Cube attack and SHA-3

Liu zhang

Department of Computer Science, Shdantou University

September 21, 2020

Outline

The structure of cube attack

- Let us consider cryptosystem described by the polynomial:

  $$p\left(v_1,\dots,v_m,x_1,\dots,x_n\right)$$

  depending on m public variables $v_1,\dots,v_m$ (the initial value or plaintext) and on n secret variables $x_1,\dots,x_n$ (the key).

- The value of the polynomial represents the ciphertext bit.

- The polynomial p is not explicitly known; it can be a black box.

- Considering the known plaintext attack, where at the preprocessing state the attacker has also an access to secret variables (initial values or keys).

The state of cube attack

1. **The preprocessing stage.** The attacker can change the values of public and secret variables. The task is to obtain a system of linear equations on secret variables.
2. **The stage on-line of the attack.** The key is secret now. The attacker can change the values of public variables. He adds the output bits, where the inputs run over some multi-dimensional cubes. The task is to obtain the right hand sides of linear equations. The system of linear equation can be solved giving some bits of the key.

Mathematical background

- We shall not distinguish the secret and public variables.
- Let p be a polynomial of n variables $x_1, \dots, x_n$ over the field $GF(2)$.
- For a subset of indexes $I = \{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$, let us take a monomial
$$t_I = x_{i_1} \dots x_{i_k}$$
- Then we have a decomposition
$$p\left(x_1, \dots, x_n\right) = t_I \cdot p_{S(I)} + q\left(x_1, \dots, x_n\right)$$
where the polynomial $p_{S(I)}$ does not depend on the variables $x_{i_1}, \dots, x_{i_k}$.

Example: Polynomial decomposition

1. Consider a function p as :
$$p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) = v_0v_1x_0 + v_0v_1x_1 + v_2x_0x_2 + v_1x_2 + v_0x_0 + v_0v_1 + x_0x_2 + v_1 + x_2 + 1$$
2. Let $I = \{0, 1\}$ be a chosen subset of indexes.
3. Then the polynomial p can be decomposed as:
$$p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) = v_0v_1\left(1 + x_0 + x_1\right) + \left(v_2x_0x_2 + v_1x_2 + v_0x_0 + x_0x_2 + v_1 + x_2 + 1\right)$$
4. Using the introduced above notation:

$$t_I = v_0v_1$$
$$p_{S(I)} = 1 + x_0 + x_1$$
$$q\left(v_0, v_1, v_2, x_0, x_1, x_2\right) = v_2x_0x_2 + v_1x_2 + v_0x_0 + x_0x_2 + v_1 + x_2 + 1$$

Definitioin

The maxterm of the polynomial p we call the monomial t_I, such that

$$\deg\left(p_{S(I)}\right) = 1$$

it means that the polynomial $p_{S(I)}$ corresponding to the subset of indexes I is a linear one, which is not a constant.

Summation over cubes

- Let $I = \{i_1, \dots, i_k\} \subset \{1, \dots, n\}$ be a fixed subset of k indexes.
- The set I defines the k-dimensional boolean cube C_I, where on the place of each of the indexes we put 0 or 1.
- A given vector $v \in C_I$ defines the derived polynomial p_v depending on $n-k$ variables, where in the basic polynomial p we put the values corresponding to the vector v.
- Summing over all vectors in the cube C_I we obtain the polynomial:

$$p_I = \sum_{v \in C_I} p_v$$

Therorem 1

For an polynomial p and subset of variables I, $p_I = p_{S(I)} \mod 2$.

Proof.

1. Write $p\left(x_1, .., x_n\right) \equiv t_I \cdot p_{S(I)} + q\left(x_1, .., x_n\right)$. We first examine an arbitrary term t_J of $q\left(x_1, \ldots, x_n\right)$. Since t_J misses at least one of the variables in I, it is added an even number of times, which cancels it out modulo 2 in $\sum_{v \in C} p_{|v}$.
2. Second, we examine the polynomial $t_I \cdot p_{S(I)}$: All $\boldsymbol{v} \in C_I$ zero t_I, except when we assign the value 1 to all the variables in I. This implies that the polynomial $p_{S(I)}$ is summed only once, when t_I is set to 1.

□

The prepreossing stage

- The first task is to fix dimension of the cube and the public variables over which we will sum; they are called the *tweakable variables*, and the other public variables are put to zero. In the case we know the degree d of the basic polynomial, we put the cube dimension to $d-1$.

- We do the summation over a fixed cube for several values of secret variables and collect the obtained values.

- We do the linear tests for the obtained function of secret variables and store it when it is linear:

$$f\,(x \oplus y) = f(x) \oplus f\,(y) \oplus f(0)$$

  where $x = (x_1, \dots, x_n)$ and $y = (y_1, \dots, y_n)$ are secret variables (the key).

The preproessing stage, cont.

- The next task is to calculate the exact form (the coefficients) of the obtained linear function of secret variables.
- The free term of the linear function we obtain putting its all argument equal zero.
- The coefficient of variables x_i is equal 1 if and only if the change of this variable implies the change of values of the function.
- The coefficient of the variables x_i is equal 0 if and only if the change of this variable does not imply the change of values of function.

The preprocessing stage, cont.

- The task of this stage of attack is to collect possible many independent linear terms - they constitute the system of linear equations on secret variables.
- This system of linear equations will be used in the on-line stage of attack.
- The preprocessing stage is done only once in cryptanalysis of the algorithm.

The stage online of attack

- Now an attacker has the access only to public variables (the plaintext for block ciphers, the initial values for stream ciphers), which he can change and calculates the corresponding bits of the ciphertext under the unknown value of secret variables.

- The task of this stage of attack is to find some bits of secret key with complexity, which would be lower than the exhaustive search in the brute force attack.

- The attacker uses the derived system of linear equations for secret variables (the unknown bits of the key), where the right hand sides of these equations are the values of bits of ciphertext obtained after summation over the same cubes as in the preprocessing stage, but now the key is not known.

The stage online of attack, cont.

- The cube attack is applicable to symmetric ciphers for which the polymials describing the system have relatively low degree.
- Then one can eventually find some bits of unknown key. The remaining bits of the key may be found by brute force search.
- After successful preprocessing stage, the stage on-line of attack can be done many times for different unknown keys.
- The cube attack is applicable, in general, to cryptosystems without knowing their inner structure. The attacker must have the possibility to realize the preprocessing stage and in the on-line stage an access to the implementation of the algorithm (to perform the summation over cubes under unknown key).

Example precomputation

Let us now consider a very small example of a complete cube attack. Consider a function $p\left(v_0, v_1, v_2, x_0, x_1, x_2\right)$ where v_0, v_1, v_2 are the public inputs and x_0, x_1, x_2 are the secret inputs. In order to aid understanding, we provide the definition of p:

$$\begin{aligned} p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) = {} & v_0v_1x_0 + v_0v_1x_1 + v_2x_0x_2 + v_1x_2 + v_0x_0 + v_0v_1 \\ & + x_0x_2 + v_1 + x_2 + 1 \end{aligned}$$

Example precomputation, cont.

We will begin our search for maxterms with the subterm $t_{I_0} = v_2$ defined by the cube indices $I_0 = \{2\}$. We then have

$$p_{I_0}\left(v_0, v_1, x_0, x_1, x_2\right) = p\left(v_0, v_1, 0, x_0, x_1, x_2\right) + p\left(v_1, v_2, 1, x_0, x_1, x_2\right)$$

However, when using some p_I we will always fix the public inputs not in the set of cube indices I to some constant (e.g. zero). We mention this as a technicality, and for the rest of this example will simply presume that for some p_I the public inputs not in the set of cube indices I are set to zero, i,e.

$$p_{I_0}\left(x_0, x_1, x_2\right) = p\left(0, 0, 0, x_0, x_1, x_2\right) + p\left(0, 0, 1, x_0, x_1, x_2\right)$$

Next,we test to see if p_{I_0} is constant.

$$p_{I_0}(0,0,0) = 0$$
$$p_{I_0}(1,0,1) = 1$$

Example precomputation, cont.

Since different inputs produce different results p_{I_0} cannot be constant. So,we run a linearity test choosing some $x = (1, 0, 1)$ and $y = (0, 1, 1)$, with $x \oplus y = (1, 1, 0)$.

$$\begin{aligned} p_{I_0}(0) + p_{I_0}(x) + p_{I_0}(y) &= p_{I_0}(x \oplus y) \\ p_{I_0}(0,0,0) + p_{I_0}(1,0,1) + p_{I_0}(0,1,1) &= p_{I_0}(1,1,0) \\ 0 + 1 + 0 &= 0 \end{aligned}$$

So the linearity test fails,too. p_{I_0} must be nonlinear. If we cheat for a moment and peek at the definition of p again we see this is true:

$$\begin{aligned} p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) &= v_0v_1x_0 + v_0v_1x_1 + v_2x_0x_2 + v_1x_2 + v_0x_0 + v_0v_1 \\ &+ x_0x_2 + v_1 + x_2 + 1 \\ &= t_{I_0} \cdot p_{S(I_0)} + q_0\left(v_0, \ldots, x_2\right) \\ &= v_2\left(x_0x_2\right) + \left(v_0v_1x_0 + v_0v_1x_1 + v_1x_2 + v_0x_0 + v_0v_1\right. \\ &\left. + x_0x_2 + v_1 + x_2 + 1\right) \end{aligned}$$

Example precomputation, cont.

Next, we try the subterm $t_{I_1} = v_0 v_2$ defined by cube indices $I_1 = \{0, 2\}$. We then have

$$\begin{aligned} p_{I_1}(x_0, x_1, x_2) &= p(0,0,0,x_0,x_1,x_2) + p(0,0,1,x_0,x_1,x_2) \\ &\quad + p(1,0,0,x_0,x_1,x_2) + p(1,0,1,x_0,x_1,x_2) \end{aligned}$$

Testing p_{I_1} shows it to be constant:

$$\begin{array}{ll} p_{I_1}(0,0,0) = 0 & p_{I_1}(0,0,1) = 0 \\ p_{I_1}(0,1,0) = 0 & p_{I_1}(0,1,1) = 0 \\ p_{I_1}(1,0,0) = 0 & p_{I_1}(1,0,1) = 0 \\ p_{I_1}(1,1,0) = 0 & p_{I_1}(1,1,1) = 0 \end{array}$$

This indicates that the subterm t_{I_1} does not exist in p. If we cheat again for a moment and peek at the definition of p, we see this is true; no term in p contains $v_0 v_2$.

Example precomputation, cont.

Next we try subterm $t_{I_2} = v_0 v_1$ with $I_2 = \{0, 1\}$. It proves to be nonconstant

$$p_{I_2}(0,0,0) = 1$$
$$p_{I_2}(1,0,1) = 0$$

$\cdots$ and linear $\cdots$

$$\begin{aligned} p_{I_2}(0) + p_{I_2}(x) + p_{I_2}(y) &= p_{I_2}(x \oplus y) \\ p_{I_2}(0,0,0) + p_{I_2}(1,0,1) + p_{I_2}(0,1,1) &= p_{I_2}(1,1,0) \\ 1 + 0 + 0 &= 1 \end{aligned}$$

Normally, one would perform several linearity tests to make sure that t_{I_2} really was a maxterm, but for the sake of brevity, we will just show this one. Now, we must deduce the superpoly $p_{S(I_2)}$ that corresponds to t_{I_2}. First we compute the free term:

$$p_{I_2}(0,0,0) = 1$$

Example precomputation, cont.

Next we test for the presence of each secret variable:

$$p_{I_2}(1,0,0) = 0$$
$$p_{I_2}(0,1,0) = 0$$
$$p_{I_2}(0,0,1) = 1$$

Since the value differs from the free term for variable x_0 and x_1, we now know $p_{S(2)}$ to be :

$$p_{S(I_2)} = 1 + x_0 + x_1$$

Example precomputation, cont.

Hence, we now have the maxterm $v_0 v_1$ defined by cube indices $\{0, 1\}$ with corresponding superpoly $1 + x_0 + x_1$. If we cheat again for a moment and peek at the definition of p, we can see this is true:

$$\begin{aligned} p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) &= v_0 v_1 x_0 + v_0 v_1 x_1 + v_2 x_0 x_2 + v_1 x_2 + v_0 x_0 + v_0 v_1 \\ &+ x_0 x_2 + v_1 + x_2 + 1 \\ &= t_{I_2} \cdot p_{S(I_2)} + q_2\left(v_0, \ldots, x_2\right) \\ &= v_0 v_1 \left(1 + x_0 + x_1\right) \\ &+ \left(v_2 x_0 x_2 + v_1 x_2 + v_0 x_0 + x_0 x_2 + v_1 + x_2 + 1\right) \end{aligned}$$

Example precomputation, cont.

Next we try subterm $t_{I_3} = v_0$ with $I_3 = \{0\}$. It proves to be non-constant

$$p_{I_3}(0,0,0) = 0$$
$$p_{I_3}(0,0,1) = 1$$

$\cdots$ and linear $\cdots$

$$\begin{aligned} p_{I_3}(0) + p_{I_3}(x) + p_{I_3}(y) &= p_{I_3}(x \oplus y) \\ p_{I_3}(0,0,0) + p_{I_3}(1,0,1) + p_{I_3}(0,1,1) &= p_{I_3}(1,1,0) \\ 0 + 1 + 0 &= 1 \end{aligned}$$

We find the superpoly free term $p_{I_3}(0,0,0) = 0$ and coefficients:

$$p_{I_3}(1,0,0) = 1$$
$$p_{I_3}(0,1,0) = 0$$
$$p_{I_3}(0,0,1) = 0$$

Which gives us $p_{S(I_3)} = x_0$

Example precomputation, cont.

If we cheat again for a moment and peek at the definition of p, we can see this is true:

$$\begin{aligned} p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) &= v_0 v_1 x_0 + v_0 v_1 x_1 + v_2 x_0 x_2 + v_1 x_2 + v_0 x_0 + v_0 v_1 \\ &+ x_0 x_2 + v_1 + x_2 + 1 \\ &= t_{I_3} \cdot p_{S(I_3)} + q_3\left(v_0, \ldots, x_3\right) \\ &= v_0\left(v_1 x_0 + v_1 x_1 + x_0 + v_1\right) \\ &+ \left(v_2 x_0 x_2 + v_1 x_2 + x_0 x_2 + v_1 + x_2 + 1\right) \end{aligned}$$

At first this might seem confusing: Shouldn't $p_{S(I_3)} = v_1 x_0 + v_1 x_1 + x_0 + v_1$? Well, technically it is. However, when we sum over the cube defined by I_3, we set all the public variables not in I_3 (i.e. v_1 and v_2) to zero. Doing so results in

$$\begin{aligned} p_{S(I_3)} &= v_1 x_0 + v_1 x_1 + x_0 + v_1 \\ &\Rightarrow 0 \cdot x_0 + 0 \cdot x_1 + x_0 + 0 \\ &= x_0 \end{aligned}$$

Example precomputation, cont.

Finally we try subterm $t_{I_4} = v_1$ with $I_4 = \{1\}$. It proves to e non-constant

$$p_{I_4}(0,0,0) = 1$$
$$p_{I_4}(0,0,1) = 0$$

$\cdots$ and linear $\cdots$

$$\begin{aligned} p_{I_4}(0) + p_{I_4}(x) + p_{I_4}(y) &= p_{I_4}(x \oplus y) \\ p_{I_4}(0,0,0) + p_{I_4}(1,0,1) + p_{I_4}(0,1,1) &= p_{I_4}(1,1,0) \\ 0 + 0 + 0 &= 0 \end{aligned}$$

We find the superpoly free term $p_{I_4}(0,0,0) = 1$ and coefficients:

$$p_{I_4}(1,0,0) = 1$$
$$p_{I_4}(0,1,0) = 1$$
$$p_{I_4}(0,0,1) = 0$$

Which gives us $p_{S(I_4)} = 1 + x_2$.

Example precomputation, cont.

Peeking at the definition of p one last time shows this is correct:

$$\begin{aligned} p\left(v_0, v_1, v_2, x_0, x_1, x_2\right) &= v_0 v_1 x_0 + v_0 v_1 x_1 + v_2 x_0 x_2 + v_1 x_2 + v_0 x_0 + v_0 v_1 \\ &+ x_0 x_2 + v_1 + x_2 + 1 \\ &= t_{I_4} \cdot p_{S(I_4)} + q_4\left(v_0, \ldots, x_4\right) \\ &= v_1\left(v_0 x_0 + v_0 x_1 + x_2 + v_0 + 1\right) \\ &+ \left(v_2 x_0 x_2 + v_0 x_0 + x_0 x_2 + x_2 + 1\right) \end{aligned}$$

Once again, we must remember that when summing over the cube defined by I_4 all public inputs not in I_4 (i.e. v_0 and v_2) are set to zero. This results in

$$\begin{aligned} p_{S(I_4)} &= v_0 x_0 + v_0 x_1 + x_2 + v_0 + 1 \\ &\Rightarrow 0 \cdot x_0 + 0 \cdot x_1 + x_2 + 0 + 1 \\ &= 1 + x_2 \end{aligned}$$

Example precomputation, cont.

At this point, we have found three cubes with linearity independent superpolys, as shown in Table 1. We may now proceed with the on-line attack.

Table: Maxterms (given as cube indices) and superpolys

Superpoly polynomial $p_{S(I)}$	Cube indices I
x_0	$\{0\}$
$1 + x_0 + x_1$	$\{0, 1\}$
$1 + x_2$	$\{1\}$

Example online attack

- Let $g\left(v_0, v_1, v_2\right)$ be implemented as $p\left(v_0, v_1, v_2, x_0, x_1, x_2\right)$ with a hidden secret defined by $x_0 = 1, x_1 = 0, x_2 = 1$. We will use our precomputed cube attack on p to discover this secret by querying the oracle g.
- To begin our attack, we must find the values of each superploy. Recall from Theorem 1 that $p_{S(I)} = p_I \bmod 2$. This applies to the oracle g, so for one of our cube index sets I we can find the value of its superpoly $g_{S(I)}$ in the oracle by computing g_i.

$$\begin{aligned} x_0 = q_{\{0\}} &= g(0,0,0) + g(1,0,0) && = 1 \\ 1 + x_0 + x_1 = q_{\{0,1\}} &= g(0,0,0) + g(1,0,0) + g(0,1,0) + g(1,1,0) && = 0 \\ 1 + x_2 = q_{\{1\}} &= g(0,0,0) + g(0,1,0) && = 0 \end{aligned}$$

- This gives us a system of linear equations which we solve to find

$$x_0 = 1 \; x_1 = 0 \; x_2 = 1$$

Cube Tester

Cube tester is a distinguisher to detect some algebraic property of cryptographic primitives. The idea is to reveal non-random behaviour of a Boolean function with algebraic degree d by summing its values when cube variables of size $k(k \leq d)$ run over all of their 2^k inputs.

Theorem

Given a polynomial $f : \{0,1\}^n \rightarrow \{0,1\}$ of degree d. Suppose that $0 < k \leq d$ and t is the monomial $\prod_{i=0}^{k-1} x_i$. Write f as:

$$f(X) = t \cdot P_t\left(x_k, \ldots, x_{n-1}\right) + Q_t(X),$$

where none of the monomials in $Q_t(X)$ is divisible by t. Then the sum of f over all values of the cube (cube sum) is

$$\sum_{x' \in C_t} f\left(x', x_k, \ldots, x_{n-1}\right) = P_t\left(x_k, \ldots, x_{n-1}\right),$$

where the cube C_t contains all binary vectors of the length k.

Cube Tester, cont.

(n+1)-Round Cube Tester on Keccak Sponge Functions. A cube tester can be constructed based on algebraic properties of Keccak sponge function to distinguish a round-reduced Keccak from a random permutation. An adversary can easily select a combination of $2^n + 1$ cube variables such that they are not multiplied with each other after the first round of Keccak. Note that after nround Keccak the degree of these cube variables is at most 2^n. So the adversary can sum the output values over a cube of dimension $2^n + 1$ to get zero for a $(n + 1)$-round Keccak.

Outline

Keccak sponge function

Keccak is a family of sponge function. It can be used as a hash funciton, but can also generate an infinite bit stream, making it suitable to work as s stream cipher or a pseudorandom bit generator.
The sponge function works on a b-bit internal state, divided according to two main parameters r and c, which are called bitrate and capacity rpspectively. Initially, the $(r + c)$ -bit state is filled with 0's, and the message is split into r-bit blocks. Then, the sponge function processes the message in two phases.

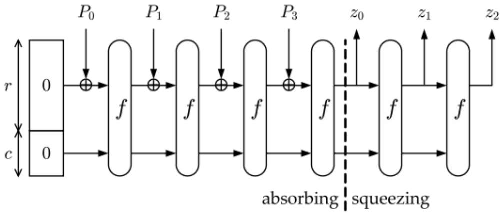


Figure: Keccak sponge construction

Keccak sponge function, cont.

1. **The absorbing phase**: the r-bit message blocks are XORed into the state, interleaved with applications of the internal permutation. After all message blocks have been processed, the sponge functions moves to the second phase (also called squeezing phase).
2. **The squeezing phase**: the first r bits of the state are returned as part of the output interleaved with applications of the internal permutation. The squeezing phase is finished after the disired length of the output digest has been produced.

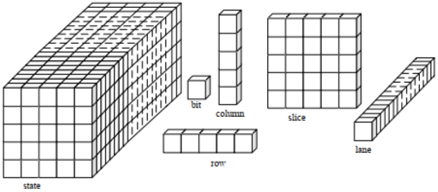


Figure: Terminologies used in Keccak

Analysis of the Keccak compression function

Keccak-$f[b]$ round function R

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta, \text{ with}$$

- θ : $a[x][y][z] \leftarrow a[x][y][z] + \sum_{y'=0}^{4} a[x-1][y'][z] + \sum_{y'=0}^{4} a[x+1][y'][z-1]$
- ρ : $a[x][y][z] \leftarrow a[x][y][z-(t+1)(t+2)/2]$, with t satisfying $0 \le t < 24$ and $\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^{t} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$ in $GF(5)^{2\times 2}$ or $t = -1$ if $x = y = 0$
- π : $a[x][y] \leftarrow a[x'][y']$, with $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$
- χ : $a[x] \leftarrow a[x] + (a[x+1]+1)a[x+2]$
- ι : $a \leftarrow a + RC[i_r]$

Analysis of the Keccak compression function, cont.

Keccak-$f[b]$ round function R

- θ : *Linear* $\deg(\theta(x)) \leq \deg(x)$

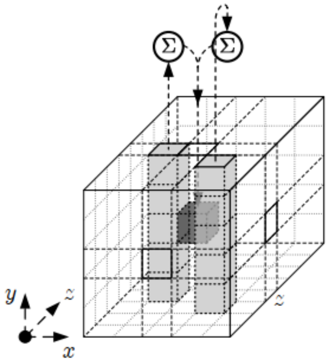


Figure: θ applied to a single bit

Analysis of the Keccak compression function, cont.

Keccak-$f[b]$ round function R

- ρ : *Permutation* $\deg(\rho(x)) = \deg(x)$

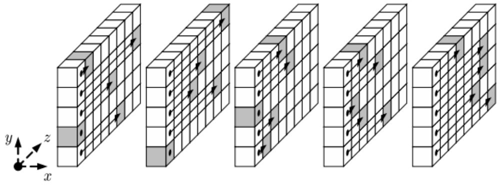


Figure: ρ applied to a slice

Analysis of the Keccak compression function, cont.

Keccak-$f[b]$ round function R

- π : *Permutation* $\deg(\pi(x)) = \deg(x)$

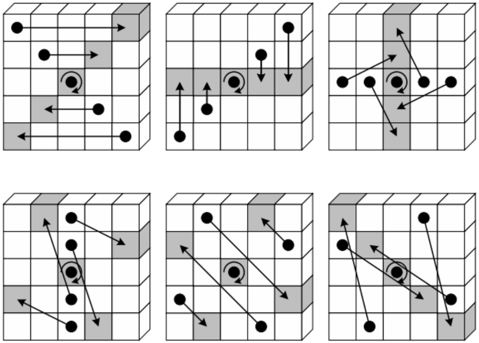


Figure: π applied to a slice

Analysis of the Keccak compression function, cont.

Keccak-$f[b]$ round function R

- χ : Non-linear $\deg(\chi(x)) \leq 2 \cdot \deg(x)$

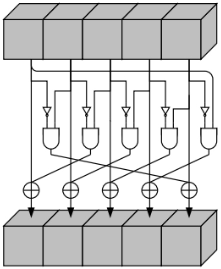


Figure: χ applied to a sigle row

Analysis of the Keccak compression function, cont.

Keccak-$f[b]$ round function R

$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$, with

- θ : *Linear* $\deg(\theta(x)) \leq \deg(x)$
- ρ : *Permutation* $\deg(\rho(x)) = \deg(x)$
- ρ : *Permutation* $\deg(\pi(x)) = \deg(x)$
- χ : Non-linear $\deg(\chi(x)) \leq 2 \cdot \deg(x)$
- ι : *Linear* $\deg(\iota(x)) \leq \deg(x)$

So the degree of Keccak-$f[b]$ at most double with each round.

keyed modes of Keccak

- **MAC based on Keccak** A message authentication code (MAC) is used for verifying data integrity and authentication of a message. A secure MAC is expected to satisfy two main security properties.Assuming that an adversary has access to many valid message-tag pairs:
  - it should be infeasible to recover the secret key used
  - it should be infeasible for the adversary to forge a MAC namely, provide a valid message-tag pair (M, T) for a message M that has not been previously authenticated.

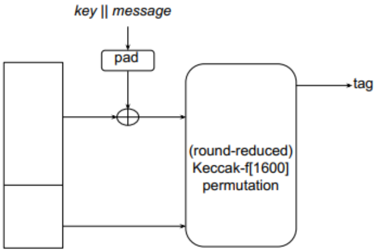


Figure: MAC based on Keccak

keyed modes of Keccak, cont.

- **stream cipher based on Keccak** In the stram cipher mode, the state is initialized with the secret key, concatenated with a public initialization vector (IV). After each Keccak permutation call, an r-bit keystream (where r is the bitrate of the Keccak instance) is extracted and used to encrypt a plaintext via bitwise XOR. In this paper, we only exploit the first r bits of keystream, such that the internal permutation is applied only once.

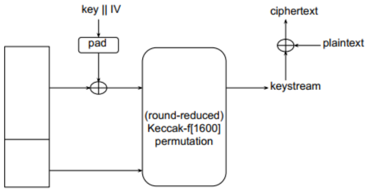


Figure: stram cipher based on Keccak

key recovery attack on 5-round Keccak working as MAC

We attack the default variant of Keccak with a 1600-bit state (r=1024,c=576), where the number of rounds of the internal permutation is reduced to 5. The key and tag sizes are both 128 bit.

- **Preprocessing Phase** To find useful cubes for our attack, we randomly pick 31 out of the 128 public variable and check whether the superpoly consists of any secret variable or it is a constant. With the simple strategy, we have been able to find 117 linearly independent expressions in a few days, but as only 20-25% of the superpolys are useful.
- **Online Phase** The attacker computes the actual binary value of a given superpoly by summing over the outputs obtained from the corresponding cube. There are 19 cubes used in this attack, each cube with 31 variables. Thus, the attacker obtains $19 \cdot 2^{31} \cong 2^{35}$ outputs for the 5-round Keccak.

key recovery attack on 6-round Keccak working in stream cipher mode

- A direct extension of the attack to 6 rounds seems infeasible as we would deal with polynomials of approximate degree $2^6 = 64$ and it is very unlikely to find (in reasonable time) cubes with linear superpolys.
- However, one more round can be reached by exploiting a specific property of the Keccak χ step. As χ operates on the rows independent, if a whole row (5 bits) is known, we can invert these bits through ι and χ from given output bits. Consequently, the final nonlinear step χ can be inverted and the cube attack is reduced to 5.5 round.

Table: Boolean components of χ and χ^{-1}

Output	Corresponding Boolean function	Output	Corresponding Boolean function
χ_0	$x_0 + x_2 + x_1x_2$	χ_0^{-1}	$x_0 + x_2 + x_4 + x_1x_2 + x_1x_4 + x_3x_4 + x_1x_3x_4$
χ_1	$x_1 + x_3 + x_2x_3$	χ_1^{-1}	$x_0 + x_1 + x_3 + x_0x_2 + x_0x_4 + x_2x_3 + x_0x_2x_4$
χ_2	$x_2 + x_4 + x_3x_4$	χ_2^{-1}	$x_1 + x_2 + x_4 + x_0x_1 + x_1x_3 + x_3x_4 + x_0x_1x_3$
χ_3	$x_0 + x_3 + x_0x_4$	χ_3^{-1}	$x_0 + x_2 + x_3 + x_0x_4 + x_1x_2 + x_2x_4 + x_1x_2x_4$
χ_4	$x_1 + x_4 + x_0x_1$	χ_4^{-1}	$x_1 + x_3 + x_4 + x_0x_1 + x_0x_3 + x_2x_3 + x_0x_2x_3$

practical cube tester for 6.5-round Keccak permutation

We show how to construct a practical cube tester for the 6.5-round Keccak permutation. As the expected algebraic degree for 6-round Keccak is 64, such an attack may seem at first impractical. However, if we carefully choose the cube variables, we can exploit a special property of θ in order to considerably reduce the output degree after 6 rounds and keep the complexity low.

- The well-known property of θ is that its action depends on the **column parities** only. Thus, if we set the cube variables in such a way that all the column parties are constants for all the possible variable values, then θ will not diffuse these variables throughout the state.
- Moreover, as ρ and π only permute he bits of the state, it is easy to choose the cube variables such that after the linear part of the round, they are not multiplied with each other through the subsequent non-linear χ layer. Consequently, the algebraic degree of the state bits in the cube variables remains 1 after the first round, and it is at most 32 after 6 rounds.

practical cube tester for 6.5-round Keccak permutation, cont.

- We choose the 33-dimensiional cube $\{v_0, v_1, \ldots, v_{32}\}$ such that $v_i = A[0, 2, i]$, while ensuring that the column parities remain constant by setting the additional linear constraints $A[0, 3, i] = v_i \oplus c_i$, for arbitrary binary constants c_i. In other words, we sum over the outputs of the 33-dimensional linear subspace defined on the 66 state bits $A[0, 2, i], A[0, 3, i]$ by 33 equations $A[0, 2, i] = A[0, 3, i] \oplus c_i$ for $i \in \{0, 1, \ldots, 32\}$. The remaining bits of the iinput state are set to arbitrary constants. At teh input to χ, each 5-bits row of the state contains at most one variable, and therefore, the variables are not multiplied together in the first round as required.

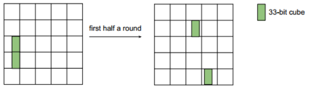


Figure: The initial state of a cube tester and the transition through the first linear part of the round (θ,ρ, π steps)

practical cube tester for 6.5-round Keccak permutation, cont.

- Since the degree of the output polynomials in the cube variable agter 6 rounds in only 32, the cube sum of any output bit after 6 rounds is equal to zero, which is a clear non-random property. Moreover, we can add a (linear) half-round and obtain a 6.5 round distinguisher using the same cube. If we assume that we can obtain sufficiently many output bits in order ot partially invert the non-linear layer, we can extend the attack to 7 rounds in practical time.
- Assume that the 33-dimensional cube is missing an output for one value of the public variable. Then, as the sums of all 2^{33} output is zero, the missing output is equal to the sum of the remaining $2^{33} - 1$ outputs. Thus, the distinguisher attack can be used to predict the output of the cipher for a previously unseen input value.

Divide-and-Conquer key recovery attack on Keccak-based MAC

- As it is not clear how to use the standard key recovery techniques in our case, we use a different approach. The main idea in our attack is to select the public variables of the cube in such a way that the superpolys depend only on a (relatively) small number of key bits, whose value can be recovered independently of the rest of the key. Thus, the full key can be recovered in several phases in a divide-and-conquer manner.
- **Borderline Cubes** The starting point of the attack is the cube tester, which is based on a 33-variable cube, whose column parities remain constant for all of their 2^{33} possible values. As the cube variables are not multiplied together in the first round and the degree of 6-round Keccak in the state variables after one round is $2^5 = 32$, then the cube sums for the 33-variables are zero for all output bits. When we remove one variable from this cube, the sums are no longer guaranteed to be zero and they depend on the values of some of the constant bits of the state. This leaves us a borderline situation.

Divide-and-Conquer key recovery attack on Keccak-based MAC, cont.

If a state bit is not multiplied with the cube variables in the first round, then the cube sums do not depend on the value of this bit. On the other hand, if a state bit is multiplied with the cube variables in the first round, then the cube sums generally depend on the value of the bit (assuming sufficient mixing of the state by the Keccak mapping). Thus, by a careful selection of a ”borderline” cube of dimension 32, we can assure that the cube sums depend only on a (relatively) small number of key bits.

Divide-and -Conquer key recovery attack on Keccak-based MAC, cont.

Basic 6-round attack

- According to the Keccak MAC specification the 128-bit key is placed in $A[0,0]$ and $A[1,0]$. However, it is worth noting that our attack could be easily adapted to any other placements of the secret key. We select 32 cube variables $v_1, v_2, \ldots, v_{32}$ in $A[2,2]$ and $A[2,3]$, such that the column parities of $A[2,*]$ remain constant for the 2^{32} possible values of the variables. This careful selection of the cube variables leads to two properties on which our attack is based:

Property

The cube sum of each output bit after 6 rounds does not depend on the value of $A[1,0]$.

Property

The cube sum of each output bit after 6 rounds depend on the value of $A[0,0]$.

Divide-and -Conquer key recovery attack on Keccak-based MAC, cont.

- We now describe the attack which exploits the two properties to retrieve the value of $A[0,0]$. We separate the attack to preprocessing and online phase, where the preprocessing phase does not depend on the online values of the secret key. However, we take into account both of the phases when calculating the complexity of the full attack.
  The preprocessing phase is described below.
  - Set the capacity lanes $(A[1,4], A[2,4], A[3,4], A[4,4])$ to zero. Set all other state bits (beside $A[0,0]$ and the cube variables) to an arbitrary constant.
  - For each of the 2^{64} possible values of $A[0,0]$: calculate the cube sums after 6 rounds for all the output bits. Store the cube sums in a sorted list L, next to the value of the corresponding $A[0,0]$.

  The online phase, which is retrieves $A[0,0]$, is described below.
  - Request the outputs for the 2^{32} messages that make up the chosen cube (using the same constant as in the preprocessing phase).
  - Calculate the cube sums for the output bits and search them in L
  - For each match in L, retrieve $A[0,0]$ and store all of its possible values.

Divide-and -Conquer key recovery attack on Keccak-based MAC, cont.

Although the actual online value of $A[1,0]$ does not necessarity match its value used during preprocessing ,according to Property 1; it does not affect the cube sums. Thus, we will obtain a match with the correct value of $A[0,0]$. In order to recover $A[1,0]$, we independently apply a similar attack using 32 public variables in $A[4,2]$ and $A[4,3]$ (for which properties corresponding to Property 1 and Property 2 would apply). Finally, in order to recover the full key, we enumerate and test all combinations of the suggestions independently obtained for $A[0,0]$ and $A[1,0]$.

Outline

conditional cube tester for Keccak sponge function

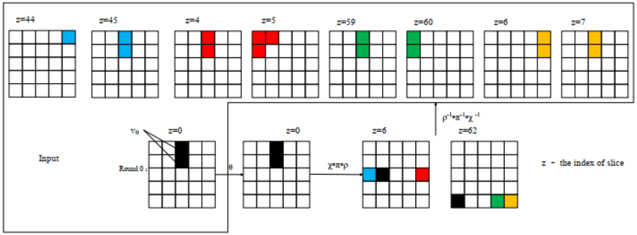


Figure: Overview of bit conditions

The idea of our new model-the *conditional cube tester*, is to attack some bit conditions to a cube tester.

conditional cube tester for Keccak sponge function

Definition

Cube variables that have propagation controlled in the first round and are not multiplied with each other after the second round of Keccak are called **conditional cube variables**. Cube variables that are not multiplied with each other after the first round and are not multiplied with any conditional cube variable after the second round are called **ordinary cube variables**.

Theorem

For $(n+2)$-round Keccak sponge function $(n>0)$, if there are $p\ (0 \leq p < 2^n + 1)$ conditional cube variables $v_1, \ldots, v_p$, and $q = 2^{n+1} - 2p + 1$, ordinary cube variables, $u_1, \ldots, u_q$ (If $q = 0$, we set $p = 2^n + 1$), then the term $v_1 v_2 \ldots v_p u_1 \ldots u_q$ will not appear in the output polynomials of $(n+2)$-round Keccak sponge function.

Properties of Keccak sponge function

Definition

Given a Boolean function $f\left(x_0, x_1, \ldots, x_{n-1}\right)$, the bitwise derivative of f with respect to the variable x_m is defined as

$$\delta_{x_m} f = f_{x_m=1} + f_{x_m=0}$$

The 0-th bitwise derivative is defined to be f itself. The i-th, where $i \geq 2$, bitwise derivative with respect to the variable sequence $\left(x_{m_1}, \ldots, x_{m_i}\right)$ is defined as

$$\delta^{(i)}_{x_{m_1},\ldots,x_{m_i}} f = \delta_{x_{m_i}} \left(\delta^{(i-1)}_{x_{m_1},\ldots,x_{m_{i-1}}} f \right)$$

Properties of Keccak sponge function

Property

*(**Bit Conditions**) IF $\delta_{v_0}F = (1,0,0,0,0)$, then $\delta_{v_0}G = (1,0,0,0,0)$ if and only if $f_1 = 0$ and $f_4 + 1 = 0$.*

Proof.

By the structure of χ, the algebraic representation of the output Boolean function G is given by the following equations:

$$\begin{aligned}
g_0 &= f_0 + (f_1 + 1)f_2 \\
g_1 &= f_1 + (f_2 + 1)f_3 \\
g_2 &= f_2 + (f_3 + 1)f_4 \\
g_3 &= f_3 + (f_4 + 1)f_0 \\
g_4 &= f_4 + (f_0 + 1)f_1
\end{aligned}$$

□

Properties of Keccak sponge function, cont.

Proof.

From the definition of the bitwise derivative, it can be deduced that $\delta_{v_0}G = (1, 0, 0, f_4 + 1, f_1)$. It is clear that $\delta_{v_0}G = (1, 0, 0, 0, 0)$ if and only if $f_1 = 0$ and $f_4 + 1 = 0$. $\square$

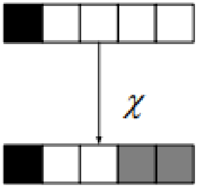
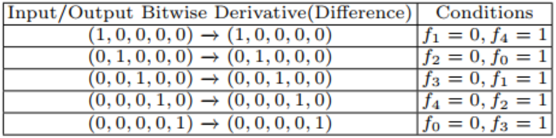


Figure: Diffusion caused byoperation χ

Properties of Keccak sponge function, cont.

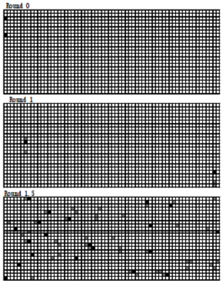


Input/Output Bitwise Derivative(Difference)	Conditions
$(1,0,0,0,0) \to (1,0,0,0,0)$	$f_1 = 0, f_4 = 1$
$(0,1,0,0,0) \to (0,1,0,0,0)$	$f_2 = 0, f_0 = 1$
$(0,0,1,0,0) \to (0,0,1,0,0)$	$f_3 = 0, f_1 = 1$
$(0,0,0,1,0) \to (0,0,0,1,0)$	$f_4 = 0, f_2 = 1$
$(0,0,0,0,1) \to (0,0,0,0,1)$	$f_0 = 0, f_3 = 1$

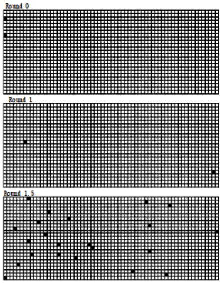
Figure: Summary of conidtions for bitwise derivative of χ

The input and output have the same vector of bitwise derivatives so that the propagation of v_0 by χ is under control. This will be used in constructing our conditional cube tester.

Properties of Keccak sponge function, cont.



(a) Propagation of an ordinary cube variable



(b) Propagation of a conditional cube variable

Figure: 1.5 round differential of an ordinary and a conditional cube variable

Properties of Keccak sponge function, cont.

Property

*(**Multiplication**) Assume that* $\delta_{v_0} F = (\delta_{v_0} f_0, 0, 0, 0, 0)$ *and* $\delta_{v_1} F = (0, \delta_{v_1} f_1, 0, 0, 0)$ *with* $\delta_{v_0} f_0 \cdot \delta_{v_1} f_1 \neq 0$*, then the term* $v_0 v_1$ *will be in the output of* χ*.*

Proof.

The component g_4 of the output $G = (g_0, g_1, g_2, g_3, g_4)$ is $f_4 + (f_0 + 1) f_1$. From

$$\delta^{(2)}_{v_0, v_1} g_4 = \delta_{v_1} \left(\delta_{v_0} g_4\right) = \delta_{v_1} \left(\delta_{v_0} f_0\right) \cdot f_1 + \delta_{v_0} f_0 \cdot \delta_{v_1} f_1 = \delta_{v_0} f_0 \cdot \delta_{v_1} f_1$$

we see that $\delta^{(2)}_{v_0, v_1} g_4 \neq 0$ and hence g_4 contains the term $v_0 v_1$. In particular, if $\delta_{v_0} f_0 = \delta_{v_1} f_1 = 1$, then $g_4 = v_0 v_1 + h$, where h is a Boolean function not divisible by $v_0 v_1$. $\square$

Properties of Keccak sponge function, cont.

Property

*(**Exclusion**) If $\delta_{v_0}F = (1,0,0,0,0)$ and $\delta_{v_1}F = (0,0,1,0,0)$, then at least one of $\delta_{v_0}G = (1,0,0,0,0)$ and $\delta_{v_1}G = (0,0,1,0,0)$ is false.*

Proof.

The conditions $\delta_{v_0}F = (1,0,0,0,0)$ and $\delta_{v_0}G = (1,0,0,0,0)$ would imply $f_1 = 0, f_4 = 1$. Under the assumption $\delta_{v_1}F = (0,0,1,0,0)$, if $\delta_{v_1}G = (0,0,1,0,0)$ also holds true, then we would have $f_1 = 1, f_3 = 0$. □

General process for key recovery attack on Keccak-MAC

(1) Assign free variables with random values.

(2) Guess values of the s equivalent key bits.

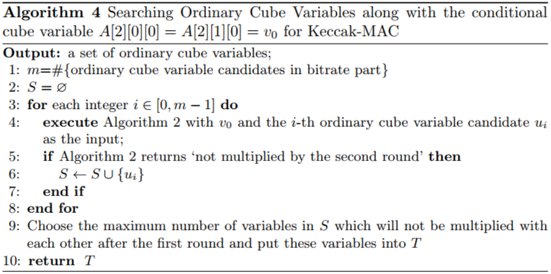
(3) Calculate the values of conditional variables under the guess of key bits.

(4) For each possible set of values of cube variables, compute the corresponding tag and then sum all of the 128-bit tags over the $\left(2^{n+1} - p + 1\right)$- dimension cube.

(5) If the sum is zero, the guess of these s key bits is probable correct and the process terminates; otherwise the guess is invalid, go back to Step 2.

Time Complexity: $\frac{1}{s} \cdot 2^{n+1} - p + s + 8 = \frac{2^{s-p}}{s} \cdot 2^{2^{n+1}+8}$

General process for key recovery attack on Keccak-MAC, cont.



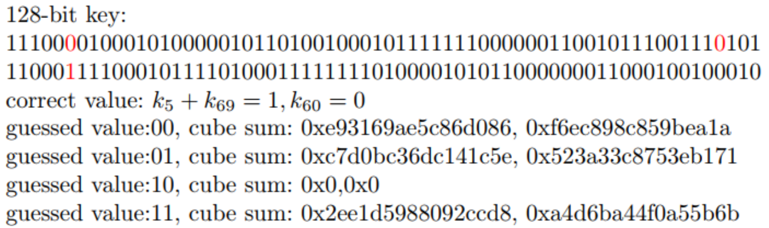
Algorithm 4 Searching Ordinary Cube Variables along with the conditional cube variable $A[2][0][0] = A[2][1][0] = v_0$ for Keccak-MAC

```
Output: a set of ordinary cube variables;
 1: $m$=#{ordinary cube variable candidates in bitrate part}
 2: $S = \varnothing$
 3: for each integer $i \in [0, m-1]$ do
 4:     execute Algorithm 2 with $v_0$ and the $i$-th ordinary cube variable candidate $u_i$
        as the input;
 5:     if Algorithm 2 returns 'not multiplied by the second round' then
 6:         $S \leftarrow S \cup \{u_i\}$
 7:     end if
 8: end for
 9: Choose the maximum number of variables in $S$ which will not be multiplied with
    each other after the first round and put these variables into $T$
10: return $T$
```

Figure: Search ordinary cube variable

key recovery on 5/6/7-round Keccak-MAC

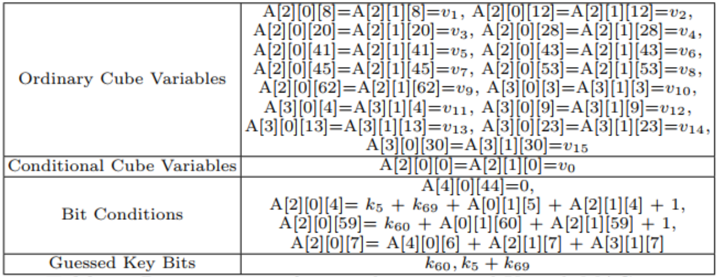
conditional cube variable: $A[2][0][0] = A[2][1][0] = v_0$
bit conditions: $\delta_{v_0}A[2][0][6] = \delta_{v_0}A[2][4][6] = \delta_{v_0}A[4][3][62] = \delta_{v_0}A[4][4][62] = 0$



128-bit key:
1110000100010100000101101001000101111111000000110010111001110101
1100011110001011110100011111111010000010101100000011000100100010
correct value: $k_5 + k_{69} = 1, k_{60} = 0$
guessed value:00, cube sum: 0xe93169ae5c86d086, 0xf6ec898c859bea1a
guessed value:01, cube sum: 0xc7d0bc36dc141c5e, 0x523a33c8753eb171
guessed value:10, cube sum: 0x0,0x0
guessed value:11, cube sum: 0x2ee1d5988092ccd8, 0xa4d6ba44f0a55b6b

Figure: Distinguish the correct key

key recovery on 5/6/7-round Keccak-MAC, cont.



Ordinary Cube Variables	A[2][0][8]=A[2][1][8]=v_1, A[2][0][12]=A[2][1][12]=v_2, A[2][0][20]=A[2][1][20]=v_3, A[2][0][28]=A[2][1][28]=v_4, A[2][0][41]=A[2][1][41]=v_5, A[2][0][43]=A[2][1][43]=v_6, A[2][0][45]=A[2][1][45]=v_7, A[2][0][53]=A[2][1][53]=v_8, A[2][0][62]=A[2][1][62]=v_9, A[3][0][3]=A[3][1][3]=v_{10}, A[3][0][4]=A[3][1][4]=v_{11}, A[3][0][9]=A[3][1][9]=v_{12}, A[3][0][13]=A[3][1][13]=v_{13}, A[3][0][23]=A[3][1][23]=v_{14}, A[3][0][30]=A[3][1][30]=v_{15}
Conditional Cube Variables	A[2][0][0]=A[2][1][0]=v_0
Bit Conditions	A[4][0][44]=0, A[2][0][4]= $k_5 + k_{69}$ + A[0][1][5] + A[2][1][4] + 1, A[2][0][59]= k_{60} + A[0][1][60] + A[2][1][59] + 1, A[2][0][7]= A[4][0][6] + A[2][1][7] + A[3][1][7]
Guessed Key Bits	k_{60}, $k_5 + k_{69}$

Figure: Parameters set for attack on 5-round Keccak-MAC-512

Thank You For Your Attention!