Construction of Differential Characteristics in ARX Designs Application to Skein

Gaëtan Leurent

UCL Crypto Group[**]
`Gaetan.Leurent@uclouvain.be`

Abstract. In this paper, we study differential attacks against ARX schemes. We build upon the generalized characteristics of De Cannière and Rechberger and the multi-bit constraints of Leurent.
Our main result is an algorithm to build complex non-linear differential characteristics for ARX constructions, that we applied to reduced versions of the hash function Skein. We present several characteristics for use in various attack scenarios: on the one hand we show attacks with a relatively low complexity, in relatively strong settings; and on the other hand weaker distinguishers reaching more rounds. Our most notable results are practical free-start and semi-free-start collision attacks for 20 rounds and 12 rounds of Skein-256, respectively. Since the full version of Skein-256 has 72 rounds, this result confirms the large security margin of the design.
These results are some of the first examples of complex differential trails built for pure ARX designs. We believe this is an important work to assess the security those functions against differential cryptanalysis. Our tools are publicly available from the ARXtools webpage.

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

1 Introduction

ARX is a popular alternative to S-Box based designs for the design of symmetric key cryptographic primitives. ARX designs use only 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. ARX designs have been quite popular recently; in particular, two of the SHA-3 finalists, BLAKE and Skein, follow this design strategy. This stategy as also been used for stream ciphers such as Salsa20 and ChaCha, and block ciphers, such as TEA, XTEA or HIGHT (RC5 uses additions and data-dependant rotations, but we only consider construction with

[**] Part of this work was done when the author was at the University of Luxembourg.

fixed rotations). Recently, a dedicated short-input PRF, SipHash [1], has been built following the ARX design. We note that Salsa20 is in the eStream portfolio, while SipHash is already deployed as the default hash table implementation of the Perl and Ruby languages. 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 designs usually happen at the word-level; differential characteristics are relatively easy to build, but efficient attacks often need novel techniques, such as the rebound attack against hash functions [20]. 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 [28,30,29], and a significant effort has been dedicated to building tools to construct automatically such characteristics [6,24,10,17,25,18,16]. 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 [11], the construction of a rogue MD5 CA certificate [26], and attacks against combiners [19].

However, this body of work is mainly focused on MD/SHA designs, as opposed to pure ARX designs such as Skein, BLAKE or Salsa20. In MD/SHA-like functions, the Boolean functions play an important role, and the possibility to absorb differences gives a lot of freedom for the construction of differential characteristics. In pure ARX designs, the addition is the only source of non-linearity (over $\mathbb{F}_2$), and the freedom in the carry expansions is much harder to use than the absorption property of Boolean functions.

To this effect, Leurent introduced multi-bit constraints [14] involving several consecutive bits of a variable (*i.e.* $x^{[i]}$ and $x^{[i-1]}$), instead of considering bits one by one. He describes reduced sets of 1.5-bit and 2.5-bit constraints, and explains how to propagate these constraints using S-systems and automata. This set of constraints is well suited to study ARX designs because it can extract a lot of information about the carry extensions in modular additions. A set of tools to propagate these constraints is given in [14], and the main result is a negative result (for the cryptanalyst) showing that several previous attacks are invalid.

1.1 Our Results

In this paper, we study the problem of constructing differential characteristics for ARX schemes. This work is heavily inspired by the framework of generalized characteristics from De Cannière and Rechberger [6], and the multi-bit constraints of [14]. As opposed to the results of [14], we give positive results for cryptanalysts.

We first recall how to describe a differential characteristic, and the main ideas for constraint propagation in Section 2. Then, we describe a differential characteristic search algorithm in Section 3 using a constraint propagation tool, and we present our results on Skein in Section 4. Finally, we describe our technical improvements over the previous constraint propagation tools in the full version of this paper.

Construction of differential characteristics. We use a propagation tool to construct differential characteristics automatically. Using an efficient constraint propagation tool and some simple heuristics, we show that we can actually build complex non-linear characteristics. We obtain some of the first complex differential trails for ARX designs and we believe that this automated approach is an important step to assess the security of ARX designs against differential cryptanalysis.

Application to Skein. We apply this technique to reduced versions of the Skein hash function, where we build rebound-like characteristics by connecting two high-probability trails.

We compare our results with previous works in Table 1. Most previous works on Skein are either weak distinguishers (such as boomerang properties or free-tweak free-start partial-collisions), or attack with marginal improvement over brute-force (such as some biclique-based results). In this work, we present attacks in relatively strong settings (collisions and free-start collisions) with a relatively low complexity (several attacks are practical, and all our attack gain at least a factor 2^8).

Constraint propagation. Finally, we describe an alternative way to perform the constraint propagation for multi-bit constraints. Our approach is significantly more efficient that the technique of [14], and uses the full set of 2^{32} constraints instead of a reduced set of 16 carefully chosen constraints. The reduced set is sufficient in most situations, but we show that the full set extracts some more information. This improvement was crucial to allow the characteristic search to work in practice.

In addition, our approach can also deal with larger systems that the previous technique with a reasonable complexity. In particular, we can deal with the 3-input modular sums, and 3-input Boolean functions used in functions of the MD/SHA family. We can also propagate 4 simultaneous trails in a boomerang configuration through an addition or an xor, with full 2-bit constraints.

1.2 Related work

A recent result by Yu *et al.* achieves a similar result as our free-start free-tweak partial-collision on 32 rounds, and is also based on a complex non-linear trail for Skein-256. This work has been available on ePrint since April 2011 [31], but the characteristic given in that version of the paper was flawed [14]. This has motivated our work on building such characteristics automatically.

More recently, they managed to build a valid characteristic and their work has been presented at FSE [33]; this result was achieved simultaneously and

Table 1. Comparison of attacks on reduced versions of Skein-256 (we omit attack on previous versions, and weak distinguishers). The full Skein-256 has 72 rounds.
In order to compare various attack settings, we count the number of extra degrees of freedom used by the attack.

	Extra Degrees of freedom	Rounds	Time	Generic	Ref, notes
Collision	0	4	2^{96}	2^{128}	[13], biclique
		8	2^{120}		
		9	2^{124}		
		12	$2^{126.5}$		
Free-start collision	8	22[†]	$2^{253.8\dagger}$	2^{256}	[15], biclique
		37[†]	$2^{255.7\dagger}$		
Related-tweak[‡] partial q-multicol	10	20	$q \cdot 2^{97}$	$2^{\frac{q-1}{q+1}\cdot 130}$	[27], 126 bits
Free-tweak partial q-multicol	12	32	$q \cdot 2^{85}$	$2^{\frac{q-1}{q+1}\cdot 205}$	[33], 51 bits
Collision	0	12	$\approx 2^{100\star}$	2^{128}	4.4
Semi-free-start collision	4	12	$\approx 2^{40}$	2^{128}	4.4
Free-start collision	8	20	$\approx 2^{40}$	2^{128}	4.5
Free-start near-collision	8	24	$\approx 2^{40}$	$2^{88.4}$	4.5, 15 bits
Related-tweak[‡] near-collision	10	24	$\approx 2^{40}$	$2^{117.3}$	4.6, 3 bits
Related-tweak[‡] partial q-multicol	10	32	$\approx q \cdot 2^{119\star}$	$2^{\frac{q-1}{q+1}\cdot 205}$	4.6, 51 bits
Free-tweak partial q-multicol	12	32	$q \cdot 2^{105}$	$2^{\frac{q-1}{q+1}\cdot 205}$	4.6, 51 bits
Block cipher attacks					
Key recovery (Threefish-512)		32	2^{181}	2^{512}	[32], Boomerang
		33	2^{305}		
		34	2^{424}		

† Attacks on Skein-512. For Skein-256, fewer round will be attacked, with a complexity slightly below 2^{128}.
‡ Using freedom degrees in the tweak *difference*, but the tweak *value* can be arbitrary.
* Using heuristic assumptions about the search for a large number of characteristics.

independently from our work. Building such a trail by hand is impressive, but this kind of result it is very challenging to replicate or to apply to another primitive. We hope that our automatic approach will be easier to adapt to new settings.

2 Analysis of Differential Characteristics

The first step for working with differential characteristics (or trails) is to choose a way to represent a characteristic, and to evaluate its probability. The main idea of differential cryptanalysis is to consider the computation of the function for a pair of inputs X, X', and to specify the difference between x and x' for every internal state variable x. The difference can be the xor difference, the modular difference, or more generally, use any group operation. However, this approach is not efficient for ARX design, because both the modular difference and the xor difference play an important role. Several works have proposed better way to represent a differential characteristic for ARX designs.

Signed bitwise difference. The groundbreaking results of Wang *et al.* [28,30,29] are based on a bitwise signed difference. For each bit of the state, they specify whether the bit is inactive ($x = x'$), active with a positive sign ($x = 0, x' = 1$), or active with a negative sign ($x = 1, x' = 0$). This information express both the xor difference and the modular difference.

Generalized characteristics. This was later generalized by De Cannière and Rechberger [6]: for each bit of the state, they look at all possible values of the pair (x, x'), and they specify which values are allowed. The constraints `-`, `u` and `n` correspond to the bitwise signed difference of Wang. De Cannière and Rechberger also describe an algorithm to build differential characteristics using this set of constraints.

Multi-bit constraints. Recently, Leurent studied differential characteristics for ARX designs, and introduced multi-bit constraints [14]. These constraints are applied to the values of consecutive bits of a state variable (*e.g.* $x^{[i]}$ and $x^{[i-1]}$) instead of being purely bitwise. Multi-bit constraints are quite efficient to study ARX designs because they can capture the behaviour of carries in the modular addition. Two set of constraints are introduced in [14]:

- a set of 16 constraints involving $(x^{[i]}, x'^{[i]}, x^{[i-1]})$ called 1.5-bit constraints;
- a set of 16 constraints involving $(x^{[i]}, x'^{[i]}, x^{[i-1]}, x'^{[i-1]}, x^{[i-2]})$ called 2.5-bit constraints.

The full sets of 2^8 1.5-bit constraints and 2^{32} 2.5-bit constraint are not used because the propagation method of [14] becomes impractical with such large sets.

2.1 Constraint Propagation and Probability Computation

In [14], the constraints are studied using the theory of S-functions introduced in [22]. 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 $S[t-1]$.

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. A system of equation that can be written as a S-function is called an S-system.

2.2 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 a series a constraints $\varDelta$, we write $\delta x = \varDelta$ to denote that the pair (x, x') follows the difference pattern $\varDelta$. For instance, $\delta x = \texttt{x--0}$ is equivalent to $x \oplus x' = \texttt{1000}$ and $x^{[0]} = 0$.

For each operation $\odot$, we can write a system:

$$\delta x = \varDelta_x \qquad \delta y = \varDelta_y \qquad \delta z = \varDelta_z \qquad z = x \odot y \qquad z' = x' \odot y', \tag{1}$$

where x, y, z, x', y', z' are unknowns, and $\varDelta_x, \varDelta_y, \varDelta_z$ are parameters. In an ARX design, all the operations except the rotations are S-function, and the difference operation δ can be written with bitwise operations and left-shifts; therefore system (1) is an S-system. Using tools to analyze this S-system, we can verify if the specified input and output patterns for each operation are compatible. We deal with the rotations $y = x \ggg i$ by just rotating the constraint pattern: if $\delta x = \varDelta_x$ then we use $\delta y = \varDelta_x \ggg i$.

We can also find new constraints that must be satisfied for any solution to the system. This allows to propagate constraints between the inputs and outputs of the operation $\odot$. When we consider a characteristic for a cipher, this process will be iterated for each operation, until no new constraints are found.

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.

2.3 Tools for S-systems

In [14], a set of constraints is represented by an S-system, and an automaton is built to compute the probability of each operation. To perform constraints propagation, each constraint is split into two disjoint subsets; if one of the subsets results in an incompatible system, the constraint can be restricted to the other subset without reducing the number of solutions.

This approach allows to achieve a good efficiency when the automaton is fully determinized: one can test whether a system is compatible with only n table accesses. However, the table becomes impractically large if the set of constraints is too large, or if the operation is too complex. In [14], the automaton is fully determinized for 1.5-bit constraints, but could not be determinized for 2.5-bit constraints; this results in a quite inefficient propagation algorithm for 2.5-bit constraints.

In this work, we explore a different option using non-deterministic automata. This allows to deal with large set of constraints and more complex operations. We need to perform many operations to verify whether a system is compatible, but the automata are very sparse and can be represented by tables small enough to fit in the cache (the tables of [14] take hundreds of megabytes for an addition);

this gives better results in practice. In addition, we show special properties of the automata allowing an efficient propagation algorithm without splitting the constraints into subsets. Due to space constraints, the technical details of our new approach are given in the full version of this paper.

2.4 Comparison

Table 2. Experiments with toy versions of Skein. We give the number of input/output differences accepted by each technique, and the ratio of false positive.

		2 rounds / 4 bits		3 rounds / 6 bits (sparse*)		
Method		Accepted	F pos.	Accepted	F pos.	Time†
Exhaustive search		$2^{25.1}$ (35960536)	–	$2^{18.7}$ (427667)	–	
2.5-bit full set		$2^{25.3}$ (40597936)	0.13	$2^{19.2}$ (619492)	0.4	2.5 ms
2.5-bit reduced set	[14]	$2^{25.3}$ (40820032)	0.14	$2^{19.5}$ (746742)	0.7	50 ms
1.5-bit reduced set	[14]	$2^{25.3}$ (40820032)	0.14	$2^{20.4}$ (1372774)	2.2	0.5 ms
1-bit constraints	[6]	$2^{25.4}$ (43564288)	0.21	$2^{20.7}$ (1762857)	3.1	0.5 ms
Check adds independently		$2^{25.8}$ (56484732)	0.57			

* Weight 4 differences. The number of input/output differences is $\left(\sum_{i=0}^{4}\binom{24}{i}\right)^2 \approx 2^{26.7}$
† Average time to verify one input/output difference (over the false positives of the 1.5-bit reduced set).

We show a comparison with previous methods in Table 2. We use the same settings as [14]:

1. A reduced Skein with two rounds and 4 words of 4 bits each; In this setting the full 2.5-bit constraints offer a little advantage over the reduced set of 2.5-bit constraints.
2. 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. In this setting, the full 2.5-bit constraints give a significant improvement over the reduced set of 2.5-bit constraints.

These experiments show that using the full set of 2.5-bit constraints gives better results than using the reduced set of [14]. We also give timing informations[1]: our new approach for constraint propagation is one order of magnitude faster that the previous method with a reduced set of 2.5-bit constraints, and somewhat slower than the previous method with 1.5-bit constraints.

[1] The comparison is done with similar implementations.

3 Automatic Construction of Differential Characteristics

In order to mount a differential attack for a hash function or a block cipher, an important task is to build a differential characteristic. For the analysis of ARX primitives (and MD/SHA-like designs), the characteristic is usually designed at the bit level. This turns out to be a very challenging task because of the complex interactions between the operations, and the large number of state elements to consider.

This problem has been heavily studied for attacks on the MD/SHA family of hash functions: a series of attacks by X. Wang and her team are based on differential characteristics built by hand [28,30,29,33], while later works gave algorithms to build such characteristics automatically [6,24,10,17,25]. Unfortunately, most of those tools are not publicly available.

In this section, we show that the multi-bit constraints can be used to design a successful algorithm for this task on pure ARX designs. Our algorithm is heavily inspired by the pioneer work of De Cannière and Rechberger [6], and the more detailed explanation given in [23] and [21].

3.1 Types of Trails

Differential trails can be classified in two categories: iterative and non-iterative. An iterative characteristic exploits the round-based nature of many cryptographic constructions: if a trail can be built over a few rounds with the same input and output difference Δ, then this characteristic can be repeated to reach a larger number of rounds. In practice very few iterative characteristics have been found for ARX constructions, because many designs use different rotation amounts or Boolean functions over the rounds, or a non-iterative key-schedule. Notable exceptions include the attacks of den Boer and Bosselaers against MD5 [7], and the recent work of Dunkelman and Khovratovich on BLAKE [8]. In this work, we focus on non-iterative trails.

The main way to build non-iterated trails is to connect two simple and high-probability trails using a complex and low-probability section in between. The choice of the high-probability trails will depend on the attack setting, and should be done by the cryptanalyst using specific properties of the design, while the complex section will be build automatically by an algorithm (or by hand). When the characteristic is used in a hash-function attack, the cost of the low-probability section can usually be avoided.

For instance, the characteristics used for the attacks on SHA-1 use a linear section built using local collisions [4,29], and a non-linear section to connect a given input difference to the linear characteristic. This general idea is also the core of the rebound attack [20]: it combines two high-probability trails using a low-probability transition through an S-box layer.

In our applications, we will use a rebound-like approach to connect two high-probability trails with a complex low-probability section. Using rebound-like differential trails for ARX designs has also been proposed in [33].

3.2 Algorithm

Our algorithm takes as input a characteristic representing two high-probability trails $\varDelta_1 \to \varDelta_2$ and $\varDelta_3 \to \varDelta_4$. The middle section is initially unconstrained, *i.e.* filled with `?`. The main part of the algorithm is a search phase which tries to fill the middle part with a valid characteristic. We follow the general idea of the algorithm of De Cannière and Rechberger, by repeating the following operations:

Propagation: deduce more information from the current characteristic by running the propagation algorithm on each operation.

Guessing: select an unconstrained state bit (*i.e.* a `?` constraint), and reduce the set of allowed values (*e.g.* to a `-` or `x` constraint).

When a contradiction is found, we go back to the last guess, and make the opposite choice, leading to a backtracking algorithm. However, we abort after some number of trials and restart from scratch because mistakes in the early guesses would never be corrected.

Our algorithm is built from the idea that the constraint propagation is relatively efficient to check if a transition $\varDelta \to \varDelta'$ is possible. Therefore to connect the differences $\varDelta_2$ and $\varDelta_3$ from the high-probability trails, we essentially guess the middle difference $\varDelta'$ and we check whether the transitions $\varDelta_2 \to \varDelta'$ and $\varDelta' \to \varDelta_3$ are possible.

This leads to the following difference with the algorithm of De Cannière and Rechberger:

– We only use signed differences, *i.e.* we use the constraints `-`, `u`, and `n`.
– We specify in advance which words of the state will be restricted in the guessing phase, using state words in the middle of the unspecified section.
– We guess from the low bits to the high bits, and we can compare incomplete characteristics by counting how many bits have been guessed before aborting the search.
– Every time the backtracking process is aborted, we remember which guess was best and the random guesses of the next run are strongly biased toward this choice.

Thanks to this approach, we can use the best path of the previous run as an input for the search algorithm, and explore solutions with few differences in the guesses. Finally, we use a simulated annealing algorithm in order to find better characteristics.

3.3 Finding pairs

The hardest part of our attacks in to build the differential trails. Finding conforming pairs for the middle section is relatively easy using the propagation algorithm: one just has to make random choices for the unconstrained bits in the middle and run the propagation algorithm after each choice. In practice the paths we found leave very few choices to make, and most of them lead to valid pairs. The degrees of freedom in the key can then be used to build many different pairs. This can be compared to the rebound attack on AES-like designs [20]: in this attack the trails are easy to build, and finding pairs for the inbound phase has a small amortized cost.

4 Application to Skein-256

In this section, we apply our algorithm to build characteristics for several attack scenarios on Skein-256.

4.1 Short Description of Threefish and Skein

The compression function of Skein is based on the block cipher Threefish. In this paper we only study Threefish-256, which uses a 256-bit key (as 4 64-bit values), a 128-bit tweak (as 2 64-bit values), and a 256-bit state (as 4 64-bit values). The full version of Skein has 72 rounds. We denote the ith word of the state after r rounds as $e_{r,i}$. There is a key addition layer every 4 rounds:

$$e_{r,i} = \begin{cases} v_{r,i} + k_{r/4,i} & \text{if } r \bmod 4 = 0 \\ v_{r,i} & \text{otherwise} \end{cases}$$

where $k_{r/4,i}$ is the ith word of the round key at round $r/4$. The state $v_{r+1,i}$ (for $i = 0, 1, .., n_w$) after round $r+1$ is obtained from $e_{r,i}$ by applying a $\mathtt{MIX}$ transformation and a permutation of 4 words as following:

$$\begin{array}{lll} (f_{r,2j}, f_{r,2j+1}) & := \mathtt{MIX}_{r,j}(e_{r,2j}, e_{r,2j+1}) & \text{for } j = 0, 1, .., n_w/2 \\ v_{r+1,i} & := f_{r,\sigma(i)} & \text{for } i = 0, 1, .., n_w \end{array}$$

where σ is the permutation (0 3 2 1) (specified in [9]) and $(c, d) = \mathtt{MIX}_{r,j}(a, b)$ is defined as:

$$\begin{aligned} c &= a \boxplus b \\ d &= (b \lll R_{r \bmod 8,j}) \oplus c \end{aligned}$$

The rotations $R_{r \bmod 8,j}$ are specified in [9]. The key scheduling algorithm of Threefish produces the round keys from a tweak (t_0, t_1) and a key as following:

$$\begin{array}{ll} k_{l,0} = k_{(l \quad) \bmod 5} & k_{l,1} = k_{(l+1) \bmod 5} + t_{l \bmod 3} \\ k_{l,2} = k_{(l+2) \bmod 5} + t_{(l+1) \bmod 3} & k_{l,3} = k_{(l+3) \bmod 5} + l, \end{array}$$

where $k_4 = C_{240} \oplus \bigoplus_{i=0}^{4} k_i$ with C_{240} a constant specified in [9], and $t_2 = t_0 \oplus t_1$. The compression function F for Skein is given as $F(M, H, T) = E_{H,T}(M) \oplus M$, where H is the chaining value, M is the message, and T is a block counter. This follows the Matyas-Meyer-Oseas construction for the compression function, and the Haifa construction for the iteration.

In this work, we only consider attacks on multiples of four rounds, because the structure of Skein is built with 4-round blocks with key additions in between.

We describe attacks in three different settings in Sections 4.4, 4.5, and 4.6. The attacks are based on different kinds of trails shown in Figures 2, 3, and 4. Due to space constraints, we do not include differential characteristics, but they are given in the full version of this paper. All the characteristics have been verified by building a conforming pair, and we give example of colliding pairs in Appendix A.

4.2 Building Characteristics

To describe a differential characteristic for Skein with our framework, we write constraints for each $e_{r,i}$ value, and for the $v_{r,i}$ values before a key addition (*i.e.* when $r \bmod 4 = 0$). For each round, we have 4 equations and 2 rotations, corresponding to two `MIX` functions. We also write the full key schedule as a system of equations.

We note that the variables $e_{r,2j}$ with $r \bmod 4 = 0$ are only involved in modular additions: $f_{r,2j} = e_{r,2j} \boxplus e_{r,2j+1}$ and $e_{r,2j} = v_{r,2j} \boxplus k_{r/4,2j}$. Therefore, we could remove these variables, and write $f_{r,2j} = v_{r,2j} \boxplus k_{r/4,2j} \boxplus e_{r,2j+1}$ using a three-input modular addition. In practice, the propagation algorithm for three-input modular addition takes significantly longer, so we keep the variables, but we try to avoid constraining them since the multi-bit constraints can propagate the modular difference.

Choosing the high-probability characteristics. In attack setting with differences in the key, we build the high-probability trails starting from a non-active state, with a low-weight key difference. When we go through the key addition, a difference is introduced in the state, and we propagate the difference by linearizing the function. If we have no difference in the key, we start with a single active bit in the state and we propagate the difference for a few rounds by linearizing the function. Most of our trails use the most significant bit as the active bit in order to increase their probabilities.

4.3 General Results

For the algorithm to work successfully, we need to find a delicate balance in the initial characteristic. If the unconstrained section is too short, there will not be enough degrees of freedom to connect the high-probability parts. On the other hand, if the unconstrained section is too long, the propagation algorithm will not filter bad characteristics efficiently.

In practice, we can only build characteristics when we have a key addition layer in the unconstrained part of the characteristic. This way, the algorithm can use degrees of freedom from the key to connect the initial characteristics. In general it seems hard to find enough degrees of freedom to build a valid trail without using degrees of freedom from the key: for a random function f and arbitrary differences Δ_2 and Δ_3, we expect on average a single pair satisfying $f(x + \Delta_2) = f(x) + \Delta_3$. We can consider the intermediate differences for one

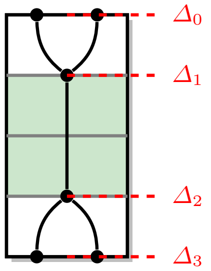

Fig. 1. Previous trails: rel-key, rel-tweak.

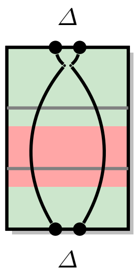

Fig. 2. Collision trails: fixed key.

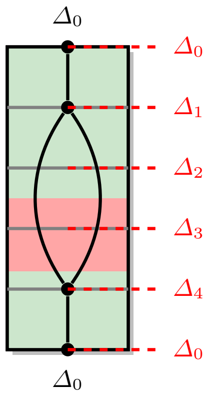

Fig. 3. Collision trails: related-key.

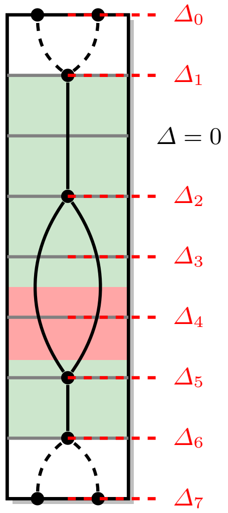

Fig. 4. (Near-) Collision trails: rel-key, rel-tweak.

such pair as a differential characteristics but a differential characteristic with a single valid pair is not very useful for a differential attack.

In order to let the algorithm use the degrees of freedom in the key efficiently, we use the registers before and after a key addition as guessing points: $v_{r,0}, v_{r,1}, v_{r,2}, v_{r,3}, e_{r,1}, e_{r,3}$ with $r \bmod 4 = 0$ (as discussed above we do not constrain $e_{r,0}$ and $e_{r,2}$).

We find that the characteristics built by the algorithm are rather dense, and use many degrees of freedom in the state, and many degrees of freedom in the key. This is not a problem for attacks on the compression function, but the characteristics are harder to use in attacks against the full hash function, where fewer degrees of freedom are available to the attacker. We note that this problem is less acute for attack against functions of the MD/SHA family, where the message block is much larger than the state.

On the other hand, the trail built by hand by Yu *et al.* [33] is somewhat sparser, and leaves more degrees of freedom for the key and the middle state.

4.4 Collision Attacks

We first study attacks with no difference in the chaining value so that they can be applied to the full hash function. Since Skein uses the MMO mode, the chaining value of the hash function is the key to the block cipher. We try to build characteristics for a collision attack, therefore we use the same difference in the initial state and in the final state so that they can cancel out in the feed-forward[2].

[2] We could build characteristics for 20 rounds if we consider near-collisions, but this would not work on the full hash function because of the finalization step.

We start with a low-weight difference in one of the first rounds and we propagate by linearization through rounds 0–4 and backward through round 11.

We give an example of a colliding pair for the compression function of Skein-256 reduced to 12 rounds in Table 3.

Full collision attack. To build a collision attack on the full hash function, we have to deal with the fact that the characteristic is only valid for a small fraction of the keys, *i.e.* a small fraction of the chaining values. We use a large number of characteristics, and a large number of random chaining values, in a meet-in-the-middle fashion.

More precisely, the characteristics given by the algorithm have many constraints of the key, which define a set of valid keys, and the number of conforming pairs estimated by looking at the probability of each step is even lower. We assume that each solution will correspond to a different key, and the number of solutions of the characteristics gives the number of key (*i.e.* chaining values) for which we can actually build a collision. Our experiments indicate that we can expect to build characteristics with more than 2^{106} solutions for a cost of 2^{50}. If we extrapolate this experimental result, we expect that it is possible to build many such characteristics. Let's assume that we can build N characteristics for a cost of $N \times 2^{50}$; where each characteristic has 2^{106} solutions out of 2^{200} valid keys. In a second phase, we will hash M random message blocks and test if they can give a collision using one of the characteristics. Out of the M chaining values generated, we expect that $M \times N \times 2^{200-256}$ will be valid for one characteristic, and $M \times N \times 2^{106-256}$ values will actually lead to a collision after verification. An important step of the attack will be to find for which characteristic a given chaining value can be valid, but this can be done efficiently using a hash table indexed by the bits of the chaining value which are imposed by the characteristics.

The optimal complexity is achieved with $N = 2^{50}$ and $M = 2^{100}$. With these parameters we only have to verify 2^{94} valid chaining values, so the verification step is negligible. This gives a collision attack on 12-round Skein-256 with a time complexity of 2^{100}, using memory to store 2^{50} characteristics[3].

The assumption that we can build so many good characteristics is a strong assumption, and it is hard to verify. However, we believe that this estimation is a safe upper bound, and that better characteristics would be found by running the search algorithm for longer times. In our experiments, we tested a few different high probability trails as input to the algorithms, and we spend an effort equivalent to about 2^{50} hash computations on our best candidate.We ran our algorithm 128 times with different initialization of the PRNG, and we report the best paths found by each run after 120 CPU hours in Figure 5. In addition, we ran a few parallel experiments for 120 hours with 32 cores. All these experiments generated more than 400.000 different characteristics; the best characteristic

[3] To store a characteristic, we just need to store masks defining the valid keys, and one state in the middle (if is not necessary to store all the intermediate constraints). Then, we can test a chaining value candidate by just computing all the intermediate states and checking if we reach a collision. This would take about 4×256 bits.

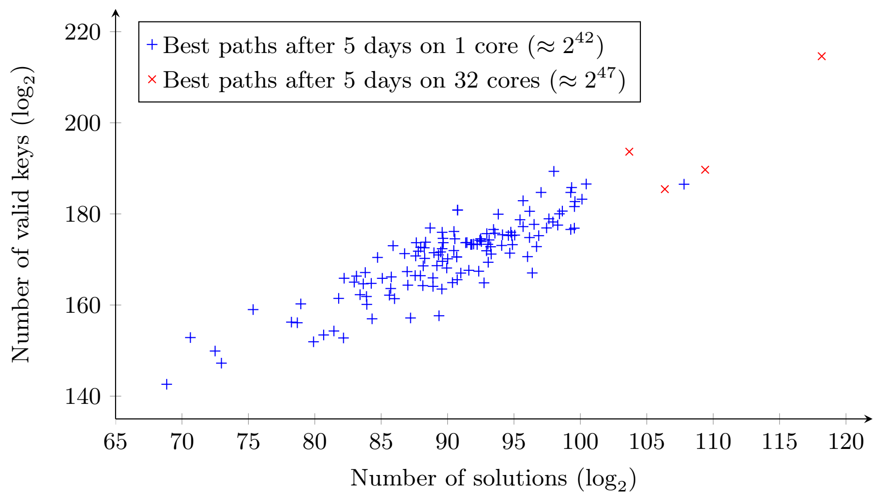

Fig. 5. Best characteristic found after each run. The experiments were run on Intel Xeon L5420 CPUs.

allow 2^{118} solutions, and 30.000 of these allow more than 2^{106} solutions (only the best characteristic of each run is shown in Figure 5). We note that in order to build a large number of characteristics, we would also use several different starting points for the linear part.

4.5 Free-start Collision Attack

For a collision attack on the compression function, *i.e.* a free-start attack on the hash function, we can use a difference in the chaining value (*i.e.* the key). We note that the key schedule of Skein-256 repeats itself every 20 rounds when there is no tweak difference. Therefore, we build trails with two inactive blocks as shown in Figure 3: the difference introduced in the initial state by k_0 cancels out with the difference introduced in the final state by k_5.

We give a collision pair built using this strategy in Table 4.

We can also extend this path to a free-start near-collision attack against 24-round Skein, if we extend the trail to 4 more rounds at the end. A linearized trail gives near-collisions with 15 active bits, and the cost of finding a conforming pair is negligible before the cost of finding the trail.

4.6 Free-tweak Free-start Near-collision Attack

Finally, we can use degrees of freedom in the tweak to reach the maximum number of rounds possible. Previous works have shown that the key schedule allows to have one round without any active key words if we use a difference in the tweak in order to cancel a difference in the key. Using this property we can build a 24-round trail, and extend it to 32 round by propagating the external difference for four extra rounds in each direction, as shown in Figure 4. This is the approach used in [31].

We give a characteristic built using this idea in the full version of this paper. This results in a low weight difference for the input and output, with many zero bits in predetermined position. Moreover, we can follow the approach of [31] and also specify a fixed characteristic for round 0 to 4 and 28 to 32. It costs about 2^{40} to build a characteristic that allows 2^{20} solutions, so we can estimate that the amortized costs of building a valid pair for rounds 4 to 28 is about 2^{20}. Using the analysis of [31], we would build a conforming pair for rounds 0 to 32 for a cost of $2^{20+43+43} = 2^{119}$, assuming that we can find 2^{66} different characteristics.

Alternatively, if we can choose the value of the tweak, then we only need a single characteristic, and we follow the same attack as [33] with the same complexity.

Note that the complexity of these attack is higher than the generic complexity of a partial-collision attack on $256 - 51$ pre-specified bits, $2^{102.5}$. However, the generic complexity to reach the fixed 256-bit difference with 51 pre-sepcified active bits is still 2^{128}. Alternatively, this attack can be considered as a q-multicollision attack [2].

Conclusion

In this paper we describe an algorithm to build differential characteristics for ARX designs, and we apply the algorithm to find characteristics for various attack scenarios on Skein. Our attacks do not threaten the security of Skein, but we achieve good results when compared to previous attacks; our main results are low-complexity attacks in relatively strong settings. In particular, we show practical free-start and semi-free-start collision attacks for 20 rounds and 12 rounds of Skein-256, respectively.

We obtain some of the first complex differential trails for pure ARX functions (as opposed to MD/SHA-like functions with Boolean functions). Since our approach is rather generic, we expect that our technique can be applied to other ARX designs, and will be used to evaluate the security of these designs against differential cryptanalysis.

Our improvements to the tools of [14], and the code to build differential characteristics for Skein are publicly available from the ARXtools webpage: `http://www.di.ens.fr/~leurent/arxtools.html`. We hope that this will promote cooperation between researchers, and avoid a situation where several teams have to develop their own implementation.

Acknowledgement

We would like to thank Pierre-Alain Fouque and Thomas Peyrin for fruitful discussions about differential characteristics and propagation of constraints.

The author is supported by the ERC project CRASH. Part of this work was done while the author was at the university of Luxembourg, supported by the AFR grant PDR-10-022 of the FNR.

Computational resources have been provided by HPC facility of the University of Luxembourg, as well as by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11.

References

1. Aumasson, J.P., Bernstein, D.J.: SipHash: a fast short-input PRF. In Galbraith, S., Nandi, M., eds.: INDOCRYPT. Lecture Notes in Computer Science, Springer (2012)
2. Biryukov, A., Khovratovich, D., Nikolic, I.: Distinguisher and Related-Key Attack on the Full AES-256. [12] 231–249
3. Canteaut, A., ed.: Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers. In Canteaut, A., ed.: FSE. Volume 7549 of Lecture Notes in Computer Science., Springer (2012)
4. Chabaud, F., Joux, A.: Differential Collisions in SHA-0. In Krawczyk, H., ed.: CRYPTO. Volume 1462 of Lecture Notes in Computer Science., Springer (1998) 56–71
5. Cramer, R., ed.: Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings. In Cramer, R., ed.: EUROCRYPT. Volume 3494 of Lecture Notes in Computer Science., Springer (2005)
6. De Cannière, C., Rechberger, C.: Finding SHA-1 Characteristics: General Results and Applications. In Lai, X., Chen, K., eds.: ASIACRYPT. Volume 4284 of Lecture Notes in Computer Science., Springer (2006) 1–20
7. den Boer, B., Bosselaers, A.: Collisions for the Compressin Function of MD5. In Helleseth, T., ed.: EUROCRYPT. Volume 765 of Lecture Notes in Computer Science., Springer (1993) 293–304
8. Dunkelman, O., Khovratovich, D.: Iterative differentials, symmetries, and message modification in BLAKE-256. In: ECRYPT2 Hash Workshop. (2011)
9. Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., Callas, J., Walker, J.: The Skein hash function family. Submission to NIST (2008/2010)
10. Fouque, P.A., Leurent, G., Nguyen, P.: Automatic Search of Differential Path in MD4. In: ECRYPT Hash Worshop. (2007) `http://eprint.iacr.org/2007/206`.
11. Fouque, P.A., Leurent, G., Nguyen, P.Q.: Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In Menezes, A., ed.: CRYPTO. Volume 4622 of Lecture Notes in Computer Science., Springer (2007) 13–30
12. Halevi, S., ed.: Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings. In Halevi, S., ed.: CRYPTO. Volume 5677 of Lecture Notes in Computer Science., Springer (2009)
13. Khovratovich, D., Rechberger, C., Savelieva, A.: Bicliques for Preimages: Attacks on Skein-512 and the SHA-2 Family. [3] 244–263

14. Leurent, G.: Analysis of Differential Attacks in ARX Constructions. In Wang, X., Sako, K., eds.: ASIACRYPT. Volume 7658 of Lecture Notes in Computer Science., Springer (2012) 226–243
15. Li, J., Isobe, T., Shibutani, K.: Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to sha-2. In Canteaut, A., ed.: FSE. Volume 7549 of Lecture Notes in Computer Science., Springer (2012) 264–286
16. Mendel, F., Nad, T., Schläffer, M.: Finding collisions for round-reduced sm3. In Dawson, E., ed.: CT-RSA. Volume 7779 of Lecture Notes in Computer Science., Springer (2013) 174–188
17. Mendel, F., Nad, T., Schläffer, M.: Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In Lee, D.H., Wang, X., eds.: ASIACRYPT. Volume 7073 of Lecture Notes in Computer Science., Springer (2011) 288–307
18. Mendel, F., Nad, T., Schläffer, M.: Collision Attacks on the Reduced Dual-Stream Hash Function RIPEMD-128. [3] 226–243
19. Mendel, F., Rechberger, C., Schläffer, M.: MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners. In Matsui, M., ed.: ASIACRYPT. Volume 5912 of Lecture Notes in Computer Science., Springer (2009) 144–161
20. Mendel, F., Rechberger, C., Schläffer, M., Thomsen, S.S.: The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Dunkelman, O., ed.: FSE. Volume 5665 of Lecture Notes in Computer Science., Springer (2009) 260–276
21. 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. Volume 5932 of Lecture Notes in Computer Science., Springer (2009) 206–225
22. Mouha, N., Velichkov, V., De Cannière, C., Preneel, B.: The Differential Analysis of S-Functions. In Biryukov, A., Gong, G., Stinson, D.R., eds.: Selected Areas in Cryptography. Volume 6544 of Lecture Notes in Computer Science., Springer (2010) 36–56
23. Peyrin, T.: Analyse de fonctions de hachage cryptographiques. PhD thesis, University of Versailles (2008)
24. Schläffer, M., Oswald, E.: Searching for Differential Paths in MD4. In Robshaw, M.J.B., ed.: FSE. Volume 4047 of Lecture Notes in Computer Science., Springer (2006) 242–261
25. 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. Volume 4515 of Lecture Notes in Computer Science., Springer (2007) 1–22
26. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A.K., Molnar, D., Osvik, D.A., de Weger, B.: Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. [12] 55–69
27. Su, B., Wu, W., Wu, S., Dong, L.: Near-Collisions on the Reduced-Round Compression Functions of Skein and BLAKE. In Heng, S.H., Wright, R.N., Goi, B.M., eds.: CANS. Volume 6467 of Lecture Notes in Computer Science., Springer (2010) 124–139
28. Wang, X., Lai, X., Feng, D., Chen, H., Yu, X.: Cryptanalysis of the Hash Functions MD4 and RIPEMD. [5] 1–18
29. Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. In Shoup, V., ed.: CRYPTO. Volume 3621 of Lecture Notes in Computer Science., Springer (2005) 17–36
30. Wang, X., Yu, H.: How to Break MD5 and Other Hash Functions. [5] 19–35
31. Yu, H., Chen, J., Jia, K., Wang, X.: Near-Collision Attack on the Step-Reduced Compression Function of Skein-256. IACR Cryptology ePrint Archive, Report 2011/148 (2011)

32. Yu, H., Chen, J., Wang, X.: The Boomerang Attacks on the Round-Reduced Skein-512. In Knudsen, L.R., Wu, H., eds.: SAC. Lecture Notes in Computer Science, Springer (2012)
33. Yu, H., Chen, J., Wang, X.: Partial-Collision Attack on the Round-Reduced Compression Function of Skein-256. In Moriai, S., ed.: FSE. Lecture Notes in Computer Science, Springer (2013)

A Collision Pairs for reduced compression functions of Skein-256

In this section we give some examples of colliding pair for reduced versions of the compression function of Skein-256. The differential characteristics used to build those pair are given in the full version of the paper.

The inputs are $k = cv$ and $v_0 = m$. The output is $h = E_k(m) \oplus m$. The examples are given for $T = 0$.

Table 3. Semi-free-start collision for 12-round Skein-256 (rounds 0–12).

	Input 1	Input 2		Output 1	Output 2
k_0	968cb2e66b0fb527	968cb2e66b0fb527	$e_{12,0}$	2798a30c07459007	2398930c07459007
k_1	37fce3361809b06a	37fce3361809b06a	$e_{12,1}$	2410f135e024aace	2410e135e024aace
k_2	4bb032fb1894a60b	4bb032fb1894a60b	$e_{12,2}$	60490bbd9ddcb933	60490bbd9ddcb933
k_3	d917aa4640682db6	d917aa4640682db6	$e_{12,3}$	7fd51384c7b528f3	7fd51384c7b528f3
m_0	e7395021238d7d18	e3396021238d7d18	h_0	c0a1f32d24c8ed1f	c0a1f32d24c8ed1f
m_1	7229b06628958c1a	7229a06628958c1a	h_1	56394153c8b126d4	56394153c8b126d4
m_2	3ea410b0b8f1b533	3ea410b0b8f1b533	h_2	5eed1b0d252d0c00	5eed1b0d252d0c00
m_3	fc0aa7147201f560	fc0aa7147201f560	h_3	83dfb490b5b4dd93	83dfb490b5b4dd93

Table 4. Free-start collision for 20-round Skein-256 (rounds 0–20).

	Input 1	Input 2		Output 1	Output 2
k_0	5f977cfdd64d2f57	5f977cfdd64d2f57	$e_{20,0}$	6627a3d5c18e2057	6627a3d5c18e2057
k_1	35839193022be6f4	b5839193022be6f4	$e_{20,1}$	7a1eeeee92b2202d	fa1eeeee92b2202d
k_2	05e168930700458f	85e168930700458f	$e_{20,2}$	2bf3a5067fac9218	abf3a5067fac9218
k_3	6f47d57f8b6f9b78	6f47d57f8b6f9b78	$e_{20,3}$	b0ccc2f709dc2e35	b0ccc2f709dc2e35
m_0	627f37f95152438c	627f37f95152438c	h_0	0458942c90dc63db	0458942c90dc63db
m_1	0532b3fdf499d0d7	8532b3fdf499d0d7	h_1	7f2c5d13662bf0fa	7f2c5d13662bf0fa
m_2	91c792ab31ba535c	11c792ab31ba535c	h_2	ba3437ad4e16c144	ba3437ad4e16c144
m_3	72e80ac1aaee8118	72e80ac1aaee8118	h_3	c224c836a332af2d	c224c836a332af2d