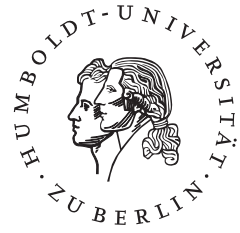


HUMBOLDT-UNIVERSITÄT ZU BERLIN



MATHEMATISCH NATURWISSENSCHAFTLICHE FAKULTÄT II  
INSTITUT FÜR INFORMATIK

# Keccak: A Sponge Construction

Seminar Aktuelle Entwicklungen in der Kryptographie

Dozent: PROF. DR. JOHANNES KÖBLER

Maxim Rohmoser

13. Oktober 2010

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>2</b>
<b>2</b>	<b>Sponge Construction</b>	<b>2</b>
2.1	Definition . . . . .	2
2.2	Auswertung . . . . .	3
<b>3</b>	<b>Sponges als Referenz-Modell</b>	<b>3</b>
3.1	Kollisionen . . . . .	3
3.2	Zufalls-Sponge . . . . .	4
3.3	Zufalls-Sponge und Zufalls-Orakel unterscheiden . . . . .	4
3.4	innere Stärke von Sponges als Hashfunktion . . . . .	5
<b>4</b>	<b>Keccak</b>	<b>6</b>
4.1	Aufbau . . . . .	6
4.2	SHA-3 Kandidaten . . . . .	7
4.3	Keccak-f . . . . .	7
4.3.1	$\theta$ . . . . .	8
4.3.2	$\rho$ ( <a href="http://keccak.noekeon.org/">http://keccak.noekeon.org/</a> ) . . . . .	8
4.3.3	$\pi$ . . . . .	9
4.3.4	$\chi$ . . . . .	9
4.3.5	$\iota$ . . . . .	10
4.4	Sicherheitsbetrachtungen . . . . .	10
<b>5</b>	<b>Schlussbemerkung</b>	<b>11</b>
<b>6</b>	<b>Literaturverzeichnis</b>	<b>11</b>

# 1 Einleitung

Diese Ausarbeitung beschäftigt sich im Allgemeinen mit Sponge-Funktionen als Referenz für iterierte Hashfunktionen und im Detail mit dem SHA-3 Kandidaten Keccak als Beispiel einer Sponge-Funktion. Sie basiert auf Veröffentlichungen von Guido Bertoni, Joan Daemen, Gilles Van Assche (STMicroelectronics) und Michaël Peeters (2NXP Semiconductors), welche alle unter <http://sponge.noekeon.org/> oder <http://keccak.noekeon.org/> zu finden sind. Als Motivation ist folgendes Szenario einer inneren Kollision gegeben.

Sei  $h$  eine iterierte Hashfunktion und seien  $M_1$  und  $M_2$  zwei Nachrichten, dann folgt aus  $h(M_1) = h(M_2)$  für jedes Suffix  $N : h(M_1|N) = h(M_2|N)$ .

Im Zufalls-Orakel sind die Hashwerte von  $M_1|N$  und  $M_2|N$  jedoch zufällig und unabhängig von den Hashwerten der Nachrichten  $M_1$  und  $M_2$ . Daher ist das Zufalls-Orakel Modell (ZOM) nicht die geeignetste Referenz für iterierte Hashfunktionen, welche aufgrund der Existenz von innerer Kollision Schwächen aufweisen, die für das ZOM nicht gelten.

Um dieses Problem zu lösen bieten sich sofort 2 Möglichkeiten an. Man könnte einerseits auf iterierte Hashfunktionen verzichten, müsste dann aber die gesamte Nachricht im Speicher halten. Andererseits kann man auch ein neues Referenz-Modell hernehmen, welches uns zum Beispiel Sponge-Funktionen liefern können.

Die Bezeichnung Sponge leitet sich daraus ab, dass die Funktion die Eingabe einem Schwamm ähnlich aufsaugt und die Ausgabe durch drücken dieses Schwammes entsteht.

## 2 Sponge Construction

### 2.1 Definition

Schauen wir uns nun an wie eine Sponge-Funktion aufgebaut ist. Dazu definieren wir zunächst Bezeichner, die wir später verwenden werden.

- Sei  $\mathcal{A}$  eine Gruppe mit Gruppenoperation  $+$  und neutralem Element  $0$ , die sowohl Eingabe als auch Ausgabe Zeichen repräsentiert
- Sei  $\mathcal{C}$  eine endliche Menge, deren Elemente die inneren Zustände eines Sponges repräsentieren.
- Sei  $0 \in \mathcal{C}$  Teil vom Anfangswert des Spongezustandes.
- Sei  $\mathcal{M}$  eine Menge von Nachrichten.
- Sei  $pa : \mathcal{M} \rightarrow \mathcal{A}$  injektiv und gelte  $\forall m \in \mathcal{M} : 0 \neq pa(m)$  sowie  $|pa(m)| \geq 1$ .

Eine Sponge-Funktion wird durch eine Transformation  $f : \mathcal{A} \times \mathcal{C} \rightarrow \mathcal{A} \times \mathcal{C}$  bestimmt. Sie nimmt einen Eingabestring variabler Länge  $p \in \mathcal{A}^*$ , für den gilt  $|p| \geq 1$  und  $p$  endet nicht auf  $0$ , und produziert einen unendlichen Ausgabestring  $z \in \mathcal{A}^\infty$ . Die einzelnen Zeichen von  $p$  werden mit  $p_i \in \mathcal{A}$  bezeichnet, wobei  $0 \leq i < |p|$ . Der Zustand eines Sponges wird mit  $S = (S_{\mathcal{A}}, S_{\mathcal{C}}) \in \mathcal{A} \times \mathcal{C}$  bezeichnet, wobei  $S_{\mathcal{C}}$  den inneren Zustand bezeichnet, und hat den Anfangswert  $(0, 0)$ . Zwei wichtige Eigenschaften einer Sponge-Funktion sind die Rate eines Sponges:  $r = \log_2 A$ , wobei  $A = |\mathcal{A}|$  und die Kapazität eines Sponges:  $c = \log_2 C$ , wobei  $C = |\mathcal{C}|$ .

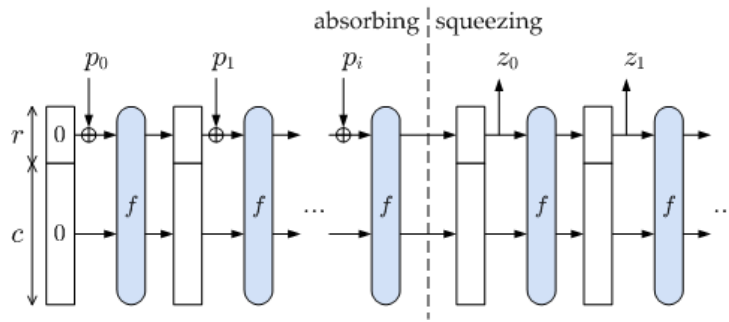


Abbildung 1: SpongeConstruction (<http://sponge.noekeon.org/>)

## 2.2 Auswertung

Das Auswerten einer Sponge-Funktion, wie in Abbildung 1 zu sehen, erfolgt in 2 Phasen. In der Absorbing-Phase wird für jedes Eingabezeichen  $p_i$  der Zustand des Sponges durch  $S \leftarrow f(S_A + p_i, S_C)$  aktualisiert. In der Squeezing-Phase wird  $z$  zeichenweise mit  $z_j \in \mathcal{A}$  und  $z_j = S_A$  und  $S \leftarrow f(S)$  ausgegeben. Für einen Eingabestring  $p$  bezeichnet  $S_f[p]$  den Wert, den der Zustand annimmt nach dem  $p$  absorbiert wurde. Wenn  $S = S_f[p]$ , dann heißt  $p$  ein Pfad (unter  $f$ ) zu  $S$ .  $S_f = (S_{A,f}, S_{C,f})$  ist durch folgende Rekursion definiert:  $S_f[\epsilon] = (0, 0)$ ,  $S_f[x|a] = f(S_f[x] + a)$  für jeden String  $x$  und jedes Zeichen  $a$ ,  $S + a = (S_A + a, S_C)$ .

Das  $j$ -te Zeichen der Ausgabe ist  $z_j = S_{A,f}[p|0^j]$ ,  $j \geq 0$ . Durch  $S_f$  wird der Zustand beschrieben, den der Sponge annimmt. Die angenommenen Zustände sind  $S_f[p']$  für jedes Präfix  $p'$  von  $p|0^\infty$ .

## 3 Sponges als Referenz-Modell

### 3.1 Kollisionen

Um Sponge-Funktionen als Referenz für iterierte Hashfunktionen zu nutzen, führen wir erst noch 2 Arten von Kollisionen ein, die in einem Sponge auftreten können.

Als erstes wäre die State-Kollision zu betrachten. Hierbei handelt es sich um ein Paar von 2 Pfaden  $p \neq q$  zum selben Zustand  $S_f[p] = S_f[q]$ . Bei Betrachtung der Auswertung eines Sponges ist klar, dass eine State-Kollision in der Absorbing-Phase zu identischen Hashwerten  $S_f[p] = S_f[q]$  führt und, dass wenn  $S_f[p] = S_f[p|0^d]$  gilt, der Ausgabestring periodisch ist.

Eine innere Kollision ist gegeben durch ein Paar von 2 Pfaden  $p \neq q$  zum selben inneren Zustand  $S_{C,f}[p] = S_{C,f}[q]$ . Man sieht sofort, dass eine State-Kollision eine innere Kollision erzwingt. Außerdem gilt, wenn  $p \neq q$  und  $S_{C,f}[p] = S_{C,f}[q]$ , dass  $p|a \neq q|b$  für alle  $a, b \in \mathcal{A}$  mit  $S_{A,f}[p] + a = S_{A,f}[q] + b$  eine State-Kollision erzeugt.

## 3.2 Zufalls-Sponge

Nun führen wir Zufalls-Sponges ein, welche unser neues Referenz-Modell bilden werden. Für eine gegebene Auswahl von  $\mathcal{A}$ ,  $\mathcal{C}$  und dem Startwert  $(0, 0)$  bestimmt die Abbildung  $f$  die Sponge-Funktion. Es existieren demnach  $(AC)^{AC}$  mögliche Sponge-Funktionen.

Als Zufalls-T(ransformierenden)-Sponge bezeichnen wir eine zufällig gleichverteilt gewählte Sponge-Funktion aus der Menge der  $(AC)^{AC}$  möglichen Sponge-Funktion.

Eine zufällig gleichverteilt gewählte Sponge-Funktion aus der Menge der  $(AC)!$  möglichen Sponge-Funktion, für die  $f$  eine Permutation ist, wird hingegen als Zufalls-P(ermutierender)-Sponge bezeichnet.

## 3.3 Zufalls-Sponge und Zufalls-Orakel unterscheiden

Benutzen wir nun Zufalls-Sponges, wie oben definiert, als Referenz, muss man sich damit auseinandersetzen, wie sie sich im Vergleich zum Zufalls-Orakel verhalten. Dazu wollen wir zeigen:

Wenn ein Zufalls-Sponge(ZS) in einer Black Box benutzt wird, so ist er nur durch das Auftreten oder die Abwesenheit von innerer Kollision von einem Zufalls-Orakel(ZO) zu unterscheiden.

Wir stellen uns also folgendes Szenario vor.:

Der Gegner hat Zugang zu einer Black Box, die entweder ein ZS oder ein ZO ist. Er stellt nun Anfragen  $m^{(1)} \dots m^{(q)}$  an die Black Box und erhält Antworten  $z^{(1)} \dots z^{(q)}$  einer festen Länge. Darauf aufbauend muss er sagen ob es sich bei der Black Box um einen ZS oder ein ZO handelt.

Wenn wir davon ausgehen, dass die Ausgabezeichen eines Zufalls-Sponges, angewendet auf eine Serie von Anfragen, gleich und unabhängig verteilt sind, falls keine innere Kollision stattgefunden hat(1), so ist das einzige, was der Gegner tun kann, seine Entscheidung darauf zu stützen eine innere Kollision zu entdecken.  $P(IC|ZS)$  bezeichnet die Wahrscheinlichkeit, dass eine innere Kollision stattgefunden hat, unter der Bedingung, dass ein Zufalls-Sponge vorliegt.

Es bleibt also nur zu zeigen, dass (1) gilt.

Die Bedingung, dass keine innere Kollision stattgefunden hat, bedeutet, dass die inneren Zustände eines Sponges  $S_{\mathcal{C},f}[x]$  alle unterschiedlich sind. Daraus folgt, dass der Wert von  $S_f[x]$  in  $\mathcal{A} \times (\mathcal{C} \setminus S_{\mathcal{C}}^x)$ , wobei  $S_{\mathcal{C}}^x$  all diejenigen inneren Zustände bezeichnet, die der Sponge beim absorbieren von  $x$  annimmt, liegen muss. Aufgrund der Konstruktion der Zufalls-Sponges sind diese Werte alle gleichwahrscheinlich. Damit ist (1) gezeigt.

Sei  $R_{ZS}$  die Menge von Beobachtungen, für die der Angreifer annimmt, dass es sich bei der Black Box um einen ZS handelt.

Dann ist

$$P(erfolg) = \frac{1}{2} + \frac{1}{2} \sum_{r \in R_{ZS}} P(r|ZS) - P(r|ZO)$$

Da  $P(IC|ZO) = 0$  per Definition gilt, kann man folgern, dass die Erfolgswahrscheinlichkeit  $P(erfolg)$  für korrektes Unterscheiden von Zufalls-Sponge und Zufalls-Orakel durch  $P(erfolg) \leq \frac{1}{2} + \frac{1}{2}P(IC|ZS)$  nach oben begrenzt ist.

### 3.4 innere Stärke von Sponges als Hashfunktion

Um eine Sponge-Funktion außerhalb einer Black Box einsetzen zu können, muss sie den üblichen Angriffen auf Hashfunktionen, wie Kollision, Urbild und 2. Urbild, standhalten. Es werden nun zuerst einige Operationen vorgestellt, die für Angriffe benutzt werden können.

- innere Kollision:  
Finden von 2 Pfaden  $p \neq q$  zum selben inneren Zustand  $S_{c,f}[p] = S_{c,f}[q]$ .
- Pfad zu einem inneren Zustand:  
Zu gegebenem inneren Zustand  $S_c$  einen Pfad  $p$  finden, so dass  $S_{c,f}[p] = S_c$ .
- Ausgabe-Zyklus:  
Finden eines Eingabestrings  $p$  und einer Zahl  $d \in \mathbb{Z}$ , so dass  $S_f[p] = S_f[p|0^d]$ .
- Binden eines Ausgabestrings an einen Zustand:  
Für einen gegebenen String  $t = t_0, t_1, \dots, t_m$  einen Zustandswert  $S$  finden, so dass der Sponge  $t$  generiert, also  $S_{\mathcal{A}} = t_0, f_{\mathcal{A}}(S) = t_1, f_{\mathcal{A}}(f(S)) = t_2, \dots, f_{\mathcal{A}}(f^{m-1}(S)) = t_m$ .

Mit diesen Operationen im Hinterkopf können wir uns anschauen, wie stark Zufalls-Sponges im Vergleich zum Zufalls-Orakel bei den genannten Angriffen sind. Im folgenden bezeichne  $c$  die Eingangs erwähnte Kapazität eines Sponges und  $n$  die Länge des Ausgabestrings.

Kollisionsangriff:

Wie bereits gesehen lässt sich aus einer inneren Kollision  $p, q$  eine State Kollision  $p|a, q|b$  für alle  $a, b \in \mathcal{A}$  mit  $S_{\mathcal{A},f}[p] + a = S_{\mathcal{A},f}[q] + b$  erzeugen. Daraus ergibt sich eine Kollision für Nachrichtenpaare  $p|a|m, q|b|m$ . Der zu erwartende Aufwand eine innere Kollision zu finden liegt für einen Zufalls-Sponge bei einer Größenordnung von  $2^{(c+3/2)}$ . Für ein Zufalls-Orakel liegt der Aufwand, eine Kollision zu finden, bei  $2^{(n+3/2)}$ . Für  $c \geq n$  bringt es demnach keinen Vorteil nach einer inneren Kollision zu suchen. Um dieses Verhalten bei Multikollisionen beizubehalten, sollte  $c \geq 2n$  gewählt werden.

2.Urbild:

Da das Finden eines 2. Urbildes eine innere Kollision erfordert, ergibt sich für  $c \geq 2n$  ein ähnlicher Aufwand wie bei einem Zufalls-Orakel.

Urbildangriff:

Für einen Urbildangriff kann ein Ausgabestring an einen Zustand gebunden werden. Je nachdem ob ein Zufalls- T- oder P-Sponge benutzt wird, sollte  $c \geq n$  bzw.  $c \geq 2n$  sein, um einen ähnlichen Aufwand wie bei einem Zufalls-Orakel zu erzeugen.

## 4 Keccak

Nachdem wir uns nun mit Sponge-Funktionen im allgemeinen beschäftigt haben, wollen wir uns am Beispiel von Keccak eine ausgewählte Sponge-Funktion näher anschauen. Es handelt sich bei Keccak um einen der SHA-3 Kandidaten, die es bis in die 2. Runde des Wettbewerbs geschafft haben. Der Name wurde wahrscheinlich von dem Balinesischem Tanz Kecak abgeleitet.

### 4.1 Aufbau

Fangen wir damit an, den Algorithmus kurz vorzustellen. Die Parameter  $r$  und  $c$  entsprechen der Eingangs erwähnten Eigenschaften, Rate und Kapazität eines Sponges. Auf den Parameter  $d$  kommen wir später noch zurück.

---

**Algorithm 1** Keccak[ $r, c, d$ ]

---

```
1: Eingabe  $M \in \mathbb{Z}_2^*$ 
2: Ausgabe  $Z \in \mathbb{Z}_2^*$ 
3:  $P = pad(M, 8) || enc(d, 8) || enc(r/8, 8)$ 
4:  $P = pad(P, r)$ 
5: Sei  $P = P_0 || P_1 || \dots || P_{|P|-1}$  mit  $P_i \in \mathbb{Z}_2^r$ 
6:  $s = 0^{r+c}$ 
7: for  $i = 0$  to  $|P| - 1$  do
8:    $s = s \oplus (P_i || 0^c)$ 
9:    $s = Keccak-f[r + c](s)$ 
10: end for
11:  $Z =$ 
12: while output is requested do
13:    $Z = Z || [s]_r$ 
14:    $s = Keccak - f[r + c](s)$ 
15: end while
```

---

In den Zeilen 3 – 5 finden wir die Preprocessing Phase. Die Funktion  $pad(M, n) = M || 10^k$  mit  $|M || 10^k| \equiv_n 0$  und  $\exists k' < k$  mit  $|M || 10^{k'}| \equiv_n 0$

gibt die Nachricht  $M$  erweitert um eine 1 und die kleinste Anzahl an 0, sodass das Ergebnis durch  $n$  teilbar ist. Während  $enc(x, n) = x_0, x_1, \dots, x_{n-1}$

mit  $x_i \in \mathbb{Z}_2$  falls  $x = \sum_{i=0}^{n-1} x_i 2^i$

die Binärdarstellung von  $x$  ausgibt. In dieser Phase erkennt man auch, dass für die Parameter  $r$  und  $d$  folgende Einschränkungen gelten:

$r > 0$  und  $r \equiv_8 0$

$d \in \{0, 1, 2, \dots, 2^8 - 1\}$

In Zeile 6 wird der Initialwert des Zustandes auf  $(0, 0)$  gesetzt, wobei dies hier  $(0^r, 0^c)$  bedeutet.

Die for-Schleife in den Zeilen 7 – 10 ist die Absorbing Phase, wie oben beschrieben. In den Zeilen 12 – 14 wird die Squeezing Phase ausgeführt. In diesen beiden Phasen erkennt man auch eine weitere Bedingung an die Parameter. So muss  $r + c$  von Keccak-f unterstützt werden. Die Standardwerte für  $r, c, d$  sind 1024, 576, 0.

## 4.2 SHA-3 Kandidaten

Als SHA-3 Kandidaten wurden die folgenden Keccak Varianten eingereicht.

- SHA3-224:  $\lfloor \text{Keccak}[r = 1152, c = 448, d = 28] \rfloor_{224}$
- SHA3-256:  $\lfloor \text{Keccak}[r = 1088, c = 512, d = 32] \rfloor_{256}$
- SHA3-384:  $\lfloor \text{Keccak}[r = 832, c = 768, d = 48] \rfloor_{384}$
- SHA3-512:  $\lfloor \text{Keccak}[r = 576, c = 1024, d = 64] \rfloor_{512}$

Der Parameter  $d$  (istinguisher) sorgt dafür, dass SHA3- $x$  kein Präfix von SHA3- $y$  für  $y > x$  ist. Man kann bei diesen Kandidaten sehr gut sehen, dass die Kapazität auf  $2^*$  Ausgabelänge gesetzt wurde, um die erforderliche Stärke gegen Angriffe zu erfüllen.

## 4.3 Keccak-f

Es existieren 7 Keccak-f Permutationen. Diese werden mit Keccak-f[b] bezeichnet, wobei  $b = 25 * 2^\ell$  und  $\ell \in \{0, 1, \dots, 6\}$ .  $b$  heißt die Weite der Permutation über  $s \in \mathbb{Z}_2^b$ . Keccak-f[b] besteht aus einer Abfolge von  $n_r = 12 + 2\ell$  Runden  $R$  ( $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ ), deren einzelne Phasen nun erläutert werden.

Im folgenden wird Keccak-f[b] über einen Zustand  $a[5][5][w]$  mit  $w = 2^\ell$  beschrieben, wobei  $s[w(5y + x) + z] = a[x][y][z]$ . In Abbildung 2 sind die einzelnen Teile eines Zustan-

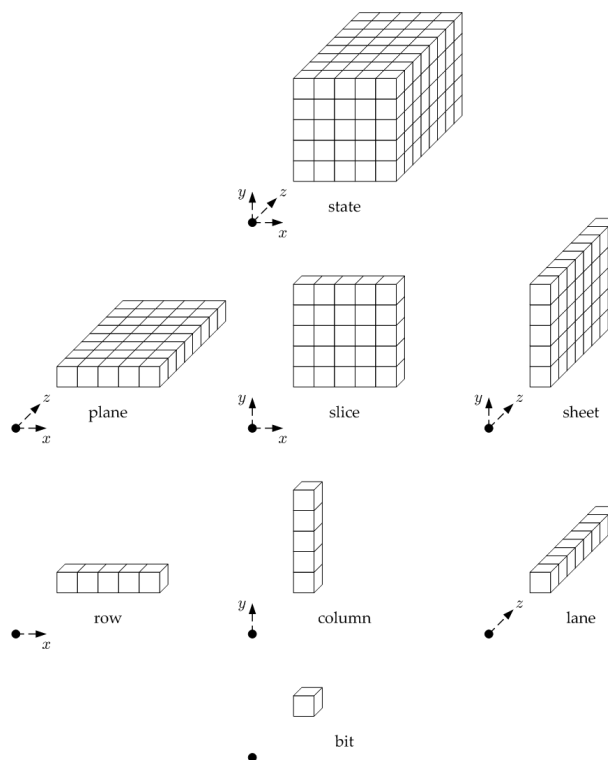


Abbildung 2: Teile des Zustandes (<http://keccak.noekeon.org/>)

des zu sehen. Zu beachten ist, dass  $a[0][0][w]$  die lane in der Mitte ist.

### 4.3.1 $\theta$

$$a[x][y][z] \leftarrow a[x][y][z] \oplus \sum_{y'=0}^4 a[x-1][y'][z] \oplus \sum_{y'=0}^4 a[x+1][y'][z-1]$$

In Abbildung 3 ist eine schematische Darstellung von  $\theta$  zu sehen. Das dunkelgraue Bit ist

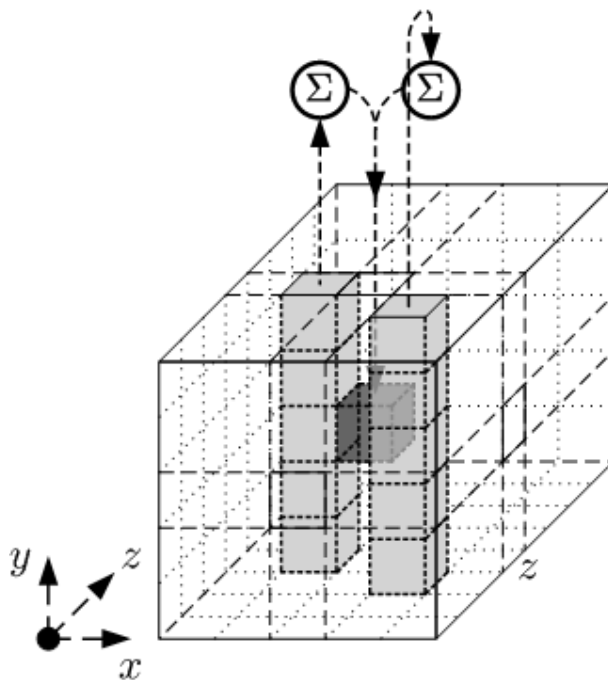


Abbildung 3:  $\theta$  (<http://keccak.noekeon.org/>)

dasjenige, welches manipuliert werden soll. Dazu werden die hellgrauen Spalten mittels xor auf dieses aufsummiert. Diese Phase sorgt für Diffusion im Zustand.

### 4.3.2 $\rho$ (<http://keccak.noekeon.org/>)

$$a[x][y][z] \leftarrow a[x][y][z - (t+1)(t+2)/2]$$

mit  $0 \leq t < 24$  und  $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$   
 oder  $t = -1$  wenn  $x = y = 0$

Hierbei handelt es sich, wie in Abbildung 4 zu sehen, um einen Shift bestimmter Länge in den einzelnen Lanes.

$\rho$  beschleunigt die Diffusion zwischen den einzelnen Scheiben des Zustandes.

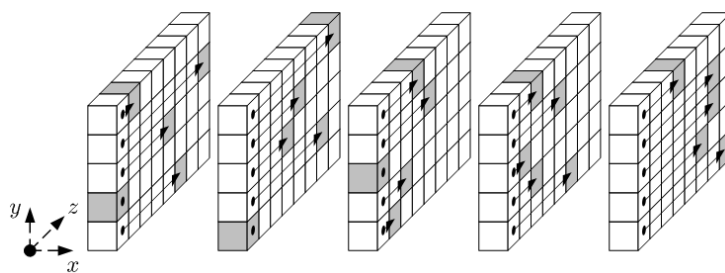


Abbildung 4:  $\rho$  (<http://keccak.noekeon.org/>)

### 4.3.3 $\pi$

$$a[x][y] \leftarrow a[x'][y'], \text{ mit } \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

Wie in Abbildung 5 zu sehen ist, werden in  $\pi$  die Bits innerhalb einer Scheibe auf

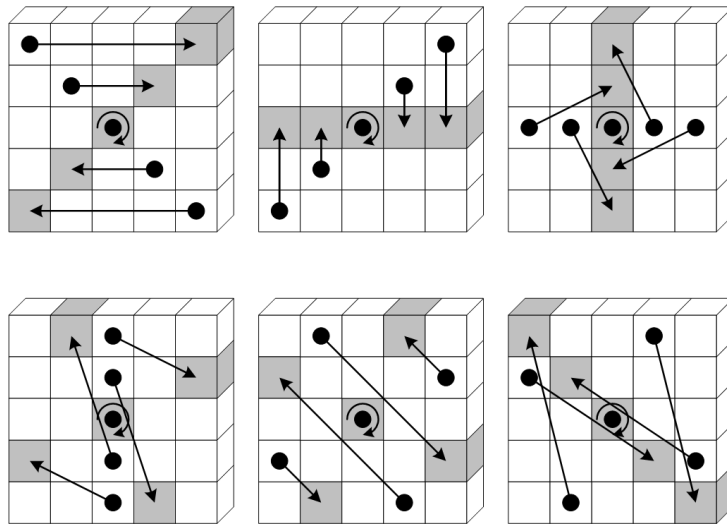


Abbildung 5:  $\pi$  (<http://keccak.noekeon.org/>)

bestimmte Weise vertauscht. Dies verstärkt die Streuung der Diffusion.

### 4.3.4 $\chi$

$$a[x] \leftarrow a[x] \oplus (a[x+1] \oplus 1)a[x+2]$$

$\chi$  bildet den einzigen nicht-linearen Teil von Keccak-f.

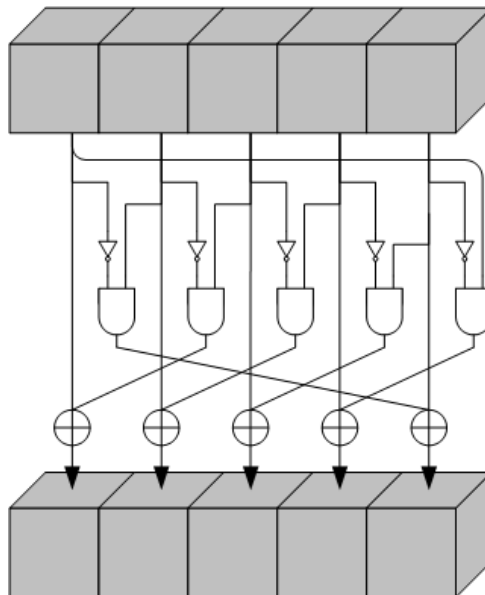


Abbildung 6:  $\chi$  (<http://keccak.noekeon.org/>)

### 4.3.5 $\iota$

$a \leftarrow a \oplus RC[i_r]$  wobei  $RC[i_r][0][0][2^j - 1] = rc[j + 7i_r]$  für alle  $0 \leq j \leq \ell$  und  $rc[t] = (x^t \bmod x^8 + x^6 + x^5 + x^4 + 1) \bmod x$ . Für alle anderen Werte  $RC[i_r][x][y][z] = 0$ . Diese Phase stört durch das Addieren der Rundenkonstanten, welche nach obigem Schema berechnet werden und in der nachfolgenden Tabelle dargestellt sind, die Symmetrie.

Tabelle 1: RC[i]

RC[0]	0x0000000000000001	RC[12]	0x000000008000808B
RC[1]	0x00000000000008082	RC[13]	0x800000000000008B
RC[2]	0x8000000000000808A	RC[14]	0x80000000000008089
RC[3]	0x80000000000008000	RC[15]	0x80000000000008003
RC[4]	0x0000000000000808B	RC[16]	0x80000000000008002
RC[5]	0x00000000800000001	RC[17]	0x8000000000000080
RC[6]	0x8000000080008081	RC[18]	0x000000000000800A
RC[7]	0x80000000000008009	RC[19]	0x800000008000000A
RC[8]	0x0000000000000080A	RC[20]	0x8000000080008081
RC[9]	0x0000000000000088	RC[21]	0x8000000000008080
RC[10]	0x0000000080008009	RC[22]	0x0000000080000001
RC[11]	0x000000008000000A	RC[23]	0x8000000080008008

## 4.4 Sicherheitsbetrachtungen

Abschließend möchte ich noch kurz auf die Sicherheit von Keccak eingehen. Die Erfinder von Keccak haben dazu folgende Definition einer Shortcut Attack geliefert.

- Ein Angriff auf eine bestimmte Sponge-Funktion mit einer erwarteten Erfolgswahrscheinlichkeit, die höher ist als für denselben Angriff auf ein Zufalls-Orakel.

Darauf aufbauend stellten Sie folgende Behauptung auf.

- Die erwartete Erfolgswahrscheinlichkeit für jede Shortcut Attack auf Keccak[r,c,d] mit einem Aufwand von  $N$  Keccak-f Aufrufen ist nach oben beschränkt durch:  
 $1 - \exp(-N(N+1)2^{-(c+1)})$

Bis jetzt wurden nur Angriffe gegen reduzierte Versionen von Keccak vorgestellt (siehe Tabelle). Zu bemerken sei hierbei noch, dass die erste Version von Keccak nur aus 18

Angriffsart:	Angriffsziel:	Hashgröße:	Parameter:	Aufrufe:
Urbild	Hash	1024	3 Runden, $r = 40$	1852 Sekunden
Distinguisher	Permutation	alle	18 Runden	$2^{1370}$
Schlüssel	geheimes Präfix(MAC)	224	4 Runden	$2^{19}$

Runden bestand und man aufgrund des Distinguishers die Rundenzahl bei der zweiten Version auf 24 erhöhte.

## 5 Schlussbemerkung

Wir haben nun also Sponge-Funktionen kennengelernt, gesehen wie man diese als Referenz-Modell für iterierte Hashfunktionen verwenden kann und worauf ihre Stärke basiert. Anschließend befassten wir uns mit dem SHA-3 Kandidaten Keccak als Beispiel für eine Sponge-Funktion.

Auf der Webseite <http://celan.informatik.uni-oldenburg.de/kryptos/info/keccak/overview/> existiert eine schöne animierte Illustration von Keccak.

## 6 Literaturverzeichnis

### Literatur

- [1] <http://ehash.iaik.tugraz.at/wiki/Keccak>
- [2] <http://keccak.noekeon.org/>
- [3] <http://sponge.noekeon.org/>
- [4] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche: Keccak-specifications  
Keccak sponge function family main document  
Sponge Functions