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

December 2, 2023

Cryptography Research Center, Technology Innovation Institute, Abu Dhabi, UAE

Contents

Cryptography Research Centre

Introduction

Overview

Cryptography Research Centre

CLAASP is a library whose **goal** is to provide an extensive toolbox gathering state-of-the-art techniques aimed at simplifying the manual tasks of symmetric ciphers' designers and analysts.

`github.com/Crypto-TII/claasp`

- **opensource**, library is built on top of Sagemath
- **extendable**
- **easy-to-use**, especially with its web app
- **generic**
- **automated**

Cipher representation

CLAASP basic idea

Cryptography Research Centre

- How a cipher is represented in CLAASP?
  - a symmetric cipher is a python class represented as a list of "connected components"
  - a component is a python class that refers to the building blocks of ciphers (S-Boxes, XOR, etc.).

- What can be done from this representation?
  1. generate the Python or C code of the encryption function,
  2. execute a wide range of statistical and avalanche tests on the primitive,
  3. automatically generate SAT, SMT, CP and MILP models to find, for example, differential and linear trails,
  4. measure algebraic properties of the cipher,
  5. test neural-based distinguishers.

ToySPN1: CLAASP code and diagram

Cryptography Research Centre

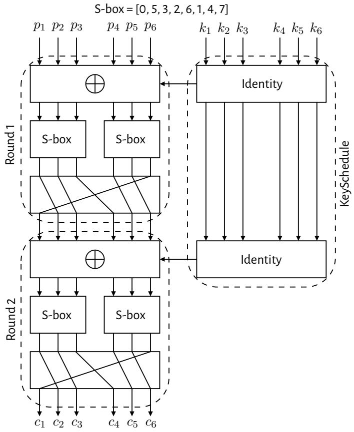
```
from claasp.cipher import Cipher

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)

toyspn = ToySPN()
hex(toyspn.evaluate([0x3F,0x3F]))
```



CLAASP pre-defined ciphers so far

Block Ciphers	Permutations	Hash Functions	Stream Ciphers
AES	ASCON	SHA-1	ChaCha
DES	ChaCha	SHA-2	ZUC
LEA	GIFT-128	MD5	A5/1
LowMC	GIMILI	BLAKE	Bivium
Midori	Grain core	BLAKE2	Trivium
PRESENT	KECCAK-p	Whirlpool	Snow3g
Raiden	PHOTON		BluetoothE0
SIMON	Salsa		
Speck	SPARKLE		
Sparx	Spongent-π		
SKINNY	TinyJAMBU		
TEA - XTEA	Xoodoo		
Twofish			
Threefish			
Kasumi			
LBlock			
QARMA			
RC5			
SCARF			

Evaluator module

From cipher representation to evaluation code

- From a cipher class, we can **automatically generate:**
  - Python and C code to evaluate the cipher
  - a vectorized implementation of the cipher, when the evaluation of millions of inputs are required
  - the class of the inverse of the cipher

- Future works:
  1. A CUDA-based parallel evaluation with GPUs
  2. Optimization of the automatically generated code

Vectorized evaluation example: AES-128

Cryptography Research Centre

Example

```
sage: from claasp.ciphers.block_ciphers.aes_block_cipher import AESBlockCipher
sage: aes = AESBlockCipher()
sage: import numpy as np
sage: from os import urandom

sage: n = 1000
sage: key = np.frombuffer(urandom(n*16), dtype = np.uint8).reshape((-1, n))
sage: plaintext = np.frombuffer(urandom(n*16), dtype = np.uint8).reshape((-1, n))

sage: result = aes.evaluate_vectorized([key, plaintext])
```

Trails search modules

Constraints solvers

- From a given cipher object, CLAASP can automatically generate models for:
  - Differential trails - Truncated differential trails
  - Linear trails
  - Impossible trails
- CLAASP implements the generation of models to find:
  - One optimal trail
  - All trails for which the weight value is within a fixed range
  - Trails for single-key or related-key scenarios
- Trails can be found for ARX, SPN and Feistel ciphers
- The following solvers can be used: MILP (GLPK, Gurobi, CPLEX, GLOP), SAT (Cadical, Cryptominisat, MiniSAT, Kissat, Par-Kissat), SMT (Yices, MathSAT, Z3), CP (Choco, ORTools, MiniZinc)

New results and future works

- New results:
  - Differential trail: We managed to find an optimal differential trail for 10 rounds of Speck128-128 with a probability weight of 49
  - Linear trail: we found a linear trail for 8 rounds of Salsa with a theoretical correlation of 2^{-31} instead of 2^{-34} as described in Coutinho et al. (2022)

- Future works:
  - Differential-linear trails
  - Rotational XOR trails
  - Boomerang trails

Statistical tests module

Multiple tests

We have integrated in CLAASP the following tests:

- NIST STS and Dieharder suites, Rukhin et al. (2001); Bassham et al. (2010)
- Avalanche properties, Daemen et al. (2018)
- Continuous avalanche properties, Coutinho et al. (2020)

- Future release: High-order avalanche tests defined in "ACE-HOT: Accelerating an extreme amount of symmetric Cipher Evaluations for High-Order avalanche Tests" that has been presented in LatinCrypt 2023

Avalanche entropy example

Cryptography Research Centre

Example

```
sage: from claasp.ciphers.block_ciphers.speck_block_cipher import SpeckBlockCipher
sage: speck = SpeckBlockCipher(block_bit_size=8, key_bit_size=16, number_of_rounds=5)
sage: d = speck.diffusion_tests(number_of_samples=1000)
sage: d["test_results"]["plaintext"]["round_output"]
        ["avalanche_entropy_vectors"]["differences"][0]["output_vectors"][2]
[0.99896, 0.96544, 0.83272, 0, 0, 0.99805, 0.7967, 0.99999]
```

We obtained the entropy vector of all output bits of the round 3 due to a difference injected in position 0.

Heatmap for the avalanche entropy criterion

- Cell shading: Greener when entropy, with a 0.01 bias from a single input bit difference, nears 1; otherwise, redder.

	state bit position							
rounds	0							
1	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
2	1.00	0.81	1.00	1.00	0.00	0.00	0.00	0.00
3	1.00	0.96	0.82	0.00	0.00	1.00	0.81	1.00
4	1.00	1.00	0.93	0.83	0.97	1.00	0.87	0.96
5	1.00	1.00	1.00	1.00	1.00	1.00	0.88	0.99

Figure 5.1: Speck: avalanche entropy heatmap - difference injected in position 0

Algebraic module

Current state

- Generate a multivariate polynomial system corresponding to the cipher
- Try to solve this symbolic system by using Gröbner basis

- Future works:
  - Cube attacks: generation of superpolies
  - Division trails search

Cryptography Research Centre

Algebraic Test by Solving the algebraic System

Cryptography Research Centre

Example

```
sage: from claasp.cipher_modules.models.algebraic.algebraic_model import AlgebraicModel
sage: from claasp.cipher_modules.algebraic_tests import algebraic_tests
sage: from claasp.ciphers.toys.toyspn1 import ToySPN1
sage: toyspn1 =ToySPN1()
sage: result = algebraic_tests(toyspn1,120)
sage: result["test_passed"]
[False, False]
```

- If the test fails (returns False), the cipher is not secure against the algebraic attack based on solving its symbolic system by using Gröbner basis. If it returns True, we cannot claim that it is secure.

Neural aided cryptanalysis module

Current state

Cryptography Research Centre

- Built from Bellini et al. (2021), CLAASP provides a test that returns the accuracy of distinguishing a ciphertext coming from an instance of the cipher with a certain key and the output of a random permutation.
- CLAASP implements the neural distinguisher described by Gohr in Gohr (2019). Specifically, the neural distinguisher is trained to label samples $[C_0 = E_K(P_0), C_1 = E_K(P_1)]$ as 0 (if $P_0 \oplus P_1$ is random) or 1 if $P_0 \oplus P_1$ is a given, fixed value δ.
- Finding good differences for Gohr's approach for any cipher: Bellini et al. (2022) accepted to FES 2024

Gohr with CLAASP

Cryptography Research Centre

Example

```
sage: from claasp.ciphers.block_ciphers.speck_block_cipher import SpeckBlockCipher
sage: speck = SpeckBlockCipher()
sage: from claasp.cipher_modules.neural_network_tests import make_resne
sage: from claasp.cipher_modules.neural_network_tests import get_differential_dataset

sage: net = make_resnet(input_size = 64)
sage: X, Y = get_differential_dataset(cipher = speck,
    input_differences = [0x400000, 0], number_of_rounds = 5, samples =10**6)
sage: X_val, Y_val = get_differential_dataset(cipher = speck,
    input_differences = [0x400000, 0], number_of_rounds = 5, samples =10**5)
sage: net.compile(optimizer='adam',loss='mse',metrics=['acc']);
sage: h = net.fit(X, Y, batch_size=5000, validation_data=(X_val, Y_val), epochs = 2)
```

Conclusion

Conclusion

Cryptography Research Centre

- CLAASP gathers a large array of cipher analysis techniques, all in one framework
- CLAASP team is strongly committed to include new state-of-the-art techniques
- Open-source statut is an invitation to researchers to not only use it, but also collaborate

Contacts:

- emanuele.bellini@tii.ae
- juan.grados@tii.ae
- mohamed.rachidi@tii.ae

Demo

References I

Cryptography Research Centre

Bassham, L., Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Leigh, S., Levenson, M., Vangel, M., Heckert, N., and Banks, D. (2010). Special Publication (NIST SP) - 800-22 Rev 1a: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.

Bellini, E., Gerault, D., Hambitzer, A., and Rossi, M. (2022). A cipher-agnostic neural training pipeline with automated finding of good input differences. Cryptology ePrint Archive, Paper 2022/1467. `https://eprint.iacr.org/2022/1467`.

Bellini, E., Hambitzer, A., Protopapa, M., and Rossi, M. (2021). Limitations Of The Use Of Neural Networks In Black Box Cryptanalysis. In *Innovative Security Solutions for Information Technology and Communications: 14th International Conference, SecITC 2021, Virtual Event, November 25–26, 2021, Revised Selected Papers*, page 100–124, Berlin, Heidelberg. Springer-Verlag.

Coutinho, M., de Sousa Júnior, R. T., and Borges, F. (2020). Continuous Diffusion Analysis. *IEEE Access*, 8:123735–123745.

References II

Cryptography Research Centre

Coutinho, M., Passos, I., Vásquez, J. C. G., de Mendonça, F. L. L., de Sousa, R. T., and Borges, F. (2022). Latin dances reloaded: Improved cryptanalysis against salsa and chacha, and the proposal of forró. In Agrawal, S. and Lin, D., editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I*, volume 13791 of *Lecture Notes in Computer Science*, pages 256–286. Springer.

Daemen, J., Hoffert, S., Assche, G. V., and Keer, R. V. (2018). The design of Xoodoo and Xoofff. *IACR Trans. Symmetric Cryptol.*, 2018(4):1–38.

Gohr, A. (2019). Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning. In Boldyreva, A. and Micciancio, D., editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 150–179. Springer.

References III

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, N., Dray, J., and Vo, S. (2001). Special Publication (NIST SP) - 800-22: A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.

Comparison of cryptanalysis libraries features with CLAASP

Cryptography Research Centre

		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	-

Technology
Innovation
Institute

Cryptography
Research
Centre

tii.ae