- Affine Transformation
- Affine Equivalence
- Problem
  - Problem 1. Linear equivalence
  - Problem 2. Affine equivalence
- The Linear Equivalence Algorithm [1]
  - Naive approach
  - Improved Naive approach
  - Linear equivalence algorithm
    - Notation
    - Exploit two ideas
    - Complexity of LE
- The Affine Equivalence Algorithm [1]
  - A straightforward solution
  - A second approach
  - Finding the linear representative
    - Defination of linear representative
    - Construst the representative R_S
    - Explain the algorithm
    - Update the sets
    - Complexity of AE
- Complexities of linear and affine algorithms
- Extensions of LE and AE algorithms
  - Self-Equivalent S-Boxes
  - Equivalence of Non-invertible S-Boxes
  - Almost Affine Equivalent S-Boxes
- Apply AE and LE to AES
- An Improved Affine Equialence Algorithm (Dinur) [2]
  - Multivariate Polynomials
  - Half-Space Masks and Coefficients
  - Canonical Affine Transformations
  - Rand Tables and Histograms
  - The New Affine Equivalence Algorithm
- Reference

Affine Transformation

An `affine transformation` $A : \{0,1\}^m \to \{0,1\}^n$ over $GF(2)^m$ is defined using a Boolean matrix $L_{n\times m}$ and a word $a \in \{0,1\}^n$ as

$$A(x) = L(x) + a$$

$L(x)$ is simply matrix multiplication.

The transformation is **invertible** if $m = n$ and L is an invertible matrix.

If $a = 0$, then the A is called a `linear transformation` (such functions are a subclass of affine functions),that is,

$$A(x) = L(x)$$

Affine Equivalence

Two functions $\boldsymbol{F} : \{0,1\}^n \to \{0,1\}^m, \boldsymbol{G} : \{0,1\}^n \to \{0,1\}^m$ are `affine equivalent` is there exist two **invertible affine transformations** $A_1 : \{0,1\}^n \to \{0,1\}^n$ and $A_2 : \{0,1\}^m \to \{0,1\}^m$ such that

$$\boldsymbol{G} = A_2 \circ \boldsymbol{F} \circ A_1$$

It is easy to show that the affine equivalence relation partitions the set of all functions into (affine) equivalence classes. We denote

$$\boldsymbol{F} \equiv \boldsymbol{G}$$

if $\boldsymbol{F}$ is affine equivalent to $\boldsymbol{G}$.

Problem

Problem 1. Linear equivalence

By linear mapping we mean a mapping $L(x)$ over $GF(2)^n$ that satisfies

$$L(x + y) = L(x) + L(y)$$

Let us consider the prblem of checking `linear equivalence` between two S-boxes S_1 and S_2. The problem is to find two **invertible linear mappings** L_1 and L_2, such that

$$L_2 \circ S_1 \circ L_1 = S_2$$

Problem 2. Affine equivalence

We want an algorithm that takes two $n \times n$-bit S-boxes S_1 and S_2 as input, and checks whether there exists a pair of **invertible affine mappings** A_1 and A_2 such that

$$A_2 \circ S_1 \circ A_1 = S_2$$

The Linear Equivalence Algorithm [1]

Naive approach

Guess one of the mappings, for example L_1. Then one can extract L_2 from the equation:

$$L_2 = S_2 \circ L_1^{-1} \circ S_1^{-1}$$

and check if it is a linear, invertible mapping.

There are $O(2^{n^2})$ choices of invertible linear mappings over n-bit vectors.
For each guess one will need about n^3 steps to check for linearity and invertibility using `Gaussian elimination` [1].

In total the naive algorithm would require $O(n^3 2^{n^2})$ steps.

A similar naive affine equivalence algorrithm will use $O(n^3 2^{n(n+1)})$.

Improved Naive approach

We need only n equations in order to check L_2 for invertibility and linearity.
If one guesses only $\log_2 n$ vectors from L_1 one may span a space of n points (by trying all linear combinations of the guessed vectors), evaluate the results through L_1, S_1 and S_2 and have n constraints required to check for linearity of L_2.
If the n new equations are not independent one will need to guess additional vectors of L_1.

Such an algorithm would require guessing of $n \log_2 n$ bits of L_1 and the total complexity would be $O(n^3 2^{n \log n})$

Linear equivalence algorithm

Notation

Symbol	Notation
A, B^{-1}	the linear mappings L_1 and L_2 respectively
C_A, C_B	the sets of ***checked points*** for which the mapping (A or B respectively) is known
U_A, U_B	the sets of yet ***unknown points***
N_A, N_B	all the ***new points*** for which we know the mapping (either A or B, respectively), but which are linearly independent from points of C_A or C_B, respectively

C, N, U are always disjoint.

Exploit two ideas

1. `needlework effect` : in which guesses of portions from L_1 provide us with free knowledge of the values of L_2.
2. `exponential amplification of guesses` : due to the linear (affine) structure of the mappings, support that the new values from L_2 allow us to extract new free information about L_1.
3. knowing k vectors from the mapping L_1, we know 2^k linear combinations of these vectors for free.

Linear Equivalence (LE)

$U_A \Leftarrow \{0,1\}^n; U_B \Leftarrow \{0,1\}^n$
$N_A \Leftarrow \varnothing; N_B \Leftarrow \varnothing$
$C_A \Leftarrow \varnothing; C_B \Leftarrow \varnothing$
while ($U_A \neq \varnothing$ and $U_B \neq \varnothing$) or (All guesses rejected) **do**
  if $N_A = \varnothing$ and $N_B = \varnothing$ **then**
    If previous guess rejected, restore C_A, C_B, U_A, U_B.
    Guess $A(x)$ for some $x \in U_A$
    Set $N_A \Leftarrow \{x\}$, $U_A \Leftarrow U_A \setminus \{x\}$
  end if
  while $N_A \neq \varnothing$ **do**
    Pick $x \in N_A$; $N_A \Leftarrow N_A \setminus \{x\}$; $N_B \Leftarrow S_2(x \oplus C_A) \setminus C_B$
    $C_A \Leftarrow C_A \cup (x \oplus C_A)$
    if $|N_B| + \log_2 |C_B| > const \cdot n$ **then**
      if B is invertible linear mapping **then**
        Derive A and check A, B at all points, that are still left in U_A and U_B.
      else
        Reject latest guess; $N_A \Leftarrow \varnothing; N_B \Leftarrow \varnothing$
      end if
    end if
  end while
  while $N_B \neq \varnothing$ **do**
    Pick $y \in N_B$; $N_B \Leftarrow N_B \setminus \{y\}$; $N_A \Leftarrow {S_2}^{-1}(y \oplus C_B) \setminus C_A$
    $C_B \Leftarrow C_B \cup (y \oplus C_B)$
    if $|N_B| + \log_2 |C_B| > const \cdot n$ **then**
      if A is invertible linear mapping **then**
        Derive B and check A, B at all points, that are still left in U_A and U_B.
      else
        Reject latest guess; $N_A \Leftarrow \varnothing; N_B \Leftarrow \varnothing$
      end if
    end if
  end while
  $U_A \Leftarrow U_A \setminus C_A$; $U_B \Leftarrow U_B \setminus C_B$
end while

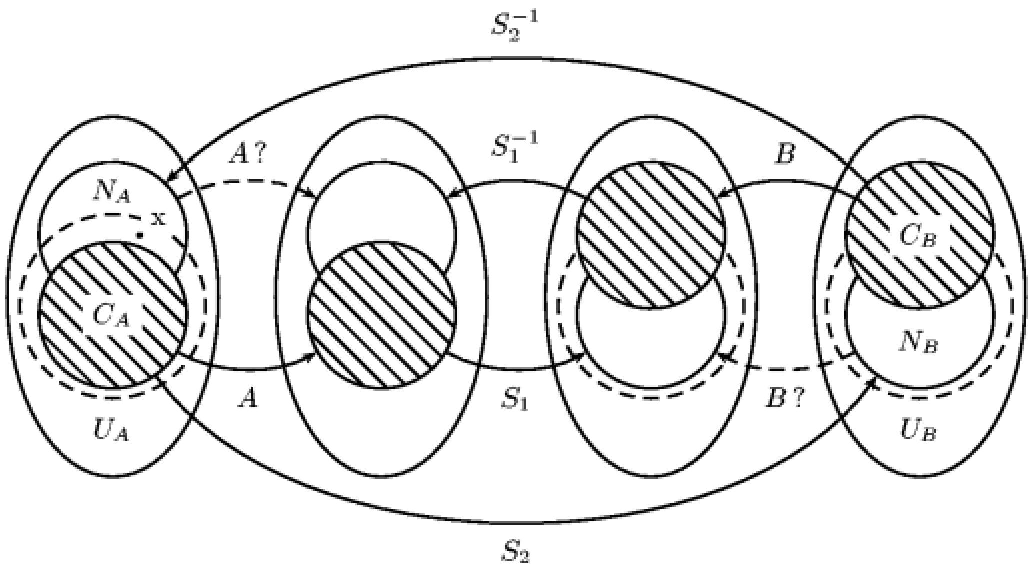

Fig. 1. The relations between the different sets for the LE algorithm.

Complexity of LE

The complexity of this approach is about $n^3 \cdot 2^n$ steps (for S-boxes that do not map zero to zero, and $n^3 \cdot 2^{2n}$ otherwise).
The complexity of the linear equicalence algorithm (LE) is $O(n^3 2^n)$.

The Affine Equivalence Algorithm [1]

$$A_2 \circ S_1 \circ A_1 = S_2$$

$$\Downarrow$$

$$A_2(S_1(A_1(x))) = S_2(x)$$

$$A_1(x) = A \cdot x \oplus a$$

$$A_2(x) = B^{-1} \cdot x \oplus b$$

$$\Downarrow$$

$$B^{-1} S_1(A \cdot x \oplus a) \oplus b = S_2(x)$$

$$\Downarrow$$

$$B^{-1} \circ S_1(x \oplus a) \circ A = S_2(x) \oplus b$$

$$\Downarrow$$

$S_1(x \oplus a)$, $S_2(x) \oplus b$ are **linearly equivalent**

A straightforward solution

for all a **do**
  for all b **do**
    check whether $S_1(x \oplus a)$ and $S_2(x) \oplus b$ are linearly equivalent
  end for
end for

This approach adds a factor 2^{2n} to the complexity of the linear algorithm, bringing the total to $O(n^3 2^{3n})$.

A second approach

Assign **a unique representative** to each linear equivalence class:

for all a **do**
  insert the representative of the lin. equiv. class of $S_1(x \oplus a)$ in a table T_1
end for
for all b **do**
  insert the representative of the lin. equiv. class of $S_2(x) \oplus b$ in a table T_2
end for
if $T_1 \cap T_2 \neq \varnothing$ **then**
  conclude that S_1 and S_2 are affine equivalent
end if

The complexity of this second algorithm is about 2^n times the work needed for **finding the linear representative**.

Finding the linear representative

The efficiency of an algorithm that finds the `linear representative` R_S for an S-box S depends on how this unique representative is chosen.

Defination of linear representative

If all S-boxes in a linear equivalence class are ordered lexicographically according to their lookup tables, then the *smallest* is called the representative of the class.

Construst the representative R_S

1. Making an initial guess.
2. Incrementally build the linear mappings A and B such that $R'_S = B^{-1} \circ S \circ A$ is as small as possible.
3. The representative R_S is obtained by taking the smallest R'_S over possible guesses.

Explain the algorithm

Symbol	Notation
A, B^{-1}	the linear mappings L_1 and L_2 respectively
D_A, D_B	values for which A or B are known respectively
C_A, C_B	points of D_A that have a corresonding point in D_B and vice versa, i.e., $S \circ A(C_A) = B(C_B)$. For these values, R'_S and R'^{-1}_S are known respectively
N_A, N_B	remaining points of D_A and D_B. We have that $S \circ A(N_A) \cap B(N_B) = \varnothing$
U_A, U_B	values for which A or B can still be chosen

The main part of the algorithm that finds a candidate R'_S consists in **repeatedly picking the smallest input** x for which R'_S is not known and trying to **assign it to the smallest available output** y.

while $N_A \neq \varnothing$ **do**
  pick $x = \min_{t \in N_A}(t)$ and $y = \min_{t \in U_B}(t)$
  complete B such that $B(y) = S \circ A(x)$ and thus $R'_S(x) = y$
  update all sets according to their definitions
  while $N_A = \varnothing$ and $N_B \neq \varnothing$ **do**
    pick $x = \min_{t \in U_A}(t)$ and $y = \min_{t \in N_B}(t)$
    complete A such that $A(x) = S^{-1} \circ B(y)$ and thus $R'_S(x) = y$
    update all sets according to their definitions
  end while
end while

Update the sets

1. Use the value of $B(y)$ to derive B for all linear combinations of y and D_B.

2.

$$D'_B \Leftarrow D_B \cup (D_B \oplus y)$$

$$U'_B \Leftarrow U_B \setminus (D_B \oplus y)$$

2. Check whether any new point inserted in D_B has a corresponding point in D_A and update C_B, D_B, C_A and N_A accordingly:
3.

$$C'_B \Leftarrow C_B \cup B^{-1}[B(D_B \oplus y) \cap S \circ A(N_A)]$$

$$N'_B \Leftarrow N_B \cup B^{-1}[B(D_B \oplus y) \setminus S \circ A(N_A)]$$

$$C'_A \Leftarrow C_A \cup A^{-1} \circ S^{-1}[B(D_B \oplus y) \cap S \circ A(N_A)]$$

$$N'_A \Leftarrow N_A \setminus A^{-1} \circ S^{-1}[B(D_B \oplus y) \cap S \circ A(N_A)]$$

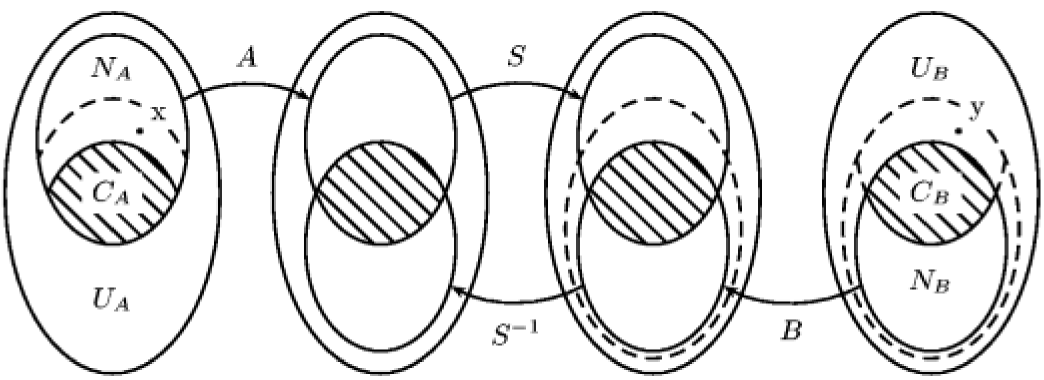

Fig. 2. The relations between the different sets for the AE algorithm.

Complexity of AE

Completely defining R'_S for a particular guess takes about 2^n steps.
However, most guesses will already be rejected after having determined only slightly more than n values, because at that point R'_S will usually turn out to be larger than the current smallest candidate.
The total complexity of finding the representative is expected to be $O(n^3 2^n)$.
The affine equivalence algorithm (AE) has complexity $O(n^3 2^{2n})$.

Complexities of linear and affine algorithms

Table 2. Complexities of linear and affine algorithms.

Dimension	$: n$	4	5	6	7	8	9	10	12	16	24	32
LE	$: n^2 2^n$	2^8	2^{10}	2^{11}	2^{13}	2^{14}	2^{15}	2^{17}	2^{19}	2^{24}	2^{33}	2^{42}
AE	$: n^2 2^{2n}$	2^{12}	2^{15}	2^{17}	2^{20}	2^{22}	2^{24}	2^{27}	2^{31}	2^{40}	2^{57}	2^{74}
AE $(n-m=1)$	$: 2^n n^2 (2!)^{\frac{n}{2}}$	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{25}	2^{32}	2^{45}	2^{58}
AE $(n-m=2)$	$: 2^n n^2 (2^2!)^{\frac{n}{2^2}}$	2^{13}	2^{15}	2^{18}	2^{21}	2^{23}	2^{26}	2^{28}	2^{33}	2^{42}	2^{61}	2^{79}
AE $(n-m=3)$	$: 2^n n^2 (2^3!)^{\frac{n}{2^3}}$	2^{16}	2^{19}	2^{23}	2^{26}	2^{29}	2^{33}	2^{36}	2^{42}	2^{55}	2^{79}	2^{103}

Note that we use n^2 for the complexity of the Gaussian elimination since $n \leq 32$ and we assume an efficient implementation using 32-bit operations.

Extensions of LE and AE algorithms

Self-Equivalent S-Boxes

$$A_2 \circ S \circ A_1 = S$$

Equivalence of Non-invertible S-Boxes

a natura extension of our equivalence problem:
find an $n \times n$-bit affine mapping A_1 and an $m \times m$-bit affine mappin A_2 such that

$$A_2 \circ S_1 \circ A_1 = S_2$$

for two given $n \times m$-bit S-boxes S_1 and S_2.

Almost Affine Equivalent S-Boxes

The S-boxes S_1 and S_2 are called `almost equivalent` if there exist two affine mappings A_1 and A_2 such that $A_2 \circ S_1 \circ A_1$ and S_2 **are equal, except in a few points** (e.g., two values in the lookup table are swapped, or some fixed fraction of the entrise are misplaced).

Apply AE and LE to AES

When our AE tool is run for the 8-bit S-box S used in AES, as many as **2040 different self-equivalence relations** are revealed.
Although this number might seem surprisingly high art first, we will show that it can easily be explained from **the special algebraic structure of the S-box of AES**.

Symbol	Notation
$[a]$	the 8×8-bit matrix that corresponds to a multiplication by a in $GF(2^8)$
Q	the 8×8-bit matrix that performs the squaring operation in $GF(2^8)$

We can now derive a general expression for all pairs of affine mappings A_1 and A_2 such that $A_2 \circ S \circ A_1 = S$:

$$A_1(x) = [a] \cdot Q^i \cdot x$$

$$A_2(x) = A(Q^{-i} \cdot [a] \cdot A^{-1}(x)), \text{with } 0 \leq i < 8 \text{ and } a \in GF(2^8) \setminus \{0\}$$

Since i takes on 8 different values [2] and there are 255 different choices for a, we obtain exactly $2040 = 255 \times 8$ different solutions, which confirms the output of the AE algorithm.

An Improved Affine Equialence Algorithm (Dinur) 2

Multivariate Polynomials

Any `Boolean function` $F : \{0,1\}^n \rightarrow \{0,1\}$ can be representend as a `multivariate polynomial` whose `algebraic mormal form (ANF)` is unique and given as

$$F(x[1], \cdots, x[n]) = \sum_{u=(u[1],\cdots,u[n]) \in \{0,1\}^n} \alpha_u M_u$$

where $\alpha_u \in \{0,1\}$ is the `coefficient` of the `monomial` $M_u = \prod_{i=1}^n x[i]^{u[i]}$, and the sum is over $GF(2)$.

The `algebraic degree` of the function F is defined as

$$deg(F) = \max\{wt(u) | \alpha_u \neq 0\}$$

Given a function $F : \{0,1\}^n \rightarrow \{0,1\}$ represented by a polynomial

$$P(x[1], \cdots, x[n])$$

define $F_{\geq d} : \{0,1\}^n \to \{0,1\}$ as the function represented by

$$P_{(\geq d)}$$

Half-Space Masks and Coefficients

Let $A : \{0,1\}^{n-1} \to \{0,1\}^n$ be an affine transformation such that $A(x) = L(x) + a$ for a matrix $L_{n \times (n-1)}$ with linearly independent columns.

Then the (affine) range of A is an $(n-1)$-dimensional affine subspace spanned by the columns of L with the addition of a .

The subspace orthogonal to the range of A is of dimension 1 and hence spanned by a single non-zero vector $h \in \{0,1\}^n$.

Namely, a vector $v \in \{0,1\}^n$ is in the range of A if and only if $h(v+a) = 0$ i.e., v satisfies the linear equation $h(v) + h(a) = 0$.

Sine h partitions the space of $\{0,1\}^n$ into two halves, we call h the `half-space mask (HSM)` of A

and call the bit $h(a)$ the `half-space free coefficient (HSC)` of A.

We call the linear subspace spanned by the columns of L the `linear range` of A. A vector $v \in \{0,1\}^n$ is in the linear range of A if and only if $h(v) = 0$.

Canonical Affine Transformations

$$\text{non-zero } h \in \{0,1\}^n$$

$$c \in \{0,1\}$$

Define the `canonical affine transformation` $C_{|h,c} : \{0,1\}^{n-1} \to \{0,1\}^n$ with respect to h, c.
ℓ: the index of the first non-zero bit of $h = (h[1], \cdots, h[n])$.

$$C_{|h,c}(x) = L(X) + a$$

$$a = c \cdot e_\ell$$

e_ℓ: the ℓ'th unit vector

$$L[i]=\begin{cases} e_i & \text{if } i<\ell \\ e_{i+1}+h[i+1]_{e_\ell} & \text{otherwise } (\ell \leq i \leq n-1) \end{cases}$$

$$\Downarrow$$

the transformation $C_{|h,c}$ is defined by the symbolic form:

$$(x[1],x[2],\cdots,x[n])=(y[1],\cdots,y[\ell-1]),\sum_{i=\ell}^{n-1}h[i+1]y[i]+c,y[\ell],\cdots,y[n-1])$$

Rand Tables and Histograms

prove $\boldsymbol{F}$ and $\boldsymbol{G}$ are affine equivalent $\Rightarrow$ the symbolic ranks of $\boldsymbol{P}$ and $\boldsymbol{Q}$ (as vectors) are equal
`rank table`

`rank group` : $(\max R, \min R)$

Although the *rank tables are different*, the size of each rank group $(\max R, \min R)$ of $\boldsymbol{F}, \boldsymbol{G}$ is *identical*.

the `rank histogram` of $\boldsymbol{F}$ (with respect to d) as a mapping from each $(\max R, \min R)$ value to the corresponding **rank group size**.

The New Affine Equivalence Algorithm

$$\boldsymbol{r}_1=(n+1-\gamma_n, n-\gamma_n)$$

$$\boldsymbol{r}_2=(n, n-\gamma_n)$$

$$\gamma_n=\left\lfloor (n/2)^{1/2} \right\rfloor$$

1. Given $\boldsymbol{F}:\{0,1\}^n \rightarrow \{0,1\}^n, \boldsymbol{G}:\{0,1\}^n \rightarrow \{0,1\}^n$
   Compute their corresonding `ANF` representations $\boldsymbol{P}_{\geq(n-2)}$ and $\boldsymbol{Q}_{\geq(n-2)}$
2. Compute the `rank table` $\mathcal{T}_{\boldsymbol{F},n-2}$ and `rank histogram` $\mathcal{H}_{\boldsymbol{F},n-2}$ for $\boldsymbol{F}$ using $\boldsymbol{P}_{\geq(n-2)}$.
   Compute $\mathcal{T}_{\boldsymbol{G},n-2}$ and $\mathcal{H}_{\boldsymbol{G},n-2}$ for $\boldsymbol{G}$ using $\boldsymbol{Q}_{\geq(n-2)}$.
   If $\mathcal{H}_{\boldsymbol{F},n-2} \neq \mathcal{H}_{\boldsymbol{G},n-2}$, return "Not Equivalent".
   Algorithm to compute the rank table and rank histogram
   For each non-zero HSM $h \in \{0,1\}^n$:
   (1) Compute $R_{\boldsymbol{F},n-2,h}=(\max R, \min R)$ as follows.
   Compute $(\boldsymbol{P}_{\geq n-2} \circ C_{|h,0})_{\geq n-2}=(\boldsymbol{P} \circ C_{|h,0})_{\geq n-2}$ and calculate its symbolic rank r_0 using

`Guassian elimination`.

Compute $(\boldsymbol{P}_{\geq n-2} \circ C_{|h,1})_{\geq n-2} = (\boldsymbol{P} \circ C_{|h,1})_{\geq n-2}$ and calculate its symbolic rank r_1 using `Guassian elimination`.

Let $\max R = \max\{r_0, r_1\}$ and $\min R = \min\{r_0, r_1\}$.

(2) Insert h into $\mathcal{T}_{\boldsymbol{F},n-2}(\max R, \min R)$, along with the value of the attached constant $c \in \{0,1\}$ such that $\max R = SR((\boldsymbol{F} \circ C_{|h,c})_{(\geq n-2)})$ (if $\max R > \min R$) In addition, increment entry $\mathcal{H}_{\boldsymbol{F},n-2}(\max R, \min R)$.

Note: <u>The time complexity of the algorithm depends on how a polynomial is represented.</u>

3. Obtain the set $U_{\boldsymbol{F}}$ running the algorithm on inputs $\mathcal{T}_{\boldsymbol{F},n-2}$ and $\boldsymbol{r}_1, \boldsymbol{r}_1$

   Obtain the set $U_{\boldsymbol{G}}$ running the algorithm on inputs $\mathcal{T}_{\boldsymbol{G},n-2}$ and $\boldsymbol{r}_1, \boldsymbol{r}_1$

   The Unique HSM Algorithm

   (1) For each $h \in \mathcal{T}_{\boldsymbol{F},n-2}(\boldsymbol{r})$, compute $\mathcal{HG}_{\boldsymbol{F},n-2,h,\boldsymbol{r'}}$ as follows:

   a. for each $h' \in \mathcal{T}_{\boldsymbol{F},n-2}(\boldsymbol{r'})$:

   Compute $h + h'$, find its rank $\boldsymbol{r''} = R_{\boldsymbol{F},n-2,h+h'}$ in $\mathcal{T}_{\boldsymbol{F},n-2}$.

   b. Insert $\mathcal{HG}_{\boldsymbol{F},n-2,h,\boldsymbol{r'}}$ along with h and its attached constant c into the multi-set $\mathcal{HM}_{\boldsymbol{F},n-2,\boldsymbol{r},\boldsymbol{r'}}$.

   (2) For each unique HSM h in $\mathcal{HM}_{\boldsymbol{F},n-2,\boldsymbol{r},\boldsymbol{r'}}$, add the triplet $(h, c, \mathcal{HG}_{\boldsymbol{F},n-2,h,\boldsymbol{r'}})$ to $U_{\boldsymbol{F}}$.

   Note: <u>The time complexity of the algorithm is product of sizes of the rank groups.</u>

4. On inputs $U_{\boldsymbol{F}}$ and $U_{\boldsymbol{G}}$ to recover affine transformation

   If it returns "Not Equivalent", return the same output.

   Otherwise, it returns a candidate for A_1.

   The Affine Transformation A_1 Recovery Algorithm

   (1) Allocate $n+1$ linear equation systems $\{E_i\}_{i=1}^{n+1}$, each of dimension $n \times n$: the first n equation systems are on the columns $L[i]$ of L and the final equation system E_{n+1} is on a.

   (2) Locate n linearly independent HSMs in $U_{\boldsymbol{G}}$.For each such HSM h:

   Lemma 5

   a. Recover the triplet $(h, c, \mathcal{HG}_{\boldsymbol{G},n-2,h,\boldsymbol{r'}})$ from $U_{\boldsymbol{G}}$.

   b. Search $U_{\boldsymbol{F}}$ for a triplet $(h', c', \mathcal{HG}_{\boldsymbol{F},n-2,h',\boldsymbol{r'}})$ such that $\mathcal{HG}_{\boldsymbol{F},n-2,h',\boldsymbol{r'}} = \mathcal{HG}_{\boldsymbol{G},n-2,h,\boldsymbol{r'}}$.

   If no mathch exists, return "Not Equivalent".

   c. Based on Lemma 5, for $i = 1, 2, \cdots, n$ add equation $h'(L[i]) = h[i]$ to E_i.

   d. Based on Lemma 5, add equation $h'(a) = c + c'$ to E_{n+1}.

   (3) Solve each one of $\{E_i\}_{i=1}^{n+1}$, recover A_1 and return its matrix L and vector a.

5. Recover a candidate for $A_2 = L_2(x) + a_2$ by evaluating inputs $v \in \{0,1\}^n$ to $\boldsymbol{F} \circ A_1$ and $\boldsymbol{G}$: each input v gives n linear equations on L_2 and a_2.

   Hence, after a bit more than n evaluations, we expect the linear equation system to have a single solution which gives a candidate for A_2.

6. Test the candidates A_1, A_2 by equating the evaluations of $\boldsymbol{G}$ and $A_2 \circ \boldsymbol{F} \circ A_1$ on all 2^n possible inputs.

   If $\boldsymbol{G}(v) \neq A_2 \circ \boldsymbol{F} \circ A_1(v)$ for some $v \in \{0,1\}^n$, return "Not Equivalent".

   Otherwise, return A_1, A_2.

Reference

[1] Biryukov A, De Canniere C, Braeken A, et al. A toolbox for cryptanalysis: Linear and affine equivalence algorithms[C]. International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Berlin, Heidelberg, 2003: 33-50.

[2] Dinur I. An improved affine equivalence algorithm for random permutations[C]. Annual International Conference on the Theory and Applications of Cryptographic Techniques. Springer, Cham, 2018: 413-442.

1. 高斯消元法算法的时间复杂度为 $O(n^3)$ ↩
2. One can easily check that $Q^8 = I$ and thus $Q^{-i} = Q^{8-i}$ ↩