 470–483. Springer, Heidelberg (2003)
11. Leurent, G., Roy, A.: Boomerang Attacks on Hash Function Using Auxiliary Differentials. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 215–230. Springer, Heidelberg (2012)
12. Leurent, G., Thomsen, S.S.: Practical Near-Collisions on the Compression Function of BMW. In: Joux, A. (ed.) FSE 2011. LNCS, vol. 6733, pp. 238–251. Springer, Heidelberg (2011)

13. Lipmaa, H., Moriai, S.: Efficient Algorithms for Computing Differential Properties of Addition. In: Matsui, M. (ed.) FSE 2001. LNCS, vol. 2355, pp. 336–350. Springer, Heidelberg (2002)
14. Manuel, S.: Classification and generation of disturbance vectors for collision attacks against SHA-1. Des. Codes Cryptography 59(1-3), 247–263 (2011)
15. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 288–307. Springer, Heidelberg (2011)
16. Mendel, F., Rechberger, C., Schläffer, M.: MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 144–161. Springer, Heidelberg (2009)
17. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and `Grøstl`. In: Dunkelman, O. (ed.) FSE 2009. LNCS, vol. 5665, pp. 260–276. Springer, Heidelberg (2009)
18. Mouha, N., De Cannière, C., Indesteege, S., Preneel, B.: Finding Collisions for a 45-Step Simplified HAS-V. In: Youm, H.Y., Yung, M. (eds.) WISA 2009. LNCS, vol. 5932, pp. 206–225. Springer, Heidelberg (2009)
19. Mouha, N., Velichkov, V., De Cannière, C., Preneel, B.: The Differential Analysis of S-Functions. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 36–56. Springer, Heidelberg (2011)
20. Murphy, S.: The Return of the Cryptographic Boomerang. IEEE Transactions on Information Theory 57(4), 2517–2521 (2011)
21. Peyrin, T.: Analyse de fonctions de hachage cryptographiques. PhD thesis, University of Versailles (2008)
22. Sasaki, Y.: Boomerang Distinguishers on MD4-Family: First Practical Results on Full 5-Pass HAVAL. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 1–18. Springer, Heidelberg (2012)
23. Schläffer, M., Oswald, E.: Searching for Differential Paths in MD4. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 242–261. Springer, Heidelberg (2006)
24. Stevens, M., Lenstra, A.K., de Weger, B.: Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 1–22. Springer, Heidelberg (2007)
25. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D.A., de Weger, B.: Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 55–69. Springer, Heidelberg (2009)
26. Wang, G., Keller, N., Dunkelman, O.: The Delicate Issues of Addition with Respect to XOR Differences. In: Adams, C., Miri, A., Wiener, M. (eds.) SAC 2007. LNCS, vol. 4876, pp. 212–231. Springer, Heidelberg (2007)
27. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
28. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)
29. Yu, H., Chen, J., Ketingjia, W.X.: Near-Collision Attack on the Step-Reduced Compression Function of Skein-256. Cryptology ePrint Archive, Report 2011/148 (2011), `http://eprint.iacr.org/` (last revised March 31, 2011)
30. Yu, H., Chen, J., Wang, X.: The Boomerang Attacks on the Round-Reduced Skein-512. In: SAC (2012)