IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2021, No. 1, pp. 239–268. DOI:10.46586/tosc.v2021.i1.239-268

Algebraic Collision Attacks on Keccak

Rachelle Heim Boissier[1], Camille Noûs[2] and Yann Rotella[1]

[1] Université Paris-Saclay, UVSQ, CNRS, Laboratoire de mathématiques de Versailles, 78000, Versailles, France yann.rotella@uvsq.fr
[2] Laboratoire Cogitamus camille.nous@cogitamus.fr

Abstract. In this paper, we analyze the collision resistance of the two smallest versions of Keccak which have a width of 200 and 400 bits respectively. We show that algebraic and linearization techniques can serve collision cryptanalysis by using some interesting properties of the linear part of the round function of Keccak. We present an attack on the Keccak versions that could be used in lightweight cryptography reduced to two rounds. For Keccak[40, 160] (resp. Keccak[72, 128] and Keccak[144, 256]) our attack has a computational complexity of 2^{73} (resp. $2^{52.5}$ and $2^{101.5}$) Keccak calls.

Keywords: Keccak · Algebraic Cryptanalysis · Hash functions · Linearization · Collision attack

1 Introduction

The family of primitives Keccak was designed by the Keccak team (Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche) as a candidate to the US National Institute of Standards and Technology's hash function competition to create a new Secure Hash Algorithm standard. In 2012, Keccak won this contest and six of its instances became standardized as SHA-3. Keccak uses the Sponge construction [BDPA11a] which was introduced in 2007 [BDPA07]. Since its creation, Keccak standardized and non standardized instances have fostered a lot of cryptanalysis. Furthermore, to stimulate practical cryptanalysis, the Keccak designers have organized a cryptanalysis contest on Keccak instances with a generic security level of 2^{80} [BDPA08b].

The first practical pre-image and collision attack on 2-round Keccak was introduced by Naya-Plasencia, Röck and Meier in 2011 using differential cryptanalysis [NRM11]. Keccak cryptanalysis quickly started to also use algebraic techniques, starting in 2012 for collision search with a practical attack on four rounds by Dinur, Dunkelman and Shamir [DDS12] and in 2013 for pre-image search with a SAT-based attack by Morawiecki and Srebrny [MS13].

About collision search, Dinur, Dunkelman and Shamir published another article in 2013 [DDS13] and found the first collisions on Keccak-384 and Keccak-512 reduced to three rounds. In 2017, building on their work, Qiao, Song, Liu and Guo [QSLG17] presented new attacks on instances of the Crunchy contest and Song, Liao and Guo [SLG17] presented the first practical attack against Keccak-224 reduced to five rounds, and against Keccak[1440,160] reduced to six rounds with digests of 160 bits. Finally, in 2016 and 2017, a number of collisions on Keccak instances from the Crunchy contest were found and later published in [GLL+20].

Pre-image attacks have been dominated since 2016 by linearization techniques following the introduction of linear structures by Guo, Liu and Song [GLS16]. The following year, building on their work, other cryptanalysts introduced cross-linear structures [LSLW17] and attacked other instances of the Crunchy Contest. On larger instances of Keccak,

Received: 2020-09-01 Revised: 2020-11-23 Accepted: 2021-01-23 Published: 2021-03-19

Kumar, Rajasree and Al-Khzaimi [KRA18] introduced the first practical attack against Keccak-512 in 2017, also thanks to linearization techniques. The following year, building up on the work [NRM11], Kumar, Mittal and Singh [KMS18] offered an improvement on an attack on Keccak-384 reduced to two rounds. In 2019, Rajasree [Raj19] improved the previous attacks on Keccak-384 and Keccak-512 reduced to three and four rounds. In 2019, Li and Sun presented the first practical attack against Keccak-224 reduced to three rounds as well as impractical attacks on three and four rounds of Keccak-224 and Keccak-256 [LS19].

In the literature, there is also a plethora of cryptanalysis of keccak instances using cube-like attacks thanks to the very low degree of the round function: the only non-linear part is χ, which is quadratic. While those attacks are of interest to distinguish the Keccak-p permutations from a random permutation, it is unlikely to use them in a collision or pre-image attack on a Keccak instance.

In a summary of the current results of the Crunchy contest, the authors notice that 'Remarkably, the smaller versions are harder to break'. Indeed, only the 1-round version of the smallest version, namely Keccak[40,160] reduced to one round has been successfully attacked [EW17] by just canceling the effect of the (single) round constant. It has thus been suggested to use the smallest versions of Keccak in constrained environments [KY10]. Moreover, the permutations of the smallest instances, namely Keccak-$p[200, n_r]$ and Keccak-$p[400, n_r]$ are used as building blocks for some Authenticated Encryption algorithms for different numbers of rounds n_r, such as Ketje [BDP+16] and two proposals present in the second round of the NIST lightweight competition that started in 2018; ISAP [DEM+20] and Elephant [BCDM20]. On the other hand, one can notice that cryptanalysts of Keccak have mainly targeted the standards [oST15]. There exists much more third-party cryptanalysis on the instances with 1600 bits states than the instances with 200 or 400 bits states. We thus decided to analyze the security of these smaller instances of Keccak against collision attacks to fill this void.

Our contribution In this paper, we show that algebraic analysis can also serve collision attacks, and not only pre-image attacks. We use a squeeze attack as described in [DDS13] in order to provide inner state collisions. Our attack can therefore be applied even if the output length is extended, or if Keccak is used as a XOF (eXtendable Output Function). We control the diffusion over one round thanks to interesting properties of the θ mapping. By analyzing the χ mapping, we derive necessary linear equations and use them as a basis to compute Keccak states such that their inner states belong to a subset of the output set. Our attack is better than existing ones for the versions of Keccak that have small width. Indeed, the existing attacks on other instances of Keccak mainly work because the rate offers a relatively large degree of freedom, whilst this is not the case for Keccak$[40, 160]$, Keccak$[72, 128]$ and Keccak$[144, 256]$ when looking for inner collisions.

Table 1 summarizes the complexity of our collision attack. The memory complexity is negligible, as stated in Section 9.

Table 1: Summary of our contribution. The time complexity for our collision attacks is given in number of calls to the Keccak construction when the width is 200 and 400 and when the number of rounds is 2.

Keccak instance, $n_r = 2$	Keccak$[40, 160]$	Keccak$[72, 128]$	Keccak$[144, 256]$
Time complexity	2^{73}	$2^{52.5}$	$2^{101.5}$

Comparison with previous works Our cryptanalysis works on the smallest KECCAK versions reduced to two rounds. It cannot beat the best known attacks when the width is 800 or 1600. Previous techniques [NRM11, DDS12, DDS13, QSLG17, SLG17, GLL+20] which have been used to build squeeze attacks to get collisions in the output cannot be employed on small versions since the attacker can only control a small amount of bits between each iteration of KECCAK-p[200] and KECCAK-p[400]. Hence our attack shows that cryptanalysis does not naturally scale with the width, even though the construction works similarly.

Our collision attack uses linearization techniques that are usually employed in pre-image attacks such as in [GLS16] or more recently in [SLG17, GLL+20] to improve the work of [DDS12]. However, the linear conditions derived by the linearization of the χ mapping in the above pre-image attacks are conditions that the state must satisfy, in other words necessary conditions. On the other hand, we derive sufficient conditions from the linearization of χ.

We use an interesting property of the θ mapping to control the diffusion on selected pairs of bits. Although many things were already known on θ [SD18], our observation is different to the extent that it is a property on pairs of bits before and after θ. This allows us to work locally, without worrying about the effect of θ on other bits on the state or parity-bit values of columns.

Finally, our attack is an inner collision attack, which can be applied no matter what the output length is. This is not the case for most collision attacks, as cryptanalysts usually look into building collisions in the output (that is, the outer part).

Outline of the paper We begin with a brief description of KECCAK in Section 2. Section 3 provides a generic description of our attack. In Section 4, we describe the properties of the KECCAK step mappings that we will use as a starting point for our cryptanalysis while Section 5 fixes the choices we make for our cryptanalysis. Section 6 describes our specific use of the χ mapping linearization and Section 7 provides improvements of the attack. Section 8 describes how to use our attack for any possible rate. Finally, Section 9 provides the exact complexity of our attack and a brief description of the implementation of our proof of concept.

2 Description of the Keccak family

In this section, we provide a short description of the KECCAK family.

2.1 The sponge construction

The family of hash functions KECCAK is built on the sponge construction [BDPA07, BDPA08a, BDPA11a, BDPA13]. As illustrated in Figure 1, the sponge construction is a mode of operation which maps an input $\mathcal{M}$ of arbitrary length called the message to an output $\mathcal{Z}$ of fixed length d, where d is called the diversifier. To do so, it uses a permutation f and a padding rule. The permutation f operates on a state S of width $b = r + c$ where c is called the capacity and r the bitrate. The bits of the state S are numbered from 0 to $b - 1$. The first r bits of a state S form the outer state, and its value is denoted by $\overline{S}$, while the next c bits correspond to the inner state, and its value is denoted by $\widehat{S}$.

The construction works as follows. The message $\mathcal{M}$ is first padded so that its length is a multiple of r. Then, it is cut into n bit strings of length r: $M_0, \ldots, M_{n-1}$. The state is initialized to 0^b. The mode of operation then proceeds in two phases. The absorbing phase consists in XORing the r first bits of the current state with M_i, apply f, and iterate. The squeezing phase consists in returning the outer part of the state Z_0, then applying f, then returning the outer state of the image $\overline{f(Z_0)} = Z_1$ and concatenate this image to Z_0,

and so on and so forth. When the length of $Z_0||Z_1||\cdots$ is greater than the desired length d, it is truncated to such length to form the output $\mathcal{Z}$.

Keccak is the family of sponge functions which use the padding rule `pad10*1` and a permutation f from the Keccak-p family as the underlying permutation.

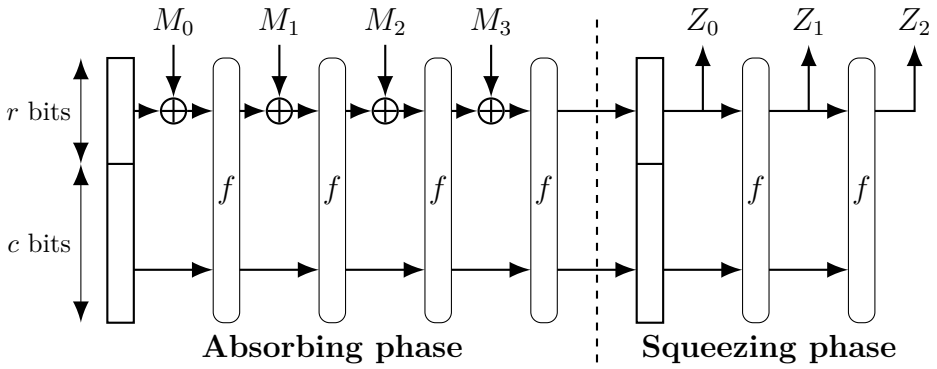

Figure 1: The Sponge Construction

2.2 The Keccak-p permutations

The Keccak-p permutations are specified with parameters b and n_r, where b is the width of the state and n_r the number of iterated rounds. We denote a permutation of this family by Keccak-$p[b, n_r]$.

The Keccak state

As specified in the last section, the Keccak-p permutations operate on a state $S \in \mathbb{Z}_2^b$, where $b \in \{25 \times 2^i\}_{i\in[0,6]}$. This state can be represented in a three dimensional array of size $A[5, 5, \omega]$ of elements of $\mathbb{F}_2$, where $\omega = \frac{b}{25}$. We let $A[x, y, z]$ be the bit with coordinates (x, y, z) in this array, where $0 \leq x < 5, 0 \leq y < 5, 0 \leq z < \omega$. The mapping between the bits of the state S and those of A is $S[\omega(5y + x) + z] = A[x][y][z]$. The labeling convention of the array is represented in Figure 2 below.

Terminology The outer state as well as the inner state are defined as sub-parts of the Keccak state. The inner state is made of the last c bits of the state. Hence, we define $\mathcal{IS}$ as the set of indices of those bits. Namely

$$\mathcal{IS} = \{(x, y, z) \mid \omega(5y + x) + z \geq r\}.$$

Adopting the representation of the Keccak designers (Guido Bertoni, Joan Daemen, Michaël Peeters and Gilles Van Assche), we will use the following notations :

- a *slice* (in orange in Figure 2) is a set of 25 bits with constant z coordinate $A[*, *, z]$;
- a *plane* is a set of 5ω bits with constant y coordinate $A[*, y, *]$;
- a *sheet* is a set of 5ω bits with constant x coordinate $A[x, *, *]$;
- a *row* (in cyan in Figure 2) is a set of 5 bits with constant y and z coordinates $A[*, y, z]$;

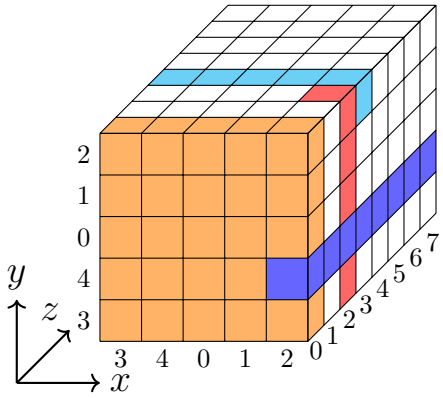

Figure 2: The KECCAK state.

– a *column* (in red in Figure 2) is a set of 5 bits with constant x and z coordinates $A[x, *, z]$;

– a *lane* (in blue in Figure 2) is a set of ω bits with constant x and y coordinates $A[x, y, *]$.

The Keccak round

The KECCAK-p permutations consist in the iteration of a set number of rounds n_r. A round consists in the composition of 5 state mappings θ, ρ, π, χ and ι. In the following, the operations on coordinates are always computed modulo 5, 5 and ω respectively. For $0 \leq x, y < 5$ and $0 \leq z < \omega$,

– θ XORs each bit of the state with the parities of two other columns of the state:

$$\theta(A)[x, y, z] = A[x, y, z] \oplus \bigoplus_{i=0}^{4} (A[x-1, i, z] \oplus A[x-1, i, z-1]) ;$$

– ρ rotates each lane by a constant:

$$\rho(A)[x, y, z] = A[x, y, z + c(x, y)] ;$$

– π modifies the position of each lane:

$$\pi(A)[x, y, z] = A[x + 3y, x, z] ;$$

– χ is a non-linear mapping. It XORs each bit with a non-linear function of two other bits of its row:

$$\chi(A)[x, y, z] = A[x, y, z] \oplus (\neg A[x+1, y, z]) \wedge A[x+2, y, z] ;$$

– ι XORs the lane $A[0][0][*]$ with a constant which depends on a round index i_r, where i_r depends on b and n_r.

The round mapping consists in the composition of these permutations. Namely,

$$R = \iota(-, i_r) \circ \chi \circ \pi \circ \rho \circ \theta .$$

2.3 The Keccak functions

The KECCAK-f family of permutations is defined as follows:

$$\text{KECCAK-}f[b] = \text{KECCAK-}p[b, 12 + 2\ell], \text{ where } \ell = \log_2(\omega)\,.$$

KECCAK is the family of sponge functions which uses the padding rule `pad10*1` (*multi-rate padding*) and the KECCAK-$f[b]$ permutations. More precisely, KECCAK$[r, c]$ denotes the sponge function which uses the multi-rate padding, the bitrate r and the permutation KECCAK-$f[r + c]$.

$$\text{KECCAK}[r, c] = \text{SPONGE}[\text{KECCAK-}f[r + c], \texttt{pad10*1}, r]$$

The permutation used in the SHA-3 standard [oST15] from the National Institute for Standard Technology is KECCAK-$f[1600]$. However, the design of KECCAK comes with several instances, including instances with a permutation KECCAK-f that operates a smaller state width b, such as KECCAK-$f[200]$. This allows the use of KECCAK in constrained environments as well. For instance, the permutations KECCAK-$f[200]$ or KECCAK-$f[400]$ are used in a proposal in an unkeyed mode for RFID [KY10]. For different number of rounds n_r, the permutations KECCAK-$p[200, n_r]$ and KECCAK-$p[400, n_r]$ also appear in keyed mode proposals, such as KETJE [BDP+16] in the CAESAR competition or ISAP [DEM+20] and Elephant [BCDM20] in the NIST lightweight cryptography competition. Finding a collision on the function using a round-reduced version of KECCAK[40,160] reduced to two rounds is a problem that was posed by the KECCAK designers in their Crunchy Contest [BDPA08b]. Hence studying smaller versions has a theoretical and practical interest.

A note on the output size. Note that the size of the output size d (the digest size) is not set, and has to be specified. For the SHA-3 standards, $d = \frac{c}{2}$. Versions with a smaller width such as the instances that use KECCAK-$f[200]$ as their permutation also have a smaller capacity (for example $c = 160$ or $c = 128$ for $b = 200$). If for these versions we had $d = \frac{c}{2}$ as well, then a generic birthday attack would lower their security to $2^{\frac{c}{4}}$, which is not secure. As a consequence, for small instances, designers usually set $d = c$. This the case for the instance of KECCAK[40,160] that is proposed as a cryptanalysis challenge in the Crunchy contest.

We will show that our attack works on several variants, but only beats the best attacks when the bitrate is somehow 'small' compared to the capacity. We will also show that it applies only to the search for inner collisions, which is relevant when the output length d is larger or equal to c. Hence, our attack is particularly relevant on versions of KECCAK with small width since in these variants, c is proportionally larger compared to b and d than in other versions so as to maintain the same security level.

3 Generic description of the attack

This section presents useful observations on the different strategies that can be used to find collisions on sponge functions with different parameters.

3.1 Building collisions on sponge functions

In the following, we use the absorb function as defined in Section 2.4.1 of [BDPA11a] which takes as input a string P with $|P|$ multiple of r and returns the value of the state obtained after absorbing P. Similarly, we use the following definitions of [BDPA11a]:

Definition 1. Let $P \in \mathbb{Z}_{2^r}^*$. P is a *path* to the state S if $S = \text{absorb}(P)$. More generally we denote by *path* P any bit string of which the size is a multiple of r.

Definition 2. A *collision* or *output collision* on a sponge function SPONGE is a pair of two different messages $\mathcal{M}$ and $\mathcal{M}'$ such that $\text{SPONGE}(\mathcal{M}) = \text{SPONGE}(\mathcal{M}')$.

Definition 3. A *state collision* is a pair of two different paths P and P' such that $\text{absorb}(P) = \text{absorb}(P')$.

Definition 4. An *inner collision* is defined as a state collision on the inner state. More precisely, it is a pair of two different paths P and P' such that $\overbrace{\text{absorb}}(P) = \overbrace{\text{absorb}}(P')$.

As observed in [BDPA11a], if one finds an inner collision they can derive an output collision from it. Indeed, for any $A, B \in \mathbb{Z}_2^r$ such that $\overline{\text{absorb}(P)} \oplus A = \overline{\text{absorb}(P')} \oplus B$, any two messages of the form $\mathcal{M} = P||A||\mathcal{N}$ and $\mathcal{M}' = P'||B||\mathcal{N}$, where $\mathcal{N}$ is any message, will lead to a collision in the output.

For KECCAK standardized hash functions [oST15], the length of the output d is half the length of the capacity c. Therefore, in these versions, a collision can be found by generating approximately $2^{\frac{d}{2}}$ outputs thanks to the classical birthday argument. On the other hand, generating generically an inner collision is more expensive since it requires the attacker to generate approximately $2^{\frac{c}{2}} = 2^d$ absorbed states. Moreover, it is noticeable that in these standardized versions, the bitrate is always larger than the diversifier. This means that in practice, the squeezing phase only consists in outputting the first d bits of the image of an absorbed message. An attacker is therefore going to seek a collision on the first d bits of the state of an absorbed message, which we call a d-collision. This strategy was used in all the KECCAK cryptanalysis papers that looked at the search of collisions so far [NRM11, DDS12, DDS13, QSLG17, SLG17, GLL+20].

In this paper, we study different instances of KECCAK, smaller versions, for which the previous observations do not apply. On small KECCAK instances, it is often smarter to look for an inner collisions rather than d-collision. There are two reasons for this. First, often, the capacity equals the output length, $c = d$ (see Section 2.2). Therefore, the birthday attack has no reason to work better on the search for d-collisions. In fact, there is already a small advantage to the search of inner collisions. Indeed, it enables the attacker to ignore the padding rule. As stated before, for any $A, B \in \mathbb{Z}^r$ such that $\overline{\text{absorb}(P)} \oplus A = \overline{\text{absorb}(P')} \oplus B$, any two messages of the form $\mathcal{M} = P||A||\mathcal{N}$ and $\mathcal{M}' = P'||B||\mathcal{N}$, where $\mathcal{N}$ is any message, will lead to a collision in the output. That means that in practice once an inner collision is found, it is easy to extend the two messages to an arbitrary length so that the padding rule will only have an effect on the new appended part and still lead to an output collision. This does not hold for d-collisions. Secondly, the bitrate is smaller than the output size, $r < d$. Therefore, looking for a d-collision would force the attacker to obtain several outer state collisions in the squeezing phase. This would require the attacker to control not only the outer but also the inner part of the state. It stems from our analysis that the best strategy when the bitrate is smaller than the output length, and the capacity is smaller or equal to the output length, is to search for an inner collision. Indeed, inner collision resistance does not depend on the output length nor on the padding rule.

3.2 Generic description of the attack

The birthday squeeze attack

The strategy we use to produce a collision is a birthday squeeze attack, as it is called by the authors of [DDS13]. Because of the birthday paradox, if a function maps the set of possible inputs to an output set E of size $|E|$, then we need to try about $\sqrt{|E|}$ inputs so as to find two colliding outputs. But if we are able to pick inputs so that they are all mapped to a predefined subset $E' \subsetneq E$ of size $|E'| < |E|$, then we will only need to produce about $\sqrt{|E'|}$ of these inputs so as to find a collision. In our case, since we are

looking for a collision on an output of size c (the inner state of an absorbed message), a generic birthday attack requires about $2^{\frac{c}{2}}$ inputs. Yet we are going to exploit the degree of freedom provided by the bit string M so as to produce outputs that are all in a predefined subset of smaller size, thereby improving the complexity of the attack.

Birthday squeeze attacks on inner collisions

In [DDS13], the birthday squeeze attack on Keccak was used in order to search for d-collisions. In our case however, we are looking for an inner collision. It was thus necessary for us to adapt the birthday squeeze attack. To do so, we rely on the following theorem which is essentially a reformulation of results of [BDPA11a].

Theorem 1. *Finding an inner collision is equivalent to finding two paths P, P' and two bit strings $M, M' \in \mathbb{Z}_2^r$ such that the two following conditions are true*

$$P \neq P' \text{ or } M \neq M' \tag{1}$$

$$\widehat{f(M||\widehat{S})} = \widehat{f(M'||\widehat{S}')} \tag{2}$$

where $S = absorb(P)$ and $S' = absorb(P')$.

To prove this theorem, we first demonstrate the following lemma. This lemma demonstrates one implication of Theorem 1. The other implication is easy to derive from the definition of an inner collision. Again we derive this result from [BDPA11a], which one can refer to for more details.

Lemma 1. *Let $M, M' \in \mathbb{Z}_2^r$, and P, P' two paths such that $M \neq M'$ or $P \neq P'$. Suppose we have $\widehat{f(M||\widehat{S})} = \widehat{f(M'||\widehat{S}')}$. Then $P||(\overline{S} \oplus M)$ and $P'||(\overline{S'} \oplus M')$ is an inner collision.*

Proof. Let $M, M' \in \mathbb{Z}_2^r$, and P, P' two paths such that $M \neq M'$ or $P \neq P'$. Let $S := \text{absorb}(P)$ and $S' := \text{absorb}(P')$. Suppose we have $\widehat{f(M||\widehat{S})} = \widehat{f(M'||\widehat{S}')}$. We have

$$\begin{aligned}
\widehat{\text{absorb}(P||(M \oplus \overline{S}))} &= \widehat{f(S \oplus ((M \oplus \overline{S})||0^c)} \\
&= \widehat{f(M||\widehat{S})} \\
&= \widehat{f(M'||\widehat{S}')} \\
&= \widehat{f(S' \oplus ((M' \oplus \overline{S'})||0^c))} \\
&= \widehat{\text{absorb}(P'||(M' \oplus \overline{S'}))}\,.
\end{aligned}$$

If $P \neq P'$, then $P||(M \oplus \overline{S}) \neq P'||(M' \oplus \overline{S'})$. If $P = P'$, then $M \neq M'$, $P||(M \oplus \overline{S}) \neq P'||(M' \oplus \overline{S'})$ and we have proven the lemma. □

In order to seek an inner collision, we are thus going to look for two pairs (P, M) and (P', M') that respect the two conditions of Theorem 1.

Since $r < \frac{c}{2}$, it is unlikely to obtain a collision in the inner states produced by taking the image by f of the same initial inner state $\widehat{S}$ and only modifying the bit string M. Indeed, for a single S, there are 2^r possible $M||\widehat{S}$ and therefore exactly 2^r possible values for $f(M||\widehat{S})$ as f is a permutation. To use a squeeze attack with 2^r inner states, we would need these inner states to all belong to a predetermined subset of $\mathbb{F}_2^c$ of size $(2^r)^2 = 2^{2r} < 2^c$. Yet the high diffusion and confusion provided by a round of Keccak make it unlikely *a priori* that this is true. In particular, there is no reason to always consider $S = 0^c$. As a consequence, our attack uses inner states of absorbed random bits strings. On the other hand, attacks on standardized versions of Keccak rely on the search for d-collisions, and

since $r > \frac{d}{2}$, considering the 2^r possible $M||0^c$ is sufficient. In our attack algorithm, we decided arbitrarily to use paths of size $10r$ in the case of Keccak[40, 160]. We could have chosen a different coefficient j as long as $jr \gg c$ in order to assume accurately that the states we obtain follow a uniform distribution over the inner states in $\mathbb{F}_2^c$.

We can now provide the reader with a generic algorithmic description of our attack.

Generic description of our attack algorithm

1. Start with an empty table.

2. Produce a random inner state.
   To do so, produce a random padded message $P = M_1||M_2||\cdots||M_j$ where $jr \gg c$ by concatenating random r-bit strings $M_1, M_2, \ldots, M_j$. Absorb it so as to produce a random state $S = \text{absorb}(P)$

3. Produce an inner state belonging to a predefined subset of $X \subset \mathbb{F}_2^c$.
   Exploit the different properties of Keccak in order to find an r-bit string M such that the inner state of $f(M||\widehat{S})$ belongs to a predetermined proper subset of $\mathbb{F}_2^c$ with high probability.

   If the inner state of $f(M||\widehat{S})$ belongs to the desired subset, store it in a hash table, and continue. Else, discard it and go back to step 2.

4. Look for collisions.
   Check for a collision in the table. If a collision is found, output the two pairs (P, M) and (P', M'). Else go back to step 2.

The birthday squeeze attacks works in our case by using the degree of freedom provided by the r-bit strings M in the absorbing phase. For each random state S, we are going to choose the next absorbed bit string M so as to ensure that the inner state of $f(M||\widehat{S})$ belongs to a predetermined proper subset of $\mathbb{F}_2^c$ with high probability. Once a collision is found in the table of the above algorithm, condition (2) in Theorem 1 is automatically satisfied. Furthermore, using random paths to produce random inner states enables us to satisfy condition (1) in Theorem 1 with high probability.

The proper subset to which we seek our inner states to belong to is predefined in the sense that it is common for all P, and depends neither on previous elements of the table nor on the random inner state considered. Similarly, the choice of each M does not depend on previous computation, but only on the current $\widehat{S}$ considered and the predefined proper subset. This is so as to ensure the applicability of the birthday paradox. This predefined proper subset is denoted by X. There are several X's possible. Our work consists in describing a subset X, together with an algorithm faster than processing random paths that can produce paths P such that $\widehat{\text{absorb}(P)} \in X$, that is Step 3.

4 Properties of Keccak-p permutations

In this section, we describe properties of the Keccak state mappings that are at the heart of our cryptanalysis.

4.1 Preliminary properties and definitions

4.1.1 About ρ

As stated in [BDPA11b], the mapping ρ consists of translations within the lanes: its effect is independent on each lane. Therefore, a zero difference between two states in a lane at

the input of ρ is equivalent to a zero difference at the output. Let $A, A' \in \mathbb{F}_2^b$,

$$\forall 0 \leq i, j < 5, \rho(A)[i, j, *] = \rho(A')[i, j, *] \iff A[i, j, *] = A'[i, j, *]\,. \tag{3}$$

4.1.2 About χ

As stated in [BDPA11b], χ can be seen as the parallel application of 5ω S-boxes operating on rows. Therefore, a zero difference between two states in a row at the input of χ is equivalent to a zero difference at the output. In the case of collisions, this means that having a collision on a full row before χ is equivalent to have a collision after χ.

4.1.3 About π

As stated in [BDPA11b], the mapping π consists in a reorganization of the lanes of the state. Therefore, a zero difference in a lane at the output π can be easily traced back to a zero difference at the input of π.

Further, π and π^{-1} operate on the coordinates (x, y) in a linear way. The linear map is such that two lanes of a state which belong to the same plane are mapped by π^{-1} to two different sheets. Indeed, let $A \in \mathbb{F}_2^b$, let $0 \leq y_0 < 5$, $0 \leq z_0 < \omega$, such that $A[*, y_0, *]$ is a plane of the state. For any $0 \leq x \neq x' < 5$, $\pi^{-1}(A)[x, y_0, *] = A[x + 3y_0, x, *]$ and $\pi^{-1}(A)[x', y_0, *] = A[x' + 3y_0, x', *]$ are located in two different sheets since $x \neq x'$.

4.2 Understanding inner collisions

Let Keccak-$p[b, n_r]$ be a Keccak-p permutation with capacity c and bitrate r. Recall that $\omega = \frac{b}{25}$. We study what it means for two states $A, A' \in \mathbb{F}_2^b$ to be such that $\widehat{f(A)} = \widehat{f(A')}$ so as to respect condition (2) of Theorem 1. We use the following notations to analyze Keccak round function. We denote the output of the $(i+1)$-st round by A^{i+1}, where $0 \leq i < n_r$, and the initial state by A^0 or A. We define A^i_θ, A^i_ρ, A^i_π, A^i_χ as follows.

$$A^i \xrightarrow{\theta} A^i_\theta \xrightarrow{\rho} A^i_\rho \xrightarrow{\pi} A^i_\pi \xrightarrow{\chi} A^i_\chi \xrightarrow{\iota} A^{i+1}$$

Note that A^{n_r} is the same as $f(A)$.

4.2.1 The alternative inner state

In this section, we define a set of bit positions, such that having a collision on those bits before the last application of π is equivalent to having an inner collision when the inner state is made of full planes. This set of positions defines, for a state S an *alternative inner state*. The alternative inner state of any $S \in \mathbb{F}_2^b$ corresponds to the lanes that will be reorganized by π into the inner state of A^{nr-1}_π.

Definition 5 (Alternative inner state)**.** The alternative inner state is made of the bits such that their coordinates are in the set

$$\mathcal{AIS} = \{(x, y, z) \mid (x + 3y, x, z) \in \mathcal{IS}\}\,.$$

The alternative inner state has the following important property.

Proposition 1. *Let $A, A' \in \mathbb{F}_2^b$. Suppose 5ω divides c. A collision on the alternative inner state of $A^{n_r-1}_\theta$ and $(A')^{n_r-1}_\theta$ is equivalent to $\widehat{f(A)} = \widehat{f(A')}$.*

Proof. Let $A, A' \in \mathbb{F}_2^b$ such that $\widehat{f(A)} = \widehat{f(A')}$. Since 5ω divides c, the inner state is made of $\frac{c}{5\omega}$ planes. Let $0 \leq y_0 < 5$ such that the plane of coordinate $y = y_0$ is in the inner state. We have the following equivalence

$$f(A)[*, y_0, *] = f(A')[*, y_0, *] \iff A^{n_r-1}_\pi[*, y_0, *] = (A')^{n_r-1}_\pi[*, y_0, *]\,. \tag{4}$$

Indeed, ι does not affect the difference between two states, and as shown in Section 4.1.2, a zero difference on a row before the application of χ is equivalent to a zero difference after. Further, (4) is also equivalent to

$$A_\rho^{n_r-1}[x,y,z] = (A')_\rho^{n_r-1}[x,y,z] \text{ for any } (x,y,z) \in \mathcal{IS}\,.$$

In other words, (4) corresponds to a zero difference between the bits of the alternative inner state of $A_\rho^{n_r-1}$ and $(A')_\rho^{n_r-1}$. Lastly, since a zero difference between two states in a lane at the input of ρ is equivalent to a zero difference at the output and since the alternative inner state contains only full lanes, this is also equivalent to

$$A_\theta^{n_r-1}[x,y,z] = (A')_\theta^{n_r-1}[x,y,z] \text{ for any } (x,y,z) \in \mathcal{IS}\,.$$

□

We illustrate Proposition 1 on one slice when the outer state is only one plane. It is the case for example in KECCAK[40, 160]. In this case, $\mathcal{AIS} = \{(x,y,z)|x \neq y\}$.

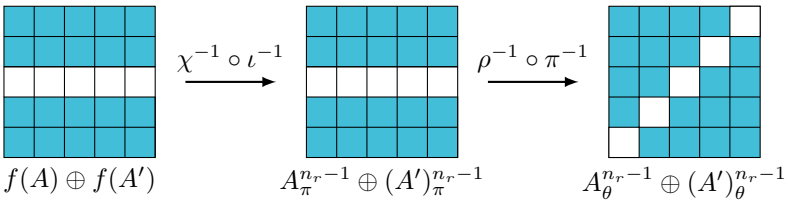

Figure 3: Illustration of Proposition 1 on one slice when the outer state is one plane.

4.2.2 Avoiding θ diffusion

Effectively, defining the alternative inner state has allowed us to work on $\theta \circ R^{n_r-1}$ instead of $f = R^{n_r}$. We have gained almost one round with a probability of 1 for states with a convenient inner state. θ however is not so easy to 'reverse' since it neither consists in bit reorganization nor is a permutation on a substructure of the state. We still managed to limit its diffusion effect thanks to the following theorem.

Theorem 2. *The sum of two bits located in the same column after θ is equal to the sum of the same two bits before θ. More precisely, let $A \in \mathbb{F}_2^b$ be any state, let $0 \leq x < 5$ and $0 \leq z < \omega$. Let $A^i[x,*,z]$ and $A_\theta^i[x,*,z]$, $0 \leq i < n_r$, be a column before and after applying θ. Then*

$$A_\theta^i[x,y,z] \oplus A_\theta^i[x,y',z] = A^i[x,y,z] \oplus A^i[x,y',z] \text{ for any } 0 \leq y,y' < 5\,.$$

Proof. Let $0 \leq x,y,y' < 5$ and $0 \leq z < \omega$. We have

$$\begin{aligned} A_\theta^i[x,y,z] \oplus A_\theta^i[x,y',z] &= A^i[x,y,z] \oplus \bigoplus_{l=0}^{4}(A^i[x-1,l,z] \oplus A^i[x-1,l,z-1]) \\ &\quad \oplus A^i[x,y',z] \oplus \bigoplus_{l=0}^{4}(A^i[x-1,l,z] \oplus A^i[x-1,l,z-1]) \\ &= A^i[x,y,z] \oplus A^i[x,y',z]\,. \end{aligned}$$

□

The next theorem is at the core of our attack. It is a direct consequence of Theorem 2.

Theorem 3. *If 5ω divides c and $\widehat{f(A)} = \widehat{f(A')}$, then $A_\chi^{n_r-2} \oplus (A')_\chi^{n_r-2}$ is constant on the bits of each column that are located in the alternative inner state.*

Proof. Let A, A' such that $\widehat{f(A)} = \widehat{f(A')}$. Let $0 \leq z < \omega$, $0 \leq x, y < 5$ so that $A[x, y, z]$ is located in the alternative inner state. From proposition 1, we deduce:

$$A_\theta^{n_r-1}[x, y, z] = (A')_\theta^{n_r-1}[x, y, z]\,. \tag{5}$$

When the inner state is made of planes, that is when 5ω divides c, each columns contains exactly $\frac{c}{5\omega}$ bits located on the alternative inner state. This is because the inner state is then made of $\frac{c}{5\omega}$ planes, and each lane of each plane will be mapped to a different sheet by π^{-1} as explained in Section 4.1.3. When $\frac{c}{5\omega} = 1$, each columns contains a single bit located on the alternative inner state. Therefore the proof of theorem is trivial in that case. Let us now focus on the case $\frac{c}{5\omega} > 1$.

Let $b_0, b_1, \ldots, b_4$ (resp. $b'_0, b'_1, \ldots, b'_4$) be five bits of a column of $A_\theta^{n_r-1}$ (resp. $(A')_\theta^{n_r-1}$). Let $a_0, a_1, \ldots, a_4$ (resp. $a'_0, a'_1, \ldots, a'_4$) be five bits of the same column of $A_\chi^{n_r-2}$ (resp. $(A')_\chi^{n_r-2}$). In the following, we assume that $\frac{c}{5\omega} = 4$ and that b_0 is the only bit of the column that is not on the inner state. This is so as to have convenient notations, but our proof is exactly the same for any other value of $\frac{c}{5\omega} > 1$. By (5), we get:

$$\begin{cases} b_1 = b'_1 \\ b_2 = b'_2 \\ b_3 = b'_3 \\ b_4 = b'_4 \end{cases}$$

which is equivalent to

$$\begin{cases} b_1 = b'_1 \\ b_1 \oplus b_2 = b'_1 \oplus b'_2 \\ b_2 \oplus b_3 = b'_2 \oplus b'_3 \\ b_3 \oplus b_4 = b'_3 \oplus b'_4\,. \end{cases}$$

From Proposition 2, it comes that

$$\begin{cases} b_1 \oplus b_2 = b'_1 \oplus b'_2 \\ b_2 \oplus b_3 = b'_2 \oplus b'_3 \\ b_3 \oplus b_4 = b'_3 \oplus b'_4 \end{cases} \text{ is equivalent to } \begin{cases} a_1 \oplus a_2 = a'_1 \oplus a'_2 \\ a_2 \oplus a_3 = a'_2 \oplus a'_3 \\ a_3 \oplus a_4 = a'_3 \oplus a'_4 \end{cases}$$

which is also equivalent to

$$a_1 \oplus a'_1 = a_2 \oplus a'_2 = a_3 \oplus a'_3 = a_4 \oplus a'_4\,.$$

It comes that $A^{n_r-2} \oplus (A')^{n_r-2}$ must be constant on $\frac{c}{5\omega}$ bits of each column. Since ι does not affect the value of this difference, this necessary condition already applies before ι. Thus $A_\chi^{n_r-2} \oplus (A')_\chi^{n_r-2}$ must be constant on $\frac{c}{5\omega}$ bits of each column. □

Constancy on columns on bits of the alternative inner state of $A_\chi^{n_r-2} \oplus (A')_\chi^{n_r-2}$ is thus a necessary condition for two states to present an inner collision. We will show in Section 8 that this can also be adapted when the outer part is not exactly full planes as long as the inner part contains at least two full planes. In Figure 4 we show how the difference between two states presenting an inner collision should therefore look like after χ on a slice.

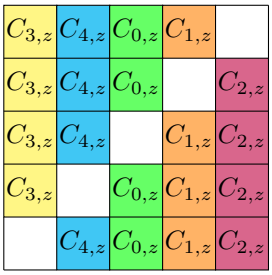

Figure 4: The difference between two inner colliding states after χ when the inner state is one plane. We call the constant for each column $C_{x,z}$, $0 \leq x < 5$, $0 \leq z < \omega$. The anti-diagonal can be set to any value.

5 Choosing the subset for the squeeze attack

The subset X which will be used for our birthday squeeze attack is comprised of inner states $\widehat{S} \in \mathbb{F}_2^c$ such that on several columns of $A_\chi^{n_r-2}$, some pre-determined bits are all equal to a constant. In this section, we will demonstrate why this choice of subset is relevant thanks to results from Section 4.

Recall that after computing S from a path P, Step 3 consists in finding M given $\widehat{S}$ such that $f(M||\widehat{S})$ belongs X. To do so, we start again from analyzing inner collisions. We study what it means for M, M', S, S' to be such that the inner states of $f(M||\widehat{S})$ and $f(M'||\widehat{S'})$ are equal.

In the following, we denote the r bits of M (respectively M') by $m_0, m_1, \ldots, m_{r-1}$ (resp. $m'_0, m'_1, \ldots, m'_{r-1}$) and the c bits of $\widehat{S}$ (resp. $\widehat{S'}$) by $s_0, s_1, \ldots, s_{c-1}$ (resp. $s'_0, s'_1, \ldots, s'_{c-1}$). Finding an inner collision is equivalent to solving a system $\mathcal{S}$ of c equations that depend on the bits of M, M', S and S'.

$$\begin{cases} f_0(m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = f_0(m'_0, \ldots, m'_{r-1}, s'_0, \ldots, s'_{c-1}) \\ f_1(m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = f_1(m'_0, \ldots, m'_{r-1}, s'_0, \ldots, s'_{c-1}) \\ \ldots \\ f_{c-1}(m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = f_{c-1}(m'_0, \ldots, m'_{r-1}, s'_0, \ldots, s'_{c-1}) \end{cases} \quad (\mathcal{S})$$

where the f_i for i from 0 to $c-1$ are the component functions of f.

Recall that we assume that c is a multiple of 5ω. By Proposition 1, under this condition, an inner collision is equivalent to a collision on the alternative inner state of $A_\theta^{n_r-1}$ and $(A')_\theta^{n_r-1}$. We also showed in the demonstration of Theorem 3 that each column of $A_\theta^{n_r-1}$ contains exactly $\frac{c}{5\omega}$ bits of the alternative inner state. Since there are 5ω columns in a state, the alternative inner state contains exactly $5\omega \times \frac{c}{5\omega} = c$ bits. If we denote by F the permutation $\theta \circ R^{n_r-1}$, it comes that $\mathcal{S}$ is equivalent to a system $\mathcal{S}'$ of the form:

$$\begin{cases} F_0(m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = F_0(m'_0, \ldots, m'_{r-1}, s'_0, \ldots, s'_{c-1}) \\ F_1(m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = F_1(m'_0, \ldots, m'_{r-1}, s'_0, \ldots, s'_{c-1}) \\ \ldots \\ F_{c-1}(m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = F_{c-1}(m'_0, \ldots, m'_{r-1}, s'_0, \ldots, s'_{c-1}) \,. \end{cases} \quad (\mathcal{S}')$$

where the F_i for i from 0 to $c-1$ are the component functions of $F = \theta \circ R^{n_r-1}$.

Proposition 2. *Let $A = M||\widehat{S}, A' = M'||\widehat{S'}$. $A_\chi^{n_r-2} \oplus (A')_\chi^{n_r-2}$ being constant on $k > 1$ bits of a column located in the alternative inner state is equivalent to satisfying $k-1$ equations of $\mathcal{S}'$.*

Proof. Let $A = M||\widehat{S}, A' = M'||\widehat{S}'$. As in the proof of Theorem 3, let $b_0, b_1, \ldots, b_4$ (resp. $b'_0, b'_1, \ldots, b'_4$) be five bits of a column of $A_\theta^{n_r-1}$ (resp. $(A')_\theta^{n_r-1}$). Let $a_0, a_1, \ldots, a_4$ (resp. $a'_0, a'_1, \ldots, a'_4$) be five bits of the same column of $A_\chi^{n_r-2}$ (resp. $(A')_\chi^{n_r-2}$). Let k be the number of constant bits of a column located in the alternative inner state of $A_\chi^{n_r-1} \oplus (A')_\chi^{n_r-1}$. We assume without loss of generality that these bits are located next to each other.

Let $0 < i \leq 4$ such that $a_{i+j} \oplus a'_{i+j} = a_{i+j+1} \oplus a'_{i+j+1}$ for $0 \leq j \leq k-2$. This is equivalent to

$$a_{i+j} \oplus a_{i+j+1} = a'_{i+j} \oplus a'_{i+j+1} \quad \text{for} \quad 0 \leq j \leq k-2 \tag{6}$$

which is in turn equivalent to

$$b_{i+j} \oplus b_{i+j+1} = b'_{i+j} \oplus b'_{i+j+1} \quad \text{for} \quad 0 \leq j \leq k-2$$

which is equivalent to the $k-1$ last lines of

$$\begin{cases} b_i = b'_i \\ b_i \oplus b_{i+1} = b'_i \oplus b'_{i+1} \\ \cdots \\ b_{i+k-2} \oplus b_{i+k-1} = b'_{i+k-2} \oplus b'_{i+k-1} \end{cases}$$

which is equivalent to the $k-1$ last lines of

$$\begin{cases} b_i = b'_i \\ b_{i+1} = b'_{i+1} \\ \cdots \\ b_{i+k-1} = b'_{i+k-1}\,. \end{cases} \tag{7}$$

System (7) is equivalent to k equations of $\mathcal{S}'$ since it effectively represents a collision on k bits of the alternative inner state of $A_\theta^{n_r-1}$ and $(A')_\theta^{n_r-1}$. Satisfying (6) is thus equivalent to satisfying $k-1$ equations of $\mathcal{S}'$. □

Proposition 2 shows that a good strategy is to produce states between which the difference on certain columns is constant on 2, 3 or 4 bits of the alternative inner state after χ since it is equivalent to satisfying some equations of $\mathcal{S}'$. However one must be cautious in the production of these states: to ensure the applicability of the birthday argument, each new state that we produce must satisfy the difference constancy with *all* states already produced. Then, let n be the number of equations of the system that are automatically satisfied by any pair of states produced. The size of the predetermined subset X where we send our states is thus 2^{c-n}. Hence, a collision can be found by producing about $2^{\frac{c-n}{2}} < 2^{\frac{c}{2}}$ such states, therefore improving the memory complexity of a simple birthday attack.

We decided to produce states that are constant on certain columns. Indeed, if two states are constant on a given column, then their difference will also be constant on this column. This choice is arbitrary, and we could have chosen any other 'pattern' than constancy.

6 S-box linearization

Whilst the analysis we have provided so far is general, the following is specific to two rounds Keccak functions. We narrow down our analysis to Keccak functions with $n_r = 2$. The notations from Section 4.2 can be simplified. We only wish to linearize the

first round since the necessary condition of constancy on columns applies after the first application of χ. We denote the initial state by A^0 or A. We define A_θ, A_ρ, A_π, A_χ as follows :

$$A \xrightarrow{\theta} A_\theta \xrightarrow{\rho} A_\rho \xrightarrow{\pi} A_\pi \xrightarrow{\chi} A_\chi \,.$$

Producing states that are constant on certain columns after χ corresponds to solving a system of the form $A_\chi[x, y, z](m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = C_{x,z}$ for a number of specific x, y, z. Yet χ is not linear, making this system *a priori* very hard to solve. We overcome this difficulty by linearizing χ thanks to a technique that we will detail in this section. It is inspired by methods that are usually employed in pre-image attacks or in a keyed setting analysis [GLS16, QSLG17, LSLW17, SLG17, DLWQ17, SGSL18, FNR18, KRA18, KMS18, LS19, Raj19, GLL+20]. The main idea is to construct a linear system $\mathcal{L}$ such that satisfying $\mathcal{L}$ is equivalent to satisfying as many equations of the non-linear system $\mathcal{S}'$ as possible.

6.1 Well-known properties of χ

To linearize χ, one must first recall a number of its properties that can be naturally derived from the observations made in [Dae95]. Proposition 3 is also present in [GLS16] and Proposition 4 could be derived from the observations in [GLS16]. Let $A \in \mathbb{F}_2^b$. Let $0 \leq y < 5$ and $0 \leq z < \omega$. Let $A_\rho[*, y, z] = (c_0, c_1, \ldots, c_4)$ and $A_\chi[*, y, z] = (d_0, d_1, \ldots, d_4)$, $0 \leq i < n_r$ be a row before and after applying χ.

Proposition 3. *For any $0 \leq j < 5$, $c_j = d_j$ holds with probability 0.75.*

Proof. For any $0 \leq j < 5$, $d_j = c_j \oplus (\neg c_{j+1}) \wedge c_{j+2}$ and $(\neg c_{j+1}) \wedge c_{j+2} = 0$ holds with probability 0.75. □

Proposition 4. *For any $0 \leq j < 5$, if c_j is known, then d_{j-1} and d_{j-2} can be written as linear expressions of the other c_k, $k \neq j$.*

Proof. This comes from:

$$d_{j-1} = c_{j-1} \oplus (\neg c_j) \wedge c_{j+1} \,,$$
$$d_{j-2} = c_{j-2} \oplus (\neg c_{j-1}) \wedge c_j \,.$$

When c_j is known, d_{j-1} and d_{j-2} are linear expressions of the other c_k, $k \neq j$. □

6.2 Basic linearization technique

We wish to construct a linear system $\mathcal{L}$ such that satisfying $\mathcal{L}$ is equivalent to satisfying as many equations of $\mathcal{S}'$ as possible. A linear system has a solution with high probability if the number of variables is equal to the number of equations. We do not control the s_i since they correspond to the value of the inner state of random absorbed bit strings. On the other hand, we can choose the value of M. We thus have r degrees of freedom (or variables), the m_i. In practice we might have more than r equations, in which case we will satisfy the system with a probability $p < 1$.

In order to construct $\mathcal{L}$, we will mainly use the result from Proposition 4. We call *fixing* a bit the allocation of a set value to a bit. After producing random inner states $\widehat{S}$, we fix bits of A_π so as to obtain a linear expression of bits of A_χ in terms of bits of A_π. Since the three first mappings of the round are linear, the bits of A_π in turn depend linearly on the bits of A. We can therefore efficiently linearize two rounds of Keccak. We give examples in the next section.

We denote by *allocation strategy* the set of decisions consisting in choosing which bits to fix. In general, defining an allocation strategy is not trivial. It depends on the

parameters c, r of the KECCAK function we wish to attack. We will give examples of allocation strategies when the outer state is one plane, as it is the case for example for KECCAK[40, 160] reduced to two rounds. Even though the examples we provide must be carefully adapted to each function, our examples give a good overview of how to define a smart allocation strategy.

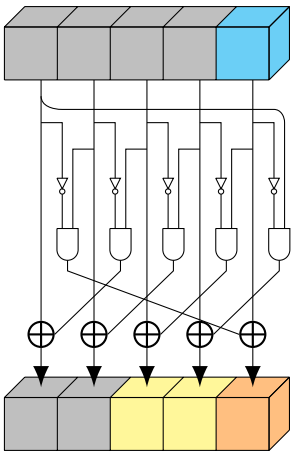

Figure 5: Fixing the bit in blue before χ enables one to obtain a linear expression of each bit in yellow. The bit in blue has the same value as the bit in orange with probability 0.75.

6.3 Allocation strategies on a slice

We start by working on a slice of the KECCAK state, that is a 5×5 array of the form $A[*, *, z]$ where $0 \leq z < \omega$. Indeed, to fix bits in a smart way, one needs to start by considering each slice of the state independently. Considering each column is too narrow because fixing a bit linearizes two other bits *in its row*. Considering each row is inadequate since we want to reach constancy *on columns*.

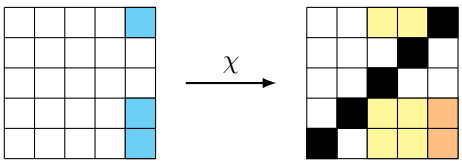

Figure 6: Fixing 3 bits on a slice before χ. The blue bits are fixed so as to linearize the expression of the yellow bits. The orange bits are equal to the blue bits with probability 0.75. The black bits do not matter, as those will go to the outer part after applying the remaining step mappings.

Example 1 (3 bits per slice)**.** We start with an empty linear system $\mathcal{L}$. We fix 3 bits in a slice of A_π, all located in the same column. Since the three first mappings are linear, each bit of A_π depends linearly on the m_i, $0 \leq i < r$. As illustrated in Figure 6, we carefully pick these three bits so that the bits of which we will get a linear expression are located on the alternative inner state. We thus add three linear equations to our linear system $\mathcal{L}$, they correspond to the expression of these three bits equal to a constant. In the example

corresponding to Figure 6, the equations added to $\mathcal{L}$ are of the following form:

$$\begin{aligned} A_\pi[2,2,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= D_2 \\ A_\pi[2,3,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= D_3 \\ A_\pi[2,4,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= D_4\,, \end{aligned}$$

where D_2, D_3 and D_4 are arbitrary constants.

Now, we know the value of three bits of A_π. We obtain a linear expression of 3 bits of two columns, the 6 yellow bits in Figure 6. We want to allocate a common value to bits of the same column. At first sight, the smart strategy is to add the linear expression of these bits to our linear system equal to a common value. In the example corresponding to 6, the equations added to $\mathcal{L}$ would be of the following form for the column $x = 1$:

$$\begin{aligned} A_\chi[1,2,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= C_{1,0} \\ A_\chi[1,3,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= C_{1,0} \\ A_\chi[1,4,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= C_{1,0}\,. \end{aligned}$$

Yet the choice of the value of $C_{1,0}$ would be arbitrary, which increases the complexity of our attack. Instead, we add the following equations:

$$\begin{aligned} A_\chi[1,2,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) \oplus A_\chi[1,3,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= 0 \\ A_\chi[1,3,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) \oplus A_\chi[1,4,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) &= 0\,. \end{aligned}$$

For each 'linearized' column, we thus add 2 linear equations to $\mathcal{L}$. In total, we thus add $3 + 2 \times 2 = 7$ equations to $\mathcal{L}$. If we solve this system, we achieve constancy on three bits of two columns of the state. By Proposition 2, we thereby satisfy $2 \times 2 = 4$ equations of $\mathcal{S}'$.

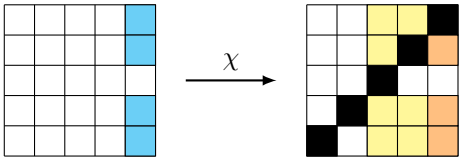

Figure 7: Fixing 4 bits on a slice before χ. The blue bits are fixed so as to linearize the expression of the yellow bits. The orange bits are equal to the bits in blue with probability 0.75. The black bits are not of interest.

Example 2 (4 bits per slice)**.** We start with an empty linear system $\mathcal{L}$. We fix 4 bits in a slice of A_π, all located in the same column. We thus add 4 linear equations to our linear system $\mathcal{L}$, they correspond to the expression of these four bits equal a constant. In the example corresponding to Figure 7, the equations added to $\mathcal{L}$ are of the following form:

$$A_\pi[2,i,0](m_0,\ldots,m_{r-1},s_0,\ldots,s_{c-1}) = D_i \qquad \text{for} \qquad 1 \le i < 5\,,$$

where D_i, $1 \le i < 5$ are arbitrary constants.

Now, we know the value of four bits of A_π. We obtain a linear expression of 4 bits of two columns. As illustrated in Figure 7, for one of these two columns, only 3 bits are of interest since the fourth one is not located on the alternative inner state. These $3 + 4 = 7$ bits are in yellow in Figure 7. We want to allocate a common value to bits of the same column. As in the previous example, we do not care about their actual value. Thus for each column where we have obtained the expression of k bits, we only need adding $k - 1$ equations to our system to ensure constancy. We thus add a total of $3 - 1 + 4 - 1 = 5$ equations to our system.

In total, we thus added $4 + 5 = 9$ equations to our system. If we solve this system, we achieve constancy on 3 bits of a column of the state, and 4 on another column. By Proposition 2, we thereby satisfy $2 + 3 = 5$ equations of $\mathcal{S}'$.

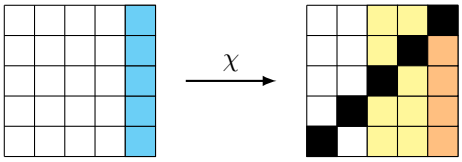

Figure 8: Fixing 5 bits on a slice before χ. The blue bits are fixed so as to linearize the expression of yellow bits. The orange bits are equal to the blue bits with probability 0.75. The black bits are not of interest.

Example 3 (5 bits per slice)**.** We start with an empty linear system $\mathcal{L}$. We fix 5 bits in a slice of A_π, all located in the same column. We thus add 5 linear equations to our linear system $\mathcal{L}$, they correspond to the expression of these four bits equal a constant. In the example corresponding to Figure 8, the equations added to $\mathcal{L}$ are of the following form:

$$A_\pi[2, i, 0](m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = D_i \qquad \text{for} \qquad 0 \leq i < 5\,,$$

where D_i, $0 \leq i < 5$ are arbitrary constants.

Now we know the value of five bits of A_π. We obtain a linear expression of 5 bits of two columns. As illustrated in Figure 8, for both of these two columns, only 4 bits are of interest since the fifth one is not located on the alternative inner state. These $4 + 4 = 8$ bits are in yellow in Figure 8. We want to allocate a common value to bits of the same column. As in the previous example, we do not care about their actual value. Thus for each column where we have obtained the expression of k bits, we only need adding $k - 1$ equations to our system to ensure constancy. We thus add a total of $2 \times (4 - 1) = 6$ equations to our system.

In total, we thus added $5 + 6 = 11$ equations to our system. If we solve this system, we achieve constancy on 4 bits of two columns of the state. By Proposition 2, we thereby satisfy $2 \times 3 = 6$ equations of $\mathcal{S}'$.

Example 4 (2 bits per slice)**.** We start with an empty linear system $\mathcal{L}$. We fix 2 bits in a slice of A_π, all located in the same column. We thus add 2 linear equations to our linear system $\mathcal{L}$, they correspond to the expression of these two bits equal a constant. In the example corresponding to Figure 8, the equations added to $\mathcal{L}$ are of the following form:

$$A_\pi[2, i, 0](m_0, \ldots, m_{r-1}, s_0, \ldots, s_{c-1}) = D_i \qquad \text{for} \qquad 3 \leq i < 5$$

where D_i, $3 \leq i < 5$ are arbitrary constants.

Now we know the value of five bits of A_π. We obtain a linear expression of 2 bits of two columns. We want to allocate a common value to bits of the same column. Using the same reasoning as above, we add $2 \times (2 - 1) = 2$ equations to our system.

In total, we thus added $2 + 2 = 4$ equations to our system. If we solve this system, we achieve constancy on 2 bits of two columns of the state. By Proposition 2, we thereby satisfy $2 \times 1 = 2$ equations of $\mathcal{S}'$.

Summary of the allocation strategies on a slice

We define ν_{slice} to be the ratio between the number of equations of $\mathcal{S}'$ that are satisfied per equation added to $\mathcal{L}$. In Table 2 are presented the different ν_{slice} for states where the outer state is one slice. It comes that the best strategy is to maximize the number n_3 of slices where we allocate 3 bits. This is inherently linked to the fact that the outer state is one plane, and is not to be taken as a general statement for Keccak sponges.

Table 2: Different choices for the number of fixed bits per slice when the outer state is made of one plane

Number of fixed bits	2	3	4	5
Number of equations added to $\mathcal{L}$	4	7	9	11
Number of equations satisfied in $\mathcal{S}'$	2	4	5	6
Ratio ν_{slice}	0.5	0.57	0.56	0.55

6.4 Allocation strategies on a state

Keeping Table 2 in mind, we present different allocation strategies at a state scale. Allocation strategies at the state scale, just like allocation strategies at the slice scale, are tightly linked to the specific parameters r, c, ω of each sponge function. Again, we give examples of strategies for KECCAK$[40, 160]$ reduced to two rounds. The examples provide a good overview of how to strategically fix bits.

A note on time complexity

Finding solutions to $\mathcal{L}$ allows us to send states into a predetermined subset X of $\mathbb{F}_2^c$ of size 2^{c-n}, where n is the number of equations of $\mathcal{S}'$ every state produced satisfies. We deduce that our memory complexity is $2^{\frac{c-n}{2}}$. Now, to compute the time complexity, we need to determine more precisely how much it costs to produce one state that satisfies n equations of $\mathcal{S}'$. Let e be the number of equations in $\mathcal{L}$. In a general setting, we have r variables. We naturally always have $e \geq \mathsf{rank}(\mathcal{L})$ and $r \geq \mathsf{rank}(\mathcal{L})$. The probability of finding a solution to $\mathcal{L}$ is $2^{\mathsf{rank}(\mathcal{L})-e}$. Once a solution to $\mathcal{L}$ is found, we obtain easily $2^{r-\mathsf{rank}(\mathcal{L})} - 1$ other solutions, and thus in total $2^{r-\mathsf{rank}(\mathcal{L})}$ different possible states that go into the desired subset X. Hence, regardless of the rank of the system, each produced random state gives us on average 2^{r-e} possible states that go into X. Letting g be the complexity of the Gaussian elimination, it comes that the time complexity equals the memory complexity multiplied by $2^{e-r} \times g$. We will show later on that we can replace g by a smaller coefficient.

Example 5. Let n_i be the number of slices where we fix i bits. Since the greatest ν_{slice} is associated to fixing 3 bits per slice, it seems that the best strategy is to fix 3 bits on as many slices as possible. On each slice, fixing 3 bits means adding a total of 7 equations to our system. The greatest a such that $7a \leq r$ is $a = 5$. We thus set $n_3 = 5$. We thereby add $7 \times 5 = 35$ linear equations to our system. For the remaining 5 equations that we can add to our system, we can fix 2 bits on $n_2 = 1$ slice, thereby adding 4 equations to $\mathcal{L}$. By solving $\mathcal{L}$, we satisfy $4n_3 + 2n_2 = 20 + 2 = 22$ equations of $\mathcal{S}'$. The subset of $\mathbb{F}_2^c$ where we send every state is thus of size $2^{c-22} = 2^{138}$. Our memory complexity is thus of $2^{\frac{138}{2}} = 2^{69}$.

Since $e = 39$ and $r = 40$, we need to find a solution to the linear system 2^{68} times. Our time complexity is thus of $2^{68} \times g$.

Example 6. We set $n_3 = 6$. In this case, we must solve a system of $e = 42$ equations. By solving $\mathcal{L}$, we satisfy $4n_3 = 24$ equations of $\mathcal{S}'$. The subset of $\mathbb{F}_2^c$ where we send every state is of size $2^{c-24} = 2^{136}$. Our memory complexity is thus of $2^{\frac{136}{2}} = 2^{68}$. As for the time complexity, since $e - r = 2$ and since we need to produce 2^{68} states, our time complexity is of $2^{68} \times 2^2 \times g = 2^{70} \times g$. In this example, we gained a factor 2 in memory complexity but lost a factor 2^2 in time complexity compared to Example 5.

Example 7. We could think that it is interesting to try and have our number of equations strictly equal to the number of variables. For example, we can decide to fix 4 bits of $n_4 = 4$ slices, thereby adding $4 \times 9 = 36$ equations to our system, and further 2 bits on $n_2 = 1$ slice, thereby adding 4 equations to our system. We obtain a system of 40 equations. Yet this is not optimal in terms of complexity. Solving this system, we satisfy $5n_4 + 2n_2 = 22$ equations of $\mathcal{S}'$, as in Example 5. We thus have the same memory complexity. Yet this time, since $e - r = 0$ rather than -1, our time complexity is greater than in Example 5 by a factor 2.

Example 6 offered an example of time-memory trade-off. Example 7 has shown us that it is not optimal in general to aim for a perfectly balanced system in terms of number of variables and equations. In the next section we will show how to slightly improve the complexities of the attack, by using simple properties of χ.

7 Improvements

In this section, we describe various ways of improving and/or optimising our attack. We also show that our attack can be adapted to the attacker's needs thanks to time-memory trade-offs.

7.1 Generic time-memory trade-off

When doing a birthday squeeze attack, it is always possible to do time-memory trade-offs. On our example of allocation on a slice, we only worried about constancy on two columns. We could decide that on another arbitrary column of the slice, we require constancy on k bits as well, for $0 \leq k < 5$. After having produced the state $A = M||\widehat{S}$, we check if it meets the constancy requirement on this k extra bits, and if it does not, we discard it.

The probability that A fulfills this requirement is $p = 2^{-(k-1)} < 1$, but it allows us to create a subspace of pairs that all satisfy $k-1$ extra equations of $\mathcal{S}'$. Satisfying an extra $k-1$ equations of $\mathcal{S}'$ improves our memory complexity by $2^{\frac{k-1}{2}}$ (it is multiplied by $2^{\frac{1-k}{2}}$). Since we need to produce less states, our time complexity is also improved by $2^{\frac{k-1}{2}}$. However, it is more costly to produce one state, and thus we also multiply it by $\frac{1}{p}$. In total, our time complexity is multiplied by $\frac{1}{p} \times 2^{\frac{1-k}{2}} = 2^{k-1-\frac{k-1}{2}} = 2^{\frac{k-1}{2}}$.

This extends easily at a state scale. If we require this constancy on an extra column on n'_k slices, the probability that A will be kept is $p' = (2^{-(k-1)})^{n'_k} = 2^{n'_k \times (1-k)}$, yet it allows us to create a subspace of pairs that all satisfy $n'_k \times (k-1)$ extra equations of $\mathcal{S}'$. Again, our memory complexity is improved by $2^{n'_k \frac{(k-1)}{2}}$. Yet our time complexity is multiplied by $2^{n'_k \frac{(1-k)}{2}} \times \frac{1}{p'} = 2^{n'_k \frac{(k-1)}{2}}$.

7.2 Improvement of the time-memory trade-off

When we build our system $\mathcal{L}$, we fix some bits of the states (the blue bits in Figure 6, 7 and 8). By Proposition 3, the value of each orange bit after the χ mapping (also in Figures 6, 7 and 8) is equal to the value of a corresponding blue bit before the χ mapping with probability 0.75. Hence, the value of the blue bits after they've been fixed can play a role in the complexity of the attacks. In this section, we will show how to improve the generic time-memory trade-off thanks to Proposition 3 and those improvements will help us to slightly improve the attack presented in Examples 5, 6 and 7.

In Example 5 of Section 6, 2 bits that we fixed are located on the alternative inner state, that is $A_\pi[2,3,0]$ and $A_\pi[2,4,0]$. We set these two bits to a constant by adding the two following equations to our system $\mathcal{L}$:

$$A_\pi[2,3,0](m_0,\dots,m_{r-1},s_0,\dots,s_{c-1}) = D_3$$
$$A_\pi[2,4,0](m_0,\dots,m_{r-1},s_0,\dots,s_{c-1}) = D_4$$

where D_3 and D_4 are arbitrary constants.

By Proposition 3, for each $i = 3, 4$, we have $A_\chi[2,i,0] = D_i$ with a probability $p = \frac{3}{4}$. Thus, if we allocate the same value to both bits (we choose D_3 and D_4 such that $D_3 = D_4$), there is a probability $p = \left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2 = \frac{5}{8} > \frac{1}{2}$ that $A_\chi[2,3,0] = A_\chi[2,4,0]$. Thus, if we do so on n_3 slices, there is a probability $p = \left(\frac{5}{8}\right)^{n_3}$ that A_χ is constant on two bits of the column $y = 2$ of these n_3 slices.

More generally, if we allocate the same value to k bits of the alternative inner state before χ, there is a probability $p = \left(\frac{3}{4}\right)^k + \left(\frac{1}{4}\right)^k$ that they will also have the same value after χ, which allows us to satisfy extra equations of $\mathcal{S}'$. It is thus smarter to allocate the same value to blue bits (in the figures) of the same columns. We could thus decide to only keep the states $A = M||\widehat{S}$ such that they are equal on these 2 bits, and thereby improve the memory complexity of the previous examples as follows.

Example 8. We set $n_3 = 5$. We thereby add $7 \times 5 = 35$ linear equations to our system. For the remaining 5 equations that we can add to our system, we fix 2 bits on $n_2 = 1$ slice, thereby adding 4 equations to $\mathcal{L}$. In total $\mathcal{L}$ thus contains $e = 39$ equations. By solving $\mathcal{L}$, we satisfy $4n_3 + 2n_2 = 20 + 2 = 22$ equations of $\mathcal{S}'$. We also allocate the same value ($D_3 = D_4$) to all blue bits on each of the $n_3 + n_2$ slices, and discard any state such that the equality is not preserved by χ. This allows us to satisfy an extra $n_3 + n_2 = 6$ equations of $\mathcal{S}'$. The subset X of $\mathbb{F}_2^c$ where we send every state is thus of size $2^{c-28} = 2^{132}$. Our memory complexity is thus of $2^{\frac{132}{2}} = 2^{66}$.

As for the time complexity, producing $2^{r-e} = 2$ states that satisfy the first set of requirements costs g. Each of these states has a probability $p = \left(\frac{5}{8}\right)^{n_3+n_2} = \left(\frac{5}{8}\right)^6 \approx 2^{-4.1}$ to satisfy the extra 6 equations. Our time complexity is thus around $2^{66} \times 2^{e-r} \times 2^{4.1} \times g = 2^{69.1} \times g$.

Example 9. We set $n_4 = 4$. We thereby add $9 \times n_4 = 36$ linear equations to our system. For the remaining 4 equations that we can add to our system, we can fix 2 bits on $n_2 = 1$ slice, thereby adding 4 equations to $\mathcal{L}$. In total $\mathcal{L}$ thus contains $e = 40$ equations. By solving $\mathcal{L}$, we satisfy $5n_4 + 2n_2 = 20 + 2 = 22$ equations of $\mathcal{S}'$. Further, we allocate the same value 3 blue bits on n_4 slices, and to 2 blue bits on n_2 slice, and discard any state such that this equality is not preserved by χ. This allows us to satisfy an extra $2n_4 + n_2 = 9$ equations of $\mathcal{S}'$. The subset X of $\mathbb{F}_2^c$ where we send every state is thus of size $2^{c-31} = 2^{129}$. Our memory complexity is thus of $2^{64.5}$.

As for the time complexity, producing $2^{r-e} = 1$ state that satisfies the first set of requirements costs g. Each state has a probability

$$p = \left(\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2\right)^{n_2} \times \left(\left(\frac{3}{4}\right)^3 + \left(\frac{1}{4}\right)^3\right)^{n_3} = \frac{5}{8} \times \left(\frac{7}{16}\right)^4 \approx 2^{-5.4}$$

to satisfy the extra 9 equations of $\mathcal{S}'$. Our final time complexity is thus around $2^{64.5} \times 2^{5.4} \times g = 2^{69.9} \times g$.

Example 10. We set $n_5 = 3$. We thereby add $11 \times 3 = 33$ linear equations to our system. For the remaining 7 equations that we can add to our system, we can fix 3 bits of a slice. By solving $\mathcal{L}$, we satisfy $6n_5 + 4n_3 = 18 + 4 = 22$ equations of $\mathcal{S}'$. Further, we allocate the same value to 4 bits on n_5 slices, and to 2 bits on n_3 slice, and discard any state such that this equality is not preserved by χ. This allows us to satisfy an extra $3n_5 + n_3 = 10$

equations of $\mathcal{S}'$. The subset X of $\mathbb{F}_2^c$ where we send every state is thus of size $2^{c-32} = 2^{128}$. Our memory complexity is thus of $2^{\frac{128}{2}} = 2^{64}$.

Producing $2^{r-e} = 1$ state that satisfies the first set of requirements costs g. Each state has a probability

$$p = \left(\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2\right)^{n_3} \times \left(\left(\frac{3}{4}\right)^4 + \left(\frac{1}{4}\right)^4\right)^{n_5} = \frac{5}{8} \times \left(\frac{41}{128}\right)^3 \approx 2^{-5.6}$$

to satisfy the 10 extra equations of $\mathcal{S}'$. Our final time complexity is around $2^{64} \times 2^{5.6} \times g = 2^{69.6} \times g$.

One can notice that all the trade-offs presented in the Examples 8, 9 and 10 are worst in terms of time complexity than the original attack made in Example 5. The reason for that is the following: in the trade-offs, we have a probability of $p_1 = \frac{5}{8}$, $p_2 = \frac{7}{16}$ and $p_3 = \frac{41}{128}$ to satisfy 1, 2 and 3 equations respectively. As p_1, p_2 and p_3 are respectively smaller than $\frac{1}{\sqrt{2}}$, $\frac{1}{2}$ and $\frac{1}{2\sqrt{2}}$, our trade-offs cannot beat the first attack used in Example 5.

7.3 Optimizing the complexity

The existence of time-memory trade-offs that are better than the generic one described in Section 7.1 suggests that the way the bits are allocated (in blue in Figures 6, 7 and 8) impacts the complexity of the attack, even when the attacker does not discard states as we presented in the trade-offs. In the following, we will compute the advantage to allocate the same value to bits of the same column.

Fixing two bits on a slice in the alternative state When two bits on a slice are fixed before χ to the same value (say 00 without loss of generality), then the output on those two bits after χ is 00 with probability $\frac{9}{16}$, 01 with probability $\frac{3}{16}$, 10 with probability $\frac{3}{16}$ and 11 with probability $\frac{1}{16}$.

Let A_1 and A_2 be two states produced after allocating the same value to two blue bits of the same column and let s_1 and s_2 be the values on those two bits after χ (located in the same column and in the alternative inner state). We know from Proposition 2 that being constant on those bits, i.e. $s_1 \oplus s_2 = 00$ or 11 exactly satisfies an equation of $\mathcal{S}'$. Using the probabilities above, we find that if the bits in blue are allocated to the same value, then this equation will be satisfied, not with a probability of $\frac{1}{2}$, but with a probability of

$$\frac{9}{16} \times \frac{9}{16} + 2 \times \frac{9}{16} \times \frac{1}{16} + \frac{1}{16} \times \frac{1}{16} + 4 \times \frac{3}{16} \times \frac{3}{16} = \frac{17}{32} > \frac{1}{2}\,.$$

Fixing three bits on a slice in the alternative state Using the same reasoning, when we fix three bits, we can look at the probability of satisfying the corresponding two equations of $\mathcal{S}'$. Considering three bits in the same column with the input pattern 000, the probability that two states at the output have the same difference in this column is exactly

$$\left(\frac{27}{64}\right)^2 + 3\left(\frac{9}{64}\right)^2 + 3\left(\frac{3}{64}\right)^2 + 3 \times 2 \times \frac{3}{64} \times \frac{9}{64} + \left(\frac{1}{64}\right)^2 + \frac{27}{64} \times \frac{1}{64} \times 2 = \frac{19}{64} > \frac{1}{4}\,.$$

Fixing four bits on a slice in the alternative state When we have four bits with the input pattern 0000, we find that the probability of satisfying three equations in $\mathcal{S}'$ is

$$\left(\frac{81}{2^8}\right)^2 + \left(\frac{1}{2^8}\right)^2 + \frac{81}{2^8} \times \frac{2}{2^8} + 4\left(\frac{27}{2^8}\right)^2 + 4\left(\frac{3}{2^8}\right)^2 + 12 \times \left(\frac{9}{2^8}\right)^2 + \frac{27}{2^8} \times \frac{4 \times 2 \times 3}{2^8}$$
$$= \frac{353}{2048} > \frac{1}{8}\,.$$

In conclusion, it is always better to allocate the same value to the blue bits of the same column in the alternative inner state. The equations are then satisfied with a higher probability than in the random case. Naturally, on the other hand, allocating a different value to bits of the same column decreases the probability of getting a collision. The full complexity of our attack will be given in Section 9.

8 Attacks in the general context

So far, we have only described our attack when the outer state is exactly one plane. However, the Keccak versions can absorb more (or less) bits per Keccak-p permutations calls, for different width and rate. A user can use an arbitrary rate, as long as both the capacity and the output length are at least twice the security level this user wants to achieve. In other words, the outer part can be two planes, or it can also be strictly contained in one or more plane. In this section, we will show how our attack can be applied in a general setting as long as the inner state contains at least two planes. We apply it to Keccak$[72, 128]$ and Keccak$[144, 256]$. Those two versions are proposed in [KY10].

8.1 Getting collisions in a general setting

Let r be any rate and c a capacity such that $\frac{c}{5\omega} \geq 2$. Then finding a collision is equivalent to solving the system of equations $\mathcal{S}$ defined in Section 5. Define r' as the smallest multiple of 5ω such that $r \leq r'$. Then, our system $\mathcal{S}$ can be rewritten as the concatenation of two systems: one with $r' - r$ equations, and one with $c - (r' - r)$ equations.

$$\begin{cases} f_0(m_0, \dots, m_{r-1}, s_0, \dots, s_{c-1}) = f_0(m'_0, \dots, m'_{r-1}, s'_0, \dots, s'_{c-1}) \\ f_1(m_0, \dots, m_{r-1}, s_0, \dots, s_{c-1}) = f_1(m'_0, \dots, m'_{r-1}, s'_0, \dots, s'_{c-1}) \\ \dots \\ f_{r'-r-1}(m_0, \dots, m_{r-1}, s_0, \dots, s_{c-1}) = f_{r'-r-1}(m'_0, \dots, m'_{r-1}, s'_0, \dots, s'_{c-1}) \end{cases} \quad (\mathcal{S}_1)$$

$$\begin{cases} f_{r'-r}(m_0, \dots, m_{r-1}, s_0, \dots, s_{c-1}) = f_{r'-r}(m'_0, \dots, m'_{r-1}, s'_0, \dots, s'_{c-1}) \\ f_{r'-r+1}(m_0, \dots, m_{r-1}, s_0, \dots, s_{c-1}) = f_{r'-r+1}(m'_0, \dots, m'_{r-1}, s'_0, \dots, s'_{c-1}) \\ \dots \\ f_{c-1}(m_0, \dots, m_{r-1}, s_0, \dots, s_{c-1}) = f_{c-1}(m'_0, \dots, m'_{r-1}, s'_0, \dots, s'_{c-1}) \end{cases} \quad (\mathcal{S}_2)$$

where the f_i for i from 0 to $c-1$ are the component functions of the Keccak permutation for the bits located in the inner state.

Since r' is a multiple of 5ω, we also have that $c - (r' - r)$ is also a multiple of 5ω. Applying Proposition 2 and Theorem 3 to the system $\mathcal{S}_2$, we know that the system $\mathcal{S}_2$ is equivalent to a system $\mathcal{S}'_2$, where satisfying equations of this system can be done by satisfying constancy on some bits located in the same column.

The attack then works the same as if the outer part was made exactly of full planes. By building a linear system $\mathcal{L}$, we try to satisfy as many equations as possible in $\mathcal{S}'_2$ and we consider that $\mathcal{S}_1$ is satisfied with probability $2^{r-r'}$.

8.2 Attack on concrete instances

We apply this to Keccak$[72, 128]$ and Keccak$[144, 160]$. Both correspond to 16 lanes of capacity, and therefore 9 lanes of rate. We attack these instances as if we sought a collision on three planes. It is exactly the same idea as Section 6.3. The results of this analysis are summed up in Table 3. To help the reader understanding these results, one can rely on Figures 9 and 10.

Table 3: Different choices for the number of bits per slice to fix. (outer state made of two planes)

Number of allocated bits	2	3	4	5
Number of equations added to $\mathcal{L}$	3	5	7	9
Number of equations satisfied in $\mathcal{S}'$	1	2	3	4
Ratio ν_{slice}	0.33	0.40	0.43	0.44

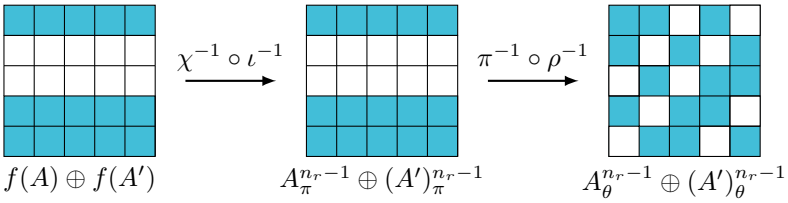

Figure 9: Illustration of the effect of the Keccak mappings when the outer state is two planes.

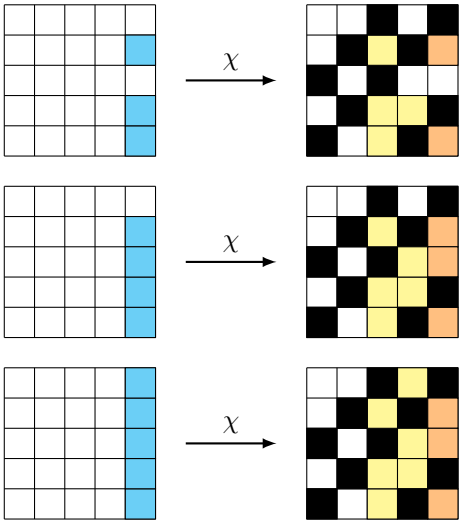

Figure 10: Linear equations derived when the outer state is two planes. The blue bits in blue are fixed. We satisfy constancy on the yellow bits by finding solutions to $\mathcal{L}$. The orange bits are equal to the blue bits with high probability.

Example 11 (KECCAK[72, 128])**.** We study an allocation strategy on KECCAK[72,128] reduced to two rounds. Since the greatest ν_{slice} is associated to fixing 5 bits per slice, it seems that the best strategy is to fix 5 bits on as many slices as possible. On each slice, fixing 5 bits means adding a total of 9 equations to our system. We set $n_5 = 8$. Note that $\omega = 8$ so this is possible. We thereby add $8 \times 9 = 72$ linear equations to our system $\mathcal{L}$. Further, for each 5 bits of a column we fix, we allocate the same value to the 3 of them that are located on the alternative inner state when the rate is $r' = 80$ as defined above, as we know from Section 7.3 that this is the best choice.

In total, we satisfy 32 equations of $\mathcal{S}'$. The subset X of $\mathbb{F}_2^c$ where we send every state is thus of size $2^{c-32} = 2^{96}$. Our memory complexity is thus slightly smaller than $2^{\frac{96}{2}} = 2^{48}$, as some extra equations are satisfied with higher probability (see Section 7.3). We will give the exact time complexity in Section 9.2, but we already know that it is smaller than $2^{48} \times g$.

Example 12 (KECCAK[144, 256])**.** We study an allocation strategy on KECCAK[144, 256]. Since the greatest ν_{slice} is associated to fixing 5 bits per slice, it seems that the best strategy is to fix 5 bits on as many slices as possible. On each slice, fixing 5 bits means adding a total of 9 equations to our system. We set $n_5 = 16$. Note that $\omega = 16$ so this is possible. We thereby add $16 \times 9 = 144$ linear equations to our system $\mathcal{L}$. Further, for each 5 bits of a column we fix, we allocate the same value to the 3 of them that are located on the alternative inner state when the rate is $r' = 160$ as defined above.

In total, we satisfy 64 equations of $\mathcal{S}'$. The subset of $\mathbb{F}_2^c$ where we send every state is thus of size $2^{c-64} = 2^{192}$. Our memory complexity is thus slightly smaller than $2^{\frac{192}{2}} = 2^{96}$, while our time complexity is smaller than $2^{96} \times g$ (see the exact time complexity in Section 9.2).

9 Complexity and implementation

In this section, we provide the exact complexity of our attack and a brief description of the implementation of our proof of concept.

9.1 Pre-computation of Gaussian elimination

In general, the cost of solving a linear system of r equations is r^3 bit operations. If the size of the system is small enough to fit in the processor register, it can also be reduced to r^2 64-bit operations for instance if the processor works on 64 bit words. However, one can notice that the Gaussian elimination operation is the same for all produced states. This allows us to put this computation in a pre-computation phase, replacing the multiplication by g from all previous complexities with one multiplication matrix-vector, where the matrix is of size $(b - r) \times r$ and the vector is of size $b - r$, to provide the right message(s) of length r. Ahead of this multiplication matrix-vector we sequentially determine if the state is a right candidate. This is done by looking if the system is solvable and requires $e - \mathsf{rank}(\mathcal{L})$ scalar products on elements of length $b - r$ which can be considered as negligible in front of the KECCAK round function.

Let's now compare the cost of the KECCAK round function with the multiplication matrix-vector. θ requires 25 XORs to compute the parities of the 5ω columns, 5 rotations, and 50 XORs to add for every bit. ρ requires 24 rotations, π requires 25 change of coordinates and χ requires 25×3 gates. Finally, taking into account the round constant, we count one round function of KECCAK as 205 logic operations. We can then say that two rounds of KECCAK require 410 operations, when our matrix-vector multiplication can require up to $r \times (b - r)$ operations.

For KECCAK[40, 160], the ratio between the cost of system solving and the cost of two rounds of KECCAK is then $\frac{40\times160}{410}$. Then we replace g by $g' = \frac{640}{41}$. For KECCAK[72, 128], g has to be replaced by $g' = \frac{4608}{205}$. As for KECCAK[144, 256], we replace g by $g' = \frac{18432}{205}$.

9.2 Complexity of our attack

For the birthday argument, if the size of the subset we're looking at is of size 2^s, then the Birthday paradox says that a collision is found with probability 2^{-1} when we collected $2^{s/2}$ states. Taking this into account, as well as the the fact that we can pre-compute the Gaussian elimination (see Section 9.1) together with the improvements that are due to the χ mapping properties (see Section 7.3), we can obtain the exact complexities of the best attacks on KECCAK[40, 160], KECCAK[72, 128] and KECCAK[144, 256].

For KECCAK[40, 160], the Crunchy Contest version, our attack is described by Example 5. The best time-memory trade-off is described in Example 10. For the attack with the best time complexity, $n_3 = 5$ and $n_2 = 1$, where we allocate bits in blue to the same value, the number of equations satisfied with probability 1 is $n = 22$. Each of the remaining 6 equations are satisfied with probability $\frac{17}{32}$ (only 2 bits in blue per slice are relevant) between any pair of states produced. Hence, the probability that we get an inner collision between a pair of states S and S' is exactly

$$\Pr[\widehat{S} = \widehat{S}'] = 2^{n+6-c}\left(\frac{17}{32}\right)^6 .$$

The number of state to collect is the square root of the inverse of this probability. The time complexity T of the attack in number of KECCAK calls reduced to two rounds is thus

$$2 \times \frac{640}{41} \times 2^{-1} \times 2^{\frac{c-n-6}{2}} \times \left(\frac{32}{17}\right)^3 = 2^{66} \times \frac{640}{41} \times \left(\frac{32}{17}\right)^3 \approx 2^{73} .$$

The factor 2^{-1} comes from the fact that in this example, the number of equations is 39 and not 40, meaning that each time we compute a state, we can derive an other one faster than applying KECCAK.

For KECCAK[72, 128], we have $n = 32$, we fix 5 bits per slice. We allocate the same value to the 3 of them that are located on the alternative inner state. Therefore 8×2 equations are satisfied with a probability higher than 2^{-1}. The probability of getting an inner collision between two states S and S' is thus

$$\Pr[\widehat{S} = \widehat{S}'] = 2^{n+16-c}\left(\frac{19}{64}\right)^8 .$$

The time complexity T of the attack in number of KECCAK calls reduced to two rounds is thus

$$2 \times \frac{4608}{205} \times 2^{\frac{c-n-16}{2}} \times \left(\frac{64}{19}\right)^4 = 2^{41} \times \frac{4608}{205} \times \left(\frac{64}{19}\right)^4 \approx 2^{52.5} .$$

Finally, for KECCAK[144, 256], we have $n = 64$, we also fix 5 bits per slice. We allocate the same value to the 3 of them that are located on the alternative inner state. The probability of getting an inner collision between two states S and S' is thus

$$\Pr[\widehat{S} = \widehat{S}'] = 2^{n+32-c}\left(\frac{19}{64}\right)^{16} .$$

The time complexity T of the attack in number of KECCAK calls reduced to two rounds is thus

$$2 \times \frac{18432}{205} \times 2^{\frac{c-n-32}{2}} \times \left(\frac{64}{19}\right)^8 = 2^{81} \times \frac{18432}{205} \times \left(\frac{64}{19}\right)^8 \approx 2^{101.5} .$$

Our attack then beats the generic one with a factor of $2^{27.5}$.

Memory-less attack Our attack consists mainly in producing states that belong to a given subset. We can define a deterministic function that takes the inner part of a KECCAK state, and from this value derives deterministically a string that we can consider as random. This string can be used as a message which we can process with KECCAK. We can then choose a message m in order to send this message into the desired subset X.

Let then g be the function that takes a message and returns an inner part as defined by our attack. g produces an inner part that lies in the desired subset with a certain probability, which we computed exactly. Let h be the function that takes an inner part and extends it to a certain message. Finally, let $f = g \circ h$. f is a function from $\mathbb{F}_2^c$ into $\mathbb{F}_2^c$. Thus, we can chain the inner part values, and assuming that f is a random function over $\mathbb{F}_2^c$ which produces states in the desired subset with a good probability (defined by Section 7.2), we can apply a cycle finding algorithm such as Floyd's algorithm [Flo67] or Brent's algorithm [Bre80] on the functional graph of f. Such an algorithm finds a collision with a negligible amount of memory and with the same time memory as computed above. For advanced techniques, one can see the work of Van Oorschot and Wiener [vOW94].

9.3 Implementation

As a proof of concept, we implemented our attack in C on an even smaller version of KECCAK, KECCAK[30, 70], reduced to two rounds. This instance is interesting because it has allowed us to verify the theory on many points. It has a rate that is not a multiple of $5\omega = 20$. Its inner state is made of one slice and a half. We used our attack as if the outer state was made of two planes and thereby required constancy only on bits we could control. We used Xoshiro-128 as a non-cryptographic but fast PRNG, that guarantees a long period in the output sequence to produce the input messages.

We constructed $\mathcal{L}$ by allocating 5 bits on three slices, and 2 bits on a last one. In total we thus added $5n_5 + 2n_2 + 4n_5 + 1n_2 = 30$ equations to $\mathcal{L}$, and satisfied $4n_5 + 1n_2 = 13$ equations of $\mathcal{S}'$. We precomputed the Gaussian elimination of $\mathcal{L}$ and found out that our system had a rank of 27 rather than 30. As explained in Section 6.4 this did not cause any increase in our time complexity. In order to provide a fast proof of concept that can be run in a practical time, we looked for semi-collision, that is a collision on the last 60 bits of the state. We implemented the memory-less version of the attack to show how it can be done.

Furthermore, this proof of concept allowed us to verify probabilities involved in our attack; the number of times the system has a solution (one out of eight times) together with the number of times the χ mapping follows identity on well chosen bits, that is $(\frac{7}{16})^3 \approx 0,0837$. The practical statistics thus match our theoretical values. The code is freely available at https://github.com/YannRotella/AlgebraicCollisionKeccakSmall100/.

Conclusion

In this paper, we presented a collision attack on round-reduced versions of KECCAK. Our attack only beats the best attacks on the smallest versions of KECCAK. Indeed, it is only relevant when the capacity is proportionally large compared to b (or equivalently r) and d, making attacks that would be based on differential characteristics impractical. We tackled the challenge of the KECCAK team: 'surprisingly, the smallest versions are the hardest to break'. Our cryptanalysis shows that their statement is true, as even two rounds required a strong effort. Most importantly, we showed that small KECCAK instances require dedicated cryptanalysis, since the techniques used to attack the bigger versions are very different from the ones that worked for the smaller ones.

References

[BCDM20] Tim Beyne, Yu Long Chen, Christoph Dobraunig, and Bart Mennink. Dumbo, Jumbo, and Delirium: Parallel authenticated encryption for the lightweight circus. *IACR Transactions on Symmetric Cryptology*, 2020(S1):5–30, 2020.

[BDP+16] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. Ketje v2. *Submission to Caesar competition*, 2016.

[BDPA07] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Sponge Functions. Ecrypt Hash Workshop 2007, May 2007. Available at https://keccak.team/files/SpongeFunctions.pdf.

[BDPA08a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008, 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, 2008.

[BDPA08b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak crunchy crypto collision and pre-image contest. https://keccak.team/.html, 2008.

[BDPA11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011.

[BDPA11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, 2011.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 313–314. Springer, 2013.

[Bre80] Richard P. Brent. An improved monte carlo factorization algorithm. *BIT Numerical Mathematics*, 20:176–184, 1980.

[Dae95] Joan Daemen. *Cipher and hash function design strategies based on linear and differential cryptanalysis, PhD Thesis.* K.U.Leuven, 1995.

[DDS12] Itai Dinur, Orr Dunkelman, and Adi Shamir. New attacks on Keccak-224 and Keccak-256. In Anne Canteaut, editor, *Fast Software Encryption - 19th International Workshop, FSE 2012, Washington, DC, USA, March 19-21, 2012. Revised Selected Papers*, volume 7549 of *Lecture Notes in Computer Science*, pages 442–461. Springer, 2012.

[DDS13] Itai Dinur, Orr Dunkelman, and Adi Shamir. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In Shiho Moriai, editor, *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, volume 8424 of *Lecture Notes in Computer Science*, pages 219–240. Springer, 2013.

[DEM+20] Christoph Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, Bart Mennink, Robert Primas, and Thomas Unterluggauer. Isap v2.0. *IACR Transactions on Symmetric Cryptology*, 2020(S1):390–416, 2020.

[DLWQ17] Xiaoyang Dong, Zheng Li, Xiaoyun Wang, and Ling Qin. Cube-like attack on round-reduced initialization of Ketje Sr. *IACR Transactions on Symmetric Cryptology*, 2017(1):259–280, 2017.

[EW17] Maria Eichlseder and Roman Walch. *One round, width 200, collision*, 2017. https://keccak.team/crunchy_mails/coll-r1-w200-20170922.txt.

[Flo67] Robert W. Floyd. Nondeterministic algorithms. *J. ACM*, 14:636–644, 1967.

[FNR18] Thomas Fuhr, María Naya-Plasencia, and Yann Rotella. State-recovery attacks on modified Ketje Jr. *IACR Transactions on Symmetric Cryptology*, 2018(1):29–56, 2018.

[GLL+20] Jian Guo, Guohong Liao, Guozhen Liu, Meicheng Liu, Kexin Qiao, and Ling Song. Practical collision attacks against round-reduced SHA-3. *J. Cryptology*, 33(1):228–270, 2020.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I*, volume 10031 of *Lecture Notes in Computer Science*, pages 249–274, 2016.

[KMS18] Rajendra Kumar, Nikhil Mittal, and Shashank Singh. Cryptanalysis of 2 round Keccak-384. In Debrup Chakraborty and Tetsu Iwata, editors, *Progress in Cryptology - INDOCRYPT 2018 - 19th International Conference on Cryptology in India, New Delhi, India, December 9-12, 2018, Proceedings*, volume 11356 of *Lecture Notes in Computer Science*, pages 120–133. Springer, 2018.

[KRA18] Rajendra Kumar, Mahesh Sreekumar Rajasree, and Hoda AlKhzaimi. Cryptanalysis of 1-round Keccak. In Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*, volume 10831 of *Lecture Notes in Computer Science*, pages 124–137. Springer, 2018.

[KY10] Elif Bilge Kavun and Tolga Yalçin. A lightweight implementation of Keccak hash function for radio-frequency identification applications. In Siddika Berna Ors Yalcin, editor, *Radio Frequency Identification: Security and Privacy Issues - 6th International Workshop, RFIDSec 2010, Istanbul, Turkey, June 8-9, 2010, Revised Selected Papers*, volume 6370 of *Lecture Notes in Computer Science*, pages 258–269. Springer, 2010.

[LS19] Ting Li and Yao Sun. Preimage attacks on round-reduced Keccak-224/256 via an allocating approach. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 556–584. Springer, 2019.

[LSLW17] Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage attacks on the round-reduced Keccak with cross-linear structures. *IACR Transactions on Symmetric Cryptology*, 2017(4):39–57, 2017.

[MS13] Pawel Morawiecki and Marian Srebrny. A sat-based preimage analysis of reduced Keccak hash functions. *Inf. Process. Lett.*, 113(10-11):392–397, 2013.

[NRM11] María Naya-Plasencia, Andrea Röck, and Willi Meier. Practical analysis of reduced-round Keccak. In Daniel J. Bernstein and Sanjit Chatterjee, editors, *Progress in Cryptology - INDOCRYPT 2011 - 12th International Conference on Cryptology in India, Chennai, India, December 11-14, 2011. Proceedings*, volume 7107 of *Lecture Notes in Computer Science*, pages 236–254. Springer, 2011.

[oST15] The U.S. National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Technical report, 5th August 2015.

[QSLG17] Kexin Qiao, Ling Song, Meicheng Liu, and Jian Guo. New collision attacks on round-reduced Keccak. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 216–243, 2017.

[Raj19] Mahesh Sreekumar Rajasree. Cryptanalysis of round-reduced Keccak using non-linear structures. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *Progress in Cryptology - INDOCRYPT 2019 - 20th International Conference on Cryptology in India, Hyderabad, India, December 15-18, 2019, Proceedings*, volume 11898 of *Lecture Notes in Computer Science*, pages 175–192. Springer, 2019.

[SD18] Ko Stoffelen and Joan Daemen. Column parity mixers. *IACR Transactions on Symmetric Cryptology*, 2018(1):126–159, 2018.

[SGSL18] Ling Song, Jian Guo, Danping Shi, and San Ling. New MILP modeling: Improved conditional cube attacks on Keccak-based constructions. In Thomas Peyrin and Steven D. Galbraith, editors, *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 65–95. Springer, 2018.

[SLG17] Ling Song, Guohong Liao, and Jian Guo. Non-full sbox linearization: Applications to collision attacks on round-reduced Keccak. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 428–451. Springer, 2017.

[vOW94] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, and Ravi S. Sandhu, editors, *CCS '94, Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 2-4, 1994*, pages 210–218. ACM, 1994.