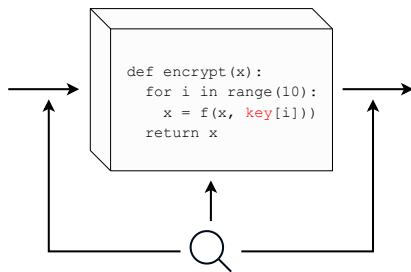
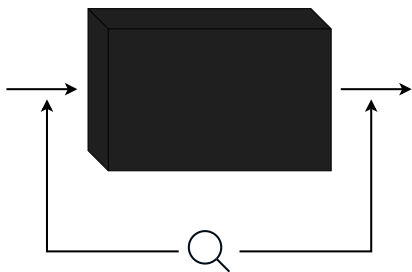




## IMPLICIT WHITE-BOX IMPLEMENTATIONS: WHITE-BOXING ARX CIPHERS

ADRIÁN RANEA, JOACHIM VANDERSMISSSEN, AND BART PRENEEL

# White-Box Cryptography



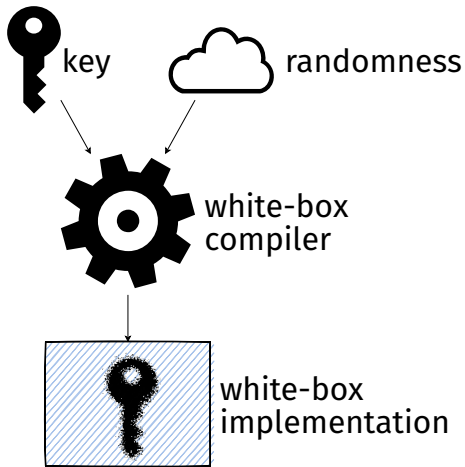
WBC: securing software crypto. implementations in the white-box model.

Applications: DRM, mobile payments, ...

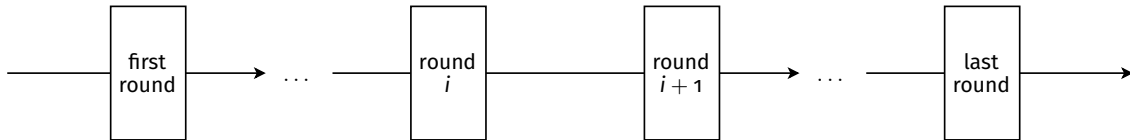
# White-Box Implementations of Block Ciphers

Academic white-box implementations:

- **Fixed** hard-coded cipher key.
- Compiler/method **public**.
- Security goal: **key-extraction resistance**.

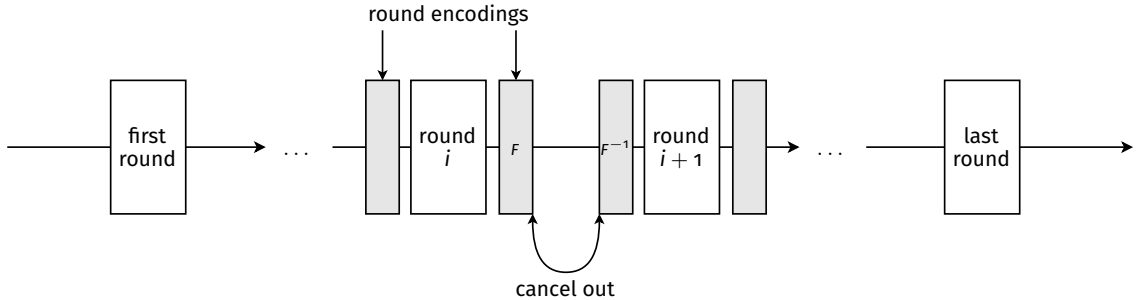


# CEJO Implementations



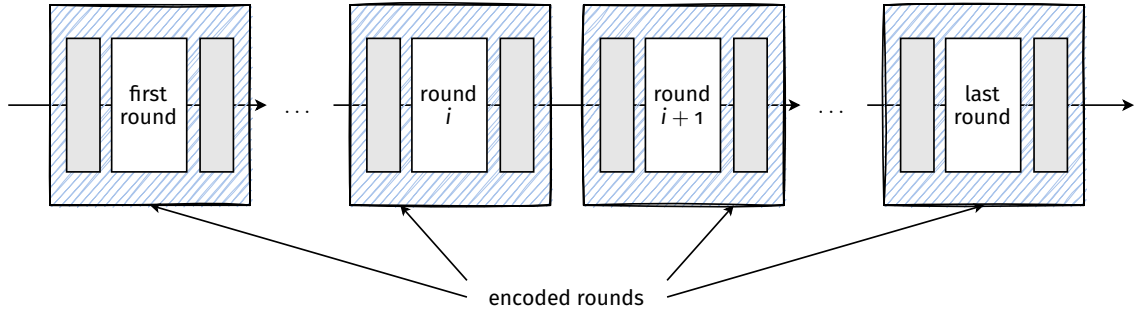
Chow S, Eisen P, Johnson H & Van Oorschot PC. *White-Box Cryptography and an AES Implementation*. SAC 2002.

# CEJO Implementations



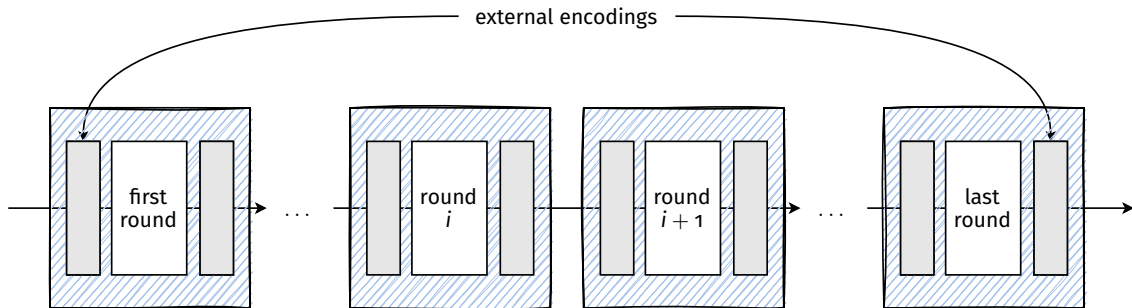
Chow S, Eisen P, Johnson H & Van Oorschot PC. *White-Box Cryptography and an AES Implementation*. SAC 2002.

# CEJO Implementations



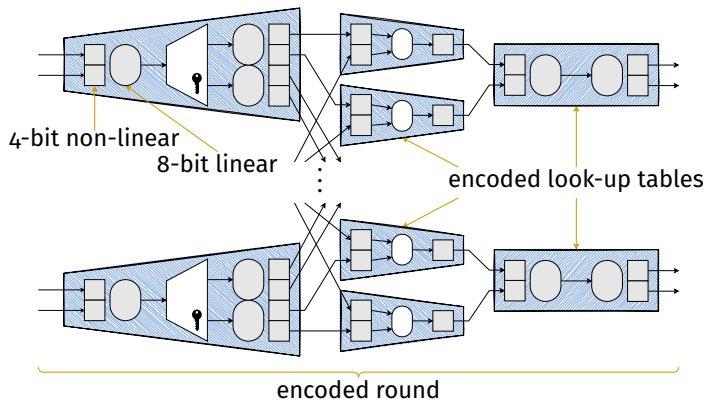
Chow S, Eisen P, Johnson H & Van Oorschot PC. *White-Box Cryptography and an AES Implementation*. SAC 2002.

# CEJO Implementations



Chow S, Eisen P, Johnson H & Van Oorschot PC. *White-Box Cryptography and an AES Implementation*. SAC 2002.

# CEJO round

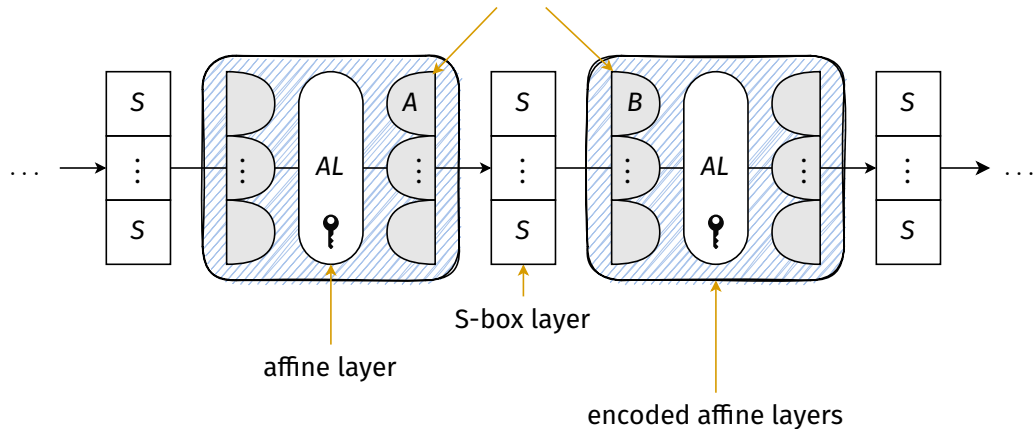


**All** CEJO implementations have been **broken**.



# Self-Equivalence Implementations

$$(A, B) \text{ self-equivalence of } S \iff S = B \circ S \circ A$$



# Self-Equivalence (SE) Implementations

- No look-up tables,  
SE efficient with **large encodings**.
- CEJO can be reduced to SE,  
but the converse doesn't hold.
- **Problem:** difficult to find  
non-linear layers with  
**many** and **large** self-equivalences.

# Self-Equivalence (SE) Implementations

- No look-up tables, SE efficient with **large encodings**.
- CEJO can be reduced to SE, but the converse doesn't hold.
- **Problem:** difficult to find non-linear layers with **many** and **large** self-equivalences.

Candidate non-linear layer: **permuted modular addition**  $x, y \mapsto (x \boxplus y, y)$

## Finding Self-equivalences of $x, y \mapsto (x \boxplus y, y)$

$F$  is **CCZ-equivalent** to  $G$  if the graph of  $F$ ,  $\{(x, F(x))\}$ , is equal to the graph of  $G$  up to an affine permutation.

- $F(x, y) = (x \boxplus y, y)$  is CCZ-equivalent to a quadratic function  $G$ .

## Finding Self-equivalences of $x, y \mapsto (x \boxplus y, y)$

$F$  is **CCZ-equivalent** to  $G$  if the graph of  $F$ ,  $\{(x, F(x))\}$ , is equal to the graph of  $G$  up to an affine permutation.

- $F(x, y) = (x \boxplus y, y)$  is CCZ-equivalent to a quadratic function  $G$ .

A **graph automorphism** of  $F$  is an affine permutation mapping the graph of  $F$  to itself.


**CCZ-based method** to find self-equivalences of  $F$ :


- Find graph automorphisms of low-degree  $G$  by solving a functional equation.
- Transform graph automorphisms of  $G$  into self-equivalences of  $F$  using CCZ-equivalence.


# github.com/ranea/BoolCrypt


## ranea/BoolCrypt


Python library for vectorial Boolean functions in cryptography




 1 Contributor

 0 Issues

 5 Stars

 1 Fork



GITHUB.COM

GitHub - ranea/BoolCrypt: Python library for vectorial Boolean functions in cryptography

### BoolCrypt

0.1

BoolCrypt 0.1

API reference

- boolcrypt.utilities module
- boolcrypt.evaluation module
- boolcrypt.functionalequations module
- boolcrypt.cczselfequivalence module**
- boolcrypt.sboxes module
- boolcrypt.modularaddition module
- boolcrypt.se\_pmodadd package
- boolcrypt.classification module
- boolcrypt.findpoly module
- boolcrypt.findpolymodp module
- boolcrypt.findpolyoptimal module

» API reference » boolcrypt.cczselfequivalence module View page source

## boolcrypt.cczselfequivalence module

Find self-equivalences of a function by finding the self-equivalences of its graph (i.e., also called graph automorphisms) parametrized by a CCZ-equivalent function with lower degree.

```
boolcrypt.cczselfequivalence.find_self_equivalence(ccz_anf,
admissible_mapping, ccz_anf_implicit=False, right_se_degree=1, inv_left_se_degree=1,
se_ct_terms=True, ignore_diagonal_equations=False, add_invertibility_equations=True,
ignore_determinant_equation=False, check_se=True, bpr=None, ccz_se_anf=None,
prefix_se_coeffs='c', input_ccz_anf_vars=None, anf=None, input_anf_vars=None,
num_input_anf_vars=None, return_ccz_se=False, verbose=False, debug=False,
filename=None, *solve_args) | source
```

Find a SE of F by finding a SE of the graph of G.

Let F be the function (optionally) given by `anf` and G its CCZ-equivalent function through the `admissible_mapping` L, that is,  $\text{Graph}(F) = L(\text{Graph}(G))$ . If (if given) and G must be in ANF form, but L can be given in ANF, as a matrix, or as a (matrix, vector) pair. If F is not given, its number of input variables must be given in `num_input_anf_vars`.

$\text{Graph}(F)$  is defined as usual,  $\{(x, y) : \text{for all } x, y = F(x)\}$ . If `ccz_anf_implicit=False`,  $\text{Graph}(G)$  is defined similarly as  $\text{Graph}(F)$ : Otherwise,  $\text{Graph}(G) = \{(x, y) : G(x, y) = 0\}$  if `ccz_anf_implicit=True`.

This method finds a self-equivalence (SE) (A, B) with given degrees of F (a pair of permutations (A, B) such that  $B \circ F \circ A = F$ ) by finding a SE (an automorphism) of the graph of F parametrized by G. A is also called a right SE and B a left SE. If no solution is found, None is returned.

If the SE degrees are both 1 and `se_ct_terms=True` (resp. False), this method finds an affine (resp. linear) SE.

This method returns SE (A, B) by finding a  $\text{Graph}(G)$ -SE  $C = (c_0, c_1)$  s.t.  $L \cdot C \cdot L^{-1}$  is diagonal and can be written as  $L \cdot C \cdot L^{-1} = (A, B^{-1})$ . This is

# Self-Equivalences of the Permuted Modular Addition

SE found for wordsize  $4 \leq n \leq 64$ :

Type	$\#\{(A, B) : S = B \circ S \circ A\}$
Linear	$3 \times 2^{2n+2}$
Affine	$3 \times 2^{2n+8}$
Affine-quadratic	$3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$

Open problem: prove these SE subsets are the full SE groups for  $n \geq 4$ .

# Self-Equivalences of the Permuted Modular Addition

SE found for wordsize  $4 \leq n \leq 64$ :

Type	$\#\{(A, B) : S = B \circ S \circ A\}$
Linear	$3 \times 2^{2n+2}$
Affine	$3 \times 2^{2n+8}$
Affine-quadratic	$3^2 \times 2^{3n+14} - 3 \times 2^{2n+8}$

Open problem: prove these SE subsets are the full SE groups for  $n \geq 4$ .

$$\left( \begin{array}{ccc|ccc} \star & & & \star & & \\ \star & 1 & & \star & & \\ \vdots & & \ddots & \vdots & & \\ \star & & & \star & & \\ \star & & 1 & \star & \star & \dots \star \star \\ \hline \star & & & \star & & \\ \star & & & \star & 1 & \\ \vdots & & & \vdots & & \ddots \\ \star & & & \star & & 1 \\ \star & & & \star & & \star 1 \end{array} \right)$$

$$(\star b \dots b \star \star \mid \star b \dots b \star \star)$$

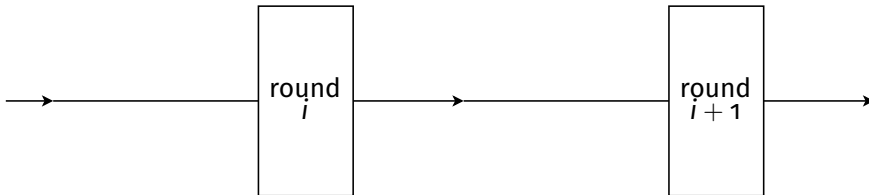


# Implicit framework

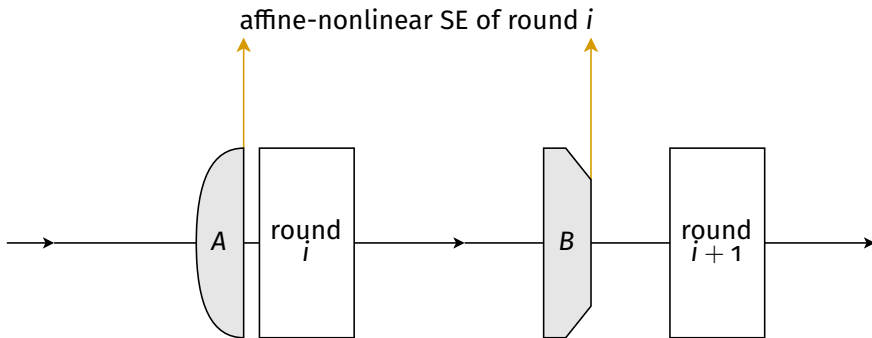
## Implicit implementation:

- an **encoded** implementation where
- the round encodings are the composition of **affine permutations** and **affine-nonlinear self-equivalences**,
- and the encoded round functions are implemented by **systems of low-degree equations**.

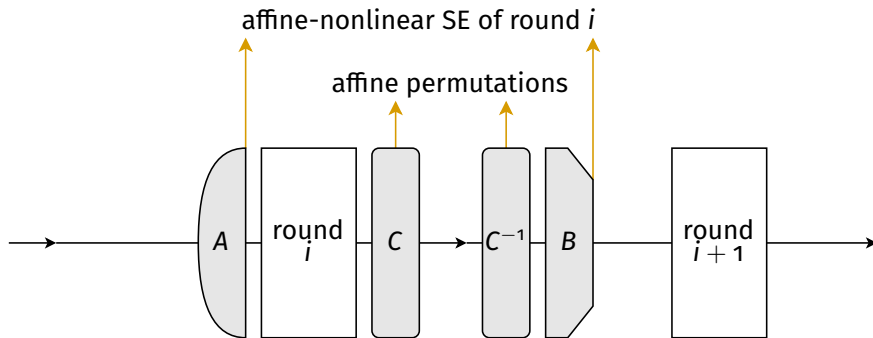
# Round Encodings in an Implicit Implementation



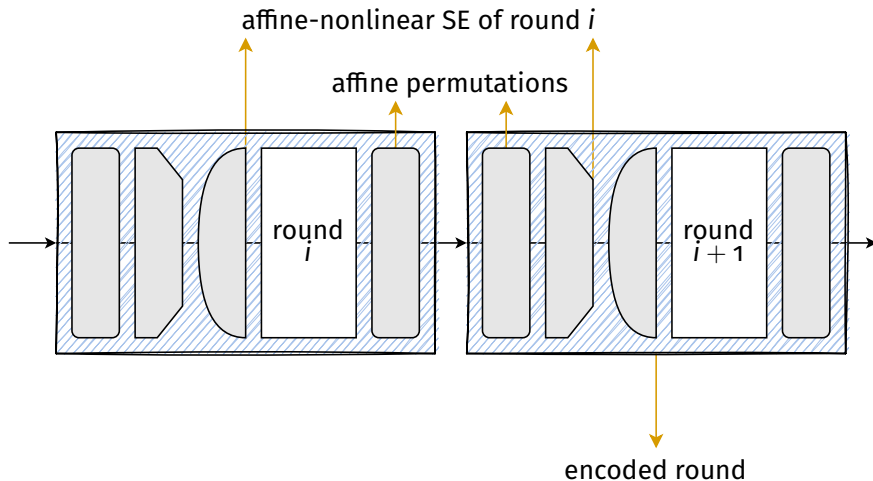
# Round Encodings in an Implicit Implementation



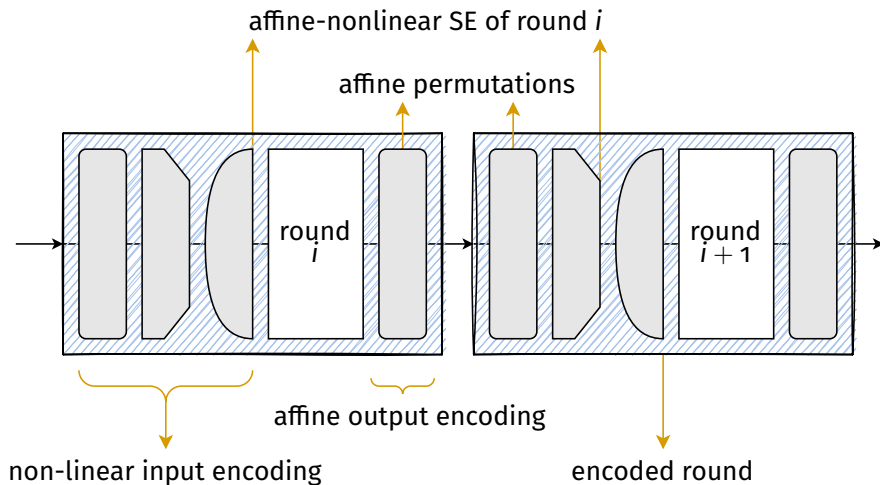
# Round Encodings in an Implicit Implementation



# Round Encodings in an Implicit Implementation



# Round Encodings in an Implicit Implementation



# Implicit Round Functions

Implemented by low-degree quasilinear implicit functions:

- $P$  is an **implicit** function of  $F$  if

$$P(x, y) = 0 \iff y = F(x).$$

- Evaluate  $F(x_0)$  by substituting  $x_0 = x$  and solving  $P(x_0, y) = 0$  for  $y$ .
- Fast solving if  $P$  is **quasilinear**:

$\forall x$ , the function  $y \mapsto P(x, y)$  is affine.

Permuted modular addition has a quasilinear quadratic implicit function!

## Size of an Implicit Round Function

Upper bound on the size of a  $(2n, n)$ -bit  $P$  for an  $n$ -bit cipher.

Cipher blocksize	Degree of $P$	Size of $P$
64	2	0.05 MB
64	3	1.42 MB
64	4	6.50 MB
128	2	0.40 MB
128	3	22.50 MB
128	4	193.19 MB

For the permuted modular addition,  $P$  is cubic or quartic if affine-quadratic self-equivalences are used.



# Security of Implicit Implementations

Security goal: **key-extraction** resistance.

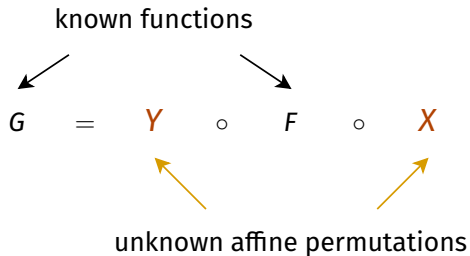
An implicit implementation is secure against **known generic** attacks if

- quadratic input encodings OR
- large non-linear layer

Implicit framework cannot secure SPN ciphers with affine encodings.

# Security of Implicit Implementations

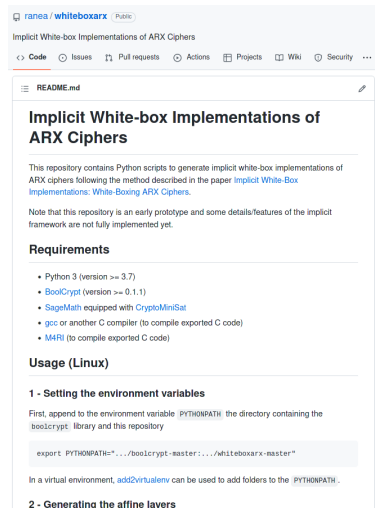
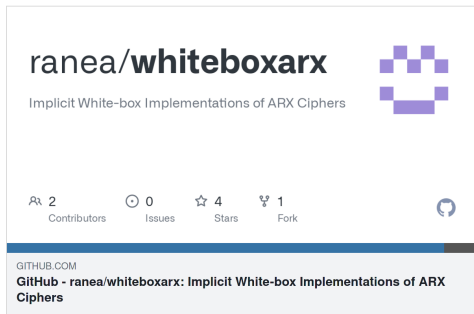
New generic attack (**reduction** to self-equivalence implementations)  
based on functional equations (**affine equivalence problems**):



# Security of Implicit Implementations

function	$\bar{E} = O \circ (L \circ S) \circ \oplus_k \circ I$	$P = V \circ T \circ U \circ (\oplus_k \parallel L^{-1}) \circ (I \parallel O^{-1})$
equation	$\bar{E} = Y \circ (L \circ S) \circ X$	$P = Y \circ T \circ X$
degree	high	low
access	black-box	white-box
goal	find any solution	find any solution and guess $U$

# github.com/ranea/whiteboxarx



# Conclusion

- **Implicit framework**: new design that prevents generic attacks, first method applicable to ARX ciphers.
- New **method to find self-equivalences** based on the CCZ-equivalence, applied to the permuted modular addition.
- Two open-source tools: **BoolCrypt** and **whiteboxarx**.
- Future work: new attacks, other non-linear layers, ...