Algebraic Cryptanalysis of Round-Reduced Keccak with Linear Structures

Meicheng Liu
joint work with Jian Guo and Ling Song

ASK 2016, September 2016

中国科学院信息工程研究所
INSTITUTE OF INFORMATION ENGINEERING,CAS

NANYANG TECHNOLOGICAL UNIVERSITY

Outline

Outline

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

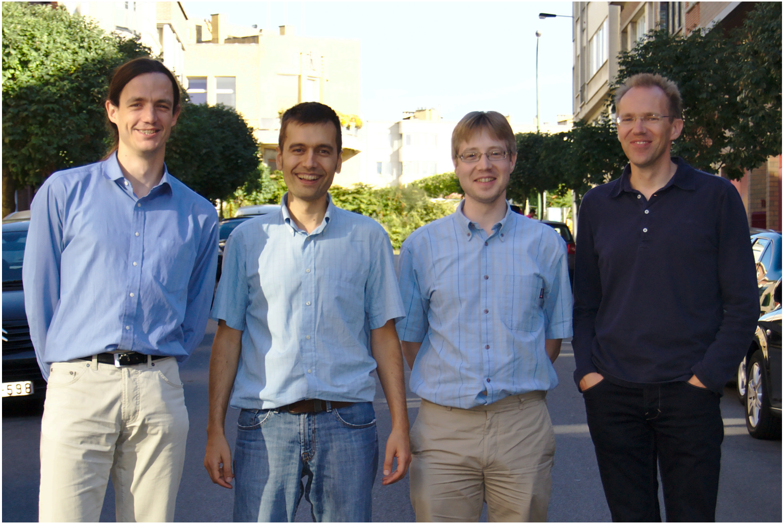
    The Keccak web site: `http://keccak.noekeon.org/`
- In August 2015 NIST announced that SHA-3 had become a hashing standard.
  - SHA3-224/256/384/512
  - SHAKE128/256 (eXtendable Output Functions, XOFs)

SHA-3 hash function

Federal Information Processing Standards (FIPS) 202 instances

Instances	r	c	Output Length	Collision Resistance	Preimage Resistance
SHA3-224	1152	448	224	112	224
SHA3-256	1088	512	256	128	256
SHA3-384	832	768	384	192	384
SHA3-512	576	1024	512	256	512
SHAKE128	1344	256	ℓ	$\min(\ell/2, 128)$	$\min(\ell, 128)$
SHAKE256	1088	512	ℓ	$\min(\ell/2, 256)$	$\min(\ell, 256)$

Table: The standard FIPS 202 instances



Michaël Peeters, Guido Bertoni, Gilles Van Assche and Joan Daemen

The Keccak Team

History of Keccak

The Road from PANAMA to Keccak via RadioGatún

$$\underset{1998}{\text{PANAMA}} \xrightarrow{\text{RadioGatún}} \underset{2008}{\text{Keccak}}$$

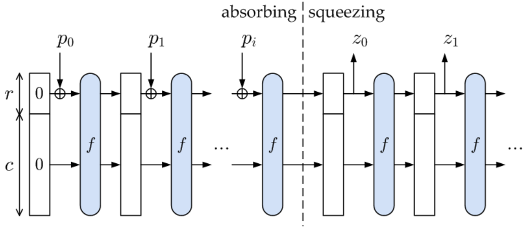
- The design was made public in 2008.
  - Sponge construction
  - 24 rounds
- It is based on earlier hash function designs PANAMA and RadioGatún.
  - PANAMA was designed by Daemen and Craig Clapp in 1998.
  - RadioGatún, a successor of PANAMA, was designed by Daemen, Peeters, and Van Assche, and was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

Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche: *The Road from PANAMA to Keccak via RadioGatún*. Symmetric Cryptography 2009.

Outline

Specifications of Keccak

- Structure of Keccak
  - Sponge construction



- Keccak-f permutation
  - 1600 bits: a 5×5 array of 64-bit lanes
  - 24 rounds
  - each round consists of five steps:

$$\iota \circ \chi \circ \pi \circ \rho \circ \theta$$

  - χ : the only nonlinear operation

Keccak permutation

Internal state A: a 5×5 array of 64-bit lanes

$$\theta \quad \begin{aligned} C[x] &= A[x,0] \oplus A[x,1] \oplus A[x,2] \oplus A[x,3] \oplus A[x,4] \\ D[x] &= C[x-1] \oplus (C[x+1] \lll 1) \\ A[x,y] &= A[x,y] \oplus D[x] \end{aligned}$$

$$\rho \quad A[x,y] = A[x,y] \lll r[x,y]$$

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

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

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

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

Outline

Zero-sum distinguishers on Keccak-f permutation

Exploiting the linear structures of Keccak-f

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

- 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 χ

#Rounds	Variant	Time	Reference
2	Keccak-224/256	2^{33}	[Naya-PlasenciaRM11]
2	Keccak-224/256	1	Our results
2	Keccak-384/512	$2^{129}/2^{384}$	Our results
3	SHAKE128	1	Our results
3	Keccak-224/256/384	$2^{97}/2^{192}/2^{322}$	Our results
3	Keccak-512	2^{482}	Our results
3	Keccak-512	2^{506}	[MorawieckiPS13]
4	SHAKE128	2^{106}	Our results
4	Keccak-224/256	$2^{213}/2^{251}$	Our results
4	Keccak-224/256	$2^{221}/2^{252}$	[MorawieckiPS13]
4	Keccak-384/512	$2^{378}/2^{506}$	[MorawieckiPS13]

- **Keccak Crunchy Crypto Contest:** we solved two 3-round preimage challenges and a 4-round preimage challenge

Outline

Setting up linear equations from the output of χ

Bilinear structure of χ

The algebraic normal form of χ 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

$$b_0 = a_0 \oplus (a_1 \oplus 1) \cdot a_2 \tag{1}$$

$$b_1 = a_1 \oplus (a_2 \oplus 1) \cdot a_3 \tag{2}$$

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

$$b_1 \cdot a_2 = (a_1 \oplus (a_2 \oplus 1) \cdot a_3) \cdot a_2 = a_1 \cdot a_2 \tag{3}$$

and thus according to (1) we obtain

$$b_0 = a_0 \oplus (b_1 \oplus 1) \cdot a_2. \tag{4}$$

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

$$b_1 = a_1 \oplus (b_2 \oplus 1) \cdot a_3. \tag{5}$$

Setting up linear equations from the output of χ

Bilinear structure of χ

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

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

for some 5×5 binary matrices S, T, Q.

Table: Number of Linear Equations on Input Bits Obtained from the Output of 5-bit Sbox χ

#Known consecutive output bits	2	3	4	5
#Linear equations on input bits	1	2	4	5

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

Linearizing the inverse of χ

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

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

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

$$\begin{aligned}
a_0 &= b_0 \oplus (b_1 \oplus 1) \cdot (b_2 \oplus 1), \\
a_1 &= b_1, \\
a_2 &= 1 \oplus b_2 \oplus (b_0 \oplus 1) \cdot b_1, \\
a_3 &= 0, \\
a_4 &= 1 \oplus (b_0 \oplus 1) \cdot b_1,
\end{aligned}$$

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 χ^{-1} becomes 1.

Outline

Introduction

- SHA-3 hash function
- Specifications of Keccak
- Main Results

Algebraic Properties of the Sbox χ

- Setting up linear equations from the output of χ
- Linearizing the inverse of χ

Linear Structures

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

Distinguishers

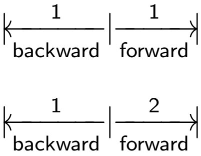
- Zero-sum distinguishers on Keccak-f

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



Outline

Techniques for keeping 2 rounds being linear

with the degrees of freedom up to 256

- Keeping one round forward being linear

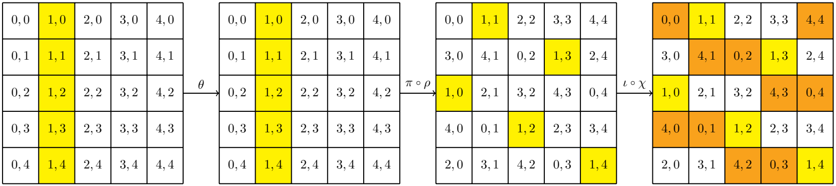


Figure: Keeping one round forward being linear with the degrees of freedom up to 256, with yellow bits of degree 1, orange bits of degree at most 1, and white bits being constants.

- Keeping one round backward being linear
  - The only nonlinear part χ operates on each 5-bit row. Since there is at most 1 variable in each row, the inverse function χ^{-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

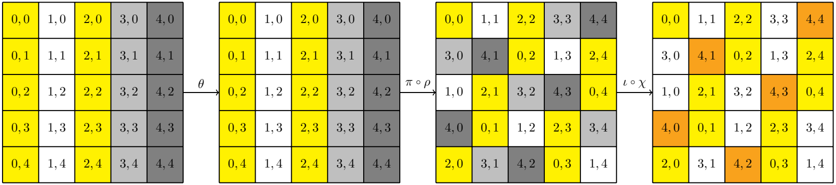


Figure: Keeping one round forward being linear with the degrees of freedom up to 512, with yellow bits of degree 1, orange bits of degree at most 1, and the other bits being constants.

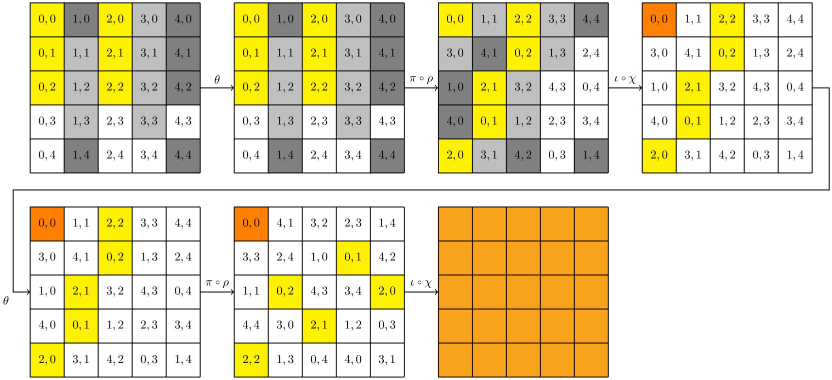
- Keeping one round backward being linear
  - linearizing the inverse of χ according to its property: restrict the bits of gray lanes to be all ones and the bits of lightgray lanes to be all zeros

Outline

Techniques for keeping 3 rounds being linear

with the degrees of freedom up to 64

- Keeping two rounds forward being linear



- Keeping one round backward being linear

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 θ of the first round, we impose 2×64 equations:

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

- After the steps χ and ι, the lane in orange equals to $A[0,0] \oplus A[2,2]_{\lll 43}$, 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 θ of the second round, we impose 3×64 equations:

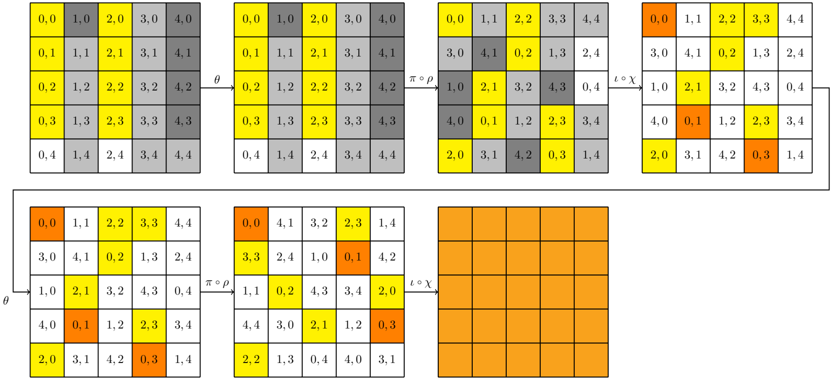
$$\begin{aligned} A[2,0]_{\lll 62} &= A[0,0] \oplus A[2,2]_{\lll 43} \\ A[2,1]_{\lll 6} &= A[0,1]_{\lll 36} \\ A[2,2]_{\lll 43} &= A[0,2]_{\lll 3} \end{aligned}$$

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

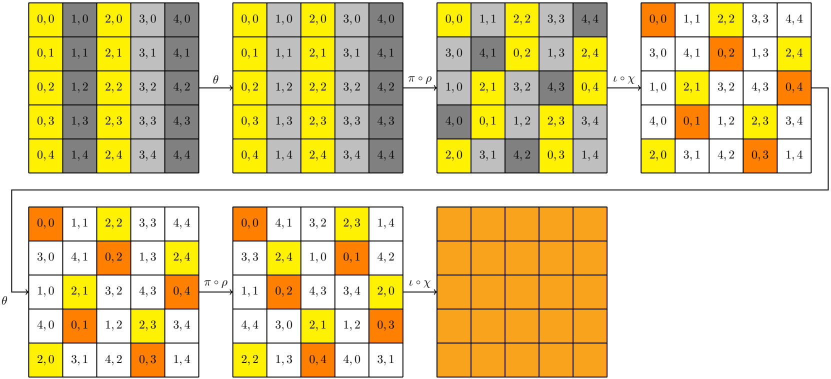


- Keeping one round backward being linear

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

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

$$\underset{\text{backward}}{\overset{m}{\longleftarrow}} \Big| \underset{\text{forward}}{\overset{1+n}{\longrightarrow}} \quad \text{or} \quad \underset{\text{backward}}{\overset{m+1}{\longleftarrow}} \Big| \underset{\text{forward}}{\overset{n}{\longrightarrow}}$$

- Our improved zero-sum distinguisher on Keccak-f permutation

$$\underset{\text{backward}}{\overset{m+\mathbf{1}}{\longleftarrow}} \Big| \underset{\text{forward}}{\overset{\mathbf{1}+n}{\longrightarrow}}$$

$$\underset{\text{backward}}{\overset{m+\mathbf{1}}{\longleftarrow}} \Big| \underset{\text{forward}}{\overset{\mathbf{2}+n}{\longrightarrow}}$$

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

$$\underbrace{\xleftarrow{\quad m \quad}}_{\text{backward}} \Big| \underbrace{\xrightarrow{\quad 1+n \quad}}_{\text{forward}} \quad \text{or} \quad \underbrace{\xleftarrow{\quad m+1 \quad}}_{\text{backward}} \Big| \underbrace{\xrightarrow{\quad n \quad}}_{\text{forward}}$$

- Our improved zero-sum distinguisher on Keccak-f permutation

$$\underbrace{\xleftarrow{\quad m+\mathbf{1} \quad}}_{\text{backward}} \Big| \underbrace{\xrightarrow{\quad \mathbf{1}+n \quad}}_{\text{forward}}$$

$$\underbrace{\xleftarrow{\quad m+\mathbf{1} \quad}}_{\text{backward}} \Big| \underbrace{\xrightarrow{\quad \mathbf{2}+n \quad}}_{\text{forward}}$$

- 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

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

*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

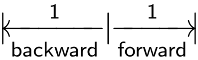
Preimage attacks on Keccak

Exploiting the linear structures of Keccak-f and bilinear structure of χ

#Rounds	Variant	Time	Reference
2	128/224/256	2^{33}	[Naya-PlasenciaRM11]
2	128/224/256	1	Our results
2	384/512	$2^{129}/2^{384}$	Our results
3	128	1	Our results
3	224/256/384	$2^{97}/2^{192}/2^{322}$	Our results
3	512	2^{482}	Our results
3	512	2^{506}	[MorawieckiPS13]
4	128	2^{106}	Our results
4	224/256	$2^{213}/2^{251}$	Our results
4	224/256	$2^{221}/2^{252}$	[MorawieckiPS13]
4	384/512	$2^{378}/2^{506}$	[MorawieckiPS13]

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



Example: Preimage Attacks on SHAKE128

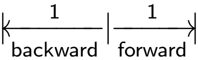
1. linearize two rounds with one round forward and one round backward, and obtain 512 variables such that the first two rounds are linear

$$\underset{\text{backward}}{\xleftarrow{\quad 1 \quad}}\Big|\underset{\text{forward}}{\xrightarrow{\quad 1 \quad}}$$

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

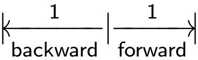


2. to make sure that the state input to the first round corresponds to a legal message, we set up 262 linear equations (256 bits for capacity and 6 bits for padding)

After the above two steps, there remains 250 free variables such that the bits input to step χ 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



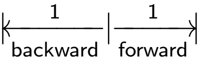
2. to make sure that the state input to the first round corresponds to a legal message, we set up 262 linear equations (256 bits for capacity and 6 bits for padding)

After the above two steps, there remains 250 free variables such that the bits input to step χ 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 χ and guessing some bits input to χ

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



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 χ 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 χ and guessing some bits input to χ
  - 4 rounds: partially linearize the third round, and set up linear equations by bilinear structure of χ

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 2×64 linear equations by guessing 64 bits input to step χ of the third round

$$\begin{aligned} a_2 &= c \\ b_1 &= a_1 \oplus (c \oplus 1) \cdot a_3 \end{aligned}$$

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 2×64 linear equations by guessing 64 bits input to step χ 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 2×64 linear equations by guessing 64 bits input to step χ of the third round

$$\begin{aligned} a_2 &= c \\ b_1 &= a_1 \oplus (c \oplus 1) \cdot a_3 \end{aligned}$$

  - 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 2×64 linear equations by guessing 64 bits input to step χ of the third round

$$\begin{aligned} a_2 &= c \\ b_1 &= a_1 \oplus (c \oplus 1) \cdot a_3 \end{aligned}$$

  - 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:

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 22×10 bits input to step χ of the third round, we linearize 22 bits of a_0's input to step χ 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 22×10 bits input to step χ of the third round, we linearize 22 bits of a_0's input to step χ 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 22×10 bits input to step χ of the third round, we linearize 22 bits of a_0's input to step χ 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 22×10 bits input to step χ of the third round, we linearize 22 bits of a_0's input to step χ 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 22×10 bits input to step χ of the third round, we linearize 22 bits of a_0's input to step χ 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 22×10 bits input to step χ of the third round, we linearize 22 bits of a_0's input to step χ 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

Keccak Crunchy Crypto Contest

Keccak team presents challenges for reduced-round Keccak instances, namely 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 ffffffffffffffff bd25fc21a299814e 0000000000000000 0000000000000000
cc85265f6f0e696a ffffffffffffffff 3a6f339c0eb075b9 0000000000000000 0000000000000000
d22ac7903b459dc2 ffffffffffffffff 903a19e9986a2ac7 0000000000000000 0000000000000000
539674b5f5e23187 ffffffffffffffff 1770d654e35ec89e 0000000000000000 0000000000000000
b326d6f339c0e9bf ffffffffffffffff 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`

`0000000056ace9cb 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:

```
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0b9eed82c23255f5 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 00000000
1c992115b20be87e 9c4db251c5fad36a 2c9060dec9357251 867a8f082ede00aa 2eaff48177a506da
79eefce6557a40ee 584677049bc52c08 6e3276d820c23daa d2d3181a1187b0b0 7ce6f00a73920b4c
e82d8f3276e85543 3cf77a79137cb68c b0d325479f4d33aa 6322817be3f75cdc 1b2d1fc33847eefa
3815737090003e07 f3ae39ce20ca35f1 fe9cf333317e463e 9cb46a02e2c495ce 4dfae61d5770ab3d
ea5218e748a57f6b 5cdac47ec1c508be c16d020b
```

Summary

- Properties of the nonlinear operation χ and its inverse χ^{-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