The Key Recovery Step in Differential Attacks

Rachelle Heim Boissier

Université Catholique de Louvain

December 5th, 2024

Context: symmetric cryptography

- 'Classical' cryptanalysis families: differential, linear, integral, ...
- New designs must come with arguments of resistance to each family.
- Difficulty to know which attack will be the most efficient.
  → Analysing a primitive is thus time-consuming, error-prone.
- In competitions: many ad-hoc cryptanalysis.
  → Difficult to outline generic criteria.

A direction: Proposing generic and automatic cryptanalytic tools.

Context: differential cryptanalysis

- Introduced by Biham and Shamir in 1990.
- One of the oldest and most famous cryptanalysis families

Yet, some primitives are still broken by differential cryptanalysis today.

- Some aspects of differential cryptanalysis are still not well-understood.
- The key recovery step is one of these aspects.

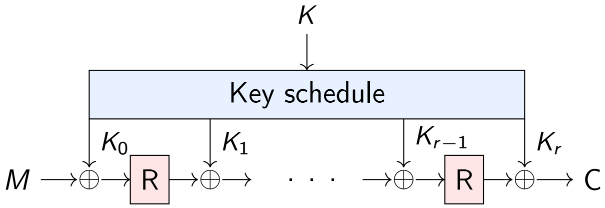
This talk/work: an attempt at providing some clarity.

This talk

- Key recovery attacks against block ciphers
- . . . using differential cryptanalysis
- . . . focusing on the key recovery step.

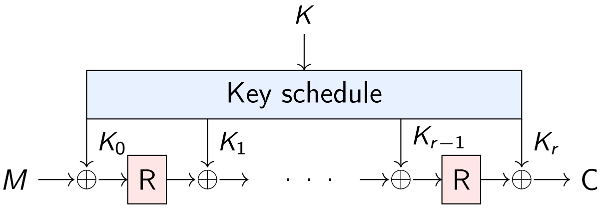
Key recovery attacks against block ciphers

General structure of an iterated block cipher

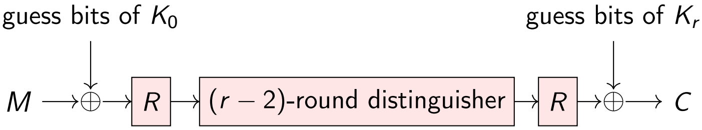


Key recovery attacks against block ciphers

General structure of an iterated block cipher

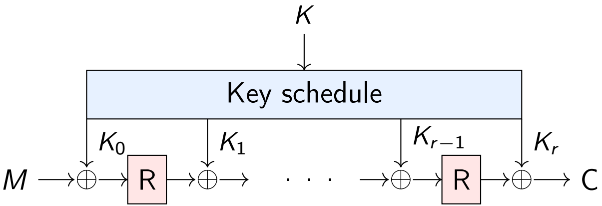


Key recovery attacks

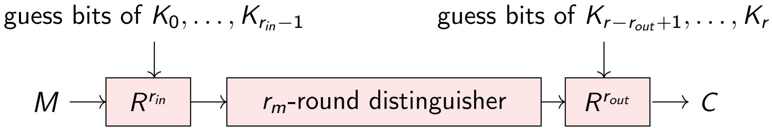


Key recovery attacks against block ciphers

General structure of an iterated block cipher



Key recovery attacks



Outline

1. Differential Cryptanalysis of Block Ciphers
2. Our Model of the Core Key Recovery Step
3. A Generic Algorithm for the Core Key Recovery Step
4. Applications

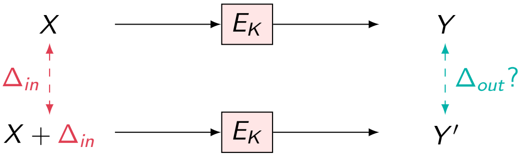
Differential cryptanalysis

For a block cipher E, a differential is a pair of input/output differences $(\Delta_{in}, \Delta_{out})$.

The probability of $(\Delta_{in}, \Delta_{out})$ is the probability p that

$$E_K(X) + E_K(X + \Delta_{in}) = \Delta_{out} \ ,$$

for a key K and an X both chosen uniformly at random.



If $p \gg 2^{-n}$, where n is the block size, then we have a differential distinguisher on E.

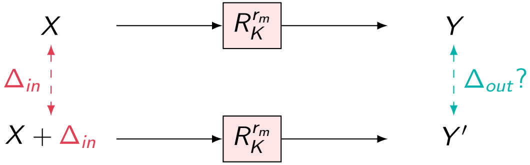
Differential cryptanalysis

For a block cipher E, a differential is a pair of input/output differences $(\Delta_{in}, \Delta_{out})$.

The probability of $(\Delta_{in}, \Delta_{out})$ is the probability p that

$$R_K^{r_m}(X) + R_K^{r_m}(X + \Delta_{in}) = \Delta_{out} \ ,$$

for a key K and an X both chosen uniformly at random.



If $p \gg 2^{-n}$, where n is the block size, then we have a differential distinguisher on R^{r_m}.

Differential key recovery attacks

A differential distinguisher can be used to mount a key recovery attack.

- New primitives should come with arguments of resistance by design against this technique.
- Most of the arguments used rely on showing that differential distinguishers of high probability do not exist after a certain number of rounds.
- Not always enough: A deep understanding of how the key recovery works is necessary to claim resistance against these attacks.

The example of SPEEDY

SPEEDY-7-192 (Leander, Moss, Moradi, Rasoolzadeh, TCHES 21) is a 7-round block cipher.

Designers claim :

- 'The probability of any differential characteristic over **6 rounds** is $\leq 2^{-192}$.
- 'Not possible to add more than one key recovery round to any differential distinguisher.'

Better Steady than Speedy: Full Break of SPEEDY-7-192. Boura, David, Heim Boissier, Naya-Plasencia. **EUROCRYPT 2023**

- Distinguisher over 5.5 rounds ($\rightarrow$ of proba 0 [BN24]).
- Key recovery on 1.5 rounds.
- This work motivated us to work more specifically on the key recovery step.

The example of SPEEDY

SPEEDY-7-192 (Leander, Moss, Moradi, Rasoolzadeh, TCHES 21) is a 7-round block cipher.

Designers claim :

- 'The probability of any differential characteristic over **6 rounds** is $\leq 2^{-192}$.
- 'Not possible to add more than one key recovery round to any differential distinguisher.' **(False)**

Better Steady than Speedy: Full Break of SPEEDY-7-192. Boura, David, Heim Boissier, Naya-Plasencia. **EUROCRYPT 2023**

- Distinguisher over 5.5 rounds ($\rightarrow$ of proba 0 [BN24]).
- Key recovery on 1.5 rounds.
- This work motivated us to work more specifically on the key recovery step.

In previous works

The key recovery step is often done

- either in a 'naive' and non-efficient way;
- or using a tedious and error-prone procedure.

Emergence of new tools for cryptanalysis.

- most tools focus on the search for a differential distinguisher;
- the key recovery step is often considered using heuristics (e.g. [DF16]).

Our contribution: KYRYDI

A Generic Algorithm for Efficient Key Recovery in Differential Attacks - and its Associated Tool.
Boura, David, Derbez, Heim Boissier, Naya-Plasencia. **EUROCRYPT 2024**

Automatic key recovery for SPN block ciphers with

- a bit-permutation as linear layer;
- an (almost) linear key schedule.

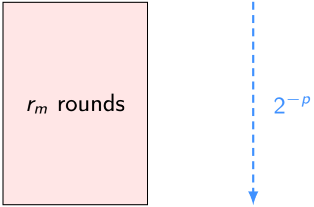
Link to our tool **KYRYDI**:

`https://gitlab.inria.fr/capsule/kyrydi`

Differential key recovery attacks

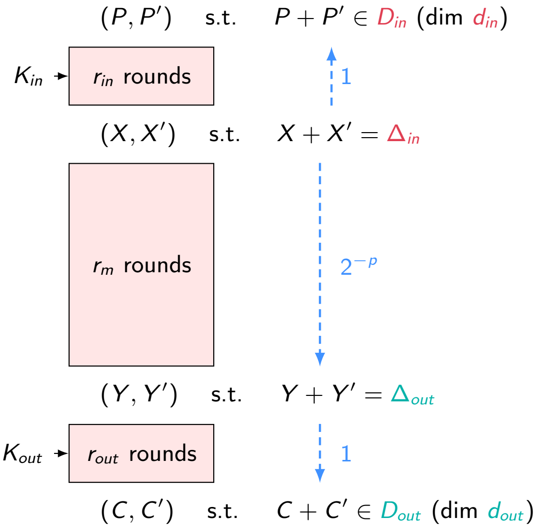
Differential distinguisher

(X, X') s.t. $X + X' = \Delta_{in}$



(Y, Y') s.t. $Y + Y' = \Delta_{out}$

Differential key recovery attacks



Differential key recovery attacks

(P, P') s.t. $P + P' \in D_{in}$ (dim d_{in})

$K_{in} \rightarrow$ r_{in} rounds — 1

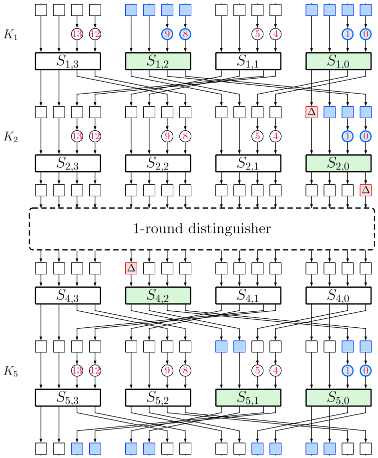
(X, X') s.t. $X + X' = \Delta_{in}$

r_m rounds — 2^{-p}

(Y, Y') s.t. $Y + Y' = \Delta_{out}$

$K_{out} \rightarrow$ r_{out} rounds — 1

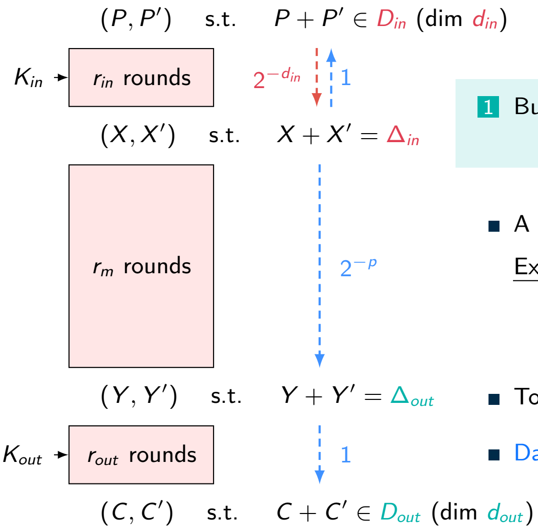
(C, C') s.t. $C + C' \in D_{out}$ (dim d_{out})



Ex: $D_{in} = \{0\}^4 \times \mathbb{F}_2^4 \times \{0\}^4 \times \mathbb{F}_2^4$, $d_{in} = 8$.

$D_{out} = \{0\}^8 \times \mathbb{F}_2^8$ $d_{out} = 8$.

Differential key recovery attacks (1/3)



1 Build enough pairs for at least one to satisfy the differential.

i.e. $2^{p+d_{in}}$ pairs $((P, C), (P', C'))$ s.t. $P + P' \in D_{in}$.

- A structure of size $2^{d_{in}}$ allows to build $2^{2d_{in}}$ pairs.

  Ex: $D_{in} = \{0\}^4 \times \mathbb{F}_2^4 \times \{0\}^4 \times \mathbb{F}_2^4$, $d_{in} = 8$.

  - Structures of the form $\{c_1\} \times \mathbb{F}_2^4 \times \{c_2\} \times \mathbb{F}_2^4$ where $c_1, c_2 \in \mathbb{F}_2^4$.

- To build enough pairs, one needs $2^{p-d_{in}}$ such structures.

- Data complexity: 2^p plaintexts/ciphertext pairs.

Differential key recovery attacks (2/3)

(P, P') s.t. $P + P' \in D_{in}$ (dim d_{in})

K_{in} → r_{in} rounds $2^{-d_{in}}$ ↓ ↑ 1

(X, X') s.t. $X + X' = \Delta_{in}$

r_m rounds 2^{-p}

(Y, Y') s.t. $Y + Y' = \Delta_{out}$

K_{out} → r_{out} rounds 1

(C, C') s.t. $C + C' \in D_{out}$ (dim d_{out})

2 Filter out pairs that cannot follow the differential.

i.e. only retain the fraction $2^{d_{out}-n}$ of pairs s.t. $C + C' \in D_{out}$.

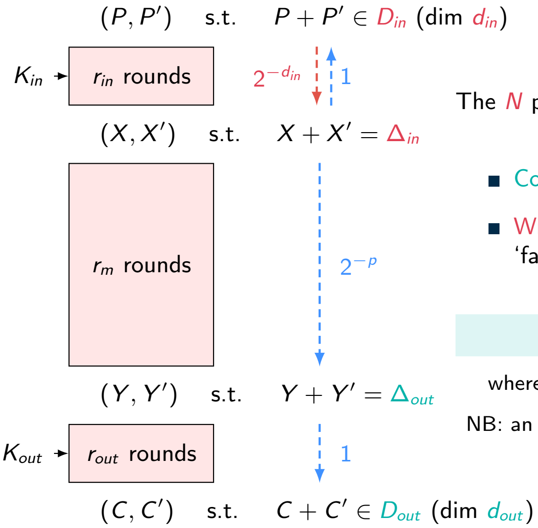
Ex: $D_{out} = \{0\}^8 \times \mathbb{F}_2^8$, $d_{out} = 8 \rightarrow$ filter 2^{-8}.

- Can be done e.g. using hash tables.
- Done for a cost at most 2^p i.e. the data complexity.

Number of pairs to consider in the key recovery step:

$$N = 2^{p+d_{in}+d_{out}-n}.$$

Differential key recovery attacks



The N pairs provide a test for each guess on the involved external key material:

- Correct key guess: one pair satisfies the differential.
- Wrong key guess: on average, $N \cdot 2^{-d_{in}-d_{out}} = 2^{p-n} \ll 1$ 'false alarm(s)'.

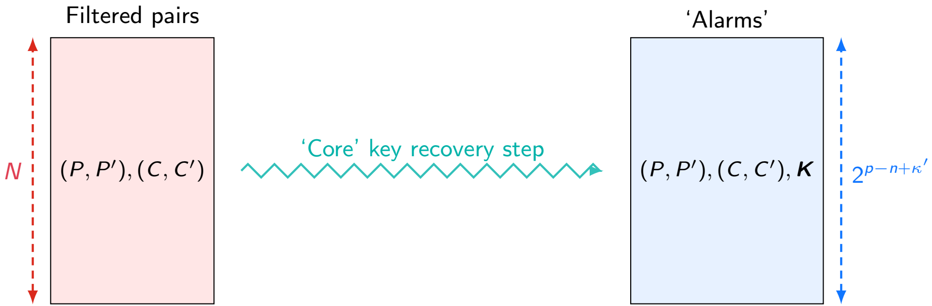
Remaining candidates: $2^{p-n+\kappa'} \ll 2^{\kappa'}$.

where κ' is the number of bits involved in the external key material.

NB: an exhaustive search on the remaining unknown key bits is required.

3. Core key recovery step

Procedure that allows to enumerate the alarms $((P, P'), (C, C'), \boldsymbol{K})$ as efficiently as possible.

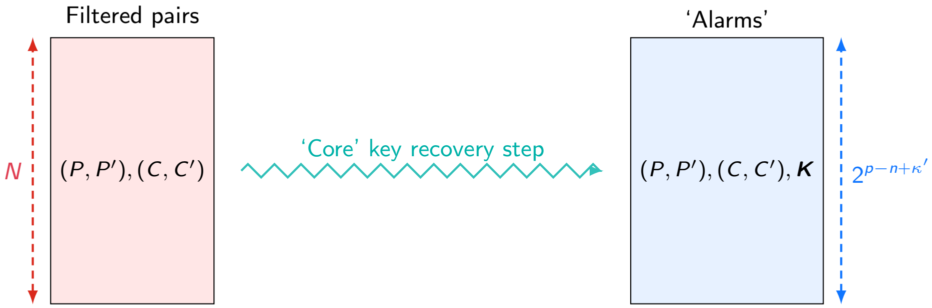


$P + P' \in D_{in}$
$C + C' \in D_{out}$

$\boldsymbol{K} \in \mathbb{F}_2^{\kappa'}$ partially encrypts/decrypts
(P, P') to Δ_{in}, (C, C') to Δ_{out}

3. Core key recovery step

Procedure that allows to enumerate the alarms $((P, P'), (C, C'), \boldsymbol{K})$ as efficiently as possible.



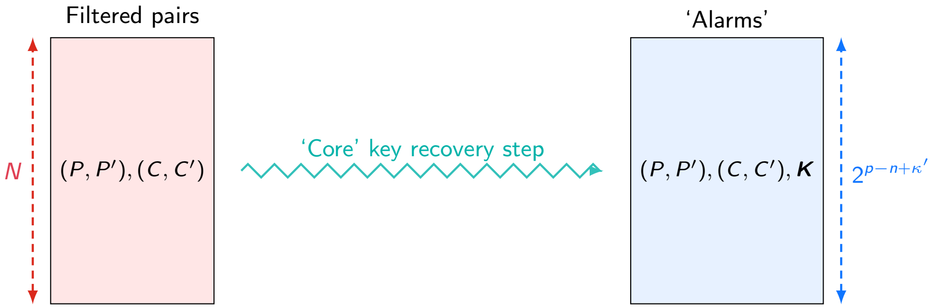
$P + P' \in D_{in}$
$C + C' \in D_{out}$

$\boldsymbol{K} \in \mathbb{F}_2^{\kappa'}$ partially encrypts/decrypts
(P, P') to Δ_{in}, (C, C') to Δ_{out}

What is the complexity of this procedure?

3. Core key recovery step

Procedure that allows to enumerate the alarms $((P, P'), (C, C'), \boldsymbol{K})$ as efficiently as possible.



$P + P' \in D_{in}$
$C + C' \in D_{out}$

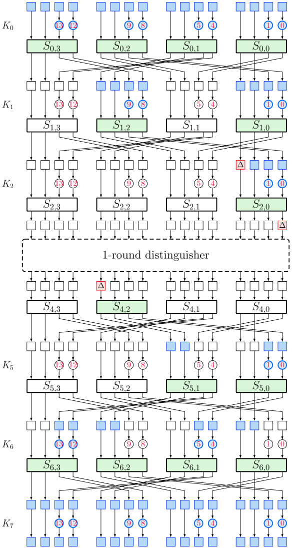
$\boldsymbol{K} \in \mathbb{F}_2^{\kappa'}$ partially encrypts/decrypts (P, P') to Δ_{in}, (C, C') to Δ_{out}

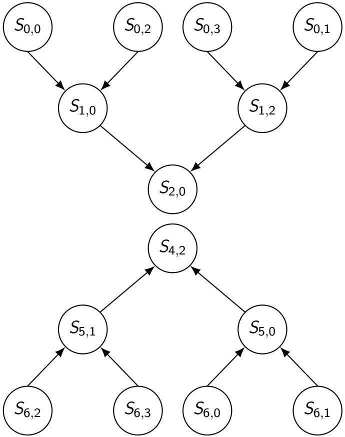
What is the complexity of this procedure?

- Upper bound: $\min(2^{\kappa}, N \cdot 2^{\kappa'})$
- Lower bound: $N + 2^{p-n+\kappa'}$

Outline

The key recovery problem as a graph

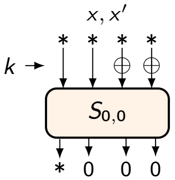




'Solving' an active S-box: For a given pair, finding the guesses on the key material that allow it to respect the differential constraints.

'Solving' S-boxes : the example of $S_{0,0}$

A solution to S is any tuple (x, x', k) s.t. $x + x' \in \nu_{in}$ and $S(x + k) + S(x' + k) \in \nu_{out}$.

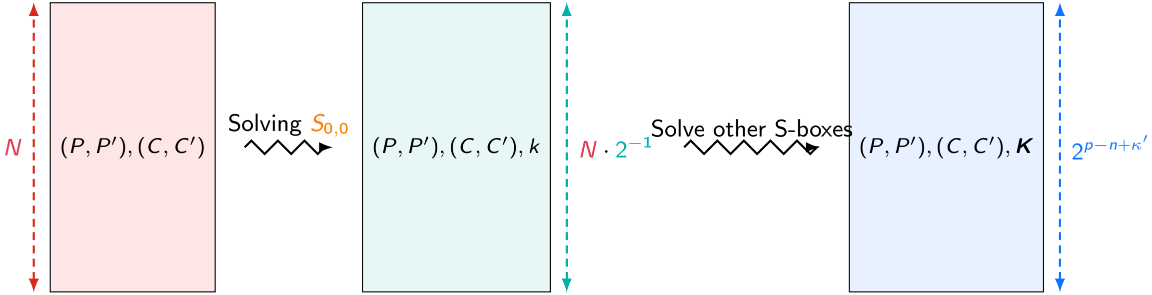


- Number of solutions (x, x', k) to $S_{0,0}$: $2^{4+1+2} = 2^7$.
- $S_{0,0}$ is an S-box of the first round :
  On any of the N pairs, the plaintext pair determines the value of (x, x').
- Probability to match a solution is $c_i = 2^7 \cdot 2^{-8} = 2^{-1}$.

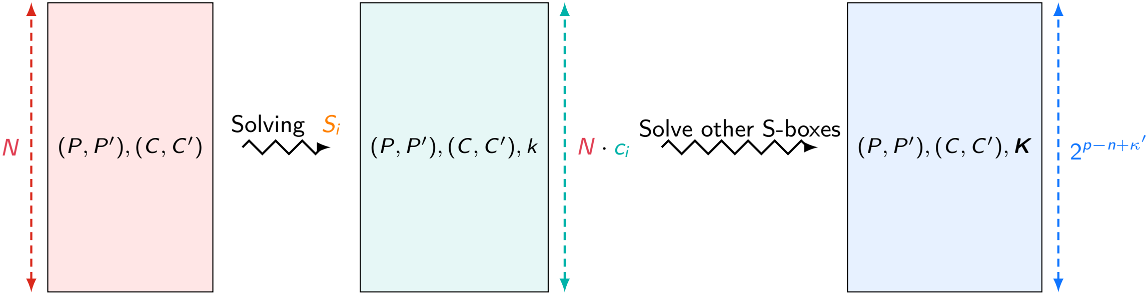
Solving $S_{0,0}$ filters $N \cdot 2^{-1}$ triplets with a determined value on 2 key bits.

Goal: Reduce the number of triplets as early as possible whilst maximizing the number of determined key bits.

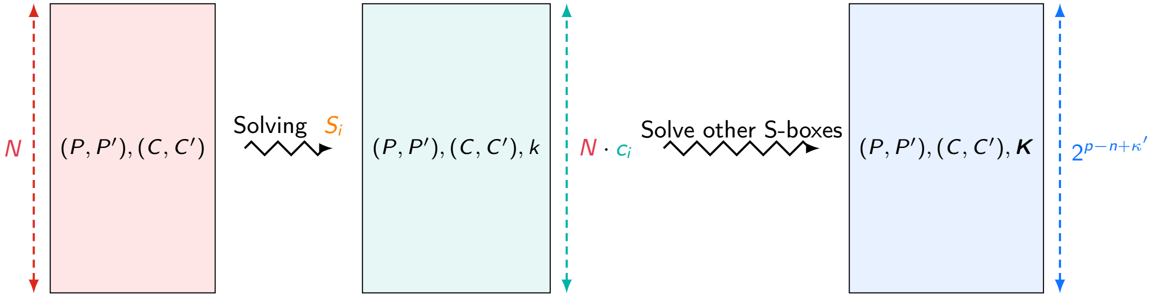
'Solving' S-boxes



'Solving' S-boxes

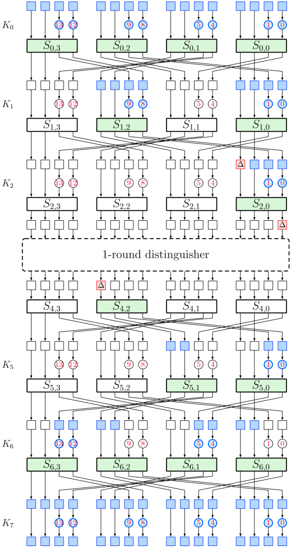


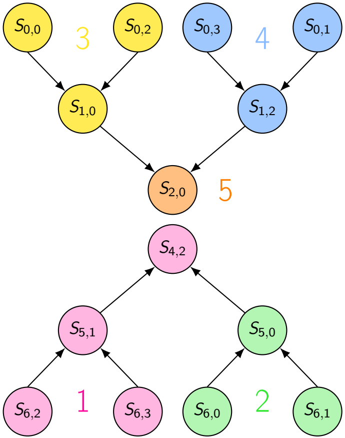
‘Solving’ S-boxes



This can be generalised to any subset of active S-boxes!

The key recovery problem as a graph





Key recovery = partition of the nodes + associated order

Outline

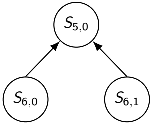
Considering strategies

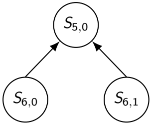
Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$





A strategy can be further refined with extra information: e.g. memory, offline time.

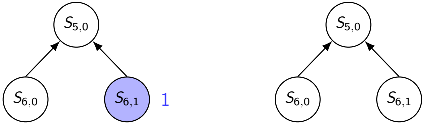
Considering strategies

Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$



A strategy can be further refined with extra information: e.g. memory, offline time.

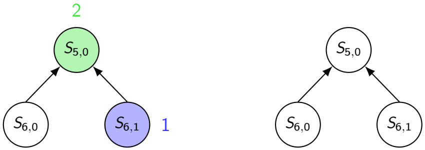
Considering strategies

Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$



A strategy can be further refined with extra information: e.g. memory, offline time.

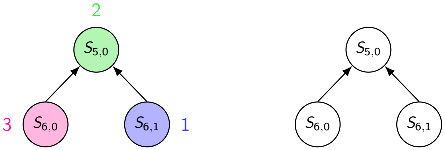
Considering strategies

Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$



A strategy can be further refined with extra information: e.g. memory, offline time.

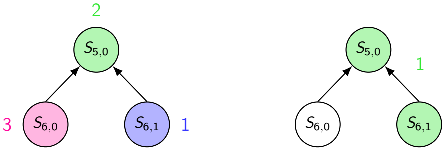
Considering strategies

Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$



A strategy can be further refined with extra information: e.g. memory, offline time.

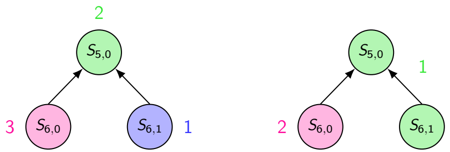
Considering strategies

Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$



A strategy can be further refined with extra information: e.g. memory, offline time.

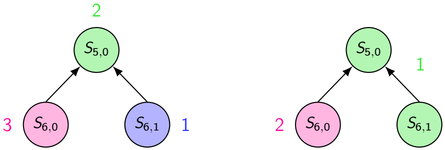
Considering strategies

Strategy $\mathscr{S}_X$ for a subgraph X

Procedure that defines a partition of X and an order in which each subgraph in the partition is solved.

Parameters of a strategy $\mathscr{S}_X$:

- number of solutions s_X
- online time complexity $T(\mathscr{S}_X)$



A strategy can be further refined with extra information: e.g. memory, offline time.

Goal: Build an efficient strategy for the whole graph.

- Based on basic strategies: strategies for a single S-box and an 'initial N pairs' strategy $\mathscr{O}$.

Merging two strategies

Assuming that $s_X < s_Y$, the merge $\mathscr{S}'$ of $\mathscr{S}_X$ and $\mathscr{S}_Y$ is the strategy which consists in

1. running $\mathscr{S}_X$, store the solutions in a hash table;
2. running $\mathscr{S}_Y$, and for each solution, look for matches.

Merging two strategies

Assuming that $s_X < s_Y$, the merge $\mathscr{S}'$ of $\mathscr{S}_X$ and $\mathscr{S}_Y$ is the strategy which consists in

1. running $\mathscr{S}_X$, store the solutions in a hash table;
2. running $\mathscr{S}_Y$, and for each solution, look for matches.

Parameters of $\mathscr{S}'$

- $s_{X \cup Y} = s_X + s_Y - \#$ bit-relations between the nodes of X and Y ⚠ log scale
- $T(\mathscr{S}') \approx \max(T(\mathscr{S}_X), T(\mathscr{S}_Y), s_{X \cup Y})$

Merging two strategies

Assuming that $s_X < s_Y$, the merge $\mathscr{S}'$ of $\mathscr{S}_X$ and $\mathscr{S}_Y$ is the strategy which consists in

1. running $\mathscr{S}_X$, store the solutions in a hash table;
2. running $\mathscr{S}_Y$, and for each solution, look for matches.

Parameters of $\mathscr{S}'$

- $s_{X \cup Y} = s_X + s_Y - \#$ bit-relations between the nodes of X and Y ⚠ log scale
- $T(\mathscr{S}') \approx \max(T(\mathscr{S}_X), T(\mathscr{S}_Y), s_{X \cup Y})$

An optimal strategy for a graph is obtained by merging two optimal strategies for two of its subgraphs.

A dynamic programming approach

'An optimal strategy for a graph is obtained by merging two optimal strategies for two of its subgraphs'

Dynamic programming approach:

- 'Clever' exhaustive search.
- Bottom-up approach: merge strategies with a small time complexity first.
- Keep only the optimal strategy found for each subgraph X.
- Restricting merges thanks to heuristics.

Comparing two strategies

Compare two strategies $\mathscr{S}_X^1$ and $\mathscr{S}_X^2$ for the same subgraph X

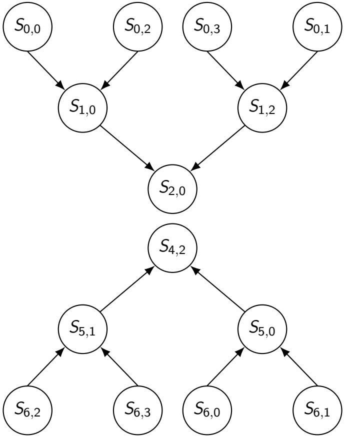
1. Choose the one with the best time complexity.
2. If same time complexity, choose the one with the best memory complexity.

Compare $\mathscr{S}_X^1$ and $\mathscr{S}_Y^2$ when $Y \subset X$

If the number of solutions and time complexity of $\mathscr{S}_X^1$ are not higher than those of $\mathscr{S}_Y^2$, then we can freely replace $\mathscr{S}_Y^2$ by $\mathscr{S}_X^1$.

Restricting merges (1/2)

1. Only allow merges between co-dependent sub-graphs:
   - An edge between two nodes;
   - Or at least a common node between two subgraphs.



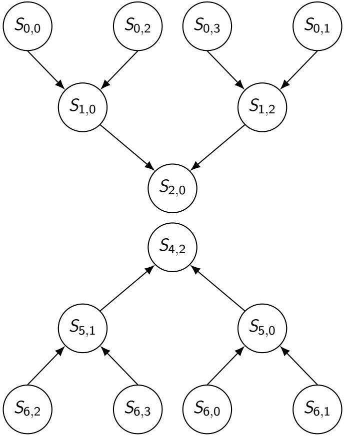
Examples:

- $\mathscr{S} = \{\mathscr{O}, S_{0,0}\}$ cannot be merged with $\mathscr{S}' = \{S_{1,2}\}$.
- $\mathscr{S} = \{\mathscr{O}, S_{6,0}\}$ can be merged with $\mathscr{S}' = \{S_{0,0}\}$.

Restricting merges (2/2)

2 A non-filtering node can be merged with an online strategy iff

- It is computed by partially encrypting/decrypting the data;
- Or it does not increase the number of solutions.



Examples:

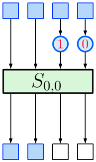
- $\mathscr{S} = \{\mathscr{O}, S_{0,0}, S_{0,2}\}$ can always be merged with $\mathscr{S}' = \{S_{1,0}\}$.
- $\mathscr{S} = \{\mathscr{O}, S_{0,0}\}$ can only be merged with $\mathscr{S}' = \{S_{1,0}\}$ if it does not increase the number of solutions.

Additional improvements (1/2) : Sieving

Idea: Use the differential constraints to filter out pairs that cannot follow the differential, regardless of the value of the key.

- Example: $(x_3, x'_3, x_2, x'_2, x_1 \oplus x'_1, x_0 \oplus x'_0)$

  Filter: $36/2^6 = 2^{-0.83}$.



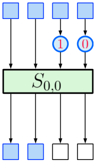
Additional improvements (1/2) : Sieving

Idea: Use the differential constraints to filter out pairs that cannot follow the differential, regardless of the value of the key.

- Example:

$$(\mathtt{x}_3, \mathtt{x}_3', \mathtt{x}_2, \mathtt{x}_2', \mathtt{x}_1 \oplus \mathtt{x}_1', \mathtt{x}_0 \oplus \mathtt{x}_0')$$

Filter: $36/2^6 = 2^{-0.83}$.



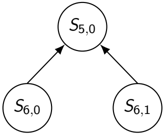
Pre-sieving

Apply a sieve on all S-boxes of the external rounds.

Advantage : The key recovery step is performed on $N' \leq N$ pairs.

Additional improvements (2/2) : Precomputing partial solutions

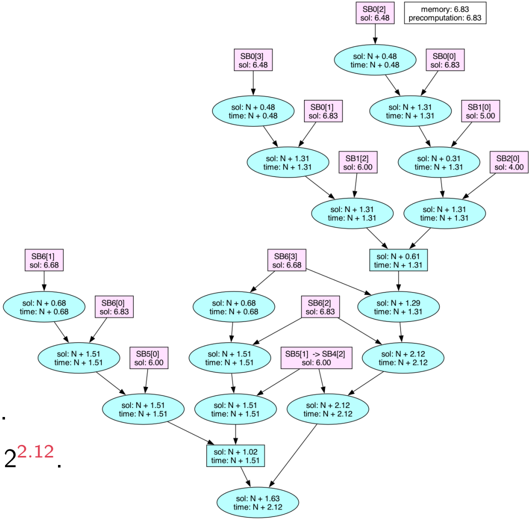
Idea: Precompute the partial solutions to some subgraph.



- Impact on the memory complexity and the offline time of the attack.
- The key recovery strategy found by the tool depends on how much memory and offline time are allowed.

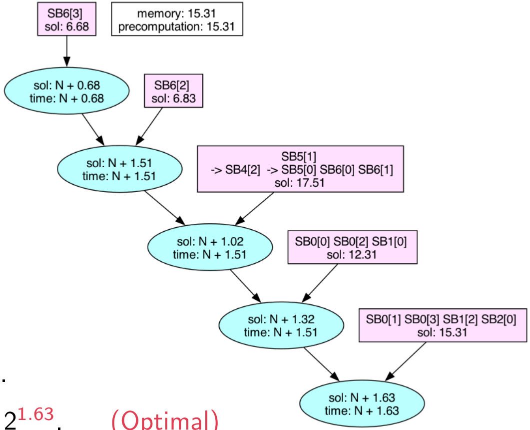
Outline

Application to the toy cipher



- **Nr solutions:** $N \cdot 2^{1.63}$.
- **Time complexity:** $N \cdot 2^{2.12}$.
- **Memory:** $N \cdot 2^{6}$.

Application to the toy cipher



- **Nr solutions:** $N \cdot 2^{1.63}$.
- **Time complexity:** $N \cdot 2^{1.63}$. (Optimal)
- **Memory:** $N \cdot 2^{15}$.

Applications

Start from an existing distinguisher that led to the best key recovery attack against the target cipher.

- `RECTANGLE-128`: Extended by one round the previous best attack.
  - From 18 to 19 rounds out of 25.
- `PRESENT-80`: Extended by two rounds the previous best differential attack.
  - From 16 to 18 rounds out of 31.
- `GIFT-64`: Best key recovery strategy without additional techniques.
  - 26 rounds out of 28.

Future improvements, open questions

- Taking into account key-schedule relations more accurately (including non-linear ones?).
- Incorporate tree-based key recovery techniques [Bro+21].
- Handle ciphers with more complex linear layers.
- Prove optimality.
- Generalise to other attacks.

The best distinguisher does not always lead to the best key recovery!

Ultimate goal

Combine the tool with a distinguisher-search algorithm to find the best possible attacks.

A dynamic programming approach

Simplified algorithm: Initialisation

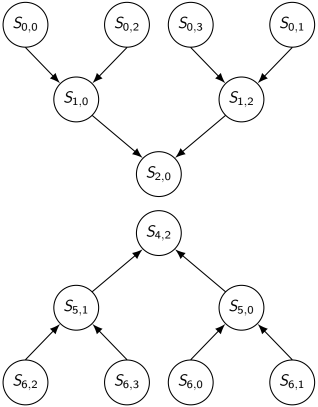
Create two lists:

- $\mathrm{L}_{done} \leftarrow \mathcal{O}$ where $\mathcal{O}$ corresponds to the 'initial N pairs' node.
- $\mathrm{L}_{current} \leftarrow$ basic strategies.

Ex: toy cipher:

$S_{0,0}, S_{0,1}, S_{0,2}, S_{0,3}, S_{1,0}, S_{1,2}, S_{2,0}, S_{6,0}, S_{6,1}, S_{6,2}, S_{6,3}, S_{5,0}, S_{5,1}, S_{4,2}$

NB: The 'online node' $\mathcal{O}$ is linked to all the plaintext/ciphertext nodes.



A dynamic programming approach

Simplified algorithm (2)

While $\mathrm{L}_{current} \neq \emptyset$:

- Let S be the strategy from $\mathrm{L}_{current}$ with the smallest T.
  - For any S' in L_{done} allowed to be merged with S:
    Let S'' be their merge.
    If no strategy from L_{done} nor $\mathrm{L}_{current}$ is better than S'':
    - Add S'' to $\mathrm{L}_{current}$.
    - Remove from both L_{done} and $\mathrm{L}_{current}$ all strategies worst than S''.
- Remove S from $\mathrm{L}_{current}$, add it to L_{done}.