Cryptanalysis of RadioGatún

Thomas Fuhr[1] and Thomas Peyrin[2]

[1] DCSSI Labs
thomas.fuhr@sgdn.gouv.fr

[2] Ingenico
thomas.peyrin@ingenico.com

Abstract. In this paper we study the security of the RadioGatún family of hash functions, and more precisely the collision resistance of this proposal. We show that it is possible to find differential paths with acceptable probability of success. Then, by using the freedom degrees available from the incoming message words, we provide a significant improvement over the best previously known cryptanalysis. As a proof of concept, we provide a colliding pair of messages for RadioGatún with 2-bit words. We finally argue that, under some light assumption, our technique is very likely to provide the first collision attack on RadioGatún.

Key words: hash functions, RadioGatún, cryptanalysis.

1 Introduction

A cryptographic hash function is a very important tool in cryptography, used in many applications such as digital signatures, authentication schemes or message integrity. Informally, a cryptographic hash function H is a function from $\{0,1\}^*$, the set of all finite length bit strings, to $\{0,1\}^n$ where n is the fixed size of the hash value. Moreover, a cryptographic hash function must satisfy the properties of preimage resistance, 2nd-preimage resistance and collision resistance [27]:

- **collision resistance**: finding a pair $x \neq x' \in \{0,1\}^*$ such that $H(x) = H(x')$ should require $2^{n/2}$ hash computations.
- **2nd preimage resistance**: for a given $x \in \{0,1\}^*$, finding a $x' \neq x$ such that $H(x) = H(x')$ should require 2^n hash computations.
- **preimage resistance**: for a given $y \in \{0,1\}^n$, finding a $x \in \{0,1\}^*$ such that $H(x) = y$ should require 2^n hash computations.

Generally, hash functions are built upon a *compression function* and a *domain extension algorithm*. A compression function h, usually built from scratch, should have the same security requirements as a hash function but takes fixed length inputs instead. Wang *et al.* [32, 34, 35, 33] recently showed that most standardized compression functions (e.g. MD5 or SHA-1) are not collision resistant. Then, a domain extension method allows the hash function to handle arbitrary length inputs by defining an (often iterative) algorithm using the compression function as a black box. The pioneering work of Merkle and Damgård [15, 28] provided to designers an easy way in order to turn collision resistant compression functions onto collision resistant hash functions. Even if preserving collision resistance, it has been recently shown that this iterative process presents flaws [16, 19, 21, 20] and new algorithms [25, 7, 2, 1, 26] with better security properties have been proposed.

Most hash functions instantiating the Merkle-Damgård construction use a block-cipher based compression function. Some more recent hash proposals are based on construction principles which are closely related to stream ciphers. For example we can cite `Grindahl` [24] or `RadioGatún` [4]. The underlying idea of *stream-oriented* functions is to first absorb m-bit message blocks into a big internal state of size $c + m$ using a simple round function, and then squeeze the hash output words out. As the internal state is larger than the output of the hash function, the cryptanalytic techniques against the iterative constructions can not be transposed to the case of stream-oriented functions. In 2007, Bertoni *et al.* published a new hash construction mode, namely the *sponge functions* [6]. At Eurocrypt 2008, the same authors [5] published a proof of security for their construction : when assuming that the internal function F is a random permutation or a random transformation, then the sponge construction is indifferentiable from a random oracle up to $2^{c/2}$ operations.

However, even though the same authors designed `RadioGatún` and defined the sponge construction, `RadioGatún` does not completely fulfill the sponge definition. For evident performance reasons, the internal function F of `RadioGatún` is not a very strong permutation and this might lead to correlations between some input and output words. This threat is avoided by applying blank rounds (rounds without message incorporation) just after adding the last padded message word. More recently, some NIST SHA-3 candidates are using permutation-based modes as well, for example `SHABAL` [10], or sponge functions, for example `Keccak` [3].

Regarding the `Grindahl` family of hash functions, apart from potential slide attacks [18], it has been shown [29, 23] that it can not be considered as collision resistant. However, `RadioGatún` remains yet unharmed by the preliminary cryptanalysis [22]. The designers of `RadioGatún` claimed that for an instance manipulating w-bit words, one can output as much as $19 \times w$ bits and get a collision resistant hash function. That is, no collision attack should exist which requires less than $2^{9,5 \times w}$ hash computations. The designers also stated [4] that the best collision attack they could find (apart from generic birthday paradox ones) requires $2^{46 \times w}$ hash computations. A first cryptanalysis result by Bouillaguet and Fouque [8] using algebraic technique showed that one can find collisions for `RadioGatún` with $2^{24,5 \times w}$ hash computations. Finally, Khovratovich [22] described an attack using $2^{18 \times w}$ hash computations and memory, that can find collisions with the restriction that the IV must chosen by the attacker (semi-free-start collisions).

Our contributions. In this paper, we provide an improved cryptanalysis of `RadioGatún` regarding collision search. Namely, using an improved computer-aided backtracking search and symmetric differences, we provide a technique that can find a collision with $2^{11\times w}$ hash computations and negligible memory. As a proof of concept, we also present a colliding pair of messages for the case $w = 2$. Finally, we argue that this technique has a good chance to lead to the first collision attack on `RadioGatún` (the computation cost for setting up a complete collision attack is below the ideal bound claimed by the designers, but still unreachable for nowadays computers).

Outline. The paper is organized as follows. First, in Section 2, we describe the hash function proposal `RadioGatún`. Then, in Section 3, we introduce the concepts of *symmetric differences* and *control words*, that will be our two mains tools in order to cryptanalyze the scheme. In Section 4, we explain our differential path generation phase and in Section 5 we present our overall collision attack. Finally, we draw the conclusion in last section.

2 Description of `RadioGatún`

`RadioGatún` is a hash function using the design approach and correcting the problems of `Panama` [14], `StepRightUp` [13] or `Subterranean` [11, 13].

`RadioGatún` maintains an internal state of 58 words of w bits each, divided in two parts and simply initialized by imposing the zero value to all the words. The first part of the state, the *mill*, is composed of 19 words and the second part, the *belt*, can be represented by a matrix of 3 rows and 13 columns of words. We denote by M_i^k the i-th word of the mill state before application of the k-th iteration (with $0 \leq i \leq 18$) and $B_{i,j}^k$ represents the word located at column i and row j of the belt state before application of iteration k (with $0 \leq i \leq 12$ and $0 \leq j \leq 2$).

The message to hash is first padded and then divided into blocks of 3 words of w bits each that will update the internal state iteratively. We denote by m_i^k the i-th word of the message block m^k (with $0 \leq i \leq 2$). Namely, for iteration k, the message block m^k is firstly incorporated into the internal state and then a permutation P is applied on it. The incorporation process at iteration k is defined by :

$$\begin{array}{lll} B_{0,0}^k = B_{0,0}^k \oplus m_0^k & B_{0,1}^k = B_{0,1}^k \oplus m_1^k & B_{0,2}^k = B_{0,2}^k \oplus m_2^k \\ M_{16}^k = M_{16}^k \oplus m_0^k & M_{17}^k = M_{17}^k \oplus m_1^k & M_{18}^k = M_{18}^k \oplus m_2^k \end{array}$$

where $\oplus$ denotes the bitwise *exclusive or* operation.

After having processed all the message blocks, the internal state is finally updated with N_{br} blank rounds (simply the application of the permutation P, without incorporating any message block). Eventually, the hash output value is generated by successively applying P and then outputting M_1^k and M_2^k as many time as required by the hash output size.

The permutation P can be divided into four parts. First, the *Belt* function is applied, then the *MillToBelt* function, the *Mill* function and eventually the *BeltToMill* function. This is depicted in Figures 1 and 2.

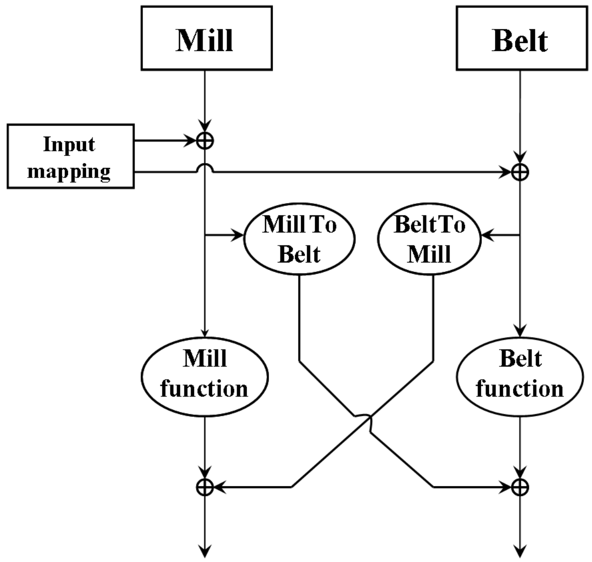

Fig. 1. The permutation P in `RadioGatún`.

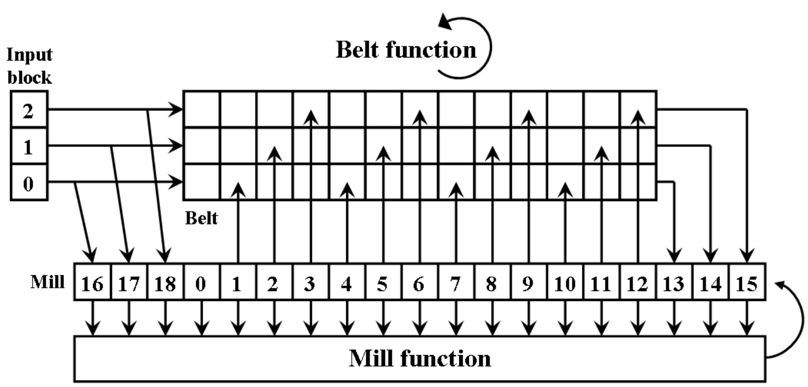

Fig. 2. The permutation P in `RadioGatún`.

The *Belt* function simply consists of a row-wise rotation of the belt part of the state. That is, for $0 \leq i \leq 12$ and $0 \leq j \leq 2$:

$$B'_{i,j} = B_{i+1 \bmod 13, j}.$$

The *MillToBelt* function allows the mill part of the state to influence the belt one. For $0 \leq i \leq 11$, we have :

$$B'_{i+1,i \bmod 3} = B_{i+1,i \bmod 3} \oplus M_{i+1}.$$

The *Mill* function is the most complex phase of the permutation P and it updates the mill part of the state (see Figure 3). In the following, all indexes should be taken modulo 19. First, a nonlinear transformation is applied on all the words. For $0 \leq i \leq 18$:

$$M'_i = M_i \oplus \overline{\overline{M_{i+1}} \wedge M_{i+2}}$$

where $\overline{X}$ denotes the bitwise negation of X and $\wedge$ represents the bitwise *and* operation. Then, a diffusion phase inside the words is used. For $0 \leq i \leq 18$:

$$M'_i = M_{7\times i} \ggg (i \times (i+1)/2)$$

where $X \ggg (y)$ denotes the rotation of X on the right over y positions. Then, a diffusion phase among all the words is applied. For $0 \leq i \leq 18$:

$$M'_i = M_i \oplus M_{i+1} \oplus M_{i+4}.$$

Finally, an asymmetry is created by simply setting $M_0 = M_0 \oplus 1$.

The *BeltToMill* function allows the belt part of the state to influence the mill one. For $0 \leq i \leq 2$, we have :

$$M'_{i+13} = M_{i+13} \oplus B_{12,i}.$$

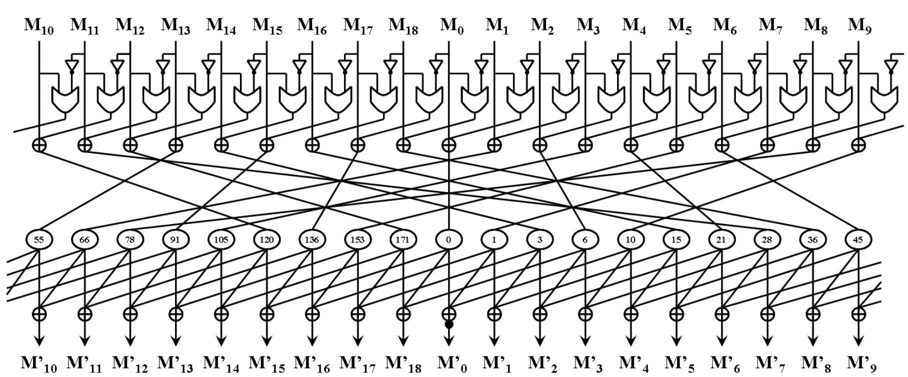

Fig. 3. The *Mill* function in `RadioGatún`.

The `RadioGatún` security claims. Although `RadioGatún` has some common features with the sponge functions, the security proof of the sponge construction does not apply for this proposal. In their original paper [4], the authors claim that `RadioGatún` can output as much as 19 words and remain a secure hash function. Thus, it should not be possible for an attacker to find a collision attack running in less than $2^{9,5\times w}$ hash computations.

3 Symmetric differences and control words

3.1 Symmetric differences

The first cryptanalysis tool we will use are symmetric differences. This technique has first been described in [30]. It was mentioned as a potential threat for `RadioGatún` in [4]. More precisely, a symmetric difference is an intra-word *exclusive or* difference that is part of a stable subspace of all the possible differences on a w-bit word. For example, in the following we will use the two difference values 0^w and 1^w (where the exponentiation by x denotes the concatenation of x identical strings), namely either a zero difference or either a difference on every bit of the word.

Considering those symmetric differences will allow us to simplify the overall scheme. Regarding the intra-word rotations during the *Mill* function, a 0^w or a 1^w difference will obviously remain unmodified. Moreover, the result of an *exclusive or* operation between two symmetric differences will naturally be a symmetric difference itself :

$$0^w \oplus 0^w = 0^w \qquad 0^w \oplus 1^w = 1^w \qquad 1^w \oplus 0^w = 1^w \qquad 1^w \oplus 1^w = 0^w$$

The nonlinear part of the *Mill* function is more tricky. We can write :

$$\overline{\overline{a} \wedge b} = a \vee \overline{b}.$$

The output of this transformation will remain a symmetric difference with a certain probability of success, given in Table 1.

Due to the use of symmetric differences, the scheme to analyze can now be simplified : we can concentrate our efforts on a $w = 1$ version of `RadioGatún`, for which the intra-word rotations can be discarded. However, when building a differential path, for each differential transition during the nonlinear part of the *Mill* function, we will have to take the corresponding probability from Table 1 in account[3]. Note that this probability will be the only source of uncertainty in the differential paths we will consider (all the differential transitions through exclusive or operation always happen with probability equal to 1) and the product of all probabilities will be the core of the final complexity of the attack.

Also, one can check that the conditions on the *Mill* function input words are not necessarily independent. One may have to control differential transitions

[3] In a dual view, all the conditions derived from Table 1 must be fulfilled.

Δ_a	Δ_b	$\Delta_{a \vee \bar{b}}$	Probability	Condition
0^w	0^w	0^w	1	
0^w	1^w	0^w	2^{-w}	$a = 1^w$
0^w	1^w	1^w	2^{-w}	$a = 0^w$
1^w	0^w	0^w	2^{-w}	$b = 0^w$
1^w	0^w	1^w	2^{-w}	$b = 1^w$
1^w	1^w	0^w	2^{-w}	$a = b$
1^w	1^w	1^w	2^{-w}	$a \neq b$

Table 1. Differential transitions for symmetric differences during the nonlinear part of the *Mill* function of `RadioGatún`. Δ_a and Δ_b denote the difference applied on a and b respectively, and $\Delta_{a \vee \bar{b}}$ the difference expected on the output of $a \vee \bar{b}$. The last column gives the corresponding conditions on the values of a and b in order to validate the differential transition. By $a = b$ (respectively $a \neq b$) we mean that all the bits of a and b are equal (respectively different), i.e. $a \oplus b = 0^w$ (respectively $a \oplus b = 1^w$).

for nonlinear subfonctions located on adjacent positions (for example the first subfunction, involving M_0 and M_1, and the second, involving M_1 and M_2). This has two effects : potential incompatibility or condition compression (concerning M_1 in our example). In the first case, two conditions are located on the same input word and are contradicting (for example, one would have both $M_1 = 0^w$ and $M_1 = 1^w$). Thus, the differential path would be impossible to verify and, obviously, one has to avoid this scenario. For the second case, two conditions apply on the same input word but are not contradicting. Here, there is a chance that those conditions are redundant and we only have to account one time for a probability 2^{-w}. Finally, note that all those aspects have to be handled during the differential path establishment and not during the search for a valid pair of messages.

3.2 Control words

When trying to find a collision attack for a hash function, two major tools are used : the differential path and the freedom degrees. In the next section, we will describe how to find good differential paths using symmetric differences. If a given path has probability of success equal to P, the complexity of a naive attack would be $1/P$ operations : if one chooses randomly and non-adaptively $1/P$ random message input pairs that are coherent with the differential constraints, there is a rather good chance that a one of them will follow the differential path entirely. However, for the same differential path, the complexity of the attack can be significantly decreased if the attacker chooses its inputs in a clever and adaptive manner.

In the case of `RadioGatún`, 3 w-bit message words are incorporated into the internal state at each round. Those words will naturally diffuse into the whole internal state, but not immediately. Thus, it is interesting to study how this diffusion behaves. Since the events we want to control through the differential path are the transitions of the nonlinear part of the *Mill* function (which depend on the input words of the *Mill* function), we will only study the diffusion regarding the input words of the *Mill* function.

Table 2 gives the dependencies between the message words incorporated at an iteration k, and the 19 input words of the *Mill* function at iteration k, $k+1$ and $k+2$. One can argue that a modification of a message block does not necessarily impacts the input word marked by a tick in Table 2 because the nonlinear function can sometimes "absorb" the diffusion of the modification. However, we emphasize that even if we depict here a behavior on average for the sake of clarity, all those details are taken in account thanks to our computer-aided use of the control words.

iteration	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
k																	✓		
k+1		✓	✓		✓	✓				✓			✓	✓				✓	
k+2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

iteration	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
k																		✓	
k+1		✓			✓	✓				✓			✓	✓		✓	✓		
k+2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

iteration	M_0	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}
k																			✓
k+1		✓			✓	✓		✓	✓				✓			✓	✓		
k+2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2. Dependencies between the message words incorporated at an iteration k, and the 19 input words of the *Mill* function of `RadioGatún` at iteration k, $k+1$ and $k+2$. The first table (respectively second and third) gives the dependencies regarding the message block m_0^k (respectively m_1^k and m_2^k). The columns represent the input words of the *Mill* function considered and a tick denotes that a dependency exists between the corresponding input word and message block.

4 An improved backtracking search

Our aim is to find internal collisions, i.e. collisions on the whole internal state before application of the blank rounds.

In order to build a good differential path using symmetric differences, we will use a computer-aided meet-in-the-middle approach, similar to the technique in [29]. More precisely, we will build our differential path DP by connecting together separate paths DP_f and DP_b. We emphasize that, in this section, we only want to build the differential path and not to look for a colliding pair of messages. DP_f will be built in the forward direction starting from an internal state containing no difference (modeling the fact that we have no difference after the initialization of the hash function), while DP_b will be built in the backward direction of the hash computation starting from an internal state containing no difference (modeling the fact that we want a collision at the end of the path).

Starting from an internal state with no difference, for each round the algorithm will go through all the possible difference incorporations of the message input (remember that we always use symmetric differences, thus we only have $2^3 = 8$ different cases to study) and all the possible symmetric differences transitions during the *Mill* function according to Table 1 (the differential transitions through exclusive or operations are fully deterministic). The algorithm can be compared to a search tree in which the depth represents the number of rounds of `RadioGatún` considered and each node is a reachable differential internal state.

4.1 Entropy

An exhaustive search in this tree would obviously imply making useless computations (some parts of the tree provide too costly differential paths anyway). To avoid this, we always compute an estimation of the cost of finding a message pair fulfilling the differential paths during the building phase of the tree, from an initial state to the current leaf in the forward direction, and from the current leaf to colliding states in the backward direction.

A first idea would be to compute the current cost of DP_f and DP_b during the *meet-in-the-middle* phase. But, as mentioned in Section 3, some words of the mill only depend on the inserted message block after 1 or 2 rounds. Therefore, some conditions on the mill value have to be checked 2 rounds earlier, and some degrees of freedom may have to be used to fulfill conditions two rounds later. As DP_f and DP_b are computed round per round, it is difficult to compute their complexity during the search phase, while having an efficient early-abort algorithm.

Therefore, we use an *ad hoc* parameter, denoted H^k and defined as follows. If c^k is the total number of conditions on the mill input words at round k (from Table 1), we have for a path of length n :

$$\begin{cases} H^k = \max(H^{k+1} + c^k - 3, 0), \ \forall k < n \\ H^n = 0 \end{cases}$$

The idea is to evaluate the number of message pairs required at step k in order to get $2^{w \times H^{k+1}}$ message pairs at step $k+1$ of the exhaustive search phase. To achieve this, one needs to fulfill $c^k \times w$ bit conditions on the mill input values, with $3 \times w$ degrees of freedom. Therefore, the values of H^k can be viewed as the relative entropies on the successive values of the internal state during the hash computation.

The final collision search complexity would be $2^{w \times H_{max}}$, where H_{max} is the maximum value of H^i along the path, if the adversary could choose 3 words of his choice at each step, and if each output word of the *Mill* function depended on all the input words. In the case of `RadioGatún`, the computation cost is more complex to evaluate, and this is described in Section 5. The maximum entropy can be linked to the *backtracking cost* C_b, as defined in [4]. One has the relation $C_b = H_{max}+3$. The difference between these two notions is that the backtracking cost takes in account the randomization of the input message pairs, which has a cost 2^{3w}.

4.2 Differential path search algorithm

The path search algorithm works as follows. Keep in mind that the values of the entropy along the path are relative values - any constant value can therefore be added or subtracted to all the H_i. A zero entropy at step i means that one expects $2^0 = 1$ message pair to follow the path until step i. To evaluate a path, we then set the minimal value of the entropy along the path to zero, the cost being the maximal value of the entropy. Therefore we first compute candidates for DP_f with a modified breadth-first search algorithm, eliminating those for which the maximum entropy exceeds the minimum entropy by more than $8 \times w$ (because we want to remain much lower than the $9,5 \times w$ bound from the birthday paradox). The algorithm differs from a traditional breadth-first search as **we do not store all the nodes, but only those with an acceptable entropy** : to increase the probability of linking it to DP_b, one only stores the nodes whose entropy is at least $(H_{max} - 4) \times w$. We also store the state value of the previous node with entropy at least $(H_{max} - 4) \times w$, to enable an efficient backtracking process once the path is found.

We then compute DP_b, using a depth-first search among the backwards transitions of the *Mill* function, starting from colliding states. We set the initial entropy to $H^n = 0$, and we do not search the states for which $H > 8$ (same reason as for DP_f : we want to remain much lower than the bound from the birthday paradox). For each node having an entropy at most 4, we try to link it with a candidate for DP_f.

4.3 Complexity of the path search phase

The total amount of possible values for a symmetric differential on the whole state is $2^{13 \times 3+19} = 2^{58}$. We use the fact that for `RadioGatún`, the insertion of $M \oplus M'$ can be seen as the successive insertions of M and M' without applying the round function. Therefore, we can consider setting the words $16, 17, 18$ of the stored mill to 0 by a message insertion before storing it in the forward phase, and doing the same in the backward phase before comparing it to forward values. Therefore, the space on which the meet-in-the-middle algorithm has to find a collision has approximately 2^{55} elements. We chose to store 2^{27} values of DP_f, and thus we have to compare approximately 2^{28} values for DP_b.

5 The collision attack

In this section, we depict the final collision attack, and compute its complexity. Once a differential path is settled, the derived collision attack is classic : we will use the control words to increase as much as possible the probability of success of the differential path.

5.1 Description

The input for this attack is a differential path, with a set of sufficient conditions on the values of the mill to ensure that a pair of messages follow the path. The adversary searches the colliding pairs in a tree, in which the nodes are messages following a prefix of the differential path. The leaves are messages following the whole differential path. Thanks to an early-abort approach, the adversary eliminates candidates as soon as they differ from the differential path. Nodes are associated with message pairs, or equivalently by the first message of a message pair – the second message is specified by the differential trail. Therefore, they will be denoted by the message they stand for. The sons of node M are then messages $M||b$, where b is a given message block, and the hash computation of $M||b$ fulfills all the conditions.

The adversary then uses a depth-first approach to find at least one node at depth n, where n is the length of the differential path. It is based on the trail backtracking technique, described in [4, 29]. To decrease the complexity of the algorithm, we check the conditions on the words of the mill as soon as they cannot be modified anymore by a message word inserted later.

From Table 2, we know that the k-th included message block impacts some words of the mill before the k-th iteration of the *Mill* function, some other words before the $k+1$-th iteration, and the rest of the mill words before the $k+2$-th iteration. We recall that m^k is the k-th inserted block, and we now set that M_j^k is the value of the j-th mill word after the k-th message insertion. Let also $\hat{M}_j^k$ be the value of the j-th word of the mill after the k-th nonlinear function computation.

After inserting m^k, one can then compute $M_{16}^k, M_{17}^k, M_{18}^k$, but also M_j^{k+1} for $j = \{1, 2, 4, 5, 7, 8, 9, 12, 13, 15\}$, and M_j^{k+2} for $j = \{0, 3, 6, 10, 11, 14\}$.

Some other conditions imply differences or non-differences between state words, $M_j^k \oplus M_{j+1}^k$. When writing these variables as functions of the input message words at step k and $k-1$, and of the state variables before message insertion $k-1$, one can notice the following : before the k-th message insertion, one can compute $M_j^k \oplus M_{j+1}^k$, for $j = \{15, 16, 17, 18\}$, $M_j^{k+2} \oplus M_{j+1}^{k+2}$ for $j = \{7, 10\}$, and $M_j^{k+1} \oplus M_{j+1}^{k+1}$ for all other possible values of j. Therefore, the adversary has to check conditions on three consecutive values of the mill on message insertion number k.

The most naive way to do it would consist in choosing m^k at random and hoping the conditions are verified, but one can use the following facts to decrease the number of messages to check :

- The conditions on words M_{16}^k, M_{17}^k and M_{18}^k as well as these on the values $M_{15}^k \oplus M_{16}^k$, $M_{16}^k \oplus M_{17}^k$, $M_{17}^k \oplus M_{18}^k$ and $M_{18}^k \oplus M_0^k$ at step k can be fulfilled by *xor*-ing the adequate message values at message insertion k.
- Using the linearity of all operations except the first one, the adversary can rewrite the values M_j^{k+1} as a linear combination of variables $\hat{M}_j^k$, with $j = \{0, \ldots, 18\}$. Words $\hat{M}_0^k$ to $\hat{M}_{13}^k$ do not depend on the last inserted message value, therefore can be computed before the message insertion.
- A system of equations in variables $\hat{M}_{14}^k, \ldots, \hat{M}_{18}^k$ remains. These equations are derived from conditions on round $k+1$, by reversing the linear part of the *Mill* function. More precisely, these equations define the possible values of these variables, or of the *xor* of two of these variables, one of them being rotated.

The computation of the sons of a node at depth k work as follows :

1. The adversary checks the consistency of the equations on $\hat{M}_{14}^k, \ldots, \hat{M}_{18}^k$. If these equations are not consistent, the adversary does not search the node. The probability that this system is consistent depends on dimension of the Kernel of the system and can be computed *a priori*.
2. The adversary exhausts the possible joint values of $\hat{M}_{14}^k, \ldots, \hat{M}_{18}^k$, M_{16}^k, M_{17}^k and M_{18}^k, considering all the conditions on these variables, which can be expressed bitwise (as the nonlinear part of the *Mill* function also works bitwise). The cost of this phase is then linear in w. The mean number of sons depends on the number of conditions.
3. For each remaining message block, the adversary checks all the other linear conditions on $\hat{M}_{14}^k, \ldots, \hat{M}_{18}^k$ and the conditions on the mill values 2 rounds later.

5.2 Computation of the cost

We will now explain how to compute the complexity of the collision search algorithm. The most expensive operation is the search of the sons of nodes. The total complexity of a given depth level k is the product of the number of nodes that have to be explored at depth k by the average cost of the search of these nodes. These parameters are exponential in w, therefore the total cost of the search can be approximated by the search of the most expensive nodes.

To compute the search cost, we assume that for all considered messages, the words of the resulting states for which no condition is imposed are independent and identically distributed. This is true at depth 0, provided the attacker initializes the search phase with a long random message prefix. The identical distribution of the variables can be checked recursively, their independence is an hypothesis for the attack to work. This assumption is well-known in the field of hash function cryptanalysis for computing the cost associated to a differential path (see e.g. [29]).

Let A^k be the number of nodes that have to be reached at depth k, and C^k the average cost of searching one of these nodes. Let P^k be the probability that a random son of a node at depth k follows the differential path, and Q^k the probability that a given node at depth k has at least one valid son. At depth k, the average number of explored nodes is related to the average number of explored nodes at depth $k+1$. When only a few nodes are needed, the average case is not sufficient, and one has to evaluate the cost of finding at least one valid node of depth $k+1$.

One has the following relations, for $k \in \{0, \ldots, n-1\}$:

$$\begin{cases} A^k = \max(\dfrac{A^{k+1}}{2^{3w} P^k}, \dfrac{1}{Q^k}) \\ A^n = 1 \end{cases}$$

Let K^k be the dimension of the Kernel of the linear system that has to be solved at depth k, and $\hat{P}^k$ the probability that the bitwise system of equations on the values of the mill before and after the nonlinear function has solutions. $\hat{P}^k$ can be computed exhaustively *a priori* for each value of k. A random node at depth k has at least one valid son if the two following conditions happen :

- The bitwise conditions at depth k and $k+1$ can be fulfilled,
- The remaining freedom degrees can be used to fulfill all the remaining conditions.

The first item takes in account the fact that some conditions might not depend on all the freedom degrees. Therefore, we have :

$$Q^k = \min(2^{-K^k} \hat{P}^k, 2^{3w - N^k_{COND}}),$$

where N^k_{COND} is the total number of conditions that has to be checked on the k-th message insertion. We also have $P^k = 2^{-N^k_{COND}}$, because each condition is supposed to be fulfilled with probability half in the average case, which is true provided the free words - *i.e.* without conditions fixing their values, or linking it to another word - are *i.i.d.* .

Searching a node works as follows : one solves the bitwise system of equations on the values of $M_{16}, M_{17}, M_{18}, \hat{M}_{14}, \ldots, \hat{M}_{18}$. The set of message blocks that fulfill this equations system then has to be searched exhaustively to fulfill the other conditions, and to generate nodes at depth $k+1$. C^k is then the cost of this exhaustive search, and can be computed as the average number of message blocks that fulfill the system of equations. Therefore, we have $C^k = 2^{3w} \hat{P}^k$.

For each node at depth k, the attacker can first check the consistency of the conditions on the mill words at steps k and $k+1$, which allows him not to search inconsistent nodes. Therefore, we have the following overall complexity :

$$T = O(\max_k (\frac{C^k A^k}{2^{K^k}}))$$

The best path we found has complexity about $2^{11 \times w}$, which is above the security claimed by the designers of `RadioGatún`[4], it is given in Appendix. As a proof of concept, we also provide in Appendix an example of a colliding pair of messages following our differential path for RadioGatún with $w = 2$. One can check that the observed complexity confirms the estimated one.

5.3 Breaking the birthday bound

Finding a final collision attack for RadioGatún with a computation complexity of 2^{11w} required us to own a computer with a big amount of RAM for a few hours of computation. Yet, the memory and computation cost of the differential path search phase is determined by the H_{max} chosen by the attacker. We conducted tests that tend to show that the search tree is big enough in order to find a collision attack with an overall complexity lower than the birthday bound claimed by the designers[4]. **The problem here is that the memory and computation cost of the differential path search will be too big for nowadays computers, but much lower than the birthday bound**. This explains why we are now incapable of providing a fully described collision attack for RadioGatún. However, we conjecture that applying our techniques with more memory and computation resources naturally leads to a collision attack for RadioGatún, breaking the ideal birthday bound.

Conclusion

In this paper, we presented an improved cryptanalysis of RadioGatún regarding collision search. Our attack can find collisions with a computation cost of about 2^{11w} and negligible memory, which is by far the best known attack on this proposal.

We also gave arguments that shows that RadioGatún might not be a collision resistant hash function. We conjecture that applying our differential path search technique with more constraints will lead to collision attacks on RadioGatún.

Acknowledgments

The authors would like to thank Guido Bertoni, Joan Daemen, Michal Peeters annd Gilles Van Assche for their comments on sponge functions.

References

1. Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-Property-Preserving Iterated Hashing: ROX. In *ASIACRYPT*, pages 130–146, 2007.

[4] Note also that the size of the search tree can be increased by considering more complex symmetric differences, such as 0^w, 1^w, $01^{w/2}$ and $10^{w/2}$.

2. Mihir Bellare and Thomas Ristenpart. Multi-Property-Preserving Hash Domain Extension and the EMD Transform. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2006.
3. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak specifications. Submission to NIST, 2008.
4. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Radiogatun, a belt-and-mill hash function. Presented at Second Cryptographic Hash Workshop, Santa Barbara (August 24-25, 2006). See http://radiogatun.noekeon.org/.
5. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the Indifferentiability of the Sponge Construction. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.
6. Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge Functions, presented at ECRYPT Hash Workshop 2007.
7. Eli Biham and Orr Dunkelman. A framework for iterative hash functions: Haifa. Second NIST Cryptographic Hash Workshop, 2006.
8. Charles Bouillaguet and Pierre-Alain Fouque. Analysis of radiogatun using algebraic techniques. In Liam Keliher Roberto Avanzi and Francesco Sica, editors, *SAC*, Lecture Notes in Computer Science. Springer, 2008.
9. Gilles Brassard, editor. *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*. Springer, 1990.
10. Emmanuel Bresson, Anne Canteaut, Benoît Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-François Misarsky, Maria Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-René Reinhard, Céline Thuillet, and Marion Videau. Shabal – a submission to advanced hash standard. Submission to NIST, 2008.
11. Luc J. M. Claesen, Joan Daemen, Mark Genoe, and G. Peeters. Subterranean: A 600 mbit/sec cryptographic vlsi chip. In *ICCD*, pages 610–613, 1993.
12. 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.
13. Joan Daemen. Cipher and hash function design strategies based on linear and differential cryptanalysis. PhD thesis, Katholieke Universiteit Leuven, 1995.
14. Joan Daemen and Craig S. K. Clapp. Fast hashing and stream encryption with panama. In Serge Vaudenay, editor, *FSE*, volume 1372 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 1998.
15. Ivan Damgård. A Design Principle for Hash Functions. In Brassard [9], pages 416–427.
16. R.D. Dean. Formal aspects of mobile code security. PhD thesis, Princeton University, 1999.
17. Thomas Fuhr and Thomas Peyrin. Cryptanalysis of `Radiogatún`, 2008.
18. Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide attacks on hash functions. In Joseph Pieprzyk, editor, *ASIACRYPT*, Lecture Notes in Computer Science. Springer, 2008.
19. Antoine Joux. Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions. In Matthew K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.

20. John Kelsey and Tadayoshi Kohno. Herding Hash Functions and the Nostradamus Attack. In Serge Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.
21. John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^{n} Work. In Cramer [12], pages 474–490.
22. Dmitry Khovratovich. Two attacks on radiogatun. In *INDOCRYPT*, Lecture Notes in Computer Science. Springer, 2008.
23. Dmitry Khovratovich. Cryptanalysis of hash functions with structures, presented at ECRYPT Hash Workshop 2008.
24. Lars R. Knudsen, Christian Rechberger, and Søren S. Thomsen. The Grindahl Hash Functions. In Alex Biryukov, editor, *FSE*, volume 4593 of *Lecture Notes in Computer Science*, pages 39–57. Springer, 2007.
25. Stefan Lucks. A Failure-Friendly Design Principle for Hash Functions. In Bimal K. Roy, editor, *ASIACRYPT*, volume 3788 of *Lecture Notes in Computer Science*, pages 474–494. Springer, 2005.
26. Ueli M. Maurer and Stefano Tessaro. Domain Extension of Public Random Functions: Beyond the Birthday Barrier. In Alfred Menezes, editor, *CRYPTO*, volume 4622 of *Lecture Notes in Computer Science*, pages 187–204. Springer, 2007.
27. A.J. Menezes, S.A. Vanstone, and P.C. Van Oorschot. Handbook of applied cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1996.
28. Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [9], pages 428–446.
29. Thomas Peyrin. Cryptanalysis of Grindahl. In *ASIACRYPT*, pages 551–567, 2007.
30. Vincent Rijmen, Bart Van Rompay, Bart Preneel, and Joos Vandewalle. Producing collisions for panama. In *FSE*, pages 37–51, 2001.
31. Victor Shoup, editor. *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*. Springer, 2005.
32. Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu. Cryptanalysis of the hash functions md4 and ripemd. In Cramer [12], pages 1–18.
33. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Shoup [31], pages 17–36.
34. Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Cramer [12], pages 19–35.
35. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on sha-0. In Shoup [31], pages 1–16.

Appendix A: collision for `RadioGatún`[2]

To generate a collision for `RadioGatún`[2], we use a 143-block differential path of cost 2^{11w}.

We give here a collision for the 2-bit version of `RadioGatún`. One can easily check that it follows the differential path given above. We write the message words using values between 0 and 3, which stand for the possible values of 2-bit words. The differential path, and some statistics about the collision search, can be found in the longer version of this paper [17].

To ensure that one has enough starting points, we used a 5-block common prefix.

The two colliding messages are :

$M_0 =$ 330 000 000 000 000 113 311 012 012 112 300 202
020 302 233 030 030 000 223 222 220 111 000 010
031 001 033 020 000 000 222 103 110 312 231 321
102 012 322 023 323 232 001 023 032 220 130 103
203 003 200 232 023 011 222 222 133 110 211 031
232 122 033 122 021 202 302 003 120 003 300 203
133 021 302 311 101 031 200 003 013 231 032 312
002 202 131 331 122 201 333 301 032 230 031 220
012 130 312 100 020 322 222 220 201 012 000 201
200 010 230 130 310 330 201 103 130 210 102 001
200 321 112 110 232 223 010 301 213 000 133 123
323 222 331 132 103 021 012 330 201 100 203 321
013 332 020 000

$M_1 =$ 330 000 000 000 000 113 311 312 022 122 030 202
020 332 103 303 303 003 113 222 120 121 030 020
031 001 303 313 000 330 222 103 110 312 202 321
201 011 022 010 313 202 031 023 032 120 130 103
200 303 233 232 013 321 111 211 203 123 121 031
132 112 300 122 011 202 032 003 210 300 300 100
203 311 302 012 101 002 100 303 013 231 302 322
032 131 102 001 211 232 300 301 302 230 301 120
011 103 022 200 013 022 212 113 131 311 003 131
200 010 230 200 020 000 231 103 100 113 132 031
233 321 112 220 232 220 010 332 223 300 100 123
013 122 302 131 200 311 012 300 202 230 133 321
013 331 023 003

The common value of the internal state is then :

$$\begin{aligned}
\texttt{belt}[0] &= (0,0,2,1,2,0,3,0,2,1,1,1,3),\\
\texttt{belt}[1] &= (3,1,0,2,3,2,2,3,1,2,3,0,2),\\
\texttt{belt}[2] &= (2,3,3,2,2,2,1,1,1,3,2,0,3),\\
\texttt{mill} &= (2,0,2,2,1,0,1,0,3,1,3,3,2,2,3,3,0,3,3)
\end{aligned}$$