TAGADA

Tool for Automatically Generation of Abstraction-Based Differential Attacks

SKCAM 25' | March 15, 2025

Loïc Rouquette

EPITA Research Laboratory (LRE), Lyon, France

Outline

Why Tagada ?

What is TAGADA ?

How TAGADA works?

Our results

Further work

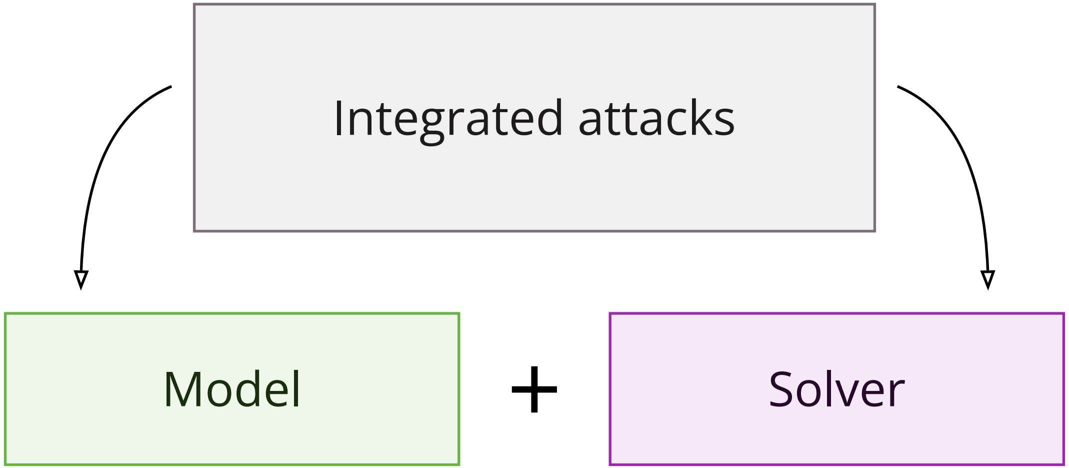
Bibliography

Why Tagada ?

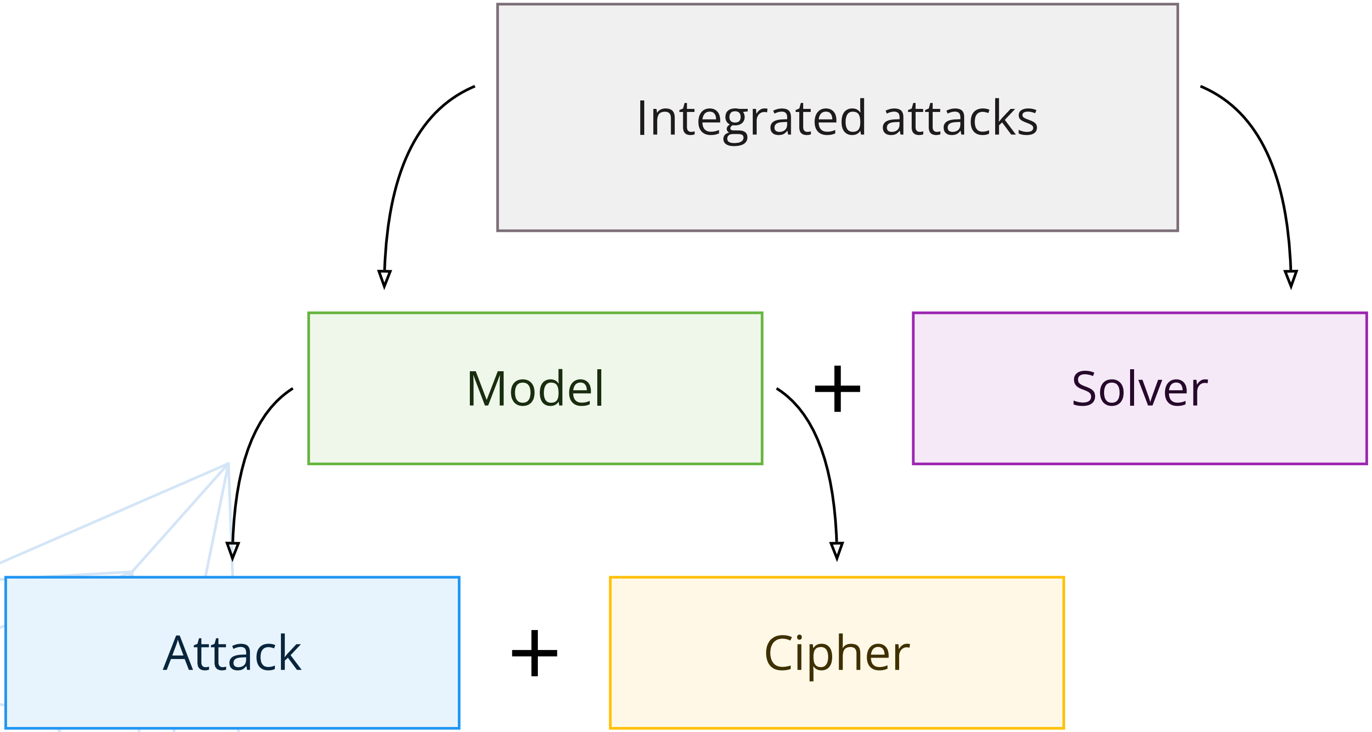
The idea

Integrated attacks

The idea



The idea



Why ?

How block ciphers are created ?

Basic known primitives

- $\oplus$, $\otimes$, $\boxplus$, $\boxminus$, $\lll$, $\ggg$, SBoxes, etc.

Common structures

- SPN (Substitution Permutation Network)
- Feistel Networks
- ARX (Add Rotate Xor)

AES vs Midori

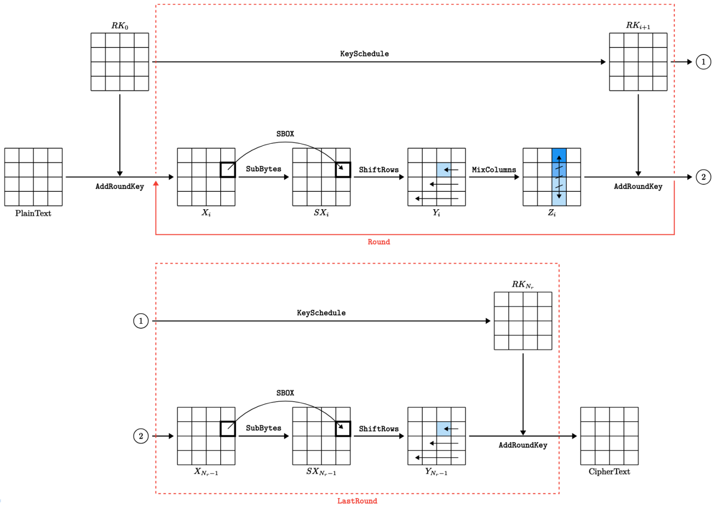


Figure 1: Representation of AES [1] cipher (from [2]).

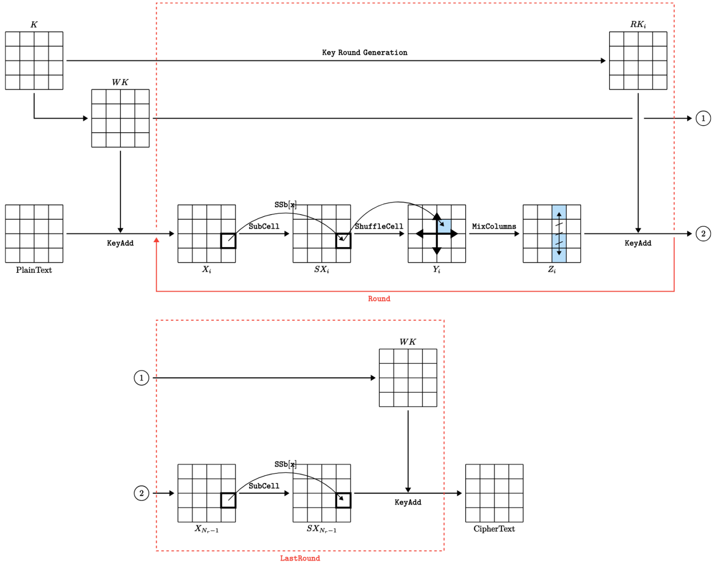


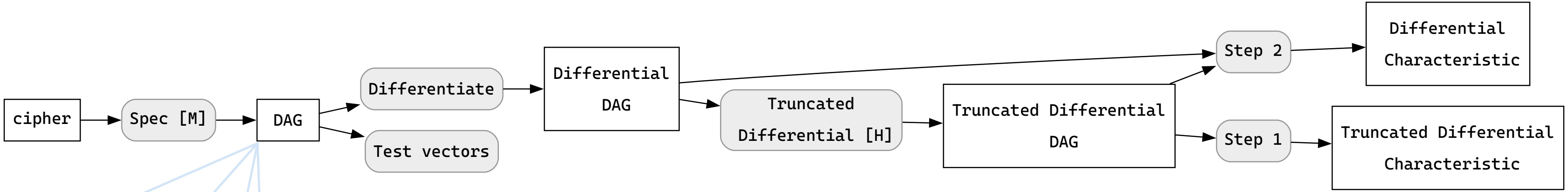
Figure 2: Representation of Midori [3] encryption (from [2]).

The expected benefit

Some vulnerabilities discovered in one cipher can be exploited in other ciphers that reuse the same construction.

What is TAGADA ?

Tool for Automatic Generation of Abstraction-Based Differential Attacks

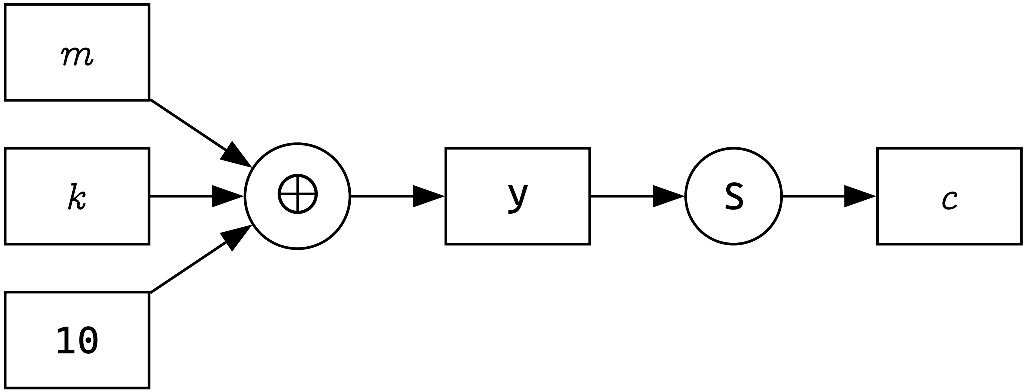


Data Representation

A cipher function E_K is represented as a bipartite DAG (Directed Acyclic Graph) $G = (N, E)$ where N is the states and operations of the cipher and E is the set of relations that links the states and the operations.

Example

$$c = S[m \oplus k \oplus 10]$$



$$G = (N, E):$$
$$N = \{m, k, 10, y, z, \oplus, S\},$$
$$E = \{((m, k, 10), \oplus, (y)), ((y), S, (c))\}$$

Specification (manual)

Objective

Transforms the source cipher into a DAG builder.

What is TAGADA ?

```ruby
def encryption(x, mk, nr)
  k = mk.each_slice(16).to_a
  @builder.named("K_0", k[0])
  @builder.named("K_1", k[1])
  for r in 1 ... nr
    for i in 0 ... BR_HALF
      x[2 * i + 1] = @xor3.call(
        @s.call(x[2 * i]),
        k[(r - 1) % 2][i],
        x[2 * i + 1]
      )
    end
    x[1] = @xor2.call(x[1], RC0[r])
    x[3] = @xor2.call(x[3], RC1[r])
    permutation(x)
    @builder.named("X_#{r}", x)
  end
  for i in 0 ... BR_HALF
    x[2 * i + 1] = @xor3.call(
      @s.call(x[2 * i]),
      k[(nr - 1) % 2][i],
      x[2 * i + 1]
    )
  end
  x[1] = @xor2.call(x[1], RC0[nr])
  x[3] = @xor2.call(x[3], RC1[nr])
  x
end
```

Listing 1: TAGADA WARP builder.

Algorithm Encryption(K, M)

1. $(K_0^0 \| K_1^0 \| \ldots \| K_{15}^0, K_0^1 \| K_1^1 \| \ldots \| K_{15}^1) \leftarrow K$
2. $X_0 \| X_1 \| \ldots \| X_{31} \leftarrow M$
3. **for** $r = 1$ **to** 40 **do**
4.   **for** $i = 0$ **to** 15 **do**
5. $X_{2i+1} \leftarrow S(X_{2i}) \oplus K_i^{(r-1) \bmod 2} \oplus X_{2i+1}$
6.   **end for**
7. $X_1 \leftarrow X_1 \oplus RC_0^r, X_3 \leftarrow X_3 \oplus RC_1^r$
8. $X'_0 \| X'_1 \| \ldots \| X'_{31} \leftarrow X_0 \| X_1 \| \ldots \| X_{31}$
9.   **for** $i = 0$ **to** 31 **do**
10. $X_{\pi[j]} \leftarrow X'_j$
11.   **end for**
12. **end for**
13. **for** $i = 0$ **to** 15 **do**
14. $X_{2i+1} \leftarrow S(X_{2i}) \oplus K_i^0 \oplus X_{2i+1}$
15. **end for**
16. $X_1 \leftarrow X_1 \oplus RC_0^{41}, X_3 \leftarrow X_3 \oplus RC_1^{41}$
17. $C \leftarrow X_0 \| X_1 \| \ldots \| X_{31}$
18. **return** C

Figure 4: Encryption algorithm of WARP [4].

Command

```
# Generation of the full AES-128
docker run -t tagada \
  bundle exec specs/ciphers/rijndael.rb -p 128 -k 128 > aes.spec.json
```

```
# Generation of 3 rounds of WARP
docker run -t tagada \
  bundle exec specs/ciphers/warp.rb -r 3 > warp-r3.spec.json
```

What is TAGADA ?

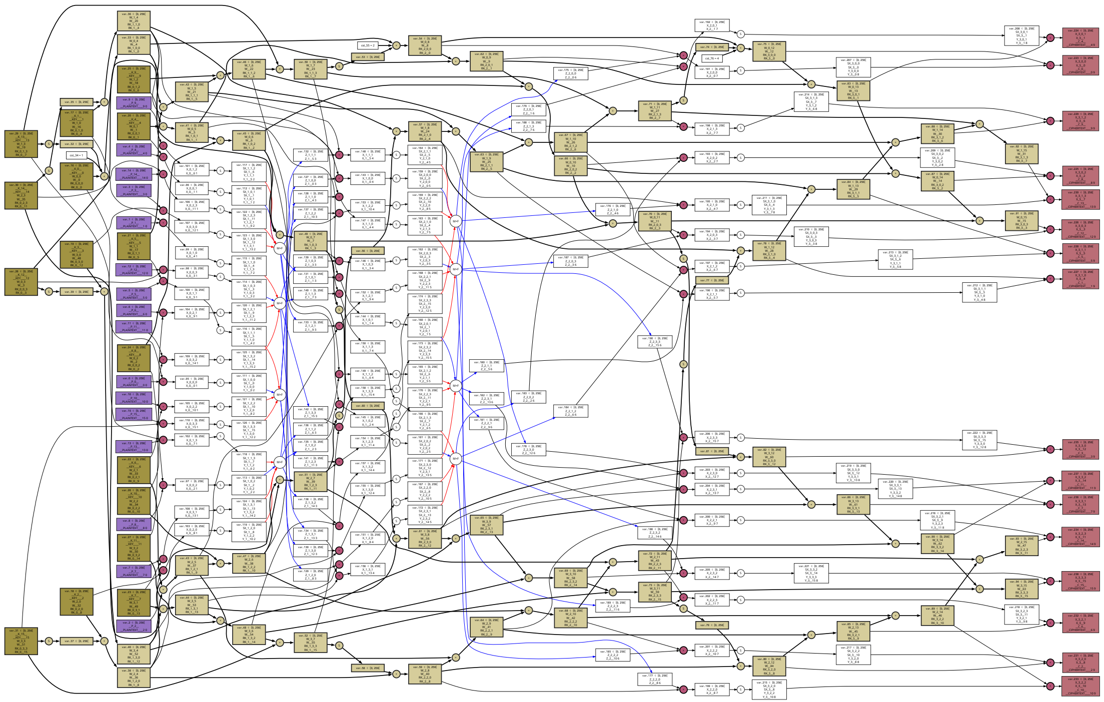


Figure 5: Representation of 3 rounds of AES in the TAGADA library.

Test vectors

D.3 Other Block Lengths and Key Lengths

The values in this section correspond to the ciphertexts obtained by encrypting the all-zero string with the all-zero key (values on the first lines), and by encrypting the result again with the all-zero key (values on the second lines). The values are given for the five different block lengths and the five different key lengths. The values were generated with the program listed in Appendix E.

```
block length 128  key length 128
66E94BD4EF8A2C3B884CFA59CA342B2E
F795BD4A52E29ED713D313FA20E98DBC
```

Figure 6: AES-128 test vectors.

```
#!/bin/bash
java -jar 'tagada.jar' \
  test vector aes128.spec.json \
  --radix hex \
  --plaintext  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00 \
  --key        00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00 \
  --ciphertext 66,E9,4B,D4,EF,8A,2C,3B,88,4C,FA,59,CA,34,2B,2E

### Output ###
ok
```

How TAGADA works?

Differentiate

Differential Attack

What is the probability to observe an output difference δ_c when we have injected an input difference δ_m in the plaintext and a difference δ_k in the key ?

Differentiate

Differential Attack

What is the probability to observe an output difference δ_c when we have injected an input difference δ_m in the plaintext and a difference δ_k in the key ?

Objective

Transform the specification graph into a differential graph.

Differentiate

Differential Attack

What is the probability to observe an output difference δ_c when we have injected an input difference δ_m in the plaintext and a difference δ_k in the key ?

Objective

Transform the specification graph into a differential graph.

$$c = S[m \oplus k \oplus 10]$$

Differentiate

Differential Attack

What is the probability to observe an output difference δ_c when we have injected an input difference δ_m in the plaintext and a difference δ_k in the key ?

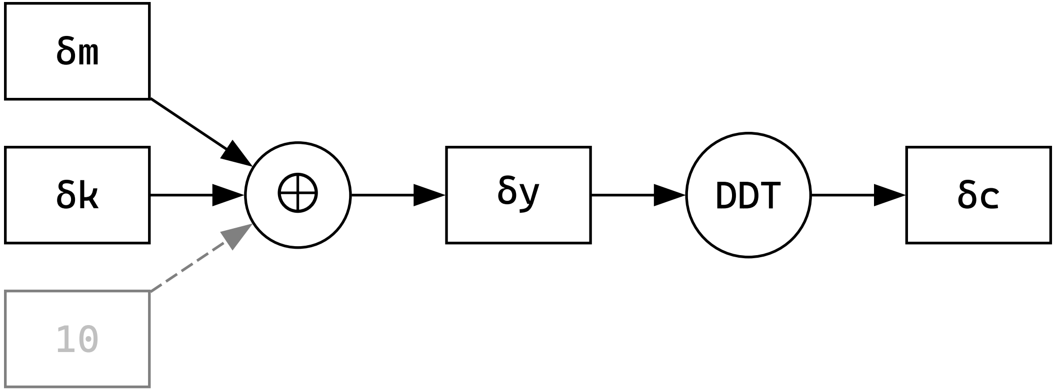
Objective

Transform the specification graph into a differential graph.

$$c = S[m \oplus k \oplus 10] \rightarrow \Pr[\delta_c \mid \delta_m \wedge \delta_k]$$

How TAGADA works?

$$\Pr[\delta_c \mid \delta_m \wedge \delta_k] = \Pr\left[\delta_c \mid \delta_y\right] \cdot \Pr\left[\delta_y \mid \delta_m \wedge \delta_k\right]$$



$$G = (N, E):$$

$$N = \{\delta_m, \delta_k, \delta_y, \delta_z, \oplus, \mathrm{DDT}\},$$

$$E = \{((\delta_m, \delta_k), \oplus, (\delta_y)), ((\delta_y), \mathrm{DDT}, (\delta_c))\}$$

The model

$$\text{var } \delta_m : 0..15;$$

$$\text{var } \delta_k : 0..15;$$

$$\text{var } \delta_y : 0..15;$$

$$\text{var } \delta_c : 0..15;$$

$$\text{var } \Pr\left[\delta_z \mid \delta_y\right] : \left\{1, 2^{-2}, 2^{-3}\right\};$$

$$\text{constraint } \delta_y = \delta_m \oplus \delta_k;$$

$$\text{constraint } \left(\delta_y, \delta_c, \Pr\left[\delta_c \mid \delta_y\right]\right) \in \text{Table}_{\text{DDT}};$$

$$\text{var objective} = \Pr\left[\delta_z \mid \delta_y\right];$$

$$\text{maximize objective};$$

Command

```
# Transform a specification graph into a differential one
java -jar 'tagada.jar' transform differentiate \
  aes128-r3.spec.json > aes128-r3.diff.json
```

Truncate Differential

Objective

Transforms the differential graph into a truncated differential graph.

Truncate Differential

Objective

Transforms the differential graph into a truncated differential graph.

Truncated Differential Attack
What is the probability to have Δ_c when we have Δ_m and Δ_k ?

Truncate Differential

Objective

Transforms the differential graph into a truncated differential graph.

Truncated Differential Attack
What is the probability to have Δ_c when we have Δ_m and Δ_k ?

We must transform the representation of the graph into a truncated differential attack representation.

Truncate Differential

Objective

Transforms the differential graph into a truncated differential graph.

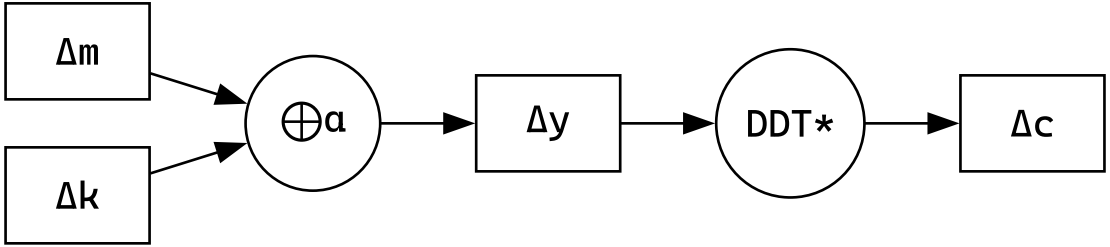
Truncated Differential Attack

What is the probability to have Δ_c when we have Δ_m and Δ_k ?

We must transform the representation of the graph into a truncated differential attack representation.

$$\Pr[\delta_c \mid \delta_m \wedge \delta_k] \rightarrow \Pr[\Delta_c \mid \Delta_m \wedge \Delta_k]$$

$$\Pr[\Delta_c \mid \Delta_m \wedge \Delta_k] = \Pr\left[\Delta_c \mid \Delta_y\right] \cdot \Pr\left[\Delta_y \mid \Delta_m \wedge \Delta_k\right]$$



$$G = (N, E):$$

$$N = \left\{\Delta_m, \Delta_k, \Delta_y, \Delta_z, \oplus_\alpha, \mathrm{DDT}^*\right\},$$

$$E = \left\{\left((\Delta_m, \Delta_k), \oplus_\alpha, (\Delta_y)\right), \left((\Delta_y), \mathrm{DDT}^*, (\Delta_c)\right)\right\}$$

The model

$$\text{var } \Delta_m : 0..1;$$

$$\text{var } \Delta_k : 0..1;$$

$$\text{var } \Delta_y : 0..1;$$

$$\text{var } \Delta_c : 0..1;$$

$$\text{var } \Pr[\Delta_c \mid \Delta_y] : \{1, 2^{-2}\};$$

$$\text{constraint } (\Delta_y, \Delta_m, \Delta_k) \in \{(0,0,0), (1,0,1), (0,1,1), (1,1,1)\};$$

$$\text{constraint } (\Delta_y, \Delta_c, \Pr[\Delta_c \mid \Delta_y]) \in \text{Table}_{\text{DDT}^*};$$

$$\text{var objective} = \Pr[\Delta_z \mid \Delta_y];$$

$$\text{maximize objective;}$$

Command

```
# Transform a differential graph into a truncated differential one
java -jar 'tagada.jar' transform truncate-differential \
  aes128-r3.diff.json \
  # Added improvement of [5], this is necessary for
  # ciphers with MDS or Pseudo MDS properties (e.g. AES, Midori)
  --generate-xors 5 \
  --diff-variables --transitivity \
  > aes128-r3.trunc.json
```

Searching Truncated Differential Characteristic (Step1)

```
# Transform a differential graph into a truncated differential one
java -jar 'tagada.jar' search \
  best-truncated-differential-characteristic \
  aes128-r3.trunc.json
```

```
{"__P_0__":1,...,"in_59":0,"out_59":0,"objective":3000}
```

// $2^{-30.00}$

Searching Differential Characteristic (Step2)

```
java -jar 'tagada.jar' \
  search best-differential-characteristic \
  aes128-r3.trunc.json aes128-r3.diff.json
```

```
{"__P_0__":18,...,"in_59":0,"out_59":0,"objective":3100}
```

// $2^{-31.00}$

Our results

Step 1 [6]

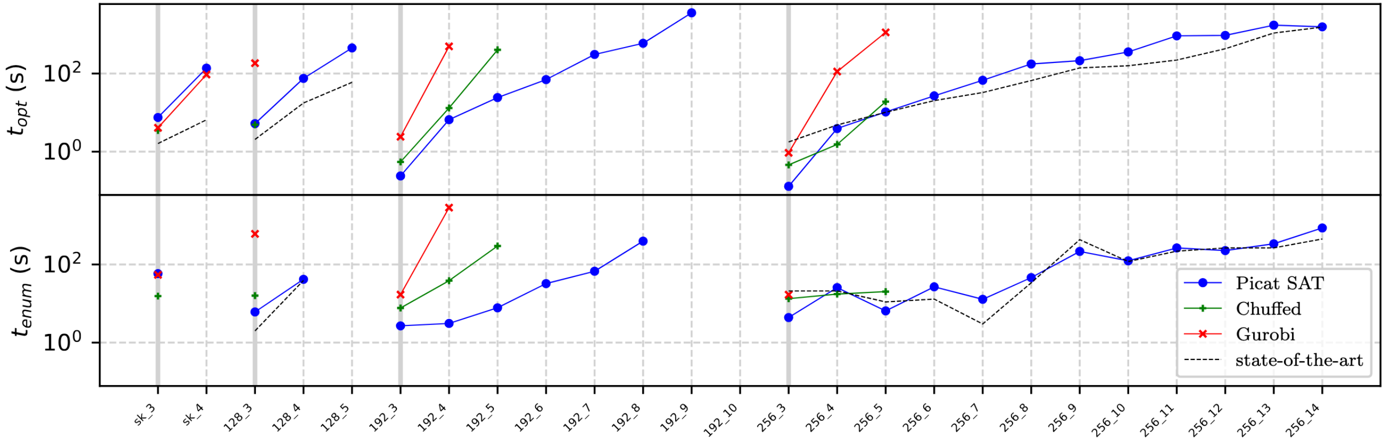


Figure 7: CPU time of Picat-SAT, Chuffed and Gurobi on the model generated by Tagada for AES instances when `--generate-xors` = 5 and is selected (top plot for Step1-opt and bottom plot for Step1-enum). State-of-the art is the handcrafted model of [5] run with Picat-SAT.

Step 2 [7]

Cipher	$R_{\max}$	Pr	Ref.
Midori-64	16	2^{-16}	[8]
Midori-128	20	2^{-40}	[8]
Warp	41	2^{-40}	[9]
Twine-80	18	2^{-64}	[10]
Twine-128	16	2^{-52}	[10]
Skinny-64-TK1	11	2^{-64}	[11]
Skinny-128-TK1	11	2^{-74}	[11]
Rijndael-128-128	5	2^{-105}	[12]
Rijndael-128-160	7	2^{-120}	[12]
Rijndael-128-192	9	2^{-146}	[12]
Rijndael-128-224	12	2^{-212}	[12]

Cipher	$R_{\max}$	Pr	Ref.
Rijndael-128-256	14	2^{-146}	[12]
Rijndael-160-128	4	2^{-112}	[12]
Rijndael-160-160	6	2^{-138}	[12]
Rijndael-160-192	8	2^{-141}	[12]
Rijndael-160-224	9	2^{-190}	[12]
Rijndael-160-256	11	2^{-204}	[12]
Rijndael-192-128	3	2^{-54}	[12]
Rijndael-192-160	5	2^{-118}	[12]
Rijndael-192-192	7	2^{-153}	[12]
Rijndael-192-224	8	2^{-205}	[12]
Rijndael-192-256	9	2^{-179}	[12]
Rijndael-224-128	3	2^{-54}	[12]

Cipher	$R_{\max}$	Pr	Ref.
Rijndael-224-160	4	2^{-122}	[12]
Rijndael-224-192	5	2^{-124}	[12]
Rijndael-224-224	7	2^{-196}	[12]
Rijndael-224-256	8	2^{-182}	[12]
Rijndael-256-128	3	2^{-54}	[12]
Rijndael-256-160	4	2^{-130}	[12]
Rijndael-256-192	5	2^{-148}	[12]
Rijndael-256-224	4	2^{-115}	[12]
Rijndael-256-256	6	2^{-129}	[12]

Table 1: Best differential trails recovered with Tagada (time limit of one day). Detailed results will be available in an extended version of the current paper on eprint.

Further work

Codebase refactor

Tagada 1.0

Python, Ruby and Rust (Python bindings)

Tagada 2.0 [7]

Ruby, Rust, Kotlin/Java, MiniZinc (+ external dependencies Picat, OrTools, Gurobi ,etc.)

Tagada 3.0 [WIP]

Ruby - Java

Working directions

- Creating dedicated constraints
  - Very poor filtering quality for the XOR operator
  - How to handle large S-Boxes?
- Include information specific to cryptographic problems in solvers
  - Round functions
  - Cipher symmetries
- Improving abstractions for truncated attacks
  - Creating tight linear system abstractions
  - Handle bit oriented ciphers
- Implementing new attacks

Thank you

Bibliography

[1] "Advanced Encryption Standard (AES)." Nov. 2001.

[2] L. Rouquette, "Improving scalability and reusability of differential cryptanalysis models using constraint programming," 2022. [Online]. Available: https://hal.science/tel-03894661

[3] S. Banik *et al.*, "Midori: A Block Cipher for Low Energy," in *Advances in Cryptology – ASIACRYPT~2015, Part~II*, T. Iwata and J. H. Cheon, Eds., in Lecture Notes in Computer Science, vol. 9453. Springer Berlin Heidelberg, Germany, Nov. 2015, pp. 411–436. doi: 10.1007/978-3-662-48800-3_17.

[4] S. Banik *et al.*, "WARP : Revisiting GFN for Lightweight 128-Bit Block Cipher," in *SAC 2020: 27th Annual International Workshop on Selected Areas in Cryptography*, O. Dunkelman, M. J. Jacobson Jr., and C. O'Flynn, Eds., in Lecture Notes in Computer Science, vol. 12804. Springer, Cham, Switzerland, Oct. 2020, pp. 535–564. doi: 10.1007/978-3-030-81652-0_21.

[5] D. Gérault, P. Lafourcade, M. Minier, and C. Solnon, “Computing AES related-key differential characteristics with constraint programming,” *Artif. Intell.*, vol. 278, 2020, doi: 10.1016/J.ARTINT.2019.103183.

[6] L. Libralesso, F. Delobel, P. Lafourcade, and C. Solnon, “Automatic Generation of Declarative Models For Differential Cryptanalysis,” in *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, L. D. Michel, Ed., in LIPIcs, vol. 210. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 1–18. doi: 10.4230/LIPICS.CP.2021.40.

[7] F. Delobel, P. Derbez, A. Gontier, L. Rouquette, and C. Solnon, “A CP-Based Automatic Tool for Instantiating Truncated Differential Characteristics,” in *Progress in Cryptology - INDOCRYPT~2023: 24th International Conference in Cryptology in India, Part~I*, A. Chattopadhyay, S. Bhasin, S. Picek, and C. Rebeiro, Eds., in Lecture Notes in Computer Science, vol. 14459. Springer, Cham, Switzerland, Dec. 2023, pp. 247–268. doi: 10.1007/978-3-031-56232-7_12.

[8] D. Gerault, “Security analysis of contactless communication protocols,” 2018. [Online]. Available: https://theses.hal.science/tel-02536478

[9] J. S. Teh and A. Biryukov, “Differential Cryptanalysis of WARP.” [Online]. Available: https://eprint.iacr.org/2021/1641

[10] K. Sakamoto *et al.*, “Security of Related-Key Differential Attacks on TWINE, Revisited,” *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, no. 1, pp. 212–214, 2020, doi: 10.1587/TRANSFUN.2019CIL0004.

[11] S. Delaune, P. Derbez, P. Huynh, M. Minier, V. Mollimard, and C. Prud'homme, “Efficient Methods to Search for Best Differential Characteristics on SKINNY,” in *ACNS 21: 19th International Conference on Applied Cryptography and Network Security, Part~II*, K. Sako and N. O. Tippenhauer, Eds., in Lecture Notes in Computer Science, vol. 12727. Springer, Cham, Switzerland, Jun. 2021, pp. 184–207. doi: 10.1007/978-3-030-78375-4_8.

[12] L. Rouquette, D. Gérault, M. Minier, and C. Solnon, “And Rijndael?: Automatic Related-Key Differential Analysis of Rijndael,” in *AFRICACRYPT 22: 13th International Conference on Cryptology in Africa*, L. Batina and J. Daemen, Eds., in Lecture Notes in Computer Science, vol. 2022. Springer, Cham, Switzerland, Jul. 2022, pp. 150–175. doi: 10.1007/978-3-031-17433-9_7.