CLAASP

Halil İbrahim Kaplan

TDD / Kripto Analiz Laboratuvarı

halil.kaplan@tubitak.gov.tr

2024

Overview

1. General Review

2. Some Test Results

3. Conclusions

CLAASP: a Cryptographic Library for the Automated Analysis of Symmetric Primitives

Emanuele Bellini[1], David Gerault[1], Juan Grados[1], Yun Ju Huang[1], Rusydi Makarim[2], Mohamed Rachidi[1], and Sharwan Tiwari[1]

[1] Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE
{emanuele.bellini, david.gerault, juan.grados, yunju.huang, mohamed.rachidi,sharwan.tiwari}@tii.ae
[2] rusydi.hasan@gmail.com

Abstract. This paper introduces CLAASP, a Cryptographic Library for the Automated Analysis of Symmetric Primitives. The library is designed to be modular, extendable, easy to use, generic, efficient and *fully* automated. It is an extensive toolbox gathering state-of-the-art techniques aimed at simplifying the manual tasks of symmetric primitive designers and analysts. CLAASP is built on top of Sagemath and is open-source under the GPLv3 license.
The central input of CLAASP is the description of a cryptographic primitive as a list of connected components in the form of a directed acyclic graph. From this representation, the library can automatically: (1) generate the Python or C code of the primitive evaluation function, (2) execute a wide range of statistical and avalanche tests on the primitive, (3) generate SAT, SMT, CP and MILP models to search, for example, differential and linear trails, (4) measure algebraic properties of the primitive, (5) test neural-based distinguishers. We demonstrate that CLAASP can reproduce many of the results that were obtained in the literature and even produce new results.
In this work, we also present a comprehensive survey and comparison of other software libraries aiming at similar goals as CLAASP.

Keywords: Cryptographic library · Automated analysis · Symmetric primitives

General Review

The basic block of CLAASP is the description of a cryptographic primitives in the form of a **list of connected components** (S-Box, LinearLayer, Constants, Input/Output, etc.).

The basic block of CLAASP is the description of a cryptographic primitives in the form of a **list of connected components** (S-Box, LinearLayer, Constants, Input/Output, etc.).

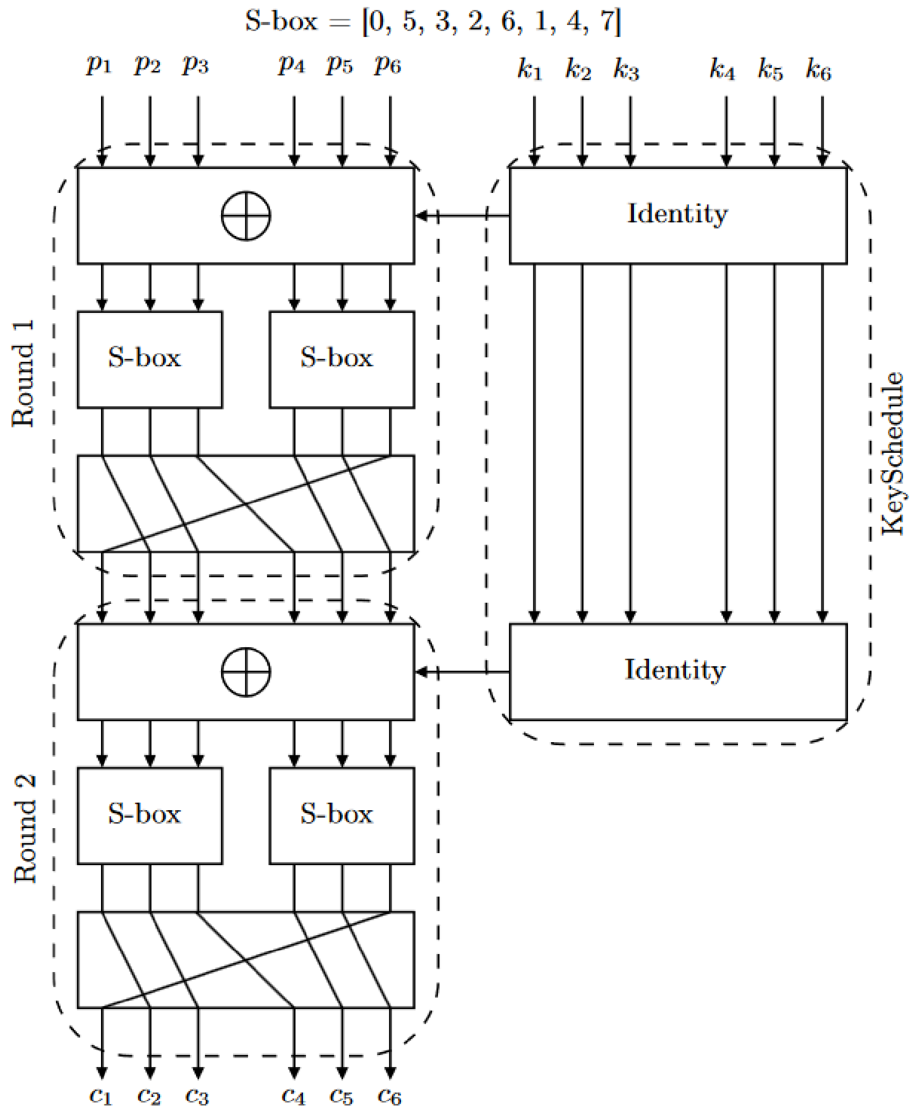
From this representation, the library can:

- generate the Python or C code of the encryption function,
- execute a wide range of statistical and avalanche tests on the primitive,
- automatically generate SAT, SMT, CP and MILP models to search, for example, differential and linear trails,
- measure algebraic properties of the cipher,
- test neural-based distinguishers.

General Review

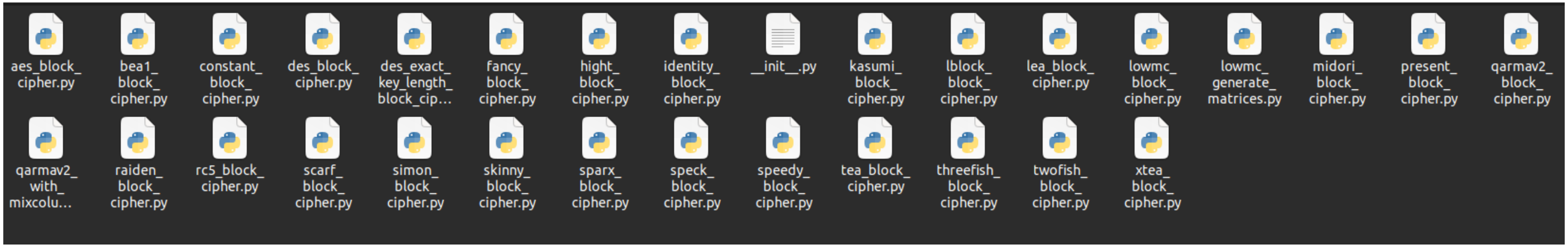
		TAGADA	CASCADA	CryptoSMT	lineartrails	YAARX	Autoguess	CLAASP
Cipher types		SPN	All	All	SPN	ARX	All	All
Cipher representation		DAG	Python code	Python code	C++ code	C code	Algebraic representation	DAG
Statistical/Avalanche tests		-	-	-	-	-	-	Yes
Continuous diffusion tests		-	-	-	-	-	-	Yes
Components analysis tests		-	-	-	-	-	-	Yes
Constraint solvers	Differential trails	Truncated	Yes	Yes	-	Yes	-	Yes
	Differentials	-	Yes	Yes	-	Yes	-	Yes
	Impossible differential	-	Yes	-*	-	-	-	Yes
	Linear trails	-	Yes	Yes	Yes	-	-	Yes
	Linear hull	-	-*	-*	-	-	-	Yes
	Zero correlation approximation	-	Yes	-*	-	-	-	Yes
	Supported solvers	CP, (MiniZinc)	SMT	SMT	-	-	SAT, SMT, MILP, CP, Groebner basis	SAT, SMT, MILP, CP, Groebner basis
	Supported Scenarios	single-key related-key	single-key related-key	single-key related-key	single-key	single-key	single-key related-key single-tweak related-tweak	single-key related-key single-tweak related-tweak
Algebraic tests		-	-	-	-	-	-	Yes**
Neural-based tests		-	-	-	-	-	-	Yes
State Recovery		-	-	-	-	-	Yes	-
Key-bridging		-	-	-	-	-	Yes	-

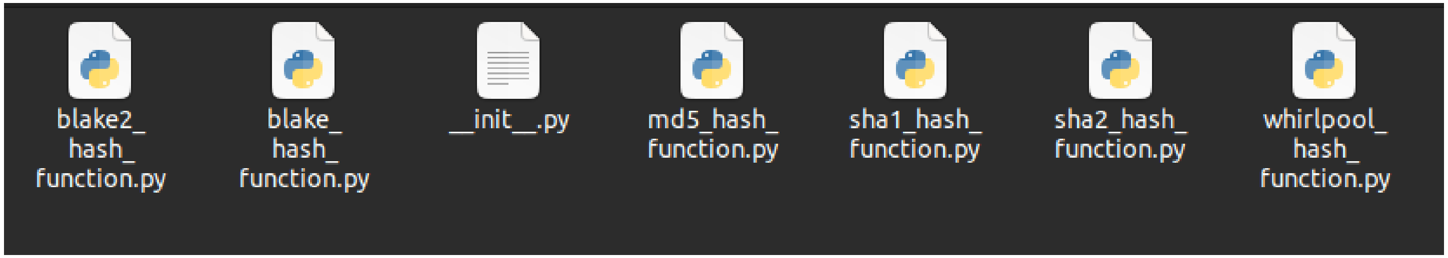
Table 1: Comparison of cryptanalysis libraries features with CLAASP. -* means that the functionality is not supported, but could easily be added from the existing code. ** means the algebraic tests works on algebraic model for cipher preimages.

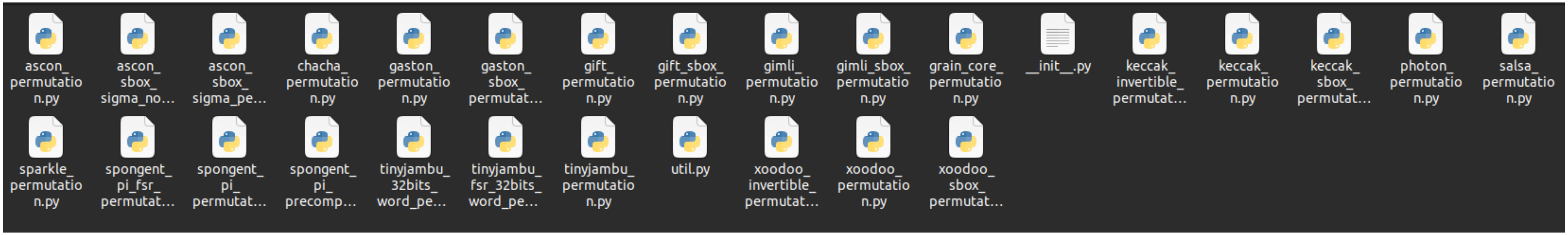

S-box = [0, 5, 3, 2, 6, 1, 4, 7]
p_1 p_2 p_3 p_4 p_5 p_6
k_1 k_2 k_3 k_4 k_5 k_6
Identity
S-box
S-box
Round 1
KeySchedule
Identity
S-box
S-box
Round 2
c_1 c_2 c_3 c_4 c_5 c_6

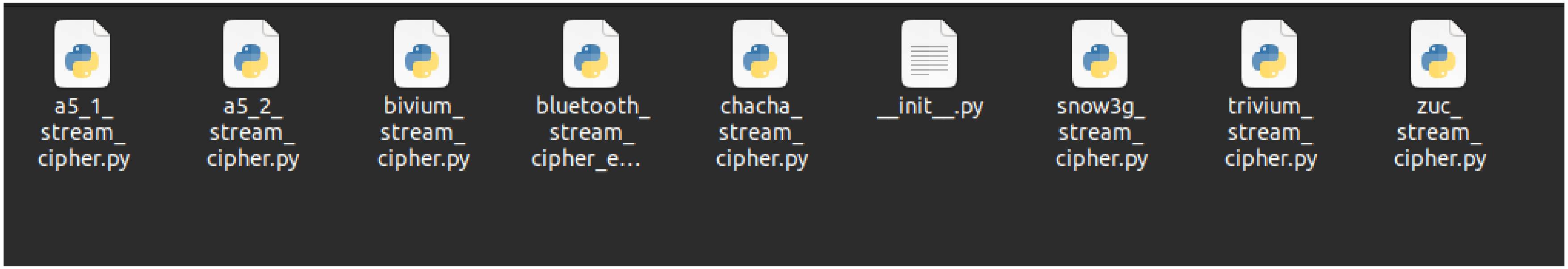
```
sage: from claasp.cipher import Cipher
....:
....: class ToySPN(Cipher):
....:     def __init__(self):
....:         super().__init__(
....:             family_name="toyspn",
....:             cipher_type="block_cipher",
....:             cipher_inputs=["plaintext", "key"],
....:             cipher_inputs_bit_size=[6, 6],
....:             cipher_output_bit_size=6
....:             )
....:         sbox = [0, 5, 3, 2, 6, 1, 4, 7]
....:         self.add_round()
....:         xor = self.add_XOR_component(["plaintext", "key"],[[0,1,2,3,4,5],[0,1,2,3,4,5]],6)
....:         sbox1 = self.add_SBOX_component([xor.id], [[0, 1, 2]], 3, sbox)
....:         sbox2 = self.add_SBOX_component([xor.id], [[3, 4, 5]], 3, sbox)
....:         rotate = self.add_rotate_component([sbox1.id, sbox2.id],[[0, 1, 2], [0, 1, 2]], 6, 1)
....:
....:         self.add_round_output_component([rotate.id], [[0, 1, 2, 3, 4, 5]], 6)
....:         self.add_round()
....:         xor = self.add_XOR_component([rotate.id, "key"],[[0,1,2,3,4,5],[0,1,2,3,4,5]],6)
....:         sbox1 = self.add_SBOX_component([xor.id], [[0, 1, 2]], 3, sbox)
....:         sbox2 = self.add_SBOX_component([xor.id], [[3, 4, 5]], 3, sbox)
....:         rotate = self.add_rotate_component([sbox1.id, sbox2.id],[[0, 1, 2], [0, 1, 2]], 6, 1)
....:
....:         self.add_cipher_output_component([rotate.id], [[0, 1, 2, 3, 4, 5]], 6)
....:
sage:
```

General Review









General Review

CLAASP: Cryptographic Library for Automated Analysis of Symmetric Primitives 1.1.0 documentation » CLAASP: Cryptographic Library for Automated Analysis of Symmetric Primitives

Table of Contents

CLAASP: Cryptographic Library for Automated Analysis of Symmetric Primitives

This is a sample reference manual for CLAASP.

To use this module, you need to import it:

```
from claasp import *
```

This reference shows a minimal example of documentation of CLAASP following SageMath guidelines.

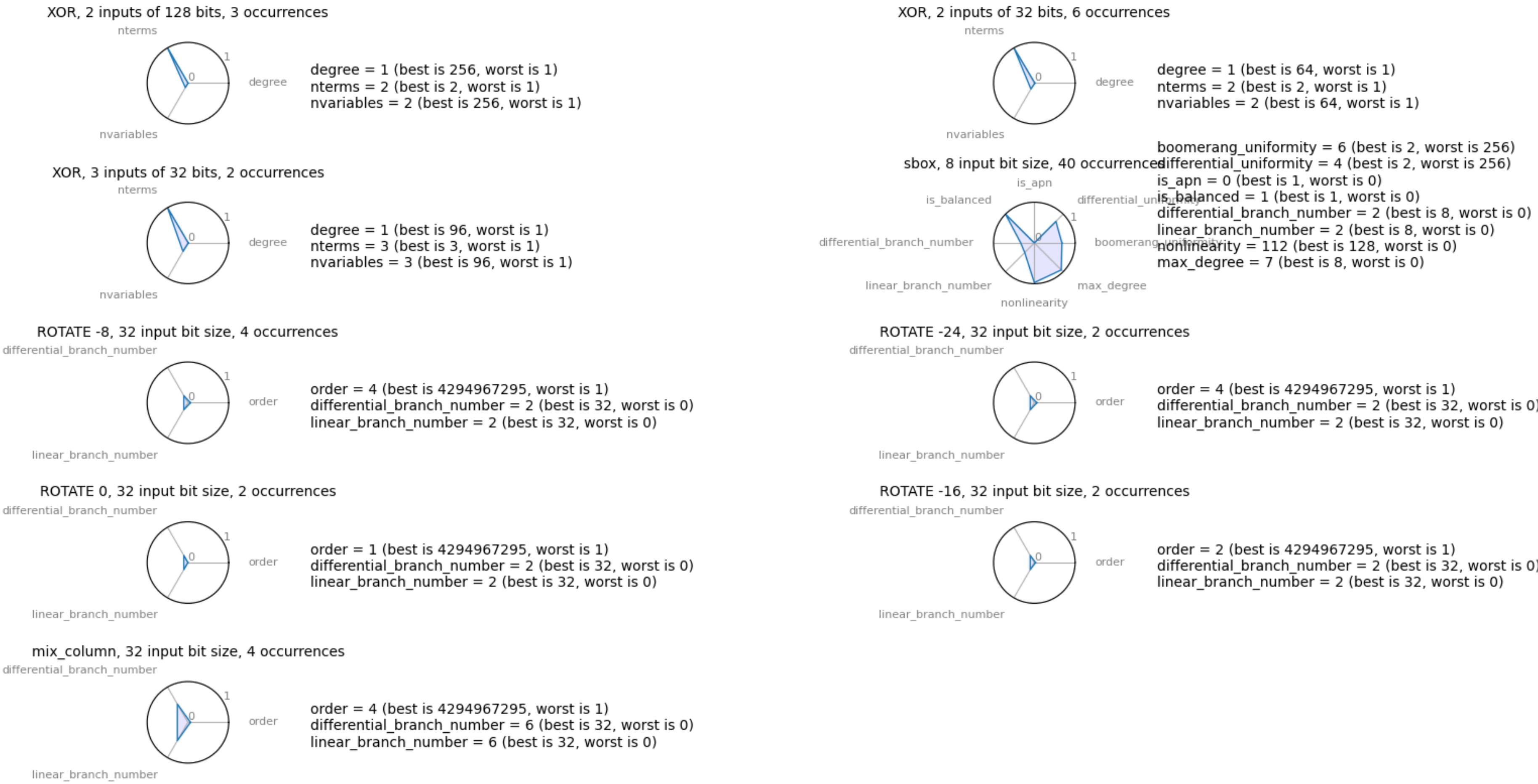
- Compound xor differential cipher
- Editor
- Cipher
- Component
- Rounds
- Round
- Input

Cipher modules

Some Test Results

Some Test Results

```
sage: from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
....: from claasp.cipher_modules.component_analysis_tests import CipherComponentsAnalysis
....: from claasp.cipher_modules.report import Report
....: cipher = AESBlockCipher(number_of_rounds=2)
....: test = CipherComponentsAnalysis(cipher)
....: results = test.component_analysis_tests()
....: report = Report(results)
....: report.show()
```


XOR, 2 inputs of 128 bits, 3 occurrences
nterms
degree
nvariables
degree = 1 (best is 256, worst is 1)
nterms = 2 (best is 2, worst is 1)
nvariables = 2 (best is 256, worst is 1)
XOR, 3 inputs of 32 bits, 2 occurrences
nterms
degree
nvariables
degree = 1 (best is 96, worst is 1)
nterms = 3 (best is 3, worst is 1)
nvariables = 3 (best is 96, worst is 1)
ROTATE -8, 32 input bit size, 4 occurrences
differential_branch_number
order
linear_branch_number
order = 4 (best is 4294967295, worst is 1)
differential_branch_number = 2 (best is 32, worst is 0)
linear_branch_number = 2 (best is 32, worst is 0)
ROTATE 0, 32 input bit size, 2 occurrences
differential_branch_number
order
linear_branch_number
order = 1 (best is 4294967295, worst is 1)
differential_branch_number = 2 (best is 32, worst is 0)
linear_branch_number = 2 (best is 32, worst is 0)
mix_column, 32 input bit size, 4 occurrences
differential_branch_number
order
linear_branch_number
order = 4 (best is 4294967295, worst is 1)
differential_branch_number = 6 (best is 32, worst is 0)
linear_branch_number = 6 (best is 32, worst is 0)
XOR, 2 inputs of 32 bits, 6 occurrences
nterms
degree
nvariables
degree = 1 (best is 64, worst is 1)
nterms = 2 (best is 2, worst is 1)
nvariables = 2 (best is 64, worst is 1)
sbox, 8 input bit size, 40 occurrences
is_apn
is_balanced
differential_uniformity
differential_branch_number
boomerang_uniformity
linear_branch_number
max_degree
nonlinearity
boomerang_uniformity = 6 (best is 2, worst is 256)
differential_uniformity = 4 (best is 2, worst is 256)
is_apn = 0 (best is 1, worst is 0)
is_balanced = 1 (best is 1, worst is 0)
differential_branch_number = 2 (best is 8, worst is 0)
linear_branch_number = 2 (best is 8, worst is 0)
nonlinearity = 112 (best is 128, worst is 0)
max_degree = 7 (best is 8, worst is 0)
ROTATE -24, 32 input bit size, 2 occurrences
differential_branch_number
order
linear_branch_number
order = 4 (best is 4294967295, worst is 1)
differential_branch_number = 2 (best is 32, worst is 0)
linear_branch_number = 2 (best is 32, worst is 0)
ROTATE -16, 32 input bit size, 2 occurrences
differential_branch_number
order
linear_branch_number
order = 2 (best is 4294967295, worst is 1)
differential_branch_number = 2 (best is 32, worst is 0)
linear_branch_number = 2 (best is 32, worst is 0)

Some Test Results

```python
from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model import SatXorDifferentialModel
from claasp.cipher_modules.report import Report
cipher = AESBlockCipher(number_of_rounds=2)
model = SatXorDifferentialModel(cipher)
trail = model.find_lowest_weight_xor_differential_trail()
report = Report(trail)
report.show()
```

```
plaintext
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ 1 _ _ _ _ _ _ _ _ _
 active words positions = [37, 39, 116, 118]
local weight = 0        total weight = 0

intermediate_output_0_37        Input Links : ['xor_0_36']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ 1 _ _ _ _ _ _ _ 1 _ _ _ _ _ _ _ _ _ _ _ 1 1 1 _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 active words positions = [35, 37, 45, 59, 60, 61]
local weight = 0        total weight = 12

cipher_output_1_32        Input Links : ['xor_1_31']
_ _ _ _ _ _ _ _ 1 _ _ 1 _ _ _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ _ 1 1 _ _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 1 1 1 1 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 active words positions = [8, 11, 15, 32, 35, 36, 39, 88, 89, 90, 91, 92, 93, 94, 95]
local weight = 0        total weight = 30
```

Some Test Results

```
sage: from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
....: from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model import SatXorDifferentialModel
....: from claasp.cipher_modules.report import Report
....: cipher = AESBlockCipher(number_of_rounds=2)
....: model = SatXorDifferentialModel(cipher)
....: trail = model.find_lowest_weight_xor_differential_trail()
....: report = Report(trail)
....: report.show(word_size=8)
_ _ _ _ A _ _ _ _ _ _ _ _ _ _ A _    plaintext

_ _ _ _ A A _ A _ _ _ _ _ _ _ _    intermediate_output_0_37

_ A _ _ A _ _ _ _ _ _ _ A _ _ _ _    cipher_output_1_32


total weight = 30.0
```

Some Test Results

```
sage: from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
....: from claasp.cipher_modules.models.sat.sat_models.sat_xor_differential_model import SatXorDiffer
....: entialModel
....: from claasp.cipher_modules.report import Report
....: cipher = AESBlockCipher(number_of_rounds=4)
....: model = SatXorDifferentialModel(cipher)
....: trail = model.find_lowest_weight_xor_differential_trail()
....: report = Report(trail)
....: report.show(show_modadd=True, verbose=True, word_size=8)
plaintext
A A A A A A A A A A A A A A A A         active words positions = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
local weight = 0        total weight = 0

intermediate_output_0_37        Input Links : ['xor_0_36']
_ _ _ A A _ _ _ _ A _ _ _ _ A _         active words positions = [3, 4, 9, 14]
local weight = 0        total weight = 96

intermediate_output_1_36        Input Links : ['xor_1_35']
_ _ _ _ _ A _ _ _ _ _ _ _ _ _ _         active words positions = [5]
local weight = 0        total weight = 120

intermediate_output_2_36        Input Links : ['xor_2_35']
A A A A _ _ _ _ _ _ _ _ _ _ _ _         active words positions = [0, 1, 2, 3]
local weight = 0        total weight = 126

cipher_output_3_32        Input Links : ['xor_3_31']
A _ _ _ _ _ _ A _ _ A _ _ A _ _         active words positions = [0, 7, 10, 13]
local weight = 0        total weight = 150


total weight = 150.0
```

```
sage: report.show(show_modadd=True, verbose=True)
plaintext
1 _ _ 1 _ _ _ _ _ _ 1 1 _ _ _ 1 1 1 1 1 _ _ 1 _ _ 1 _ 1 1 1 1 1 _ _ _ 1 _ _ 1 1 _ 1 _ _ _ _ _ _ 1 _ 1 _ _ 1 1 1 1 _ 1 1 _ _ 1 1 _ 1 1 _ _ _ 1 1 _ _ _ 1 1 1
 1 _ 1 1 1 _ _ 1 _ 1 1 1 1 _ 1 1 _ _ 1 _ _ 1 _ _ 1 1 _ _ 1 _ _ _ 1 1 1 _ _ _ _ _ 1 _ 1 1 _ 1 1 _ _ _    active words positions = [0, 3, 10, 11, 15, 16, 17,
 18, 19, 22, 25, 27, 28, 29, 30, 31, 35, 38, 39, 41, 48, 50, 53, 54, 55, 56, 58, 59, 62, 63, 65, 66, 70, 71, 75, 76, 77, 78, 80, 81, 82, 85, 87, 88, 89, 90
, 92, 93, 96, 99, 102, 103, 106, 110, 111, 112, 118, 120, 121, 123, 124]
local weight = 0        total weight = 0

intermediate_output_0_37        Input Links : ['xor_0_36']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 _ 1 _ _ _ 1 1 1 _ 1 _ _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 1 _
 _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 _ _ _ 1 _ _ 1 _ _ _ _ _ _ _ _    active words positions = [24, 25, 27, 31, 32, 33, 3
5, 38, 39, 74, 75, 76, 79, 112, 116, 119]
local weight = 0        total weight = 96

intermediate_output_1_36        Input Links : ['xor_1_35']
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 1 _ 1 1 _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _    active words positions = [40, 41, 42, 44, 45, 47]
local weight = 0        total weight = 120

intermediate_output_2_36        Input Links : ['xor_2_35']
_ 1 _ 1 1 _ 1 _ _ 1 1 _ 1 1 _ _ _ _ 1 1 _ 1 1 _ _ _ 1 1 _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _    active words positions = [1, 3, 4, 6, 9, 10, 12, 13
, 18, 19, 21, 22, 26, 27, 29, 30]
local weight = 0        total weight = 126

cipher_output_3_32        Input Links : ['xor_3_31']
1 1 _ 1 1 1 _ 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 _ _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
 _ _ _ 1 1 _ _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1 1 _ _ 1 1 _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _    active words positions = [0, 1, 3, 4, 5, 7, 57, 58,
 61, 62, 81, 82, 85, 86, 106, 107, 110, 111]
local weight = 0        total weight = 150


total weight = 150.0
```

Some Test Results

```
sage: from claasp.cipher_modules.statistical_tests.nist_statistical_tests import
....:  NISTStatisticalTests
....: from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
....: from claasp.cipher_modules.report import Report
....:
....: cipher = AESBlockCipher(number_of_rounds=5)
....: test = NISTStatisticalTests(cipher)
....: results = test.nist_statistical_tests('avalanche')
....:
....: report = Report(results)
....: report.show()
      Statistical Testing In Progress.........
```

```
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <1048576>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
 55  43  35  35  34  42  47  32  28  33  0.067989    379/384     Frequency



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 374 for a
sample size = 384 binary sequences.

For further guidelines construct a probability table using the MAPLE program
provided in the addendum section of the documentation.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Some Test Results

```
sage: from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
....: from claasp.cipher_modules.models.milp.milp_models.milp_xor_differential_model import MilpXorDifferentialModel
....: from claasp.cipher_modules.report import Report
....: aes=AESBlockCipher(number_of_rounds=2)
....: milp = MilpXorDifferentialModel(aes)
....: milp_trail = milp.find_lowest_weight_xor_differential_trail(solver_name='SCIP_EXT')
....: print(f"Found a trail of weight {milp_trail['total_weight']}:\n")
....: trail_report = Report(milp_trail)
....: trail_report.show(show_modadd=True, verbose=True)
Writing problem data to 'aes_block_cipher_k128_p128_o128_r2_xor_differential_1719831435.077382.lp'...
907039 lines were written
```

Some Test Results

```
sage: from claasp.ciphers.block_ciphers.speck_block_cipher import SpeckBlockCiph
....: er
....: from claasp.cipher_modules.algebraic_tests import AlgebraicTests
....: from claasp.cipher_modules.report import Report
....:
....: cipher = SpeckBlockCipher(number_of_rounds=4)
....: test = AlgebraicTests(cipher)
....: results = test.algebraic_tests(timeout_in_seconds=5)
....:
....: report = Report(results)
....: report.show()
/usr/local/lib/python3.10/dist-packages/kaleido/scopes/base.py:188: DeprecationWarning:

setDaemon() is deprecated, set the daemon attribute instead

sage: report
<claasp.cipher_modules.report.Report object at 0x77bc66e12a40>
sage: report.save_as_image()
saving image
image saved
Report saved in /home/sage/tii-claasp/test_reports/speck_p32_k64_o32_r4_date:2024-07-01time:12:07:48.0004
71/algebraic_tests
```

Some Test Results

	1	2	3	4
number_of_variables	112	256	400	544
number_of_equations	64	192	320	448
number_of_monomials	157	391	626	860
max_degree_of_equations	2	2	2	2
test_passed	True	True	True	True

Conclusions

Conclusions

PROS	CONS
• Still in development process. • Implementation of the algorithm is easy. • To make all the tests. Only implementation of the algorithm is enough. • Searching best linear and differential trails are easy	• Only uses 1 processor. • Documentation is not correct. • Some functions are not working. • Some tests works fine with basic ciphers like Speck but have problem with working AES • MILP generates too many equations. • Makes bitwise testing not bytewise.

References

- Bellini, E., Gerault, D., Grados, J., Huang, Y. J., Makarim, R., Rachidi, M., & Tiwari, S. (2023, August). CLAASP: a cryptographic library for the automated analysis of symmetric primitives. In *International Conference on Selected Areas in Cryptography* (pp. 387-408). Cham: Springer Nature Switzerland.