The making of Keccak

Guido Bertoni[1], Joan Daemen[1], Michaël Peeters[2] and Gilles Van Assche[1]

[1] STMicroelectronics
{guido.bertoni,joan.daemen,gilles.vanassche}@st.com
[2] NXP Semiconductors
michael.peeters@nxp.com

Abstract. The sponge function Keccak is the versatile successor of SHA-1 and the SHA-2 series of hash functions. Its structure and components are quite different from its predecessors and at first sight it seems like a complete break with the past. In this paper we show that Keccak is the endpoint of a long learning process involving many intermediate designs, mostly gradual adaptations but also some drastic changes of direction. We take off from our attempts at fixing Panama [26], resulting in RadioGatún [4] and our insights on trail backtracking applied to generalizations of Panama and RadioGatún, known as *alternating-input* and *belt-and-mill* structures. We explain how we originally presented the sponge construction to compactly express security claims for our proposals and how we finally decided to use it in an actual design, that would become Keccak. Then we explain the design choices made in Keccak and how some of its building blocks can be traced back to its predecessor RadioGatún and even earlier.

Keywords: Keccak, SHA-3, sponge functions

1 Introduction

On Tuesday, October 2, 2012, the American National Institute of Standards and Technology (NIST) announced the selection of Keccak as the winner of the SHA-3 Cryptographic Hash Algorithm Competition [40,41]. Its structure and building blocks are quite different from that of its predecessors SHA-1 and the SHA-2 series of hash functions and at first sight it seems like a complete break with the past. Keccak is the result of a long and winding process that we report on in this paper, starting from our initial attempts to tweak the Panama hash function [26] after it had been broken in [42] to the final version of Keccak that is the basis for the SHA-3 standard.

This paper is structured as follows. In Section 2 we recall the alternating-input structure, that is shared by RadioGatún and its predecessor Panama. We describe a powerful attack that applies to this structure called *trail backtracking* and the parameters relevant for its expected workload. In Section 3 we recall the belt-and-mill structure that is a refinement of alternating-input. Section 4 is dedicated to the Panama hash function, the attacks that broke it and our attempts at fixing it. In Section 5 we describe our design RadioGatún and our initial analysis of it. In Section 6 we discuss the role played by Gnoblio, a work-in-progress similar to to RadioGatún but aimed at gaining more insight and third-party cryptanalysis of RadioGatún. In section 7 we sketch why we introduced the sponge construction and how it evolved from a mere theoretical tool for expressing security claims to the construction underlying Keccak. In Section 8 we explain for each of the building blocks of Keccak how they got their final form. Finally we derive some conclusions and look towards the future in Section 9.

2 Generating internal collisions in the alternating-input structure

The RadioGatún and Panama hash functions and their predecessors are all based on a simple structure that we formalized in [4] under the name *alternating-input*. In this section we recall this structure, describe a generic attack that applies to it called trail backtracking and the parameters relevant for its expected workload.

2.1 The alternating-input structure

The alternating-input structure operates in three phases. The first phase consists of the alternation of the injection of r-bit input blocks and the application of an invertible round function R to the state. We call r the bitrate. The second phase consists of a fixed number of round function iterations on the state without input or output (blank rounds). The third phase consists of the iterated application of the round function while returning part of the state in between the rounds. The construction is specified in Algorithm 1.

Algorithm 1 The alternating-input construction

```
Input: sequence of $r$-bit blocks $p_0$ to $p_{n_p-1}$
Output: sequence of $r_o$-bit blocks $z_0$ to $z_{n_z-1}$
Operates on an $b$-bit state $S$
$S \leftarrow 0$ {State initialization}
for $i = 0$ to $n_p - 1$ do
    $T = S \oplus F_i(p_i)$ { $F_i$: input mapping}
    $S \leftarrow R(T)$ { $R$: round function}
end for{Injection}
for $i = 0$ to $n_b - 1$ do
    $S \leftarrow R(S)$
end for{Mangling}
for $i = 0$ to $n_z - 1$ do
    $S \leftarrow R(S)$
    $z_i = F_o(S)$ { $F_o$: output mapping}
end for{Extraction}
```

The input mapping F_i maps the r bits of an input block to bits of the state and the output mapping F_o selects r_o bit from the state to form an output block. Both are linear operations.

Adopting the alternating-input construction reduces the design to that of the round function, the input and output mappings and the number of blank rounds n_b. Informally speaking, the goal is to choose these such that the resulting function is *secure*. In the design approach internal collisions play a central role. An internal collision is a pair of different inputs that leads to the same value of the state. The design approach is the following:

Internal collisions Design R and F_i such that generating internal collisions is hard.

State recovery Design R and F_o such that recovering the state from a sequence of output blocks is hard.

Decorrelation Design R and choose n_b such that the permutation formed by the blank rounds does not have input-ouput correlations significantly higher than in a random permutation.

Difference propagation Design R and choose n_b such that the permutation formed by the blank rounds does not have differentials with differential probability (DP) value significantly higher than in a random permutation.

We illustrate how this helps in satisfying the often-cited requirements of cryptographic hash functions: collision resistance and (2nd-) preimage resistance [38, Table 9.2]:

Collision and 2nd-preimage resistance If generating internal collisions is infeasible, any pair of inputs will have a non-zero difference at the input of the blank rounds. The difference propagation property makes controlling the difference after the blank rounds infeasible. Note that a 2nd-preimage implies a collision.

Preimage resistance Having a preimage implies recovering a state that leads to the given output. So if state recovery is infeasible, finding a preimage for a given digest is also infeasible.

When hashing long messages with alternating-input constructions, the speed is determined by the complexity of R and size of the input blocks. These in turn determine the resistance against internal collisions. For that reason, our research effort spends most time studying resistance against internal collisions.

We investigated several aspects of this such as finding fixed points of the round function, trying to generate collisions from arbitrary state differences and computing the round function backwards. Still, the central criterion in the design of alternating-input structure is the resistance against generating internal collisions making use of so-called *differential collision trails*. The remainder of this section is dedicated to these trails, how to exploit them to generate internal collisions and the characteristics of these trails that determine the attack complexity: the *backtracking cost* and *backtracking depth*.

2.2 Differential trails

When applying a function f to two different inputs x and x^*, the result is a pair of outputs y and y^* with $y = f(x)$ and $y^* = f(x^*)$. The output difference $y' = y \oplus y^*$ depends on the input difference $x' = x \oplus x^*$ and the absolute values of the inputs. We call a couple (x', y') a differential over f. The differential probability (DP) of (x', y') is the proportion of *input pairs* $\{x, x \oplus x'\}$ such that $f(x) \oplus f(x \oplus x') = y'$. If $\mathrm{DP} > 0$, we say the differential is *possible*.

We denote a differential over the round function R in round i by (t'_i, s'_{i+1}) and call it a *round differential*. The (*restriction*) *weight* of a possible differential, $\mathrm{w}(t'_i, s'_{i+1})$, is defined by

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

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

$$Q : \left((s'_0, p'_0, t'_0), (s'_1, p'_1, t'_1), \ldots, (s'_{\ell-1}, p'_{\ell-1}, t'_{\ell-1}), s'_\ell\right) .$$

A trail describes a propagation of differences 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 $t'_i = s'_i \oplus F_i(p'_i)$. We say this trail *starts in* s'_0 and *ends in* s'_ℓ. Figure 1 schematically depicts a trail.

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

$$\mathrm{w}(Q) = \sum_{i=0}^{\ell-1} \mathrm{w}(t'_i, s'_{i+1}) .$$

If we assume that the conditions imposed by the round differentials are independent, the weight determines the DP of the trail: $\mathrm{DP}(Q) \approx 2^{-\mathrm{w}(Q)}$. For high DP values, this is often

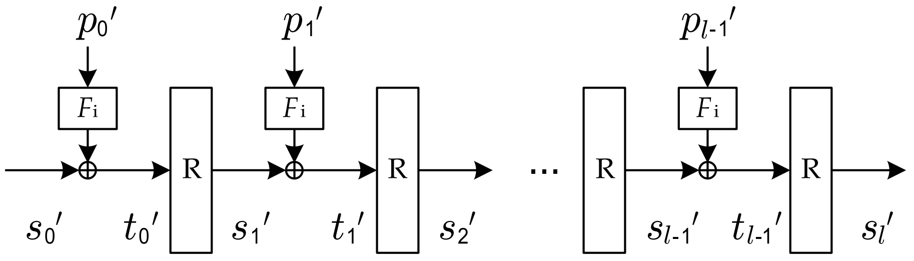

Fig. 1. A differential trail

a good approximation. However, for small DP values or if the round function has specific structural properties, the approximation may become meaningless.

In many round functions the restriction weight $\mathrm{w}(t_i', s_{i+1}')$ equals to the number of Boolean equations that the members of a pair must satisfy at the round input to result in the specified output difference. In order to find an input pair that follows a given trail, the bit values of the members of a pair must satisfy equations at the input of each round determined by the corresponding round differential. To satisfy these conditions, the attacker has degrees of freedom in choosing the absolute values of the input block.

For finding a pair that follows a trail there are two main techniques: statistical and algebraic. A purely statistical attack simply tries many input pairs that exhibit a difference until one is found that follows the trail. A purely algebraic attack considers the set of equations formed by the round differentials and tries to solve them. Combined attacks algebraically generate input pairs that already satisfy a subset of the equations before trying them out statistically.

We call a trail that starts with a zero difference in the state and ends with a zero difference in the state a *collision trail*. In collision trails it holds that $s_0' = s_\ell' = 0$.

As the round function is injective, this implies $t_{\ell-1}' = 0$ and hence $s_{\ell-1}' = F_{\mathrm{i}}(p_{\ell-1}')$.

2.3 A naive statistical attack

A collision trail Q with sufficiently low weight can be used to generate internal collisions. An attacker just applies pairs of inputs that exhibit the difference sequence p_i' specified by the trail Q and verifies whether this results in an internal collision. These pairs look like this:

$$\begin{array}{l} p_{-d}\, p_{1-d} \cdots p_0 \qquad p_1 \qquad \cdots p_{\ell-1} \\ p_{-d}\, p_{1-d} \cdots p_0 \oplus p_0'\ p_1 \oplus p_1' \cdots p_{\ell-1} \oplus p_{\ell-1}'. \end{array} \tag{1}$$

The attacker has to try about $1/\mathrm{DP}(Q)$ pairs in order to find a pair that follows the trail till the end. The required number of pairs to find an internal collision is typically smaller: for the same sequence of input differences $p_0', p_1', \ldots, p_{\ell-1}'$ other collision trails may exist. The workload $\mathcal{L}$ of the attack is determined by the sum of $\mathrm{DP}(Q)$ over all collision trails compatible with a given sequence of input differences $p_0', p_1', \ldots, p_{\ell-1}'$.

2.4 Trail backtracking

Trail backtracking is a technique in which the attacker applies pairs of inputs as specified in Equation (1) and tracks the difference propagation as it proceeds through the rounds. In this section we will consider a round as the application of the round function followed by the injection of an input block. We call a pair *entering* round i a pair with first member $p_{-d} \dots p_i$ and second member differing from the first one by the input difference $0 \dots p'_0 \dots p'_i$ and that has followed the trail up to round i. For such a pair the difference at the input of the round i is t'_i.

Say we have N random pairs of inputs entering round 0. For these pairs, $s'_0 = 0$ and hence $t'_0 = F_{\mathrm{i}}(p'_0)$. For each pair we compute the difference after the round function. If this is equal to s'_1, we say the pair has *survived* round 0. The total number of pairs that are expected to survive round 0 is $N2^{-\mathrm{w}(t'_0, s'_1)}$. For each such 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^r input block values, this results in $N2^{r-\mathrm{w}(t'_0, s'_1)}$ pairs *entering* round 1. Following this reasoning, and assuming the conditions imposed by the round differentials are independent, the number of pairs entering round g is

$$N2^{\sum_{i=0}^{g-1}\left(r - \mathrm{w}(t'_i, s'_{i+1})\right)}.$$

We have an internal collision if the number of pairs surviving round $\ell - 1$ is at least 1. We can use this as a basis for computing the required number of pairs entering each round. This leads us to define the *accumulative excess weight*.

Definition 1. *The accumulative excess weight of a trail Q at round i is defined recursively as:*

- $H_{\ell-1} = 0$,
- $H_i = \max(H_{i+1} + \mathrm{w}(t'_i, s'_{i+1}) - r, 0)$

The accumulative excess weight H_i is the number of conditions of the round differentials from round i to $\ell - 1$ that cannot be satisfied by the input blocks starting from p_i. The expected number of pairs entering round i that are required to have an internal collision is hence given by $\min(2^{H_i - r}, 1)$ and the required number of pairs surviving round $i - 1$ is 2^{H_i}. Note that H_i can never become negative. If it is zero, typically only a single pair surviving round $i - 1$ is needed to find a pair for the complete trail.

We now define the backtracking cost of a trail as the maximum of the accumulative excess weight over all its rounds.

Definition 2. *The backtracking cost of a differential trail Q is*

$$\mathrm{C_b}(Q) = r + \max_{0 \le i < \ell - 1} H_i.$$

The backtracking cost gives an idea of the number of evaluations of R to find an inner collision in a naive statistical backtracking attack based on a trail Q. Typically, the profile of H_i values has a single peak and this number is very close to $2^{\mathrm{C_b}(Q)}$. In the extreme and highly unlikely case of a profile with H_i values that is flat, the number of evaluations can grow up to $(\ell - 1)2^{\mathrm{C_b}(Q)}$.

Table 1 illustrates a weight profile and corresponding backtracking cost.

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 backtracking attack. For a given alternating-input design, 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 or convincing arguments that there are no trails with low backtracking cost.

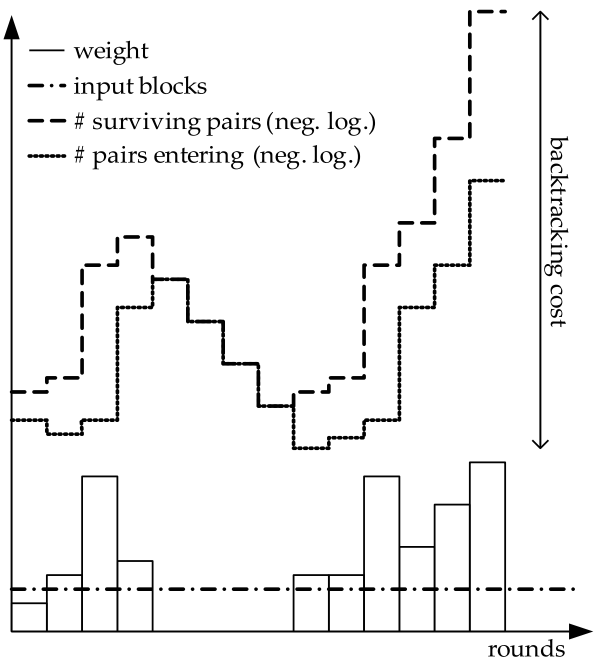

Table 1. Example of trail weight profile and its backtracking cost

2.5 The backtracking depth

One can reduce the workload of a backtracking attack by using algebraic solving techniques. Each round differential imposes a number of conditions on the state at the round input equal to its weight. Some of these conditions can be satisfied algebraically using the input block p_i. The latter must simply be chosen from a subset determined by the intermediate state and the conditions. In general, the ability to choose input pairs that satisfy x binary conditions with certainty, reduces the expected number of pairs to try by a factor 2^x.

In the best case (from the attacker's point of view), r of these equations for S_i can be converted to equations in bits of the input block p_i. For the remaining conditions, the attacker has no choice but to transfer them via the round function to equations in bits of earlier input blocks $p_{i-1}, p_{i-2}, \ldots$. Now the total number of conditions at the input of round i is upper bound by $r + H_i$. As each input block can only satisfy r conditions, some of these $r + H_i$ conditions must be transferred over $x = (r + H_i)/r$ rounds to input block $i - x$. Transferring a condition over the round function complicates its algebraic expression. While the complexity of this strongly depends on the algebraic properties of the round function, it typically increases exponentially with the number of rounds that must be bridged. This leads us to the definition of the *backtracking depth*.

Definition 3. *The backtracking depth* $\mathrm{D_b}(Q)$ *of a trail* Q *is its backtracking cost divided by the bitrate:*

$$\mathrm{D_b}(Q) = \frac{\mathrm{C_b}(Q)}{r}.$$

For a fully algebraic attack, the backtracking depth is an indication of the number of rounds over which condition are transferred. If the backtracking depth is between i and $i + 1$, this number of rounds is i.

In a backtracking attack that combines statistical and algebraic techniques the work factor is determined by the backtracking depth and the number of rounds over which the attacker is able to transfer conditions. If we denote the latter by n_{R}, the expression $2^{r(\mathrm{D_b}-n_{\mathrm{R}})}$ gives a good idea of the number of round function evaluations in the attack is approximately .

2.6 Determining the bitrate *r*

Given a round function, choosing r is a balancing exercise between performance and security. It has a threefold impact on the latter:

– It expresses degrees of freedom for trail construction and decreasing r increases the minimum average weight per round over all collision trails.
– For a given trail weight profile, decreasing r increases the excess weight of round differentials given by $\mathrm{w}(t'_j, s'_{j+1}) - r$ and hence the backtracking cost of the trail.
– For a given trail weight profile and backtracking cost, decreasing r increases the backtracking depth.

Figure 2 illustrates the latter two effects for an imaginary trail. At round 4, there are 13 conditions, r of which can be solved using the available degrees of freedom, hence $H_4 = 13 - r$. At round 3, $9 - r$ condition that cannot be solved are immediately added and so on. The sum of r and the maximum in the profile of the accumulated excess weights is the backtracking cost and the backtracking depth is the cost divided by r. Clearly, decreasing the bitrate has a dramatic effect on the backtracking cost and even more so on the backtracking depth.

round number	weight	H_i			
i	$\mathrm{w}(t'_i, s'_{i+1})$	$r = 8$	$r = 6$	$r = 4$	$r = 2$
0	11	5	**15**	**25**	**35**
1	10	2	10	18	26
2	2	0	6	12	18
3	9	**6**	10	14	18
4	13	5	7	9	11
5	-	0	0	0	0
backtracking cost		14	21	29	37
backtracking depth		1.75	3.5	7.25	19.5

Fig. 2. Example of trail weight profile and backtracking features for different bitrates

3 The belt-and-mill structure

In this section we explain a special case of the alternating-input structure, called *belt-and-mill*. It is the asymmetric structure underlying the round function of RadioGatún and a generalization of that of Panama. The state consists of two parts, the belt and the mill, and the round function treats them very differently.

In this architecture the role of the mill is similar to that of a round function in a SPN block cipher or the F-function in a Feistel block cipher. Its purpose is to 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. Its purpose is to provide global diffusion and it is expected to play an important role in avoiding collision trails with low cost.

A round 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 and the structure is illustrated in Figure 3.

Algorithm 2 A belt-and-mill round function

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

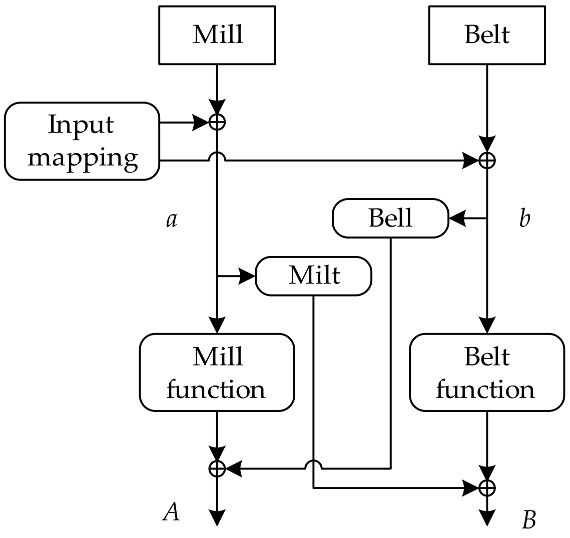

Fig. 3. The belt-and-mill structure

The positions of the bits that are fed forward shall be chosen so that the resulting round function is invertible. We have

$$\textsc{Bell}(b) = \textsc{Bell}\left(\textsc{Belt}^{-1}\left(B + \textsc{Milt}(a)\right)\right) = \textsc{Bell}\left(\textsc{Belt}^{-1}(B)\right) + \textsc{Bell}\left(\textsc{Belt}^{-1}\left(\textsc{Milt}(a)\right)\right)$$

It follows that if $\textsc{Bell}(\textsc{Belt}^{-1}(\textsc{Milt}(a))) = 0$ for any value of a, then $\textsc{Bell}(b)$ can be computed from B. This allows the recovery of a and hence makes the round function invertible.

The only non-linear component in the round is the mill function. This has an important impact on the differential propagation properties of the round function. Using the linearity of the functions, given a difference (a', b') at the input of the round, the output difference (A', B') is:

$$A' = \textsc{Mill}(a) \oplus \textsc{Mill}(a \oplus a') \oplus \textsc{Bell}(b')$$
$$B' = \textsc{Milt}(a') \oplus \textsc{Belt}(b')$$

B' is fully determined by (a', b') and hence independent of the value of the state (a, b). 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:

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

Any differential with $a' = 0$ has weight 0. Moreover, the weight of any round differential cannot be higher than the width of the mill minus 1. So the range of possible weights of round and trail differentials is fully determined by the mill size and independent of the belt.

As for algebraic attacks, only the mill function results in nonlinear equations and all other equations are linear. When considering multiple rounds one only needs to introduce a number of internal variables equal to the width of the mill for keeping the equations of the same algebraic degree as the round function.

4 The breakdown of the Panama hash function

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 ($\textsc{Milt}(a) = 0$), leading to a linear dependence of all belt bits from the input sequence. In this section we concentrate on generating inner collisions during the input injection phase and only consider the push mode.

4.1 Summary of Panama

Panama was designed by Joan Daemen and Craig Clapp in 1998 [26] as a variant of StepRightUp that appeared in [24]. We refer to the original papers for the full specification and give here a short description focusing on the aspects that are relevant for the origins of Keccak.

The belt and mill are arrays of 32-bit words:

- the *mill* (originally called state): 17 words of 32 bits (a_i, $i = 0 \ldots 16$)
- the *belt* (originally called buffer): 8×32 words of 32 bits

The belt function $\textsc{Bell}(b)$ is a simple linear feedback register operating on 8-word stages, the mill function $\textsc{Mill}(A)$ an invertible non-linear function with high local diffusion and consisting of the sequence of 4 steps $\iota \circ \theta \circ \pi \circ \gamma$. These steps operate in parallel on the bits of the words. Indexing should be taken modulo 17, $+$ denotes bitwise addition in GF(2) and $\ggg$ cyclic shift.

- $\gamma : a_i \leftarrow a_i + (a_{i+1} + 1)a_{i+2} + 1$ for non-linearity;
- $\pi : a_i \leftarrow a_{7i} \ggg i(i+1)/2$ for inter- and intra-word dispersion;
- $\theta : a_i \leftarrow a_i + a_{i+1} + a_{i+4}$ for bit mixing;
- $\iota : a_0 \leftarrow a_0 + 1||0^{31}$ for asymmetry;

The bell function $\textsc{Bell}(b)$ simply takes an 8-word stage of the belt and adds it to the last 8 words of the mill. The milt function $\textsc{Milt}(a)$ is empty (in the push mode). In between the rounds, 8 words of input are added into the first stage of the belt and into 8 words of the mill. All additions here are bitwise additions. Figure 4 illustrates the structure of Panama in the push mode.

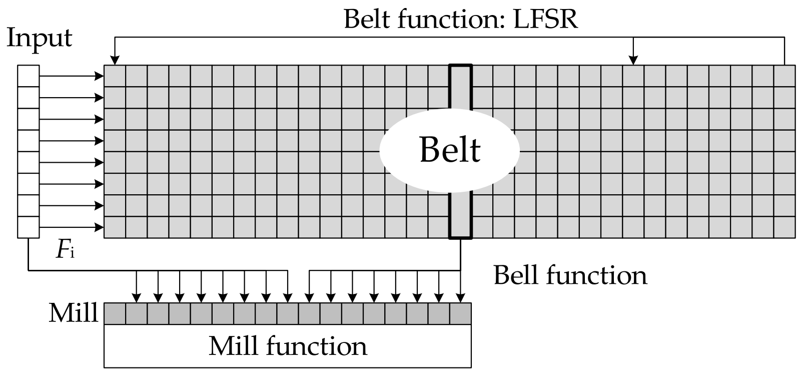

Fig. 4. The structure of Panama

4.2 The collision attacks

In Panama, the role of the belt is to ensure that internal collisions can only occur for input pairs with a difference that satisfies certain criteria. As in the push configuration the belt evolves independently from the mill, an internal collision implies that there must be a collision in the belt.

We illustrate the difference propagation in the Panama belt in Figure 5. Each square denotes a 256-bit unit (8 words of 32 bits) and one horizontal line denotes the differences in input and state. The simplest input difference pattern that one can apply that leads to a collision in the belt is indicated by the black squares in the input vector: a non-zero difference is injected in three time instances spanning 33 rounds. The 256-bit difference in the first time instance fully determines the two other ones. The simplest such pattern has a single active bit in each of the three differences. Any input difference sequence that leads to a collision in the belt will be the superposition of translated versions of this pattern. This can be seen as the convolution of an *atomic input sequence* and the simple pattern.

The propagation of input differences through the belt is illustrated in Figure 5. Clearly, as can be seen in this figure, even the simplest input difference injects differences in the mill in five time instances: three times when it appears at the input and two times via the bell function when it appears in the stage of the belt indicated by the bold frame in the figure. Input differences based on more complex atomic input sequences result in even more difference injections into the mill. At first sight the fact that any collision trail spans at least 33 rounds implies it is difficult to find collisions. In 2001, Vincent Rijmen et al. demonstrated that this is not the case. They produced a collision trail and a procedure to use it for finding colliding input pairs with a workload below that in our claimed security level[42]. In 2007, we refined this attack in [25] to generate collisions instantaneously using some simple equation-solving techniques.

The attack in [42] makes use of a very simply atomic input difference sequence spanning three rounds. This results in 5 separate input difference sequences in the mill. They used an atomic input difference sequence such that for each of these input difference sequences, there exists a 3-round mill collision trail. The atomic input difference sequence consists of three subsequent input differences, where the middle one is always zero. At word-level the difference patterns are either all-1 or all-0. We call such patterns *1-symmetric*. The resulting input difference corresponds to the black and dark grey squares in Figure 5. The attack of [42] can be seen as a form of trail backtracking and the presented collision trail has backtracking cost $14 \times 32 = 448$ and backtracking depth 1.75.

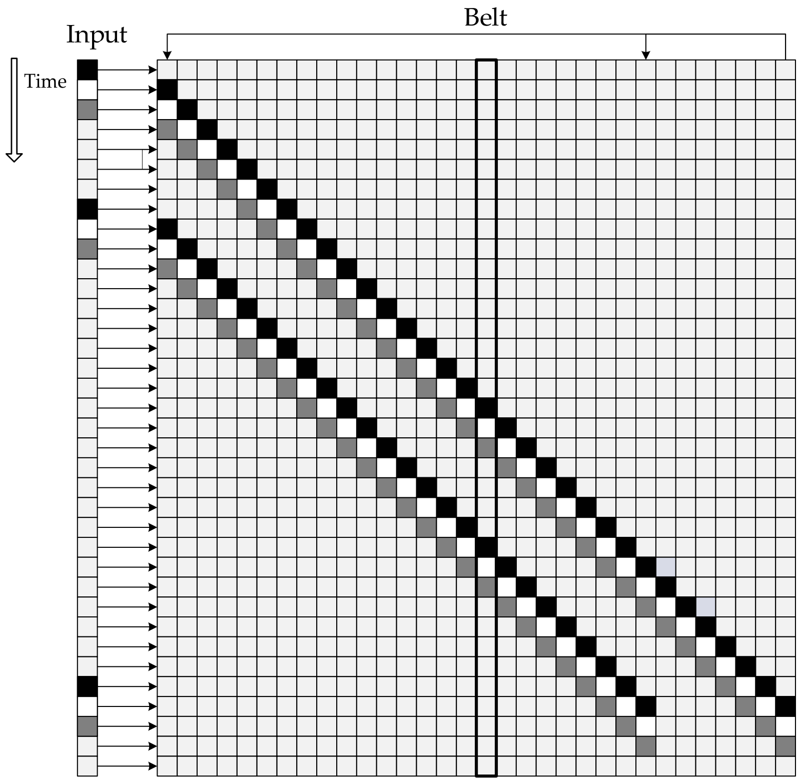

Fig. 5. Collision trails in the Panama belt

As said, only the differences in the mill contribute to the trail weight profile and hence the backtracking cost and depth. The large bitrate allows the attacker to address the mill collision trails separately. The input difference sequence is chosen in advance and each of the five mill collision trail can be satisfied using the input blocks just preceding them. This allows solving for the mill collision trails serially.

For each of the mill collision trails, the authors of [42] noticed that some of conditions can be easily converted to conditions on input bits and propose to satisfy the remaining conditions by trying many input pairs. The complexity of their attack is 2^n evaluations of the Panama round function, with n the backtracking cost minus the number of conditions that can be satisfied by choosing input blocks with the correct values. They pushed their attack just far enough until they reached a value of n that breaks the security claim of Panama.

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

– For the atomic input difference sequence in [42] they only considered sequences of the form $dx, 0, dy$. In fact it is likely that there are differences of type dx, dy, dz for which there are collision trails in the mill in all 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. The resulting input difference corresponds to the black, white and dark grey squares in Figure 5.
– 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.

– More effort can be put in solving 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.

After we had abandoned attempting to fix Panama and had proposed RadioGatún, we tried out these ideas aiming at a publishable attack. We indeed found a better atomic difference sequence that allowed us to satisfy all equations with the input blocks injected just before the mill collision trails. This resulted in the generation of collisions with an effort of 2^6 calls to the Panama round function. The effort to generate a collision is actually smaller than that of hashing the colliding messages. The trail we produced has backtracking cost $13 \times 32 = 416$ and backtracking depth 1.625. We reported on this work in [25], including an example collision.

4.3 Attempts at fixing Panama

An initial idea to strengthen Panama against this kind of attacks was to modify the belt LFSR so that the number of times a difference is injected in the state grows from 5 to a larger number. This imposes more restrictions on the atomic difference sequence as for each of the difference injections in the mill, there must exist a mill collision trail. It is then likely that the weight per round goes up and with it the backtracking cost. However, we soon realized that there is a generic collision attack that works for any efficient belt-and-mill structure that has no feedforward from mill to belt.

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

– Apply two partial messages to the function, leading to two values of the state.
– Consider the difference in the belt. Compute an input difference sequence that will set the difference in the belt to zero. This can be done by solving a set of linear equations. The typical required length of this input sequence is the size of the belt divided by the bitrate.
– Pick a pair of trailing parts that have the correct difference and perform a number of rounds for injecting them, minus the last z rounds, with z is the size of mill divided by the bitrate, multiplied with a small factor (e.g. 1.5).
– Consider the mill at this stage. Try to find from the current difference in the mill a differential trail leading to zero. This trailfinding seems not so difficult if z is small because the difference injected into the mill in the last round is known and the input difference of the current round is known. A meet-in-the-middle search can be conducted here.
– Try now to convert the conditions on the bits of the mill imposed by the mill round differentials to bits of the input. This involves algebraic computations that become more difficult with the number of rounds have to be bridged, i.e., with z. Converting conditions in the mill in round t to conditions in the input in round $t - m$ becomes more difficult as m grows. If z is small, we expect the weight of the trail to be a small factor times the size of the mill.

In Panama the mill has 17 words and the input 8 words, so we can take $z = 3$. To find an internal collision in the mill a at time ℓ, it is sufficient to express some conditions on bits of $a^{\ell-1}$, $a^{\ell-2}$ and $a^{\ell-3}$ to conditions on $p^{\ell-2}$, $p^{\ell-3}$ and $p^{\ell-4}$. In short, for a function to be resistant against this attack, either it shall be infeasible to find a differential trail that leads to an internal collision under a fixed input difference sequence for z rounds, or it shall be infeasible to find an input pair that follows such a trail.

For small z and low-degree round functions such as the one in Panama this does not inspire much confidence. The only method to make such a function resistant against such attacks is increasing the ratio between the mill size and bitrate to a large value, e.g., 8 or more. However, the speed of the function is proportional to the bitrate divided by the mill size, as the mill function takes the largest portion of the computational resources. From this we concluded that having no feedforward from mill to belt would never lead to an efficient hash function.

5 The research cipher RadioGatún

Based on the insights we gained from investigating Panama, we set out in late 2005 to design an improved version. We added a mill-to-belt feedforward, reduced the size of the belt and the bitrate, increased the size of the mill by two words, tweaked the feedforward from belt to mill and called the result RadioGatún. We presented it at the NIST Hash Workshop in 2006 [4].

5.1 Summary of RadioGatún

In RadioGatún, the belt and mill are arrays of words of size w, a parameter ranging from 64 (the default) down to 1. This results in 64 different functions, where the version with word size w is denoted as RadioGatún$[w]$.

– the *mill* is composed of 19 words of w bits each, and
– the *belt* is composed of 3×13 words.

The belt function Belt(b) is a cyclic register operating on 3-word stages, the mill function Mill(A) a function similar to the one in Panama, but with indices taken modulo 19 instead of 17.

– $\gamma : a_i \leftarrow a_i + (a_{i+1} + 1)a_{i+2} + 1$ for non-linearity;
– $\pi : a_i \leftarrow a_{7i} \ggg i(i+1)/2$ for inter- and intra-word dispersion;
– $\theta : a_i \leftarrow a_i + a_{i+1} + a_{i+4}$ for bit mixing;
– $\iota : a_0 \leftarrow a_0 + 1||0^{w-1}$ for asymmetry;

We can define a partition of the bits in the belt that is orthogonal to the partition in words. We call the elements of this partition *slices*. A slice is a subset of statebits with the same position within the words. Clearly, the belt of RadioGatún$[w]$ has w slices. The steps γ and θ operate on the slices independently.

The bell function Bell(b) takes the last 3-word stage of the belt and adds them into 3 words of the mill. The milt function Milt(a) takes 12 words of the mill and injects them in 12 words of the belt. In between the rounds, 3 words of input are added into the first stage of the belt and into 3 words of the mill. An output block z consists of 2 words $z[i]$. All additions are bitwise additions. Figure 6 illustrates the structure of RadioGatún.

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.

The modifications with respect to Panama result in a smaller memory footprint thanks to the reduction of the belt size. The security level has been increased at the cost of performance by introducing mill-to-belt feedforward and decreasing the ratio between bitrate and mill size. Note that this is expected to result in a strong increase in security as it has a threefold impact on the minimum backtracking depth (see Section 2.6).

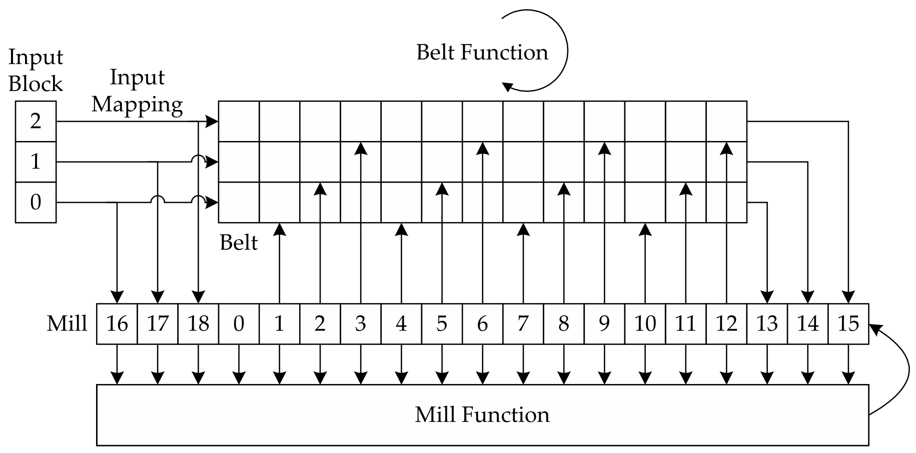

Fig. 6. The structure of RadioGatún

5.2 Investigations of RadioGatún

We tried to get an idea of the security margin by investigating RadioGatún[1], the version with single-bit words. Thanks to its smalls state size of only $19 + 3 \times 13 = 58$ bits, it is feasible to find internal collisions using the birthday paradox. A set of about $2^{58/2} = 2^{29}$ random inputs likely contains a pair that cause an internal collision.

We looked for internal collisions by using inputs that could result in colliding pairs as in Equation (1) with $d = 7$ and $\ell = 7$. The input block p_{-7} is the first block of the input. There are thus seven input blocks of preparation, that determine the absolute value of the state before the differences can appear and that all the elements have in common. Eight blocks are then input to RadioGatún[1], that are chosen randomly for each input. In our experiment we found more than 4 million internal collisions. The minimum backtracking cost found was $C_b = 46$ and there were two trails with this cost. Then, a few internal collisions had cost $C_b = 51$.

We were not confident in the safety margin of RadioGatún and wanted to investigate versions with larger word sizes. However, we soon realized that even for 2-bit words randomly generating collisions was out of reach because the state size grew to 116 bits.

6 The road to Keccak

Immediately after our publication of RadioGatún in August 2006, we decided to create a test bench for different configurations to gain more insight in the role of the belt and the relation between the mill, belt and input sizes on the security margin. We called this test bench Gnoblio. The mill of Gnoblio is composed of 11 words (instead of 17 or 19), allowing the investigation of larger word lengths. By default, the round function operating on the mill is the same as in Panama or RadioGatún, except for its size. For the belt and feedforward we tried different configurations. The rate is 1 or 2 words, allowing us to bracket the input to mill size ratio of RadioGatún: $\frac{1}{11} < \frac{3}{19} < \frac{2}{11}$.

In our investigations with Gnoblio we gained two insights that caused us to reconsider the design approach for our SHA-3 candidate. These are related to the role of the belt and stream-versus-block considerations and made us switch from an alternating-input belt-and-mill approach to a block mode calling a monolithic permutation. We started considering this option in the last months of 2007 and made the final decision at the Early

Symmetric Crypto Seminar (ESC) in Echternach in January 2008. In the course of 2008 and 2009 a number of papers were published with (third-party) cryptanalysis of RadioGatún. As discussed at the end of this section, most of these papers confirmed to us that we had taken the right decision. We reported on our insights gained with Gnoblio at the Dagstuhl seminar in January 2009 [8].

6.1 The role of the belt

In Panama, the role of the belt is to guarantee a minimal length for collision trails: any collision trail is at least 33-round long, due to the linearity of the belt updating function and the fact that it evolves independently of the mill.

In RadioGatún, the role of the belt is to provide long-term diffusion. A difference in the mill can propagate to the belt and then come back later in the mill through the belt-to-mill feedback. Due to the latter, the impact of the belt on differential trails becomes more subtle. To illustrate this, consider a differential trail as in Table 2 with differences p'_i in the message blocks, a'_i in the mill and b'_i in the belt. The belt evolves linearly and so do the differences. Hence, the difference in the belt b'_i at round i is a linear function of the message difference and on the differences at round $i-1$: $b'_i = \lambda(a'_{i-1}, b'_{i-1}, p'_i)$.

i	Δ input	Δ mill	Δ belt
0	p'_0	a'_0	$b'_0 = 0$
1	p'_1	a'_1	$b'_1 = \lambda(a'_0, b'_0, p'_1)$
…			
$\ell-1$	$p'_{\ell-1}$	$a'_{\ell-1}$	$b'_{\ell-1} = \lambda(a'_{\ell-2}, b'_{\ell-2}, p'_{\ell-1})$
ℓ	p'_ℓ	$a'_\ell = 0$	$b'_\ell = \lambda(a'_{\ell-1}, b'_{\ell-1}, p'_\ell) = 0$

Table 2. General structure of a differential trail in a belt and mill hash function

A collision trail ends with a zero difference in the state: $a'_\ell = 0$ and $b'_\ell = 0$. The latter results in v Boolean equations. These conditions can be replaced by linear conditions in bits of $a'_{\ell-1}$, $b'_{\ell-1}$ and p'_ℓ. In these equations one can eliminate the bits of $b'_{\ell-1}$ by linear functions of message block differences and mill differences. By applying this iteratively, all belt difference bits can be eliminated to end up in v linear Boolean equations in input difference bits and mill difference bits.

In other words, the trail seen as a sequence of p'_i and a'_i must be compatible with the belt-and-mill structure. This is expected to increase the minimum length for collision trails, as a trail must be long enough to satisfy the v conditions. The impact of this effect proportionally decreases when we consider longer trails. The number of conditions v is fixed and the number of conditions per round in an ℓ-round trail is only v/ℓ.

While we expected the belt to bring added value at reasonable cost, it now turned out that the effect of the belt on differential trails does not scale well with the trail length. At first sight it may seem that the absence of short collision trails will have a positive effect on the minimum backtracking depth. However, this was not confirmed by experiments we did with Gnoblio. We did a tree search for collision trails with truncation criterion the backtracking depth of the partial trail. We observed that when this parameter is above a certain threshold value, arbitrarily many collision trails of arbitrary (high) length can be generated with the given backtracking depth. For instance, with 4-bit words, an 11-word mill, and with a two-word rate this threshold backtracking depth is only 1.125. The found trails have rather flat weight profiles where all round differentials have a weight near the number of input bits. One can see this as a kind of *steady-state* statistical process.

So our experiments suggested that for interesting rate/mill-size enforcing a minimum length in collision trails does not imply high backtracking depth. Moreover, a large belt implies a large state, a disadvantage both in lightweight implementations and on high-end CPUs when using registers. For these reasons we decided to abandon having a belt altogether and rather increased the number of words in the mill.

6.2 From stream to blocks

In Panama and RadioGatún, with their alternating-input structure, the message block insertions alternate with the rounds. We wondered what would be the effect on the security margin if instead we would insert two message blocks in the even-numbered rounds and none in the other ones, and then further generalize this.

The number of bits inserted has an impact on generic attacks. For a constant state size, increasing the input size means that the part of the state that the attacker cannot directly control shrinks.

For specific attacks we consider the effect on the trail backtracking cost. For simplicity, we compare the input of r bits every round with the input of $2r$ bits every two rounds and we assume the same trail weight profile. We are aware that the difference in message insertion will undoubtedly lead to optimal trails being different for the two cases, but the average degrees of freedom per round are the same and hence we find it reasonable to assume that the best trails have similar weight profiles. We consider in both cases the values of H_{2j}, H_{2j+1} and the maximum of the two.

In the first case, we get

$$\begin{aligned} H_{2j} &= \max(H_{2j+1} + W_{2j} - r, 0) \\ H_{2j+1} &= \max(H_{2j+2} + W_{2j+1} - r, 0) \\ &\to \max(H_{2j+2} + W_{2j+1} + W_{2j} - 2r, W_{2j} - r, 0), \end{aligned}$$

while in the second case

$$\begin{aligned} H_{2j} &= \max(H_{2j+1} + W_{2j}, 0) \\ H_{2j+1} &= \max(H_{2j+2} + W_{2j+1} - 2r, 0) \\ &\to \max(H_{2j+2} + W_{2j+1} + W_{2j} - 2r, W_{2j}, 0). \end{aligned}$$

It is clear that, for the same weight profile, lumping the insertion of message blocks cannot decrease the backtracking cost. This was experimentally verified on Gnoblio.

If we extend this reasoning to a larger number of rounds, we tend to a *block mode*, where (relatively) large message blocks are inserted between the application of a sequences of a (relatively) large number of rounds. The trail backtracking cost now gives a different picture. Consider a sequence of ℓ rounds between two message block insertions. Then $H_{\ell-1} = 0$ and $H_i = \max(H_{i+1} + W_i, 0) = H_{i+1} + W_i$, and the trail backtracking cost becomes $C_b = \sum_i W_i$, where the sum of the round differential weights in the trail becomes the determining quantity.

Compared to alternating-input, the block mode has the following advantages:

- The weight of a collision trail over a single block simply consists of the sum of the weights of the round differentials

– The attacker can control part of the state only every ℓ rounds and hence conditions must be transferred over more rounds.

A drawback is that trail clustering must be taken into account, as different trails can contribute to the same differential.

Based on our analysis, we decided to abandon the alternating-input structure and go for a blocked approach.

6.3 Third-party cryptanalysis of RadioGatún

In the meanwhile RadioGatún had caught the attention of the cryptographic community and several papers were published in the course of 2008 and 2009. While the security claim of RadioGatún was not invalidated, most of these attacks confirmed to us we had taken the right decision by abandoning alternating-input and belt-and-mill and go for a block approach without a belt.

In an article presented at SAC 2008 Charles Bouillaguet and Pierre-Alain Fouque reported on some interesting experiments with RadioGatún[1] [19]. They showed how to propagate conditions originating from a differential trail to a single round using Gröbner bases. This turned out to be feasible thanks to the fact that the RadioGatún round function is quadratic and new (linear) conditions are introduced each round.

In their paper they also showed that the round differentials in the collision trails with smallest backtracking cost found are not independent. Although we did not think these results formed a threat for RadioGatún, they did confirm our decision to change design approach. Actually we believe the observed effects are an artefact from the method by which the trails were generated: by finding actual collisions and deriving the trails from them.

In a paper presented at Asiacrypt 2008, Michael Gorski, Stefan Lucks and Thomas Peyrin performed slide attacks on Grindahl and variants of RadioGatún with the padding rule changed [29]. We considered this paper an interesting exercise but it did not force us to adapt our design strategy as the analyzed variants of RadioGatún lacked the elements we put there to protect against this kind of attack.

At FSE 2009 Thomas Fuhr and Thomas Peyrin presented how to construct collision trails using a meet-in-the-middle method [28]. They performed a tree search where a variant of the backtracking cost is used to truncate efficiently. They found a collision trail leading to an attack complexity of 2^{11w} and confirmed this number with a concrete collision for $w = 2$. The used collision trail has length 143 rounds, backtracking cost 11 and backtracking depth 3.67. Remarkably, this collision trail is very long while it has a relatively low backtracking depth. This confirmed what we observed in our experiments with Gnoblio (see Section 6.1). The authors concluded that even for larger RadioGatún instances collision trails can be found that would have an attack complexity below $2^{9.5w}$ and hence invalidate the RadioGatún security claim. We agree with this conclusion and at the time of publication, this paper confirmed to us that for our SHA-3 candidate we had taken the right decision to abandon streaming-based hashing.

In papers presented at Indocrypt 2008 and SAC 2009, Dmitry Khovratovich proposes generalizations of truncated differential trails, also named *structures* for improving collision-generating attacks on RadioGatún (and also Grindahl) [32,33]. These attacks also confirmed to us we had taken the right decision by abandoning alternating-input and belt-and-mill and go for a block approach without a belt.

7 The sponge construction

In this section we explain that the sponge construction originally was just a theoretical concept for expressing security claims and thanks to our proof in the indifferentiability framework it became suited for use in actual cryptographic functions.

7.1 The problem of the security claim

When designing and proposing a cryptographic primitive, it is important to understand and state which security criteria it satisfies. For cryptographic hash functions, the established security criteria are collision resistance, pre-image resistance and 2nd pre-image resistance [38]. Often no explicit claims are made and the hash function is supposed to offer a security level implied by the length of its digest. Unfortunately, these criteria do not fully cover what we have come to expect of a cryptographic hash function. Sometimes hash functions are applied to destroy some mathematic properties of a public key primitive such as RSA [35]. In use cases such as straightforward use for computing message authentication codes (MAC), resistance against length-extension [43] is a requirement. In the years preceding our work on RadioGatún, a series of attacks [30,31,23,34] had shown that certain hash function constructions do not offer as much security as expected, leading to the introduction of yet other criteria, such as *chosen target forced prefix preimage resistance* [34]. As was already predicted in [1], there is no reason to assume that no new criteria will appear, so the design of a hash function seemed like a moving target.

Remarkably, a random oracle [3] is a theoretical construction that satisfies all known security criteria for hash functions and it seems hard to imagine that new security criteria will be introduced that a random oracle does not satisfy. So a plausible solution to this problem seems to replace all security criteria by a single one: *a good hash function behaves as a random oracle*. There are however a number of problems with this approach.

Informally speaking, a random oracle maps a variable-length input message to an infinite output string. It is completely random, i.e., the produced bits are uniformly and independently distributed. The only constraint is that identical input messages produce identical outputs. A hash function produces only a fixed number of output bits, say, n bits. So, a hash function should behave as a random oracle whose output is truncated to n bits. In general, it is easy to compute the resistance of a random oracle (truncated to n bits) to certain attacks. For instance, the expected number of calls to the oracle to generate a collision is of the order of $2^{n/2}$. To find a (second) pre-image, this number is 2^n. The hash function is then considered broken if someone finds an attack on the hash function with a complexity smaller than for a random oracle.

When we published RadioGatún, we wanted to have a compact security claim. However, it was clear that we could not make a truncated random oracle claim for RadioGatún due to its arbitrary output length. By just increasing the output length, it would be possible to increase the resistance against collisions and (second) preimages indefinitely. In fact, the finite size of the state in RadioGatún puts an upper limit to the security that can be achieved. RadioGatún shares this property with most practical hash functions that iteratively update a chaining value of fixed size with a function taking a message block as an argument.

Iterated hash functions have *state collisions*, that is, collisions in the chaining value. The existence of state collisions results in properties that do not exist for random oracles. For instance, assume that M_1 and M_2 are two messages that form a state collision in an iterated hash function. Then, for any suffix N, the messages $M_1|N$ and $M_2|N$ will produce identical hash values. A random oracle does not have this property: even if M_1 and M_2

produce the same hash value (of finite length n), $M_1|N$ and $M_2|N$ produce hash values that are independent of the hash value obtained from M_1 and M_2.

In the light of state collisions, the claimed reference model cannot be a random oracle as it is an unreachable goal. There are two ways to address this problem. First, one can abandon iterated hash functions and use non-streamable hash functions such as the zipper hash construction [36]. This is however unsuitable for many applications of hash functions since the entire message must be available in memory. A second approach is to stick to iterated hash function constructions and learn to live with state collisions. This is the approach we followed by trying to define a reference model that is as close as possible to a random oracle, but exhibits state collisions.

Our first attempt to such a reference model appeared in [4] by the name of *random iterative mangling functions (IMF)*. Later we decided to spend some time to refine this reference model and soon this secondary thread became a research subject of independent interest. The random IMF went through some modifications and lead to *random sponges*. We presented our random sponge concept first at the Dagstuhl seminar in January 2007. Later we wrote our findings in a paper [5] that we presented at the Ecrypt hash workshop in Barcelona in May 2007.

7.2 Specification of the sponge construction

The sponge construction [5] builds a function $\textsc{sponge}[f, \text{pad}, r]$ with variable-length input and arbitrary output length using a fixed-length permutation (or transformation) f, a padding rule "pad" and a parameter *bitrate* r.

The permutation f operates on a fixed number of bits, the *width* b. The sponge construction has a state of b bits. First, all the bits of the state are initialized to zero. The input message is padded with the function $\text{pad}[r]$ and cut into r-bits blocks. Then it proceeds in two phases: the *absorbing phase* followed by the *squeezing phase*:

Absorbing phase The r-bit input message blocks are XORed into the first r bits of the state, interleaved with applications of the function f. When all message blocks are processed, the sponge construction switches to the squeezing phase.

Squeezing phase The first r bits of the state are returned as output blocks, interleaved with applications of the function f. The number of iterations is determined by the requested number of bits.

Finally the output is truncated to the requested length. The sponge construction is illustrated in Figure 7.

The value $c = b - r$ is called the *capacity*. The last c bits of the state are called the *inner part* and its first r bits are called the *outer part*. The bits of the inner part are never directly affected by the input blocks and are never output during the squeezing phase. The capacity c actually determines the attainable security level of the construction [7,14].

7.3 Provable security reductions and generic security

In [5] we proved that the success probability of distinguishing a sponge calling a random permutation (or transformation) f from a random oracle is upper bound by $N^2 2^{-(c+1)}$ with N the number of calls to f. This is basically the success probability of generating state collisions.

Our first application of the sponge construction was the reformulation of the security claim for RadioGatún on the RadioGatún site in the summer of 2007. We defined a simplified security claim, called a *flat sponge claim* where the security level is summarized in

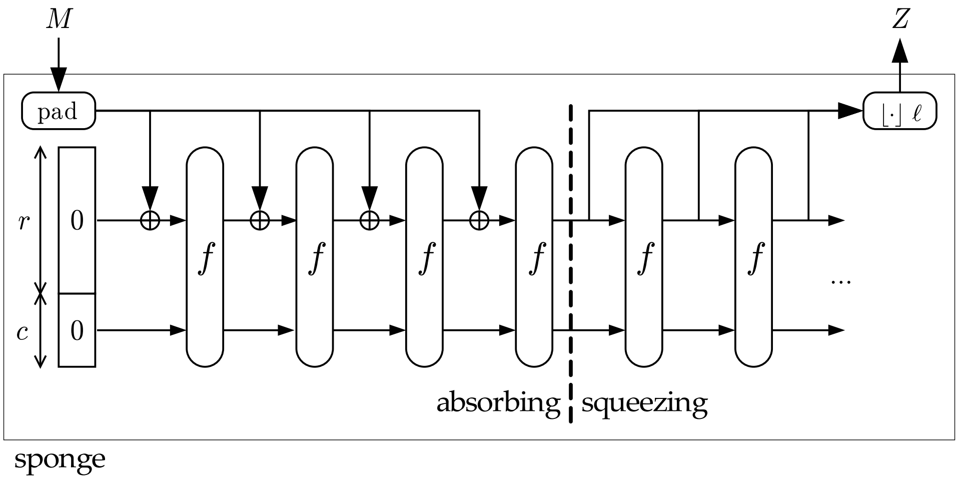

Fig. 7. The sponge construction

a single parameter: the capacity. We made a flat sponge claim for RadioGatún$[w]$ with a capacity $c_{\text{claim}} = 19w$. For the 64-bit version RadioGatún this results in an excessive capacity of 1216 bits, for the 32-bit version this gives 608. Clearly, we were putting the bar very high.

A limitation of our proof in [5] is that the adversary is not given direct access to f. Initially this was no problem for us as we considered sponge functions just as security references. In our struggle with RadioGatún and Gnoblio we had started to consider the possibility of using the sponge construction in an actual design and so it would be nice to have security bounds against generic attacks. Unfortunately, our proof could not provide that for the simple reason that in any concrete function f would be publically specified and we should have a proof against an adversary having direct access to f (and its inverse).

We learned that such proofs can be made in the indifferentiability framework [37] and it had effectively been applied to iterated hash functions in [23]. We applied this to the sponge construction and presented our result at Eurocrypt 2008 [7]. In this paper we proved that any generic attack on a sponge function has success probability of at most $N^2 2^{-(c+1)}$ plus the success probability of this attack on a random oracle. This opened up the sponge construction as a tool for design, with a tight security bound against generic attacks.

7.4 The hermetic sponge strategy

In RadioGatún there is a gap between the size of its inner state and the capacity c_{claim} used in its security claim. This accounts for the fact that the round function is not designed to be a strong cryptographic primitive by itself and thus reflects the acceptance that it does not prevent attacks that are better than generic ones. The design philosophy of RadioGatún is to avoid internal collisions in the absorbing phase and to decorrelate the output from the input using blank rounds.

When we started working on what would become Keccak, we adopted a very different design philosophy that we called the *hermetic sponge strategy* [15]. Its main principle is to instantiate a sponge function with a permutation that should be so strong that it does not allow attacks better than generic ones. This design strategy tolerates no gap between the claimed security level and the capacity of the sponge construction used, namely, $c_{\text{sponge}} = c_{\text{claim}}$.

Designing and analyzing such a strong permutation is very much like design and analysis of a block cipher in a known-key setting: differential and linear trail propagation, algebraic analysis, clustering of trails and truncated attacks, etc.

There are however two important differences with a block cipher. The first difference is that it has no key schedule, so there is no need to design one. There needs to be some asymmetry between the rounds to avoid slide attacks [18]. This can be addressed by including the addition of round constants that differ from round to round to the state. These constants may also provide asymmetry to the round function to avoid symmetric properties (see Section 8). The second difference is that the inverse of the round function does not have to be efficient as in the sponge construction the inverse of f is never called. This gives more degrees of freedom to the designer and may allow achieving the same security level in a more efficient way.

8 Keccak

After deciding to adopt the hermetic sponge strategy, we had to design a strong permutation. From the beginning, it was clear this would have an iterated structure, repeating a simple round function diversified by round constants for asymmetry. The starting point for designing this round function were the belt updating functions of Panama, RadioGatún and Gnoblio. We wanted to keep the high amount of symmetry, the general structure of having separate steps for the non-linearity, mixing and bit transposition and the scaleability by means of variable-length words. In the first six months of 2008 we converged reasonably quickly to what would become Keccak-f: the permutations at the core of Keccak. In this section we report on this process.

8.1 Increasing the diffusion

Our first attention point was the mixing layer θ. In differential trail propagation experiments with Gnoblio we had noticed that the diffusion was less than we would like.

We set out to modify θ to have a better average diffusion, while keeping an eye on implementation cost. In RadioGatún and its predecessor, the mixing layer θ is basically a linear *convolutional* mapping with periodic boundary conditions and its operation on a single-bit slice of the words can be modeled as multiplication by a polynomial $p(x)$ modulo $x^n + 1$ [24]. Here the state value is interpreted as a binary polynomial and n equals the number of words. In RadioGatún, $p(x)$ has three terms and the number of operations per bit is 2 bitwise additions. A single-bit difference propagates to 3 bits. The worst-case diffusion is expressed by the (bit-level) branch number [24] and is 4. Increasing the diffusion can be done by adopting a polynomial with more terms. By taking a polynomial $p(x)$ with w terms a single-bit difference propagates to w bits and by a judicious choice a branch number equal to $w + 1$ can potentially be achieved. The downside is that this increases the cost from 2 bitwise additions per bit to roughly $w - 1$ bitwise additions per bit.

The propagation from single-bit differences can be greatly increased at almost no cost by the following trick. Let the number of words n be a product of two factors: $n = n_1 n_2$ and a multiplication polynomial $p(x)$ of the following form:

$$p(x) = 1 + \sum_{0 \le j < n_2} x^{1 + j n_1}.$$

This polynomial has $n_2 + 1$ terms so a single-bit difference propagates to $n_2 + 1$ bits. At first sight, the implementation cost is $n_2 + 1$ bitwise adddditions per bit.

We can express the new value of bit in position i as:

$$b_i = a_i + \sum_{0 \leq j < n_2} a_{i+1+jn_1} = a_i + c_{i+1}(a) \text{ with } c_i(a) = \sum_{0 \leq j < n_2} a_{i+jn_1}. \qquad (2)$$

Clearly $c_i(a) = c_{i+jn_1}$ for all j. This suggests a two-stage implementation approach:

- Compute $c_i(a)$ for all $i : 0 \leq i < n_2$. This additional stage takes $n_1(n_2 - 1)$ binary additions
- Compute $b_i = a_i + c_{i \bmod n_2}$. This takes n binary additions

It follows that the implementation cost of this mapping is only $2 - 2/n_1$ binary additions per state bit. The downside is that the branch number is only 4: a two-bit difference with bits in positions rn_1 apart will go through this mapping unaffected. This is because all terms $c_i(a)$ will be zero.

The propagation from single-bit differences can practically be doubled at very little additional cost by taking for $p(x)$:

$$p(x) = 1 + \sum_{0 \leq j < n_2} x^{1+jn_1} + \sum_{0 \leq j < n_2} x^{-1+jn_1}.$$

or equivalently:

$$b_i = a_i + c_{i-1}(a) + c_{i+1}(a) = a_i + d_i(a) \text{ with } d_i = c_{i-1}(a) + c_{i+1}(a).$$

A three-stage implementation of this can be realized by adding a new stage computing the terms d_i for all $i : 0 \leq i < n_2$. This takes in total n_2 binary additions or $1/n_1$ per state bit, resulting in a total cost of 2 binary additions per state bit. With respect to the θ mapping in RadioGatún the branch number has not been increased but a single bit difference propagates to $2n_1 + 1$ bits instead of just 3.

8.2 From a two-dimensional to a three-dimensional structure

The nonlinear step γ in the RadioGatún round function is also a convolutional mapping with periodic boundary conditions. We replaced it by its complement, that was already studied in [24] and named χ. In our first attempts at the round function, it operated on the n-bit array of words. We did not change the definition of the intra-word rotations of ρ. However, the generalization of the word transposition of ρ of type $b_i = a_{gi \bmod n}$ (with g coprime to n) posed a problem: it leads to trails of arbitrary length with a low weight per round. We will now show how.

In this discussion a specific symmetry property plays an important role. In fact, all mappings in the round function of Panama, RadioGatún and Gnoblio (except the round constant addition ι) commute with the operation of cyclically shifting all words of the state over the same offset. The result of this is that propagation structures that are not affected by the addition of round constants for a given word length w can be used to construct propagation structures for word lengths that are a multiple of w by mere duplication. This is still the case for the final version of Keccak and we call this the Matryoshka structure. We refer the interested reader to [15] for a more in-depth discussion.

For the discussion in this section we use the Matryoshka structure to extend differential trails in the single-bit word version to trails in versions with w-bit words by just replacing each word equal to 1 by 1^w and each word equal to 0 by 0^w.

Consider a version with single-bit words. We apply a difference with two active bits a multiple of n_1 positions apart, say positions 0 and rn_1. First, isolated active bits pass through χ unchanged with probability 1/4. The two-bit pattern also passes through θ unchanged because all $c_i(a)$ terms will be zero. Then ρ moves these bits to positions 0 and $grn_1 \bmod n$. The result is again two active bits a multiple of n_1 positions apart that will enter the next round. This can be repeated indefinitely leading to a differential trail with a weight of only 4 per round. Applying the Matryoshka principle one can construct a differential trail for word size w with weight $4w$ per round.

The obvious solution to this problem is to change the word transposition in ρ. A possible route is adopting a different type of function $f(x)$ for computing the new index from the old index in: $b_i = a_{f(i)}$. We tried different options but were not satisfied with the result. Then the structure of θ gave the key to the solution we adopted.

The terms $c_i(a)$ in θ naturally partition the n words in n_2 subsets of each n_1 words. We can arrange the words of the state as n_2 columns each containing n_1 words. The expression $c_i(a)$ then becomes the bitwise addition of all words of column i. The effect of θ is simply the addition to each word of the parities of the two neighboring columns. In this new structure, the position of a word in the state is determined by a couple of indices (x, y). In the following, we call a set of bits in the same slice with common y coordinate a *row* and with common x coordinate a *column* and we can index the slices with coordinate z. Hence, the state is a three-dimensional array and the position of a bit in the state is determined by a triplet (x, y, z).

The two-dimensional arrangement of words imposes a choice for χ. In RadioGatún the words of the state form a single circular array and γ (complement of χ) is applied to its slices independently. With the new structure, we can either arrange the words of the state in a single array, or have χ operate on n_1 circular arrays. We decided to go for the latter option and let χ operated on the n_1-bit rows separately.

The arrangement of words in a two-dimensional array opened up new possibilities for the word transposition in ρ, that we gave a dedicated name at this point: π. In the light of low-weight trail problem, the task of π is to move active bits in a column to different columns. A simple way to achieve that is to apply an operation like `ShiftRows` in Rijndael [27]. If the number of rows n_1 is not smaller than the number of columns n_2 this moves the bits in a column to n_1 different columns. Still, the `ShiftRows` solution has a remaining undesired property.

Consider again a single-bit word version. A difference with all bits in a row active and all other bits passive will pass through χ unchanged with probability 2^{1-n_2}. It will also pass through θ unchanged as all column parities will be 1 and hence $d_i = c_{i-1}(a) + c_{i+1}(a)$ will be zero. It will also pass through π unchanged and hence appears unchanged at the input of the next round. This results in a trail of arbitrary length with a weight of only $n_2 - 1$ per round. Applying Matryoshka results in a trail for word size w with weight $(n_2 - 1)w$ per round.

This problem will occur for any π that does not move bits in a row to different rows. So we needed to find a function $(x, y) = f(x', y')$ that computes the new coordinates of a bit (x, y) from the old coordinates (x', y') such that bits in the same column are moved to different columns and that not all bits in a row are moved to the same row. Looking into this problem, we saw that taking $n_1 = n_2$ allowed an elegant solution that generalized both the ρ word transposition in RadioGatún and `ShiftRows` of Rijndael. It is a linear function f defined by:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

with addition and multiplication in $\mathbb{Z}_{n_1}$. This type of mapping has interesting properties and lends itself for analysis. For example, it maps the points arranged in straight lines to straight lines.

In the new arrangement, θ can be expressed in a very simple way: it updates each bit of the state by adding to it the parities of the two neighboring columns in the same slice. The subset of all states that have an even parity in each column is the *(column parity) kernel*. For any state in the column parity kernel, θ behaves as the identity.

Note that π, χ and θ operate independently on slices.

8.3 Determining the dimensions

The value of n_1 in Keccak-f was determined by three elements:

- the NIST SHA-3 security requirements
- the fact that χ is not invertible for even lengths
- our wish for Keccak-f to be efficient on 64-bit processors

In its call for SHA-3 proposals [39], NIST required 4 output lengths n: 224, 256, 384 and 512 bits. For each of these lengths it required a resistance against collisions of $2^{n/2}$, preimages of 2^n and second preimages of $2^n/\ell$ calls to the hash function, with ℓ the length of the first message.

From the start we wanted to address all four output lengths with a single permutation and adapt the capacity to the required security level. In our original sponge paper [5], we had given a generic algorithm for generating a second preimage in $2^{c/2}$ calls to the underlying permutation. Hence to formally satisfy the second preimage security requirement, we had to take a capacity of 1024 bits for our 512-bit SHA-3 candidate. In order to have a reasonable performance, we figured the bitrate should not be smaller than one third of the state, and this put a lower bound on the width of Keccak-f of about 1500 bits. For the 256-bit SHA-3 candidate this would give a comfortable bitrate equal to two thirds of the width, making it twice as fast as the 512-bit SHA-3 candidate.

Due to the fact that χ is not invertible for even lengths and it operates on n_1-bit rows, n_1 must be odd. Taking $n_1 = 3$ would impose a minimum word size of $1500/9 \approx 160$. Taking $n_1 = 5$ and a word size of 64 bits would result in a permutation width of 1600 bits and the combination $n_1 = 7$ with 32-bit words would result in a 1568-bit permutation. A priori a 5-column version would be more efficient on 64-bit CPUs, dominant in high-end CPUS and a 7-column version would be more efficient on 32-bit CPUs, widely applied in embedded devices. However, we noticed that it is easier to implement 64-bit word step mappings on 32-bit CPUs than vice versa.

To avoid confusion between the CPU words and the words in the state, we renamed the latter to *lanes* at this point. The operations for which the lane size really matters are the lane rotations is ρ and θ. If the CPU word size equals the lane size, it can be implemented with cyclic shift instructions. If the CPU word size is smaller than the lane size, in a straightforward implementation each lane is coded in multiple words. For example, a 64-bit lane is split in two 32-bit words, with bits $0 - 31$ in one word and bits $32 - 63$ in the other. Performing the rotation to a lane takes multiple shift and bitwise Boolean operations. With the technique of *bit-interleaving* [17], the bits of a lane are organized differently. In its simplest form, namely factor-2 interleaving, it groups the bits of a lane with odd index in one word and the ones with even index in another. The rotations in θ and ρ can now be performed with cyclic shifts on 32-bit words, making them efficient on a 32-bit processor. The computational cost of the conversion of the input message into this representation is small compared to the evaluation of the permutation itself. So 64-bit lanes can be efficient on 32-bit CPUs but not vice versa. This favours the $5 \times 5 \times 64$ option.

RadioGatún had a size parameter in the form of its word size and was defined for any value in the range 1 to 64. For Keccak-f, we also wanted to support multiple sizes. However, we restricted the lane size to powers of two: $1, 2, 4, 8, 16, 32$ and 64. The reason for this is that the rotation constants in ρ have good distributions if the lane size is a power of two, while this is not necessarily the case for other lane size values. Moreover, this choice limits the number of instances to analyze with an eye on our security claims while it also optimizes the use of the Matryoshka principle in the analysis. The supported permutation widths range from toy versions of sizes 25 and 50, useful in performing experiments, over lightweight permutation with a relatively small state 100, 200, 400 to high-speed versions of 800 and 1600 bits.

8.4 Tweaking the θ mixing layer

For determining the remaining parameters we did a number of experiments. We made a test bench of Keccak-f with the π matrix as a configurable parameter, a number of different choices for mappings from (x, y) to i for ρ and different orders of the step mappings. We started conducting differential and linear trail analysis and experiments with the algebraic normal form (ANF) on these versions with the goal to gain insight for making a good choice.

When looking at the versions with 2-bit lanes, we found something remarkable. Namely, in the trails with lowest weight, often patterns occurred where the columns in one of the two slices had all odd parity, with often a single active bit per column. Note that these patterns are not in the column parity kernel but for which θ still acts as the identity. Computing the number of such patterns with Hamming weight 5 and comparison with the kernel reveals why they occur so often. In Keccak-$f[50]$ the kernel contains 200 patterns with Hamming weight 2 and 4550 with Hamming weight 4. The number of patterns with Hamming weight 5 for which θ acts as the identity is 6250. Moreover, if these 5 bits are not mapped to 5 different rows at the input of χ, the contribution to the weight is less than 2 times the number of active bits. In the worst case all bits are mapped to the same row, contributing only 4 to the weight.

We decided to look at alternatives for our initial choice of θ that would solve this problem. From the different options we considered, the following surfaced as achieving the best improvement with reasonable additional implementation cost. Instead of adding to each bit the parities of the two neighboring columns in the same slice, in our improved version one of these columns is in the neighboring slice. In a straightforward software implementation this can be realized by 5 additional cyclic shifts. For the previously problematic 5-bit patterns, the new θ flips all bits of the pattern. Generally speaking, the pattern outside the column parity kernel with the smallest Hamming weight for which θ acts as the identity now has an active bit in each column of the state. For Keccak-$f[50]$ this result in at least 10 active bits and for Keccak-$f[1600]$ this is 320. A side effect of this modification is that the inverse of θ changes drastically. The inverse of the old θ has a structure similar to θ itself: it consists of adding to each bit the parities of the two columns in the same slice that are not adjacent to it. For the new θ, it consists of adding the parities of about half of the columns in the state. To obtain a difference with a single active bit at the output of θ, about half of the bits of the state at its input must be active.

8.5 Determining the π transposition matrix

From a first thought experiment followed a constraint on the π matrix. In fact, π is a permutation operating over a set of 25 elements and can be described by a number of cycles. Due to its linearity, the origin $(0, 0)$ always maps to the origin. We can define axes in 6 *directions*:

- Horizontal: points with $y = 0$
- Vertical: points with $x = 0$
- Slope 1: points with $y = x$
- Slope -1: points with $y = -x$
- Slope 2: points with $y = 2x$
- Slope -2: points with $y = -2x$

Thanks to its linearity, π moves the 5 points on an axis to 5 points on an axis. So at the axis level, π can again be described as a number of cycles.

Earlier in our analysis we have already used several times the property of χ that a row with a single active bit it its input appears unchanged at its output with probability $1/4$ (so weight 2). It follows that for difference patterns in which all active rows have a single active bit, χ acts as the identity with probability 2^{-2a} with a the number of active rows. If we model χ as the identity, all step mappings are linear and the input difference determines all differences in a trail. A difference pattern that is in the kernel at the input of θ goes through θ unchanged and so leaves the number of active bits invariant. A possible strategy for finding a low-weight differential trail is to define an input difference that results in patterns that are in the kernel for as many rounds as possible.

With χ and θ both equal to the identity, the round function is reduced to the bit transposition $\pi \circ \rho$. For simplicity, let us focus on a single-slice version, where even ρ becomes the identity and only π remains. Let us consider the number of independent conditions imposed by being in the kernel for 1, 2, 3, …rounds. For a single round, membership in the kernel translates to 5 linear equations: each column must have an even number of active bits. So for a single round the fraction of all patterns that is in the kernel is 2^{-5}.

Adding one round results in 5 additional equations at the input of the second round. These equations can be transferred over π to equations at the input of the first round. If π maps columns to columns, these equations are the same as for the first round and hence can be discarded. If π maps e.g. lines with slope 1 to columns, it results in 5 additional equations expressing that the parity of bits in lines with slope 1 shall be zero. Of the resulting 10 equations only 9 are independent: the sum of the 5 column parity equation and the sum of the 5 slope-1 parity equations both state that the parity of the full slice must be even. So the fraction of states in the kernel in two successive rounds is 2^{-9}.

This reasoning can be extended by adding more rounds. When adding the n-th round in the case that π^n maps the vertical axis to itself, no new equations are added. Otherwise, 4 new equations are added. It follows that if π cycles through all 6 axes, the total number of independent linear equations for staying in the kernel for more than 5 rounds is $5 + 4 + 4 + 4 + 4 + 4 = 25$. Applying the 25 linear equations reduces the space of 2^{25} possible patterns to a single one: the all-zero pattern. If in π the vertical axis is in a cycle with period $n < 6$, the number of conditions to stay in the kernel for n rounds or more is $4n + 1 < 25$. This is the main reason for choosing a π that runs through all axes in a single cycle. Note that this reasoning can be generalized to the multi-slice case, leading to the same conclusion.

8.6 The configuration of ρ

In RadioGatún the shift offsets of the intra-word rotations are expressed as $i(i-1)/2$ with i the word index. However, in Keccak-f the lanes have two-component indices (x, y). Hence we needed to abandon the offset recipe of RadioGatún or to define a mapping between (x, y) and i. We considered different options but the following one struck us as more elegant than all others. There are matrices for π such that the 24 lane positions outside the origin form a single cycle. This provides a natural mapping between (x, y) and the sequence i. This restricted our choice of π matrices to those with order 24. From our experiments with the test bench no single candidate stood out and we took the one with the *simplest look* (with elements 0, 1, 2, 3).

8.7 The round constants

The round constants are there to disrupt symmetry, both in the temporal as in the three spatial dimensions. We decided to apply round constants in the lane in position $(0, 0)$ and only non-zero in a single bit position in Keccak-f[25], two positions in Keccak-f[50], ... up to seven bit positions in Keccak-f[1600]. The bits of the round constants are different from round to round and are taken as the output of a maximum-length eight-bit linear feedback shift register. We tested the asymmetry introduced by the round constants by means of a number of experiments making use of the algebraic normal form [15, Sect. 4.1.2].

8.8 The order of steps within a round

The round function consists of 5 step mappings and that a priori can be arranged in $5! = 120$ different orders. However, thanks to the fact two of the five steps are mere bit transpositions and there is only a single nonlinear step, these orders can be partitioned in a small number of subsets where the orders within a set have equivalent propagation properties.

Still, an essential choice is whether the nonlinear layer χ comes before the mixing layer θ or after it. We chose to put θ before χ for the following reason. In keyed applications of Keccak the adversary does not know the value of of the inner part of the state and may know the value of the outer part of the state. By putting the mixing layer θ before the non-linear layer χ, the unknown inner part whitens the χ input. If π would come before θ, for at least part of χ no such whitening would be present. Finally, the bit transpositions can be put at the beginning (or end) of the round or between θ and χ. We chose for the latter option based on the results of trail propagation experiments on Keccak-f[50] and Keccak-f[100]. The final order is $\iota \circ \chi \circ \pi \circ \rho \circ \theta$.

8.9 The number of rounds

In our original SHA-3 submission [6,10], we took 12 rounds for Keccak-f[25] and added one round for each doubling of the permutation width. This was based on our estimations of the maximum number of round that could possibly be attacked and adding a comfortable safety margin. Moreover, at the time we naively believed that adding a round doubles the algebraic degree of the permutation up to $b - 1$.

During the first round of the SHA-3 competition Jean-Philippe Aumasson and Willi Meier published their *zero-sum distinguishers* in [2] that exploited the low algebraic degree of the Keccak-f round function and its inverse. In theory, these zero-sum distinguishers can be used to distinguish reduced-round variants of Keccak-f[1600] up to 16 rounds from a random permutation. However, as we described in [11], the relevance of these distinguishers for the security of Keccak is very small. Still, in our view of the hermetic sponge strategy at the time, we wanted the underlying permutation to have no structural distinguishers, independent of their applicability in actual attacks on Keccak. As for the second round of SHA-3 the semi-finalists had the permission to tweak their proposals, we decided to augment the number of rounds in Keccak-f from $12 + \ell$ to $12 + 2\ell$ (from 18 to 24 rounds for Keccak-$f[1600]$). Sticking to 18 rounds would have left a security margin of only 2 rounds against a distinguisher of Keccak-f. We believed that the new definition leads to a more homogeneous safety margin: the number of rounds in Keccak-f is essentially twice the maximum number of rounds we believe to be attackable when used in a Keccak instance. Moreover, at the time we naively believed it would sufficiently increase the algebraic degree of both the permutation and its inverse, as exploited in zero-sum distinguishers.

During the second round of the SHA-3 competition, the zero-sum distinguishers were further refined by Christina Boura and Anne Canteaut in two papers [21,20] extending the distinguishers up to 20 rounds in [20]. They were later joined by Christophe De Cannière in [22] and extended the distinguisher to the full 24 rounds of Keccak-$f[1600]$. Finally, Ming Duan and Xuejia Lai achieved a slight reduction of the data complexity of this distinguisher. These results have the merit of explaining the evolution of the algebraic degree with the number of rounds in a wide class of block ciphers and permutations.

By this time we realized that these zero-sum distinguishers did not jeopardize the hermetic sponge strategy thanks to their large data complexity. For Keccak-$f[1600]$ exploiting a zero-sum distinguisher requires 2^{1575} calls to the permutation while flat sponge claim does not make any statements about the security against attacks requiring more than $2^{c/2}$ calls. As $c < b$, this is at most $2^{799.5}$. So when Keccak was chosen as one of the five finalists in the SHA-3 competition, we decided not to tweak the number of rounds any longer, despite the existence of a full-round distinguisher. We motivated our decision in our presentation at the Second SHA-3 candidate conference in Santa Barbara in August 2010 and in the documentation of our SHA-3 third-round submission [15,16].

8.10 The padding of the input

In our original SHA-3 submission [6,10], we had a padding rule that appends an encoding of the rate and a single-byte *diversifier*. For inputs that consist of an integer sequence of bytes, this rule appended more than 3 bytes. This was not so much a problem for the sponge construction, but was a real overhead for the related *duplex* construction, that we were working on at the time [9,13]. The main purpose of the diversifier was domain separation between different SHA-3 candidates and the inclusion of the rate for the joint security of multiple Keccak instances based on the same Keccak-f instance but with different rates. We refer to [10] for a more detailed motivation.

During the second round of the SHA-3 competition, we had discovered that the joint security of multiple Keccak instances with different rates can be achieved with a much simpler padding. We called it *multi-rate padding* and it consists of appending a single 1-bit, n 0-bits and again a single 1-bit, with n the smallest number such that the length of the result is a multiple of the rate. For byte-sequence inputs, this appends only a single byte at least. So for the third-round submission, we replaced our original padding by the multi-rate padding. We achieved domain separation between our SHA-3 candidates for different output lengths by adopting capacity values equal to twice the output length, hence resulting in 4 different capacity values.

8.11 Specification of Keccak

In this subsection we give a compact specification of Keccak. For the full specification we refer to [15].

There are 7 Keccak-f permutations, indicated by Keccak-$f[b]$, where $b = 25 \times 2^\ell$ and ℓ ranges from 0 to 6. The lanes have size $w = 2^\ell$. The name Keccak covers all sponge instances with multi-rate padding and any of the Keccak-f permutations and compatible r and c values, i.e. $c + r = b$.

The steps of the round function of Keccak-f are specified as operating on a three-dimensional state a with a bit in position (x, y, z) denoted by $a_{x,y,z}$ with $x, y \in \mathbb{Z}_5$ and $z \in \mathbb{Z}_w$. In our description, we may sometimes omit the z index, both y and z indices or all three indices, implying that the statement is valid for all values of the omitted indices.

The number of rounds in Keccak-$f[25 \times 2^\ell]$ is $n_\mathrm{r} = 12 + 2\ell$. A single round consists of a sequence of five steps:

$$
\begin{aligned}
\theta : a_{x,y,z} &\leftarrow a_{x,y,z} + c_{x-1,z} + c_{x+1,z-1} \text{ with } c_{x,z} = \sum_{y=0}^{4} a_{x,y,z} \\
\rho : a_{x,y,z} &\leftarrow a_{x,y,z-(t+1)(t+2)/2} \\
&\qquad \text{with } t \text{ satisfying } 0 \le t < 24 \text{ and } \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} \\
&\qquad \text{or } t = -1 \text{ if } x = y = 0 \\
\pi : a_{x,y} &\leftarrow a_{x',y'}, \text{ with } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix} \\
\chi : a_x &\leftarrow a_x + (a_{x+1} + 1) a_{x+2} \\
\iota : a &\leftarrow a + \mathrm{RC}_{i_\mathrm{r}}
\end{aligned}
$$

The round constants are given by

$$\mathrm{RC}[i_\mathrm{r}]_{0,0,2^j-1} = \mathrm{rc}[j + 7i_\mathrm{r}] \text{ for all } 0 \le j \le \ell,$$

and all other values of $\mathrm{RC}[i_\mathrm{r}]_{x,y,z}$ are zero. The values $\mathrm{rc}[t] \in \mathrm{GF}(2)$ are defined as:

$$\mathrm{rc}[t] = \left(x^t \bmod x^8 + x^6 + x^5 + x^4 + 1 \right) \bmod x \text{ in } \mathrm{GF}(2)[x].$$

9 The future: permutation-based cryptography

In this paper we have described how Keccak is the result of a long and iterative design process. But it is not an endpoint. Soon after we adopted the sponge construction for design, we realized that on top of hashing, it can also be used as a stream cipher, for MAC computation and as a mask generating function. In the summer of 2009 we started considering sponge variants with simultaneous input and output of data, as if the absorbing phase and squeezing phase would overlap. It turned out that we could model this as a mode of use on top of the sponge construction. This resulted in a paper on reseedable pseudorandom sequence generation that we presented at the CHES workshop in August 2010 [12]. We then formalized this in the so-called *duplex* construction and proposed a number of modes on top of it, including an efficient authenticated encryption mode. We presented this at the second SHA-3 candidate workshop in August 2010 [9]. The sponge and duplex constructions allow to build all symmetric cryptographic services on top a fixed-width permutation. As such, one of the by-products of the Keccak design process is a new branch of symmetric crypto: *permutation-based cryptography*.

References

1. R. Anderson, *The classification of hash functions*, Proceedings of the IMA Conference in Cryptography and Coding, 1993, 1993.
2. J.-P. Aumasson and W. Meier, *Zero-sum distinguishers for reduced Keccak-f and for the core functions of Luffa and Hamsi*, Available online, 2009, `http://131002.net/data/papers/AM09.pdf`.
3. M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM Conference on Computer and Communications Security 1993 (ACM, ed.), 1993, pp. 62–73.
4. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, RadioGatún, *a belt-and-mill hash function*, Second Cryptographic Hash Workshop, Santa Barbara, August 2006, `http://radiogatun.noekeon.org/`.
5. ———, *Sponge functions*, Ecrypt Hash Workshop 2007, May 2007, also available as public comment to NIST from `http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html`.
6. ———, Keccak *specifications*, NIST SHA-3 Submission, October 2008, `http://keccak.noekeon.org/`.
7. ———, *On the indifferentiability of the sponge construction*, Advances in Cryptology – Eurocrypt 2008 (N. P. Smart, ed.), Lecture Notes in Computer Science, vol. 4965, Springer, 2008, `http://sponge.noekeon.org/`, pp. 181–197.
8. ———, *The road from* panama *to* keccak *via* radiogatún, Symmetric Cryptography (Dagstuhl, Germany) (H. Handschuh, S. Lucks, B. Preneel, and P. Rogaway, eds.), Dagstuhl Seminar Proceedings, no. 09031, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009.
9. ———, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Second SHA-3 candidate conference, August 2010.
10. ———, Keccak *sponge function family main document*, NIST SHA-3 Submission (updated), June 2010, `http://keccak.noekeon.org/`.
11. ———, *Note on zero-sum distinguishers of* Keccak-f, Comment on the NIST Hash Competition Forum, January 2010, `http://keccak.noekeon.org/NoteZeroSum.pdf`.
12. ———, *Sponge-based pseudo-random number generators*, CHES (S. Mangard and F.-X. Standaert, eds.), Lecture Notes in Computer Science, vol. 6225, Springer, 2010, pp. 33–47.
13. ———, *Duplexing the sponge: single-pass authenticated encryption and other applications*, Selected Areas in Cryptography (SAC), 2011.
14. ———, *On the security of the keyed sponge construction*, Symmetric Key Encryption Workshop (SKEW), February 2011.
15. ———, *The* Keccak *reference*, January 2011, `http://keccak.noekeon.org/`.
16. ———, *The* Keccak *SHA-3 submission*, January 2011, `http://keccak.noekeon.org/`.
17. G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer, Keccak *implementation overview*, May 2012, `http://keccak.noekeon.org/`.
18. A. Biryukov and D. Wagner, *Slide attacks*, Fast Software Encryption (L. R. Knudsen, ed.), Lecture Notes in Computer Science, vol. 1636, Springer, 1999, pp. 245–259.
19. C. Bouillaguet and P. Fouque, *Analysis of the collision resistance of radiogatún using algebraic techniques*, SAC 2008 (Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, eds.), Lecture Notes in Computer Science, vol. 5381, Springer, 2008, pp. 245–261.
20. C. Boura and A. Canteaut, *Zero-sum distinguishers on the Keccak-f permutation with 20 rounds (working draft)*, private communication, 2010.
21. ———, *A zero-sum property for the Keccak-f permutation with 18 rounds*, Available online, 2010, `http://www-roc.inria.fr/secret/Anne.Canteaut/Publications/zero_sum.pdf`.
22. C. Boura, A. Canteaut, and C. De Cannière, *Higher-order differential properties of Keccak and Luffa*, Fast Software Encryption 2011, 2011.
23. J. Coron, Y. Dodis, C. Malinaud, and P. Puniya, *Merkle-Damgård revisited: How to construct a hash function*, Advances in Cryptology – Crypto 2005 (V. Shoup, ed.), LNCS, no. 3621, Springer-Verlag, 2005, pp. 430–448.
24. J. Daemen, *Cipher and hash function design strategies based on linear and differential cryptanalysis, PhD thesis*, K.U.Leuven, 1995.
25. J. Daemen and G. Van Assche, *Producing collisions for PANAMA, instantaneously*, Fast Software Encryption 2007 (A. Biryukov, ed.), LNCS, Springer-Verlag, 2007, pp. 1–18.
26. J. Daemen and C. S. K. Clapp, *Fast hashing and stream encryption with PANAMA*, Fast Software Encryption 1998 (S. Vaudenay, ed.), LNCS, no. 1372, Springer-Verlag, 1998, pp. 60–74.
27. J. Daemen and V. Rijmen, *The design of Rijndael — AES, the advanced encryption standard*, Springer-Verlag, 2002.
28. T. Fuhr and T. Peyrin, *Cryptanalysis of radiogatún*, FSE 2009 (Orr Dunkelman, ed.), Lecture Notes in Computer Science, vol. 5665, Springer, 2009, pp. 122–138.
29. Michael Gorski, Stefan Lucks, and Thomas Peyrin, *Slide attacks on a class of hash functions*, ASIACRYPT 2008 (Josef Pieprzyk, ed.), Lecture Notes in Computer Science, vol. 5350, Springer, 2008, pp. 143–160.

30. A. Joux, *Multicollisions in iterated hash functions. Application to cascaded constructions*, Advances in Cryptology – Crypto 2004 (M. Franklin, ed.), LNCS, no. 3152, Springer-Verlag, 2004, pp. 306–316.
31. J. Kelsey and B. Schneier, *Second preimages on n-bit hash functions for much less than 2^n work*, Advances in Cryptology – Eurocrypt 2005 (R. Cramer, ed.), LNCS, no. 3494, Springer-Verlag, 2005, pp. 474–490.
32. D. Khovratovich, *Two attacks on radiogatún*, INDOCRYPT 2008 (Dipanwita Roy Chowdhury, Vincent Rijmen, and Abhijit Das, eds.), Lecture Notes in Computer Science, vol. 5365, Springer, 2008, pp. 53–66.
33. ———, *Cryptanalysis of hash functions with structures*, SAC 2009 (Michael J. Jacobson Jr., Vincent Rijmen, and Reihaneh Safavi-Naini, eds.), Lecture Notes in Computer Science, vol. 5867, Springer, 2009, pp. 108–125.
34. T. Kohno and J. Kelsey, *Herding hash functions and the Nostradamus attack*, Advances in Cryptology – Eurocrypt 2006 (S. Vaudenay, ed.), LNCS, no. 4004, Springer-Verlag, 2006, pp. 222–232.
35. RSA Laboratories, *PKCS # 1 v2.2 RSA Cryptography Standard*, 2012.
36. M. Liskov, *Constructing secure hash functions from weak compression functions: The case for non-streamable hash functions*.
37. U. Maurer, R. Renner, and C. Holenstein, *Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology*, Theory of Cryptography - TCC 2004 (M. Naor, ed.), Lecture Notes in Computer Science, no. 2951, Springer-Verlag, 2004, pp. 21–39.
38. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.
39. NIST, *Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) family*, Federal Register Notices **72** (2007), no. 212, 62212–62220, `http://csrc.nist.gov/groups/ST/hash/index.html`.
40. ———, *NIST selects winner of secure hash algorithm (SHA-3) competition*, October 2012, `http://www.nist.gov/itl/csd/sha-100212.cfm`.
41. ———, *Third-round report of the SHA-3 cryptographic hash algorithm competition*, November 2012, `http://dx.doi.org/10.6028/NIST.IR.7896`.
42. V. Rijmen, B. Van Rompay, B. Preneel, and J. Vandewalle, *Producing collisions for PANAMA*, Fast Software Encryption 2001 (M. Matsui, ed.), LNCS, no. 2355, Springer-Verlag, 2002, pp. 37–51.
43. Wikipedia, *Cryptographic hash function*, 2008, `http://en.wikipedia.org/wiki/Cryptographic_hash_function`.

Biographical sketches

Guido Bertoni obtained a Dr. Eng degree in computer engineering and a PhD degree from Politecnico di Milano in 1999 and 2004 respectively. He joined STMicroelectronics in fall 2003 as researcher in the field of cryptography in the Advanced System Technology system research organization. His research interests include cryptographic algorithms, hardware and software implementations, and problems related to side channels attack. He has taught cryptography at Politecnico di Milano as contract professor. He contributed to the Bluetooth consortium for the development of the Secure Simple Pairing. He has authored or co-authored about 40 scientific papers, has served in the program committee of various conferences and will be program co-chair of the CHES conference in Santa Barbara in August 2013.

Joan Daemen got his PhD in Cryptography at the Katholieke Universiteit Leuven in March 1995. He has continued to design and cryptanalyze block ciphers, stream ciphers and cryptographic hash functions ever since. In 1997, he designed in collaboration with Vincent Rijmen the block cipher Rijndael that NIST selected to become the Advanced Encryption Standard (AES) in October 2000. During the twelve years that have passed, AES has become ubiquitous and it has influenced the majority of symmetric cryptography primitives designed since. Nowadays, Joan works in the Secure Microcontrollers Division of STMicroelectronics in Diegem, Belgium. There, with Guido Bertoni, Michaël Peeters and Gilles Van Assche, he formed the Keccak team, that designed of the Keccak sponge function. NIST selected Keccak to become the SHA-3 standard hash function in October 2012. In parallel with his crypto work, Joan designs and specifies secure microcontroller based security architectures. His work in cryptography has lead to numerous scientific publications including a book on Rijndael. Joan has also served as a jury member of about a dozen PhDs, served in numerous program committees and has given many invited talks at cryptography and security conferences.

Michaël Peeters has a master degree of science in electronics and telecommunication engineering with orientation in computer science. He has more than 10 years of experience in the security field, focusing on highly demanding sectors like banking, identification, eGovernment and telematics. He's the co-author of several papers in cryptography, and is the co-designer of Keccak, a cryptographic hash function that was recently selected by NIST as the new SHA-3 standard. He is currently hired as Security Architect for the telematics project within the NXP Semiconductors' Business Unit Automotive.

Gilles Van Assche currently works in the Secure Microcontrollers Division of STMicroelectronics in Diegem, Belgium and teaches cryptography at the École Supérieure d'Informatique in Brussels. He receives the Physics Engineer degree from the Université Libre de Bruxelles (ULB) in 1998. He then joins the company Proton World, which later became part of STMicroelectronics. Between 2000 and 2005, in parallel with his job, he works on a PhD thesis at the Center for Quantum Information and Communication of the ULB. He is a co-recipient of the prize Le Prix La Recherche mention Mobilités 2004 for his work in quantum cryptography. He is the author of the book "Quantum Cryptography and Secret-Key Distillation" (Cambridge University Press) and of about 30 scientific papers, and has served in the program committee of various conferences. His current research interests are hash function design, modes of operation and side channel attacks. Together with Guido Bertoni, Joan Daemen and Michaël Peeters, he is a co-designer of the Keccak sponge function, which was selected by NIST as the winner of the SHA-3 competition. At ST, he works on security and testing aspects of software on secure microcontrollers.

The Keccak Team: during the last few years Guido, Joan, Michaël and Gilles have joined forces, concentrating on permutation-based cryptography with a focus on actual usability. This has lead among other things to the new sponge and duplex constructions, with their unique combination of simplicity and flexibility. These constructions allow hashing, encryption, authentication and authenticated encryption, all based on one single simple component: an iterated permutation. Based on them, they designed the primitive Keccak and submitted it to the NIST SHA-3 competition. The inner workings of Keccak are equally innovative, making use of a fresh set of round components. Aside from Keccak, the sponge and duplex constructions are also adopted by several so-called lightweight functions.