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, sponge functions.

1 Introduction

A cryptographic hash functions 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 [26]:

- **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.* [30, 32, 33, 31] 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, 27] 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, 18, 20, 19] and new algorithms [24, 7, 2, 1, 25] with better security properties have been proposed.

One of the alternative candidate for building cryptographic hash functions are *sponge constructions*. This domain extension algorithm has recently been proposed by Bertoni *et al.* [6]. The underlying idea of sponge functions is to first absorb all the m-bit message blocks into a big internal state of size $c+m$, and then squeeze the hash output words out. Then, for each iteration, a round function F is applied to the internal state. At Eurocrypt 2008, Bertoni *et al.* [5] published a proof of security for their constructions : when assuming that the internal function F is ideally secure, then the sponge construction is indifferentiable from a random oracle up to $c/2$ operations. However, for evident performance reasons[3], in practice the internal function F is clearly not ideal and this threat is patched by applying blank rounds (rounds without message incorporation) just after adding the last padded message word. Several hash proposals follow the sponge framework or a closely related one, for example `Grindahl` [23] or `RadioGatún` [4]. More recently, some NIST SHA-3 candidates are using sponge-related framework as well, for example `Keccak` [3] or `SHABAL` [10].

Regarding the `Grindahl` family of hash functions, apart from potential slide attacks [17], it has been shown [28, 22] that it can not be considered as collision resistant. However, `RadioGatún` remains yet unarmed by the preliminary cryptanalysis [21]. 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 perfectly secure 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 [21] 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

[3] The internal state of sponge functions is usually quite big in order to avoid generic attacks applying to iterative constructions.

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]. At the same time, RadioGatún is an instance of the sponge functions framework [6], which directly provides a security proof of the domain extension algorithm when assumed that the internal main function is ideal.

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 belt 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 mill 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 m 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.

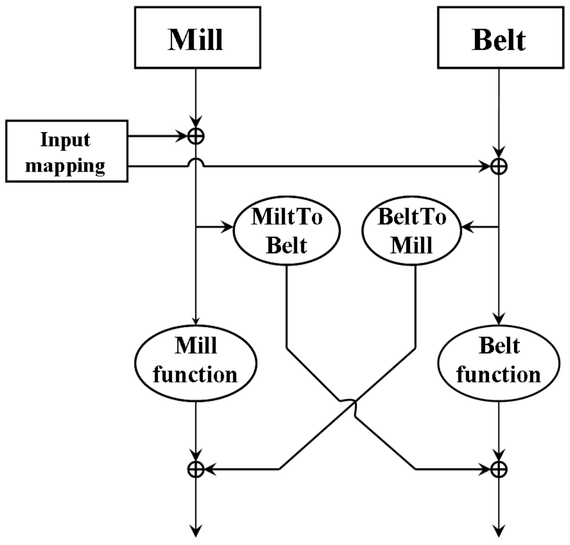


Fig. 1. The permutation P in `RadioGatún`.

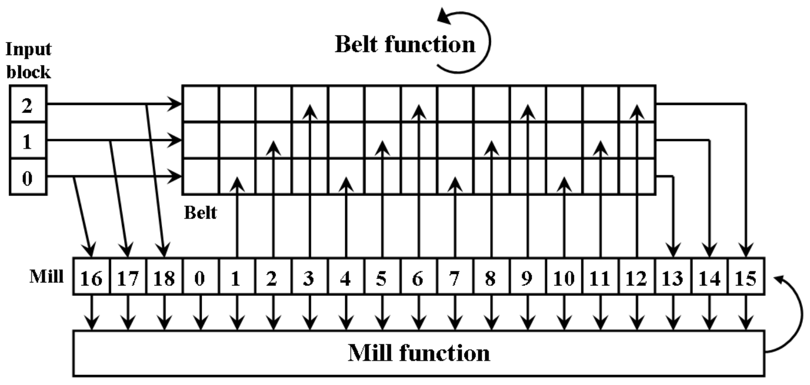


Fig. 2. The permutation P in `RadioGatún`.

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 the indexes should be taken modulo 19. First, a non linear 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 $0 \leq i \leq 18$:

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

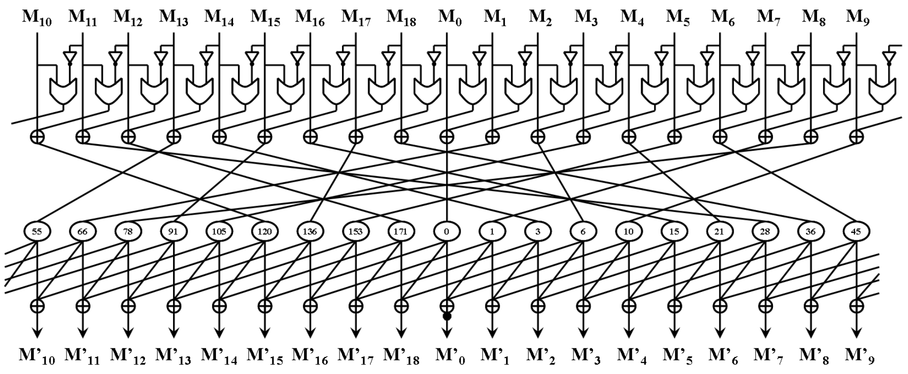
where $X \ggg (y)$ denotes the rotation of X on the right over y positions. Then, a diffusion phase among all the words is applied. For $0 \leq i \leq 18$:

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

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 :

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



M10 M11 M12 M13 M14 M15 M16 M17 M18 M0 M1 M2 M3 M4 M5 M6 M7 M8 M9
55 66 78 91 105 120 136 153 171 0 1 3 6 10 15 21 28 36 45
M'10 M'11 M'12 M'13 M'14 M'15 M'16 M'17 M'18 M'0 M'1 M'2 M'3 M'4 M'5 M'6 M'7 M'8 M'9

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

The `RadioGatún` security claims. 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, already mentioned 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 non linear 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 non linear part of the *Mill* function, we will have to take the corresponding probability from Table 1 in account[4]. 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 for non linear 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

[4] 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 non linear 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 $a \oplus b = 1^w$).

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 inputs that are coherent with the differential constraints, there is a rather good chance that a pair 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 non linear 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 non linear 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 [28]. 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 differences incorporation 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 leaf or sub-leaf 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 path 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.

4.2 Differential path search algorithm

The path search algorithm works as follows. 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 messages, 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, 28]. 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 $j = \{0, 3, 6, 10, 11, 14\}$. Similarly, 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 $j = \{7, 11\}$, 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 be to choose 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. 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 $\hat{M}_{14}^k, \ldots, \hat{M}_{18}^k$. 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 $\hat{M}^k_{14}, \ldots, \hat{M}^k_{18}$, M^k_{16}, M^k_{17} and M^k_{18}. This can be achieved bitwise, as the nonlinear part of the *Mill* function 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 $\hat{M}^k_{14}, \ldots, \hat{M}^k_{18}$ 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. [28]).

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. which is true provided the free words - *i.e.* without conditions fixing their values, or linking it to another word - are *i.i.d.* 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 account of 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 equation 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[5]. **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

[5] 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 $10^{w/2}$.

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.

References

1. Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-Property-Preserving Iterated Hashing: ROX. In *ASIACRYPT*, pages 130–146, 2007.
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, Michael 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, Michael 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, Michael 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 radiogatn 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, Benot Chevallier-Mames, Christophe Clavier, Thomas Fuhr, Aline Gouget, Thomas Icart, Jean-Franois Misarsky, Mara Naya-Plasencia, Pascal Paillier, Thomas Pornin, Jean-Ren Reinhard, Cline 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. Michael Gorski, Stefan Lucks, and Thomas Peyrin. Slide attacks on hash functions. In Joseph Pieprzyk, editor, *ASIACRYPT*, Lecture Notes in Computer Science. Springer, 2008.
18. 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.
19. 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.
20. 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.
21. Dmitry Khovratovich. Two attacks on radiogatn. In *INDOCRYPT*, Lecture Notes in Computer Science. Springer, 2008.
22. Dmitry Khovratovich. Cryptanalysis of hash functions with structures, presented at ECRYPT Hash Workshop 2008.
23. 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.
24. 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.
25. 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.
26. A.J. Menezes, S.A. Vanstone, and P.C. Van Oorschot. Handbook of applied cryptography. CRC Press, Inc., Boca Raton, FL, USA, 1996.
27. Ralph C. Merkle. One Way Hash Functions and DES. In Brassard [9], pages 428–446.
28. Thomas Peyrin. Cryptanalysis of Grindahl. In *ASIACRYPT*, pages 551–567, 2007.
29. 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.
30. 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.

31. Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding collisions in the full sha-1. In Shoup [29], pages 17–36.
32. Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Cramer [12], pages 19–35.
33. Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin. Efficient collision search attacks on sha-0. In Shoup [29], pages 1–16.

Appendix A: the differential path

We give here the differential path for the $2^{11\times w}$ collision attack for `RadioGatún`. For each step, it gives the input value of the internal state after the message insertion, and the output value of the state after the update function.

As the path is 143-block long, we use a hexadecimal notation to describe the differential values of internal states. Each mill value is written as $\Sigma_{i=0}^{18}\delta M_i 2^i$ where $\delta M_i = 1$ if word i of the mill contains a difference and $\delta M_i = 0$ otherwise. Similarly, we write the belt values as $\Sigma_{i=0}^{12}\delta B_{i,j} 2^i$. The belt values are given in the order $B_{\_,0}, B_{\_,1}, B_{\_,2}$.

We also give an estimation of the search cost at each step, as computed in section 5. In the column `Nodes`, we give the estimated value of $log_{2^w}(A^i)$, which is the logarithmic value of the estimated number of nodes the attacker has to search at depth i. In the column `Cost`, we give the estimated value of $log_{2^w}(\frac{C^i A^i}{2^{-K^i}})$, which is the logarithmic value of the estimated search cost at depth i.

	Input		**Output**			
`Step`	`Belt`	`Mill`	`Belt`	`Mill`	`Nodes`	`Cost`
0	0000 0000 0000	00000	0000 0000 0000	00000	1.000	4.000
1	0000 0000 0000	00000	0000 0000 0000	00000	4.000	6.000
2	0001 0000 0000	10000	0002 0000 0000	20034	4.000	6.000
3	0002 0001 0000	00034	0014 0026 0000	7a065	2.000	1.678
4	0014 0027 0000	5a065	0028 006a 0040	30000	0.000	3.000
5	0029 006b 0040	00000	0052 00d6 0080	00000	0.000	3.000
6	0052 00d6 0080	00000	00a4 01ac 0100	00000	1.000	4.000
7	00a4 01ac 0100	00000	0148 0358 0200	00000	4.000	7.000
8	0148 0359 0200	20000	0290 06b2 0400	19000	5.000	8.000
9	0291 06b3 0400	29000	0522 0d66 1800	71800	4.000	6.000
10	0523 0d67 1801	01800	0a46 12ce 0003	6c000	3.000	4.193
11	0a47 12cf 0002	1c000	148e 059f 0004	40034	2.000	4.000
12	148e 059f 0005	00034	090d 0b1a 000a	30000	1.000	4.000
13	090c 0b1b 000a	00000	1218 1636 0014	00000	4.000	7.000
14	1218 1636 0014	00000	0431 0c6d 0028	06000	7.000	7.193
15	0430 0c6d 0028	16000	0860 18da 0050	20034	5.000	7.000
16	0860 18db 0050	00034	10d0 1193 00a0	30464	3.000	2.000
17	10d0 1192 00a0	10464	05a1 0301 0100	20000	0.000	3.000
18	05a1 0300 0100	00000	0b42 0600 0200	00000	0.000	3.000

Continued on next page

Continued from next page

19	0b42 0600 0200	00000	1684 0c00 0400	00000	3.000	6.000
20	1684 0c00 0400	00000	0d09 1800 0800	02000	5.000	6.193
21	0d08 1801 0800	32000	1a10 1003 1000	7a440	4.000	6.000
22	1a11 1002 1001	0a440	1023 0005 0043	00020	0.000	3.000
23	1023 0005 0043	00020	0047 002a 0086	30000	0.000	3.000
24	0046 002b 0086	00000	008c 0056 010c	00000	0.000	3.000
25	008c 0056 010c	00000	0118 00ac 0218	00000	0.000	3.000
26	0118 00ac 0218	00000	0230 0158 0430	00000	2.000	5.000
27	0230 0158 0430	00000	0460 02b0 0860	00000	5.000	8.000
28	0460 02b0 0860	00000	08c0 0560 10c0	00000	7.000	10.000
29	08c0 0561 10c1	60000	1180 0ac2 0183	12390	5.000	7.000
30	1180 0ac2 0183	12390	0391 1484 0106	51400	1.000	2.000
31	0390 1484 0107	01400	0320 0909 120e	60000	0.000	3.000
32	0320 0909 120f	20000	0640 1212 041f	11000	2.000	5.000
33	0641 1212 041f	01000	0c82 0425 183e	60000	2.000	5.000
34	0c82 0424 183f	00000	1904 0848 107f	08000	4.000	7.000
35	1904 0849 107f	28000	1209 1092 00ff	30446	4.000	5.000
36	1209 1093 00ff	10446	0011 0123 01be	20000	0.000	3.000
37	0011 0122 01be	00000	0022 0244 037c	00000	1.000	4.000
38	0022 0244 037c	00000	0044 0488 06f8	00000	4.000	7.000
39	0044 0488 06f8	00000	0088 0910 0df0	00000	7.000	9.000
40	0089 0910 0df0	10000	0112 1220 1be0	20034	7.000	9.000
41	0112 1220 1be0	20034	0234 0465 17c1	21400	6.000	8.000
42	0234 0465 17c1	21400	0068 08ca 1f83	75000	4.000	6.000
43	0068 08ca 1f82	35000	00d0 1194 0f05	11000	2.000	5.000
44	00d1 1194 0f05	01000	01a2 0329 0e0a	60000	2.000	5.000
45	01a2 0328 0e0b	00000	0344 0650 1c16	00000	5.000	8.000
46	0344 0650 1c16	00000	0688 0ca0 182d	08000	7.000	10.000
47	0688 0ca1 182d	28000	0d10 1942 105b	11000	9.000	11.000
48	0d11 1943 105b	21000	1a22 1287 10b7	71000	8.000	11.000
49	1a23 1286 10b6	01000	1447 050d 116d	66800	7.000	7.193
50	1447 050c 116c	06800	088f 0218 02d9	2a006	4.000	6.000
51	088e 0219 02d9	1a006	111e 0436 05b2	60038	4.000	6.000
52	111e 0437 05b3	00038	022d 084e 0b6e	30000	2.000	5.000
53	022c 084f 0b6e	00000	0458 109e 16dc	00000	5.000	8.000
54	0458 109e 16dc	00000	08b0 013d 0db9	0c000	8.000	9.193
55	08b1 013d 0db9	1c000	1162 027a 1b72	20034	7.000	9.000
56	1162 027b 1b72	00034	02d5 04d2 16e5	70001	6.000	9.000
57	02d4 04d3 16e4	00001	05a8 09a6 0dc9	40001	8.000	11.000
58	05a8 09a6 0dc9	40001	0b50 134c 1b92	51001	8.000	10.000
59	0b50 134d 1b92	71001	16a0 069b 0725	00035	4.000	7.000
60	16a0 069b 0725	00035	0d51 0d12 0e4a	30000	2.000	5.000
61	0d50 0d13 0e4a	00000	1aa0 1a26 1c94	00000	4.000	7.000

Continued on next page

Continued from next page

62	1aa0 1a26 1c94	00000	1541 144d 1929	0e000	7.000	7.193
63	1540 144c 1929	3e000	0a81 0899 1253	70200	6.000	9.000
64	0a80 0899 1252	20200	1500 1132 06a5	01028	3.000	6.000
65	1500 1132 06a5	01028	0a01 0245 1d42	50000	1.000	4.000
66	0a00 0245 1d43	00000	1400 048a 1a87	08000	3.000	6.000
67	1401 048b 1a87	38000	0803 0916 150f	10200	5.000	8.000
68	0802 0917 150f	20200	1004 122e 081f	01028	2.000	5.000
69	1004 122e 081f	01028	0009 047d 0036	50000	0.000	3.000
70	0008 047d 0037	00000	0010 08fa 006e	00000	2.000	5.000
71	0010 08fa 006e	00000	0020 11f4 00dc	00000	4.000	7.000
72	0020 11f5 00dd	60000	0040 03eb 01ba	041a2	4.000	6.000
73	0041 03eb 01ba	141a2	0000 06f6 0374	10000	0.000	3.000
74	0001 06f6 0374	00000	0002 0dec 06e8	00000	0.000	3.000
75	0002 0dec 06e8	00000	0004 1bd8 0dd0	00000	3.000	6.000
76	0004 1bd8 0dd0	00000	0008 17b1 1ba0	04000	6.000	7.193
77	0009 17b0 1ba0	34000	0012 0f61 1741	15000	6.000	8.000
78	0012 0f60 1741	35000	0024 1ec0 0e83	11000	4.000	6.000
79	0024 1ec1 1e83	31000	0048 1d83 1d07	71000	2.000	5.000
80	0049 1d82 0d06	01000	0092 1b05 1a0c	60000	2.000	5.000
81	0092 1b04 0a0d	00000	0124 1609 141a	04000	5.000	6.193
82	0125 1608 141a	34000	024a 0c11 0835	71000	5.000	8.000
83	024b 0c10 0834	01000	0496 1820 1068	64000	5.000	6.193
84	0496 1821 0069	04000	092c 1043 00d2	24006	4.000	4.678
85	092c 1042 00d3	44006	1258 0085 01a6	00038	3.000	4.000
86	125a 0081 01a6	00038	04b5 0102 034c	30000	2.000	5.000
87	04a4 0123 0344	00000	0948 0246 0688	00000	5.000	8.000
88	0948 0246 0688	00000	1290 048c 0d10	00000	8.000	10.000
89	1291 048d 0d10	30000	0523 091a 1a20	3b034	6.000	8.000
90	0522 091a 1a20	2b034	0a44 1234 1441	41400	3.000	4.000
91	0a54 1210 0440	01400	14a8 0421 0880	60000	2.000	5.000
92	10a8 0420 1881	00000	0151 0840 1103	0a000	4.000	7.000
93	0150 0841 1103	3a000	02a0 1082 0207	11000	5.000	8.000
94	02a1 1082 0207	01000	0542 0105 040e	60000	4.000	7.000
95	0542 0104 140f	00000	0a84 0208 081f	08000	6.000	9.000
96	0a85 0208 081f	18000	150a 0410 103e	20034	7.000	9.000
97	150a 0411 103e	00034	0a15 0822 007d	70001	6.000	9.000
98	0a04 0807 007c	00001	1408 100e 00f8	51001	7.000	9.000
99	1409 100f 00f8	61001	0813 001f 01f0	30201	6.000	9.000
100	0812 001f 11f1	60201	1024 003e 03e3	4002a	2.000	5.000
101	1024 003e 01e2	0002a	0049 007c 03c4	30000	0.000	3.000
102	004a 005d 03cc	00000	0094 00ba 0798	00000	0.000	3.000
103	0094 00ba 0798	00000	0128 0174 0f30	00000	2.000	5.000
104	0128 0174 0f30	00000	0250 02e8 1e60	00000	5.000	8.000

Continued on next page

Continued from next page

105	0250 02e8 1e60	00000	04a0 05d0 1cc1	08000	7.000	10.000
106	04a1 05d1 1cc1	38000	0942 0ba2 1983	31006	7.000	10.000
107	0943 0ba3 1983	01006	1286 1746 1307	20444	3.000	3.000
108	1285 1743 0307	10444	050b 0e87 060e	20000	1.000	4.000
109	010b 0e82 064e	00000	0216 1d04 0c9c	00000	3.000	6.000
110	0216 1d04 0c9c	00000	042c 1a09 1938	04000	6.000	7.193
111	042c 1a08 1938	24000	0858 1411 1271	51034	4.000	7.000
112	0859 1411 1270	01034	10b2 0823 04e1	10020	1.000	3.000
113	10a2 0806 14e1	30020	0145 100c 09c3	21000	1.000	4.000
114	0145 102d 09c3	01000	028a 005b 1386	60000	0.000	3.000
115	028a 005a 0387	00000	0514 00b4 070e	00000	2.000	5.000
116	0514 00b4 070e	00000	0a28 0168 0e1c	00000	5.000	8.000
117	0a28 0168 0e1c	00000	1450 02d0 1c38	00000	8.000	10.000
118	1451 02d1 1c38	30000	08a3 05a2 1871	10200	9.000	11.000
119	08a3 05a2 1871	10200	1146 0b44 10e3	18028	5.000	8.000
120	1146 0b44 12e2	58028	028d 1688 05c5	01d40	4.000	5.000
121	028d 16a8 05cd	01d40	051a 0d51 0b9a	60000	0.000	3.000
122	011a 0450 1bdb	00000	0234 08a0 17b7	08000	0.000	3.000
123	0234 08a1 17b7	28000	0468 1142 0f6f	11000	1.000	3.000
124	0469 1142 0f6f	01000	08d2 0285 1ede	60000	1.000	4.000
125	08d2 0284 0edf	00000	11a4 0508 1dbe	00000	4.000	7.000
126	11a4 0508 1dbe	00000	0349 0a10 1b7d	0a000	6.000	9.000
127	0348 0a11 1b7d	3a000	0690 1422 16fb	11000	7.000	10.000
128	0691 1422 16fb	01000	0d22 0845 0df7	68000	5.000	8.000
129	0d22 0844 1df6	08000	1a44 1088 1bed	0b440	5.000	6.000
130	1a44 1088 1bec	4b440	1489 0111 17d9	50028	2.000	5.000
131	1088 0111 0798	00028	0111 0222 0f30	30000	1.000	4.000
132	0110 0203 0f38	00000	0220 0406 1e70	00000	4.000	7.000
133	0220 0406 1e70	00000	0440 080c 1ce1	08000	6.000	9.000
134	0440 080d 1ce1	28000	0880 101a 19c3	11032	5.000	7.000
135	0880 101a 19c2	51032	1100 0035 1385	7002a	0.000	2.000
136	1113 0014 0385	4002a	0227 0028 070a	30000	0.000	3.000
137	0224 0009 0702	00000	0448 0012 0e04	00000	1.000	4.000
138	0448 0012 0e04	00000	0890 0024 1c08	00000	4.000	7.000
139	0890 0024 1c08	00000	1120 0048 1811	08000	5.000	8.000
140	1120 0048 1810	48000	0241 0090 1021	105e2	5.000	6.000
141	0241 0090 1020	505e2	0482 0120 0041	40000	0.000	3.000
142	0000 0000 0000	00000	0000 0000 0000	00000	0.000	3.000

For large values of w, we can approximate the total search cost by the search cost of the most expensive states. Here we have 4 steps with search cost 2^{11w}, therefore we can approximate the collision search cost by :

$$T = 4 \times 2^{11w}.$$

Appendix B: collision for RadioGatún[2]

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. In the column *Nodes*, we give the number A^i of nodes that have been searched at depth i to find a collision. In the column $log_4(Nodes)$, we give the logarithm with base 4 of A_i, which can be compared with the theoric values given by the computation of the path, as $4 = 2^w$.

We can notice some differences between the theoric cost and the observed cost. Let us recall that the theoric number of nodes at step i linearly depends on the theoric number of nodes at step $i + 1$. As a consequence, if at some step i_0, more nodes have to be searched than expected, it will also affect the number of searched nodes at the previous steps. In our collision, we notice that these differences mainly arise at steps for which only a few nodes are needed, which can be explained as the theoric number of nodes is computed on average.

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

The common value of the internal state is then :

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

Step i	M_0	M_1	$Nodes$	$log_4(Nodes)$
-5	330	330	16	2.000
-4	000	000	16	2.000
-3	000	000	16	2.000
-2	000	000	16	2.000
-1	000	000	16	2.000
0	113	113	16	2.000
1	311	311	1014	4.993
2	012	312	974	4.964
3	012	022	57	2.916
4	112	122	1	0.000
5	300	030	1	0.000
6	202	202	4	1.000
7	020	020	227	3.913
8	302	332	915	4.919
9	233	103	245	3.968
				Continued on next page

Continued from next page

10	030	303	57	2.916
11	030	303	13	1.850
12	000	003	1	0.000
13	223	113	1	0.000
14	222	222	59	2.941
15	220	120	4	1.000
16	111	121	1	0.000
17	000	030	1	0.000
18	010	020	1	0.000
19	031	031	5	1.161
20	001	001	69	3.054
21	033	303	18	2.085
22	020	313	1	0.000
23	000	000	1	0.000
24	000	330	1	0.000
25	222	222	1	0.000
26	103	103	43	2.713
27	110	110	2738	5.709
28	312	312	43959	7.712
29	231	202	2793	5.724
30	321	321	16	2.000
31	102	201	2	0.500
32	012	011	22	2.230
33	322	022	22	2.230
34	023	010	358	4.242
35	323	313	313	4.145
36	232	202	1	0.000
37	001	031	11	1.730
38	023	023	657	4.680
39	032	032	42041	7.680
40	220	120	42301	7.684
41	130	130	10299	6.665
42	103	103	611	4.628
43	203	200	42	2.696
44	003	303	37	2.605
45	200	233	2353	5.600
46	232	232	37597	7.599
47	023	013	601697	9.599
48	011	321	150451	8.599
49	222	111	37874	7.604
50	222	211	588	4.600
51	133	203	589	4.601
52	110	123	29	2.429
53	211	121	1798	5.406
54	031	031	115031	8.406
55	232	132	28707	7.405
56	122	112	6956	6.382
57	033	300	110762	8.379

Continued on next page

Continued from next page

58	122	122	110814	8.379
59	021	011	389	4.302
60	202	202	21	2.196
61	302	032	323	4.168
62	003	003	20644	7.167
63	120	210	5110	6.160
64	003	300	81	3.170
65	300	300	6	1.292
66	203	100	73	3.095
67	133	203	1136	5.075
68	021	311	17	2.044
69	302	302	2	0.500
70	311	012	100	3.322
71	101	101	1583	5.314
72	031	002	1731	5.379
73	200	100	1	0.000
74	003	303	3	0.792
75	013	013	177	3.734
76	231	231	11317	6.733
77	032	302	11369	6.736
78	312	322	706	4.732
79	002	032	45	2.746
80	202	131	33	2.522
81	131	102	2083	5.512
82	331	001	2105	5.520
83	122	211	2088	5.514
84	201	232	505	4.490
85	333	300	123	3.471
86	301	301	33	2.522
87	032	302	2068	5.507
88	230	230	132333	8.507
89	031	301	8132	6.495
90	220	120	117	3.435
91	012	011	33	2.522
92	130	103	525	4.518
93	312	022	2068	5.507
94	100	200	578	4.587
95	020	013	9209	6.584
96	322	022	37022	7.588
97	222	212	9150	6.580
98	220	113	37059	7.589
99	201	131	9453	6.603
100	012	311	40	2.661
101	000	003	1	0.000
102	201	131	1	0.000
103	200	200	19	2.124
104	010	010	1155	5.087
105	230	230	18501	7.088

Continued on next page

Continued from next page				
106	130	200	18326	7.081
107	310	020	60	2.953
108	330	000	6	1.292
109	201	231	84	3.196
110	103	103	5331	6.190
111	130	100	306	4.129
112	210	113	6	1.292
113	102	132	8	1.500
114	001	031	1	0.000
115	200	233	17	2.044
116	321	321	1027	5.002
117	112	112	65692	8.002
118	110	220	263409	9.003
119	232	232	1087	5.043
120	223	220	309	4.136
121	010	010	1	0.000
122	301	332	1	0.000
123	213	223	3	0.792
124	000	300	3	0.792
125	133	100	129	3.506
126	123	123	2007	5.485
127	323	013	7965	6.480
128	222	122	469	4.437
129	331	302	487	4.464
130	132	131	9	1.585
131	103	200	4	1.000
132	021	311	242	3.959
133	012	012	3825	5.951
134	330	300	914	4.918
135	201	202	2	0.500
136	100	230	1	0.000
137	203	133	2	0.500
138	321	321	115	3.423
139	013	013	447	4.402
140	332	331	480	4.453
141	020	023	1	0.000
142	000	003	1	0.000