- CEJO framework
- Conclusion
- Baek et al's Affine Equivalence Algorithm with Multiple S-boxes [1]
  - Theorem 3
  - Prove
    - Two Suppose
    - Set
    - Consider affine equivalence problem
  - Without the oracle of the inverse of F
  - When A is split
- Introduction of the Generic Algorithm [2]
  - Problem 1
  - Remark 1
    - access to F, but not to F^{-1}
    - access to both F and F^{-1}
  - Remark 2
  - Remark 3
  - Remark 4
- Overview of the Algorithm
  - Computing the V_i's
  - Computing the I_i's
  - Recovering affine layers
    - the case of different S-boxes
- Complexity of the algorithm
- Reference

CEJO framework

Basic strategy: the whole cipher is decomposed into round functions and the round functions are represented by summation of *lookup tables with small size*.

Chow et al.'s strategy (a white-box AES implementation and a white-box DES implementation) provided a framework, called " `CEJO framework` ", for designing white-box implementation of using table lookups.

Conclusion

1. As we can see from previous implementations, it is very difficult to design a white-box implementation with a security level similar to the black-box model.
   Hence, the practical objective of white-box implementations is to **increase the complexity of cryprtanalysis**.
2. All of the implementations mentioned above suffered unpredicted attacks soon after their designs were announced.
   This is mainly because **there are no standard attack tools** such as differential cryptanalysis and linear cryptanalysis for block ciphers.

Baek et al's Affine Equivalence Algorithm with Multiple S-boxes [1]

Theorem 3

Let F and S be two permutations on n bits where $S = (S_1, \cdots, S_k)$ with nonlinear permutations S_i on m bits for $i = 1, \cdots, k$.
Assume that we can easily access the inversion of F.
Then, we can find all affine mappings A and B such that $F = B \circ S \circ A$ in time $O(kn^3 2^{3m})$ if they exists.

Prove

First, we assume that F and S are `linear equivalent`.

Suppose that A and B are `invertible linear mapping` over $\mathbb{Z}_2^n$ with $F = B \circ S \circ A$.

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_k \end{bmatrix}, B^{-1} = \begin{bmatrix} B_1 \\ \vdots \\ B_k \end{bmatrix}, S = \begin{bmatrix} S_1 \\ \vdots \\ S_k \end{bmatrix} \qquad (1)$$

fine two sets:

$$\{x_1, x_2, \cdots, x_n\}$$

$$\{B_i \circ F(x_1), B_i \circ F(x_2), \cdots, B_i \circ F(x_n)\}$$

such that

$$\{F(x_1), F(x_2), \cdots, F(x_n)\}^{-1}$$

is `linearly independent` in order to recocer B_i

$$B_i = [B_i \circ F(x_1), B_i \circ F(x_2), \cdots, B_i \circ F(x_n)][F(x_1), F(x_2), \cdots, F(x_n)]^{-1} \tag{2}$$

Two Suppose

1. ***Suppose***: two sets $\{x_1, \cdots, x_\ell\}$ and $\{y_1 = B_i \circ F(x_1), \cdots, y_\ell = B_i \circ F(x_\ell)\}$ such that $\{x_1, \cdots, x_\ell\}$ is `linearly independent` .

$$x = \sum_{j=1}^{\ell} b_j x_j (b_j \in \{0, 1\})$$

$$\Downarrow$$

compute $y = B_i \circ F(x)$ from $y_1, \cdots, y_\ell$

$$\Downarrow$$

$$y = S_i \circ A_i(x) = S_i(\sum_{j=1}^{\ell} b_j A_i(x_j)) = S_i(\sum_{j=1}^{\ell} b_j S_i^{-1}(y_i)) \tag{3}$$

F is a `nonlinear bijection`

$$\Downarrow$$

$$F(x) \notin \mathbb{Z}_2 F(x_1) + \cdots + \mathbb{Z}_2 F(x_\ell)$$

with high probability by assuming F is random bijection.

2. ***Suppose***: two sets $\{F(x_1), \cdots, F(x_\ell)\}$ and $\{y_1 = B_i \circ F(x_1), \cdots, y_\ell = B_i \circ F(x_\ell)\}$ such that $\{F(x_1), \cdots, F(x_\ell)\}$ is `linearly independent` .

$$x' = F^{-1}(\sum_{j=1}^{\ell} b'_j F(x_j))(b'_j \in \{0, 1\})$$

$$\Downarrow$$

compute $y' = B_i \circ F(x')$ from $y_1, \cdots, y_\ell$

$$\Downarrow$$

$$y' = B_i \circ F(F^{-1}(\sum_{j=1}^{\ell} b'_j F(x_j))) = \sum_{j=1}^{\ell} b'_j B_i \circ F(x_j) = \sum_{j=1}^{\ell} b'_j y_j \qquad (4)$$

F^{-1} is a `nonlinear bijection`

$$\Downarrow$$

$$x' \notin \mathbb{Z}_2 x_1 + \cdots + \mathbb{Z}_2 x_\ell$$

with high probability by assuming F^{-1} is random bijection.

Set

$$x_0 = 0, y_0 = B_i \circ F(x_0), F(x_1) = 0 \Rightarrow x_1 = F^{-1}(0)$$

$$y_0 = S_i \circ A(x_0) = S_i(0), y_1 := B_i \circ F(x_1) = 0$$

$$x_2 \in \{0,1\}^n \setminus \{x_0, x_1\}, y_2 := B_i \circ F(x_2)$$

x_1, x_2 are `linearly independent`.

$$x_3 = x_1 + x_2, F \text{ is nonlinear}, x_3 \notin \{x_0, x_1, x_2\} \Rightarrow F(x_3) \notin \mathbb{Z}_2 F(0) + \mathbb{Z}_2 F(2)$$

1. repeat above process in the equation $y = S_i \circ A_i(x)$ and $y' = B_i \circ F(x')$ several times
2. obtain n vectors whose F values are linearly independent
3. For each successful guessing, get an $m \times n$ linear mapping B_i

$$F = B \circ S \circ A$$

$$\Downarrow$$

$$B_i \circ F(x) = S_i \circ A_i(x)$$

$$\Downarrow$$

$$S_i^{-1} \circ B_i \circ F(x) = A_i(x)$$

4. Check whether the mapping $S_i^{-1} \circ B_i \circ F$ is **linear** and reject the incorrect guesses
5. n^3 operations for each guessing $\Rightarrow$ the complexity becomes $kn^3 2^m$ to find full matrix B.

Consider affine equivalence problem

$$B_i \circ F(x) + b_i = S_i(A_i(x) + a_i)$$

for $m \times n$ linear mappings A_i, B_i and the m-bit constant vectors $a_i, b_i, i = 1, \cdots, k$

For each pair $(a_i, b_i) \in \mathbb{Z}_2^m \times \mathbb{Z}_2^m$, inputs $F(x)$ and $S_i(x+a_i)+b_i$, solve the affine equivalence problem

Total complexity: $O(kn^3 2^{3m})$ < $O(n^3 2^{2n})$
(by additionally choosing two m-bit constant vertors)

The dominant parts of the complexities depend on m.

More efficient whenever S ***is a concatenation of several*** S***-boxes*** as in the white-box implementaton.

Without the oracle of the inverse of F

When the oracle of inversion of F is not given, ***use only the property in the equation (3)***

guess about $\log m_A$ vectors, instead of one vector

obtain m_A linearly independent vectors

$$\Downarrow$$

$$O(kn^3 2^{m(\log m_A + 2)}) = O(kn^{m+3} 2^{2m})$$

On the other hand, we can use the relation (4) if we evaluate the requied inverse value of F in the equation (4)

When A is split

consider $A \in (\mathbb{Z}_2)^{n \times n}$ as a $\widetilde{A} \in (\mathbb{Z}_2^{m \times m})^{k \times k}$ with $n = km$

If $\widetilde{A}$ is of form $\begin{bmatrix} * & 0 & * \\ 0 & A^* & 0 \\ * & 0 & * \end{bmatrix}$ for some $A^* \in (\mathbb{Z}_2^{m \times m})^{k_0 \times k_0}$ and $k_0 \geq 1$, we say that A is ***split***, and ***unsplit*** otherwise.

A is split

$$\Downarrow$$

recover the encoding that corresponds to A^* with complexity $k_0(k_0 m)^3 2^{3m}$

Introduction of the Generic Algorithm [2]

For solving the affine equivalence problem in the case where the inner **non-linear layer** is composed of **parallel S-boxes**.

Our algorithm solves the following problem:

Problem 1

Let F be an n-bit to n-bit permutation such that $F = B \circ S \circ A$, where:

1. A and B are n-bit affine layers;
2. $S = (S_1, \cdots, S_k)$ consists of the parallel application of k permutations S_i on m bits each (called S-boxes). Note that $n = km$.

Knowing S, and given oracle access to F (but not F^{-1}), find affine A', B' such that $F = B' \circ S \circ A'$.

Remark 1

Our statement of the problem allows the algorithm to query F, but not F^{-1}.

access to F, but not to F^{-1}

CEJO framwork
The output of F is computed as a sum of sum hard-coded table outputs, and inverting F would require knowing how to split a given output of F into the appropriate sum.

Baek et al. also propose an algorithm when only F is accessible, but it is much slower.

access to both F and F^{-1}

Our own algorithm:

1. **isolate the input and output space of each S-box**
2. **exhaust that space in 2^m operations for each S-box**, which will allow us to access the inverse mapping of each S-box.

Essentially, our algorithm will allow us to revert back to the case where the direct and inverse mappings are both available.

In particular, it is not obvious how our algorithm could be improved even if F^{-1} were accessible.

Baek et al. explicitly provide an algorithm to solve Problem 1 when F and F^{-1} are both available, in $O(n^4 2^{3m}/m)$ operations.
However this is slower than our algorithm for all reasonable parameter ranges, even though our algorithm does not require access to F^{-1}.

Remark 2

Problem 1 asks to recover **some** affine encodings A', B' such that $F = B' \circ S \circ A'$, but not necessarily A and B.

Reason: A and B may not be uniquely defined.

$(A, B) \to (P \circ A, B \circ P^{-1})$, where P is any permutation swapping S-box inputs.

Problem 1 merely asks to recover **a solution**.

Use the algorithm by Dinur to solve the Problem 1, that algorothm is able enumerate all solutions if desired, it is straightforward to adapt our algorithm so that it outputs **every solution**.

Remark 3

The special case: encodings are linear instead of affine

Our algorithm eventually reduced Problem 1 to the `affine equivalence` problem **for each S-box separately**.

Using a `linear equivalence algorithm` on each S-box, instead of an affine one.

Remark 4

The special case: $k = 1$, *i.e.* S is composed of a single S-box.

Practical complexity for n upwards of 128 bits, by using the fact that S is split into relatively small m-bit S-boxes.

Overview of the Algorithm

1. isolate the input and output subspaces of each S-box
2. apply the generic affine equivalence algorithm by Dinur to each S-box separately

The first step: find the input subspace of each S-box
a subspace of dimension m of the input space
this subspace span all 2^m possible values at the input of a single fixed S-box
yields a constant value at the input of all other S-boxes.

Symbol	Notation
Δ	an input difference that uniformly picked at random, yields a zero difference at the input of a particular S-box
V_i	input space which active at most $k-1$ S-boxes
U_i	output space
I_i	input space which active **only one** of the S-boxes
O_i	output space

Computing the V_i's

Computing all input spaces V_i which active at most $k-1$ S-boxes.

$$\Delta \in V_i$$

The input of the i-th S-box:

$$A(x) \oplus A(x \oplus \Delta) = 0$$

and **non-zero differences** at the output of all other $k-1$ S-boxes.

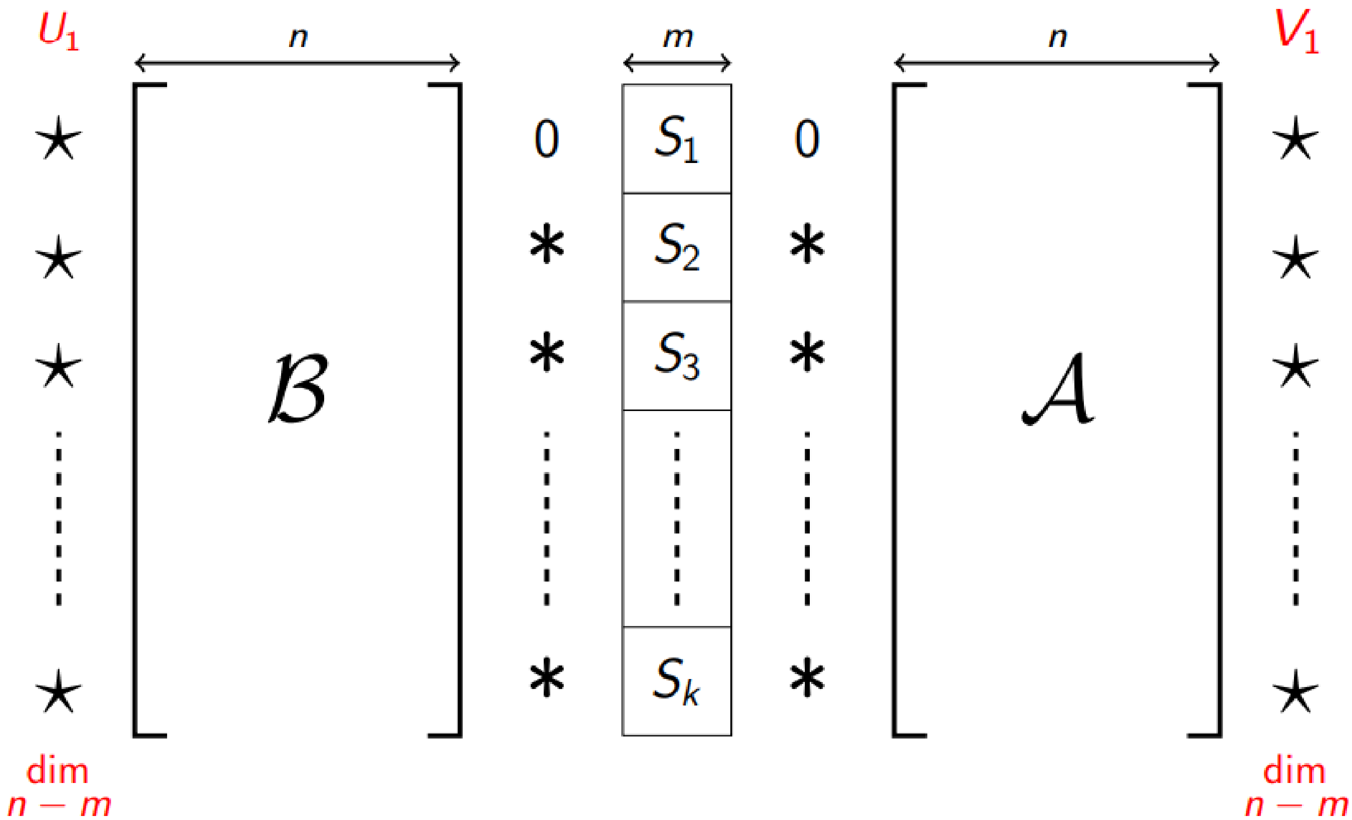
$$\Downarrow$$

one S-box will be inactive

$$\Downarrow$$

$$dim(U_i) = n - m$$

Finding input subspace of each S-box



Testing if $\Delta \in V_1$:

$$X = \{x_i \in \mathbb{F}_n^2, x_i \ random\}$$

$$U = \{F(x_i) \oplus F(x_i \oplus \Delta), x_i \in X\}$$

If $\lim(Span(U_i)) = n - m$, then $\Delta \in V_i$.

$$\Downarrow$$

build all $V_1, \cdots, V_k$

Building V_1

Testing if $\Delta \in V_1$:

- $X = \{x_i \in \mathbb{F}_2^n, x_i \text{ random}\}$ "big enough"
- $U = \{F(x_i) \oplus F(x_i \oplus \Delta), x_i \in X\}$ (output difference space)
- If $\dim(\mathrm{Span}(U)) = n - m$, then $\Delta \in V_1$ w.h.p.

Build a basis of V_1 by doing the same test on independent vectors, and by testing if the resulting output difference space is the same.

Do this k times to build all $V_1, \ldots, V_k$.

Computing the I_i's

Find all input difference spaces I_i which **active only one of the S-boxes**.

for i from 1 to k:

$$\Delta \in V_i, x \in \mathbb{F}_2^n$$

Except on m consecutive bits corresponding to the input of the i-th S-box:

$$A(x) \oplus A(x \oplus \Delta) = 0$$

$$\Downarrow$$

computer the intersection of $k - 1$ spaces V_i

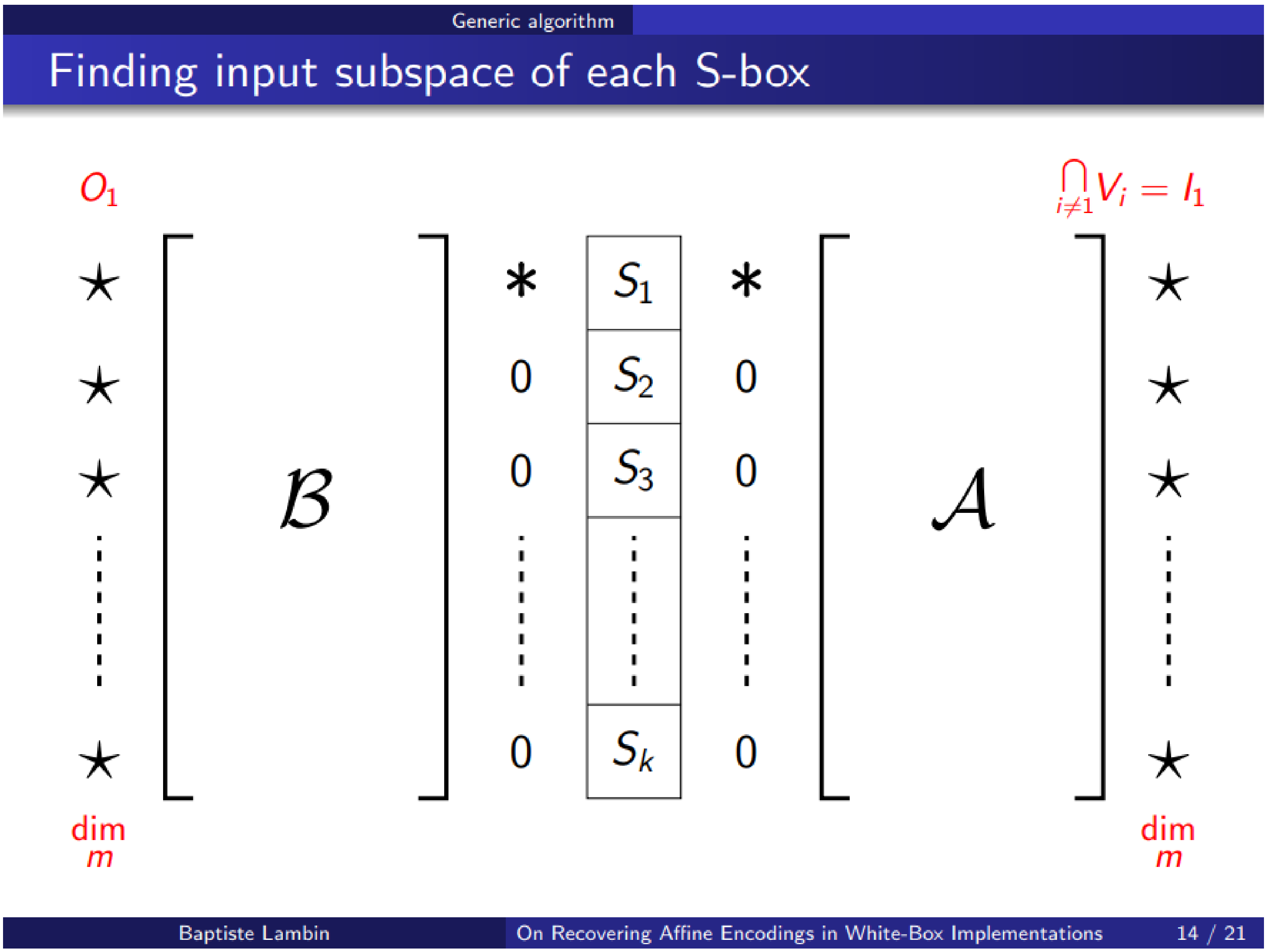
$$\cap V_i = I_i$$

$$\Downarrow$$

recover all the input spaces I_i

$$dim(I_i) = m$$

$$dim(O_i) = m$$



Recovering affine layers

$$F = B \circ P^{-1} \circ (S, \cdots, S) \circ P \circ A$$

P is a **permutation** over the consecutive blocks of m bits.

Build block diagonal affine mapping with block size m: $\mathcal{D}_A, \mathcal{D}_B$

$$\mathcal{D}_A = A \circ \mathcal{P} = diag(A_1, \cdots, A_k)$$

$$\mathcal{D}_B = \mathcal{Q} \circ B = diag(B_1, \cdots, B_k)$$

Build two affine mappings $\mathcal{P}$ and $\mathcal{Q}$

$$\mathcal{P} = \left(\mathcal{P}_1 | \cdots | \mathcal{P}_k\right)$$

$\mathcal{P}_i$: from $\mathbb{F}_2^m$ to I_i

$$\mathcal{Q} = \begin{pmatrix} \mathcal{Q}_1 \\ \hline \vdots \\ \hline \mathcal{Q}_k \end{pmatrix}$$

$\mathcal{Q}_i$: from O_i to $\mathbb{F}_2^m$

$$\Downarrow$$

$$A' = \mathcal{D}_A \circ \mathcal{P}^{-1}$$

$$B' = \mathcal{Q}^{-1} \circ \mathcal{D}_B$$

$$\Downarrow$$

$$F = B' \circ (S, \cdots, S) \circ A'$$

the case of different S-boxes

$$\mathbb{F}_2^m \xleftarrow{\mathcal{Q}_i} O_i \quad \xleftarrow{B \circ \begin{bmatrix} S_1 \\ \vdots \\ S_k \end{bmatrix} \circ A} \quad I_i \xleftarrow{\mathcal{P}_i} \mathbb{F}_2^m$$

$$F = B \circ (S_1, \cdots, S_k) \circ A$$

$$F_i = \mathcal{Q}_i \circ F \circ \mathcal{P}_i$$

$$F_i = B_i \circ S_i \circ A_i$$

$$\Downarrow$$

$$F = B' \circ (S_1, \cdots, S_k) \circ A'$$

Recovering affine layers

$$
\begin{tikzcd}
\mathbb{F}_2^m & O_i \arrow[l, "\mathcal{Q}_i"'] & & & I_i \arrow[llll, "\mathcal{B} \circ \begin{bmatrix} S_1 \\ \vdots \\ S_k \end{bmatrix} \circ \mathcal{A}"'] & \mathbb{F}_2^m \arrow[l, "\mathcal{P}_i"'] \\
& \begin{array}{c}\text{dim}\\ m\end{array} & & & \begin{array}{c}\text{dim}\\ m\end{array} &
\end{tikzcd}
$$

- Apply the Affine Equivalence Algorithm on each $F_i = \mathcal{Q}_i \circ F \circ \mathcal{P}_i$
- Lead to 2 affine mappings $\mathcal{A}_i, \mathcal{B}_i$ such that $F_i = \mathcal{B}_i \circ S_i \circ \mathcal{A}_i$
- Build $\mathcal{A}'$ from all $\mathcal{A}_i$'s and $\mathcal{P}_i$'s, $\mathcal{B}'$ from all $\mathcal{B}_i$'s and $\mathcal{Q}_i$'s such that $\mathcal{B}' \circ (S_1, \ldots, S_k) \circ \mathcal{A}' = F$

We can now inverse F easily as $F^{-1} = \mathcal{A}'^{-1} \circ \left(S_1^{-1}, \ldots, S_k^{-1}\right) \circ \mathcal{B}'^{-1}$!

Algorithm 1 Computing $\widetilde{A}$ and $\widetilde{B}$.

1: **for** $i = 1 \dots k$ **do**
2: $\quad \Delta \leftarrow$ random element in $\mathbb{F}_2^n$
3: $\quad X \leftarrow \{n - m + l$ random elements in $\mathbb{F}_2^n\}$
4: $\quad O_i \leftarrow F(X) \oplus F(X \oplus \Delta)$
5: $\quad$ **if** $(rank(O_i) > n - m)$ OR $(O_i = O_j$ for any $j < i)$ **then**
6: $\quad\quad$ Go back to line 2
7: $\quad$ **else** *With probability* 2^{-m}
8: $\quad\quad V_i = \{\Delta\}$ *V_i will contain a basis of $n - m$ elements*
9: $\quad\quad$ **while** $\#V_i < n - m$ **do**
10: $\quad\quad\quad \Delta \leftarrow$ random element in $\mathbb{F}_2^n$ s.t. $\Delta \notin \mathrm{span}(V_i)$ $\sim 2^m$ *values for* Δ
11: $\quad\quad\quad x \leftarrow$ random element in $\mathbb{F}_2^n$ *l values for x*
12: $\quad\quad\quad$ **if** $F(x) \oplus F(x \oplus \Delta) \in Span(O_i)$ **then** *Using a parity-check matrix of O_i*
13: $\quad\quad\quad\quad V_i = V_i \cup \{\Delta\}$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end while**
16: $\quad$ **end if**
17: **end for**

18: **for each** intersection I_j of $k - 1$ spaces V_i **do** $j = 1 \dots k$
19: $\quad$ Compute a m-bit to n-bit projection $\mathcal{P}_j$ from $\mathbb{F}_2^m$ to I_j
20: $\quad$ Compute a n-bit to m-bit projection $\mathcal{Q}_j$ from O_j to $\mathbb{F}_2^m$
21: $\quad S' \leftarrow \mathcal{Q}_j \circ F \circ \mathcal{P}_j$
22: $\quad$ *S' is a bijection over $\mathbb{F}_2^m$ which is affine equivalent to S*
23: $\quad$ Use the affine equivalence algorithm from Dinur to recover two affine mappings $\mathcal{A}_j, \mathcal{B}_j$ of size m such that $S' = \mathcal{B}_j \circ S \circ \mathcal{A}_j$
24: **end for**

25: $\mathcal{D}_{\mathcal{A}} \leftarrow diag(\mathcal{A}_1, \dots, \mathcal{A}_k)$ *Block diagonal affine mapping with block size m*
26: $\mathcal{D}_{\mathcal{B}} \leftarrow diag(\mathcal{B}_1, \dots, \mathcal{B}_k)$ *Block diagonal affine mapping with block size m*
27: $\mathcal{P} \leftarrow (\mathcal{P}_1 | \dots | \mathcal{P}_k)$ $\cancel{\mathcal{B}' = \mathcal{Q} \circ \mathcal{B}}$
28: $\mathcal{Q} \leftarrow \begin{pmatrix} \mathcal{Q}_1 \\ \hline \vdots \\ \hline \mathcal{Q}_k \end{pmatrix}$ $\cancel{\mathcal{A}' = \mathcal{A} \circ \mathcal{P}}$
29: $\mathcal{A}' \leftarrow \mathcal{D}_{\mathcal{A}} \circ P^{-1}$ and $\mathcal{B}' \leftarrow Q^{-1} \circ \mathcal{D}_{\mathcal{B}}$ *That way, we have $F = \mathcal{B}' \circ (S, \dots, S) \circ \mathcal{A}'$*

Complexity of the algorithm

1. compute all vector spaces V_i

(1) compute the output space O_i

- check whether $\Delta \in \cup_{j=1}^{k} V_j$
- $k2^{-m}$ (2^m values for Δ on average to determine all the k output spaces)

(2) compute the rank of O_i
take $n-m+l$ elements in X

Δ activates all S-boxes:

$$rank(O_i) = n - m$$

$$\Downarrow$$

$$(n-m+l)^2 n = O(n^3)$$

$$\Downarrow$$

the computation of the output spaces $O_1, O_2, \cdots, O_k$ has complexity

$$O(2^m n^3)$$

2. compute a basis of the input space V_i which is of dimension $n-m$

- 2^m tries for Δ
- each value of Δ will be tested using l values of x
- the parity-check matrix of O_i can be computed at size $m \times n$

$$\Downarrow$$

check if one output difference belongs to O_i costs about $O(mn)$ operations

$$\Downarrow$$

$$n = km$$

$$\Downarrow$$

$$O(k(n-m)2^m lmn) = O(2^m klmn^2) = O(2^m ln^3)$$

3. the total complexity of our algorithm

computer all intersection of $k-1$ vevtor spaces V_i can be done in $O(kn^3)$

make k calls to the affine equivalence allgotirhm, which leads to a complexity of $O(km^3 2^m)$

$$\Downarrow$$

$$O(2^m n^3 + 2^m l n^3 + \frac{n^4}{m} + 2^m m^2 n).$$

$$\Downarrow$$

the algorithm from Dinur fails, use the algorithm from Biryukov *et al*.

$$\Downarrow$$

$$O(2^m n^3 + 2^m l n^3 + \frac{n^4}{m} + 2^{2m} m^2 n).$$

Reference

[1] Baek C H, Cheon J H, Hong H. White-box AES implementation revisited[J]. Journal of Communications and Networks, 2016, 18(3): 273-287.

[2] Derbez P, Fouque P A, Lambin B, et al. On recovering affine encodings in white-box implementations[J]. IACR Transactions on Cryptographic Hardware and Embedded Systems, 2018: 121-149.