Algebraic Cryptanalysis of Round-Reduced Keccak with Linear Structures

Meicheng Liu
joint work with Jian Guo and Ling Song

ASK 2016, September 2016
Outline

Introduction

SHA-3 hash function
Specifications of Keccak
Main Results

Algebraic Properties of the Sbox $\chi$

Setting up linear equations from the output of $\chi$
Linearizing the inverse of $\chi$

Linear Structures

Linear structures of Keccak-f permutation
Techniques for keeping 2 rounds being linear
Techniques for keeping 3 rounds being linear

Distinguishers

Zero-sum distinguishers on Keccak-f

Preimage Attacks

Preimage attacks on Keccak
Keccak Crunchy Crypto Preimage Contest
Outline

Introduction

SHA-3 hash function
Specifications of Keccak
Main Results

Algebraic Properties of the Sbox $\chi$
Setting up linear equations from the output of $\chi$
Linearizing the inverse of $\chi$

Linear Structures
Linear structures of Keccak-f permutation
Techniques for keeping 2 rounds being linear
Techniques for keeping 3 rounds being linear

Distinguishers
Zero-sum distinguishers on Keccak-f

Preimage Attacks
Preimage attacks on Keccak
Keccak Crunchy Crypto Preimage Contest
Cryptographic hash function

- A cryptographic hash function is a mathematical algorithm that maps data of arbitrary size to a bit string of a fixed size, which is designed to also be one-way function.

- Properties
  - **Collision resistance**
    - It should be difficult to find a pair of different messages $m_1$ and $m_2$ such that $H(m_1) = H(m_2)$.
  - **Preimage resistance**
    - Given an arbitrary $n$-bit value $x$, it should be difficult to find any message $m$ such that $H(m) = x$.
  - **Second preimage resistance**
    - Given message $m_1$, it should be difficult to find any different message $m_2$ such that $H(m_1) = H(m_2)$. 
SHA-3 hash function

- NIST SHA-3 hash function competition (2007–2012)
- Winner: Keccak
  - The winner was announced to be Keccak in October 2012.
  - Designers: Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche
  - Official versions: Keccak-224/256/384/512
    The Keccak web site: [http://keccak.noekeon.org/](http://keccak.noekeon.org/)
- In August 2015 NIST announced that SHA-3 had become a hashing standard.
  - SHA3-224/256/384/512
  - SHAKE128/256 (eXtendable Output Functions, XOFs)
## SHA-3 hash function

Federal Information Processing Standards (FIPS) 202 instances

<table>
<thead>
<tr>
<th>Instances</th>
<th>r</th>
<th>c</th>
<th>Output Length</th>
<th>Collision Resistance</th>
<th>Preimage Resistance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA3-224</td>
<td>1152</td>
<td>448</td>
<td>224</td>
<td>112</td>
<td>224</td>
</tr>
<tr>
<td>SHA3-256</td>
<td>1088</td>
<td>512</td>
<td>256</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>SHA3-384</td>
<td>832</td>
<td>768</td>
<td>384</td>
<td>192</td>
<td>384</td>
</tr>
<tr>
<td>SHA3-512</td>
<td>576</td>
<td>1024</td>
<td>512</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>SHAKE128</td>
<td>1344</td>
<td>256</td>
<td>( \ell )</td>
<td>( \min(\ell/2, 128) )</td>
<td>( \min(\ell, 128) )</td>
</tr>
<tr>
<td>SHAKE256</td>
<td>1088</td>
<td>512</td>
<td>( \ell )</td>
<td>( \min(\ell/2, 256) )</td>
<td>( \min(\ell, 256) )</td>
</tr>
</tbody>
</table>

**Table:** The standard FIPS 202 instances
History of Keccak
The Road from PANAMA to Keccak via RadioGatún

PANAMA \rightarrow \text{RadioGatún} \rightarrow \text{Keccak}

1998 \rightarrow 2008

- The design was made public in 2008.
  - Sponge construction
  - 24 rounds
- It is based on earlier hash function designs PANAMA and RadioGatún.
  - PANAMA was designed by Daemen and Craig Clapp in 1998.
  - RadioGatún, a successor of PANAMA, was designed by Daemen, Peeters, and Van Assche, and was presented at the NIST Hash Workshop in 2006.

Outline

Introduction

SHA-3 hash function

Specifications of Keccak

Main Results

Algebraic Properties of the Sbox $\chi$

Setting up linear equations from the output of $\chi$

Linearizing the inverse of $\chi$

Linear Structures

Linear structures of Keccak-f permutation

Techniques for keeping 2 rounds being linear

Techniques for keeping 3 rounds being linear

Distinguishers

Zero-sum distinguishers on Keccak-f

Preimage Attacks

Preimage attacks on Keccak

Keccak Crunchy Crypto Preimage Contest
Specifications of Keccak

- Structure of Keccak
  - Sponge construction

- Keccak-\(f\) permutation
  - 1600 bits: a \(5 \times 5\) array of 64-bit lanes
  - 24 rounds
  - each round consists of five steps:
    \[\iota \circ \chi \circ \pi \circ \rho \circ \theta\]
  - \(\chi\): the only nonlinear operation
Keccak permutation

Internal state $A$: a $5 \times 5$ array of 64-bit lanes

\[ \theta \quad C[x] = A[x, 0] \oplus A[x, 1] \oplus A[x, 2] \oplus A[x, 3] \oplus A[x, 4] \]
\[ D[x] = C[x - 1] \oplus (C[x + 1] \ll 1) \]
\[ A[x, y] = A[x, y] \oplus D[x] \]

\[ \rho \quad A[x, y] = A[x, y] \ll r[x, y] \]

\[ \pi \quad B[y, 2 \ast x + 3 \ast y] = A[x, y] \]

\[ \chi \quad A[x, y] = B[x, y] \oplus ((\sim B[x + 1, y]) \& B[x + 2, y]) \]

\[ \iota \quad A[0, 0] = A[0, 0] \oplus RC \]

- The constants $r[x, y]$ are the rotation offsets.
- $RC[i]$ are the round constants.
- The only non-linear operation is $\chi$ step - algebraic degree 2
Outline

Introduction

- SHA-3 hash function
- Specifications of Keccak

Main Results

Algebraic Properties of the Sbox \( \chi \)
- Setting up linear equations from the output of \( \chi \)
- Linearizing the inverse of \( \chi \)

Linear Structures
- Linear structures of Keccak-f permutation
- Techniques for keeping 2 rounds being linear
- Techniques for keeping 3 rounds being linear

Distinguishers
- Zero-sum distinguishers on Keccak-f

Preimage Attacks
- Preimage attacks on Keccak
- Keccak Crunchy Crypto Preimage Contest
Zero-sum distinguishers on Keccak-\(f\) permutation

Exploiting the linear structures of Keccak-\(f\)

<table>
<thead>
<tr>
<th>#R</th>
<th>inv+forw</th>
<th>Best Known</th>
<th>inv+forw</th>
<th>Improved</th>
<th>inv+forw</th>
<th>Further</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3+4</td>
<td>(2^{13}) [JN15]</td>
<td>3+4</td>
<td>(2^{10})</td>
<td>(2+5)</td>
<td>(2^9)</td>
</tr>
<tr>
<td>8</td>
<td>3+5</td>
<td>(2^{18}) [AM09, JN15]</td>
<td>3+5</td>
<td>(2^{17})</td>
<td>3+5</td>
<td>(2^{10})</td>
</tr>
<tr>
<td>9</td>
<td>4+5</td>
<td>(2^{33^*}) [AM09]</td>
<td>4+5</td>
<td>(2^{28})</td>
<td>(3+6)</td>
<td>(2^{17})</td>
</tr>
<tr>
<td>10</td>
<td>4+6</td>
<td>(2^{65^*}) [AM09]</td>
<td>4+6</td>
<td>(2^{33})</td>
<td>4+6</td>
<td>(2^{28})</td>
</tr>
<tr>
<td>11</td>
<td>5+6</td>
<td>(2^{82^*}) [AM09]</td>
<td>4+7</td>
<td>(2^{65})</td>
<td>4+7</td>
<td>(2^{33})</td>
</tr>
<tr>
<td>12</td>
<td>5+7</td>
<td>(2^{129}) [AM09]</td>
<td>5+7</td>
<td>(2^{82})</td>
<td>(4+8)</td>
<td>(2^{65})</td>
</tr>
<tr>
<td>13</td>
<td>6+7</td>
<td>(2^{244}) [AM09]</td>
<td>(5+8)</td>
<td>(2^{129})</td>
<td>(5+8)</td>
<td>(2^{82})</td>
</tr>
<tr>
<td>14</td>
<td>6+8</td>
<td>(2^{257}) [AM09]</td>
<td>6+8</td>
<td>(2^{244})</td>
<td>(5+9)</td>
<td>(2^{129})</td>
</tr>
<tr>
<td>15</td>
<td>6+9</td>
<td>(2^{513}) [AM09]</td>
<td>6+9</td>
<td>(2^{257})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>12+12</td>
<td>(2^{1575}) [BCC11, DL11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Extend the previous zero-sum distinguishers by 2 rounds without increasing the complexities
  - **11 rounds**: practical complexity
  - **12 rounds**: used in Keyak and Ketje

*Corrected.
Preimage attacks on Keccak
Exploiting the linear structures of Keccak-\( f \) and bilinear structure of \( \chi \)

<table>
<thead>
<tr>
<th>#Rounds</th>
<th>Variant</th>
<th>Time</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Keccak-224/256</td>
<td>( 2^{33} )</td>
<td>[Naya-PlasenciaRM11]</td>
</tr>
<tr>
<td>2</td>
<td>Keccak-224/256</td>
<td>1</td>
<td>Our results</td>
</tr>
<tr>
<td>2</td>
<td>Keccak-384/512</td>
<td>( 2^{129}/2^{384} )</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>SHAKE128</td>
<td>1</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>Keccak-224/256/384</td>
<td>( 2^{97}/2^{192}/2^{322} )</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>Keccak-512</td>
<td>( 2^{482} )</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>Keccak-512</td>
<td>( 2^{506} )</td>
<td>[MorawieckiPS13]</td>
</tr>
<tr>
<td>4</td>
<td>SHAKE128</td>
<td>( 2^{106} )</td>
<td>Our results</td>
</tr>
<tr>
<td>4</td>
<td>Keccak-224/256</td>
<td>( 2^{213}/2^{251} )</td>
<td>Our results</td>
</tr>
<tr>
<td>4</td>
<td>Keccak-224/256</td>
<td>( 2^{221}/2^{252} )</td>
<td>[MorawieckiPS13]</td>
</tr>
<tr>
<td>4</td>
<td>Keccak-384/512</td>
<td>( 2^{378}/2^{506} )</td>
<td>[MorawieckiPS13]</td>
</tr>
</tbody>
</table>

Keccak Crunchy Crypto Contest: we solved two 3-round preimage challenges and a 4-round preimage challenge
Outline
Introduction
SHA-3 hash function
Specifications of Keccak
Main Results

Algebraic Properties of the Sbox $\chi$
Setting up linear equations from the output of $\chi$
Linearizing the inverse of $\chi$

Linear Structures
Linear structures of Keccak-f permutation
Techniques for keeping 2 rounds being linear
Techniques for keeping 3 rounds being linear

Distinguishers
Zero-sum distinguishers on Keccak-f

Preimage Attacks
Preimage attacks on Keccak
Keccak Crunchy Crypto Preimage Contest
Setting up linear equations from the output of $\chi$

Bilinear structure of $\chi$

The algebraic normal form of $\chi$ mapping 5-bit $a$ into 5-bit $b$ can be written as $b_i = a_i \oplus (a_{i+1} \oplus 1) \cdot a_{i+2}$, and specially we have

$$b_0 = a_0 \oplus (a_1 \oplus 1) \cdot a_2 \quad (1)$$

$$b_1 = a_1 \oplus (a_2 \oplus 1) \cdot a_3 \quad (2)$$

Given two consecutive bits of the output of $\chi$, one linear equation on the input bits can be set up. By (2), we have

$$b_1 \cdot a_2 = (a_1 \oplus (a_2 \oplus 1) \cdot a_3) \cdot a_2 = a_1 \cdot a_2 \quad (3)$$

and thus according to (1) we obtain

$$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2. \quad (4)$$

Given three consecutive bits of the output of $\chi$, to say $b_0$, $b_1$, and $b_2$, an additional linear equation can be similarly set up:

$$b_1 = a_1 \oplus (b_2 \oplus 1) \cdot a_3. \quad (5)$$
Setting up linear equations from the output of $\chi$

Bilinear structure of $\chi$

The input $a$ and output $b$ of 5-bit Sbox $\chi$ satisfy $F(a, b) = 0$ with

$$F(u, v) = uSv + Tu + Qv,$$

for some $5 \times 5$ binary matrices $S, T, Q$.

**Table:** Number of Linear Equations on Input Bits Obtained from the Output of 5-bit Sbox $\chi$

<table>
<thead>
<tr>
<th>#Known consecutive output bits</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Linear equations on input bits</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Setting up more linear equations

1. The first method is to guess the value of an input bit.
   ▶ guess the value of input bit $a_1$
   ▶ obtain the linear equation $b_0 = a_0 \oplus (a_1 \oplus 1) \cdot a_2$

2. The second method is to make use of the probabilistic equation $b_i = a_i$ with probability 0.75.
Outline

Introduction
  SHA-3 hash function
  Specifications of Keccak

Main Results

Algebraic Properties of the Sbox $\chi$
  Setting up linear equations from the output of $\chi$
  Linearizing the inverse of $\chi$

Linear Structures
  Linear structures of Keccak-f permutation
  Techniques for keeping 2 rounds being linear
  Techniques for keeping 3 rounds being linear

Distinguishers
  Zero-sum distinguishers on Keccak-f

Preimage Attacks
  Preimage attacks on Keccak
  Keccak Crunchy Crypto Preimage Contest
Linearizing the inverse of $\chi$

The inverse $\chi^{-1}: b \mapsto a$ has algebraic degree 3, and its algebraic normal form can be written as

$$a_i = b_i \oplus (b_{i+1} \oplus 1) \cdot (b_{i+2} \oplus (b_{i+3} \oplus 1) \cdot b_{i+4}) \quad (6)$$

where $0 \leq i \leq 4$ and the indexes are operated on modulo 5.

If we impose $b_3 = 0$ and $b_4 = 1$, then we have

$$a_0 = b_0 \oplus (b_1 \oplus 1) \cdot (b_2 \oplus 1),$$
$$a_1 = b_1,$$
$$a_2 = 1 \oplus b_2 \oplus (b_0 \oplus 1) \cdot b_1,$$
$$a_3 = 0,$$
$$a_4 = 1 \oplus (b_0 \oplus 1) \cdot b_1,$$

and thus all $a_i$'s are linear on $b_0$ and $b_2$. That's, for $b_3 = 0$, $b_4 = 1$ and any fixed $b_1$, the algebraic degree of $\chi^{-1}$ becomes 1.
Outline
Introduction
  SHA-3 hash function
  Specifications of Keccak
Main Results
Algebraic Properties of the Sbox $\chi$
  Setting up linear equations from the output of $\chi$
  Linearizing the inverse of $\chi$
Linear Structures
  Linear structures of Keccak-f permutation
  Techniques for keeping 2 rounds being linear
  Techniques for keeping 3 rounds being linear
Distinguishers
  Zero-sum distinguishers on Keccak-f
Preimage Attacks
  Preimage attacks on Keccak
  Keccak Crunchy Crypto Preimage Contest
Linear structures of Keccak-f permutation

- Several known attacks are based on the technique of linearizing 1-round Keccak-f
  - Zero-sum distinguishers [AM09]
  - Cube-attack-like cryptanalysis on keyed variants of Keccak [DMP+15]
- We find that 2- and 3-round Keccak-f can be linearized

\[
\begin{array}{c|c|c}
1 & 1 & \\
\text{backward} & \text{forward} & \\
\hline
1 & 2 & \\
\text{backward} & \text{forward} & \\
\end{array}
\]
Outline

Introduction
- SHA-3 hash function
- Specifications of Keccak

Main Results
- Algebraic Properties of the Sbox $\chi$
  - Setting up linear equations from the output of $\chi$
  - Linearizing the inverse of $\chi$

Linear Structures
- Linear structures of Keccak-f permutation
  - Techniques for keeping 2 rounds being linear
  - Techniques for keeping 3 rounds being linear

Distinguishers
- Zero-sum distinguishers on Keccak-f

Preimage Attacks
- Preimage attacks on Keccak
- Keccak Crunchy Crypto Preimage Contest
Techniques for keeping 2 rounds being linear with the degrees of freedom up to 256

▶ Keeping one round forward being linear

![Diagram](image)

**Figure:** Keeping one round forward being linear with the degrees of freedom up to 256, with yellow bits of degree 1, orange bits of degree at most 1, and white bits being constants.

▶ Keeping one round backward being linear

▶ The only nonlinear part $\chi$ operates on each 5-bit row. Since there is at most 1 variable in each row, the inverse function $\chi^{-1}$ is linear on these variables.
Techniques for keeping 2 rounds being linear with the degrees of freedom up to 512

- Keeping one round forward being linear

Figure: Keeping one round forward being linear with the degrees of freedom up to 512, with yellow bits of degree 1, orange bits of degree at most 1, and the other bits being constants.

- Keeping one round backward being linear
  - linearizing the inverse of $\chi$ according to its property: restrict the bits of gray lanes to be all ones and the bits of lightgray lanes to be all zeros
Outline
Introduction
  SHA-3 hash function
  Specifications of Keccak
Main Results
Algebraic Properties of the Sbox $\chi$
  Setting up linear equations from the output of $\chi$
  Linearizing the inverse of $\chi$
Linear Structures
  Linear structures of Keccak-f permutation
  Techniques for keeping 2 rounds being linear
  Techniques for keeping 3 rounds being linear
Distinguishers
  Zero-sum distinguishers on Keccak-f
Preimage Attacks
  Preimage attacks on Keccak
  Keccak Crunchy Crypto Preimage Contest
Techniques for keeping 3 rounds being linear with the degrees of freedom up to 64

▶ Keeping two rounds forward being linear

<table>
<thead>
<tr>
<th></th>
<th>0,0</th>
<th>1,0</th>
<th>2,0</th>
<th>3,0</th>
<th>4,0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0,0</td>
<td>1,0</td>
<td>2,0</td>
<td>3,0</td>
<td>4,0</td>
</tr>
<tr>
<td>0,1</td>
<td>0,1</td>
<td>1,1</td>
<td>2,1</td>
<td>3,1</td>
<td>4,1</td>
</tr>
<tr>
<td>0,2</td>
<td>0,2</td>
<td>1,2</td>
<td>2,2</td>
<td>3,2</td>
<td>4,2</td>
</tr>
<tr>
<td>0,3</td>
<td>0,3</td>
<td>1,3</td>
<td>2,3</td>
<td>3,3</td>
<td>4,3</td>
</tr>
<tr>
<td>0,4</td>
<td>0,4</td>
<td>1,4</td>
<td>2,4</td>
<td>3,4</td>
<td>4,4</td>
</tr>
</tbody>
</table>

Keeping one round backward being linear

<table>
<thead>
<tr>
<th></th>
<th>0,0</th>
<th>1,1</th>
<th>2,2</th>
<th>3,3</th>
<th>4,4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0,0</td>
<td>1,1</td>
<td>2,2</td>
<td>3,3</td>
<td>4,4</td>
</tr>
<tr>
<td>3,0</td>
<td>3,0</td>
<td>4,1</td>
<td>0,2</td>
<td>1,3</td>
<td>2,4</td>
</tr>
<tr>
<td>1,0</td>
<td>1,0</td>
<td>2,1</td>
<td>3,2</td>
<td>4,3</td>
<td>0,4</td>
</tr>
<tr>
<td>4,0</td>
<td>4,0</td>
<td>0,1</td>
<td>1,2</td>
<td>2,3</td>
<td>3,4</td>
</tr>
<tr>
<td>2,0</td>
<td>2,0</td>
<td>3,1</td>
<td>4,2</td>
<td>0,3</td>
<td>1,4</td>
</tr>
</tbody>
</table>

θ

π • ρ

π • ρ

θ

ι

χ

θ

π • ρ

π • ρ

ι

χ

θ

π • ρ

π • ρ

ι

χ
Keeping two rounds forward being linear

Let $A[i, j]$ with $i = 0, 2$ and $j = 0, 1, 2$ be variables.

- To make sure that the variables do not affect the other bits after step $\theta$ of the first round, we impose $2 \times 64$ equations:

$$A[i, 0] \oplus A[i, 1] \oplus A[i, 2] = 0, \ i = 0, 2.$$

- After the steps $\chi$ and $\iota$, the lane in orange equals to $A[0, 0] \oplus A[2, 2] \ll 43$, the lanes in yellow remain unchanged up to constants, and the white lanes are all constants.

- To make sure that the variables do not affect the other bits after step $\theta$ of the second round, we impose $3 \times 64$ equations:

$$A[2, 0] \ll 62 = A[0, 0] \oplus A[2, 2] \ll 43$$
$$A[2, 1] \ll 6 = A[0, 1] \ll 36$$
$$A[2, 2] \ll 43 = A[0, 2] \ll 3$$

This linear system of $5 \times 64 = 320$ equations on $6 \times 64 = 384$ variables has 64 degrees of freedom.
Techniques for keeping 3 rounds being linear
with the degrees of freedom up to 128

- Keeping two rounds forward being linear

- Keeping one round backward being linear
Techniques for keeping 3 rounds being linear
with the degrees of freedom up to 194

- Keeping two rounds forward being linear

- Keeping one round backward being linear
Outline

Introduction
- SHA-3 hash function
- Specifications of Keccak

Main Results

Algebraic Properties of the Sbox $\chi$
- Setting up linear equations from the output of $\chi$
- Linearizing the inverse of $\chi$

Linear Structures
- Linear structures of Keccak-f permutation
- Techniques for keeping 2 rounds being linear
- Techniques for keeping 3 rounds being linear

Distinguishers
- Zero-sum distinguishers on Keccak-f

Preimage Attacks
- Preimage attacks on Keccak
- Keccak Crunchy Crypto Preimage Contest
Zero-sum distinguishers on Keccak-\(f\)

Exploiting the linear structures of Keccak-\(f\)

What’s a zero-sum distinguisher?
- Find a set \(S\) such that \(\sum_{x \in S} x = 0\) and \(\sum_{x \in S} f(x) = 0\).
- Known zero-sum distinguisher on Keccak-\(f\) permutation

\[
\begin{array}{c|c|c}
\text{backward} & m & 1+n \\
\hline
\text{forward} & m+1 & n \\
\end{array}
\]

or

\[
\begin{array}{c|c|c}
\text{backward} & m+1 & n \\
\hline
\text{forward} & m & 1+n \\
\end{array}
\]

- Our improved zero-sum distinguisher on Keccak-\(f\) permutation

\[
\begin{array}{c|c|c}
\text{backward} & m+1 & 1+n \\
\hline
\text{forward} & m+1 & 2+n \\
\end{array}
\]

\[
\text{Complexity: } 2^{1+\max(2^n, 3^m)} - \text{Since } \deg(\chi) = 2 \text{ and } \deg(\chi-1) = 3, \text{ the algebraic degree of } n \text{ forward Keccak-}\(f\) \text{ rounds is bounded by } 2^n, \text{ and } m \text{ backward rounds by } 3^m.
\]
Zero-sum distinguishers on Keccak-\(f\)

Exploiting the linear structures of Keccak-\(f\)

What's a zero-sum distinguisher?

- Find a set \(S\) such that \(\sum_{x \in S} x = 0\) and \(\sum_{x \in S} f(x) = 0\).

Known zero-sum distinguisher on Keccak-\(f\) permutation

\[
\begin{array}{c|c}
\text{backward} & \text{forward} \\
m & 1+n \\
\end{array}
\]

or

\[
\begin{array}{c|c}
\text{backward} & \text{forward} \\
m+1 & n \\
\end{array}
\]

Our improved zero-sum distinguisher on Keccak-\(f\) permutation

\[
\begin{array}{c|c}
\text{backward} & \text{forward} \\
m+1 & 1+n \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{backward} & \text{forward} \\
m+1 & 2+n \\
\end{array}
\]

Complexity: \(2^{1+\max(2^n,3^m)}\)

- Since \(\deg(\chi) = 2\) and \(\deg(\chi^{-1}) = 3\), the algebraic degree of \(n\) forward Keccak-\(f\) rounds is bounded by \(2^n\), and \(m\) backward rounds by \(3^m\).
Zero-sum distinguishers on Keccak-f
Exploiting the linear structures of Keccak-f

- Extend the previous zero-sum distinguishers by 2 rounds without increasing the complexities

<table>
<thead>
<tr>
<th>#R</th>
<th>inv+forw</th>
<th>Best Known</th>
<th>inv+forw</th>
<th>Improved</th>
<th>inv+forw</th>
<th>Further</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3+4</td>
<td>2^{13} [JN15]</td>
<td>3+4</td>
<td>2^{10}</td>
<td>2+5</td>
<td>2^{9}</td>
</tr>
<tr>
<td>8</td>
<td>3+5</td>
<td>2^{18} [AM09, JN15]</td>
<td>3+5</td>
<td>2^{17}</td>
<td>3+5</td>
<td>2^{10}</td>
</tr>
<tr>
<td>9</td>
<td>4+5</td>
<td>2^{33*} [AM09]</td>
<td>4+5</td>
<td>2^{28}</td>
<td>3+6</td>
<td>2^{17}</td>
</tr>
<tr>
<td>10</td>
<td>4+6</td>
<td>2^{65*} [AM09]</td>
<td>4+6</td>
<td>2^{33}</td>
<td>4+6</td>
<td>2^{28}</td>
</tr>
<tr>
<td>11</td>
<td>5+6</td>
<td>2^{82*} [AM09]</td>
<td>4+7</td>
<td>2^{65}</td>
<td>4+7</td>
<td>2^{33}</td>
</tr>
<tr>
<td>12</td>
<td>5+7</td>
<td>2^{129} [AM09]</td>
<td>5+7</td>
<td>2^{82}</td>
<td>4+8</td>
<td>2^{65}</td>
</tr>
<tr>
<td>13</td>
<td>6+7</td>
<td>2^{244} [AM09]</td>
<td>5+8</td>
<td>2^{129}</td>
<td>5+8</td>
<td>2^{82}</td>
</tr>
<tr>
<td>14</td>
<td>6+8</td>
<td>2^{257} [AM09]</td>
<td>6+8</td>
<td>2^{244}</td>
<td>5+9</td>
<td>2^{129}</td>
</tr>
<tr>
<td>15</td>
<td>6+9</td>
<td>2^{513} [AM09]</td>
<td>6+9</td>
<td>2^{257}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>12+12</td>
<td>2^{1575} [BCC11, DL11]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Corrected.
Zero-sum distinguishers on Keccak-$f$

Exploiting the linear structures of Keccak-$f$

- Practical distinguisher for 11 rounds
  - The 12-round Keccak-$f$ permutations can be distinguished with complexity $2^{65}$ or $2^{82}$.
    - This is of special interests since the 12-round Keccak-$f$ permutation variants are used in the CAESAR candidates Keyak and Ketje.
    - Nevertheless, we stress here that this distinguisher does not affect the security of Keyak or Ketje.
Outline

Introduction
SHA-3 hash function
Specifications of Keccak
Main Results

Algebraic Properties of the Sbox $\chi$
Setting up linear equations from the output of $\chi$
Linearizing the inverse of $\chi$

Linear Structures
Linear structures of Keccak-f permutation
Techniques for keeping 2 rounds being linear
Techniques for keeping 3 rounds being linear

Distinguishers
Zero-sum distinguishers on Keccak-f

Preimage Attacks
Preimage attacks on Keccak
Keccak Crunchy Crypto Preimage Contest
## Preimage attacks on Keccak

Exploiting the linear structures of Keccak-f and bilinear structure of $\chi$

<table>
<thead>
<tr>
<th>#Rounds</th>
<th>Variant</th>
<th>Time</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>128/224/256</td>
<td>$2^{33}$</td>
<td>[Naya-PlasenciaRM11]</td>
</tr>
<tr>
<td>2</td>
<td>128/224/256</td>
<td>1</td>
<td>Our results</td>
</tr>
<tr>
<td>2</td>
<td>384/512</td>
<td>$2^{129}/2^{384}$</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>128</td>
<td>1</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>224/256/384</td>
<td>$2^{97}/2^{192}/2^{322}$</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>512</td>
<td>$2^{482}$</td>
<td>Our results</td>
</tr>
<tr>
<td>3</td>
<td>512</td>
<td>$2^{506}$</td>
<td>[MorawieckiPS13]</td>
</tr>
<tr>
<td>4</td>
<td>128</td>
<td>$2^{106}$</td>
<td>Our results</td>
</tr>
<tr>
<td>4</td>
<td>224/256</td>
<td>$2^{213}/2^{251}$</td>
<td>Our results</td>
</tr>
<tr>
<td>4</td>
<td>224/256</td>
<td>$2^{221}/2^{252}$</td>
<td>[MorawieckiPS13]</td>
</tr>
<tr>
<td>4</td>
<td>384/512</td>
<td>$2^{378}/2^{506}$</td>
<td>[MorawieckiPS13]</td>
</tr>
</tbody>
</table>
Example: Preimage Attacks on SHAKE128

1. linearize two rounds with one round forward and one round backward, and obtain 512 variables such that the first two rounds are linear

\[
\begin{array}{c|c}
1 & 1 \\
\text{backward} & \text{forward}
\end{array}
\]

37/45
Example: Preimage Attacks on SHAKE128

1. linearize two rounds with one round forward and one round backward, and obtain 512 variables such that the first two rounds are linear

\[
\begin{array}{c|c}
1 & 1 \\
\text{backward} & \text{forward}
\end{array}
\]

2. to make sure that the state input to the first round corresponds to a legal message, we set up 262 linear equations (256 bits for capacity and 6 bits for padding)
Example: Preimage Attacks on SHAKE128

1. linearize two rounds with one round forward and one round backward, and obtain 512 variables such that the first two rounds are linear

   \[
   \begin{array}{c|c}
   1 & 1 \\
   \text{backward} & \text{forward}
   \end{array}
   \]

2. to make sure that the state input to the first round corresponds to a legal message, we set up 262 linear equations (256 bits for capacity and 6 bits for padding)

After the above two steps, there remains 250 free variables such that the bits input to step $\chi$ of the third round are all linear.
Example: Preimage Attacks on SHAKE128

1. linearize two rounds with one round forward and one round backward, and obtain 512 variables such that the first two rounds are linear

\[
\begin{array}{c|c|c}
\text{1} & \text{1} \\
\hline
\text{backward} & \text{forward} \\
\end{array}
\]

2. to make sure that the state input to the first round corresponds to a legal message, we set up 262 linear equations (256 bits for capacity and 6 bits for padding)

After the above two steps, there remains 250 free variables such that the bits input to step $\chi$ of the third round are all linear.

- Preimage attacks on SHAKE128 with output length 128
  - 3 rounds: set up linear equations by exploiting bilinear structure of $\chi$ and guessing some bits input to $\chi$
Example: Preimage Attacks on SHAKE128

1. linearize two rounds with one round forward and one round backward, and obtain 512 variables such that the first two rounds are linear

\[
\begin{array}{c|c}
1 & 1 \\
\text{backward} & \text{forward}
\end{array}
\]

2. to make sure that the state input to the first round corresponds to a legal message, we set up 262 linear equations (256 bits for capacity and 6 bits for padding)

After the above two steps, there remains 250 free variables such that the bits input to step $\chi$ of the third round are all linear.

- Preimage attacks on SHAKE128 with output length 128
  - 3 rounds: set up linear equations by exploiting bilinear structure of $\chi$ and guessing some bits input to $\chi$
  - 4 rounds: partially linearize the third round, and set up linear equations by bilinear structure of $\chi$
Example: Preimage Attacks on SHAKE128

- for 3-round SHAKE128, given a 128-bit hash value $h$:
Example: Preimage Attacks on SHAKE128

- for 3-round SHAKE128, given a 128-bit hash value $h$:
  - we set up 64 linear equations on the 250 free variables (the first two output bits $b_0$ and $b_1$ of 64 Sboxes are known)

\[
  b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2
\]
Example: Preimage Attacks on SHAKE128

- for 3-round SHAKE128, given a 128-bit hash value \( h \):
  - we set up 64 linear equations on the 250 free variables (the first two output bits \( b_0 \) and \( b_1 \) of 64 Sboxes are known)
    
    \[
    b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2
    \]
  - set up extra \( 2 \times 64 \) linear equations by guessing 64 bits input to step \( \chi \) of the third round
    
    \[
    a_2 = c
    \]
    
    \[
    b_1 = a_1 \oplus (c \oplus 1) \cdot a_3
    \]
Example: Preimage Attacks on SHAKE128

- for 3-round SHAKE128, given a 128-bit hash value $h$:
  - we set up 64 linear equations on the 250 free variables (the first two output bits $b_0$ and $b_1$ of 64 Sboxes are known)
    
    $$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2$$
  
  - set up extra $2 \times 64$ linear equations by guessing 64 bits input to step $\chi$ of the third round
    
    $$a_2 = c$$
    $$b_1 = a_1 \oplus (c \oplus 1) \cdot a_3$$
  
  - obtain a linear system of 192 equations on 250 variables, and each solution corresponds to a preimage of $h$
Example: Preimage Attacks on SHAKE128

- for 3-round SHAKE128, given a 128-bit hash value $h$:
  - we set up 64 linear equations on the 250 free variables (the first two output bits $b_0$ and $b_1$ of 64 Sboxes are known)

  \[ b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 \]

  - set up extra $2 \times 64$ linear equations by guessing 64 bits input to step $\chi$ of the third round

  \[ a_2 = c \]

  \[ b_1 = a_1 \oplus (c \oplus 1) \cdot a_3 \]

  - obtain a linear system of 192 equations on 250 variables, and each solution corresponds to a preimage of $h$
  - the time complexity of this attack is 1
Example: Preimage Attacks on SHAKE128

- for 3-round SHAKE128, given a 128-bit hash value $h$:
  - we set up 64 linear equations on the 250 free variables (the first two output bits $b_0$ and $b_1$ of 64 Sboxes are known)

\[
b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2
\]

- set up extra $2 \times 64$ linear equations by guessing 64 bits input to step $\chi$ of the third round

\[
a_2 = c
\]
\[
b_1 = a_1 \oplus (c \oplus 1) \cdot a_3
\]

- obtain a linear system of 192 equations on 250 variables, and each solution corresponds to a preimage of $h$
  - the time complexity of this attack is 1

- similar techniques help us solve two 3-round preimage challenges in Keccak Crunchy Crypto Contest
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:

  \[
  b_1 = 1 \Rightarrow b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2
  \]

  by guessing $22 \times 10$ bits input to step $\chi$ of the third round,

  we linearize 22 bits of $a_0$'s input to step $\chi$ of the fourth round - each $a_0$ has 11 quadratic terms, two of which have a common factor, so by guessing 10 bits we can linearize $a_0$

  obtain a linear system of 242 equations on 250 variables, and each solution matches at least 22 bits ($b_0$'s) of $h$

  this attack gives a correct preimage in $2^{128} - 2^{22} = 2^{106}$

  similar techniques show that one 4-round preimage challenge can be solved in $2^{54}$, about $2^{20}$ CPU core hours

  we recently cut down the time to $2^{34}$!
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:
  - we expect 32 zeros and 32 ones among the last 64 bits ($b_1$’s) of $h$, and for $b_1 = 1$ we have

$$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0$$
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:
  - we expect 32 zeros and 32 ones among the last 64 bits ($b_1$'s) of $h$, and for $b_1 = 1$ we have
    \[ b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0 \]
  - by guessing $22 \times 10$ bits input to step $\chi$ of the third round, we linearize 22 bits of $a_0$'s input to step $\chi$ of the fourth round
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:
  - we expect 32 zeros and 32 ones among the last 64 bits ($b_1$’s) of $h$, and for $b_1 = 1$ we have
    $$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0$$
  - by guessing $22 \times 10$ bits input to step $\chi$ of the third round, we linearize 22 bits of $a_0$’s input to step $\chi$ of the fourth round
    - each $a_0$ has 11 quadratic terms, two of which have a common factor, so by guessing 10 bits we can linearize $a_0$
Example: Preimage Attacks on SHAKE128

* for 4-round SHAKE128, given a 128-bit hash value \( h \):
  * we expect 32 zeros and 32 ones among the last 64 bits \( (b_1’s) \) of \( h \), and for \( b_1 = 1 \) we have
    \[
    b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0
    \]
  * by guessing \( 22 \times 10 \) bits input to step \( \chi \) of the third round, we linearize 22 bits of \( a_0 \)’s input to step \( \chi \) of the fourth round
    - each \( a_0 \) has 11 quadratic terms, two of which have a common factor, so by guessing 10 bits we can linearize \( a_0 \)
  * obtain a linear system of 242 equations on 250 variables, and each solution matches at least 22 bits \( (b_0’s) \) of \( h \)
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:
  - we expect 32 zeros and 32 ones among the last 64 bits ($b_1$'s) of $h$, and for $b_1 = 1$ we have
    \[
    b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0
    \]
  - by guessing $22 \times 10$ bits input to step $\chi$ of the third round, we linearize 22 bits of $a_0$'s input to step $\chi$ of the fourth round
    - each $a_0$ has 11 quadratic terms, two of which have a common factor, so by guessing 10 bits we can linearize $a_0$
  - obtain a linear system of 242 equations on 250 variables, and each solution matches at least 22 bits ($b_0$'s) of $h$
  - this attack gives a correct preimage in $2^{128-22} = 2^{106}$
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:
  - we expect 32 zeros and 32 ones among the last 64 bits ($b_1$'s) of $h$, and for $b_1 = 1$ we have
    \[
    b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0
    \]
  - by guessing $22 \times 10$ bits input to step $\chi$ of the third round, we linearize 22 bits of $a_0$'s input to step $\chi$ of the fourth round
    - each $a_0$ has 11 quadratic terms, two of which have a common factor, so by guessing 10 bits we can linearize $a_0$
  - obtain a linear system of 242 equations on 250 variables, and each solution matches at least 22 bits ($b_0$'s) of $h$
  - this attack gives a correct preimage in $2^{128-22} = 2^{106}$

- similar techniques show that one 4-round preimage challenge can be solved in $2^{54}$, about $2^{20}$ CPU core hours
Example: Preimage Attacks on SHAKE128

- for 4-round SHAKE128, given a 128-bit hash value $h$:
  - we expect 32 zeros and 32 ones among the last 64 bits ($b_1$’s) of $h$, and for $b_1 = 1$ we have
    $$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2 = a_0$$
  - by guessing $22 \times 10$ bits input to step $\chi$ of the third round, we linearize 22 bits of $a_0$’s input to step $\chi$ of the fourth round
    - each $a_0$ has 11 quadratic terms, two of which have a common factor, so by guessing 10 bits we can linearize $a_0$
  - obtain a linear system of 242 equations on 250 variables, and each solution matches at least 22 bits ($b_0$’s) of $h$
  - this attack gives a correct preimage in $2^{128-22} = 2^{106}$

- similar techniques show that one 4-round preimage challenge can be solved in $2^{54}$, about $2^{20}$ CPU core hours

NEW we recently cut down the time to $2^{34}$!
Outline

Introduction
- SHA-3 hash function
- Specifications of Keccak

Main Results

Algebraic Properties of the Sbox $\chi$
- Setting up linear equations from the output of $\chi$
- Linearizing the inverse of $\chi$

Linear Structures
- Linear structures of Keccak-f permutation
- Techniques for keeping 2 rounds being linear
- Techniques for keeping 3 rounds being linear

Distinguishers
- Zero-sum distinguishers on Keccak-f

Preimage Attacks
- Preimage attacks on Keccak
- Keccak Crunchy Crypto Preimage Contest
Keccak Crunchy Crypto Contest

Keccak team presents challenges for reduced-round Keccak instances, namely $\text{Keccak}[c = 160, r = b - c]$ with $b \geq 200$:

- The capacity is fixed to 160 bits: this implies a security level of $2^{80}$ against generic collision search.
- The width $b$ of Keccak-$f[b]$ is in $\{200, 400, 800, 1600\}$: the width values that support the chosen capacity.
- The number of rounds $n_r$ ranges from 1 to 12.

For each of these Keccak instances there are two challenges:

- generating a collision in the output truncated to 160 bits;
- generating a preimage of an output truncated to 80 bits.
Keccak Crunchy Crypto Preimage Contest
A solution for 3-round preimage challenge of width 1600

Challenge: 06 25 a3 46 28 c0 cf e7 6c 75
Keccak Crunchy Crypto Preimage Contest

A solution for 3-round preimage challenge of width 1600

Challenge: 06 25 a3 46 28 c0 cf e7 6c 75

Preimage:

01e0bc766796d36f ffffffffffffffffff bd25fc21a299814e 0000000000000000 0000000000000000
cc85265f6f0e696a ffffffffffffffffff 3a6f339c0eb075b9 0000000000000000 0000000000000000
d22ac7903b459dc2 ffffffffffffffffff 903a19e9986a2ac7 0000000000000000 0000000000000000
539674b5f5e23187 ffffffffffffffffff 1770d654e35ec89e 0000000000000000 0000000000000000
b326d6f339c0e9bf ffffffffffffffffff d71d16ae
Keccak Crunchy Crypto Preimage Contest
A solution for 3-round preimage challenge of width 800

Challenge: 00 7b b5 c5 99 80 66 0e 02 93
Keccak Crunchy Crypto Preimage Contest
A solution for 3-round preimage challenge of width 800

Challenge: 00 7b b5 c5 99 80 66 0e 02 93
Preimage:

ffffffff1097e68a 069e5c9097c2a342 9128124400000000 3bc3a3a300000000 0000000000000000
000000056ace9cb 00000000cb56ace9 2ba3ccb200000000 990fc4d300000000 ff2c346d00000000
Keccak Crunchy Crypto Preimage Contest
A solution for 4-round preimage challenge of width 1600

Challenge: 7d aa d8 07 f8 50 6c 9c 02 76
Keccak Crunchy Crypto Preimage Contest

A solution for 4-round preimage challenge of width 1600

Challenge: 7d aa d8 07 f8 50 6c 9c 02 76
Preimage:

0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0b9eed82c23255f5 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x0000000000000000 0x1c992115b20be87e 0x9c4db251c5fad36a 0x2c9060dec9357251 0x867a8f082ede00aa 0xeaff48177a506da 0x79eefce6557a40ee 0x584677049bc52c08 0x6e3276d820c23daa 0xd2d3181a1187b0b0 0x2afff48177a506da
Summary

- Properties of the nonlinear operation $\chi$ and its inverse $\chi^{-1}$
- Linear structures of Keccak-\(f\) permutation
- Improved zero-sum distinguishers on Keccak-\(f\) permutation
  - extend the previous zero-sum distinguishers by 2 rounds without increasing the complexities
  - practical distinguisher for 11 rounds
- Preimage attacks on Keccak
  - practical preimage attacks on 3-round SHAKE128
  - solve two 3-round preimage challenges and a 4-round preimage challenge in the Keccak Crunchy Crypto Contest
- Directions of future work
  - more applications of linear structures