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. We describe a more efficient way to propagate multi-bit constraints, that allows us to use the complete set of 2^{32} 2.5-bit constraints, instead of the reduced sets used by Leurent.

As a result, we are able to build complex non-linear differential characteristics for reduced versions of the hash function Skein. We present several characteristics for use in various attack scenarios; this results in attacks with a relatively low complexity, 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.

To the best of our knowledge, these are the first examples of complex differential trails built for pure ARX designs. We believe this is an important work to assess the security of ARX designs against differential cryptanalysis. Our improved tools will be publicly available with the final version of this paper.

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; differential characteristics are relatively easy to build, but efficient attacks often need novel techniques, such as the rebound attack against hash functions [MRST09]. 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 [WLF+05,WY05,WYY05,YCKW11], and an important effort has been devoted to building tools to construct automatically such characteristics [dCR06,SO06,FLN07a,MNS11,SLdW07]. 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 [FLN07b], the construction of a rogue MD5 CA certificate [SSA+09], and attacks against combiners [MRS09].

However, this body of work is mainly focused on MD/SHA designs, as opposed to pure ARX designs such as Skein or BLAKE. 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, and the freedom in the carry expansions is much harder to use than the absorption property of Boolean functions.

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

To this effect, Leurent introduced multi-bit constraints [Leu12] 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 carries extensions in modular additions.

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 [dCR06], and the multi-bit constraints of Leurent. We build upon those previous works and introduce a more efficient way to perform the constraint propagation in Section 2. We show how to use this constraint propagation tool in a differential characteristic search algorithm in Section 3, and we present our results on Skein in Section 4.

Constraint propagation. Our first result is an alternative way to perform the constraint propagation for multi-bit constraints. Our approach is significantly more efficient that the previous one, 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. 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.

Construction of differential characteristics. We use this new tool to construct of differential characteristics automatically. We show that we can actually build complex non-linear characteristics using some simple heuristics and our efficient constraint propagation tool.

To the best of our knowledge, this the first time complex differential trails are build for ARX designs (a previous attempt by Yu *et al.* [YCKW11] has been shown to be flawed [Leu12]). We believe this is an important result to assess the security of ARX designs against differential cryptanalysis.

Application to Skein. Finally, 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 work on Skein are either weak distinguishers (such as boomerang properties or free-tweak free-start near-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).

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 input X, X', and to specify the difference between x and x' for every internal state variable x. The difference can be an xor difference, a 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.

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

Extra Degrees of freedom		Ref	Rounds	Time	Note
Collision	0	[KRS11]	4	2^{96}	Biclique based
			8	2^{120}	
			9	2^{124}	
			12	$2^{126.5}$	
Free-start collision	8	[LIS12]	$22^{\dagger}$	$2^{253.8\dagger}$	Biclique based
			$37^{\dagger}$	$2^{255.7\dagger}$	
Free-tweak partial-collision	12	[YCKW11]	32	2^{105}	51 active bits — **Invalid characteristic**
Collision	0	4.4	12	$\approx 2^{114}$	
Semi-free-start collision	4	4.4	12	$\approx 2^{40}$	
Free-start collision	8	4.5	20	$\approx 2^{40}$	
Free-tweak near-collision	$10^{\star}$	4.6	24	$\approx 2^{40}$	3 active bits
Free-tweak partial-collision	$10^{\star}$	4.6	32	$\approx 2^{119}$	51 active bits

† Attacks on Skein-512. For Skein-256, fewer round will be attacked, with a complexity slightly below 2^{256}.
* We use freedom degrees in the tweak *difference*, but the tweak *value* can be arbitrary.

Signed bitwise difference. The groundbreaking results of Wang *et al.* [WLF+05,WY05,WYY05] 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 [dCR06]: for each bit of the state, they look at all possible values of the pair (x, x'), and they specify which values are allowed. This give a set of 16 constraints as shown in Table 2. 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 [Leu12]. 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 [Leu12]:

- 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 and 2^{32} constraint are not used because the propagation method of [Leu12] becomes impractical with such large sets.

2.1 Constraint Propagation and Probability Computation

In [Leu12], a set of constraints is represented by an S-system, and an automaton is built to compute the probability of a each operation. To perform constraints propagation, each constraint is split into two disjoint subsets; if one of the subsets result 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 access. However, the table become impractically large if the set of constraint is too large, or if the operation is too complex. In [Leu12], 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.

Table 2. Generalized 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	$x = x' = 1$	-	-	-	✓
#	*incompatible*	-	-	-	-
3	$x = 0$	✓	✓	-	-
5	$x' = 0$	✓	-	✓	-
7		✓	✓	✓	-
A	$x' = 1$	-	✓	-	✓
B		✓	✓	-	✓
C	$x = 1$	-	-	✓	✓
D		✓	-	✓	✓
E		-	✓	✓	✓

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 small tables fitting in the cache (the tables of [Leu12] take hundreds of megabytes); this gives good results in practice. In addition we show special properties of the automata allowing an efficient propagation algorithm without splitting the constraints into subsets.

2.2 Our New Approach

In this work we describe a method that is specific to systems of the following form:

$$\begin{gathered} u = f(a, b, c, \dots) \quad u' = f(a', b', c', \dots) \\ \delta(a, a') = A \quad \delta(b, b') = B \quad \delta(c, c') = C \quad \dots \\ \delta(u, u') = U, \end{gathered} \tag{1}$$

where f is an S-function, and the difference δ is given by a set of constraints which fully determines $x^{[i]}$, $x'^{[i]}$, $x^{[i-1]}$, and $x'^{[i-1]}$. We consider a, a', b, b' … as variables, and A, B, … U as parameters.

Building the automaton. To deal with 2.5-bit constraints, we use a base alphabet $\mathcal{B}$ of 32 constraints, each specifying one possible value for $x^{[i]}$, $x'^{[i]}$, $x^{[i-1]}$, $x'^{[i-1]}$, $x^{[i-2]}$ (for 2-bit constraints, the base alphabet has 16 constraints). Since the system given by (1) with the constraints in $\mathcal{B}$ is an S-system, we can compute a set of *states* $\mathcal{S}$, and a transition function:

$$\begin{aligned} \tau : \qquad \mathcal{S} \times (\mathcal{B} \times \{0,1\} \times \{0,1\})^{p-1} \times \mathcal{B} &\to \mathcal{S} \\ q, (\overline{A}, a, a'), (\overline{B}, b, b'), \dots, \overline{U} &\mapsto q' \end{aligned}$$

so that each solution to the system corresponds to a path in the automaton with transition function τ. More details about the construction of τ are given in [MVCP10,Leu12]. In our implementation, we use the tools of [Leu12] to compute the transition table.

When we describe a differential characteristic, we use an alphabet $\mathcal{A} = \mathcal{P}(\mathcal{B})$ consisting the 2^{32} subsets of the base alphabet $\mathcal{A}$ (2^{16} subsets for 2-bit constraints). We transform an automaton on the alphabet $\mathcal{B}$ to operate on the alphabet $\mathcal{A}$ by changing the transition function into a non-deterministic transition function:

$$\begin{aligned} \tau' : \qquad & S \times (\mathcal{A} \times \{0,1\} \times \{0,1\})^{p-1} \times \mathcal{A} \to \mathcal{P}(S) \\ & q, (A, a, a'), (B, b, b'), \dots, U \mapsto \bigcup_{\overline{A} \in A, \dots, \overline{U} \in U} \tau\left(q, (\overline{A}, a, a'), \dots, \overline{U}\right) \end{aligned}$$

This automaton can test whether the constraints are satisfied for given values of the parameters $A, B, \dots, U$, of the variables $a, a', b, b', \dots$, and with $u = f(a, b, c, \dots)$, $u' = f(a', b', c', \dots)$. We further transform the automaton be removing the information about $a, a', \dots$:

$$\begin{aligned} \tau'' : \qquad & S \times \mathcal{A}^p \to \mathcal{P}(S) \\ & q, A, B, \dots, U \mapsto \bigcup_{a, a', b, b', \dots \in \{0,1\}} \tau'\left(q, (A, a, a'), \dots, U)\right) \end{aligned}$$

This new automaton can decide whether there exists solutions to System (1) for given parameters $A, B, \dots, U$. The transition function is highly non-deterministic, but we still use the original automaton by relabelling the transitions, and reading several transitions at each step.

Lemma 1. *The transition automaton of a system following (1) with p parameters, v variables, and s bits of state has the following properties:*

i) *Each state can be labelled with a 1-bit value for value of a, a', b, b' $\dots$, x, x'. All the input transitions share this value for $a^{[i]}$, $a'^{[i]}$, $b^{[i]}$, $b'^{[i]}$ $\dots$, $x^{[i]}$, $x'^{[i]}$, while all the output transitions share this value for $a^{[i-1]}$, $a'^{[i-1]}$, $b^{[i-1]}$, $b'^{[i-1]}$ $\dots$, $x^{[i-1]}$, $x'^{[i-1]}$.*
ii) *No pair of states are linked by two different transitions;*
iii) *Each state has exactly 2^{2v} output transitions (the transition table is sparse);*

Proof. *i)* In order to reject incoherent constraints for bit $i-1$ and i of a variable, the automaton must store the values of the previous bits that are used for the constraint on bit i in the state.
ii) Let's assume we have two transitions from a state q to a state q'. Since the two transition go to the same state, they must specify the same values of the parameters on bit i. Moreover, the two transition come from the same state, so they must also specify the same values on bits before i. Therefore the two transitions are the same.
iii) Because the system follows the form $x = f(a, b, c, \dots), x' = f(a', b', c', \dots)$, any choice of the variables a, a', b, b', $\dots$ is valid with exactly one value of x, x'.

Propagation. We use the properties of Lemma 1 in order to build an efficient propagation algorithm. Thanks to property *ii)*, we have a one to one correspondence between the paths in the original automaton, and the paths in the relabelled automaton. Therefore we can easily identify the constraints corresponding to actual solutions of the system. To propagate constraints, we first build the set of paths allowed by the initial constraints, we look at which edges are actually used in paths, and we build the new constraints by identifying the constraints corresponding to the edges.

Notations. We use the symbols from [Leu12] to denote the most common constraints as shown in Table 5. When a characteristic uses a less common constraint, we use an hexadecimal mask to represent it. The less common constraints used in the characteristics given in Appendix C are given in Table 9.

When the constraints on the current bit and the constraints on previous bits are independent, we write the constraints involving previous bit in exponent (*e.g.* see Figure 7). For instance, we have can write the constraints `<` as $\mathtt{u}^{\mathtt{u}} \cup \mathtt{n}^{\mathtt{n}}$.

2.3 Propagation for a Differential Characteristic

A differential characteristic is given by a set a constraints for each internal state variable. An ARX design (or a more general MD/SHA-like desing) is built with two kinds of operations:

- Operations that are **S-functions**: additions, xors, and bitwise Boolean function. We build a system for each operation following (1), and we use them to propagate constraints between the inputs and the output of the operation (the propagation goes both ways). To propagate a full characteristic, we propagate every operation until no new constraints are found.
- **Rotations**: since the constraints are local and only involve consecutive bits, we deal with a rotation $y = x \ggg i$ by just rotating the constraint pattern: if $\delta x = \Delta_x$ then we use $\delta y = \Delta_x \ggg i$. However, we have to relax some constraints if the multi-bit relations are broken by the rotation.

2.4 Propagation Example

Let us show how the propagation operates with a simple example. For this example, we use 2-bit constraints, and we consider the operation $u = a \vee (a \boxplus a)$. The leads to the following system:

$$u = a \vee (a \boxplus a) \quad u' = a' \vee (a' \boxplus a') \qquad (2)$$
$$\delta(a, a') = A \quad \delta(u, u') = U.$$

This system has 2 parameters, 2 variables and 4 bits of state (two for each δ operation; the state of $a \boxplus a$ is already included in the state of $\delta(a, a')$). The automaton corresponding to this system is given in Figure 7. Note that the automaton only needs 9 states out of the $2^4 = 16$ possible values for the state of the S-system. In our work we always minimize the automata, and this usually results in a significant reduction of the number of states. We can verify that Lemma 1 is respected.

We will show how the propagation algorithm works with the following input:

$$\delta(a, a') = \texttt{-x--} \qquad \delta(u, u') = \texttt{----}. \qquad (3)$$

This correspond to a situation where an input difference must be absorbed through the operation.

We first build a graph with a copy of the transitions for each bit. Then for each bit, we remove the transitions that are not acceptable according to the initial constraints (3). More precisely, we only keep constraints that are subsets of `-/-` for the first and second bits, subsets of `x/-` for the third bit, and subsets of `-/-` for the fourth bit. We get the graph of Figure 8, and we look for paths starting for state 0 in the initial layer, and ending in any state of the final layer. (Note that the least significant bit is on the left in the graph, but on the right when we write $\delta x = \texttt{-x--}$). The nodes and edges involved in these paths are shown in black. We note that the constraints are compatible because such paths exists, and we can count the number of paths to compute the number of solutions: there are 4 different paths in the graph, so the are 4 different solutions to System (2) satisfying (3). We can read the solution by following the paths:

$\delta(a, a')$:	1n10	1u10	1n11	1u11
$\delta(u, u')$:	1110	1110	1111	1111

Let us now do the constraint propagation. For each bit, we look at the active edges in Figure 8, and we list the corresponding constraints for a and u in Table 4. The new constraints will be the union of all the active constraints. We get the following output (we disregard restriction on previous bits for bit 0):

$$\delta(a, a') = \texttt{1}^{\texttt{x}}\texttt{x}^{\texttt{1}}\texttt{1}^{\texttt{-}}\texttt{-} \qquad \delta(u, u') = \texttt{1}^{\texttt{1}}\texttt{1}^{\texttt{1}}\texttt{1}^{\texttt{-}}\texttt{-}.$$

Here, the constraints on previous bits do not add any information, so we can omit them:

$$\delta(a, a') = \texttt{1x1-} \qquad \delta(u, u') = \texttt{111-} \tag{4}$$

It is easy to verify that any solution to the System (2) satisfying the initial constraints (3) also satisfies the deduced constraints.

2.5 Comparison with Previous Works

We show a comparison of our approach with previous methods in Table 3. We use the same settings as [Leu12]:

1. a reduced Skein with two rounds and 4 words of 4 bits each;
2. a reduced Skein with three rounds and 4 words of 6 bits each.

These experiments show that using the full set of 2.5-bit constraints gives better result than using the reduced set of [Leu12]. 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.

Table 3. 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.

	2 rounds / 4 bits (total: 2^{32})		3 rounds / 6 bits (sparse[1])		
Method	Accepted	F pos.	Accepted	F pos.	Time[2]
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 [Leu12]	$2^{25.3}$ (40820032)	0.14	$2^{19.5}$ (746742)	0.7	50 ms
1.5-bit reduced set [Leu12]	$2^{25.3}$ (40820032)	0.14	$2^{20.4}$ (1372774)	2.2	0.5 ms
1-bit constraints [dCR06]	$2^{25.4}$ (43564288)	0.21	$2^{20.7}$ (1762857)	3.1	0.5 ms
Check adds independently	$2^{25.8}$ (56484732)	0.57			

[1] Weight 4 differences. The total number of input/output differences is $\left(\sum_{i=0}^{4}\binom{24}{i}\right)^2 \approx 2^{26.75}$.
[2] Average time to verify one input/output difference (over the false positives of the 1.5-bit reduced set).

3 Automatic Construction of Differential Characteristics

In order to build 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 attack by X. Wang and her team are based on differential characteristics build by hand [WLF+05,WY05,WYY05,YCKW11], while later works gave algorithms to build such characteristics automatically [dCR06,SO06,FLN07a,MNS11,SLdW07].

In this section, we show that the multi-bit constraints can be used to design a successful algorithm for this task on ARX designs. Our algorithm is heavily inspired by the pioneer work of de Cannière and Rechberger [dCR06], and the more detailed explanation given in [Pey08] and [MdCIP09].

[1] The comparison is done with similar implementations.

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 design 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 [dBB93], and the recent work of Dunkelman and Khovratovich on BLAKE [DK11]. 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 build using local collisions [CJ98,WYY05], 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 [MRST09]: 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 been proposed in [YCKW11], but the path they give has been shown to be flawed.

3.2 Algorithm

Our algorithm takes as input a characteristic representing the high-probability parts of the trail $\Delta_1 \to \Delta_2$ and $\Delta_3 \to \Delta_4$. 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, as illustrated in Figure 1:

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 build from the idea that the constraint propagation is relatively efficient to check if a transition $\Delta \to \Delta'$ is possible. Therefore to connect the differences Δ_2 and Δ_3 from the high-probability trails, we essentially guess the middle difference Δ' and we check whether the transitions $\Delta_2 \to \Delta'$ and $\Delta' \to \Delta_3$ are possible.

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

- 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 biased toward this choice.
- We only use signed differences, *i.e.* we use the constraints `-`, `u`, and `n`.

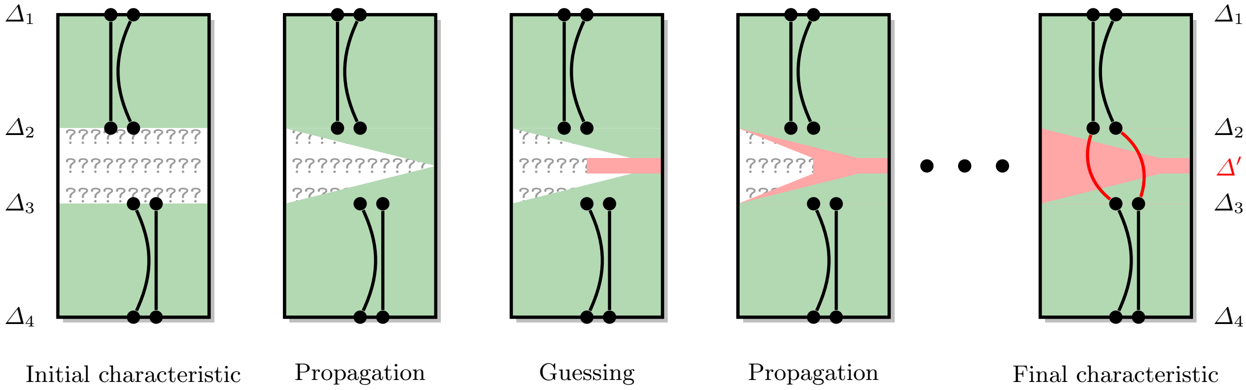


Fig. 1. Overview of the search algorithm. We start with high-probability trails $\Delta_1 \rightarrow \Delta_2$ and $\Delta_3 \rightarrow \Delta_4$, and we connect them through a difference Δ'

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

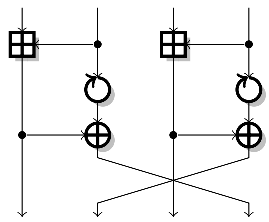


Fig. 2. Threefish-256 round

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 round function is shown by Figure 2. 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 `MIX` transformation and a permutation of 4 words as following:

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

where σ is the permutation (0 3 2 1) (specified in [FLS$^+$10]) and $(c, d) = \texttt{MIX}_{r,j}(a, b)$ is defined as:

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

The rotations $R_{r \bmod 8,j}$ are specified in [FLS+10]. The key scheduling algorithm of Threefish produces the round keys from a tweak (t_0, t_1) and a key as following:

$$k_{l,0} = k_{(l\quad) \bmod 5} \qquad\qquad k_{l,1} = k_{(l+1) \bmod 5} + t_{l \bmod 3}$$
$$k_{l,2} = k_{(l+2) \bmod 5} + t_{(l+1) \bmod 3} \qquad\qquad k_{l,3} = k_{(l+3) \bmod 5} + l,$$

where $k_4 = C_{240} \oplus \bigoplus_{i=0}^{4} k_i$ with C_{240} a constant specified in [FLS+10], 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 attack on multiples of four rounds, because the structure of Skein is build 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 attack are based on different kinds of trails shown in Figures 4, 5, and 6, and examples of characteristics are given in Tables 10, 11, and 12, respectively. All the characteristics have been verified by building a conforming pair.

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 take 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 attacks 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 active bit in order to avoid a few 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 degree of freedom to build a valid trail without using degrees of freedom from the key: for arbitrary differences Δ_2 and Δ_3, we expect on average a single pair satisfying $f(x + \Delta_2) = f(x) + \Delta_3$ and that would hardly be a differential trail.

In order to let the algorithm use the degree 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 all the 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 of [YCKW11] built by hand by Yu *et al.* was quite sparse, but it has been shown to be invalid [Leu12]. It remains an open question to see whether valid sparse trails can be built.

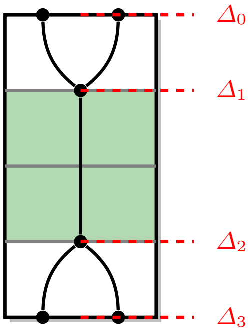


Fig. 3. Previous trails: rel-key, rel-tweak.

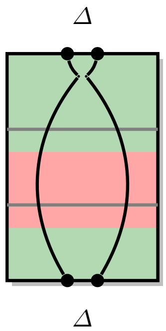


Fig. 4. Collision trails: fixed key.

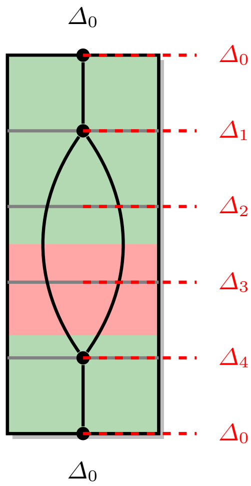


Fig. 5. Collision trails: related-key.

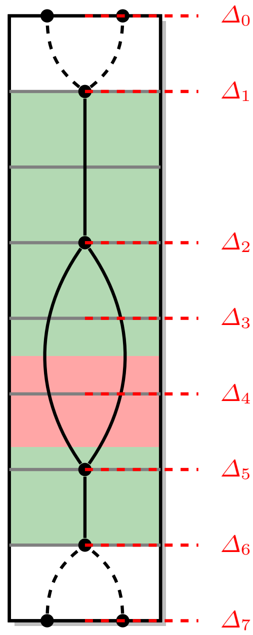


Fig. 6. (Near-)Collision trails: rel-key, rel-tweak.

4.4 Collision Attacks

We first study attacks with no difference in the key (*i.e.* the chaining value) so that they can be applied to the full hash function. 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]. 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 show an example of such characteristic in Table 10. This characteristic can be used for a practical semi-free-start collision attack on 12-round Skein, and we give an example of collision in Table 6.

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.

Our experiments indicate that we can build characteristics with about 2^{70} solutions for a cost of 2^{40}. If we extrapolate this experimental result, we expect that it is possible to build many

[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.

such characteristics. Let's assume that we can build N characteristics for a cost of $N \times 2^{40}$; each characteristic has 2^{70} solutions out of 2^{150} 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 $M \times N \times 2^{150-256}$ will be valid for one characteristic, and $M \times N \times 2^{70-256}$ 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 $N = 2^{73}$ and $M = 2^{113}$. With these parameters we only have to verify 2^{80} 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^{114}, using memory to store 2^{73} characteristics[3]. We believe that this estimation is a safe upper bound, and that better characteristics can be found be running the search algorithm for longer times.

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 key (*i.e.* the chaining value). 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 5: 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 characteristic build using this idea in Table 11, and a collision pair in Table 7.

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 has 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 6. This is the approach used in [YCKW11].

We give a characteristic build using this idea in Table 12, and an example of pair following the characteristic in rounds 4 to 28 in Table 8. 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 [YCKW11] 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 if about 2^{20}. Using the analysis of [YCKW11], we would build a conforming pair for rounds 0 to 32 for a cost of $2^{20+43+43} = 2^{119}$. That is comparable with the complexity given in [YCKW11], but this work is based on an incompatible trail.

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 with 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.

[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

This seems to be the first time complex differential trails are built 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.

Acknowledgement

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

Gaëtan Leurent was supported by the AFR grant PDR-10-022 of the FNR Luxembourg, and is supported by the CRASH ERC grant from the European Union.

References

CJ98. Florent Chabaud and Antoine Joux. Differential Collisions in SHA-0. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 56–71. Springer, 1998.

Cra05. Ronald Cramer, editor. *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*. Springer, 2005.

dBB93. Bert den Boer and Antoon Bosselaers. Collisions for the Compressin Function of MD5. In Tor Helleseth, editor, *EUROCRYPT*, volume 765 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1993.

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.

DK11. Orr Dunkelman and Dmitry Khovratovich. Iterative differentials, symmetries, and message modification in BLAKE-256. In *ECRYPT2 Hash Workshop*, 2011.

FLN07a. Pierre-Alain Fouque, Gaetan Leurent, and Phong Nguyen. Automatic Search of Differential Path in MD4. ECRYPT Hash Worshop – Cryptology ePrint Archive, Report 2007/206, 2007. `http://eprint.iacr.org/`.

FLN07b. Pierre-Alain Fouque, Gaëtan Leurent, and Phong Q. Nguyen. Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 13–30. Springer, 2007.

FLS+10. Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The Skein hash function family. Submission to NIST, 2008/2010.

KRS11. Dmitry Khovratovich, Christian Rechberger, and Alexandra Savelieva. Bicliques for preimages: Attacks on skein-512 and the sha-2 family. Cryptology ePrint Archive, Report 2011/286, 2011. `http://eprint.iacr.org/`.

Leu12. Gaëtan Leurent. Analysis of Differential Attacks in ARX Constructions. In *Asiacrypt*, 2012.

LIS12. Ji Li, Takanori Isobe, and Kyoji Shibutani. Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to sha-2. In Anne Canteaut, editor, *FSE*, volume 7549 of *Lecture Notes in Computer Science*, pages 264–286. Springer, 2012.

MdCIP09. Nicky Mouha, Christophe de Cannière, Sebastiaan Indesteege, and Bart Preneel. Finding Collisions for a 45-Step Simplified HAS-V. In Heung Youl Youm and Moti Yung, editors, *WISA*, volume 5932 of *Lecture Notes in Computer Science*, pages 206–225. Springer, 2009.

MNS11. Florian Mendel, Tomislav Nad, and Martin Schläffer. Finding SHA-2 Characteristics: Searching through a Minefield of Contradictions. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT*, volume 7073 of *Lecture Notes in Computer Science*, pages 288–307. Springer, 2011.

MRS09. Florian Mendel, Christian Rechberger, and Martin Schläffer. MD5 Is Weaker Than Weak: Attacks on Concatenated Combiners. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2009.

MRST09. Florian Mendel, Christian Rechberger, Martin Schläffer, and Søren S. Thomsen. The Rebound Attack: Cryptanalysis of Reduced Whirlpool and Grøstl. In Orr Dunkelman, editor, *FSE*, volume 5665 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2009.

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.

Pey08. Thomas Peyrin. *Analyse de fonctions de hachage cryptographiques*. PhD thesis, University of Versailles, 2008.

SLdW07. Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2007.

SO06. Martin Schläffer and Elisabeth Oswald. Searching for Differential Paths in MD4. In Matthew J. B. Robshaw, editor, *FSE*, volume 4047 of *Lecture Notes in Computer Science*, pages 242–261. Springer, 2006.

SSA+09. Marc Stevens, Alexander Sotirov, Jacob Appelbaum, Arjen K. Lenstra, David Molnar, Dag Arne Osvik, and Benne de Weger. Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate. In Shai Halevi, editor, *CRYPTO*, volume 5677 of *Lecture Notes in Computer Science*, pages 55–69. Springer, 2009.

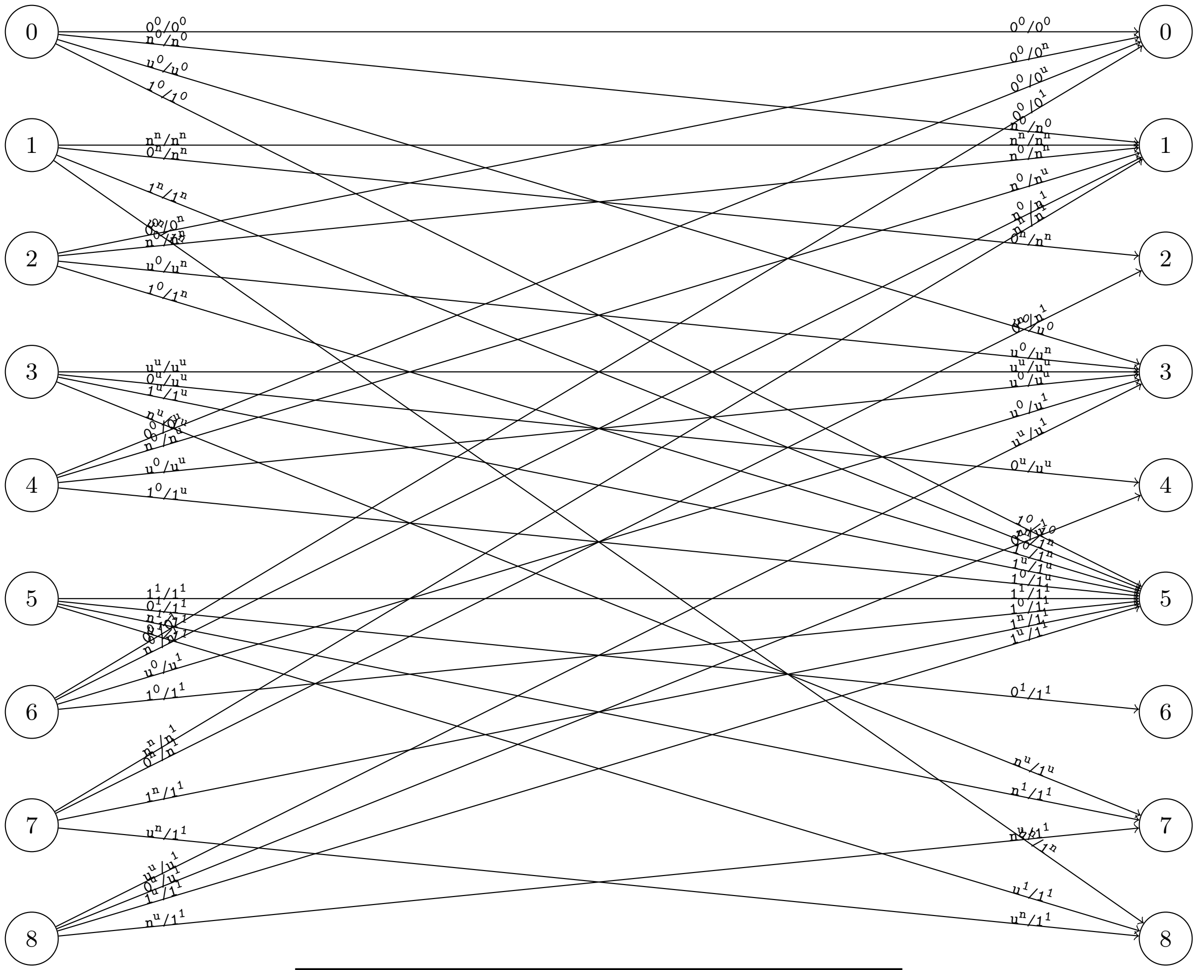
WLF+05. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the Hash Functions MD4 and RIPEMD. In Cramer [Cra05], pages 1–18.

WY05. Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions. In Cramer [Cra05], pages 19–35.

WYY05. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2005.

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/`.

A Constraint Propagation



State	Transitions			
0	$0 \xrightarrow{0^0/0^0} 0$	$0 \xrightarrow{n^0/n^0} 1$	$0 \xrightarrow{u^0/u^0} 3$	$0 \xrightarrow{1^0/1^0} 5$
1	$1 \xrightarrow{n^n/n^n} 1$	$1 \xrightarrow{0^n/n^n} 2$	$1 \xrightarrow{1^n/1^n} 5$	$1 \xrightarrow{u^n/1^n} 8$
2	$2 \xrightarrow{0^0/0^n} 0$	$2 \xrightarrow{n^0/n^n} 1$	$2 \xrightarrow{u^0/u^n} 3$	$2 \xrightarrow{1^0/1^n} 5$
3	$3 \xrightarrow{u^u/u^u} 3$	$3 \xrightarrow{0^u/u^u} 4$	$3 \xrightarrow{1^u/1^u} 5$	$3 \xrightarrow{n^u/1^u} 7$
4	$4 \xrightarrow{0^0/0^u} 0$	$4 \xrightarrow{n^0/n^u} 1$	$4 \xrightarrow{u^0/u^u} 3$	$4 \xrightarrow{1^0/1^u} 5$
5	$5 \xrightarrow{1^1/1^1} 5$	$5 \xrightarrow{0^1/1^1} 6$	$5 \xrightarrow{n^1/1^1} 7$	$5 \xrightarrow{u^1/1^1} 8$
6	$6 \xrightarrow{0^0/0^1} 0$	$6 \xrightarrow{n^0/n^1} 1$	$6 \xrightarrow{u^0/u^1} 3$	$6 \xrightarrow{1^0/1^1} 5$
7	$7 \xrightarrow{n^n/n^1} 1$	$7 \xrightarrow{0^n/n^1} 2$	$7 \xrightarrow{1^n/1^1} 5$	$7 \xrightarrow{u^n/1^1} 8$
8	$8 \xrightarrow{u^u/u^1} 3$	$8 \xrightarrow{0^u/u^1} 4$	$8 \xrightarrow{1^u/1^1} 5$	$8 \xrightarrow{n^u/1^1} 7$

Fig. 7. Transitions for System (2)

Table 4. Active edges in figure 8, and new deduced constraints.

i	edges ($\delta a/\delta u$)	a constraints		u constraints	
0	$0^0/0^0$, $1^0/1^0$	$0^0 \cup 1^0$	$\equiv -^0$	$0^0 \cup 1^0$	$\equiv -^0$
1	$1^0/1^0$, $1^1/1^1$	$1^0 \cup 1^1$	$\equiv 1^-$	$1^0 \cup 1^1$	$\equiv 1^-$
2	$n^1/1^1$, $u^1/1^1$	$n^1 \cup u^1$	$\equiv x^1$	$1^1 \cup 1^1$	$\equiv 1^1$
3	$1^n/1^1$, $1^u/1^1$	$1^n \cup 1^u$	$\equiv 1^x$	$1^1 \cup 1^1$	$\equiv 1^1$

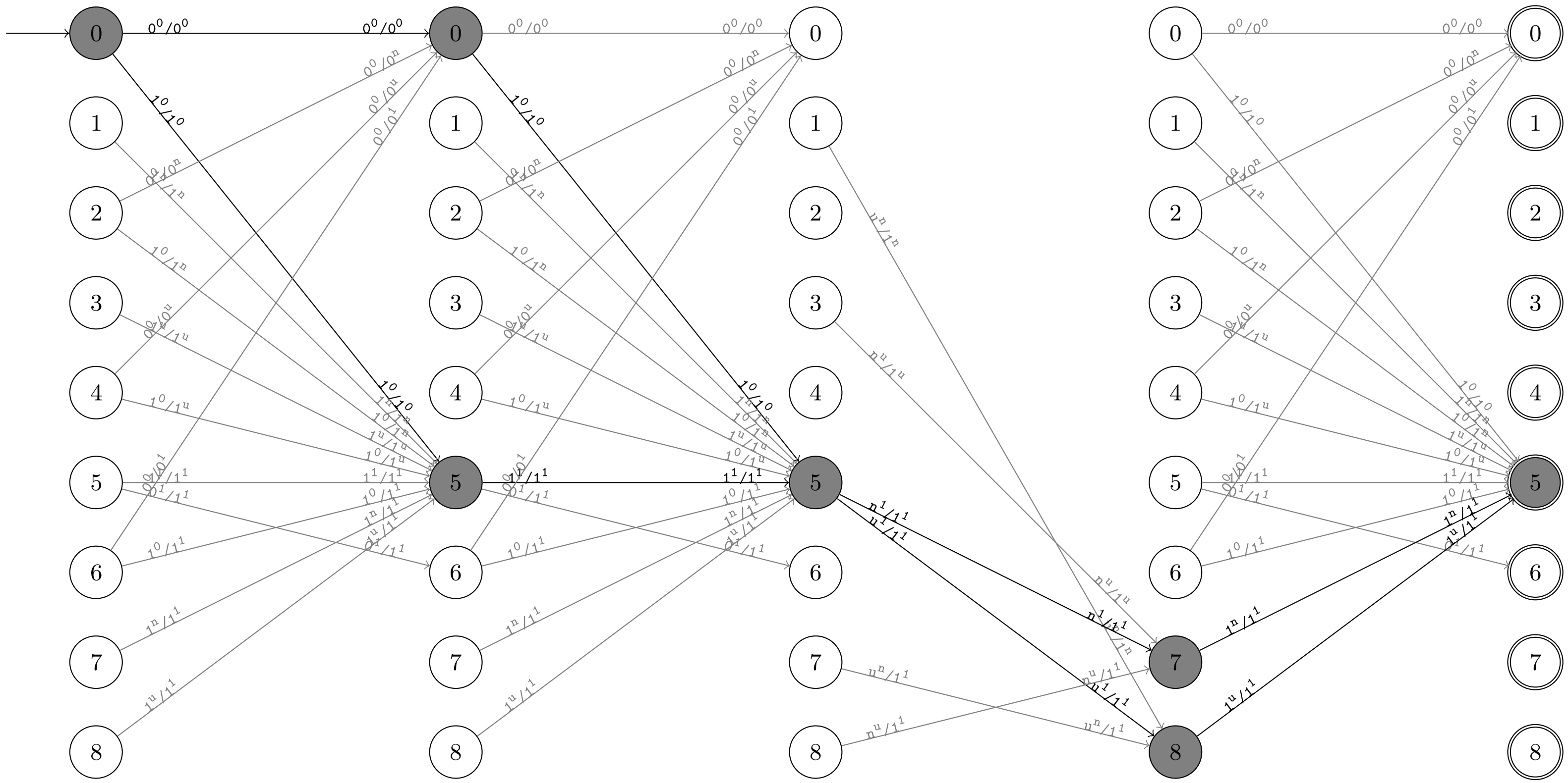


Fig. 8. Graph representation of System (2) with initial constraints (3)

$(x, x', 2x, 2x', 4x)$:	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111
?	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓
x	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
0	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
u	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
n	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓
#	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-
A	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓
B	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓
C	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
D	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
E	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
=	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓
!	-	-	-	-	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	-	-	-	-
<	-	-	-	-	-	-	-	-	✓	✓	✓	✓	-	-	-	-	-	-	-	-	✓	✓	✓	✓	-	-	-	-	-	-	-	-
>	-	-	-	-	-	-	-	-	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	-	-	-	-	-	-	-	-	-	-	-	-
X	✓	✓	✓	✓	✓	✓	✓	✓	-	-	✓	✓	✓	✓	-	-	-	-	✓	✓	✓	✓	-	-	✓	✓	✓	✓	✓	✓	✓	✓
U	✓	✓	-	-	✓	✓	✓	✓	-	-	✓	✓	-	-	-	-	-	-	✓	✓	-	-	-	-	✓	✓	-	-	✓	✓	✓	✓
N	✓	✓	✓	✓	-	-	✓	✓	-	-	-	-	✓	✓	-	-	-	-	-	-	✓	✓	-	-	✓	✓	✓	✓	-	-	✓	✓
/	✓	✓	-	✓	✓	-	✓	✓	-	-	✓	-	-	✓	-	-	-	-	✓	-	-	✓	-	-	✓	✓	-	✓	✓	-	✓	✓
\	-	-	✓	-	-	✓	-	-	-	-	-	✓	✓	-	-	-	-	-	-	✓	✓	-	-	-	-	-	✓	-	-	✓	-	-
V	-	-	✓	✓	-	-	-	-	✓	✓	-	-	✓	✓	✓	✓	✓	✓	-	-	✓	✓	✓	✓	-	-	✓	✓	-	-	-	-
M	-	-	-	-	✓	✓	-	-	✓	✓	✓	✓	-	-	✓	✓	✓	✓	✓	✓	-	-	✓	✓	-	-	-	-	✓	✓	-	-

Table 5. Constraints identified in [Leu12] and written as full 2.5-bit constraints.

B (Near-)Collision Pairs for Skein-256

Table 6. Semi-free-start collision for 12-round Skein-256. This pair is the same as given in Table 10.

	Input 1	Input 2	Difference	Weight
m_0	97c787b0252f1bef	97c787b0252f1bef	0000000000000000	0
m_1	9ba673bd9a918263	9ba673bd9a918263	0000000000000000	0
m_2	59f24b2909ae5223	59f24b2909ae5223	0000000000000000	0
m_3	963151773356523a	963151773356523a	0000000000000000	0
t_0	0000000000000000	0000000000000000	0000000000000000	0
t_1	0000000000000000	0000000000000000	0000000000000000	0
$v_{4,0}$	b9ded48b4e413597	39ded48b4e413597	8000000000000000	1
$v_{4,1}$	5a63d56d9481f1d6	5a63d56d9481f1d6	0000000000000000	0
$v_{4,2}$	0accb31ed067ae77	0accb31ed067ae77	0000000000000000	0
$v_{4,3}$	734e405bed9d64cc	734e405bed9d64cc	0000000000000000	0
	Output 1	**Output 2**	**Difference**	**Weight**
$e_{16,0}$	f3424f9d5f6d8c50	73424f9d5f6d8c50	8000000000000000	1
$e_{16,1}$	74a4ddb5e6e65d54	74a4ddb5e6e65d54	0000000000000000	0
$e_{16,2}$	bc4c51d904f3425d	bc4c51d904f3425d	0000000000000000	0
$e_{16,3}$	b511e49ca126be77	b511e49ca126be77	0000000000000000	0

Table 7. Free-start collision for 20-round Skein-256. This pair is the same as given in Table 11.

	Input 1	Input 2	Difference	Weight
m_0	5f977cfdd64d2f57	5f977cfdd64d2f57	0000000000000000	0
m_1	35839193022be6f4	b5839193022be6f4	8000000000000000	1
m_2	05e168930700458f	85e168930700458f	8000000000000000	1
m_3	6f47d57f8b6f9b78	6f47d57f8b6f9b78	0000000000000000	0
t_0	0000000000000000	0000000000000000	0000000000000000	0
t_1	0000000000000000	0000000000000000	0000000000000000	0
$v_{0,0}$	627f37f95152438c	627f37f95152438c	0000000000000000	0
$v_{0,1}$	0532b3fdf499d0d7	8532b3fdf499d0d7	8000000000000000	1
$v_{0,2}$	91c792ab31ba535c	11c792ab31ba535c	8000000000000000	1
$v_{0,3}$	72e80ac1aaee8118	72e80ac1aaee8118	0000000000000000	0
	Output 1	**Output 2**	**Difference**	**Weight**
$e_{20,0}$	6627a3d5c18e2057	6627a3d5c18e2057	0000000000000000	0
$e_{20,1}$	7a1eeeee92b2202d	fa1eeeee92b2202d	8000000000000000	1
$e_{20,2}$	2bf3a5067fac9218	abf3a5067fac9218	8000000000000000	1
$e_{20,3}$	b0ccc2f709dc2e35	b0ccc2f709dc2e35	0000000000000000	0

Table 8. Pair of input with low-weight difference for 32-round Skein-256. This pair is the same as given in Table 12; we don't specify how the differences are propagated in rounds 0 to 4 and 28 to 32.

	Input 1	Input 2	Difference	Weight
m_0	edb22ce30810011a	edb22ce30810011a	0000000000000000	0
m_1	08142e9044b0054a	08142e9044b0054a	0000000000000000	0
m_2	1e06bd5779535f97	1e06bd5779535f97	0000000000000000	0
m_3	82a5e785e5c5b836	02a5e785e5c5b836	8000000000000000	1
t_0	0000000000000000	8000000000000000	8000000000000000	1
t_1	0000000000000000	0000000000000000	0000000000000000	0
$v_{0,0}$	c0097c86ad089acd	c0decb29fae7a20d	00d7b7af57ef38c0	35
$v_{0,1}$	0eef94c587c9f8fc	91efc569f9eaf0fc	9f0051ac7e230800	23
$v_{0,2}$	a5333c6b7af97e18	a272f89740fdbae4	0741c4fc3a04c4fc	28
$v_{0,3}$	49df6d34f9ebc32f	cc9f6d0935eb8663	8540003dcc00454c	19

	Output 1	Output 2	Difference	Weight
$e_{32,0}$	650f11ac87162f96	650f119c82f63796	0000003005e01800	9
$e_{32,1}$	22ed455a3e3dd26a	e5f12d8d8431cafa	c71c68d7ba0c1890	28
$e_{32,2}$	ef0d1179583e8671	ed0d118994327e51	020000f0cc0cf820	17
$e_{32,3}$	5de99dad57671f6a	5ec99dbd5347076a	0320001004201800	8

C Differential Characteristics for Skein-256

The characteristics given in Tables 10, 11, and 12 follow the general structure described in Figures 4, 5, and 6. For more details of the attacks, see Sections 4.4, 4.5, and 4.6, respectively.

We use the following colors in the characteristics:

- red constraints for active bits;
- green constraints for inactive bits;
- orange constraints for carry bits (inactive if the previous bit is inactive);
- blue constraints for other situations.

The most common characteristics are given in Table 5, while the unusual one are assigned a two digit code, and given in Table 9 in hexadecimal notation. The 32-bit hexadecimal values correspond to the columns of Table 5; for instance the constraint `N` would be represented by `f30c0cf3`. The two-digit codes are just used as shorthand so that all the information for the trails fit in the tables.

When using those characteristics, we start with the middle state given by the characteristic, we select a key satisfying the key constraints, and we check the remaining rounds. Therefore, the probabilities given for the upper rounds are probability in the backward direction, while probabilities in the lower round are in the forward direction.

When the tweak is not given in the characteristic, it should be taken as zero.

Table 9. Description of the uncommon constraints used in the characteristics

Sym.	Mask	Sym.	Mask	Sym.	Mask	Sym.	Mask
00	00fff200	01	00f08f00	02	0001ff00		
10	cf003000	11	000c0cf3	12	cf300000	13	f00f0ff0
14	0f00f000	15	00f0f00f	16	cf303000	17	c030000f
18	8030004f	19	c330000f	1a	0ff0100f	1b	00f0000f
1c	003030cf	1d	810c0cf3	1e	000030c0	1f	000f0ff0
20	730c00f2	21	cf3030cc	22	0ff0f00f	23	f30c04e0
24	0000f00f	25	030c0000	26	0ff00000	27	0f102000
28	0400f008	29	0f00f00f	2a	f00f0000	2b	000030cf
2c	cf3030c0	2d	c0300000	2e	00000cf3	2f	830c00f2
30	030c0c03	31	00000ff0	32	0f0030c0	33	c03030cf
34	c03030c0						
40	3001cf30	41	30c0c030	42	0cf3f300	43	00f3f30c
44	00f0030c	45	000f00f0	46	0cf30000	47	30c04f00
48	00f3000c	49	0000c030	4a	30cf0000	4b	0c03f200
4c	00cf0030	4d	00cf8030	4e	0c030000		
50	8000004f	51	8f0000ff	52	f000004f	53	010000f2
54	ff000081	55	4f0000f2	56	f00000ff	57	4f0000f0
58	f200004f	59	810000ff	5a	800000ff	5b	ff0000f2
5c	f200000f	5d	0f0000ff	5e	ff000080	5f	020000ff
60	f2000001	61	ff0000f0	62	ff00000f	63	f10000ff
64	4f0000ff	65	420000ff	66	ff000040	67	ff000002
68	0f0000f2	69	0f0000f1	6a	4f000080	6b	010000ff
6c	ff00004f	6d	4000008f	6e	ff0000f1	6f	ff000004
70	080000ff	71	ff000001	72	04000008	73	ff00008f
74	f20000ff	75	ff000042	76	f1000002	77	ff000082
78	410000ff	79	280000ff	7a	f200008f	7b	f100000f
7c	8f000040	7d	82000041	7e	f100008f	7f	400000ff
80	8f0000f2	81	4f0000f1	82	f000008f	83	40000080
84	820000ff						

Table 10. Collision characteristic for rounds 4 to 16. $2^{144.1}$ valid keys, probability $2^{-71.1}$
Can be used for near-collisions for rounds 0 to 20.

	Constraints	Prob.	Example
m_0	-001!==$^{5}_{0}{}^{5}_{1}$-$^{5}_{2}$--------------------00010010$^{5}_{3}{}^{5}_{4}$-$^{5}_{2}$----$^{5}_{5}$----$^{5}_{6}$-----------		97c787b0252f1bef
m_1	100110111$^{5}_{7}$-----0011100111011110110011010!$^{5}_{8}$-----$^{5}_{9}$----------------		9ba673bd9a918263
m_2	-101!=!---$^{5}_{a}$--------------------1$^{5}_{b}$---------$^{5}_{a}$---------$^{5}_{c}$-----------		59f24b2909ae5223
m_3	-001011000-----$^{5}_{d}$----$^{5}_{c}$-----1$^{5}_{b}$---1$^{5}_{e}$-----$^{5}_{f}$1------------$^{6}_{0}$---0$^{6}_{1}{}^{5}_{2}{}^{5}_{1}$---0		963151773356523a
m_4	-1011000!==$^{5}_{a}$---1111-0101100010010$^{5}_{e}$----0$^{5}_{7}$-----------$^{6}_{2}{}^{5}_{c}$---1!$^{5}_{0}{}^{6}_{3}$---1		d873f5892cba83b7
$e_{4,0}$	x---	0.0	55854848e8d2b7fa
$e_{4,1}$	--	0.0	b45620969e3043f9
$e_{4,2}$	--	0.0	a0fe049603be00b1
$e_{4,3}$	--	0.0	4bc235e51a57e884
$e_{5,0}$	x---	0.0	09db68df8702fbf3
$e_{5,1}$	--	0.0	d86feb73899182ff
$e_{5,2}$	--	0.0	ecc03a7b1e15e935
$e_{5,3}$	x---	0.0	24e70858746a57b2
$e_{6,0}$	x---	0.0	e24b545310947ef2
$e_{6,1}$	x---1-----u-----------	0.0	6122c59537fb62a9
$e_{6,2}$	x---	0.0	11a742d3928040e7
$e_{6,3}$	x---0------------$^{5}_{1}$--------	0.0	82f4a248ea489c96
$e_{7,0}$	--u-----0-----	2.0	436e19e8488fe19b
$e_{7,1}$	----------1---------0------------1011$^{5}_{2}{}^{6}_{1}$---n--------------------$^{6}_{d}$	0.0	06a1773b59686055
$e_{7,2}$	------$^{6}_{2}$-------------0---=-$^{6}_{b}$----$^{6}_{0}$-$^{6}_{c}$----$^{6}_{0}$------------------------1	0.0	949be51c7cc8dd7d
$e_{7,3}$	------n-------------0---=-1----0-0011100!-----------n-----u$^{6}_{e}$---1	0.0	e6ea92fe1c500c11
$v_{8,0}$	-1001$^{6}_{8}{}^{6}_{8}$---0--------------------$^{5}_{3}$----------n---------u-----------	1.0	4a0f9123a1f841f0
$v_{8,1}$	-11001n111-----$^{6}_{e}$----u-----u$^{6}_{4}$---1$^{6}_{1}$-----n$^{5}_{2}$------------n---!$^{7}_{0}$n$^{5}_{1}$---0	1.3	67d6740b7ff27b70
$v_{8,2}$	-11110n1!==$^{5}_{8}$---0!==-100000011010$^{5}_{3}{}^{5}_{4}$----0$^{5}_{5}$-----------$^{6}_{f}$n---10u$^{5}_{b}$---0	1.0	7b86781a9918e98e
$v_{8,3}$	-0010011--n--------------------010100111$^{6}_{1}$-u---------u-----------	0.4	1367f176a75936cb
$e_{8,0}$	101001000000000111011100010011001010101110n001101001u10000010011		a401dc4caba69413
$e_{8,1}$	111111n0000001111100u10110u00010101100n1010010001100n10110n01010		fe07c582b348cdaa
$e_{8,2}$	010100n111111010011011011010001111000101110100110110n10101u00101		53fa6da3c5d36d45
$e_{8,3}$	1010101100n0111101111001001001101100110010u010000101u01010111100		ab2f7926cc8852bc
$e_{9,0}$	101000n0000010011010u00111u01111010111n011n01111011000110n11101		a209a1cf5eef61bd
$e_{9,1}$	100001n000u011110010101001u000101100u00011n001110110101100101110		860f2a42c0e76b2e
$e_{9,2}$	111111n100n0100111100110110010101001001001u110111100000000u00001		ff29e6ca925bc001
$e_{9,3}$	010100110110n0010000n101u0u111010110110110u0u10111011110u0n11100		53690d1d6d85de3c
$e_{10,0}$	0010nu0000u11000110un1000u01001000unn1n11n0101101100110011n01011		2818cc121fd6cceb
$e_{10,1}$	0u1010n0uunnu1000010uuun1nnnnnu1n1uuununuu11n0101u0n0101n000000n		2a3421fdc53a9581
$e_{10,2}$	010100n0nuunu010111nuunn1nn0unnnnnnnnnnnnn10u00110011110u0111101		5292f3e7ffe19e3d
$e_{10,3}$	1001nu1011n1000010nun1u01n10000010nnn0111n111010n10000n010u11101		9af0ace0bbfac29d
$e_{11,0}$	un0nuunu0n00n10unnn0nn10uu0un1n11n100n0nu00nu001unnu00nuu1n01nuu		524cee0fe511626c
$e_{11,1}$	0u01u1n101uuu0u1001111ununu1uu10010un01101110u0unuuuuuuuu11uu001		17413d524b708061
$e_{11,2}$	1110n1u1n0uuu01110n000uunnu0100010nnn01111unnnuuu1nuuuuun1u11010		ed83a0c8bbdc60da
$e_{11,3}$	1un01100101u1nn00111u011unuu01u100100n0n10uuu10uu111nuuuu1n11100		acae73452584787c
$v_{12,0}$	u11010un1000nnn00un0n0n1unnu001uuu1100uu1uuuuuu1nnnu00nu1100n10n		698e2b623081e2cd
$v_{12,1}$	u010nun0nunn1101000nnun1nu0nnuu00n110nuu10101110101n00011n110010		2abd1b9874aeb1f2
$v_{12,2}$	10u1n01uuu11001uuuu1u1u0uu0u11u11n100u0nun10000u1101nuu101010110		9a32140de160d956
$v_{12,3}$	nu000u011u10nuu1100u0uun0unu10n1un0111nunu01000n1110111011101111		81a9812b5e91eeef
$e_{12,0}$	-nnnnnnn1$^{1}_{0}$UMNVM$^{1}_{1}$NNVM$^{4}_{0}$UMNN$^{4}_{1}{}^{1}_{2}{}^{4}_{2}N^{1}_{3}$x1$^{1}_{4}{}^{4}_{3}NNN^{1}_{3}{}^{1}_{5}{}^{4}_{4}$VUUUUMVMNVMV$^{1}_{6}UMV^{1}_{7}{}^{1}_{8}{}^{1}_{9}{}^{1}_{a}$x--n	2.0	ffbf7cd963d83507
$e_{12,1}$	-uuuuunnuunn---nuuu-u0unuun0uuu1nu----0n-----------n$^{7}_{1}$---nu1$^{5}_{b}$---1	1.6	03311121a16935a9
$e_{12,2}$	-unnu0$^{4}_{5}{}^{1}_{b}{}^{1}_{c}M^{1}_{d}$NNNVUUUMVMVMVUM$^{1}_{3}$x---uuu00un$^{1}_{e}{}^{4}_{6}{}^{1}_{f}x^{2}_{0}NNN^{1}_{3}{}^{4}_{7}$UUUU$^{2}_{1}{}^{2}_{2}$x---------	0.5	31f99bbe068ff545
$e_{12,3}$	uuunnn01un-----nnnnnunu0nnnunuuunn11nuu1uu-----n----------------	1.0	1d4ff4e8f9237155
$e_{13,0}$	xuuuuunun1----u0----$^{6}_{3}$-----n----nu0----u-unuuuuununn---nunu$^{7}_{2}$n-uuu	19.7	02f08dfb05416ab0
$e_{13,1}$	xunnnnunu0----n1----$^{7}_{3}$----------011000nunu-----uunuuunnnnunuununn	2.2	bd0f720cc52c8f4b
$e_{13,2}$	-nu011n10-----0----------------unnnnnnn11-----11011001n010011010	19.5	4f4990a6ffb3669a
$e_{13,3}$	-1u000u11-----1-----------0----001----n-01000111000---001001-010	1.0	41b25f9057470892
$e_{14,0}$	xnuuuuuuuuuuuuuuuuuuuuuuuuu----1n1----!-!-----!====---unnnn=-011	14.9	c0000007ca6df9fb
$e_{14,1}$	--1-----------------------u----100------------11-!$^{6}_{1}$---1100n!----	0.0	b502f54326734b37
$e_{14,2}$	-001000!------------------1----101----$^{5}_{3}$-$^{5}_{c}$-----10!=$^{5}_{a}$---n1001!-100	1.0	90fbf03756fa6f2c
$e_{15,3}$	-11000111101001011-01111010-----00---------------!=---u0101-----	0.0	e3d2ef4416eec8b0
$e_{15,0}$	x-1----------------------------0n1------------01-$^{5}_{5}$----0100$^{5}_{5}{}^{5}_{b}$----	1.0	7502f54af0e14532
$e_{15,1}$	-0100---------------------------u1---------------$^{6}_{8}$----1101------	0.0	a5cb64c941d1c367
$e_{15,2}$	-111010$^{5}_{7}$------------------------01---------------$^{5}_{7}$----1111$^{5}_{5}$-----	1.0	74cedf7b6de937dc
$e_{15,3}$	--------------------------------1----------------------=$^{6}_{1}$-------	0.0	abd6fe9ffc78881e
$v_{16,0}$	x-$^{5}_{e}$-----------------------------0---------------------00!-------	0.0	1ace5a1432b30899
$v_{16,1}$	--	0.0	dcdd5605c1b74165
$v_{16,2}$	--------------------------------$^{6}_{2}$----------------------$^{5}_{2}$--------	0.0	20a5de1b6a61bffa
$v_{16,3}$	--	0.0	5b1f997397786c50

Table 11. Free-start collision characteristic for rounds 0 to 20. $2^{56.7}$ valid keys, probability 2^{-43}

	Constraints	Prob.	Example
m_0	-1-1-1111001$^{65}_{a9}$-----111$^{6}_{a}$--1111!--$^{5}_{d}$-010-1001001-0$^{6}_{1}$----111!--01!==-		5f977cfdd64d2f57
m_1	x0-1010110000!-110010001100100110-000-1000101!=$^{7}_{1}$----011$^{5}_{3}$--110!=-		35839193022be6f4
m_2	x0-0-101-=-0-------010$^{6}_{a}$--0010-----000-=!$^{7}_{4}$---0-$^{5}_{1}$-----010---00$^{6}_{4}$---		05e168930700458f
m_3	-11011110!-0!==$^{5}_{8}$---1010101111-1110001-110$^{6}_{8}$--11111-0110110111$^{6}_{4}$--0		6f47d57f8b6f9b78
m_4	00011011!=-000110100101101011-00111100!==$^{7}_{1}$--0101000011010111$^{5}_{b}$--0		1b634b58f1f50d76
$e_{4,0}$	x---	0.0	dd5113862e4682f2
$e_{4,1}$	x---	0.0	976b12a915df1438
$e_{4,2}$	--	0.0	2682e7ab2b50853e
$e_{4,3}$	--	0.0	c21caeeb08ac00af
$e_{5,0}$	--	0.0	74bc262f4425972a
$e_{5,1}$	--	0.0	f9c797c9b7c5d83b
$e_{5,2}$	--	0.0	e89f969633fc85ed
$e_{5,3}$	---------------------------------------n------------------------	0.0	26979807350b410f
$e_{6,0}$	--	1.0	6e83bdf8fbeb6f65
$e_{6,1}$	--------------$^{75}_{4d}$---11-----u---$^{76}_{7e}$-----x------$^{5}_{5}$-----!-----------	0.0	76b75dcddd173495
$e_{6,2}$	---------------!$^{6}_{c}$-------$^{6}_{8}$--1----------n-----------------------	1.3	0f372e9d6907c6fc
$e_{6,3}$	---------==$^{5}_{9}$---10-------!---$^{7}_{3}$-----$^{7}_{8}$----$^{5}_{a}$--!=$^{6}_{4}$---=----------10$^{6}_{1}$--	0.7	188d43891e190294
$e_{7,0}$	---$^{5}_{4}$-----------------$^{66}_{41}$----0u-----1----$^{706}_{90e}$-----------$^{5}_{5}$-----=$^{5}_{6}$----	2.5	e53b1bc6d902a3fa
$e_{7,1}$	--==$^{7}_{a}$-$^{766}_{bd3}$---$^{7}_{8}$-------!!$^{6}_{c}$---!=---------01$^{0}_{1}$-$^{7}_{c}$-$^{6}_{2}$-------!!$^{6}_{5}$---1000---	0.0	c583f4662226eac0
$e_{7,2}$	0010$^{6}_{a}$-=$^{55}_{cd}$---01000-11001000!$^{5}_{2}$---!-000011n0$^{5}_{e}$--000$^{5}_{5}$---01001100$^{6}_{4}$---0	0.3	27c472268720c990
$e_{7,3}$	10110-00=---00!==-000110101n---1-n10111u01-10n1$^{6}_{1}$---111110010--00	0.0	b0e1c6b1ee76ff28
$v_{8,0}$	-0-0-01$^{6}_{8}$-$^{6}_{a}$-$^{5}_{a}$-------!==$^{5}_{0}$--01u1-----==$^{5}_{0}$-$^{655}_{83b}$---$^{5}_{e}$--------==$^{5}_{a}$---==$^{6}_{c}$---	2.6	aabf102cfb298eba
$v_{8,1}$	-u11011u11-10u0$^{6}_{1}$---00111111n0-001n000-01!$^{7}_{1}$--0n!==-001110000u$^{6}_{4}$--1	0.1	36d0c7f0c5760e09
$v_{8,2}$	1101100010-0011000111000110n!-000n110101!$^{5}_{2}$--0n11110010001011$^{6}_{4}$--0	0.0	d8a638d87597c8b8
$v_{8,3}$	-=-!-00u10011$^{5}_{b}$-----110!--11u11--$^{5}_{5}$-===-1010101-1$^{5}_{b}$----101$^{5}_{3}$--011!=-	0.4	8899faec3eaa7adc
$e_{8,0}$	n01100001010000000111100011 0u0000000000100010100111010100 01001001		b0a078c00229d449
$e_{8,1}$	1u10011u00011u0010011101011n00000n01000011100n011010100 1100u0001		a6189d7050e5a981
$e_{8,2}$	1111010000001001100001000 01n00010n10011110001n001101011000101110		f4098431678cd62e
$e_{8,3}$	1110100u001100010111011111 11u1010000101001111011110101010001 10101		e83177ea14f7aa35
$e_{9,0}$	0n01011u10111u0100010110001100000n01001100001n1101111101110u1010		56b91630530f7dca
$e_{9,1}$	1010101n110n00001110100011101 1001n01011010110u010110100u01010010		abd0e8ecd6b16852
$e_{9,2}$	1101110u0011101011111100000110110n11110010000n001000000001100011		dc3afc1b7c848063
$e_{9,3}$	0n11000n11100101000u00100000100n0u11100101101111000n01000n0u110u		71e50209396f144c
$e_{10,0}$	uu000010nuuu1u0111111111110001110nu0101001110000001110011u00un1100		0289ff1d29c0e61c
$e_{10,1}$	nnu101n0n1nnnn0uuunn010000n00uuu101001n1n00uuu01010u10101000unnn		d6fc3420a7814a87
$e_{10,2}$	0n00111000unnnnnnnnn111000100nuu1011010 1111nuu11100n010010n0nnnn		4e1ffe24b5f394af
$e_{10,3}$	nu0u0111nunu0u11010u001u0001001nn010011100001n011000110n0uuu1010		87a34213a70d8d0a
$e_{11,0}$	11u1nuun10uuu11uuunn0011001nn10nn10nuuunu10uuu100unn000u101u0unn		d986333dd14230a3
$e_{11,1}$	n1un1u00u1uu1n1u0n0u101unu1nnn1n1nnn1n1uunu0u01n00nn0010uu0111nu		d84e4abffe43321e
$e_{11,2}$	n1un0101n1uu0u1101u0000u0011nu0uu101110nuuuuu00100nu000n1u1110un		d5c340385d0121b9
$e_{11,3}$	11uunu01n1unu101n1nnuun1100n100un00nu0101un0nu0n01u0nnnunun1nuun		c9d5f39892a94eb9
$v_{12,0}$	nu1n00u11nu1unu0u1n111un1nnnn1un110u1n1n1000un0n01n00un01n0u0001		b1d47dfdcf8562c1
$v_{12,1}$	nn00n010nu1nu0001nn00100n110n00nn10nunun00010100uu00001101 1u000u		cab0e4e9d5140360
$v_{12,2}$	nu011n1n1uunnu0nu01nuu1n1nu1uuu01110n111101u1unu011100u001n100n0		9f9933d0efaa7072
$v_{12,3}$	1u111u00uu011n01u01un010uu0u0u1uu0u0u11111n0u0nn00nuuuu100u11un0		b81d2a0207e3211a
$e_{12,0}$	xunu00u!==-nn$^{1}_{4}$$^{4}_{8}$$^{2}_{3}V^{2}_{2}$x10unn0nnnn-un01un$^{2}_{4}$xnu1$^{6}_{b}$--$^{2}_{5}$$^{4}_{9}$$^{2}_{6}$>=-1nnnn000nn$^{6}_{4}$--1	0.0	211c537d5af4fe39
$e_{12,1}$	nnnuu1nuuu-nunuuuunnuuuuunuuu-nunnuuu1nn0$^{5}_{e}$--1001uu0100011un$^{5}_{b}$--u	0.0	e6143042c70910d6
$e_{12,2}$	$^{2}_{2}$<-1-n1n0u1$^{414}_{5b7}$U$^{2}_{2}$x--nu$^{4}_{a}$$^{2}_{7}$x-nuu1$^{2}_{8}$x-$^{2}_{9}$xuuu-unnnnn$^{2}_{a}x^{2}_{b}$$^{4}_{b}NN^{1}_{3}$xnnn$^{22}_{b2}$xu0$^{12}_{e6}$>-	0.5	ff30b0cec5f79fc9
$e_{12,3}$	-n-unnunnunuuu-ununnnunnnuunununu-uun-nuuuuun1nn----nuu0--u100u-	0.3	eda0bb950a0f0811
$e_{13,0}$	xuuu0nnnuunnuuu$^{6}_{1}$---000111nuuuuuuu0100-unnnnnnnnuu-uunnnnuuuunnnn	7.1	073083c021fe0f0f
$e_{13,1}$	xnnn1uuunnuunnn$^{5}_{d}$---11100010000000-00101101000-nnnnunuuuunnnnuuuu	5.4	f8cf7c400b47d0f0
$e_{13,2}$	x110-1001101u00----unn0--n100u--n-unuuuuuuuu-1010100111111011010	11.2	ecd16c63d006a7da
$e_{13,3}$	-000-0101011n11----1000nnnnuuuunnuu---000011u0100-10011101101111	0.0	82be91e18c32276f
$e_{14,0}$	xuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu-1!=-0101000-!==-01111111111111	3.6	000000002d245dfff
$e_{14,1}$	-00----0100100011110011---000=--0$^{5}_{d}$=---100100=-1==-11-111-1100010	0.0	8691e6867e4e3762
$e_{14,2}$	x11$^{6}_{1}$-111100unnnnnnnnnnn--1000!--u$^{5}_{6}$!---000011!-0!=-00111101001001	4.4	6f8ffe455c38cf49
$e_{15,3}$	-11101000011110000-111n0001100111$^{6}_{4}$----1001010-!==-0-11010010111-	0.3	f43c3e33f255dd2e
$e_{15,0}$	-00----0100100011110011---000=--n$^{56}_{bb}$---11100$^{56}_{71}$-$^{6}_{1}$=$^{6}_{2}$-01-111-1100001	0.6	8691e686ab941761
$e_{15,1}$	-1!$^{6}_{1}$-111001$^{7}_{4}$----10-0100--==-$^{7}_{d}$---u$^{7}_{4}$----010--1$^{6}_{1}$-1=$^{5}_{6}$-!-00110111-==-	0.0	ef30a90e0533a378
$e_{15,2}$	x1!$^{6}_{2}$-0111100110000-1110--111!=--u$^{5}_{b}$----101000$^{5}_{d}$-1!$^{5}_{6}$-!-11000111!==-	0.7	63cc3c794e8eac77
$e_{15,3}$	-00----01----0111010000---011---n-$^{5}_{9}$---1$^{6}_{1}$!---$^{5}_{6}$-$^{6}_{2}$!$^{6}_{2}$-!--==--0-111-0	0.0	0c8ba11cb26d2fbc
$v_{16,0}$	-$^{5}_{0}$-----11!$^{5}_{c}$-----!=-!==----$^{5}_{c}$-----------0!$^{5}_{a}$-----$^{65}_{e8}$--$^{5}_{0}$--01!-1!$^{5}_{6}$-00-	0.0	75c28f94b0c7bad9
$v_{16,1}$	-------$^{7}_{3}$------$^{5}_{6}$!-----------$^{6}_{a}$-----------------$^{77}_{3e}$--$^{6}_{4}$--!-----01---	0.0	c23af22a0c707d2f
$v_{16,2}$	x$^{56}_{03}$----0$^{6}_{8}$----1111!-111$^{6}_{4}$---0$^{55}_{34}$---------$^{655}_{a1c}$-----$^{65}_{38}$--$^{6}_{c}$--!$^{6}_{0}$--0-10$^{5}_{3}$--	0.0	7057dd9600fbdc33
$v_{16,3}$	x------01-----------=!----$^{5}_{6}$-----------1=$^{7}_{f}$------------01---!-----	0.0	70f12cec5ff713d7

Table 12. Free-start, free-tweak, (Near-)Collision characteristic for rounds 0 to 32. $2^{44.9}$ valid keys, probability $2^{-25.1}$

	Constraints	Prob.	Example
k_0	-11011011011001000101100111000110000!===[6/2]-010000000000010001!-10		edb22ce30810011a
k_1	00001000000010100001011101001000001000100[5/d]-11000000000010101001010		08142e9044b0054a
k_2	-0![6/0]-[5/2]-[5/4]--[5/2]------0111101[6/1]-0!-111!===[5/6]-[6/8]---0[8/0]--11--011111![6/2]-10-11		1e06bd5779535f97
k_3	>====[6/0]-[5/c]--[6/3]----1111001111000010111100-0111000101--1110000[6/1]-1!-10		82a5e785e5c5b836
k_4	x-![5/8]------[5/e]------1000011[6/1]-1=-011!==-------0[8/0]--10--111001[7/4]--1--11		62d4437b79caf9d3
t_0	u000000000000000000000000000000000000000000000000000000000000000		0000000000000000
t_1	0000000000000000000000000000000000000000000000000000000000000000		0000000000000000
t_2	u000000000000000000000000000000000000000000000000000000000000000		0000000000000000
$e_{12,0}$	x---	0.0	87ad0104ef83d5a7
$e_{12,1}$	--	0.0	da552a21bd36c3ad
$e_{12,2}$	--	0.0	d9fd1666ce2e3e47
$e_{12,3}$	--	0.0	980fe1445c75f41a
$e_{13,0}$	x---	0.0	62022b26acba9954
$e_{13,1}$	--	0.0	cae71f9e1abbf0e9
$e_{13,2}$	--	0.0	720cf7ab2aa43261
$e_{13,3}$	x-----------------------![6/2]------------------------------[8/2]-------	0.0	217846a1f70e3300
$e_{14,0}$	x---[5/6]-----------	0.0	2ce94ac4c7768a3d
$e_{14,1}$	x-----------111![5/6]-------=-------------------0010[66/a3]--u[5/8]--01------	0.0	17ef213dc2826776
$e_{14,2}$	[0/2]------[5/1]--00![5/a]---!-11[6/7]1[65/a1]-----------[6/0]---[5/d]-1![5/2]----[6/b]--[5/c]-----------	0.0	93853e4d21b26561
$e_{14,3}$	n------0110100---0-110==[6/2]-------0---=---10010[6/5]---0--=-----------	0.0	d0d3387d00910c93
$e_{15,0}$	------------------101![5/6]----[5/8]--![5/8]------------------11u0[66/a3]--1[5/5]--11	1.0	44d86c0289f8f1b3
$e_{15,1}$	!=[5/d]-------0---=---01001[6/5]---0--01----------n------1000101---1-010	0.0	7b18528906b7453a
$e_{15,2}$	01100!=[6/c]--011000011101101100!-!=-0100010[6/1]-0000110111000111110100	0.0	645876ca224371f4
$e_{15,3}$	100111u[5/b]--000111110100001000!-10-1111110111100101111n00000n01110	0.0	9c87d0867ef2f82e
$v_{16,0}$	--=[6/c]------1------0111110[5/d]-[8/3]0-011!=[7/1]-------n[7/7]--00--11u11[55/33]--0--01	0.5	bff0be8b90b036ed
$v_{16,1}$	-11111n0000100101011n11101n11110001111u1[5/d]-1100011011n01010n0!-00	0.0	7e12bf7e3db1baa4
$v_{16,2}$	000000u01110000000100011101010000101000![6/2]-1101100110n01000n00010	0.0	00e04750a1366a22
$v_{16,3}$	-01[5/3]-[5/8]-[5/b]--u------1111011[5/d]-11-0011110[6/1]-[6/b]---n[5/b]--=[6/0]--10u100![5/6]-0!-00	0.9	b947fbb1eba86464
$e_{16,0}$	u010001011000101000000010000001110000101001n110110011u00011000000		22c502070a7b30c0
$e_{16,1}$	011010n1110001001110n10001n00001010001u11100000011011n01110n11110		6bc4ec6145c1bbbe
$e_{16,2}$	u00010u011110100011101011110000011100101111001100110n11101n01100		08f475e0e5e66f6c
$e_{16,3}$	1101011101u0111010111001000010010110010011n110111100u01111111111		d74eb90964fbc3ff
$e_{17,0}$	n00011n0100010011110n11001n01000010100u000n111001110110001n11110		8e89ee68503cec7e
$e_{17,1}$	u10110u101u010100100101000u100011000n00100u111011110010000100101		594a4a11891de425
$e_{17,2}$	n11000u001u0001100101110111010100100101011n000100011001101n01011		e0432eea4ae2336b
$e_{17,3}$	n01101011001u0011011n111u0u110000011111011u1u01101110110n0u01111		b591bf183ed3768f
$e_{18,0}$	1110011111u101000011n000011110011101n0u1010110101101000010n00011		e7d43879d95ad0a3
$e_{18,1}$	1u0010nu10n1n111110un101u1nn110u1u1110011100n0000u0u1111u001011n		8abfcd7cb9c80f17
$e_{18,2}$	100101u111u1u1001110n110u0u00010100010011011u10110101001n1111010		95d4ee0289b5a9fa
$e_{18,3}$	1010010110u0u00110n0n1u0110111010111n00001000010u10000u101n11101		a581acdd7842417d
$e_{19,0}$	un1100nu100nu100000001unn1nn011u1u0nuun100nuu01unnu1nnnnn0n110nu		729405f69322dfba
$e_{19,1}$	01111uu10uun01n1nn1u0nnnu10uununnu0u00uuun0nnu1nu0110nnuu0u0n111		7917e745805b360f
$e_{19,2}$	00111un1unun0110nuun1unun11uuuuuuuuuu0unnnnn0nnnn1101unnu1n10111		3b569ae001f7eb77
$e_{19,3}$	n10un1uun1uun00011100uunn11nu00n000nnuu0n11uu1111uuuu0uuu1u1nnuu		ccc8e1f118e7805c
$v_{20,0}$	nn1un0nn1u1un0n1nn1un1u1001nnnu000un00nnun1nnnnuuuun0n011100n00n		ebabed3c137e15c9
$v_{20,1}$	0u01unu011101nnn0n1nu1n1un00n0u01uu0uu1n1n0uuunnunun010111nnu0u0		14ef774883c355f0
$v_{20,2}$	u00unu0u0001n11101n1nnuu11unu00n000nn0n01n0nnnnn011010nn110nu01n		081f7cd11adf6bd3
$v_{20,3}$	110u1u0n0101011100u0u10nnuu01n000u01n0000uu1nuuu1101010u0010u11u		c957058c1818d426
$e_{20,0}$	xn011uunununnn1u0uunnun0000nn1n100un10nn[5/d]-uunnnuuuun0n1011[11/4b]<-1n	0.0	d95e1a1f1b8e16e3
$e_{20,1}$	uuunnnunuuuuuunnnunuu10nnn01n0uunnuun0uu[6/1]-nnuunnunun101100n1nun0	0.0	1d03a5d8c8735b3a
$e_{20,2}$	-01[4/c][2/2][4/4]N[4/d]UU[22/c2]x----0n1n0[1/4]<[6/2]-1u-uuu[12/ed][4/e][21/ef]x[4/a]U[2/2]xn[2/f]N[33/01]>--0010nn[3332/2346]>-nu	0.8	26263a289432cb6a
$e_{20,3}$	u10u1u-n--[7/3]----unnnunnu1uuunuuunnn11n-unnnu111nu--00110u![5/6]-0u-un	0.3	4bfced11fdde8c61
$e_{21,0}$	xnnn0nnuunnuuuu11unnnnnnnnn1unnnn1nuunuuuuuuuuununnnuunuuuun1-un	1.6	f661bff7e401721d
$e_{21,1}$	xuuu1uunnuunnnn0001111111111110000000u-0-------uunuuunnunnnnunuuu	5.6	899e3ff805e88de8
$e_{21,2}$	-nn1-01000nuu01--01001n![6/2]-1n-unu10u10-n---0[8/0]--01010101n111001011	13.3	7223273a921157cb
$e_{21,3}$	-nu001111111u00[5/d]-10110010100uuunnuu100u0001110110111010101010-10	0.0	47f15941903b7556
$e_{22,0}$	xnnnnnnnnnnnnnnnnnnnnnnnnnnunnnnn110!-0-------100000000000000-01	0.0	7fffffefe9ea0005
$e_{22,1}$	-![5/5]--11110000=[6/2]-10011001[6/1]-n1-11110010[66/17]---[77/ca]--011-10-00101!==-10	0.0	af80997f9519a95e
$e_{22,2}$	x!==-01000010![6/2]-nuuuuuu0[6/1]-11-10000100[55/da]---[68/01]--0011001nu100!==-01	1.1	ba14807c224ccd21
$e_{23,3}$	-10111001000010111-111u1100010000110!-1-------00000000u-0-------	0.0	5c85dd886614017f
$e_{23,0}$	-![5/5]--11110000=[6/2]-10011001[6/1]-10-111u111[5/d]---------111-10-00101!=[6/2]-11	0.0	2f80996f7f03a963
$e_{23,1}$	-111-100100000--1-------[5/8]-==-100n-0-----------11111011=-[6/2]-------	0.0	7483db04d7b7efd7
$e_{23,2}$	x001-110100110[5/d]-01-111!=[5/8]-==-100100![5/6]---------00110011!-[6/1]-------	0.0	169a5e048860cea0
$e_{23,3}$	-!---1-![5/8]--11=[5/d]-100=[5/d]-11[6/2]-00-0[5/d]-10-0[6/1]-----------[8/4]----11-11-=[6/1]-10	0.0	553e9b0a8157cfc6
$v_{24,0}$	-[55/7b]--1000000[66/11]--[5/e]---------1[6/1]-!==[6/0]-[6/2]------------111-0[5/3]-[55/2e]---------	0.0	a404747456bb993a
$v_{24,1}$	x[7/8]--------------------[7/8]-------[6/1]--------------------------------	0.0	ea8f36c95c86056c
$v_{24,2}$	x[6/b]---!-1[5/d]--1[56/7e]--==-=[5/d]-[65/a1]--[65/04]-=[5/6]-00-[5/d]-----------------[56/3e]---------	0.0	6bd8f90f09b89e66
$v_{24,3}$	x---------------[6/b]-----------------[5/6]-------------0-------------	0.0	73b39ba32238423e