ARXtools: A toolkit for ARX analysis

Gaëtan Leurent

University of Luxembourg

Abstract. ARX designs are quite popular, but analysis of these schemes is hard because of their bit-level structure. This makes tasks such as building a differential characteristic very tedious. In this document, we present the result of our work in trying to help with analysis of ARX designs. We describe a toolkit to study ARX constructions, and present some preliminary results obtained with it.
This is still a work in progress, but we wont to release the tools so that they can be used by anyone, and we hope this will will lead to new results. In this document, we give some explanation of how the tool works, and the full toolkit will be available for download.
We did not manage to automate the search of differential paths, but our tools can help the cryptanalyst to build paths, and help uncover incompatibilities in published paths. Our main result so far show that differential characteristics used in some recent boomerang attacks are in fact incompatible, and those attacks will not work as expected.

Keywords: ARX, tools, toolkit, FSM, automaton, hash function.

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.

The ARX design philosophy is opposed to S-Box based designs such as the AES. Analysis of S-Box based design usually happen at the word-level, and differential characteristic are easy to build but hard to use. For ARX designs, the analysis is done on a bit-level, and finding good differential characteristics remains an important challenge.

In this paper, we describe several useful tools for the analysis of ARX systems, and show some preliminary results obtained with these tools. Our hope is that those tools are general enough to be useful to other cryptographers for the analysis of various primitives.

The first tool is designed for the analysis of S-functions. In particular, it can be used to study various properties of systems of Additions, Xors, and bitwise functions (without rotations). This tool is quite similar to the tool developed by Mouha *et. al* in [MVCP10], but we have completely automated the construction of the transition matrix. As an example we show an application of this tool to the analysis of IDEA.

The second tool is designed to study differential properties of ARX designs. When we specify the differences for each internal variables, we can easily check that the differences are consistent across the rotations, and use the first tool to study differential properties of the remaining operations. We can also propagate constraints by detecting incompatibilities. This approach

is similar to the approach of de Cannière and Rechberger [dCR06], but we use a different set of constraints that can uncover more incompatibilities. Our new set of constraint allow more information to be extracted automatically when carries are used in modular additions. However we have not yet been able to build differential characteristics automatically as in [dCR06].

The third tool is 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 then automatically propagate the new constraints, and if a contradiction is found, it will highlight the specific constraints leading to the contradiction. We used this tool to study attacks on Blake and Skein, and we show that several characteristics used in previous works are in fact incompatible.

The tools and an up-to-date version of this paper will be available for download at the following address: `http://www.di.ens.fr/~leurent/arxtools.html`

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, or more precisely, as an S-function [MVCP10]. We use the following definitions:

T-function A T-function on n-bit words is a function from $(\{0,1\}^n)^k$ to $(\{0,1\}^n)^l$ with the following property:

For all t, t bits of the output words can be computed from t bits of the input words.

S-function An A-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 output words and the state $S[t] \in \mathcal{S}$ can be computed from bit t of the input words, and the state $S[t-1]$.

In practice, our analysis will be linear in the number of states, and the number of state 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. 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.

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

$$x' \boxminus x = \Delta\,; \qquad x' \oplus x = \delta$$

or equivalently

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

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 [LT11].

Other interesting systems include the S-functions used in [MVCP10] to compute the adp of the addition:

$$(x \oplus \Delta_x) \boxplus (y \oplus \Delta_y) = (x \boxplus y) \oplus \Delta_z \tag{2}$$

and the xdp of the xor:

$$(x \boxplus \Delta_x) \oplus (y \boxplus \Delta_y) = (x \oplus y) \boxplus \Delta_z, \tag{3}$$

where the parameters are $\Delta_x, \Delta_y, \Delta_z$.

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.[1]

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 [MVCP10].

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 simply 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 operation, 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. Figure 1 shows an example of such automaton for System (1).

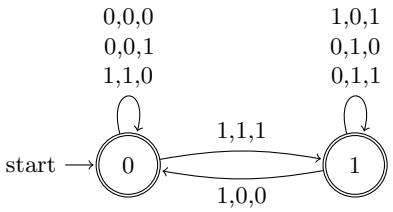
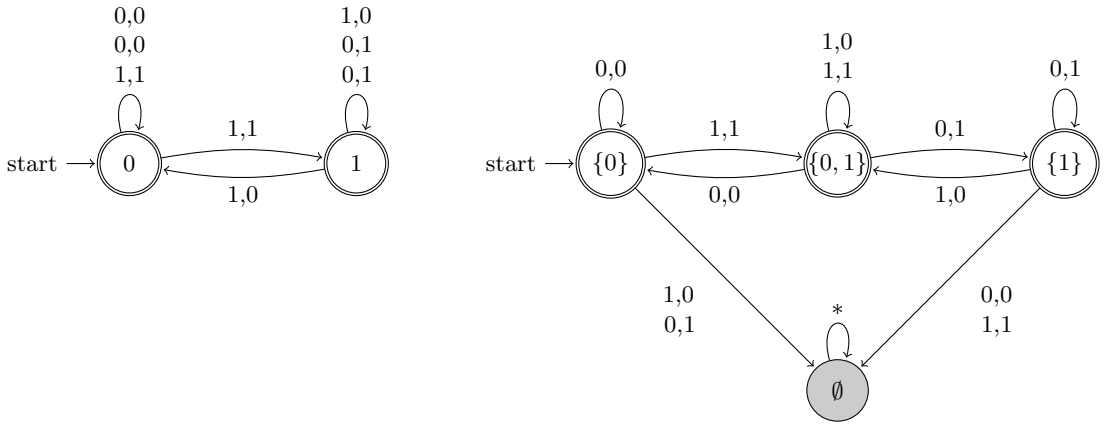


Fig. 1. Carry transitions for $x \oplus \Delta = x \boxplus \delta$. The edges are indexed by Δ, δ, x

[1] *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.

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, as shown in Figure 2.



Non-deterministic automaton. Determinized automaton.

Fig. 2. Decision automaton for $x \oplus \Delta = x \boxplus \delta$. The edges are indexed by Δ, δ

This automaton can reveal a lot of information about the system of equation, as was shown in [LT11].

2.2 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 (6) will be written as `(V0^1)&(V1^1)&(P0^1)==0; (V0)&(V1^1)&(P1^1)==0; (V0^1)&(V1)&(P2^1)==0; (V0)&(V1)&(P3^1)==0`. 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 for a random x)
- A description of the solution set, from which we can efficiently iterate over the set of solutions

2.3 Application to IDEA

As an example of the type of problem that can be studied with this tool, we briefly discuss application to cryptanalysis of IDEA with the Biryukov-Demirci relation. The Biryukov-Demirci relation was introduced in [JPV04] and used in several attacks against IDEA [Jun05,BDK06,BDK07].

More recently it has been used for Meet-in-the-middle attacks [BDKS11]. This relation can be derived by considering two IDEA rounds, and two lines of computation involving X_2^i and X_3^i. For these lines we have:

$$\left((X_2^i \boxplus Z_2^i) \oplus (s^i \boxplus t^i)\right) \boxplus Z_3^{i+1} = X_2^{i+2} \oplus t^{i+1}; \tag{4}$$

$$\left((X_3^i \boxplus Z_3^i) \oplus t^i\right) \boxplus Z_2^{i+1} = X_3^{i+2} \oplus (s^{i+1} \boxplus t^{i+1}). \tag{5}$$

Previous works reduce those equations to the least significant bit and sum the two equations in order to eliminate t^i and t^{i+1}:

$$\mathrm{lsb}\left(X_2^i \oplus X_3^i \oplus Z_2^i \oplus Z_3^i \oplus s^i \oplus Z_2^{i+1} \oplus Z_3^{i+1}\right) = \mathrm{lsb}\left(X_2^{i+2} \oplus X_3^{i+2} \oplus s^{i+1}\right).$$

If one can compute the lsb of all the parameters X_2^i, X_3^i, Z_2^i, Z_3^i, s^i, Z_2^{i+1}, Z_3^{i+1}, X_2^{i+2}, X_3^{i+2}, and s^{i+1}, this relation gives one bit of filtering.

We can improve this filtering by considering the system of equation (4,5) where t^i and t^{i+1} are considered as unknowns. Many values of the parameters will result in incompatible systems, and we can use our tools to test whether the system is compatible for a given set of parameters. We can write this system as:

```
(P2^(P0+V0))+P3 == P4^V1; (P5^V0)+P6 == P7^(P1+V1)
```

We find that this results in a 5-bit filtering instead of the 1-bit filtering used in previous works. However, it seems difficult to exploit this improved filtering in a meet-in-the-middle attack because we need to know parameters from both sides in order to check whether they are compatible.

3 Study of Differential Characteristics

We now describe how to use the previous tool to study differential trails in ARX designs. Our approach is heavily inspired by the technique of de Cannière and Rechberger in [dCR06].

3.1 Defining Constraints

Let us first describe how de Cannière and Rechberger construct a differential characteristic in [dCR06]. For each bit of the state, they consider the value x and x' of that bit in the two related computations computation of $H(M)$ and $H(M')$, and they restrict the set of possibilities.

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. For instance, $\delta x =$ `x---nuuu` means that:

$$x^{[0]} = 0 \qquad x^{[1]} = 0 \qquad x^{[2]} = 0 \qquad x^{[3]} = 1$$
$$x'^{[0]} = 1 \qquad x'^{[1]} = 1 \qquad x'^{[2]} = 1 \qquad x'^{[3]} = 0$$
$$x^{[4]} = x'^{[4]} \qquad x^{[5]} = x'^{[5]} \qquad x^{[6]} = x'^{[6]} \qquad x^{[7]} \neq x'^{[7]}$$

In this case, it is equivalent to specifying the xor-difference and the modular difference: $x' \oplus x =$ `0b10001111` and $x' \boxminus x = 2^7 - 1$. In general this way of defining a differential path can capture the xor difference and the modular differential, like Wang's signed difference, but it can also describe some constraints that can not be expressed by these differences.

Table 1. Constraints used in [dCR06]

	(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`	$x = x' = 0$	-	-	-	✓
`#`	*incompatible*	-	-	-	-
`3`	$x = 0$	✓	✓	-	-
`5`	$x' = 0$	✓	-	✓	-
`7`		✓	✓	✓	-
`A`	$x' = 1$	-	✓	-	✓
`B`		✓	✓	-	✓
`C`	$x = 1$	-	-	✓	✓
`D`		✓	-	✓	✓
`E`		-	✓	✓	✓

Table 2. Trivial encoding of the symbols

	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:

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

or equivalently

$$\bar{x}\bar{x}' \wedge \bar{P}_0 = 0 \qquad x\bar{x}' \wedge \bar{P}_1 = 0 \tag{6}$$
$$\bar{x}x' \wedge \bar{P}_2 = 0 \qquad xx' \wedge \bar{P}_3 = 0$$

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 [2]:

$$\delta x = \Delta_x \qquad \delta y = \Delta_y \qquad \delta z = \Delta_z \qquad z = x \odot y, \tag{7}$$

where 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 path 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 $\delta y = \Delta_x^{\ggg i}$. Note that we might have to relax the constraints slightly if $\Delta_x^{[i]}$ is one of `=`, `!`, `>` or `<` (it will be relaxed to `-`, `-`, `x` and `x`, respectively).

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

3.3 New constraints

In this work, we extend this framework by considering constraints on pair of consecutive bits, instead of strictly bitwise constraints. This allows to express some conditions that occur naturally when considering carry extension, such as $x^{[i]} = x^{[i-1]}$.

We don't use the full set of 2^{16} possible constraints on $(x^{[i]}, x'^{[i]}, x^{[i-1]}, x'^{[i-1]})$, because that would lead to a system with too many parameters. In our implementation, we use only the 16 constraints described in Table 3. For instance, the symbol `=` means that the current bit is inactive, and that bit i of x if equal to bit i of $2x$, *i.e.* to bit $i-1$ of x – this can written as $x^{[i]} = x^{[i-1]}$.

We can easily include more constraints in our framework, but this set of symbol is quite expressive, and it gives good results in practice. Moreover, note that some case 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`).

Table 3. New constraints

$(x \oplus x', x \oplus 2x, x)$:		$(0,0,0)$	$(0,0,1)$	$(0,1,0)$	$(0,1,1)$	$(1,0,0)$	$(1,0,1)$	$(1,1,0)$	$(1,1,1)$
`?`	*anything*	✓	✓	✓	✓	✓	✓	✓	✓
`-`	$x = x'$	✓	✓	✓	✓	-	-	-	-
`x`	$x \neq x'$	-	-	-	-	✓	✓	✓	✓
`0`	$x = x' = 0$	✓	-	✓	-	-	-	-	-
`u`	$(x, x') = (0, 1)$	-	-	-	-	✓	-	✓	-
`n`	$(x, x') = (1, 0)$	-	-	-	-	-	✓	-	✓
`1`	$x = x' = 0$	-	✓	-	✓	-	-	-	-
`#`	*incompatible*	-	-	-	-	-	-	-	-
`3`	$x = 0$	✓	-	✓	-	✓	-	✓	-
`C`	$x = 1$	-	✓	-	✓	-	✓	-	✓
`5`	$x' = 0$	✓	-	✓	-	-	✓	-	✓
`A`	$x' = 1$	-	✓	-	✓	✓	-	✓	-
`=`	$x = x' = 2x$	✓	✓	-	-	-	-	-	-
`!`	$x = x' \neq 2x$	-	-	✓	✓	-	-	-	-
`>`	$x \neq x' = 2x$	-	-	-	-	✓	✓	-	-
`<`	$x \neq x' \neq 2x$	-	-	-	-	-	-	✓	✓

We also denote the new set of constraint by δ. Since the definition of δ only involves bitwise operation and a left-shift by one bit, $\delta x = \Delta$ can by written as a S-system, similar to System (6), as shown in Figure 3.

3.4 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

```
Mux3(V0^V1,V0^(V0+V0),V0,
#                   - 0 = 5 ? 1 ! A x u < 3 # n > C
Mux4(P0,P1,P2,P3,   1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0),
Mux4(P0,P1,P2,P3,   1,0,1,0,1,1,0,1,0,0,0,0,0,0,0,1),
Mux4(P0,P1,P2,P3,   1,1,0,1,1,0,1,0,0,0,0,1,0,0,0,0),
Mux4(P0,P1,P2,P3,   1,0,0,0,1,1,1,1,0,0,0,0,0,0,0,1),
Mux4(P0,P1,P2,P3,   0,0,0,0,1,0,0,1,1,1,1,1,0,0,0,0),
Mux4(P0,P1,P2,P3,   0,0,0,1,1,0,0,0,1,0,1,0,0,1,0,1),
Mux4(P0,P1,P2,P3,   0,0,0,0,1,0,0,1,1,1,0,1,0,0,1,0),
Mux4(P0,P1,P2,P3,   0,0,0,1,1,0,0,0,1,0,0,0,0,1,1,1)
);
```

Fig. 3. Simple encoding of our new δ constraint. We just use a copy of Table 3.

the following splits:

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

For instance, if some bit is specified as `-`, we test whether the system is still compatible when it is restricted to `0` and to `1`, respectively. If one of the systems becomes incompatible, we can turn the `-` constraint into `0` or `1`, accordingly. If both are still compatible, we then try to restrict the `-` bit to `=` and `!`.

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 too large, but the extra constraints found in one system can generate new constraints in other systems involving the same variable. The technique will essentially discover *necessary* constraints that have to be followed given the input differential.

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

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

When using only the linear difference, or the constraints of [dCR06], this system seems to be compatible, and constraint propagation gives $\delta z =$ `-xxx` (and $\delta y =$ `---1`). However, using our new constraints, the algorithm further deduces $\delta z =$ `-<<x` from the first addition and $\delta z =$ `->><x` (and $\delta v =$ `---1`) from the second addition, and the incompatibility is detected.

4 Applications

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 paths that follow a natural construction, and seem valid when verified manually, are in fact incompatible. We also found problems with boomerang attacks. This is due to a problem previously identified by Murphy in [Mur11]: boomerang attacks assume that the probabilities of the paths are independent, but the states are in fact strongly correlated where the top and the bottom path joins.

We will now describe some of the patterns that can lead to unexpected problem, and discuss the validity of some previously published differential attacks.

4.1 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 known [LM01,MVCP10], 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 path used in a recent near-collision attack against Skein [YCKW11] contains a pattern similar to this one[3]:

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

This seems valid when considering signed differences, but in fact, this does not have any solution. It does not seem easy to modify the path of [YCKW11] to obtain a valid attack.

4.2 Incompatibility in Boomerang Characteristics

To study a boomerang attack, we consider it as collection of constraints for the top path and the bottom path, plus extra constraints to link the quartets. Let x be some internal state variable, and $x^{(0)}$, $x^{(1)}$, $x^{(2)}$, $x^{(3)}$ be the corresponding variable in a boomerang quartet. A boomerang property is built by specifying a top differential for $(x^{(0)}, x^{(2)})$ and $(x^{(1)}, x^{(3)})$, and a bottom differential for $(x^{(0)}, x^{(1)})$ and $(x^{(2)}, x^{(3)})$. For more generality, we allow the two characteristics to be different in each case (for instance, the signs might be different). The top differential will be mostly unconstrained for the bottom part of the cipher, while the bottom differential will be mostly unconstrained for the top part.

In addition to the systems for each operation similar to (7), we build an additional system for each variable that links the quartet:

$$\delta(x^{(0)}, x^{(2)}) = \Delta_{0,2} \qquad \delta(x^{(1)}, x^{(3)}) = \Delta_{1,3} \qquad \textit{(Top path)}$$
$$\delta(x^{(0)}, x^{(1)}) = \Delta_{0,1} \qquad \delta(x^{(2)}, x^{(3)}) = \Delta_{2,3} \qquad \textit{(Bottom path)}$$

This allows to propagate information from one side of the quartet to the other side.

We found that some very simple patterns can lead to incompatibilities. Figure 4 gives an example of a pattern that results in incompatible characteristics. A quartet following those characteristics would have to satisfy:

$$x^{(0)} \oplus x^{(2)} = x^{(1)} \oplus x^{(3)} = \texttt{01} \qquad y^{(0)} \oplus y^{(2)} = y^{(1)} \oplus y^{(3)} = \texttt{00} \qquad \textit{(Top path)} \quad (8)$$
$$x^{(0)} \oplus x^{(1)} = x^{(2)} \oplus x^{(3)} = \texttt{01} \qquad y^{(0)} \oplus y^{(1)} = y^{(2)} \oplus y^{(3)} = \texttt{01} \qquad \textit{(Bottom path)} \quad (9)$$
$$x^{(0)} \boxplus y^{(0)} = x^{(1)} \boxplus y^{(1)} \qquad x^{(2)} \boxplus y^{(2)} = x^{(3)} \boxplus y^{(3)} \quad (10)$$

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

Without loss of generality, we can assume that $\mathrm{lsb}(x^{(0)}) = 0$. This gives $\mathrm{lsb}(x^{(1)}) = 1$ from (9), and $\mathrm{lsb}(x^{(2)}) = 1$ and $\mathrm{lsb}(x^{(3)}) = 0$ from (8). We can deduce $y^{(0)} = y^{(1)} \boxplus 1$ and $y^{(3)} = y^{(2)} \boxplus 1$ from (10). Combined with (9) this yields $\mathrm{lsb}(y^{(0)}) = 1$, $\mathrm{lsb}(y^{(1)}) = 0$, $\mathrm{lsb}(y^{(2)}) = 0$, and $\mathrm{lsb}(y^{(3)}) = 1$. This is incompatible with (8).

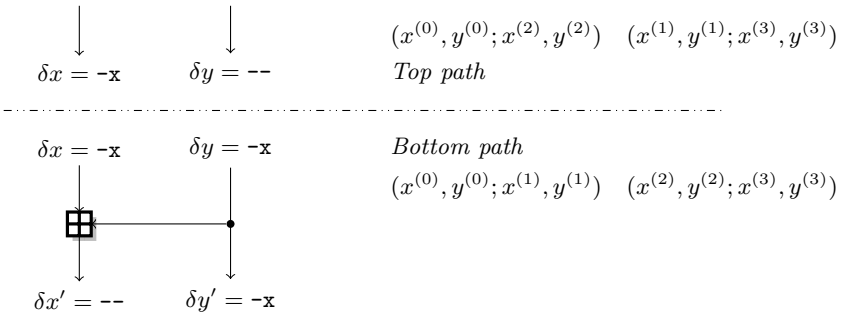


Fig. 4. Example of incompatible characteristics

This pattern seem to appear very frequently when using linearized characteristics in ARX designs, and shows that some very natural characteristics cannot be combined in a boomerang attack.

4.3 Application to Blake-256

We have studied the boomerang attacks on Blake from Biryukov *et.al* in [BNR11].

Keyed permutation attacks. The characteristic used for the attack on six-round of the keyed permutation is valid since an example of quartet is given in the paper, and a seven round attack can be mounted with the same characteristic by adding half a round at the top and bottom. However, the characteristics used for the 8-round attack, with two rounds in the middle section, are in fact incompatible.

More precisely, we look at the differences at round 5.5 proposed in [BNR11], and we look at the values v_{11} and v_{15}. We can see that in the top characteristic, v_{11} is active on bit 7, while v_{15} is active on bit 7, and those two active bits are expected to cancel in the backward G_3 function, with $c = c \boxminus d$ (where c is v_{11} and d is v_{15}). When we look at the bottom characteristic, we can see that v_{11} is active on bit 7, but v_{15} is inactive on bit 7. Therefore, we have the pattern described in Figure 4, which results in incompatible characteristics.

More explicitly, it means that 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 the help of the authors of [BNR11] 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 leads to a valid attack, but the cost of the attack is higher than reported in [BNR11].

Compression function attacks. Similarly, for the compression function attacks, 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.

4.4 Application to Skein-512

We also used our tool to study the boomerang characteristic given in [CJ10], and we found that they are in fact not compatible.

Following the notations of [CJ10], the characteristic for rounds 0–16 specifies a difference $e_{15,5}[10, 39, 49, *64]$. If this characteristic is applied to states $(e^{(0)}, e^{(1)})$ and $(e^{(2)}, e^{(3)})$, this implies that on bit 49, we have:

$$e^{(0)}_{15,5} = 0 \qquad e^{(1)}_{15,5} = 1 \qquad e^{(2)}_{15,5} = 0 \qquad e^{(3)}_{15,5} = 1 \tag{11}$$

Similarly, the characteristic for round 16–32 has $e_{16,2}[5, 11, 16, 41, 44, 47]$ and $e_{16,5}[-34, -50]$, assuming there are no carries. Since $e_{16,2}, e_{16,5} = \texttt{MIX}(e_{15,4}, e_{15,5})$, and the rotation used at that step is 56, we have $e_{15,5} = (e_{16,2} \oplus e_{16,5})^{\ggg 56}$. This results in $e_{15,5}$ being active on bits 13, 19, 24, 42, 49, 52, 55, and 58. If this characteristic is applied to states $(e^{(0)}, e^{(2)})$ and $(e^{(1)}, e^{(3)})$, this implies that on bit 49, we have $v^{(0)}_{15,5} \neq v^{(2)}_{15,5}$ and $v^{(0)}_{15,5} \neq v^{(2)}_{15,5}$. This is contradictory with (11).

We tried to fix the characteristics using a carry extension from bit 34 to 41 in $e_{16,5}$, by using different signs in the characteristics $(e^{(0)}, e^{(1)})$ and $(e^{(2)}, e^{(3)})$, or by using a carry extension in $e_{15,5}[49]$, but this always ends up with a similar contradiction.

The problem can be verified by building quartets with the specified differences before and after the middle key-addition[4], and computing the third `MIX` function backwards. The specified difference is never found in both paths simultaneously.

The boomerang attack from [AÇM+09] uses a similar path but it is based on older version of Skein with different rotation constants.

Conclusion

In this paper we have described a toolkit to study differential paths in ARX designs. Using these tools, we have discovered some problem in published attacks against such design. 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 authors of [BNR11] for helping us verify the problem with the attack, and finding an alternative path.

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

[4] That is to say, pick a random state $v^{(0)}$, a random key $k^{(0)}$, and compute the corresponding related keys $k^{(1)}$, $k^{(2)}$, $k^{(3)}$. Compute $v^{(1)}$ form $v^{(0)}$ using the top differential path. Then compute $e^{(0)} = v^{(0)} \boxplus k^{(0)}$, and $e^{(1)} = v^{(1)} \boxplus k^{(1)}$. Compute $e^{(2)}$ from $e^{(0)}$ and $e^{(3)}$ from $e^{(1)}$ using the bottom differential path. Finally, compute $v^{(3)} = e^{(3)} \boxminus k^{(3)}$ and $v^{(4)} = e^{(4)} \boxminus k^{(4)}$, and check if they satisfy the top differential path.

References

AÇM+09. Jean-Philippe Aumasson, Çagdas Çalik, Willi Meier, Onur Özen, Raphael C.-W. Phan, and Kerem Varici. Improved Cryptanalysis of Skein. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 542–559. Springer, 2009.

BDK06. Eli Biham, Orr Dunkelman, and Nathan Keller. New Cryptanalytic Results on IDEA. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 412–427. Springer, 2006.

BDK07. Eli Biham, Orr Dunkelman, and Nathan Keller. A New Attack on 6-Round IDEA. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2007.

BDKS11. Eli Biham, Orr Dunkelman, Nathan Keller, and Adi Shamir. New data-efficient attacks on reduced-round idea. Cryptology ePrint Archive, Report 2011/417, 2011. `http://eprint.iacr.org/`.

BNR11. Alex Biryukov, Ivica Nikolic, and Arnab Roy. Boomerang Attacks on BLAKE-32. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2011.

CJ10. Jiazhe Chen and Keting Jia. Improved Related-Key Boomerang Attacks on Round-Reduced Threefish-512. In Jin Kwak, Robert H. Deng, Yoojae Won, and Guilin Wang, editors, *ISPEC*, volume 6047 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2010.

dCR06. 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 *Lecture Notes in Computer Science*, pages 1–20. Springer, 2006.

JPV04. Jorge Nakahara Jr., Bart Preneel, and Joos Vandewalle. The Biryukov-Demirci Attack on Reduced-Round Versions of IDEA and MESH Ciphers. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *ACISP*, volume 3108 of *Lecture Notes in Computer Science*, pages 98–109. Springer, 2004.

Jun05. Pascal Junod. New Attacks Against Reduced-Round Versions of IDEA. In Henri Gilbert and Helena Handschuh, editors, *FSE*, volume 3557 of *Lecture Notes in Computer Science*, pages 384–397. Springer, 2005.

LM01. Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In Mitsuru Matsui, editor, *FSE*, volume 2355 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

LT11. Gaëtan Leurent and Søren S. Thomsen. Practical Near-Collisions on the Compression Function of BMW. In Antoine Joux, editor, *FSE*, volume 6733 of *Lecture Notes in Computer Science*, pages 238–251. Springer, 2011.

Mur11. Sean Murphy. The Return of the Cryptographic Boomerang. *IEEE Transactions on Information Theory*, 57(4):2517–2521, 2011.

MVCP10. Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The Differential Analysis of S-Functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 36–56. Springer, 2010.

YCKW11. Hongbo Yu, Jiazhe Chen, Ketingjia, and Xiaoyun Wang. Near-collision attack on the step-reduced compression function of skein-256. Cryptology ePrint Archive, Report 2011/148, last revised 31 Mar 2011, 2011. `http://eprint.iacr.org/`.