Attacking Pseudorandomness with Deep Learning

Ana-Maria Indreiaș

School of Computer and Communication Sciences

Semester Project

January 2024

Responsible
Prof. Serge Vaudenay
EPFL / LASEC

Supervisor
Dr. Ritam Bhaumik
EPFL / LASEC

1 Introduction

Evaluating the security of modern cryptographic primitives has been studied for decades, with various cryptanalysis styles emerging as a result. In this report, we present an overview of a new tool that can be used in conjunction: deep learning.

The report is structured as follows: section 2 offers an overview of essential theoretical concepts from two perspectives, namely cryptography and machine learning. Section 3 discusses the relevant literature on neural cryptanalysis, while section 4 describes the primitives of interest to this project. They pave the way for section 4, which discusses our experiments. We wrap up our findings by first thinking of future extensions of our work in section 6, and conclude in section 7. The appendices provide setup instructions in A, a small tutorial in B, AEAD schematics in C, and distinguisher results in D, E.

2 Theoretical Background

2.1 Cryptography and Cryptanalysis

A **block cipher** provides confidentiality. It takes as input an n-bit plaintext block and a k-bit key block. It outputs an n-bit ciphertext block, typically obtained by applying a round function multiple times. Typical attacks on block ciphers include key or plaintext recovery.

Authenticated Encryption with Associated Data (AEAD) provides confidentiality (through encryption) and authenticity (through a tag or message authentication code). It takes as input a plaintext, a nonce, a key, and associated data (AD). The nonce and key have a fixed size imposed by the AEAD scheme, while the plaintext and AD do not. Furthermore, the nonce can only be used once, otherwise the security guarantees are either weakened or may not hold. Typical attacks on AEAD schemes include key, plaintext or internal state recovery, and tag guessing.

A **distinguisher** is used in cryptographic proofs. Its task is to distinguish between two distributions (for example, between ciphertexts encrypted using AES and the random distribution). It is said to have an advantage if it performs better than a random guess. A **neural distinguisher (ND)** is a neural network that performs binary classification. It typically receives as input a pair of two ciphertexts, and has to decide whether they come from related plaintexts (i.e. with a predetermined difference δ).

Differential cryptanalysis studies the propagation of differences in cryp-

tographic primitives (measured with, e.g. bitwise XOR). It is useful for analyzing symmetric-key primitives and relies on the Differential Distribution Table (DDT), which contains, for each possible difference pair (plaintext-difference δ_1, ciphertext-difference δ_2), the probability of it being observed: $Pr(Enc(P) \oplus Enc(P \oplus \delta_1) = \delta_2)$. DDT-based distinguishers use these probabilities to gain an advantage. For a simple block cipher with a small number of rounds, computing the DDT is feasible, while for more complex primitives, the DDT can be approximated, at best. For the latter case, it is still useful to analyze a reduced-round version of the original primitive. This helps, for example, to reduce the complexity of a key-recovery attack. It is also possible to start with differences in e.g. the key or the nonce instead of the plaintext. These settings are called related-key and related-nonce.

2.2 Machine Learning

Machine learning (ML) is a field in computer science which aims to train models for classification and prediction tasks. These models have adjustable weights, which are defined during training on a given dataset in such a way that the loss function is minimized. A loss function measures the distance between the model's prediction and the ground truth, which is also part of the dataset. When a model's inner approximation of the ground truth is too complex, we say it is overfitting. Similarly, when the approximation is too simple, we say the model is underfitting. Both are undesirable and fixing them may require tweaking the model's hyperparameters (e.g. learning rate, batch size). After training, the model's performance is evaluated with accuracy - the percentage of its correct outputs. The simplest ML model is called a **multilayer perceptron (MLP)** .

Deep learning (DL) uses neural networks with complex architectures for tasks such as image recognition and natural language processing. A **neural network (NN)** takes inspiration from life science and includes multiple neuron layers. It is trained on very large datasets and tends to generalizes well. Typically, such deep learning models are good at finding patterns in natural data and are therefore also used as feature extractors. Given the complexity of deep learning models, an important subfield gaining traction is **explainability**, which aims to demystify the black-box behaviour of trained models and find the reasons behind their classification or prediction choices.

Convolution layers were first used in deep learning for efficient processing of image data, and are a good choice for condensing the input into a slimmer feature. **Residual layers** utilize **skip connections**, which essentially enable input signals to skip that layer and go deeper into the network. Residual networks have been successful in tasks such as removing blur or noise from images.

In this report, **Gohr's model** will often be referred to. Its architecture is defined in [Goh19] and consists of three parts (see Fig. 1): a convolution layer with 32 filters, followed by up to 10 residual blocks and, finally, an MLP block. The first two parts are used as feature extractors, while the latter performs binary classification. Gohr's model takes as input the ciphertext pair (C_0, C_1). Its task is to classify it as 'Real' (label 1) or 'Random' (label 0). The 'Real' pair comes from a plaintext pair with a fixed difference δ: $(C_0 = Enc(P_0), C_1 = Enc(P_0 \oplus \delta))$, while the 'Random' pair comes from two independent plaintexts: $(C_0 = Enc(P_0), C_1 = Enc(P_1))$. **Gohr's model** is an example of a **(differential) neural distinguisher** and it performs **(differential) neural cryptanalysis**. **Neural cryptanalysis** is a recent term and broadly refers to the combined use of classical cryptanalysis techniques, such as differential cryptanalysis, with neural networks.

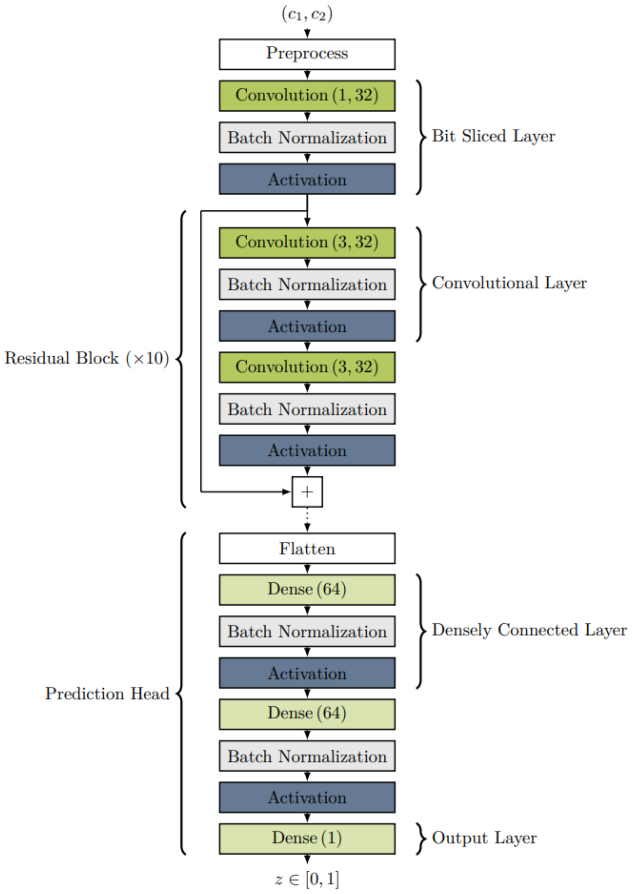


Figure 1: Gohr's 2019 Neural Network, as described in [Goh19]. Figure from [GLN22]

3 Related Work

Recent literature has shown that DL models can be used to approximate the DDT for cryptographic primitives, and even serve as distinguishers aiding in key-recovery attacks.

3.1 "Improving Attacks on Round-Reduced Speck32/64 using Deep Learning" - Gohr, CRYPTO 2019

[Goh19] shows how neural distinguishers can be used to mount a partial key-recovery attack on the round-reduced block cipher SPECK-32/64. To recover the last two round subkeys for 11-round SPECK, the author trains a DL model (see Fig. 1) to distinguish between ciphertext pairs coming from a fixed, good plaintext difference δ obtained from previous literature on SPECK ($C_0 = Enc(P_0), C_1 = Enc(P_0 \oplus \delta)$), and ciphertext pairs coming from independent plaintexts ($C_0 = Enc(P_0), C_1 = Enc(P_1)$).

Gohr obtains good distinguishers for up to 7 rounds of Speck32/64. To confirm whether this DL model uses differential features, he designs **the Real Differences experiment**, in which ciphertext pairs are XORed with a random mask, essentially feeding the model only their difference. The neural distinguisher still performs well with this limited information, leading Gohr to conclude that it must be relying on some form of differential cryptanalysis. The author also proposes a **staged training** approach for building neural distinguishers past 7 rounds. This method consists of retraining a 5-round distinguisher to progressively recognize δ_3, then δ_0, with δ_i being the difference after i rounds, and is successful in obtaining an 8-round distinguisher.

3.2 "A Deeper Look at Machine-Learning Based Cryptanalysis" - Benamira et al. EUROCRYPT 2021

Gohr's result has since motivated a flurry of new articles on NDs and neural cryptanalysis. For instance, [Ben+21] is a comprehensive study of Gohr's model from two perspectives: cryptanalysis and machine learning. The cryptanalysis section contains four main experiments and concludes that Gohr's model uses differential-linear features[1]. These experiments rely on the sets G and B, constructed by running Gohr's 5-round SPECK distinguisher (from now on denoted N5) on 10,000 samples and collecting the samples classified as 'real' with high confidence (prediction values between 0.9 and 1) in set G ('good') and the samples classified as 'random' with high

[1]This claim has been lightly rebuffed by [Bao+23], whose authors note that differential-linear features can be derived from the DDT, whereas Gohr's model already outperforms purely-DDT based distinguishers

confidence (prediction values between 0 and 0.1) in set B ('bad').

Experiment A extracts all ciphertext differences from G. For each difference, 10,000 random pairs are generated and fed to N5. For almost all of the top 1,000 differences, 75% of the corresponding fake samples are labeled 'Real'.

Experiment B decrypts the ciphertext pairs in G for i=1 and i=2 rounds and extracts these differences in set D_{5-i}. Then, for each i, 100,000 samples are created with the initial input difference. The samples are then decrypted for i rounds and discarded if the difference has not been recorded in D_{5-i}. These sanitized samples are fed to N5 as before. For i = 2, approximately 1,600 unique differences are obtained in D_3. 97.86% of the 89,000 sanitized samples are labeled 'Real' (so this experiment isolates around 88% of the true positive ciphertexts, which is very close to N5's True Positive Rate of 90.4%)

The authors observe that the ciphertext differences in G after decrypting for 1 and 2 rounds have some biased bits, and deduce two truncated differentials TD_3 and TD_4. Using constraint programming, they obtain their probabilities: 87% for TD_3 and 49.98% for TD_4. Naturally, experiment C checks whether N5 makes use of these truncated differentials. To do so, they generate 1 million plaintext pairs with the standard difference and encrypt them to 5-i rounds (for i=1;2). Samples that don't correspond to round (5-i) truncated differentials are discarded. The sanitized samples are encrypted for the remaining i rounds and then fed to N5. For i = 2 (i.e. using TD3), the authors obtain 87,000 sanitized samples, 99.2% of them being labeled 'Real'. This captures 87% of the true positive ciphertexts.

Experiment D checks whether a neural network can recognize the presence of truncated differentials. To do so, they generate 10 million plaintext pairs so that about half of them match TD_{5-2} and encrypt them for 2 rounds. Then, they train a new network to distinguish between 2 categories, and obtain an accuracy of 96%. They conclude that Gohr's network architecture is able to recognize samples conforming to the truncated differential at round 3 of SPECK.

The machine learning section aims to replace Gohr's model with a simpler architecture, aiming for better explainability. First, the authors show that Gohr's NN outperforms simpler, non-NN models on the task of distinguishing 5-round SPECK32/64. Then, they show that the prediction block of Gohr's model can be partially swapped with LBGM, an ensemble classifier. They also find that the first convolution layer of Gohr's model performs a trivial transformation of the input: from (L_0, R_0, L_1, R_1) to $(\Delta L = L_0 \oplus L_1, \Delta V =, V_0, V_1)$ and linear combinations of it. Finally, the

authors estimate Gohr's model to have around 10^5 floating parameters, and thus, conclude that it cannot store a full DDT. Since the model uses differential features (as outlined in [Goh19] and doubly verified in [Ben+21]), the authors conjecture that the model builds some compressed approximation of the DDT. Based on these observations, they build a new, more explainable pipeline with similarly performing distinguishers for 5- and 6-round SPECK32/64 (less than 1% difference in accuracy)

3.3 "An Assessment of Differential-Neural Distinguishers" - Gohr, Leander, Neumann. 2022

In 2022, Gohr, Neumann and Leander co-publish an overview of neural distinguishers [GLN22], with new experiments and tips for improving their performance. The authors evaluate the performance of Gohr's model for more cipher families: Feistel-like (SIMON, SPECK), AES-inspired (SKINNY), SPN (PRESENT), as well as a stream cipher (CHACHA) and an LFSR cipher (KATAN). They notice a link between model accuracy and differential-linear biases. Their models for CHACHA and SKINNY achieve 90%+ accuracy, while the ciphers themselves exhibit "many high absolute biases" [GLN22]. This result suggests that neural distinguishers have high accuracy on significantly skewed distributions, so the authors conjecture it may be impossible to build neural distinguishers for full-round ciphers.

On the machine learning side, the authors list the 7-8 most impactful hyperparameters on distinguisher accuracy: "batch size, the use of circular convolutions, the number of neurons in the densely connected layers, the number of filters, the filter size, the [learning rate (LR)] (while using the cyclic LR schedule), the number of residual blocks, the L2 penalty"[GLN22]. They also run the Real Differences experiment outlined in [Goh19], finding that for SIMON, PRESENT and SKINNY the model appears to use purely differential features. Finally, the authors experiment with "toy" versions of SIMON and SPECK, in the context of finding the best input differences. They observe that the input differences that maximize the differential probability are not necessarily the same as the input differences that maximize the accuracy of a neural distinguisher[2] .

3.4 "More Insight on Deep Learning-aided Cryptanalysis" - Bao, Lu, Yao, Zhang. ASIACRYPT 2023

Using set theory and mechanical analysis, [Bao+23] shows that Gohr's ND for 5-round SPECK32/64 performs well by exploiting Inter-XOR information $(L_0 \oplus L_1; R_0 \oplus R_1)$ with either Intra-XOR $(L_i \oplus R_i)$ or Cross-XOR

[2]This finding has been confirmed by several papers discussed in our report: [Ben+21] [GLN22] [Bel+22] [Bao+23]

$(L_0 \oplus R_1; L_1 \oplus R_0)$[3]. Using the result of Experiment A from [Ben+21], where 75% of artificially-constructed samples with a 'good' difference are classified as 'Real', the authors conjecture that Gohr's model must be looking for bi-bit patterns. To verify this hypothesis, they study the differential properties of modular addition, which is the last operation in SPECK's round function, and develop bi-bit constraints. Using these, they build two theoretical distinguishers which outperform simple DDT-based distinguishers.

The authors then briefly discuss performance improvement techniques for NDs. They introduce the freezing layer method, which starts with a previously-trained ND and retrains only its classifier part (the final MLP block), keeping the feature extraction part intact. They obtain results similar to Gohr's staged training. Lastly, the authors study the related-key setting, in which differences are added to the keys instead of the plaintexts. In doing so, the hope is that these differences will appear a few rounds later in the cipher's state, thus extending potential distinguishing attacks. Indeed, using this setting, the authors observe better performance for all proposed distinguishers (both classical and neural).

3.5 DBitNet and CLAASP - Bellini et al., 2022-2023

In 2022 and 2023, Bellini et al. publish several papers on the automation of neural cryptanalysis. Their first article [Bel+22] introduces DBitNet, a generic ND. The goal is to provide a basic model that is agnostic to a cipher's internal architecture. To achieve this and require as little manual modifications as possible, it uses dilated convolutions, which, unlike regular convolutions, help the model make connections between non-neighbouring input bits. Moreover, DBitNet uses the ADAM optimizer together with the AMSgrad algorithm, which 'Reddi et al. introduce [...] in "On the Convergence of Adam and Beyond" [RKK18]' [Bel+22]. This setting helps find a good learning rate while trying to avoid a possible convergence failure for ADAM. DBitNet is complemented by a method for finding input differences. The authors assume that the conclusion of Experiments C and D from [Ben+21] generalizes to all ciphers, and thus, that neural distinguishers perform well when there are biased difference bits at higher rounds. They decide to use an evolutionary search algorithm, which starts from an initial random population, probabilistically mutates it, and promotes the candidates leading to the highest difference biases. While the authors do not claim optimality, this method is useful for quickly finding a reasonable input difference for neural distinguishers.

Following DBitNet, the authors also introduce CLAASP [Bel+23], a Python

[3] $L_i; R_i$ are the left and right parts of ciphertext C_i

library for analyzing cryptographic primitives, with support for neural cryptanalysis as well. For this project, we choose to use CLAASP, as it provides a straightforward way to obtain and use input differences from a given cipher. We have briefly looked into articles explaining how to use constraint programming, SMT, SAT or MILP solvers, however found nothing truly novice-friendly[4]. Furthermore, these solvers promise to find good input differences from a "classical" perspective, which do not necessarily lead to the highest performance in neural distinguishers, cf. [Ben+21] [GLN22] [Bel+22] [Bao+23]. Finally, CLAASP provides interesting analysis tools (classical randomness tests) as object methods. The only requirement is to model the cryptographic primitive of interest using CLAASP's building blocks, which is fairly straightforward. A tutorial is provided in Appendix B.

4 Studied Constructions

In this section we present a quick overview of the block ciphers SPECK64-128, CRAX-S-10, the permutations ASCON, SPARKLE_{384} and PHOTON_{256}, and the AEAD schemes ASCON-128, ISAP-A-128A, PHOTON-Beetle[128] and SCHWAEMM-256-128, since they appear in our experiments. It should be noted that all constructions are lightweight.

4.1 Block Ciphers

SPECK64-128 has a 64-bit input and a 128-bit key. The input is split into two 32-bit parts, on which a round function (see Fig. 2) is applied 27 times. The cipher equally uses the round function for its key schedule. The full specification can be found in [Bea+13].

CRAX-S-10 has the same input and key sizes as SPECK64-128. It has 10 rounds and no key schedule. Its round function (see Fig. 2) makes use of the ARX-box Alzette and depends on the current round number i. More details on CRAX-S-10 are provided in [Bei+20].

4.2 Permutations

ASCON is a 320-bit permutation with a variable number of rounds, 12 being the default. Its state is separated into five 64-bit chunks, which can be viewed as a 5x64 matrix. Each round consists of 3 phases (see Fig. 3): constant addition, a substitution layer which uses a 5-bit S-Box, and a linear layer using a 64-bit diffusion function. Interestingly, ASCON applies its

[4]The interested reader may refer to [Del+21] and [SWW21] for overviews of the existing solvers. Constraint programming in particular is well explained in [GMS17] and [Sun+17], both co-authored by D. Gerault, who went on to work on the CLAASP library

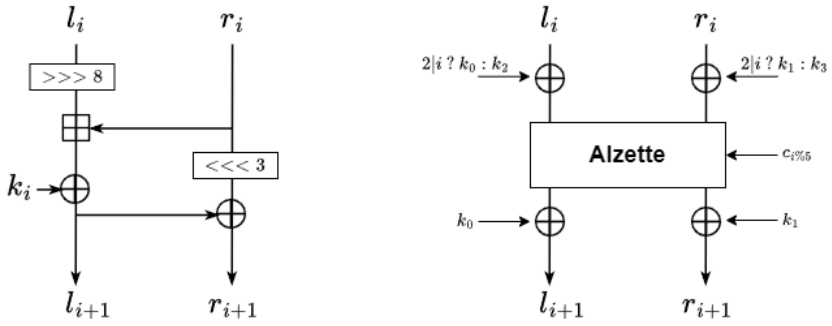


Figure 2: Round functions of SPECK64-128 (left) and CRAX-S-10 (right)

S-Box column-wise. [Dob+16] contains its in-depth specification.

PHOTON$_{256}$ has 12 rounds. The 256-bit state is treated as an 8x8 matrix with 4-bit elements, called *cells*. PHOTON$_{256}$ rounds have 4 steps (see Fig. 4) : constant addition, a substitution layer using a 4-bit S-Box, a shift layer which re-arranges cells within their respective rows, and finally, a linear mix layer, which uses serial matrix multiplication, as described in [Bao+19].

SPARKLE$_{384}$ is SPN-based and has a variable number of steps. Like CRAX-S-10, it makes use of the Alzette construction. Fig. 5 provides SPARKLE pseudocode. Interested readers can consult [Bei+19] for additional information on SPARKLE and Alzette.

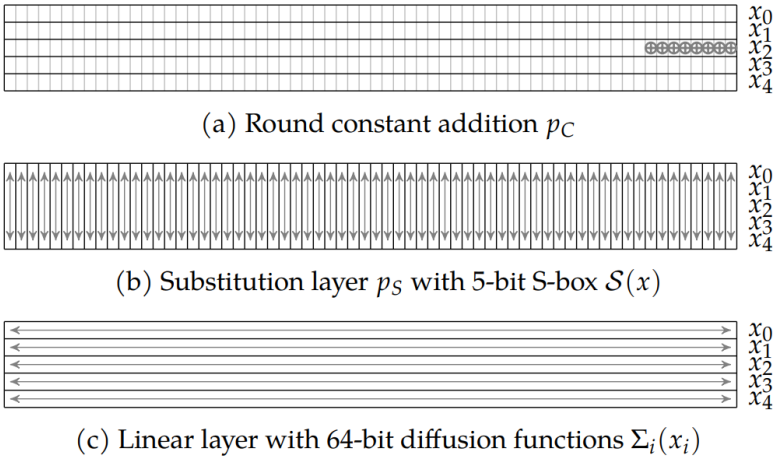


Figure 3: Steps of one ASCON round. Figure taken from [Dob+16]

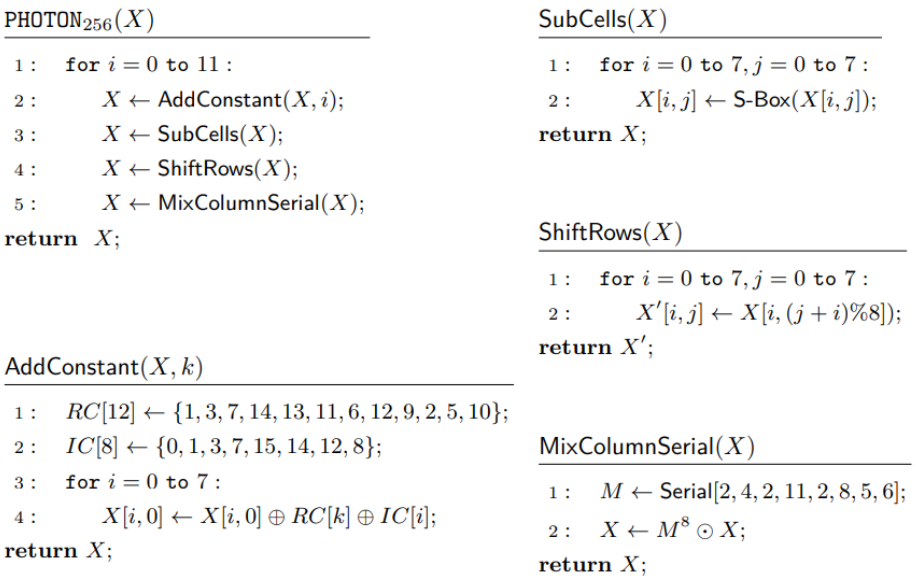


Figure 4: The pseudocode of PHOTON_{256}, taken from [Bao+19]

Algorithm 2.2 SPARKLE384_{n_s}

In/Out: $\big((x_0, y_0), ..., (x_5, y_5)\big), x_i, y_i \in \mathbb{F}_2^{32}$

$(c_0, c_1) \leftarrow$ (0xB7E15162,0xBF715880)
$(c_2, c_3) \leftarrow$ (0x38B4DA56,0x324E7738)
$(c_4, c_5) \leftarrow$ (0xBB1185EB,0x4F7C7B57)
$(c_6, c_7) \leftarrow$ (0xCFBFA1C8,0xC2B3293D)
for all $s \in [0, n_s - 1]$ **do**
  $y_0 \leftarrow y_0 \oplus c_{(s \bmod 8)}$
  $y_1 \leftarrow y_1 \oplus (s \bmod 2^{32})$
  for all $i \in [0, 5]$ **do**
    $(x_i, y_i) \leftarrow A_{c_i}(x_i, y_i)$
  end for
  $\big((x_0, y_0), ..., (x_5, y_5)\big) \leftarrow \mathcal{L}_6\big((x_0, y_0), ..., (x_5, y_5)\big)$
end for
return $\big((x_0, y_0), ..., (x_5, y_5)\big)$

Figure 5: The pseudocode of SPARKLE_{384}, taken from [Bei+19]

4.3 AEAD Schemes

ASCON-128 [Dob+16], **ISAP-A-128A** [Dob+20], **PHOTON-Beetle[128]** [Bao+19] and **SCHWAEMM-256-128** [Bei+19] are sponge-based AEAD schemes, and rely on permutations to modify their internal states. The first two use ASCON, while PHOTON-Beetle[128] uses PHOTON_{256}, and the latter - SPARKLE_{384}. Appendix C contains an illustration of their encryption functions.

5 Experiments

5.1 Overview

We want to test the robustness of various cryptographic primitives against neural cryptanalysis. Since the methods outlined in section 3 are mostly meant for block ciphers, we pick SPECK64-128, for its connection to [Goh19], and CRAX-S-10, as it has the same input, output, and key sizes as SPECK64-128, and its authors describe it as an academical alternative (speed-wise) to SPECK. We also pick four finalists of the NIST Lightweight Cryptography Competition, namely ASCON, ISAP, SCHWAEMM and PHOTON-Beetle - these are AEAD schemes and will be treated differently than the block ciphers. Lastly, since the CLAASP library provides implementations of the ASCON, SPARKLE and PHOTON permutations, we decide to test them as well.

5.2 Experimental Setup

All experiments are performed on an Ubuntu-20.04 instance hosted on WSL2, with access to approximately 100GB of storage and 15GB of RAM. The host device has an AMD Ryzen 5 4600H processor with Radeon Graphics 3.00 GHz. The setup can handle models training up to and around 1 million samples, but it cannot handle Gohr's most complex pipeline which uses 10 million training samples with 1 million validation samples [Goh19]. In the latter case, training processes have been force-stopped by the operating system.

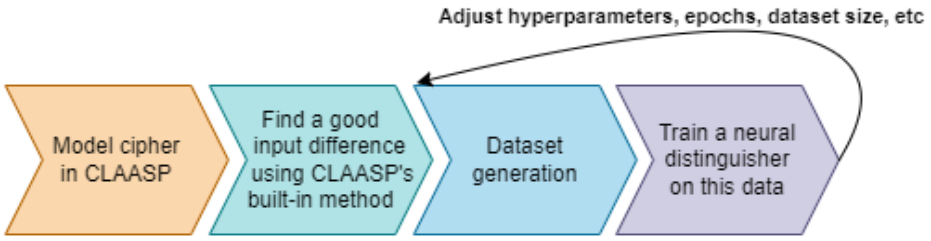


Figure 6: The basic steps of our experiments

The basic methodology is illustrated in Fig. 6, and remains the same for all primitives. The first step is to model them in the CLAASP library, using official test vectors to check for correctness. In case of discrepancies, we compare the authors' code with ours [5]. Thanks to this, we discover some interesting constant manipulations (mostly shifts) which are not mentioned in the official specifications. This is the case for PHOTON-Beetle

[5] Zip files containing the official code and test vectors of NIST finalists can be found at `https://csrc.nist.gov/projects/lightweight-cryptography/finalists`

and SCHWAEMM. Interested readers will find commented parts in our code showcasing these differences.

After modelling, we use the CLAASP method `find_good_input_differences_for_neural_distinguisher`, which uses the evolutionary search algorithm of [Bel+22], to obtain a suitable difference δ. The next step is data generation: to quickly obtain a dataset, we make use of numpy vectorization, which is provided by CLAASP through the method `evaluate_vectorized`. The data has the following structure for plaintext difference δ: a binary label y, indicating whether the sample x is real ($x = (Enc(P_0), Enc(P_0 \oplus \delta))$ or random ($x = (Enc(P_0), Enc(P_1))$. The idea remains the same for nonce differences. Following the example of [Goh19], we get pseudo-random bits from *urandom*, using `numpy.frombuffer`.

After data generation, we use Gohr's model provided in CLAASP as `make_resnet` and train it to distinguish between ciphertext pairs that come from a fixed plaintext (resp. nonce) difference ('Real') or from independent plaintexts (resp. nonces). Both training and validation data are generated on the fly, with validation samples amounting to 10% of the training samples. The decision to generate fresh data each time is due to the nature of the experiments. We have tweaked multiple hyperparameters, especially for ASCON-128 - had we re-used the same data, we might have accidentally 'overfitted' the model.

5.3 Studying AEAD Schemes

We analyze the following principal submissions of NIST LWC finalists: ASCON-128, ISAP-A-128A, PHOTON-Beetle[128], and SCHWAEMM-256-128. Since they are sponge constructions, we cannot simply use plaintext differences: the first plaintext blocks of such an encryption will have the same difference as the first ciphertext blocks. For this reason, and the fact that nonces differ for each encryption, we decide to use the related-nonce setting. To find nonce differences, we use initial population sizes of 8 and 12 for ISAP-A-128A, due to computational limitations. For the others, we use larger initial population sizes, of 16 and 32 respectively. In all cases, the algorithm is run for 15 generations. We have also tried running the method for more generations (up to 500), however the score-wise gain is negligible. This observation equally holds for the permutations and block ciphers. The table on the next page below shows our results.

For ISAP and SCHWAEMM, the CLAASP method did not find good differences. We have tried re-running the search with a larger initial population and more rounds, without success. An idea to remedy this and help differences reach further into the AEAD structure may be to insert "fake" rounds and round outputs into the CLAASP modelling code, as the search method

Scheme	Difference
ASCON	`0x40000000000000040000`
ISAP-A-128A	`0x81fa6fb0f40ac5bd140afb3a22d390eb`
PHOTON-Beetle[128]	`0xc10cb0e73ccbc12870be6ab8956ee24e`
SCHWAEMM-256-128	`0xb9bfe5b8fa55f78569a7280a75f9ed6` `61df01f1d25161e4c0bd539ea67bcb8af`

makes use of these round outputs.

After obtaining the nonce differences, we focus our attention on building successful neural distinguishers for ASCON. We notice the models overfitting easily, with the validation loss function always increasing. To stop this, we first increase the dataset sizes and the relative batch sizes (e.g. 50,000 samples per batch for a 500,000 training set) - these results are presented in Appendix D. Further modifications include decreasing the learning rate and using a combination of early stopping with more data - basically, generating 3-5 datasets and iteratively training the model for 5-10 epochs on each dataset. All of these changes, especially the former, have helped models slow down their learning, and we notice that the validation losses decrease (before, unfortunately, plateauing and then increasing again).

We train similarly-modified distinguishers for the remaining three AEAD schemes, and obtain insignificant advantages. Some improvements that could be tried as a continuation of this work would be working in the related-key scenario cf. [Bao+23], or modifying the sample structure to have more than 2 ciphertexts, as suggested in [GLN22]. Finally, we want to emphasize something that has proved crucial for this experiment: it is not necessary to wait for a model to reach 90+% training accuracy. To get results quickly, it is better to observe the evolution of the validation loss: even if the training accuracy increases, it is no use if the validation loss plateaus or increases consistently. It is a sign that the model is, at worst, overfitting or, at best, not generalizing. In such cases, manually stopping the training process early saves time.

5.4 Studying Block Ciphers

The CLAASP library already provides a SPECK64/128 model, thus we only code CRAX-10. We obtain input differences and a basic Gohr model using the same method described in 5.2. For SPECK, the best found difference reaches round 6 (22.22% of the total rounds), while for CRAX-S-10, the best difference reaches round 2 (20% of the total rounds).
We are able to comfortably train 6-, 7- and 8-round neural distinguishers for SPECK using up to 500,000 training samples and without using staged

Block Cipher	Round Reached	Difference
SPECK64/128	6 (out of 27)	0x0000008080000000
CRAX-S-10	2 (out of 10)	0x4000000000000000

training [Goh19] or the freezing layer method [Bao+23]. On the other hand, we do not obtain similar results for CRAX, for which we can only successfully train a 1-round distinguisher. This is surprising since CRAX is a lightweight block cipher with the same input size as SPECK64/128. We believe that the problem may lie in the (hyper)parameters of the neural network - perhaps tweaking the filter size will lead to non-trivial CRAX distinguishers.

We also train neural distinguishers for full-round SPECK and CRAX, which perform close to the random guess. This behaviour is expected, with some authors even suspecting that it is impossible to build full-round distinguishers for block ciphers [GLN22]. A summary of the results in this experiment can be found in Appendix E.

5.5 Studying Permutations

We analyze the following permutations: 12-round ASCON, 12-round PHOTON_{256}, and 7-round SPARKLE_{384}, which are already part of the CLAASP library. The differences we obtain reach up to round 4, 3, 3, respectively:

Permutation	Round Reached	Difference
ASCON	4 (out of 12)	0x20000000000000002000000000000000000000000000000000
PHOTON_{258}	3 (out of 12)	0x10000000000000000000000000000000000
SPARKLE_{384}	3 (out of 7)	0x8000000040000000

Despite this, our attempts to train neural distinguishers on even 1-3 rounds of these permutations have been unsuccessful, with validation accuracy no better than a random guess. We have tried increasing the number of residual layers from 1 to 5, to account for the bigger input sizes (5-6x bigger than for SPECK64/128), which had a negligible effect.

6 Future Work

One idea for the continuation of the work done during this semester project is the development of better training techniques for neural distinguishers of AEAD schemes. So far, literature has focused on training and improving neural networks for block ciphers. The main idea has been to reduce the rounds of the block cipher, train a successful distinguisher, then slowly re-

train it while increasing the number of rounds. Such an approach has no equivalent for the AEAD schemes studied in this report, since they do not rely on rounds, but rather on permutations being applied to the internal state, with information being modified between these applications. A possible approach could be studying the underlying permutations first (since they are round-based), and trying to combine this with cryptanalysis methods for duplex schemes. Another research direction for the neural cryptanalysis of AEAD schemes would be the search of differences for variable-length input. For this project, we have modeled AEAD schemes in CLAASP that only handle plaintext, nonce, and key sizes that are multiples of the scheme's block size. It would be interesting to see if these schemes weaken in the absence of such constraints.

More generally, Gohr has proposed some directions of improvement for neural cryptanalysis during the GHTC 2023 round table [Gan+23]. For instance, one could investigate different neural network architectures, or even attempt teaching cryptanalysis techniques to Large Language Models. The former is more concrete and, following a discussion with a machine learning researcher, it seems promising to investigate natural language processing concepts, such as self attention. On the other hand, the latter seems challenging from a research perspective, and may prove a suitable subject for a Master's Thesis.

7 Conclusion

In this project, we have studied the robustness of selected cryptographic primitives against basic differential neural cryptanalysis. To do this, we build and tune neural networks, training them to distinguish between ciphertext pairs derived from two related plaintexts (with a predefined difference), or from a random input pair. To compute good plaintext differences for neural distinguishers, we model these constructions in CLAASP, a recent cryptanalysis library [Bel+23]. All code and related files are available on GitHub[6]. During model training, we observe a lack of generalization, which we partially solve by manually stopping the training early and reducing the learning rate. We obtain interesting results for some reduced versions of the chosen constructions, which we believe can serve as building blocks for better distinguishers covering more rounds, and, ultimately, for mounting successful attacks such as key recovery for block ciphers, or universal forgery for AEAD schemes.

[6] `https://github.com/aindreias/cs-498`

References

[Bea+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. "The SIMON and SPECK families of lightweight block ciphers". In: *cryptology eprint archive* (2013).

[Dob+16] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. "Ascon v1. 2". In: *Submission to the CAESAR Competition* 5.6 (2016), p. 7.

[GMS17] David Gerault, Marine Minier, and Christine Solnon. "Using Constraint Programming to solve a Cryptanalytic Problem". In: *IJCAI 2017 - International Joint Conference on Artificial Intelligence (Sister Conference Best Paper Track)*. Melbourne, Australia, Aug. 2017, pp. 4844–4848. URL: `https://hal.science/hal-01528272`.

[Sun+17] Siwei Sun, David Gerault, Pascal Lafourcade, Qianqian Yang, Yosuke Todo, Kexin Qiao, and Lei Hu. "Analysis of AES, SKINNY, and Others with Constraint Programming". In: *IACR Transactions on Symmetric Cryptology* 2017.1 (Sept. 2017), pp. 281–306. URL: `https://hal.science/hal-01615487`.

[RKK18] Sashank J. Reddi, Satyen Kale, and Surinder Kumar. "On the Convergence of Adam and Beyond". In: *ArXiv* abs/1904.09237 (2018). URL: `https://api.semanticscholar.org/CorpusID:3455897`.

[Bao+19] Zhenzhen Bao, Avik Chakraborti, Nilanjan Datta, Jian Guo, Mridul Nandi, Thomas Peyrin, and Kan Yasuda. "PHOTON-beetle authenticated encryption and hash family". In: *NIST Lightweight Compet. Round* 1 (2019), p. 115.

[Bei+19] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, Qingju Wang, and Alex Biryukov. "Schwaemm and esch: lightweight authenticated encryption and hashing using the sparkle permutation family". In: *NIST round* 2 (2019).

[Goh19] Aron Gohr. *Improving Attacks on Round-Reduced Speck32/64 using Deep Learning*. Cryptology ePrint Archive, Paper 2019/037. `https://eprint.iacr.org/2019/037`. 2019. URL: `https://eprint.iacr.org/2019/037`.

[Bei+20] Christof Beierle, Alex Biryukov, Luan Cardoso dos Santos, Johann Großschädl, Léo Perrin, Aleksei Udovenko, Vesselin Velichkov, and Qingju Wang. "Alzette: A 64-Bit ARX-box: (Feat. CRAX and TRAX)". In: *Advances in Cryptology–CRYPTO 2020: 40th*

Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40. Springer. 2020, pp. 419–448.

[Dob+20] CE Dobraunig, Maria Eichlseder, Stefan Mangard, Florian Mendel, BJM Mennink, Robert Primas, and Thomas Unterluggauer. "Isap v2. 0". In: (2020).

[Ben+21] Adrien Benamira, David Gerault, Thomas Peyrin, and Quan Quan Tan. "A deeper look at machine learning-based cryptanalysis". In: *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40*. Springer. 2021, pp. 805–835.

[Del+21] Stéphanie Delaune, Patrick Derbez, Paul Huynh, Marine Minier, Victor Mollimard, and Charles Prud'Homme. "Efficient Methods to Search for Best Differential Characteristics on SKINNY". In: *ACNS 2021 - 19th International Conference on Applied Cryptography and Network Security*. Ed. by Kazue Sako and Nils Ole Tippenhauer. Vol. 12727. 19th International Conference on Applied Cryptography and Network Security. Kamakura, Japan, June 2021, pp. 184–207. DOI: 10.1007/978-3-030-78375-4_8. URL: https://hal.science/hal-03040548.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. "Accelerating the Search of Differential and Linear Characteristics with the SAT Method". In: *IACR Transactions on Symmetric Cryptology* 2021.1 (Mar. 2021), pp. 269–315. DOI: 10.46586/tosc.v2021.i1.269-315. URL: https://tosc.iacr.org/index.php/ToSC/article/view/8840.

[Bel+22] Emanuele Bellini, David Gerault, Anna Hambitzer, and Matteo Rossi. "A Cipher-Agnostic Neural Training Pipeline with Automated Finding of Good Input Differences". In: *Cryptology ePrint Archive* (2022).

[GLN22] Aron Gohr, Gregor Leander, and Patrick Neumann. "An assessment of differential-neural distinguishers". In: *Cryptology ePrint Archive* (2022).

[Bao+23] Zhenzhen Bao, Jinyu Lu, Yiran Yao, and Liu Zhang. "More Insight on Deep Learning-aided Cryptanalysis". In: *Cryptology ePrint Archive* (2023).

[Bel+23] Emanuele Bellini, David Gerault, Juan Grados, Yun Ju Huang, Mohamed Rachidi, Sharwan Tiwari, and Rusydi H Makarim. "Claasp: a cryptographic library for the automated analysis of symmetric primitives". In: *Cryptology ePrint Archive* (2023).

[Gan+23] Fatemeh Ganji, Aron Gohr, Kristin Lauter, Stjepan Picek, and Melissa Rossi [panelists]. *Crypto meets AI: AI and cryptanalysis [Round Table]*. Technology Innovation Institute. Glowing Hot Topics in Cryptography. 2023. URL: https://ghtcworkshop.tii.ae/2023/#program-2.

A Running the provided code

The code is available at `https://github.com/aindreias/cs-498`. It is meant to be inserted into the CLAASP library. This setup requires a UNIX environment. Given the instability of CLAASP configuration scripts (if done incorrectly, the PATH variable becomes empty and thus almost no commands are valid), it is highly recommended to do the setup on a fresh Virtual Machine / WSL instance with enough space.
The concrete steps are as follows:
1. Download Python3, using e.g. `https://www.python.org/downloads/`
2. Download Sage, using instructions from e.g. `https://doc.sagemath.org/html/en/installation/index.html`
3. Download a zipped CLAASP folder from the official repository `https://github.com/Crypto-TII/claasp`. For full compatibility with our code, you can download the zip release with tag 1.1 (see `https://github.com/Crypto-TII/claasp/tags`)
4. Follow CLAASP setup instructions: `https://github.com/Crypto-TII/claasp/blob/main/docs/USER_GUIDE.md`
5. Insert the provided Python code in the `/claasp-main` folder. To load or run a file, open a Sage terminal in `/claasp-main` and use the `attach` command.

B Small CLAASP tutorial

Let us model a toy cipher in CLAASP. Imagine a block cipher with a 16-bit plaintext, a 32-bit key, and a 16-bit output. The plaintext is first XOR-ed with the first 16 bits of the key, and then with the last 16 bits of the key.

The corresponding code would be:

```
from claasp.cipher import Cipher
from claasp.DTOs.component_state import ComponentState
from claasp.utils.utils import get_inputs_parameter

toyCipher = new Cipher("My Broken Cipher", "Block Cipher", ["plaintext",
"key"], [16, 32], 16)
key1 = ComponentState(["key"], [list(range(16))])
key2 = ComponentState(["key"], [list(range(16, 32))])
pt = ComponentState(["plaintext"], [list(range(16))])
ids, pos = get_inputs_parameter([pt, key1])
toyCipher.add_XOR_component(ids, pos, 16)
tmp = ComponentState([toyCipher.get_current_component_id()], [list(range(16))])
ids, pos = get_inputs_parameter([tmp, key2])
toyCipher.add_XOR_component(ids, pos, 16)
```

```
ct = ComponentState([toyCipher.get_current_component_id()], [list(range(16))])
toyCipher.add_output_component(ct, 16)
```

C Chosen AEAD Encryption Functions

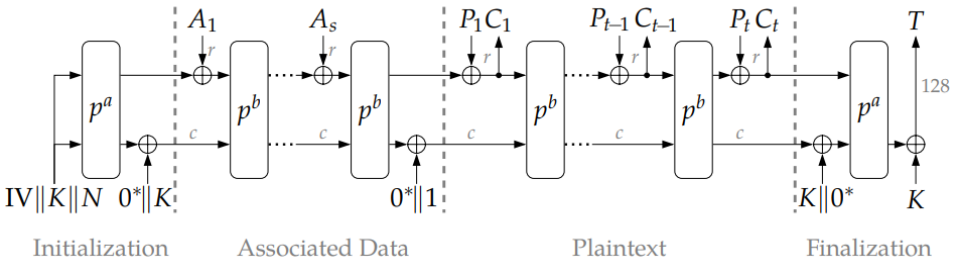
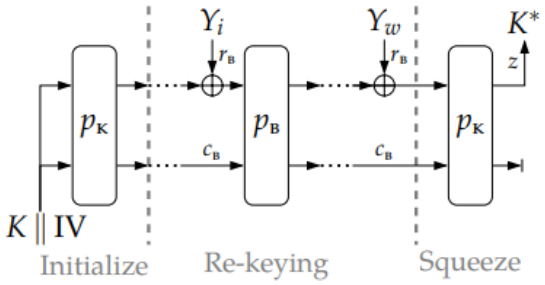
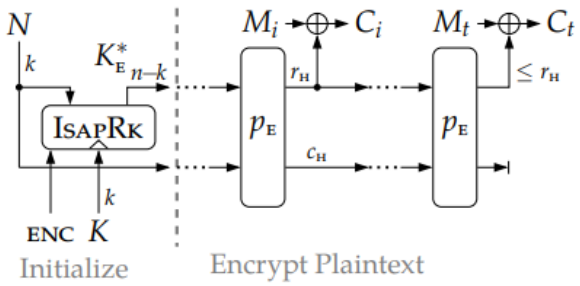


Figure 7: ASCON encryption; figure taken from [Dob+16]



(c) IsapRk , with $(\mathrm{IV}, z) = (\mathrm{IV}_{\mathrm{KE}}, n-k)$ if $f = \mathrm{ENC}$ else $(\mathrm{IV}_{\mathrm{KA}}, k)$ if flag $f = \mathrm{MAC}$



(d) IsapEnc

Figure 8: ISAP encryption; figure taken from [Dob+20]

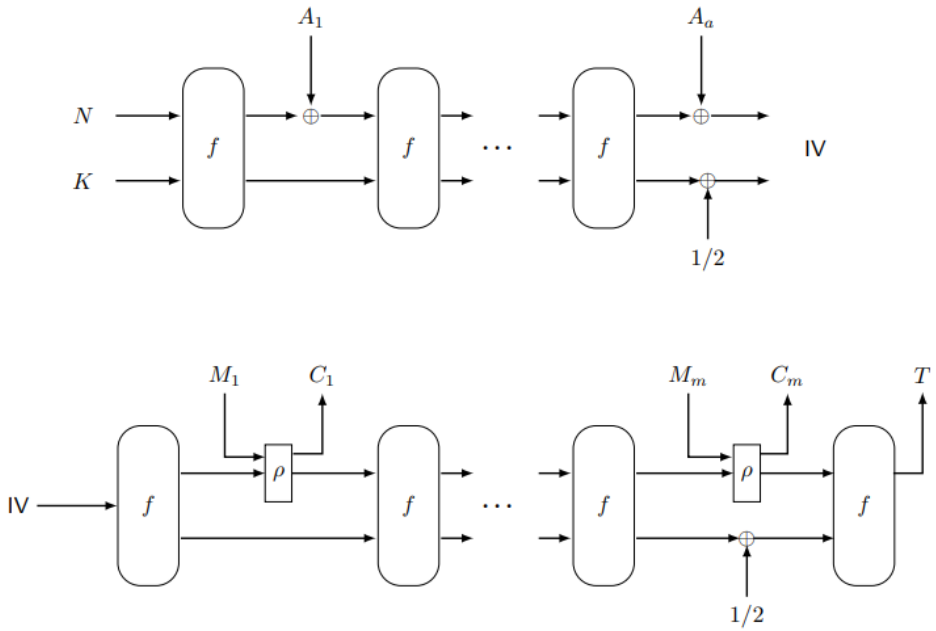


Figure 9: PHOTON-Beetle encryption; figure taken from [Bao+19]

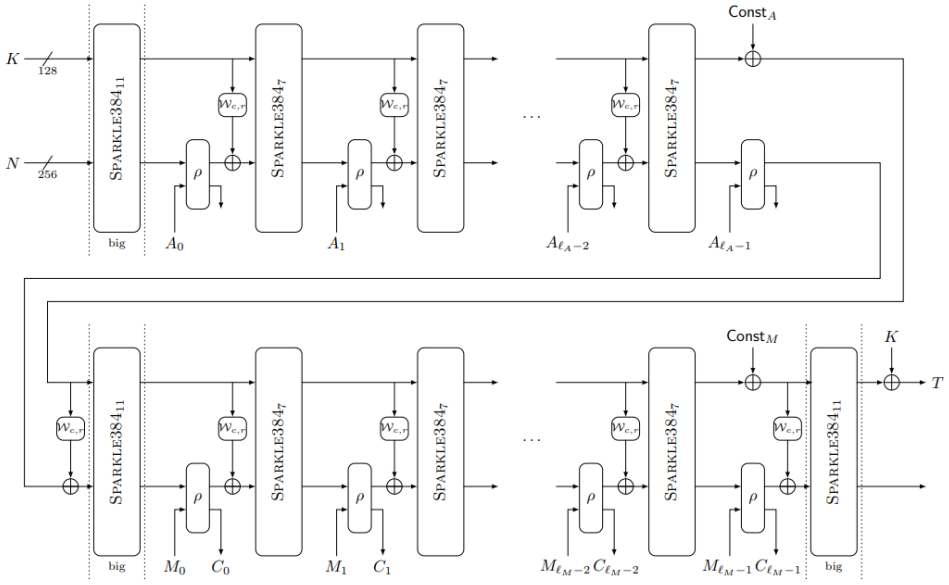


Figure 10: SCHWAEMM-256-128 encryption; figure taken from [Bei+19]

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%
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%

The setup lists the following values: number of training samples (with K = 1,000), number of samples per batch, number of epochs. The validation sample number is $\frac{1}{10}$ that of the training samples. Training and validation datasets are generated independently.

E Block Cipher Distinguishing Results

Tr-Acc. and Val-Acc. are the training and validation accuracies, respectively. Stars ('*') represent unrecorded data. The setup column lists the following values: number of training samples (with K = 1,000), number of samples per batch, number of epochs.

N.B.: We are also able to train 1-round CRAX-S-10 distinguishers, however judge this result not relevant.

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%