RadioGatún, a belt-and-mill hash function

Guido Bertoni, Joan Daemen, Gilles Van Assche
STMicroelectronics

Michaël Peeters
De Valck Consultants

July 20, 2006

Abstract

We present an approach to design cryptographic hash functions that builds on and improves the one underlying the Panama hash function. We discuss the properties of the resulting hash functions that need to be investigated and give a concrete design called RadioGatún that is quite competitive with SHA-1 in terms of performance. We are busy performing an analysis of RadioGatún and present in this paper some preliminary results.

1 Introduction

Cryptographic hash functions are symmetric primitives, used in many cryptographic protocols and as building blocks in many cryptographic functions. In general, a hash function maps a bitstring of any length, the *input*, to a fixed-length *digest*.

Almost all cryptographic hash functions in use today can be considered as strengthened versions of MD4 [17]. This includes MD5 [16], SHA-1 [9], RIPEMD-160 [8], SHA-256 [10], SHA-384 [10] and SHA-512 [10]. These hash functions are defined in terms of the repeated application of a compression function that transforms a state variable and takes a fixed length input block. For their collision-resistance they rely on the collision-resistance of this underlying compression function. The compression functions of these hash functions consist of an input expansion schedule and a state updating function. The input expansion schedule converts the (typically 16) words of an input block to 3 to 5 times as many round input words using a simple function. The state updating function transforms the state by a sequence of simple and invertible rounds each taking a round input word, as many as there are round input words. It is widely believed that this compression function, given an input block, should not be invertible. For that reason, the compression function includes a feedforward loop, whereby the initial value of the state is added to the result of the rounds. For their security all these hash functions rely heavily on the combination of XOR and integer addition. Cryptanalysis has advanced to the level that MD4, MD5 and SHA-1 are nowadays no longer considered to offer a sufficient level of collision-resistance. More recent designs attempt to inspire more confidence by including more complex input expansion schedules, more rounds and a bigger state resulting in hash functions that are slower and bulkier.

Recently, a refreshing approach to hash function design was taken in the form of the SMASH hash function [12]. Unfortunately, the SMASH design approach was broken soon after publication in [14]. Nevertheless, we believe the field of hash functions would greatly benefit from alternative design approaches. We present in this paper a design approach leading to hash functions that are more elegant than the MD4 derivatives, and we believe will offer better security at a lower implementation cost.

In our approach, the hash function consists of the iterated application to a large state of a single simple round function, alternated with the injection of input blocks. After injection of all input blocks, the state undergoes a number of blank rounds after which part of the state is returned as the hash result.

The round function by itself has no cryptographic properties such as collision resistance. As in block ciphers, the supposed cryptographic strength of the function lies in the iterated application of the round function. The design approach we present is not new, but builds further on the one underlying PANAMA [4] that in turn goes back to STEPRIGHTUP [3] and SUBTERRANEAN [2, 3]. In fact, the breaking of the PANAMA hash function in [15] revealed serious problems with its design strategy. We started by analyzing the attack to see how PANAMA could be fixed. This lead to some generalizations of the attack in [15] and a new powerful attack that forced us to abandon some of the design principles underlying PANAMA and introduce other. The result is the design approach presented in this paper. We illustrate it with a concrete design called RADIOGATÚN.

We start by describing a generic cryptographic primitive called an *iterative mangling function* (IMF), of which cryptographic hash functions are a special case. This is followed by a presentation of the alternating-input construction for realising an IMF, a presentation of a generic attack for generating internal collisions called *trail backtracking* and a discussion on other aspects. We then present the belt-and-mill structure for the round function of a alternating-input IMF and discuss implications of trail backtracking on the design. Finally we present our concrete design RADIOGATÚN, provide a rationale for our design choices and discuss hardware and software implementation aspects.

As far as we know, there are no patents on RADIOGATÚN, the belt-and-mill structure or the alternating-input construction.

2 Iterative mangling functions

First we introduce *mangling functions*, which are a generalization of a hash functions. Instead of returning a fixed-length digest, a mangling function returns an infinite output stream. It can be converted to a hash function by truncating the output to the first n bits.

A mangling function takes a variable-length input and returns an infinite output stream. Informally, a good mangling function should not have properties different from that of a random oracle [1]. A random oracle returns a completely randomly generated stream for each different input. So, for example, finding an input that matches a given pattern in the first n bits of the output stream would require computing the mangling function for on the average 2^n inputs. Finding a collision in the first n bits of the output stream would require computing the mangling function for on the average $2^{n/2}$ inputs.

Almost all practical hash functions are *iterative*. The input is padded (if necessary) and split up in a sequence of input blocks. The input blocks are sequentially injected into the state by means of what is usually called a *compression function*. Then a final transformation may be applied to the state resulting in the digest.

Iterative hash functions can be implemented in hardware or software with limited amount of working memory, irrespective of the length of the input. They have however the disadvantage that different inputs may be found that lead to the same value of the state before the final transformation. This is called an *internal collision*.

An iterative mangling function (IMF) can be defined analogously with an iterative hash function as operating on a state with fixed size. Due to the limited size of the state, an IMF cannot behave like a random oracle. If the state has u bits, it can only generate 2^u different output sequences over all possible inputs. Hence, the work factor of finding a collision in the first n bits of the output is upper bounded by $2^{u/2}$.

As behaving like a random oracle is out of reach for any IMF, we replace the objective by a more modest one: to behave not worse than what we call a *random IMF*. A random IMF has a given *capacity* [5], denoted by ℓ_c. A random IMF with capacity ℓ_c consists of the iterated application of a random oracle used as a $\ell_c + 1$-bit to ℓ_c-bit compression function. This compression function consists of calling the random oracle for a $\ell_c + 1$ input and taking the first ℓ_c bits of its output. It has a ℓ_c-bit state initialized to 0 and for each bit of the input the new value of the state is computed by applying the compression function on the concatenation of the state and the input bit. The initial value of the state is 0 and the compression function is iteratively applied for all bits of the input. After this, the output stream is generated by iteratively applying the compression function to the state concatenated with a bit equal to 0 and giving the first bit of the state as output bit.

When designing an IMF, one can claim that it offers a capacity of ℓ_c. Clearly, this claimed capacity cannot be higher than the number of bits in the state in the IMF.

3 The alternating-input construction

We present a simple construction for IMF called *alternating-input*. It consists of the alternation of input injection and a simple invertible round function, followed by a fixed number of rounds without input or output, followed by the iterated application of rounds while returning part of the state. The construction is given in Algorithm 1.

Algorithm 1 The alternating-input construction

```
takes $\ell_\mathrm{i}$-bit input blocks $p_0$ to $p_{n_\mathrm{p}-1}$
generates $\ell_\mathrm{o}$-bit output blocks $z_0$ to $z_{n_\mathrm{z}-1}$
operates on an $\ell_\mathrm{s}$-bit state $S$
$S \leftarrow 0$ {State initialization}
for $i = 0$ to $n_\mathrm{p} - 1$ do
    $T = S \oplus F_\mathrm{i}(p_i)$ { $F_\mathrm{i}$: input mapping}
    $S \leftarrow R(T)$ { $R$: round function}
end for{Injection}
for $i = 0$ to $n_\mathrm{b} - 1$ do
    $S \leftarrow R(S)$
end for{Mangling}
for $i = 0$ to $n_\mathrm{z} - 1$ do
    $S \leftarrow R(S)$
    $z_i = F_\mathrm{o}(S)$ { $F_\mathrm{o}$: output mapping}
end for{Extraction}
```

The input mapping maps the bits of an input block to bits of the state and the output mapping maps bits of the state to the bits of an output block. Both are linear operations. Note that functions such as SUBTERRANEAN and PANAMA fit this model, but also SMASH.

For a alternating-input IMF, the design is reduced to that of the round function, the input and output mappings and the number of blank rounds n_b. The goal is to choose these such that the resulting IMF behaves no worse than a random IMF with a given capacity. We believe the central design challenges are the following, in order of importance:

Internal collisions The expected workload $\mathcal{L}$ to generate internal collisions shall not be less than suggested by the capacity. This is the classic requirement for a cryptographic hash function and addresses the round function and the input layout.

State guessing The expected workload $\mathcal{L}$ to guess the value of the state given a sequence of output blocks shall not be less than suggested by the capacity. This is the classic requirement for a synchronous stream cipher and addresses the round function and the output layout.

Decorrelation The transformation consisting of a sequence of rounds shall not reveal large correlations between any parity of state bits at its output to any parity of state bits at its input. This is the classic requirement for a block cipher to be resistant to linear cryptanalysis and addresses the round function and the number of blank rounds.

Difference propagation The transformation consisting of a sequence of rounds shall not reveal large propagation probabilities between difference in the state at its output to any difference in the state at its input. This is a classical requirement for a block cipher to be resistant to differential cryptanalysis and addresses the round function and the number of blank rounds.

We believe that a alternating-input IMF that meets these challenges behaves no worse than a random IMF with a given capacity. We illustrate this for the often-cited properties of cryptographic hash functions: collision resistance and (2nd-) preimage resistance [13, Table 9.2]:

Collision and 2nd-preimage resistance When internal collisions are infeasible, any pair of inputs will start the blank rounds with a difference. The series of blank rounds operate as a block cipher on the state and the difference propagation property makes controlling the difference after the blank rounds infeasible.

Preimage resistance When state guessing is infeasible, finding a preimage for a given digest is also infeasible.

In this paper we focus on internal collisions as this seems to be the most difficult design goal to meet. State guessing is mostly relevant if the length of the digest is of the same order as the state. Difference propagation and decorrelation can be improved by increasing the number of blank rounds.

4 Internal collisions

In this section we consider approaches to generate internal collisions for alternating-input IMF. First we concentrate on techniques used in differential cryptanalysis to control the difference propagation through the rounds. This results in a criterion for the design of such hash functions: the minimum backtracking cost. This is followed by a treatment of alternative approaches to generate internal collisions.

4.1 Differential trails

Consider the round function R. We denote a differential over the round function in round i by (t'_i, s'_{i+1}) and call it a *round differential.* Its differential probability (DP) is the proportion of *state pairs* $\{T_i, T_i \oplus t'_i\}$ such that $R(T_i) \oplus R(T_i \oplus t'_i) = s'_{i+1}$. If DP > 0, we say the differential is *possible.* The (*restriction*) *weight* of a possible differential, $\mathrm{w_r}(t'_i, s'_{i+1})$, is defined by

$$\mathrm{DP}(t'_i, s'_{i+1}) = 2^{-\mathrm{w_r}(t'_i, s'_{i+1})} \ . \qquad (1)$$

We now define a *(differential) trail* through the IMF. A r-round trail consists of the concatenation of r possible round differentials and is defined by a sequence of r difference triplets plus the final state difference:

$$Q : \left((s'_0, p'_0, t'_0), \ldots, (s'_{r-1}, p'_{r-1}, t'_{r-1}), s'_r\right) . \qquad (2)$$

A trail describes a propagation of differences through the IMF during a number of rounds: p'_i denotes the difference in the input injected before round i and the three members of each triplet are related by $t'_i = s'_i \oplus F_{\mathrm{i}}(p'_i)$. We say this trail *starts in* s'_0 and *ends in* s'_r.

The probability of a trail $\mathrm{DP}(Q)$ is defined as the proportion of all *state/input pairs* with initial state difference s'_0 and r-round input sequence difference $p'_0, p'_1, \ldots, p'_{r-1}$ such that the difference in the state follows the trail. We define the weight of a trail by the sum of the weights of its round differentials:

$$\mathrm{w_r}(Q) = \sum_{i=0}^{r-1} \mathrm{w_r}(t'_i, s'_{i+1}) \ . \qquad (3)$$

If we assume that the the conditions imposed by the round differentials are independent, the weight determines the probability of the trail: $\mathrm{DP}(Q) \approx 2^{-\mathrm{w_r}(Q)}$. However, this independence is not necessarily satisfied and the probability of a trail may be larger than this or even 0 in the case of conflicting conditions.

We call a trail that starts and ends both in 0 a *collision trail.*

4.2 A naive attack

A collision trail Q with non-zero probability can be used to generate internal collisions. An attacker just applies pairs of inputs that exhibit the difference sequence p'_i and verifies whether this results in an internal collision. These pairs look like this:

$$\begin{array}{llll} p_{-d} & p_{1-d} \ldots p_0 & p_1 & \ldots p_{r-1} \\ p_{-d} & p_{1-d} \ldots p_0 \oplus p'_0 & p_1 \oplus p'_1 & \ldots p_{r-1} \oplus p'_{r-1} . \end{array} \qquad (4)$$

The attacker has to try about $1/\mathrm{DP}(Q)$ pairs. In general, the workload $\mathcal{L}$ of the attack may be smaller than $\mathrm{DP}(Q)$ suggests: for the same sequence of input differences $p'_0, p'_1, \ldots, p'_{r-1}$ other collision trails may exist that lead to a collision. The workload $\mathcal{L}$ of the attack is determined by the sum of the probabilities of all collision trails for a given sequence of input differences.

4.3 Trail backtracking

In the trail backtracking attack, the attacker applies pairs of inputs as specified in Equation (4) and tracks the difference propagation as it proceeds through the trail. We call an input *entering* round i the input sequence $p_{-d} \dots p_i$. If we speak of a pair of inputs, we always assume it has the right difference.

Say we have N random pairs of inputs entering round 0. For these pairs, $s'_0 = 0$ and $t'_0 = F_\mathrm{i}(p'_0)$. For each pair we compute the difference after round 0. If this is equal to s'_1, we say it is a right pair *coming out of* round 0. The total number of right pairs coming out of round 0 is about $N2^{-\mathrm{w_r}(t_0,s_1)}$. For each such right pair, we can append an input block p_1 to one member and $p_1 \oplus p'_1$ to the other. As there are 2^{ℓ_i} input block values, this results in $N2^{\ell_\mathrm{i}-\mathrm{w_r}(t_0,s_1)}$ right pairs *entering* round 1. If $\mathrm{w_r}(t'_0, s'_1) < \ell_\mathrm{i}$, the number of right pairs entering round 1 is even larger than the total number of pairs entering round 0. Following this reasoning, and assuming the conditions imposed by the round differentials are independent, the number of right pairs entering round g is

$$N2^{g\ell_\mathrm{i}-\sum_{i=0}^{g-1}\mathrm{w_r}(t_i,s_{i+1})}.$$

If we define the *excess weight* in round g for a trail Q as $\mathrm{W_e}(g) = \sum_{j=0}^{g-1} \ \mathrm{w_r}(t'_j, s'_{j+1}) - \ell_\mathrm{i}$, this becomes

$$N2^{-\mathrm{W_e}(g)}.$$

The number of right pairs coming out of round $h < r$ is:

$$N2^{-\ell_\mathrm{i}-\mathrm{W_e}(h+1)}.$$

The excess weight before the trail is 0, hence $\mathrm{W_e}(0) = 0$.

We can now ask two questions: how large must N be to have a collision with reasonable probability and what is the workload $\mathcal{L}(Q)$ of the resulting attack. We express the workload of the attack $\mathcal{L}(Q)$ by the number of round function evaluations that must be performed. Clearly, this is the sum of the number of right pairs entering each round.

First of all, the number of right pairs coming out of each round h must be at least 1:

$$N \geq \max_h 2^{\ell_\mathrm{i}+\mathrm{W_e}(h+1)} = 2^{\ell_\mathrm{i}+\max_h \mathrm{W_e}(h+1)} \ . \qquad (5)$$

We call the round with the least number of right pairs at its output the *lonesome round.*

For most trails, there is a single round in which the number of right pairs entering is much larger than all other. We call this the *crowded round.* The workload can then be approximated by the number of right pairs entering the crowded round:

$$\mathcal{L}(Q) \approx \max_g N2^{-\mathrm{W_e}(g)} = N2^{-\min_g \mathrm{W_e}(g)} \ .$$

It may happen that there are more than a single round where the excess weight reaches a minimum. In that case the actual workload will be a small factor higher than the value derived here. Filling in the minimum value of N given by Eq. (5) yields:

$$\mathcal{L}(Q) \approx 2^{\ell_\mathrm{i}+\max_{g,h,0\leq g<h\leq r} \mathrm{W_e}(h)-\mathrm{W_e}(g)}. \qquad (6)$$

We define the *backtracking cost* of the trail by

$$\mathrm{C_b}(Q) = \ell_\mathrm{i} + \max_{g,h,0\leq g<h\leq r} \mathrm{W_e}(h) - \mathrm{W_e}(g). \qquad (7)$$

The backtracking cost of a trail can be easily computed from its excess weight profile $\mathrm{W_e}(i)$. This is illustrated in Figure 1.

When a trail has a sequence of rounds where the weight is 0, or smaller than ℓ_i, the excess weight decreases. This suggests that the number of right pairs grows per round. Clearly, the attacker will not need all these right pairs to generate a collision, just as many to have at least one right pair coming out of the lonesome round further along the trail.

Sometimes the differential (t'_i, s'_{i+1}) is independent of some bits or bit parities in p_i. If that occurs in the lonesome round, the backtracking cost increases with the number of independent bit parities that the difference propagation (t'_h, s'_{h+1}) is independent of.

Note that the trail backtracking attack only finds collisions that follow one specific trail, while the naive attack results in an internal collision whenever a collision trail is followed for the given sequence of input differences. It turns out that in practice an input difference must have very many collision trails for the naive attack to be more efficient than the trail backtracker attack. For a given alternating-input IMF, an attacker must hence look for trails with a low backtracking cost, a designer can try to prove lower bounds for the minimum backtracking cost.

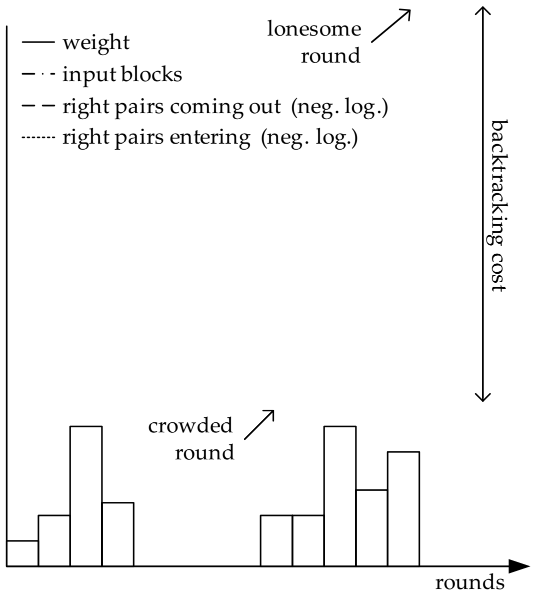

Figure 1: Example of trail with its weight and backtracking cost

4.4 Fixed points

In the attacks considered until now, two inputs that lead to an internal collision have the same length. One may however try to generate collisions with inputs of different length. One method to do so is by exploiting *fixed points*.

Let $\mathcal{F}[x](S)$ be the state transformation of applying r rounds while injecting a series of input blocks

$$x = x_0 \| x_1 \| \dots \| x_{r-1}$$

with $\|$ meaning concatenation. With q is a single input block we can define this recursively as:

$$\begin{aligned} \mathcal{F}[q](S) &= R(S \oplus F_{\mathrm{i}}(q)) \\ \mathcal{F}[x\|q](S) &= R(\mathcal{F}[x](S) \oplus F_{\mathrm{i}}(q)) \ . \end{aligned}$$

For a fixed x, the transformation $\mathcal{F}[x](S)$ is a permutation of ℓ_{s}-bit values. Such a function may have fixed points $y = \mathcal{F}[x_0](y)$. As a matter of fact, if the state space is large, the number of fixed points in a random permutation has a Poisson distribution with $\lambda = 1$ [7]. A fixed point can be used to create internal collisions: given an input sequence x that leads to a fixed point y in the state for $\mathcal{F}[x](S)$, appending any number of strings x will not affect the state and hence result in the same digest whatever the number of strings x attached.

If x is 1 or 2 input blocks and the round function is simple, one may determine the fixed points by algebraically solving a set of equations. For longer values of x, determining fixed points becomes increasingly difficult. As the number of input blocks in x grows, the permutations $\mathcal{F}[x](S)$ are expected to behave more like random permutations. For realistic values of the state size, the number of fixed points that can be determined with respect to all state values is negligible. Still, the designer may investigate whether the number of fixed points for short x is not too large or that some easy to reach state values are not fixed points. Easy to reach state values include the initial all-zero value or any value that can be reached from this after one or two rounds.

If constructing an input sequence that leads to a fixed point would be easy, generating an internal collision would also be easy: take a random input and consider the resulting state value. Then construct the input that leads to the same state value. One may try to address fixed points by having part of the state and round function to behave as a counter, or alternatively, include in the padding of the input a coding of its length. However, this only complicates the IMF and we think it is better to fix the design rather than add a counter to solve the problem.

4.5 Starting from non-zero difference

One may also consider several (or many) input sequences, each one leading to a particular value of the state, and then try to find pairs among these with a state difference from which there is *an easy route to an internal collision.* Note that a trail with low backtracking cost does not necessarily help because the attacker does not have the degrees of freedom as in the collision trails before the first round of the trail. In any given pair of input sequences, the absolute values of bits are fixed. In this context, one may conduct a quantitative analysis on the proportion of state differences from which there are trails with low weight or backtracking cost to an internal collision.

4.6 Computing backwards

As the round function is invertible, it is possible to compute backwards from a given state value. However, any valid state value must result from applying an input to an all-zero initial state. We believe that reaching the all-zero initial state computing backwards will not be less difficult than reaching a target state starting from the initial state. Still, computing backwards can be exploited in meet-in-the-middle strategies where an input leading to a particular value of the state might be constructed by appending two input sequences. Here the number of bits in the state whose value may be easily controlled by applying input blocks in consecutive rounds plays an important role and may be investigated by the designer.

4.7 Algebraic attacks and techniques

The definition of the round function allows the state at its output to be expressed as a function of the state at its input. In general, one can use this to express the output state bits as Boolean functions of the input state bits. In some cases one may find simpler equations by grouping the bits in b-bit *words* and interpreting these as elements of $\mathrm{GF}(2^b)$ or integers.

Exploiting these equations, finding an input sequence leading to a given hash result or internal state can be reduced to solving a system of algebraic equations. The difficulty of solving such a system is in general hard to evaluate and depends on the number of variables and equations, the complexity of the latter and their interconnection. Clearly, the amount of diffusion and nonlinearity of the round function plays an important role in this respect.

Consider a collision-generating attack based on a differential trail. With a round differential (t'_i, s'_{i+1}) corresponds a set of right state pairs. If the restriction weight $\mathrm{w_r}(t'_i, s'_{i+1})$ is an integer, it can be interpreted as follows. The bits of the right state pairs satisfy a set of independent Boolean equations each with probability 1/2 of being satisfied. The number of equations is the restriction weight. Now, given S_i, some of these equations can be converted to equations in bits of the input block p_i. By taking into account these conditions in the generation of input pairs, the complexity of the attack can be improved. In general, the ability to choose input pairs that satisfy b binary conditions with certainty, reduces the expected number of pairs to try by a factor 2^b.

In fact, at most ℓ_i of these equations can be converted to equations in bits of the input block p_i. The remaining ones can only be addressed by the attacker by transferring them via the round function to equations in bits of earlier input blocks. For round $i-1$, we similarly have $\mathrm{w_r}(t'_{i-1}, s'_i)$ equations that must be transferred to input blocks. If we require both round $i-1$ and round i differentials to be followed, the number of equations that must be transferred to input blocks entering round $i-1$ is the sum of the two weights minus ℓ_i. In general, when requiring the round differentials of rounds g up to h to be satisfied, the number of equations that must be transferred to input blocks entering round g is $\mathrm{W_e}(h) - \mathrm{W_e}(g) + \ell_\mathrm{i}$. If we apply this to the full trail, this number reaches a maximum when g is the crowded round and h the lonesome round and is equal to the backtracking cost $\mathrm{C_b}(Q)$ of the trail.

We can now define the *backtracking depth* d of a trail as its backtracking cost divided by ℓ_i. The backtracking depth gives the number of rounds over which an attacker performing an algebraic attack must transfer equations on the state to conditions on the input blocks.

Given a round function, decreasing ℓ_i has a negative impact on performance but a positive on security as it has a threefold impact on the minimum backtracking depth:

- It gives less freedom for trail construction and will typically increase the minimum weight of trails.

- For the same trail, it increases the $\mathrm{w_r}(t'_j, s'_{j+1}) - \ell_\mathrm{i}$ of the round differentials and hence the backtracking cost of the trail.

- For the same backtracking cost, it increases the backtracking depth.

5 The belt-and-mill structure

The round function is the central component of any alternating-input IMF. In this section we generalize the structure of the round function of Panama and call it the *belt-and-mill* structure. The state consists of two parts, the belt and the mill, and the round function treats them very differently. It consists of four operations that can take place in parallel:

Mill function an invertible non-linear function applied to the mill,

Belt function an invertible simple linear function applied to the belt,

Milt feedforward some bits of the mill are fed to the belt in a linear way,

Bell feedforward some bits of the belt are fed to the mill in a linear way.

The algorithm is given in Algorithm 2.

Algorithm 2 A belt-and-mill round function

```
$(A, B) = R(a, b)$, with
$A = \textsc{Mill}(a) \oplus \textsc{Bell}(b)$ and
$B = \textsc{Belt}(b) \oplus \textsc{Milt}(a)$
```

The positions of the bits that are fed forward shall be chosen so that the resulting round function is invertible. If $\textsc{Bell}(\textsc{Belt}^{-1}(\textsc{Milt}(a))) = 0$ for any value of a, then $\textsc{Bell}(b) = \textsc{Bell}(\textsc{Belt}^{-1}(B))$ can be found from B, allowing to recover a, hence making the round function invertible.

The only non-linear component in the round function is the mill function. This has an important impact on its differential propagation properties.

Consider differentials over the round function. Using the linearity of the functions, given an input difference (a', b'), the output difference (A', B') is:

$$A' = \textsc{Mill}(a) \oplus \textsc{Mill}(a \oplus a') \oplus \textsc{Bell}(b') \quad (8)$$

$$B' = \textsc{Milt}(a') \oplus \textsc{Belt}(b') \quad (9)$$

The output difference in the belt B' is fully determined by the input difference and hence independent of the value of the state (a, b). The output difference in the mill A' is also independent of b but depends on a through $\textsc{Mill}(a) \oplus \textsc{Mill}(a \oplus a')$. It follows that the DP and weight of a possible round differential is fully determined by the differences at the input and the output of the mill function. We have:

$$\mathrm{w_r}((a', b'), (A', B')) = \mathrm{w_r}(a', A' \oplus \textsc{Bell}(b')) \quad (10)$$

Any differential with $a' = 0$ has weight 0 and hence imposes no conditions on the state. Moreover, this upper bounds the weight of round differentials to the number of bits in the mill minus 1.

As for algebraic attacks, only the mill function results in nonlinear equations and all other equations are linear. This limits the number of internal Boolean variables that need to be introduced per round to the number of bits in the mill.

The belt and mill each have their own function. The mill is the confusion engine, with a task similar to that of a round function in a SPN block cipher or the F function in a Feistel block cipher. It should provide nonlinearity and local diffusion. The belt has a function similar to that of the key schedule in block ciphers, the input expansion in hash functions or the switching of the two halves in a Feistel cipher. It takes care of global diffusion and plays an important role in avoiding collision trails with low cost.

The Panama round function has the belt-and-mill structure. However, it has two modes called *push* and *pull*. The push mode is used in the input injection phase and the pull mode in the output extraction phase. In the push mode, there is no feedforward from the mill to the belt, leading to a linear dependence of all belt bits from the input sequence. This property was exploited in the breaking of the Panama hash function. We refer to Appendix A for more explanations on this.

6 RadioGatún

We say:

$$z = \textsc{RadioGatún}[\ell_{\mathrm{w}}](x) \quad (11)$$

with

- x: input that can be a bit string of any length

- ℓ_w: parameter word length that can have any value from 1 to 64. Each value of ℓ_w defines another function. The word length is by default 64: RadioGatún means RadioGatún[64].

- z: infinite length output stream.

We claim that RadioGatún[ℓ_w] offers a security level indicated by a capacity $\ell_c = 19\ell_w$. For the 64-bit version RadioGatún this is a capacity of 1216 bits, for the 32-bit version and 16-bit version this gives 608 and 304 bits respectively.

RadioGatún[ℓ_w] can be used as a hash function with a ℓ_h-bit digest by taking the first ℓ_h bits of the output stream. Note that taking for values $\ell_h > \ell_c$ the claimed collision-resistance level is determined by ℓ_c rather than ℓ_h.

Reference and optimized code, test vectors and extra information can be found in [18].

6.1 The building blocks

The RadioGatún function is a alternating-input IMF with the belt-and-mill structure. The mill a consists of 19 words $a[i]$, the belt b of 13 stages $b[i]$ of 3 words $b[i, j]$ each. An input block p consists of 3 words $p[i]$, an output block z consists of 2 words $z[i]$. All indexing starts from 0.

The round function is specified in Algorithm 3 and illustrated in Figure 2. It makes use of the mill function that is specified in Algorithm 4.

The input mapping is specified in Algorithm 5 and the output mapping in Algorithm 6.

Algorithm 3 The round function R

```
$(A, B) = R(a, b)$
for all $i$ do
    $B[i] = b[i + 1 \bmod 13]$
end for{Belt function: simple rotation}
for $i = 0$ to 11 do
    $B[i + 1, i \bmod 3] = B[i + 1, i \bmod 3] \oplus a[i + 1]$
end for{Mill to belt feedforward}
$A = \textsc{Mill}(a)$ {Mill function}
for $i = 0$ to 2 do
    $A[i + 13] = A[i + 13] \oplus b[12, i]$
end for{Belt to mill feedforward}
```

Algorithm 4 The mill function

```
$A = \textsc{Mill}(a)$
all indices should be taken modulo 19,
$x \ggg y$ denotes rotation of bits within $x$ over $y$ positions
for all $i$ do
    $A[i] = a[i] \oplus \overline{\overline{a[i + 1]}a[i + 2]}$
end for{$\gamma$: non-linearity}
for all $i$ do
    $a[i] = A[7i] \ggg i(i + 1)/2$
end for{$\pi$: intra-word and inter-word dispersion}
for all $i$ do
    $A[i] = a[i] \oplus a[i + 1] \oplus a[i + 4]$
end for{$\theta$: diffusion}
$A[0] = A[0] \oplus 1$ {$\iota$: asymmetry}
```

Algorithm 5 The input mapping F_i

```
$(a, b) \leftarrow 0$
for $i = 0$ to 2 do
    $b[0, i] = p[i]$
    $a[i + 16] = p[i]$
end for
Return $(a, b)$
```

Algorithm 6 The output mapping F_o

```
$z[0] = a[1]$
$z[1] = a[2]$
Return $z$
```

6.2 The function

The function is specified as follows.

- First apply reversible padding to x by appending a single bit equal to 1 and zeroes until the length of the result is a multiple of the input block length and decompose the latter in input blocks p_0 to $p_{n_p - 1}$.

- Then execute Algorithm 1 with the round function specified in Algorithm 3 until sufficient output bits are generated. The number of blank rounds is $n_b = 16$.

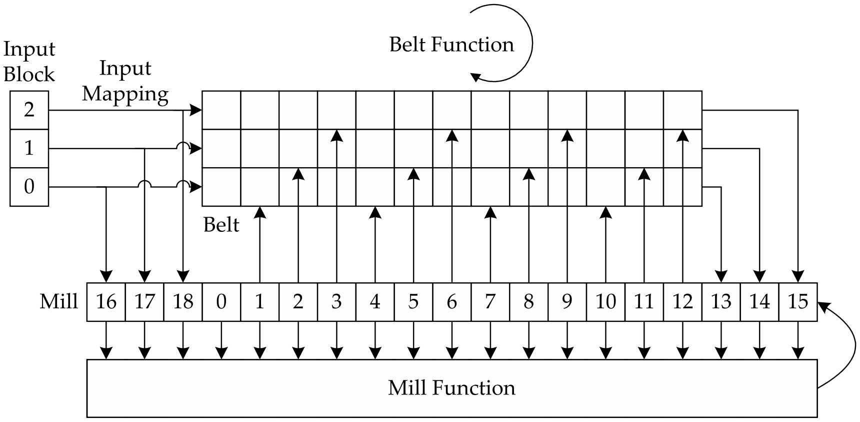

Figure 2: The RadioGatún round function and input mapping

7 Design rationale

The design of RadioGatún borrows heavily from Panama: the latter can be seen as a strengthened version of the former. In this section we first discuss the properties of the mill function followed by a treatment of the one-bit word version RadioGatún[1]. Finally we describe how we went about deciding the dimensions and layouts in RadioGatún.

7.1 The mill function

The mill function consists of a sequence of four invertible transformations and is very similar to the function ρ in Panama. The only difference is that the mill of RadioGatún has length 19 while this is 17 for Panama and the fact that the injection of input blocks and belt words is not part of the mill function. γ and θ are both shift-invariant transformations that are invertible for length 19 [3, Chapter 6]. The permutation π combines rotation of the bits within the words and a permutation of the word positions. ι is only there to break the symmetry.

Within the mill function, only γ is non-linear. It has algebraic degree 2 in GF(2) and consequently specific differential propagation properties. Note that its inverse does not have algebraic degree 2. The differential propagation properties of γ are studied in [3, Section 6.9]. We give a short summary here:

- Given an input difference a', the values c' for which (a', c') is a possible differential form an affine space $\overline{\gamma(a')} \oplus V_a$ with V_a a vector space completely determined by a'.

- The (restriction) weight of all these differentials is equal to the dimension of V_a . This means:

    - The weight of (a', c') for all possible values of c' is the same and hence completely determined by a'. We can say $\mathrm{w_r}(a', c') = \mathrm{w_r}(a')$.

    - The weight of any differential over γ is an integer.

- The differential (a', c') imposes $\mathrm{w_r}(a')$ affine Boolean conditions on the input a.

Thanks to the fact that π, θ and ι are linear, given a difference a' at the input of the mill function, the possible differences at the output of the round form an affine space. Note that the inverse is not the case: given a difference at the output of the mill function, the possible input differences do not form an affine space.

7.2 One-bit word version

The state of RadioGatún[$\ell_{\mathrm{w}} = 1$] has size $19 + 3 \times 13 = 58$ bits. Although this is too small to provide a practical cryptographic hash function, we can nevertheless learn something from the case of one-bit words.

First of all, RadioGatún[n] with $n = 1$ or small are interesting subjects for trying out algebraic attacks and getting a better understanding of their complexity.

Another interesting exercise consists in finding collision trails for RadioGatún[1]. Due to the symmetry of the construction, a trail for RadioGatún[1] can be extended to any other word size $\ell_{\mathrm{w}} > 1$: for each bit of the one-bit input or state difference, it is repeated ℓ_{w} times, i.e., 0 (resp. 1) becomes $0^{\ell_{\mathrm{w}}}$ (resp. $1^{\ell_{\mathrm{w}}}$), where the exponentiation by n denotes the concatenation of n identical strings. In RadioGatún, all the operations independently operate on each bit of the words, except for the (intra-word) rotation in π. Clearly, the repeated bit difference is insensitive to the rotation. Note that the constant in ι is not symmetric, that is, not identical for all bits. If this constant was symmetric, RadioGatún[$\ell_{\mathrm{w}} > 1$] would reduce to RadioGatún[1] for symmetric inputs.

As a convention, we say a trail is *1-symmetric* (or simply *symmetric*) if it is invariant to a rotation of 1 bit within a word. In such a trail, the words are either all zeroes or all ones. A symmetric trail in RadioGatún[ℓ_{w}] has a cost equal to the cost of the corresponding trail in RadioGatún[1] multiplied by ℓ_{w}. This follows from the fact that the restriction weight in RadioGatún[ℓ_{w}] is the sum of the restriction weights in each of the ℓ_{w} one-bit slices.

Note that this extension can be generalized to other small variants of RadioGatún. For instance, a trail of RadioGatún[2] can be extended to any even word size. Whenever a 2-bit word difference is xy, $x, y \in \mathrm{GF}(2)$, it becomes $(xy)^{\ell_{\mathrm{w}}/2}$. In general, a trail in RadioGatún[ℓ_{w}] can be extended to a ℓ_{w}-symmetric trail in RadioGatún[$n\ell_{\mathrm{w}}$] for any positive integer n. In this current study, however, we limit ourselves to the case of 1-symmetric trails.

Finding internal collisions for RadioGatún[1] is feasible: thanks to the birthday paradox, after taking about $2^{58/2} = 2^{29}$ random inputs, it is likely to find two of them that cause an internal collision.

We looked for internal collisions by using input pairs of the form as in Equation (4) with $d = 7$ and $r = 7$; the input block p_{-7} is the first block of the input. There are thus seven input blocks of preparation, which determine the absolute value of the state before the differences can appear. Eight block pairs are then input to RadioGatún[1], each possibly containing a difference. Remember that each input block consists only of three bits.

Due to time considerations, the search was restricted to the case of $p_{-7} = 0$. Nevertheless, we searched the described input space exhaustively and found more than 4 million internal collisions. The minimum backtracking cost found was $\mathrm{C_b} = 46$ and there were two trails with this cost. Then, a few internal collisions have cost $\mathrm{C_b} = 51$. The distribution is centered around a maximum in $\mathrm{C_b} = 107$.

For instance, exploiting this given trail with $\mathrm{C_b} = 46$ using the trail backtracking attack of Section 4.3 has a complexity of about 2^{46} evaluations of the round function, much higher than that of the birthday paradox attack.

Used as a symmetric trail in RadioGatún with $\ell_{\mathrm{w}} > 1$, the backtracking cost scales with the word size ℓ_{w}. Following the same example, the complexity of the backtracking attack is $2^{46\ell_{\mathrm{w}}}$. Thanks to the rotation in π, one cannot apply this attack independently on each of the ℓ_{w} bits, and thanks to the asymmetric constant in ι, one cannot work only with symmetric words as input. Also, to exploit this symmetric trail with $\mathrm{C_b} = 46\ell_{\mathrm{w}}$ using the backtracking attack, one needs to populate the crowded round with enough input pairs, which requires more than fifteen rounds with three input words each to get enough degrees of freedom.

This investigation is ongoing work and should be taken rather carefully. First of all, we found the symmetric trail with the lowest backtracking cost only within the input space described above, not in an absolute sense. There may well be trails with a lower backtracking cost. Moreover, it is likely that asymmetric trails exist for RadioGatún with $\ell_{\mathrm{w}} > 1$, with a lower backtracking cost than for symmetric ones. Yet, this experiment is interesting in that we can challenge our design. We have tried a number of configurations, focusing on the internal collisions, and the one presented in this paper came out best for the time being.

7.3 Dimensions and layouts

The dimensions of the belt and mill in RadioGatún and the input, output, bell and milt layout are the result of an iterative adjusting process that had Panama as a starting point. The adjustments were triggered by our discovery of the trail backtracking attack and the outcome of trail- and collision-search experiments for the one-bit word versions. In our choice between different alternative modifications, we always tried to choose the simplest one. The most important differences between RadioGatún and Panama are the following:

- Introduction of feedforward from mill to belt (See Appendix A)
- Reduction of input block size and belt width
- Expansion of the mill from 17 to 19 words

The number of blank rounds is 16, resulting in 18 rounds between the injection of the last input block and the output of the first output block. In the 64-bit version, it takes 18 rounds for each bit of the mill to depend on each bit of the state.

8 Performance aspects

We now give some elements on its performance with a comparison to SHA-1, SHA-256 and Panama as references.

8.1 Software

To demonstrate its suitability for fast software implementation, we have written optimized C code for RadioGatún and made it publically available at [18]. Given its similarity with Panama, we have based ourselves on the high-speed Panama code in the Crypto++ library, publically available from [6]. We expect that further optimization may result in even better performance.

Table 1: Software performance in MByte/sec.

	Windows	Linux
	(32 bits)	(x86_64)
	Visual Studio 2005	GCC 3.3.5
SHA-1	90	91
SHA-256	65	80
Panama	480	288
RadioGatún[32]	120	175
RadioGatún[64]	55	270

Table 1 compares the speed for hashing long inputs of our optimized code for RadioGatún with the Crypto++ code [6] of SHA-1, SHA-256 and Panama. These measurements were taken on a Dell Precision 670 with Intel Xeon 3GHz.

For short inputs, the fixed cost due to the blank rounds and the padding becomes significant. This cost can be modeled in terms of *extra bytes* to be processed. In RadioGatún[64], the blank rounds take as much time as iterating 384 bytes and the padding adds 1 to 24 bytes. In RadioGatún[32], the blank rounds add 192 bytes and the padding adds 1 to 12 bytes. In Panama the blank rounds add 1056 bytes and the padding adds 1 to 32 bytes. In SHA-1 and SHA-256, there are no blank rounds but the padding adds between 1 and 64 bytes. Using the numbers of the last column of Table 1, hashing an input of length smaller than a single input block takes about 0.7μsec for SHA-1, 0.8μsec for SHA-256, 3.8μsec for Panama, 1.2μsec for RadioGatún[32] and 1.5μsec for RadioGatún[64].

8.2 Hardware

RadioGatún can be implemented in a straightforward way as a circuit that keeps the state and has a state-updating function that consists of the input injection followed by a round. By analyzing the structure of the mill function and its connection to the belt function, it is clear that the critical path of this circuit consists of a NOT, a two-input NAND, a two-input XOR, a three-input XOR and a two-input XOR. In such a circuit, about 50% of the area is used for storing the bits of the state.

The structure of RadioGatún allows for area/speed trade-offs. Area can be reduced at the cost of increasing the number of cycles per input block by instantiating only a fraction of the mill function. Speed can be doubled for the same clock rate by cascading two rounds and input injections in the updating function. Such a circuit has the same number of registers for storing the state and only doubles the logic that implements the updating function. Since the critical path of a round and input injection is very short, the maximum clock frequency remains high compared to other hash functions such as SHA-1.

We have modeled a straightforward RadioGatún circuit in VHDL, mapped it to a technology library of STMicroelectronics and used a general purpose 0.13 μm library. This resulted in a circuit with 37 Kgates for RadioGatún[64] and 18.5 Kgates for RadioGatún[32] that can both run up to 1 GHz.

In the case of SHA-1 and SHA-256, a common choice is to instantiate the round logic and using it 80 and 64 times respectively for processing 512 bits of the input. Typical examples are the Helion ASICs described in datasheets available from [11]. Their designs on 0.18 μm take 20 to 26 Kgates and run from 150 MHz up to 290 MHz.

Taking into account the additional cycles due to the blank rounds and padding, we can give expressions for the number of clock cycles required for hashing an n-bit input to a 256-bit hash result (160 bits for SHA-1). Our RadioGatún[32] and RadioGatún[64] circuits take $\lceil n/96 \rceil + 20$ and $\lceil n/192 \rceil + 18$ clock cycles respectively, while the Helion SHA-1 and SHA-256 ASICs require $80 \times \lceil n/512 \rceil$ and $64 \times \lceil n/512 \rceil$ clock cycles respectively. For the same clock frequency, RadioGatún[32] is 12 times faster than SHA-256 for long inputs and 3.2 times faster for short inputs, while it takes less gates. RadioGatún[64] is even 24 times faster than SHA-256 for long inputs, but has about 50% more gates. The same exercise can be easily done for SHA-1.

9 Conclusions

Alternating-input IMF with the belt-and-mill structure are an interesting alternative for MD4-like hash functions.

References

[1] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," *ACM Conference on Computer and Communications Security 1993*, ACM, 1993, pp. 62-73.

[2] J. Daemen, L. Claesen, M. Genoe, G. Peeters, R. Govaerts and J. Vandewalle, "A Cryptographic Chip for ISDN and High Speed Multi-Media Applications," *Proceedings of VLSI Signal Processing VI*, L.D.J. Eggermont, P. Dewilde, E. Deprettere and J. van Meerbergen, Eds., IEEE, 1993, pp. 12-20.

[3] J. Daemen, "Cipher and hash function design strategies based on linear and differential cryptanalysis," *Doctoral Dissertation*, March 1995, K.U.Leuven.

[4] J. Daemen and C.S.K. Clapp, "Fast hashing and stream encryption with PANAMA," *Fast Software Encryption 1998, LNCS 1372*, S. Vaudenay, Ed., Springer-Verlag, 1998, pp. 60-74.

[5] J. Daemen and V. Rijmen, "A new MAC Construction ALRED and a Specific Instance ALPHA-MAC," *Fast Software Encryption 2005, LNCS 3557*, H. Gilbert and H. Handschuh, Eds., Springer-Verlag, 2005, pp. 1-17.

[6] W. Dai, Crypto++ 5.2.1 Benchmarks, `http://www.eskimo.com/~weidai/benchmarks.html`, 2004.

[7] Wikipedia, Random Permutation, `http://en.wikipedia.org/wiki/Random_permutation`, 2006.

[8] H. Dobbertin, A. Bosselaers and B. Preneel, "RIPEMD-160: A Strengthened Version of

RIPEMD", Fast Software Encryption 1996, LNCS 1039, Ed. D. Gollmann, pp. 71-82, 1996

[9] *Federal Information Processing Standard 180-1, Secure Hash Standard*, FIPS-180-1, NIST, April 1995.

[10] *Federal Information Processing Standard 180-2, Secure Hash Standard*, FIPS-180-2, NIST, August 1, 2002.

[11] Helion Authentication cores, `http://www.heliontech.com/auth.htm`, 2006

[12] L. Knudsen, "SMASH, A Cryptographic Hash Function," *Fast Software Encryption 2005, LNCS 3557*, H. Gilbert and H. Handschuh, Eds., Springer-Verlag, 2005, pp. 228-242.

[13] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[14] N. Pramstaller, C. Rechberger, V. Rijmen, "Breaking a New Hash Function Design Strategy Called SMASH," *Selected Areas in Cryptography 2005, LNCS 3897*, B. Preneel and S. Tavares, Eds., Springer-Verlag, 2006, pp. 233-244.

[15] V. Rijmen, B. Van Rompay, B. Preneel, J. Vandewalle; "Producing Collisions for PANAMA," *Fast Software Encryption 2001, LNCS 2355*, M. Matsui, Ed., Springer-Verlag, 2002, pp. 37-51.

[16] RFC 1321, "The MD5 message-digest algorithm", Internet Request for Comments 1321, R. Rivest, April 1992.

[17] R. Rivest, "The MD4 message digest algorithm", *Advances in Cryptology Crypto '90, LNCS 537*, A. Menezes and S. Vanstone, Ed., Springer-Verlag, 1991, pp. 303-311.

[18] RadioGatún web page, `http://radiogatun.noekeon.org`, 2006

A On Panama and its attacks

Panama can be presented as an alternating-input IMF with the belt-and-mill structure. The belt b is a linear feedback shift register (LFSR) with 32 stages each containing 8 32-bit words. The mill a is governed by a mill function. The input mapping maps an 8-word input block to the first stage of b and 8 of the 17 words of a. The Panama function was designed with collision-resistance as one of the main criteria.

In Panama, the function of the belt is to ensure that internal collisions can only occur for inputs that have a difference that satisfies certain criteria. Basically, in Panama each difference in the input that potentially leads to an internal collision leads to difference injections in the mill a in at least 5 different rounds.

It was shown that Panama is not collision-resistant in [15]. The attack actually injects a sequence of differences in 5 different instances in the mill each time over 3 rounds. A first difference, then no difference and then a second difference that results in an internal collision in the mill. All differences are 1-symmetric. We call these *local mill collisions.* In the belt, this difference sequence leads to an internal collision at the moment the last difference is injected.

Each of the local mill collisions consist of a sequence of two differentials over the mill function, where each differential imposes conditions on the absolute values of the bits of the mill at the input of the round. The authors of [15] noticed that some of these conditions can be easily converted to conditions on input bits, while for satisfying the remaining conditions one must try many input pairs and hope they are satisfied. Based on this, a collision-generating attack was found with complexity below that given by the birthday paradox. Having made their point, the authors did not attempt to improve their attack. Notice that the attack of [15] can be modeled as being based on a collision trail and that this collision trail has a backtracking cost. The complexity of the attack is given by 2^n with n the backtracking cost minus the number of conditions that can be satisfied by choosing input blocks with the correct values.

When studying the attack with the aim of patching up Panama, it became clear to us that it can be made more efficient:

- For the basic difference sequence they only considered a sequence $dx, 0, dy$. In fact it is likely that there are differences of type dx, dy, dz that also lead to local collisions in the 5 instances. As the number of such sequences is $(2^8-1)^3$ instead of $(2^8-1)^2$ for those of type $dx, 0, dy$ it is very likely that this will reduce the attack complexity.

- They only considered 1-symmetric differences. By also considering differences with less symmetry, the search space becomes larger, probably leading to even better attack complexity.

- They did not do a lot of effort to resolve the equations algebraically. One may convert conditions on bits of the mill to conditions on the input of some rounds earlier, thereby greatly reducing the number uncontrolled bits and the number of inputs to try.

The first approach to fix these problems was to modify the belt LFSR so that the number of times a difference is injected in the state grows from 5 to a large number. This imposes more restrictions on the basic difference sequence as it must have collision trails in the mill for each of the instances, and hence in general raises the weight of the trails under consideration. However, we then realized that there is a generic collision attack that works for any efficient IMF where there is no feedforward from mill to belt.

This attack takes any pair of input sequences and constructs trailing parts so that the inputs lead to an internal collision. It goes like this:

- Apply two input sequences to the function, leading to two values of the internal state.

- Consider the difference in the internal state. Compute the difference that the input blocks must have to finally result in a zero difference in the belt. This is solving a set of linear equation and is easy.

- Pick a pair of input trailers that have the correct difference and perform a number of rounds for injecting them, minus the last few rounds.

- Look now at the mill. Try to find from the current difference in the mill a differential trail leading to zero. This trailfinding seems not so difficult because the output difference of the last round is known and the input difference of the current round is known, meet-in-the-middle is possible.

- Try now to convert the conditions on the bits of the mill imposed by the mill differentials to bits of the input. This involves algebraic computations that become more difficult with the number of rounds have to be bridged. Converting conditions in the mill in round t to conditions in the input in round $t-m$ becomes more difficult as m grows. We expect the number of conditions to be about the order of magnitude of the number of bits in the mill. So the number of blocks required is the size of the mill divided by the size of an input block.

In Panama the mill has 17 words and the input 8 words, so only two input blocks have to be considered. To find an internal collision in a at time t, it is sufficient to express the conditions on bits of a^{t-1} and a^{t-2} to conditions on p^{t-2} and p^{t-3}. Hence this spans only two rounds. In order to make such a function resistant to this attack, the mill must be several times larger than an input block. However, the speed of the function is proportional to the size of an input block divided by the size of the mill, as the mill function takes the largest portion of the computational resources. From this we concluded that having no feedback from mill to belt would never lead to an efficient hash function.