École polytechnique fédérale de Lausanne

Security and Cryptography Laboratory

January 2024

DEEP LEARNING ∧ DIFFERENTIAL CRYPTANALYSIS

DIFFERENTIAL NEURAL CRYPTANALYSIS

Student: Ana-Maria Indreiaș

Project supervisor: Dr. Ritam Bhaumik

DIFFERENTIAL CRYPTANALYSIS

- Studies difference propagation
- E.g. for plaintext bitwise XOR: $Pr(\delta 1 \rightarrow \delta 2) = Pr(Enc(P) \oplus Enc(P \oplus \delta 1) = \delta 2)$
- Differences can also be inserted in keys, nonces, etc.
- Primitives should not have high-probability differential trails, as these can be used to mount attacks (e.g. key recovery)

DEEP LEARNING

- Subfield of machine learning
- Complex model architectures
- Useful for finding hidden patterns, features, connections
- Recent breakthroughs: LLMs (ChatGPT, etc.), AI art (DALL-E, etc.)

r/ChatGPT, Average Day in France

DEEP LEARNING

- Why should cryptanalysts care? Is deep learning suitable for pseudorandom data?

r/ChatGPT, Average Day in France

DEEP LEARNING

- Why should cryptanalysts care? Is deep learning suitable for pseudorandom data?
- In 2019, someone used a deep learning model ~~to steal the Mona Lisa~~ as a distinguisher for block cipher SPECK, mounting a partial key recovery attack

r/ChatGPT, Average Day in France

A SEMINAL PAPER

- Gohr 2019, or: Neural Distinguishers 101 (ND-101)
- Can use his method to quickly test the robustness of any primitive

Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning

Aron Gohr

Conference paper | First Online: 01 August 2019

3873 Accesses | **73** Citations | **1** Altmetric

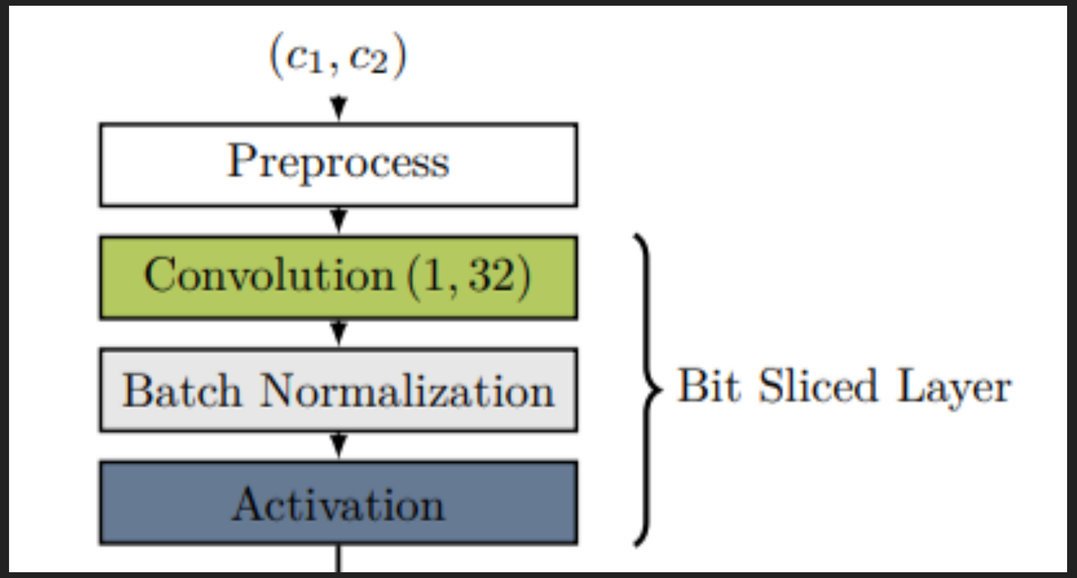
Part of the Lecture Notes in Computer Science book series (LNSC,volume 11693)

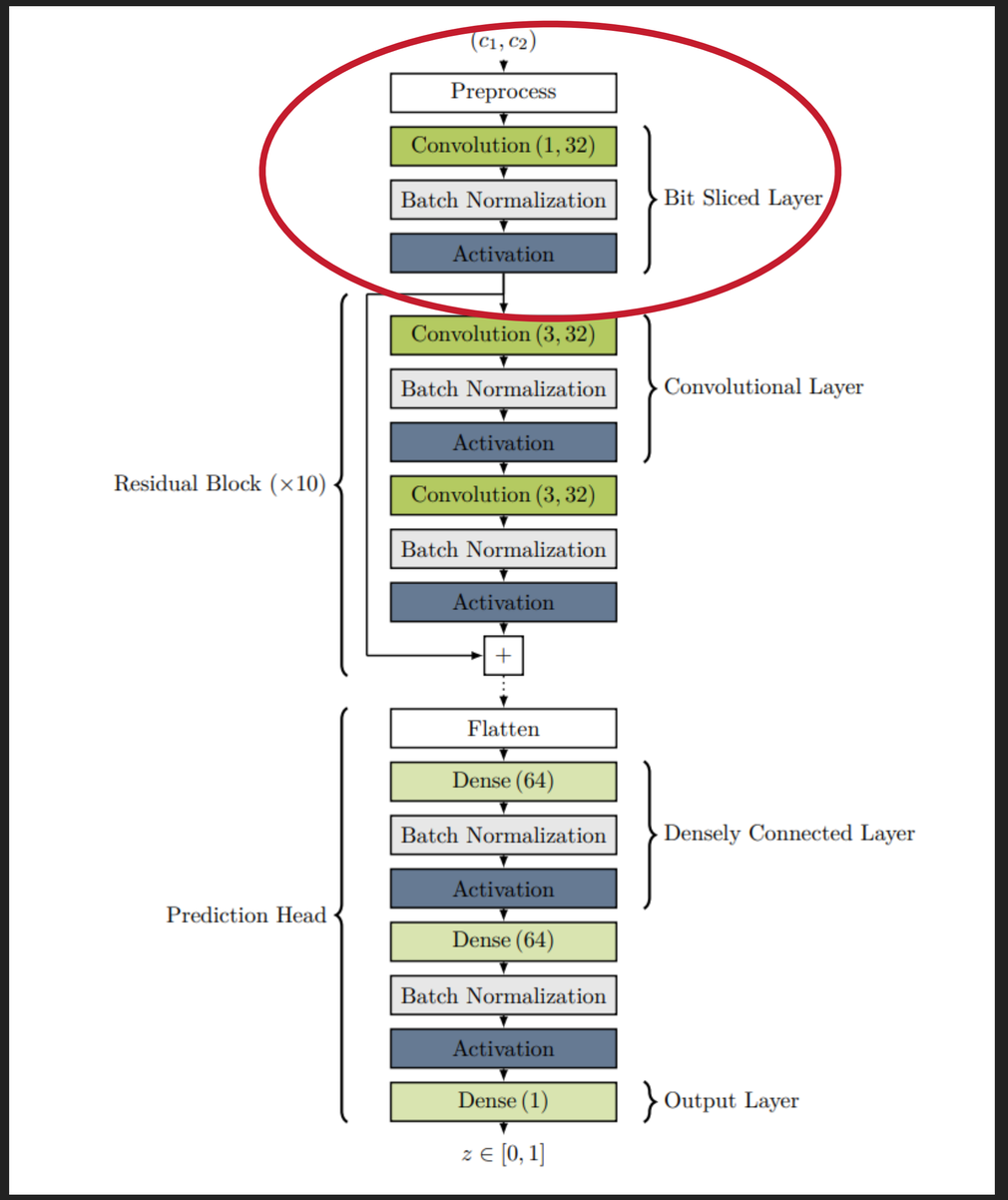
Abstract

This paper has four main contributions. First, we calculate the predicted difference distribution of Speck32/64 with one specific input difference under the Markov assumption completely for up to eight rounds and verify that this yields a globally fairly good model of the difference distribution of Speck32/64. Secondly, we show that contrary to conventional wisdom, machine learning can produce very powerful cryptographic distinguishers: for instance, in a simple low-data, chosen plaintext attack on nine rounds of Speck, we present distinguishers based on deep residual neural networks that achieve a mean key rank roughly five times lower than an analogous classical distinguisher using the full difference distribution table. Thirdly, we

GOHR'S 2019 MODEL

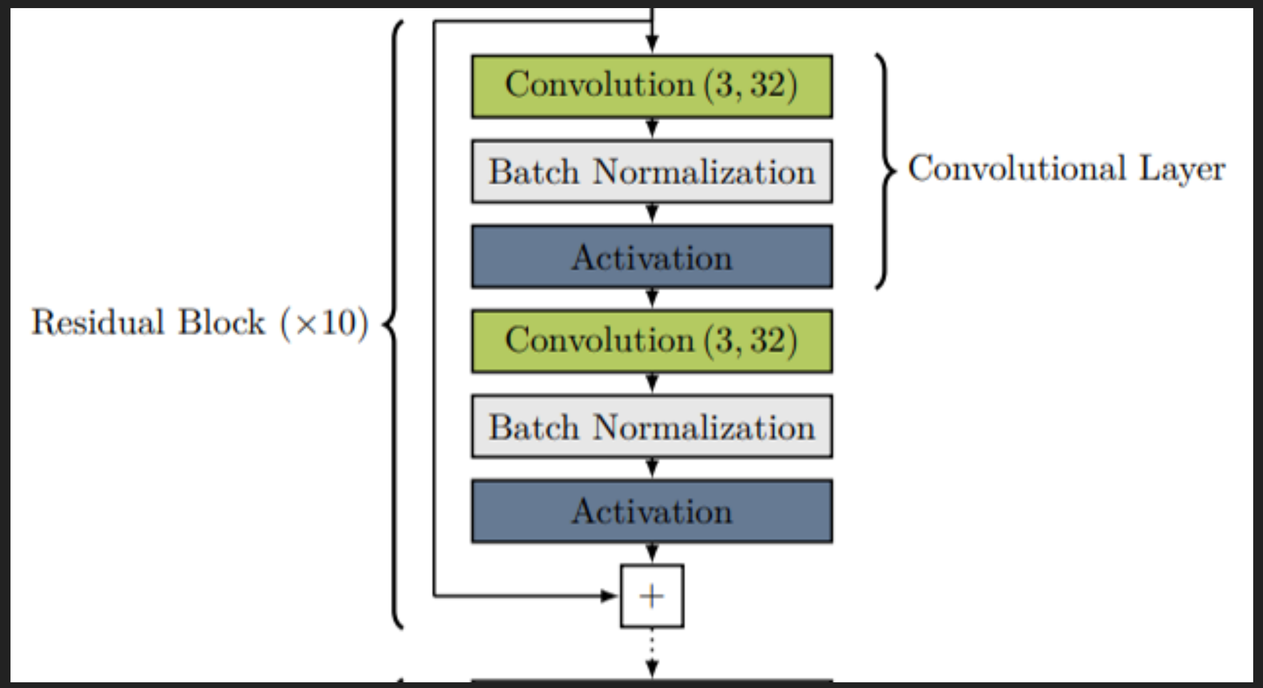
- First Layer: Light Preprocessing

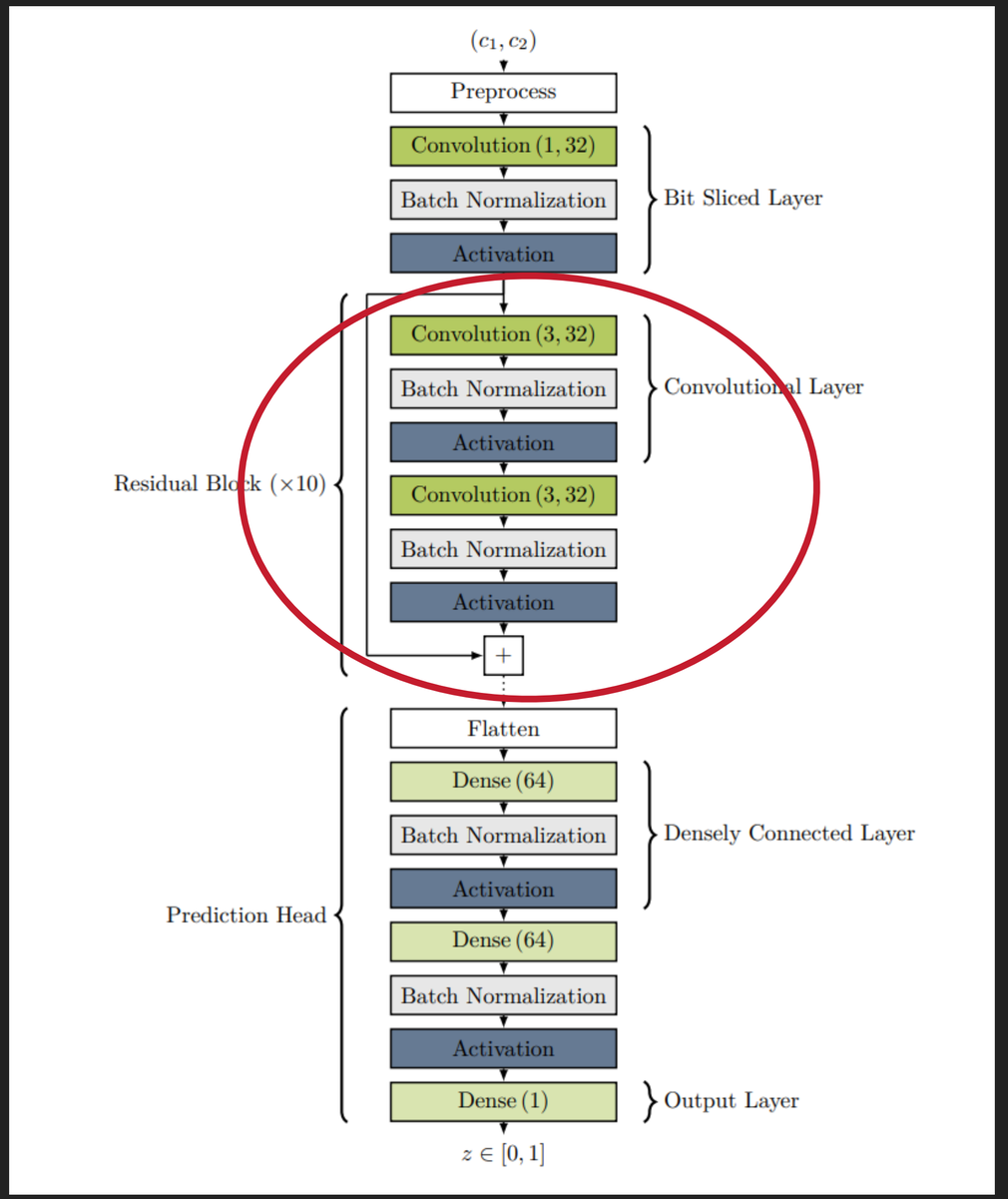




GOHR'S 2019 MODEL

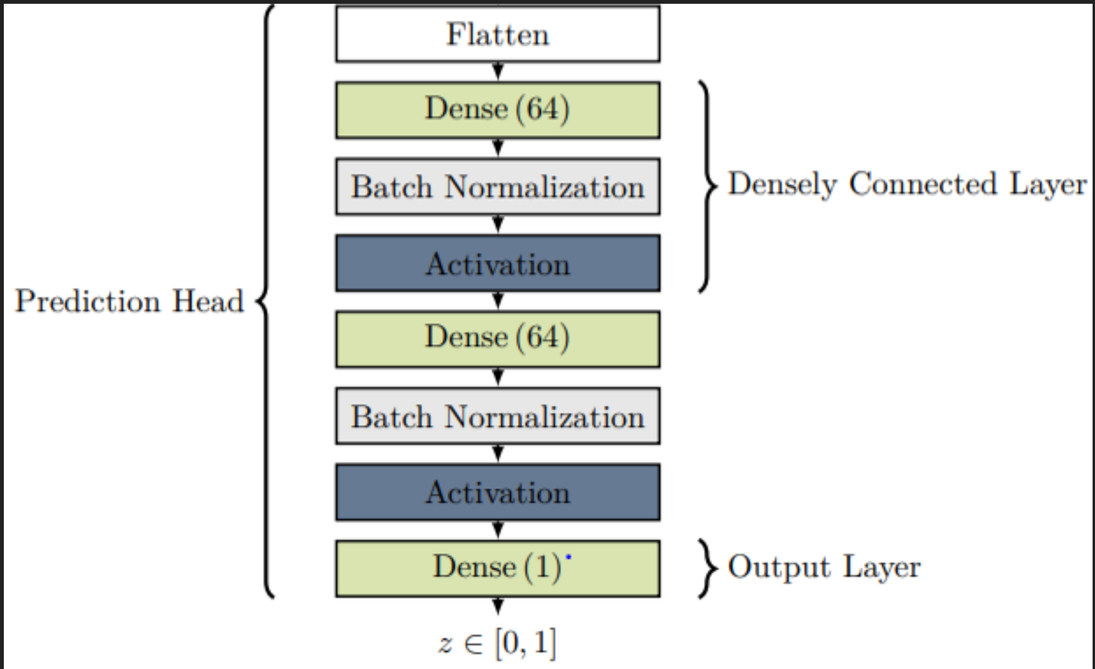
- Middle Layer: Feature Extractor
- 1-10 residual layers

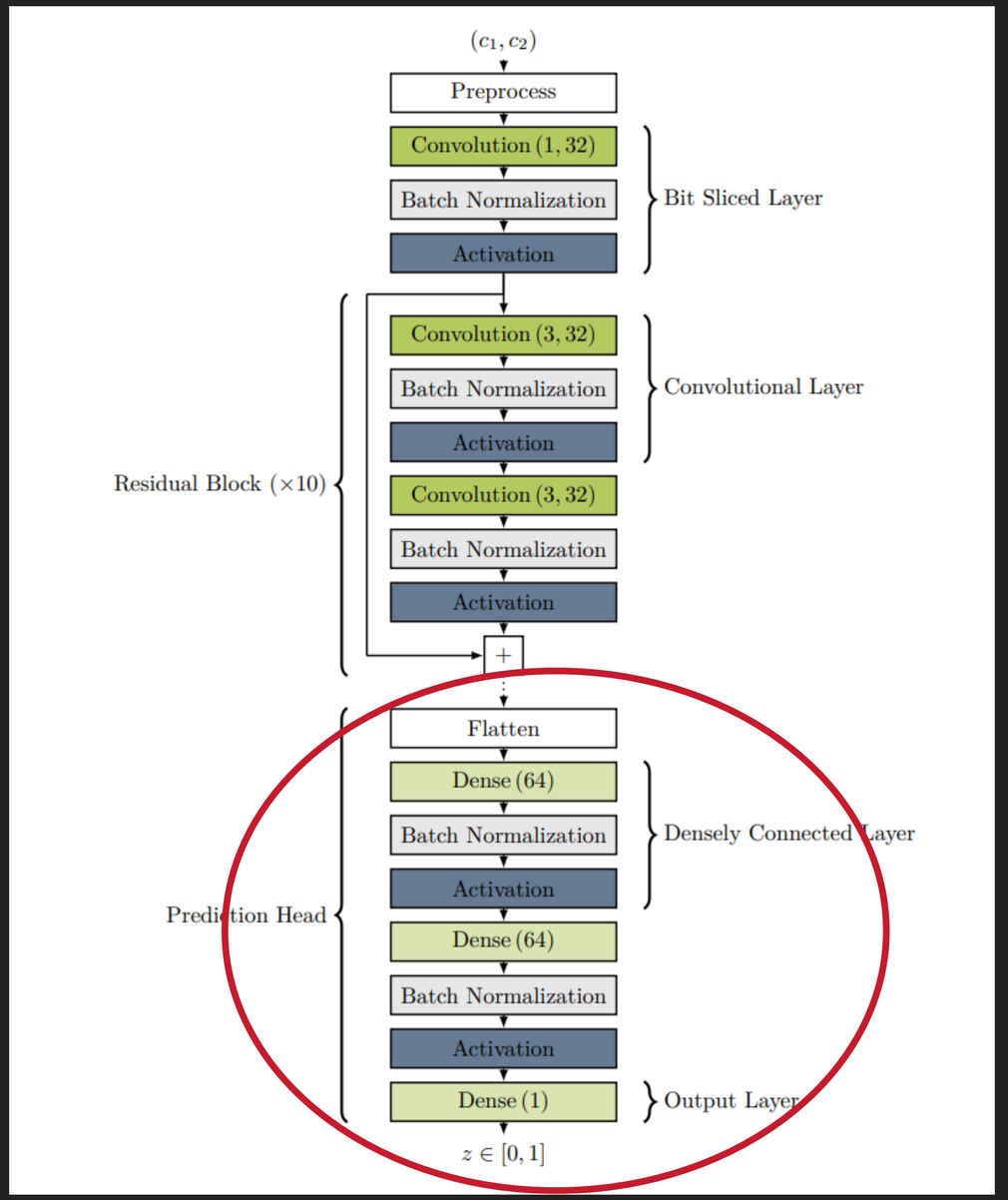




GOHR'S 2019 MODEL

- Last Layer: Prediction Head



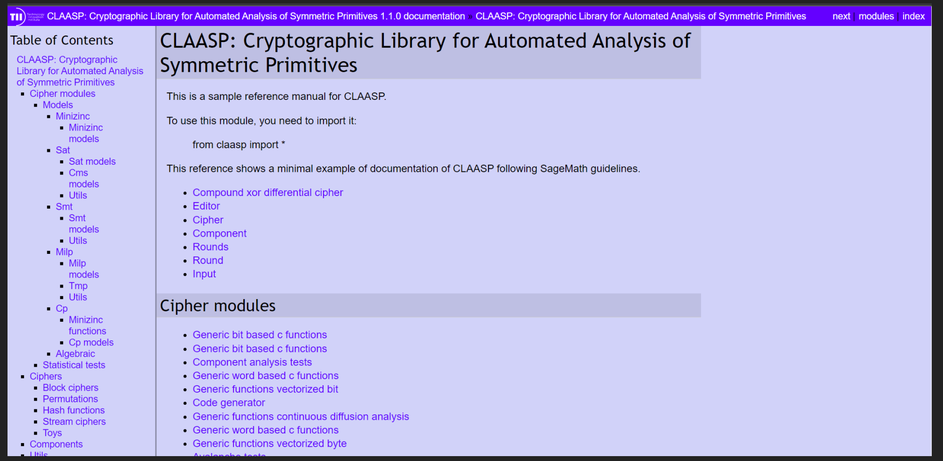


HOW TO DO THIS WITH ANOTHER CIPHER?

- Need a differential dataset
  - => need a good input difference
- Reframed question: How to obtain good input differences?
  - Answer 1: Check literature
  - Answer 2: Use something else
    - => another question: what and how?

FINDING GOOD INPUT DIFFERENCES IF LITERATURE OFFERS NONE

- Option 1: SAT, SMT, MILP Solvers; Constraint Programming
  - unfortunately unfriendly to novices; did not find step-by-step explanations
- Option 2: The CLAASP Library
  - built in Python & Sage
  - classical + neural cryptanalysis tools
  - find input differences specifically for neural networks



https://claasp.readthedocs.io/en/latest/

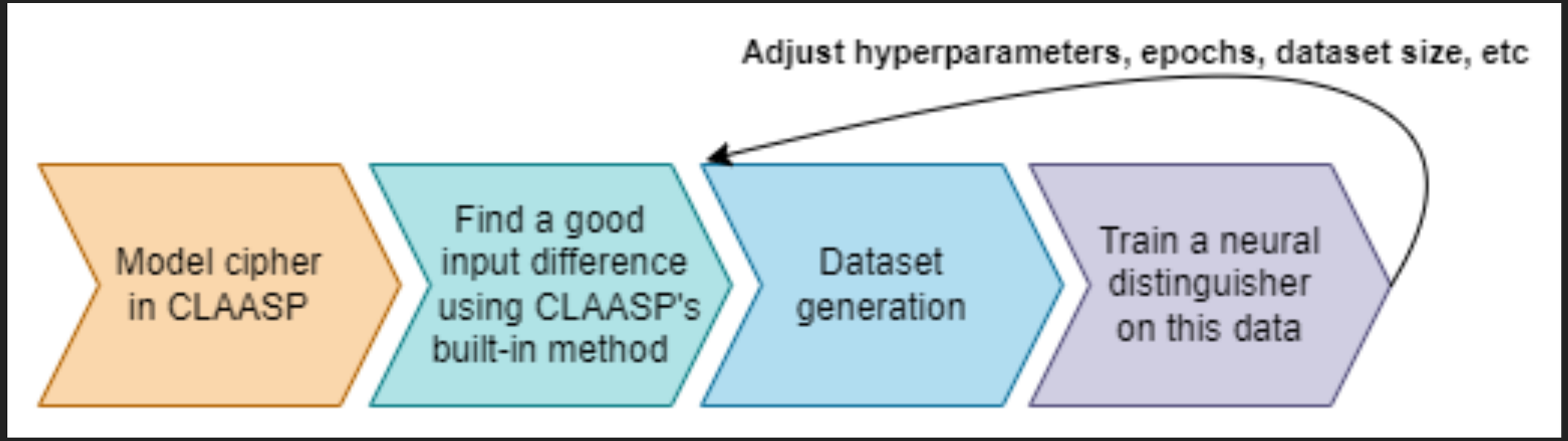
CLAASP MODELLING

```
crax = Cipher("CRAX-S-10", "block cipher", ["plaintext", "key"], [64, 128], 64)

x = ComponentState(["plaintext"], [list(range(32))])
y = ComponentState(["plaintext"], [list(range(32, 64))])

key_0 = ComponentState(["key"], [list(range(32))])
key_1 = ComponentState(["key"], [list(range(32, 64))])
key_2 = ComponentState(["key"], [list(range(64, 96))])
key_3 = ComponentState(["key"], [list(range(96, 128))])
```

CLAASP MODELLING

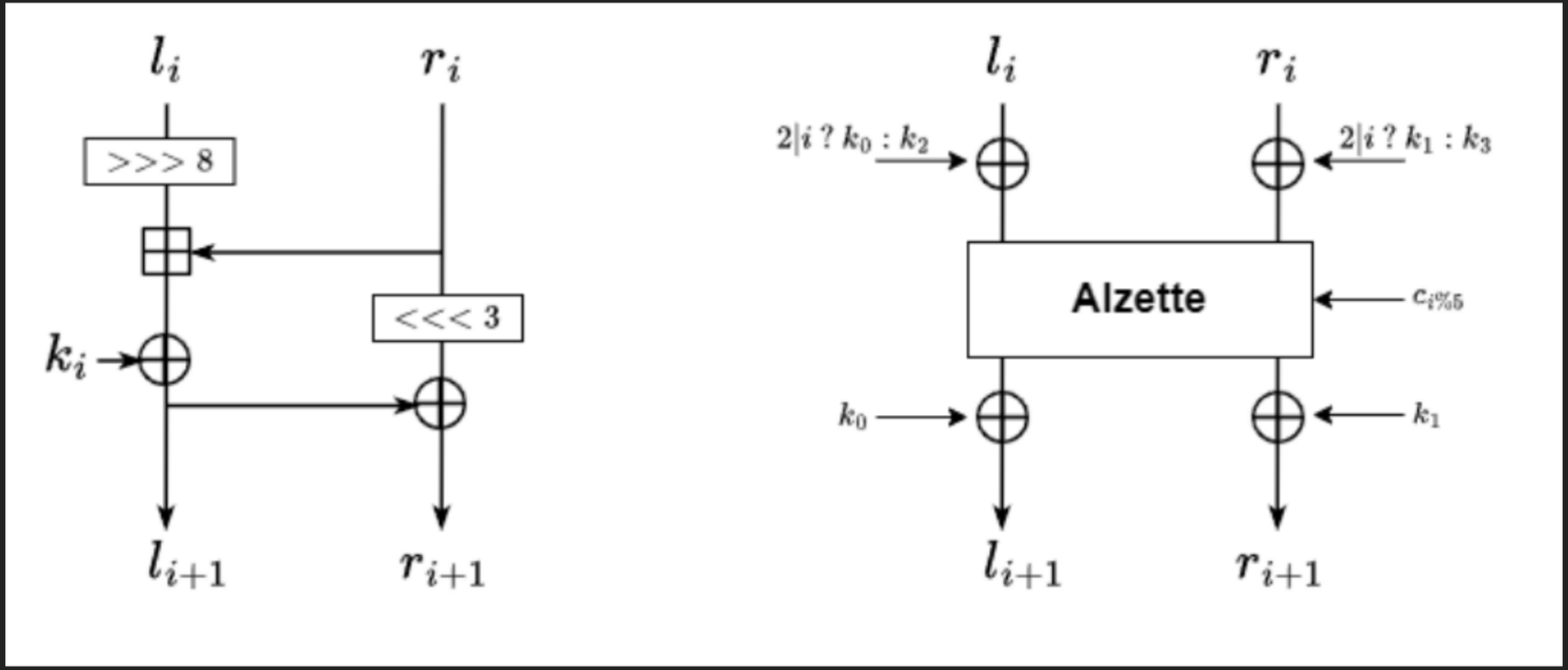
```
crax.add_round()
# x ^= step
crax.add_constant_component(32, step)
step_constant = ComponentState([crax.get_current_component_id()], [list(range(32))])
crax.add_XOR_component(x.id + step_constant.id, x.input_bit_positions + step_constant.input_bit_positions, 32)
x = ComponentState([crax.get_current_component_id()], [list(range(32))])

if (step % 2 == 0):
    # x ^= key_0
    crax.add_XOR_component(x.id + key_0.id, x.input_bit_positions + key_0.input_bit_positions, 32)
    x = ComponentState([crax.get_current_component_id()], [list(range(32))])

    # y ^= key_1
    crax.add_XOR_component(y.id + key_1.id, y.input_bit_positions + key_1.input_bit_positions, 32)
    y = ComponentState([crax.get_current_component_id()], [list(range(32))])
```

ANATOMY OF AN EXPERIMENT



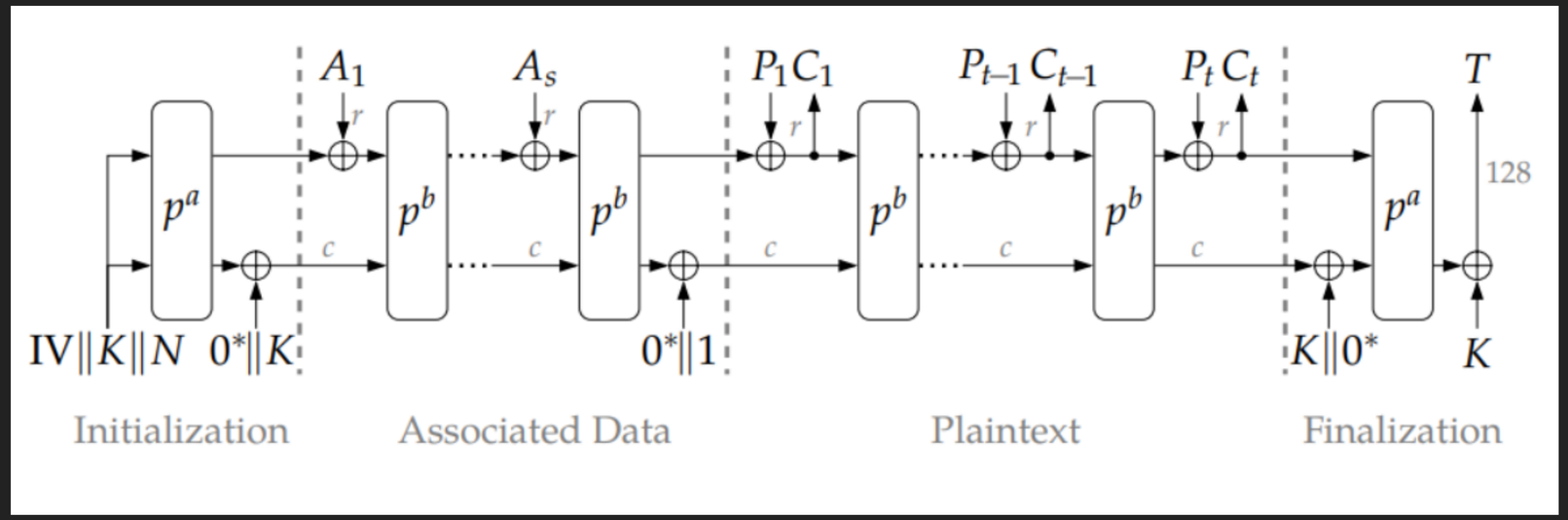
CHOSEN PRIMITIVES

- Block Ciphers: SPECK 64/128 & CRAX-S-10



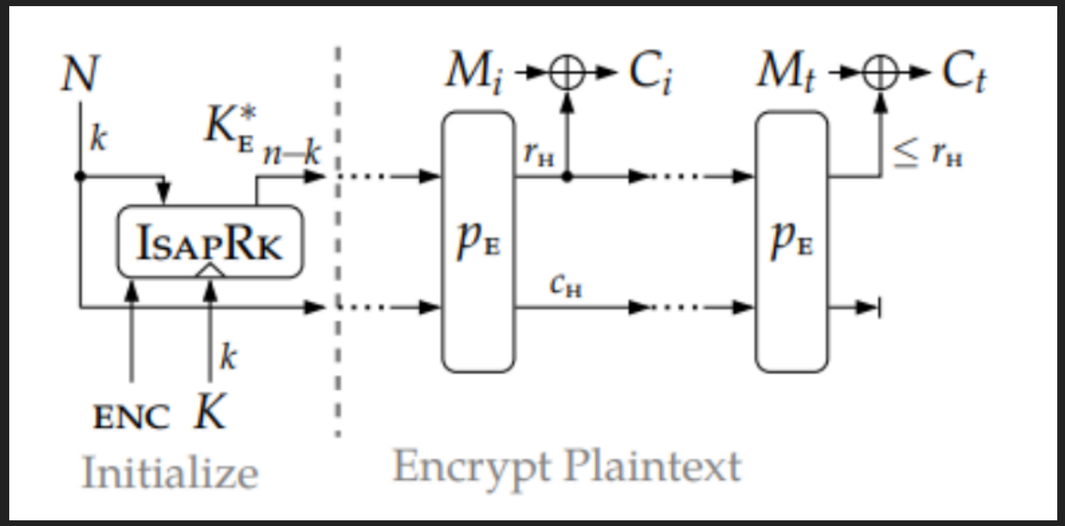
CHOSEN PRIMITIVES

- Block Ciphers: SPECK 64/128 & CRAX-S-10
- AEAD Schemes: ASCON-128-A,



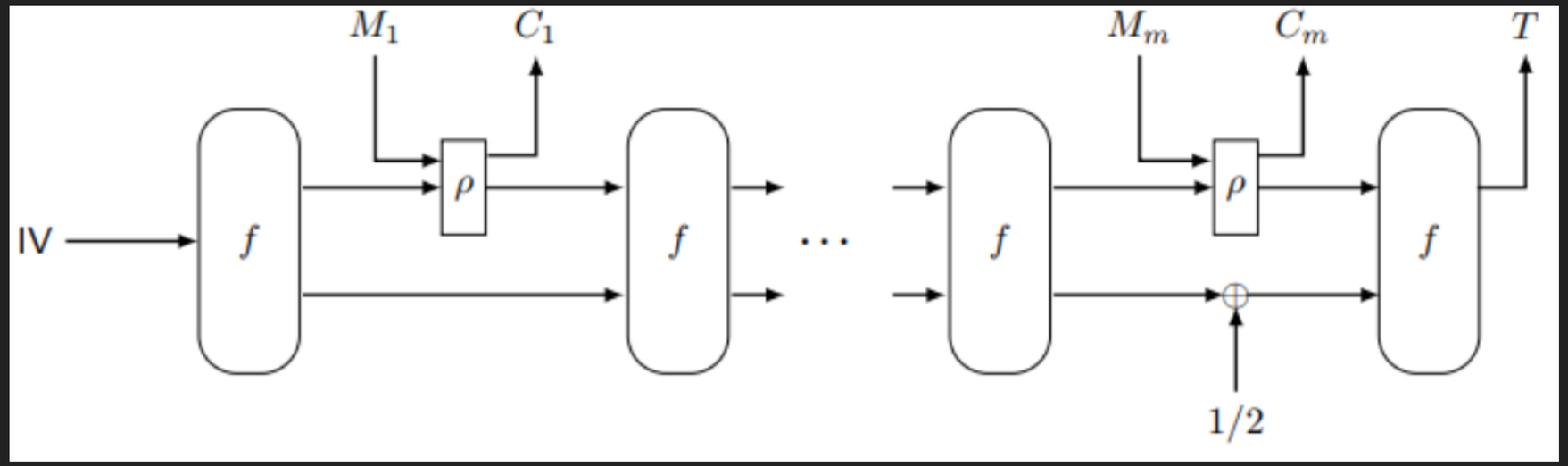
CHOSEN PRIMITIVES

- Block Ciphers: SPECK 64/128 & CRAX-S-10
- AEAD Schemes: ASCON-128-A, ISAP-A-128A,



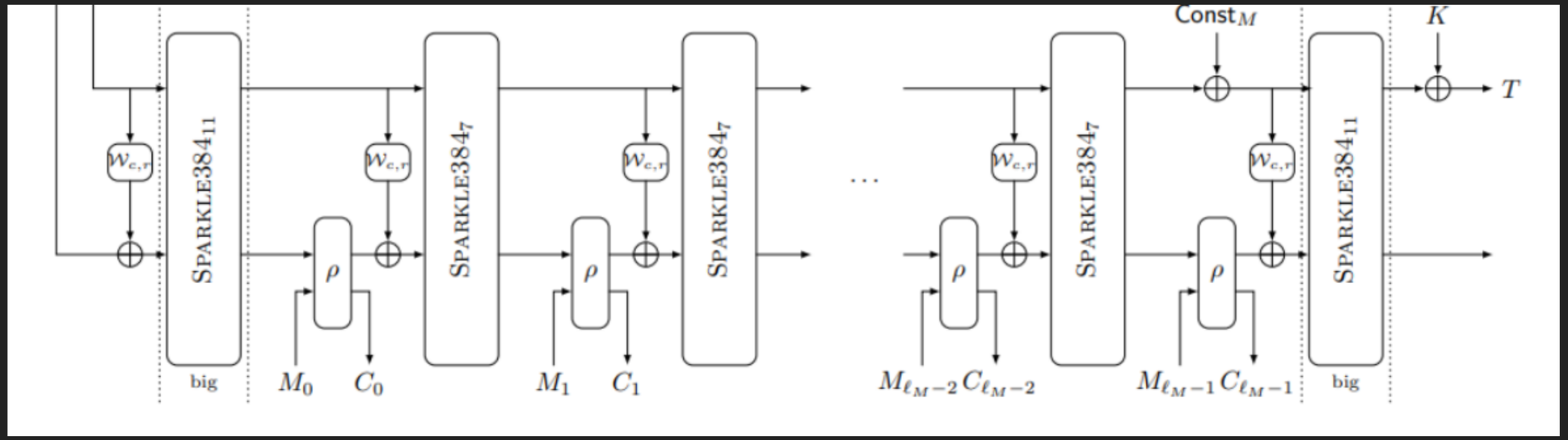
CHOSEN PRIMITIVES

- Block Ciphers: SPECK 64/128 & CRAX-S-10
- AEAD Schemes: ASCON-128-A, ISAP-A-128A, PHOTON-Beetle[128],



CHOSEN PRIMITIVES

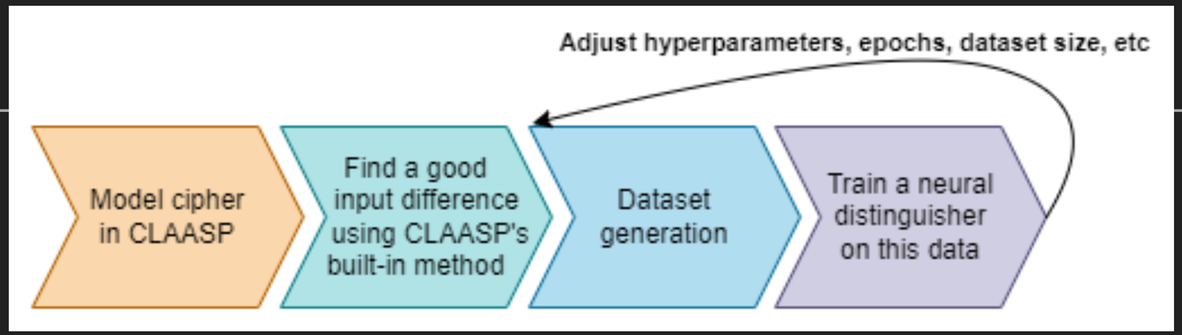
- Block Ciphers: SPECK 64/128 & CRAX-S-10
- AEAD Schemes: ASCON-128-A, ISAP-A-128A, PHOTON-Beetle[128], SCHWAEMM-256-128



CHOSEN PRIMITIVES

- Block Ciphers: SPECK 64/128 & CRAX-S-10
- AEAD Schemes: ASCON-128-A, ISAP-A-128A, PHOTON-Beetle[128], SCHWAEMM-256-128
- Permutations: ASCON(320), PHOTON256, SPARKLE384

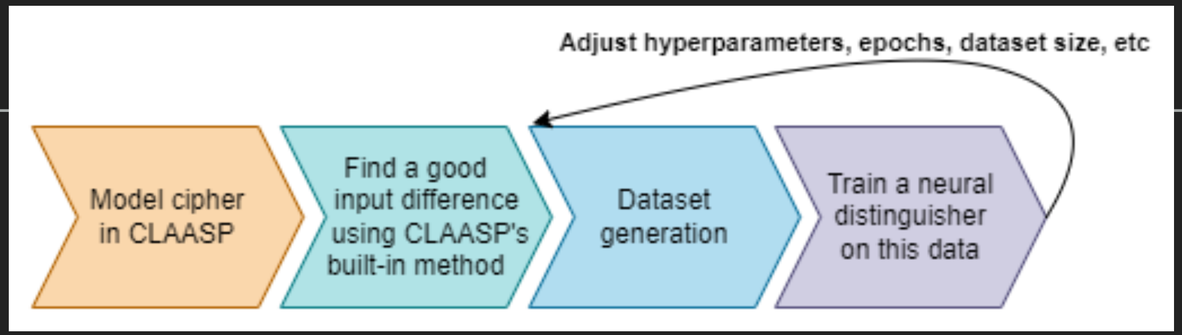
MODELLING STEP



- Created working CLAASP code for:
- CRAX-S-10
- ASCON-128-A, ISAP-A-128A, PHOTON-Beetle[128], SCHWAEMM-256-128

- CLAASP provides the remaining primitives (SPECK & the permutations)

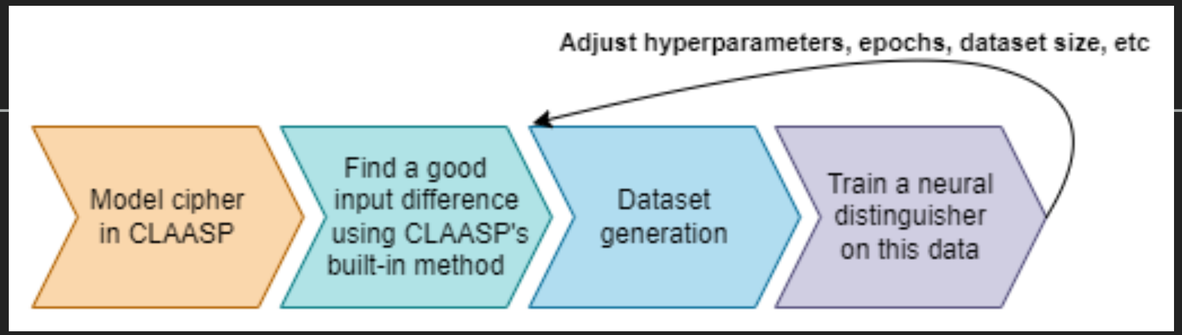
DIFFERENCE STEP



SPECK 64-128	plain text	0x0000008080000000	6 (/27)
CRAX-S-10	plain text	0x4000000000000000	2 (/10)
ASCON-p	plain text	0x20000000000000002000000000000000000000000000000000000000	4 (slim ?, big 12)
PHOTON-p	plain text	0x10000000000000000000000000000000000000	3 (/12)
SPARKLE-p	plain text	0x8000000040000000	3 (slim 7, big 11)
ASCON	nonce	0x400000000000000040000	4*
ISAP	nonce	0x81fa6fb0f40ac5bd140afb3a22d390eb	1*
PHOTON-Beetle	nonce	0xc10cb0e73ccbc12870be6ab8956ee24e	2*
SCHWAEMM	nonce	0xb9bfe5b8fa55f78569a7280a75f9ed661df01f1d25161e4c0bd539ea67bcb8af	1*

- Interesting differences obtained, with the exception of ISAP, PHOTON-Beetle and SCHWAEMM AEAD schemes
- Possible fix: Insert 'fake' rounds in CLAASP models of AEAD schemes

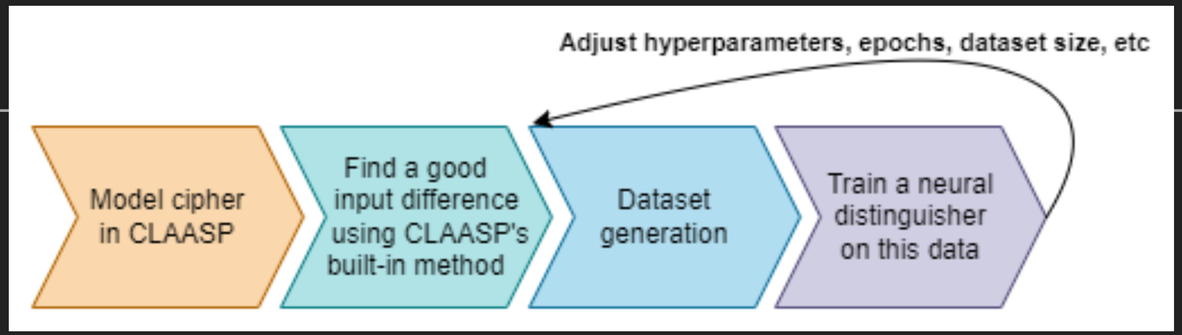
DATASET GENERATION STEP



- Get random bits from /dev/urandom + use CLAASP's vectorized evaluation

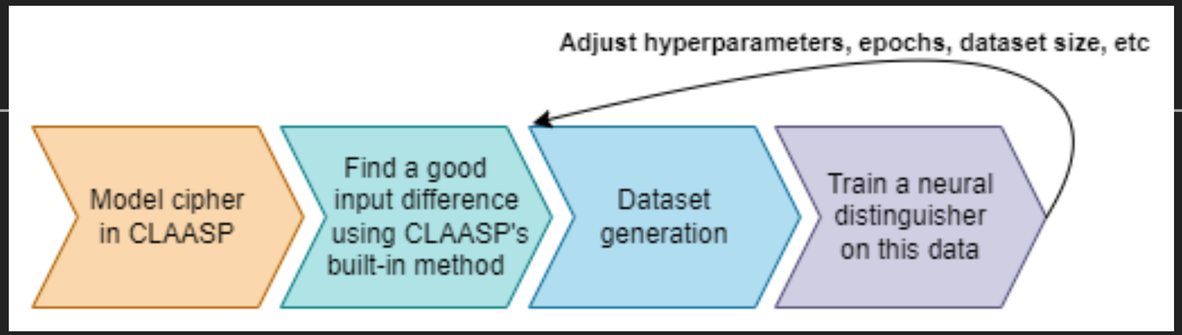
```python
def make_x_y(cipher: Cipher, plaintext_byte_size, key_byte_size, nb_samples, diff):
    pt0, pt1, ks, y = create_random_inputs(plaintext_byte_size, key_byte_size, nb_samples, diff)

    vectorized_plaintext = pt0.transpose()
    vectorized_keys = ks.transpose()

    ct0 = cipher.evaluate_vectorized([vectorized_plaintext, vectorized_keys])[0]
    ct1 = cipher.evaluate_vectorized([pt1.transpose(), vectorized_keys])[0]

    ## ... Now we have to merge ct0, ct1 and transform them to bits ... ##
    ## ... This method is already implemented in neural_network_tests::create_differential_dataset ... ##

    ct0_bits = np.unpackbits(ct0, axis=1)
    ct1_bits = np.unpackbits(ct1, axis=1)

    x = np.hstack([ct0_bits, ct1_bits])

    return x, y
```

TRAINING STEP



- 6- and 7-round neural distinguishers for SPECK64/128
- Only 1-round NDs for CRAX-S-10, permutations
- AEADs: unconvincing results; cannot break them into round-reduced versions, need another approach
  - possible fix: focus on underlying permutations
- Full-round SPECK vs CRAX: both NDs perform similarly, no reason to believe one is weaker than other to neural cryptanalysis

TRAINING TAKEAWAYS



- A good difference for x rounds helps train neural networks for ~x, x+1rounds without "fancy" techniques (e.g. staged training)
- Permutations on large states are more difficult to analyze
- AEADs need a better approach
- Always focus on validation loss and accuracy - if the model overfits, stop training immediately ('manual' early stopping). Manage overfitting by e.g. using more data in larger batches and lowering learning rates

- (Differential) neural cryptanalysis is a promising young field in need of refinement
- Using the presented pipeline is a quick way to get started
- Code + a recommended reading list: github.com/aindreias/cs-498

THANK YOU FOR HAVING LISTENED TO THIS PRESENTATION!

Contact: ana-maria.indreias@epfl.ch

E.1 Full-Round Robustness Comparison

Cipher	Setup	Tr-Acc.	Best Val-Acc.
SPECK64/128	10K; 200 batch; 20 epochs	99.99%	51.8%
CRAX-S-10	10K; 200 batch; 20 epochs	100%	53.1%
SPECK64/128	100K; 2K batch; 20 epochs	85.21%	50.7%
CRAX-S-10	100K; 2K batch; 20 epochs	85.93%	50.88%
SPECK64/128	100K; 25K batch; 20 epochs	70.53%	50.13%
CRAX-S-10	100K; 25K batch; 20 epochs	70.37%	49.9%
SPECK64/128	250K; 50K batch; 20 epochs	63.29%	50.65%
CRAX-S-10	250K; 50K batch; 20 epochs	63.64%	50.39%
CRAX-S-10	250K; 50K batch; 40 epochs	70.19%	50.63%
SPECK64/128	500K; 50K batch; 20 epochs	61.38%	50.37%
CRAX-S-10	500K; 50K batch; 20 epochs	61.55%	50.25%

E.2 Reduced-Round SPECK64/128 results

Rounds	Setup	Tr-Acc.	Best Val-Acc.
6	100K; 25K batch; 20 epochs	84.21%	60.9%
6	500K; 25K batch; 20 epochs	*	68%
6	500K; 5K batch; 20 epochs	*	**82%**
7	500K; 25K batch; 20 epochs	65.11%	53.27%
7	500K; 5K batch; 20 epochs	60.41%	**55.69%**
8	500K; 5K batch; 20 epochs	*	<50%
8	500K; 1K batch; 20 epochs	*	<50%

D ASCON-128 Neural Distinguisher Results

Setup	Training Accuracy	Best Validation Accuracy
250K; 1K batch; 20 epochs	77.09%	50.49%
250K; 25K batch; 20 epochs	69.38%	50.64%
250K; 50K batch; 20 epochs	66.41%	50.63%
300K; 50K batch; 20 epochs	66.09%	50.08%
300K; 60K batch; 20 epochs	64.66%	50.38%
300K; 75K batch; 20 epochs	63.60%	50.389%
350K; 50K batch; 20 epochs	65.14%	50.014%
350K; 70K batch; 20 epochs	63.71%	50.25%
400K; 40K batch; 50 epochs	72.12%	50.44%
400K; 50K batch; 50 epochs	72.73%	**50.70%**
400K; 80K batch; 20 epochs	62.87%	50.48%
400K; 80K batch; 50 epochs	70.20%	**50.66%**
450K; 50K batch; 20 epochs	63.78%	50.2088%
450K; 90K batch; 20 epochs	61.80%	50.38%
500K; 100K batch; 20 epochs	60.93%	50.11%

TRAINING RESULTS IN DETAIL

600K; 75K batch; 20 epochs	61.71%	50.52%
600K; 100K batch; 20 epochs	60.73%	49.96%
600K; 120K batch; 20 epochs	60.29%	50.27%
600K; 120K batch; 30 epochs	62.09%	50.10%
600K; 150K batch; 20 epochs	59.75%	50.43%
600K; 150K batch; 30 epochs	61.84%	49.99%
700K; 35K batch; 30 epochs	65.03%	50.23%
700K; 100K batch; 30 epochs	62.5%	50.23%
700K; 140K batch; 30 epochs	61.57%	50.11%
800K; 20K batch; 20 epochs	62.63%	50.33%
800K; 50K batch; 20 epochs	61.30%	50.29%
800K; 80K batch; 20 epochs	60.50%	50.19%
800K; 80K batch; 50 epochs	65.88%	50.38%
800K; 100K batch; 20 epochs	59.78%	50.22%
800K; 160K batch; 20 epochs	58.67%	50.15%
900K; 90K batch; 20 epochs	60.04%	50.27%

HYPERPARAMETERS

- Default hyperparameters used by CLAASP's make_resnet
    - Adam optimizer with amsgrad enabled
    - Residual depth of 1 (Gohr's models go up to 10)
    - Regularization parameter: 10 ^ (-5)
- Modified for some experiments:
    - Learning rate (e.g. 10 ^ (-4), default 10 ^ (-3))
    - Loss (Binary Cross-Entropy, default MSE)