A generic algorithm for efficient key recovery in differential attacks – and its associated tool

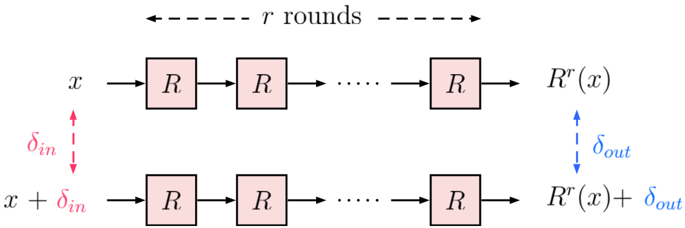
Christina Boura, Nicolas David, Patrick Derbez, Rachelle Heim Boissier, María Naya-Plasencia

UVSQ, Inria, University of Rennes

Eurocrypt 2024, Zurich, Switzerland

Differential cryptanalysis

- Cryptanalysis technique introduced by Biham and Shamir in **1990**.
- Based on the existence of a high-probability **differential** $(\delta_{in}, \delta_{out})$.



- If the probability of $(\delta_{in}, \delta_{out})$ is (much) higher than $\max(2^{-n}, 2^{-\kappa})$, where n is the block size, κ the key length, then we have a differential distinguisher.

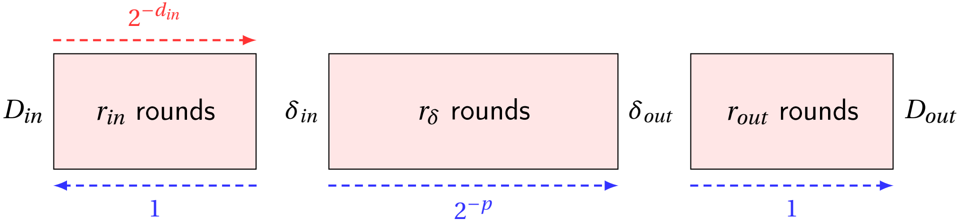
Key recovery attack

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

- This technique broke many block ciphers of the 70s-80s, e.g. `DES`, `FEAL`, etc.
- 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 key recovery problem

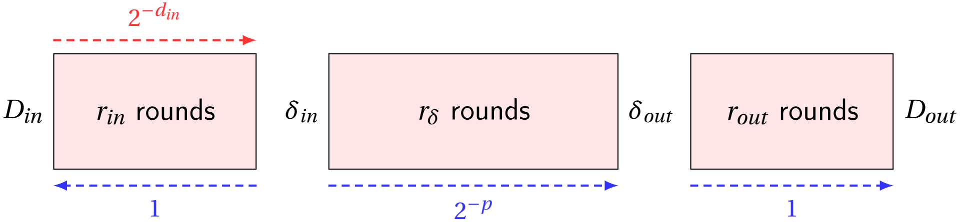
Overview of the key recovery procedure



First step: Construct $2^{p+d_{in}}$ pairs $((P,C),(P',C'))$ s.t. $P+P' \in D_{in}$.

- Use of structures of size $2^{d_{in}}$ → Data complexity: $\approx 2^{p+1}$, Memory complexity: $2^{d_{in}}$

Overview of the key recovery procedure



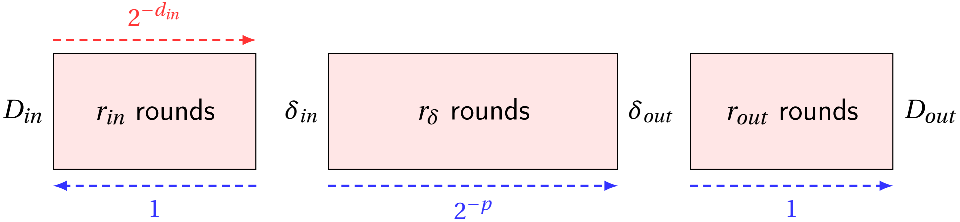
First step: Construct $2^{p+d_{in}}$ pairs $((P,C),(P',C'))$ s.t. $P+P' \in D_{in}$.

- Use of structures of size $2^{d_{in}}$ $\rightarrow$ Data complexity: $\approx 2^{p+1}$, Memory complexity: $2^{d_{in}}$

Second step: Discard pairs that are not in D_{out}.

- Number of pairs for the attack: $N = 2^{p+d_{in}-(n-d_{out})}$

Overview of the key recovery procedure



First step: Construct $2^{p+d_{in}}$ pairs $((P,C),(P',C'))$ s.t. $P+P' \in D_{in}$.

- Use of structures of size $2^{d_{in}} \rightarrow$ Data complexity: $\approx 2^{p+1}$, Memory complexity: $2^{d_{in}}$

Second step: Discard pairs that are not in D_{out}.

- Number of pairs for the attack: $N = 2^{p+d_{in}-(n-d_{out})}$

Third step: Core key recovery

Core key recovery

Goal

Determine the pairs for which there exists an associated key that leads to the differential.

A candidate is a triplet $((P, C), (P', C'), k)$ such that the (partial) key candidate k encrypts (resp. decrypts) (P, P') (resp. (C, C')) to the input (resp. output) of the differential.

Core key recovery

Goal

Determine the pairs for which there exists an associated key that leads to the differential.

A candidate is a triplet $((P,C),(P',C'),k)$ such that the (partial) key candidate k encrypts (resp. decrypts) (P,P') (resp. (C,C')) to the input (resp. output) of the differential.

What is the complexity of this procedure?

- Upper bound: $\min(2^{\kappa}, N \cdot 2^{|\mathcal{K}|})$,
- Lower bound: $N + N \cdot 2^{|\mathcal{K}|-d_{in}-d_{out}}$,
  where $N \cdot 2^{|\mathcal{K}|-d_{in}-d_{out}}$ is the number of expected candidates.

Core key recovery

Goal

Determine the pairs for which there exists an associated key that leads to the differential.

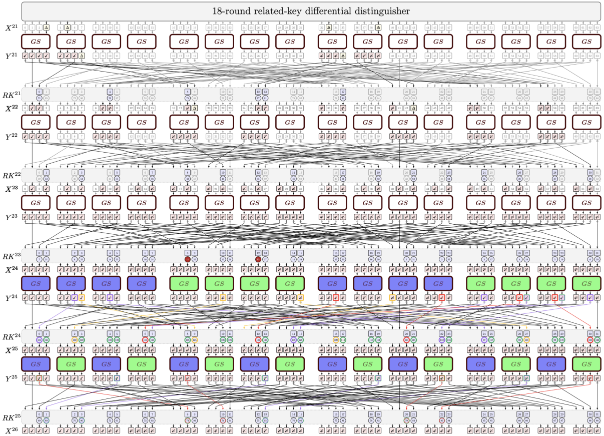
A candidate is a triplet $((P,C),(P',C'),k)$ such that the (partial) key candidate k encrypts (resp. decrypts) (P,P') (resp. (C,C')) to the input (resp. output) of the differential.

What is the complexity of this procedure?

- Upper bound: $\min(2^{\kappa}, N \cdot 2^{|\mathcal{K}|})$,
- Lower bound: $N + N \cdot 2^{|\mathcal{K}|-d_{in}-d_{out}}$,
  where $N \cdot 2^{|\mathcal{K}|-d_{in}-d_{out}}$ is the number of expected candidates.

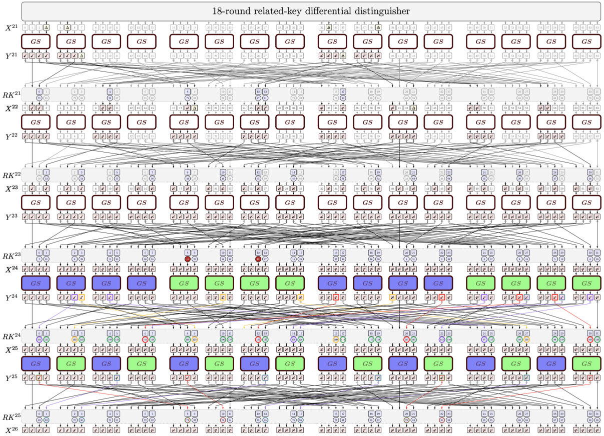
A key recovery is efficient, if its complexity is as close as possible to the lower bound.

The key recovery problem



Potentially too many active S-boxes and key guesses.

The key recovery problem

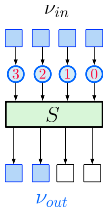


Our goal : Automatise the key recovery for SPN block ciphers with a bit-permutation as linear layer and an (almost) linear key schedule.

Efficient key recovery

Solving an active S-box S

Determine the triplets (x, x', k) s. t. $x + x' \in \nu_{in}$ and $S(x+k) + S(x'+k) \in \nu_{out}$.
Discard the other triplets.

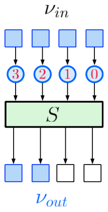


Example: this active S-box has $2^{8+4-2} = 2^{10}$ solutions.

Efficient key recovery

Solving an active S-box S

Determine the triplets (x, x', k) s. t. $x + x' \in \nu_{in}$ and $S(x+k) + S(x'+k) \in \nu_{out}$.
Discard the other triplets.



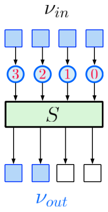
Can be generalised to any subset of active S-boxes!

Example: this active S-box has $2^{8+4-2} = 2^{10}$ solutions.

Efficient key recovery

Solving an active S-box S

Determine the triplets (x, x', k) s. t. $x + x' \in \mathcal{V}_{in}$ and $S(x+k) + S(x'+k) \in \mathcal{V}_{out}$.
Discard the other triplets.



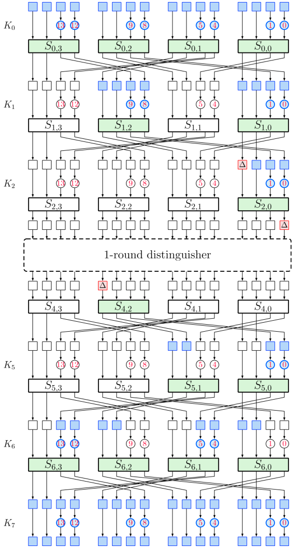
Can be generalised to any subset of active S-boxes!

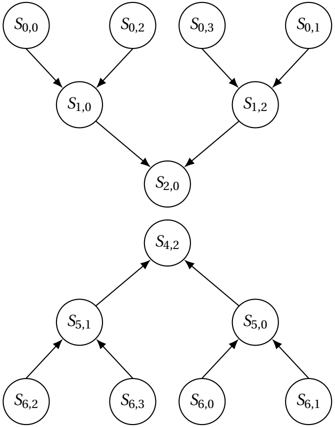
Example: this active S-box has $2^{8+4-2} = 2^{10}$ solutions.

Goal: Reduce the number of triplets as early as possible whilst maximizing the number of determined key bits in the involved key material $\mathcal{K}$.

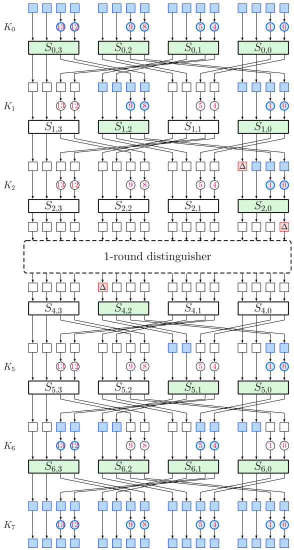
An algorithm for efficient key recovery

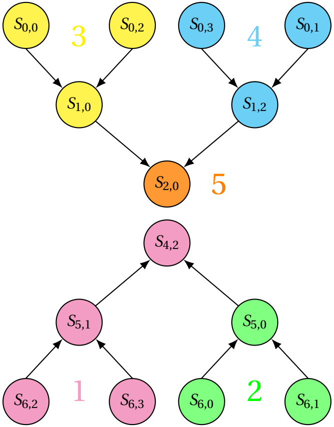
Modeling the key recovery as a graph





Modeling the key recovery as a graph





Key recovery:
partition of the nodes + associated order

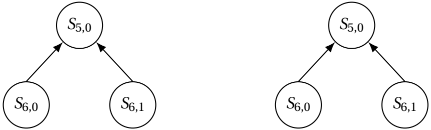
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathscr{N}$;
- online time complexity $\mathscr{T}$.



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

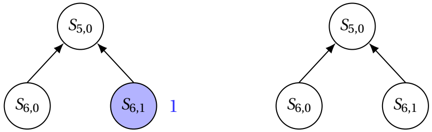
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathscr{N}$;
- online time complexity $\mathscr{T}$.



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

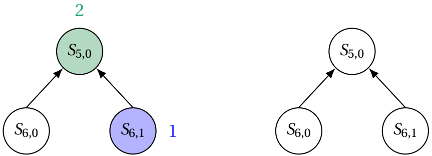
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathscr{N}$;
- online time complexity $\mathscr{T}$.



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

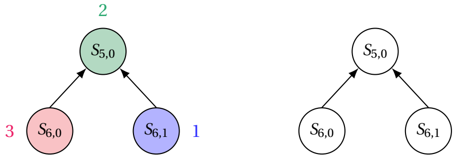
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathscr{N}$;
- online time complexity $\mathscr{T}$.



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

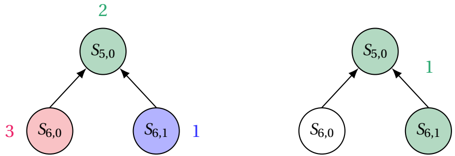
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathscr{N}$;
- online time complexity $\mathscr{T}$.



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

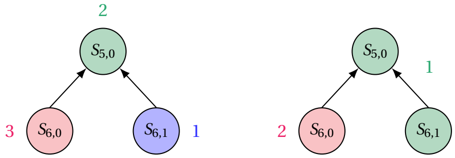
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathcal{N}$;
- online time complexity $\mathcal{T}$.



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

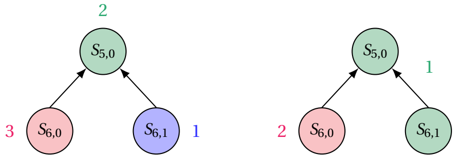
Considering strategies

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

Procedure that allows to enumerate all the possible values that the S-boxes of X can take under the differential constraints imposed by the distinguisher.

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

- number of solutions $\mathscr{N}$;
- online time complexity $\mathscr{T}$.



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

Objective: Build an efficient strategy for the whole graph.

→ Based on basic strategies, i.e. strategies for a single S-box.

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 $\mathscr{S}_X^1$.

Merging two strategies

Let $\mathscr{S}_X$ and $\mathscr{S}_Y$ two strategies for the graphs X and Y respectively.

- The number of solutions of $\mathscr{S}' = merge(\mathscr{S}_X, \mathscr{S}_Y)$ **only depends** on $X \cup Y$:

Number of solutions of $\mathscr{S}'$

$Sol(X \cup Y) = Sol(X) + Sol(Y) - \#$ bit-relations between the nodes of X and Y ⚠ log scale

Time and memory associated to $\mathscr{S}'$

- $T(\mathscr{S}') \approx \max(T(\mathscr{S}_X), T(\mathscr{S}_Y), Sol(X \cup Y))$
- $M(\mathscr{S}') \approx \max(M(\mathscr{S}_X), M(\mathscr{S}_Y), \min(Sol(\mathscr{S}_X), Sol(\mathscr{S}_Y)))$

A dynamic programming approach

- The online time complexity of $merge(\mathscr{S}_X, \mathscr{S}_Y)$ **only depends** on the time complexities of $\mathscr{S}_X$ and $\mathscr{S}_Y$.
- An optimal strategy for $X \cup Y$ **can always** be obtained by merging two optimal strategies for X and Y.
- Use a bottom-up approach, merging first the strategies with the smallest time complexity to reach a graph strategy with a minimal time complexity.

Dynamic programming approach

Ensure that, for any subgraph X, we only keep one optimal strategy to enumerate it.

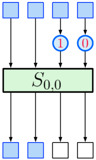
Sieving

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



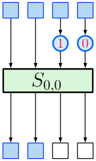
Sieving

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



Pre-sieving

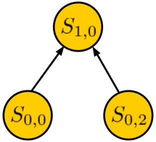
Apply a sieve on all S-boxes of the external rounds.

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

Precomputing partial solutions

Idea

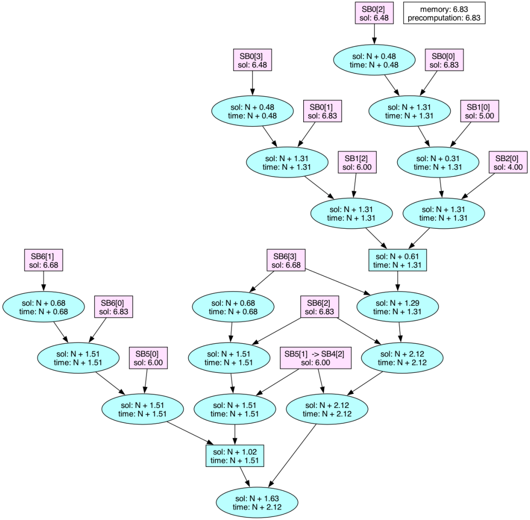
Precompute the partial solutions to some subgraph.



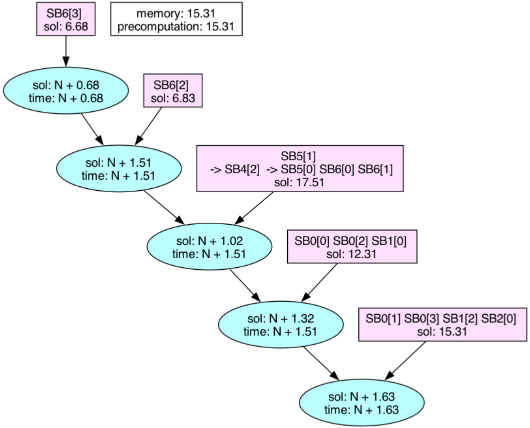
- Impact on the memory complexity and the offline time of the attack.
- The optimal key recovery strategy depends on how much memory and offline time are allowed.

Applications of our tool: KYRYDI

Application to the toy cipher



Application to the toy cipher



Application to other ciphers

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

- RECTANGLE: Extended by one round the previous best attack.
- PRESENT-80: Extended by two rounds the previous best differential attack.
- GIFT-64 and SPEEDY-7-192: Best key recovery strategy without additional techniques.

Extensions and improvements

- Handle ciphers with more complex linear layers.
- Handle ciphers with non-linear key schedules.
- Incorporate tree-based key recovery techniques by exploiting the structure of the involved S-boxes.

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.

Other open problems

- Prove optimality.
- The tool works for (impossible) differential attacks:

  → Apply a similar approach to other attacks.

Other open problems

- Prove optimality.
- The tool works for (impossible) differential attacks:

  → Apply a similar approach to other attacks.

Thanks for your attention!

Link to **KYRYDI**:

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