A Deeper Look at Machine Learning-Based Cryptanalysis

Adrien Benamira[1], David Gerault[1,2], Thomas Peyrin[1], and Quan Quan Tan[1]

[1] Nanyang Technological University, Singapore
[2] University of Surrey, UK
adrien002@e.ntu.edu.sg, dagerault@gmail.com, thomas.peyrin@ntu.edu.sg, quanquan001@e.ntu.edu.sg

Keywords: Differential Cryptanalysis, SPECK, Machine Learning, Deep Neural Networks, Interpretability

Abstract. At CRYPTO'19, Gohr proposed a new cryptanalysis strategy based on the utilisation of machine learning algorithms. Using deep neural networks, he managed to build a neural based distinguisher that surprisingly surpassed state-of-the-art cryptanalysis efforts on one of the versions of the well studied NSA block cipher SPECK (this distinguisher could in turn be placed in a larger key recovery attack). While this work opens new possibilities for machine learning-aided cryptanalysis, it remains unclear how this distinguisher actually works and what information is the machine learning algorithm deducing. The attacker is left with a black-box that does not tell much about the nature of the possible weaknesses of the algorithm tested, while hope is thin as interpretability of deep neural networks is a well-known difficult task.
In this article, we propose a detailed analysis and thorough explanations of the inherent workings of this new neural distinguisher. First, we studied the classified sets and tried to find some patterns that could guide us to better understand Gohr's results. We show with experiments that the neural distinguisher generally relies on the differential distribution on the ciphertext pairs, but also on the differential distribution in penultimate and antepenultimate rounds. In order to validate our findings, we construct a distinguisher for SPECK cipher based on pure cryptanalysis, without using any neural network, that achieves basically the same accuracy as Gohr's neural distinguisher and with the same efficiency (therefore improving over previous non-neural based distinguishers).
Moreover, as another approach, we provide a machine learning-based distinguisher that strips down Gohr's deep neural network to a bare minimum. We are able to remain very close to Gohr's distinguishers' accuracy using simple standard machine learning tools. In particular, we show that Gohr's neural distinguisher is in fact inherently building a very good approximation of the Differential Distribution Table (DDT) of the cipher during the learning phase, and using that information to directly classify ciphertext pairs. This result allows a full interpretability of the distinguisher and represents on its own an interesting contribution towards interpretability of deep neural networks.
Finally, we propose some method to improve over Gohr's work and possible new neural distinguishers settings. All our results are confirmed with

experiments we have been conducted on SPECK block cipher (source code available online).

1 Introduction

While modern symmetric-key cryptography designs are heavily relying on security by construction with strong security arguments (resistance against simple differential/linear attacks, study of algebraic properties, etc.), cryptanalysis remains a crucial part of a cipher's validation process. Only a primitive that went through active and thorough scrutiny of third-party cryptanalysts should gain enough trust by the community to be considered as secure. However, there has been more and more cipher proposals in the past decade (especially with the recent rise of lightweight cryptography) and cryptanalysis effort could not really keep up the pace: conducting cryptanalysis remains a very tough and low-rewarding task.

In order to partially overcome this shortage in cryptanalysts manpower, a recent trend arose of automating as much as possible the various tasks of an attacker. Typically, searching for differential and linear characteristics can now be modeled as Satisfiability/Satisfiability Modulo Theories [17] (SAT/SMT), Mixed Linear Integer Programming [18] (MILP) or Constraint Programming [25] (CP) problems, which can in turn simply be handled by an appropriate solver. The task of the cryptanalyst is therefore reduced to only providing an efficient modeling of the problem to be studied. Due to the impressive results considering the simplicity of the process, a lot of advances have been made in the past decade in this very active research field and this even improved the ciphers designs themselves (how to choose better cryptographic bricks and how to assemble them has been made much easier thanks to these new automated tools). One is then naturally tempted to push this idea further by even getting rid of the modeling part. More generally, can a tool recognize possible weaknesses/patterns in a cipher by just interacting with it, with as little input as possible from the cryptanalysts? One does not expect such a tool to replace a cryptanalyst's job, but it might come in handy for easily pre-checking a cipher (or reduced versions of it) for possible weaknesses.

Machine learning and particularly deep learning have recently attracted a lot of attention, due to impressive advances in important research areas such as computer vision, speech recognition, etc. Some possible connections between cryptography and machine learning were already identified in [21] and we have seen many applications of machine learning for side-channels analysis [16]. However, machine learning for black-box cryptanalysis remained mostly unexplored until Gohr's article presented at CRYPTO'19 [11].

In his work, Gohr trained a deep neural network on labeled data composed of ciphertext pairs: half the data coming from ciphering plaintexts pairs with a fixed input difference with the cipher studied, half from random values. He then checks if the trained neural network is able to classify accurately random from real ciphertext pairs. Quite surprisingly, when applying his framework to the block cipher SPECK-32/64 (the 32-bit block 64-bit key version of SPECK [2]), he managed to obtain a good accuracy for a non-negligible number of rounds. He even managed to mount a key recovery process on top of his neural distinguisher, eventually leading to the current best known key recovery attack for this number of rounds (improving over works on SPECK-32/64 such as [6, 24]). Even if his distinguisher/key recovery attack had not been improving over the state-of-the-art, the prospect of a generic tool that could pre-scan for vulnerabilities in a cryptographic primitive (while reaching an accuracy close to exiting cryptanalysis) would have been very attractive anyway.

Yet, Gohr's paper actually opened many questions. The most important, listed by the author as an open problem, is the interpretability of the distinguisher. An obvious issue with a neural distinguisher is that its black-box nature is not really telling us much about the actual weakness of the cipher analyzed. More generally, interpretability for deep neural networks has been known to be a very complex problem and represents a key challenge for the machine learning community. At first sight, it seems therefore very difficult to make any advances in this direction.

Another interesting aspect to explore is to try to match Gohr's neural distinguisher/key recovery attack with classical cryptanalysis tools. It remains very surprising that a trained deep neural network can perform better than the scrutiny of experienced cryptanalysts. As remarked by Gohr, his neural distinguisher is mostly differential in nature (on the ciphertext pairs), but some unknown extra property is exploited. Indeed, as demonstrated by one of his experiments, the neural distinguisher can still distinguish between a real and a random set that have the exact same differential distribution on the ciphertext pairs. Since we know there is some property that researchers have not seen or exploited, what is it?

Finally, a last natural question is: can we do better? Are there some better settings that could improve the accuracy of Gohr's distinguishers?

Our Contributions. In this article, we analyze the behavior of Gohr's neural distinguishers when working on SPECK-32/64 cipher. We first study in detail the classified sets of real/random ciphertext pairs in order to get some hints on what criterion the neural network is actually basing its decisions on. Looking for patterns, we observe that the neural distinguisher is very probably deducing some differential conditions not on the ciphertext pairs directly, but on the penultimate or antepenultimate rounds. We then conduct some experiments to validate our hypothesis.

In order to further confirm our findings, we construct for 5, 6 and 7-round reduced SPECK-32/64 a new distinguisher purely based on cryptanalysis, without any neural network or machine learning algorithm, that matches Gohr's neural distinguisher's accuracy while actually being faster and using the same amount of precomputation/training data. In short, our distinguisher relies on selective partial decryption: in order to attack nr rounds, some hypothesis is made on some bits of the last round subkey and partial decryption is performed, eventually filtered by a precomputed approximated DDT on $nr - 1$ rounds.

We then take a different approach by tackling the problem not from the cryptanalysis side, but the machine learning side. More precisely, as a deep learning model learns high-level features by itself, in order to reach full interpretability we need to discover what these features are. By analyzing the components of Gohr's neural network, we managed to identify a procedure to model these features, while retaining almost the same accuracy as Gohr's neural distinguishers. Moreover, we also show that our method performs similarly on other primitives by applying it on the SIMON block cipher. This result is interesting from a cryptography perspective, but also from a machine learning perspective, showing an example of interpretability by transformation of a deep neural network.

Finally, we explore possible improvements over Gohr's neural distinguishers. By using batches of ciphertexts instead of pairs, we are able to significantly improve the accuracy of the distinguisher, while maintaining identical experimental conditions.

Outline. In Section 2, we introduce notations as well as basic cryptanalysis and machine learning concepts that will be used in the rest of the paper. In Section 3, we describe in more detail the various experiments conducted by Gohr and the corresponding results. We provide in Section 4 an explanation of his neural distinguishers as well as the description of an actual cryptanalysis-only distinguisher that matches Gohr's accuracy. We propose in Section 5 a machine learning approach to enable interpretability of the neural distinguishers. Finally, we studied possible improvements in Section 6.

2 Preliminaries

Basic notations. In the rest of this article, $\oplus$, $\wedge$ and $\boxplus$ will denote the eXclusive-OR operation, the bitwise AND operation and the modular addition[3] respectively. A right/left bit rotation will be denoted as $\ggg$ and $\lll$ respectively, while $a||b$ will represent the concatenation of two bit strings a and b.

2.1 A Brief Description of SPECK

The lightweight family of ARX block ciphers SPECK was proposed by the US National Security Agency (NSA) [2] in 2013, targeting mainly good performances on micro-controllers. Several versions of the cipher have been proposed within its family, but in this article (and in Gohr's work [11]) we will focus mainly on SPECK-32/64, the 32-bit block 64-bit key version of SPECK, which is composed of 22 rounds (for simplicity, SPECK-32/64 will be referred to as SPECK in the rest of the article).

The 32-bit internal state is divided into a 16-bit left and a 16-bit right part, that we will generally denote l_i and r_i at round i respectively, and is initialised with the plaintext $(l_0||r_0) \leftarrow P$. The round function of the cipher is then a very

[3] The modulo will be stated explicitly if it is not clear from the context

simple Feistel structure combining bitwise XOR operation and 16-bit modular addition. See Figure 1 where k_i represents the 16-bit subkey at round i and where $\alpha = 7$, $\beta = 2$. The final ciphertext C is then obtained as $C \leftarrow (l_{22}||r_{22})$. The subkeys are generated with a key schedule that is very similar to the round function (we refer to [2] for a complete description, as we do not make use of the details of the key schedule in this article).

$$l_{i+1} = ((l_i \ggg \alpha) \boxplus r_i) \oplus k_i$$
$$r_{i+1} = (r_i \lll \beta) \oplus l_{i+1}$$

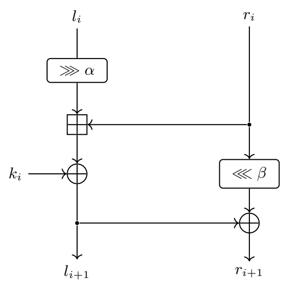


Fig. 1: The SPECK-32/64 round function.

2.2 Differential Cryptanalysis

Differential cryptanalysis studies the propagation of a difference through a cipher. Let a function $f : \mathbb{F}_2^b \rightarrow \mathbb{F}_2^b$ and x, x' be two different inputs for f with a difference $\Delta x = x \oplus x'$. Let $y = f(x)$ and $y' = f(x')$ and a difference $\Delta y = y \oplus y'$. Then, we are interested in the transition probability from Δx to Δy ($\Delta x \xrightarrow{f} \Delta y$):

$$\mathbb{P}(\Delta x \xrightarrow{f} \Delta y) := \frac{\#\{x|f(x) \oplus f(x \oplus \Delta x) = \Delta y\}}{2^b}$$

One classical tool for differential cryptanalysis is the Difference Distribution Table (DDT), which simply lists the differential transition probabilities for each possible input/output difference pairs $(\Delta x, \Delta y)$. The studied function f is usually some Sbox, or some small cipher sub-component, as the DDT of an entire 64-bit or 128-bit cipher would obviously be too large to store.

Since SPECK is internally composed of a left and right part, for a ciphertext C we will denote by C_l and C_r its 16-bit left and right parts respectively. Then, for two ciphertexts C and C', we will denote ΔL the XOR difference $C_l \oplus C'_l$ between the left parts of the two ciphertexts (respectively $\Delta R = C_r \oplus C'_r$ for the right parts). Moreover, for a round i, we will denote by V_i the difference between the two parts of the internal state $V_i = l_i \oplus r_i$.

2.3 Deep Neural Networks

Deep Neural Networks (DNN) are a family of non-linear machine learning classifiers that have gained popularity since their success in addressing a variety of data-driven tasks, such as computer vision, speech recognition, etc.

The main problem tackled by DNN is, given a dataset $D = \{(x_0, y_0)...(x_n, y_n)\}$, with $x_i \in X$ being samples and $y_i \in [0, \ldots, l]$ being labels, to find the optimal parameters θ^* for the DNN_θ model, with the parameters θ such that:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{i=0}^{n} L(y_i, DNN_\theta(x_i)) \quad (1)$$

with L being the loss function. As there is no literal expression of θ^*, the approximate solution will depend on the chosen optimization algorithm such as the stochastic gradient descent. Moreover, hyper-parameters of the problem (parameters whose value is used to control the learning process) need to be adjusted as they play an important role in the final quality of the solution.

DNN are powerful enough to derive accurate non-linear features from the training data, but these features are not robust. Indeed, adding a small amount of noise at the input can cause these features to deviate and confuse the model. In other words, the DNN is a very unbiased classifier, but has a high variance.

Different blocks can be used to implement these complex models. However, in this paper, we will be using four types of blocks: the linear neural network, the one-dimensional convolutional neural network, the activation functions (ReLU and sigmoid) and the batch normalization.

Linear neural network. Linear neural networks applies a linear transformation to the incoming data: $out = in.A^T + b$. Here we have $\theta = (A, b)$. The linear neural network is also commonly named perceptron layer or dense layer.

One-dimensional convolutional neural network. The 1D-CNN applies a convolution over a fixed (multi-)temporal input signal. The 1D-CNN operation can be seen as multiple linear neural networks (one per filter) where each one is applied to a sub-part of the input. This sub-part is sliding, its size is kernel size, its pitch is the stride and its start and end points depend on the padding.

Activation functions. The three activation functions that we discuss here are the Rectified Linear Unit (ReLU), defined as $\mathrm{ReLU}(x) = \max(0, x)$, the sigmoid, defined as $\mathrm{Sigmoid}(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ and the Heaviside step function, defined as $H(x) = \frac{1}{2} + \frac{sgn(x)}{2}$. This block, added between each layer of the DNN, introduces the non-linear part of the model.

Batch normalization. Training samples are typically randomly collected in batches to speed up the training process. It is thus usual to normalize the overall tensor according to the batch dimension.

3 A First Look at Gohr's CRYPTO 2019 Results

Since its release, the lightweight block cipher SPECK attracted a lot of external cryptanalysis, together with its sibling SIMON (this was amplified by the fact that no cryptanalysis was reported in the original specifications document [2]). Many different aspects of SPECK have been covered by these efforts, but the works from Dinur [6] and Song *et al.* [24] are the most successful advances on its differential cryptanalysis aspect so far. Dinur [6] studied all versions of SPECK, improving the best known differential characteristics (from [1,3]) as well as describing a new key recovery strategy for this cipher. In particular, he devised a 4-round attack for 11 rounds of SPECK-32/64 using a 7 round differential characteristic, that has a time complexity of 2^{46} and data complexity of 2^{22} (chosen plaintexts).

Later, at CRYPTO'19, Gohr published a cryptanalysis work on SPECK-32/64 that is based on deep learning [11]. Gohr proposed a key-recovery attack on 11-round SPECK-32/64 with estimated time complexity 2^{38}, improving the previous best attack [6] in 2^{46}, albeit with a slightly higher data complexity: $2^{14.5}$ ciphertext pairs required. In this section, we will briefly review Gohr's results [11].

Overview. In his article, Gohr proposes multiple differential cryptanalysis of SPECK, focusing on the input difference Δ_{in} = `0x0040/0000`. In this setting, the aim is to distinguish *real pairs*, *i.e.*, encryptions of plaintext pairs P, P' such that $P \oplus P' = \Delta_{in}$, from *random pairs*, which are the encryptions of random pairs of plaintext with no fixed input difference. Gohr compares a traditional (pure) differential distinguisher with a distinguisher based on a DNN for 5 to 8 rounds of SPECK-32/64 and showed that the DNN performs better.

Pure differential distinguishers. Gohr computed the full DDT for the input difference Δ_{in}, using the Markov assumption. Then, to classify a ciphertext pair (C, C'), the probability p of the output difference $C \oplus C'$ is read from the DDT and compared to the uniform probability. Let $\Delta_{out} = C \oplus C'$, then

$$\text{Classification } = \begin{cases} Real & \text{if } DDT(\Delta_{in} \to \Delta_{out}) > \frac{1}{2^{32}-1} \\ Random & \text{otherwise} \end{cases}$$

These distinguishers for reduced-round SPECK-32/64 are denoted D_{nr}, where $nr \in \{5, 6, 7, 8\}$ represents the number of rounds. The neural distinguishers are denoted as N_{nr}.

Gohr's neural distinguisher. We provide in Figure 2 a representation of Gohr's neural distinguisher. It is a deep neural network, whose main components are:

1. Block 1: a 1D-CNN with kernel size of 1, a batch normalization and a ReLU activation function
2. Blocks 2-i: one to ten layers with each layer consisting of two 1D-CNN with kernel size of 3, each followed by batch normalization and a ReLU activation function.
3. Block 3: a non-linear final classification block, composed of three perceptron layers separated by two batch normalization and ReLU functions, and finished with a sigmoid function.

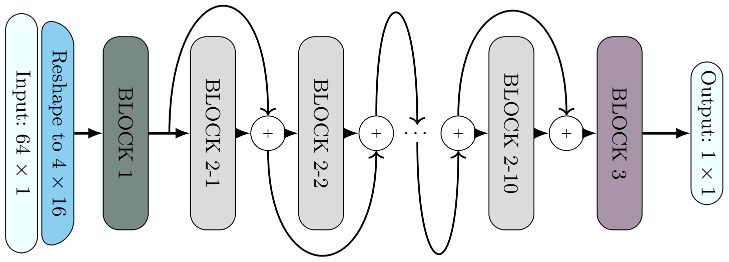


Fig. 2: The whole pipeline of Gohr's deep neural network. Block 1 refers to the initial convolution block, Block 2-1 to 2-10 refer to the residual block and Block 3 refers to the classification block.

The input to the initial convolution block (Block 1) is a 4×16 matrix, where each row corresponds to each 16-bit value in this order: C_l, C_r, C'_l, C'_r, a convolution layer with 32 filters is then applied. The kernel size of this 1D-CNN is 1, thus, it maps (C_l, C_r, C'_l, C'_r) to $(filter_1, filter_2, ..., filter_{32})$. Each $filter$ is a non-linear combination of the features (C_l, C_r, C'_l, C'_r) after the ReLU activation function depending on the value of the inputs and weights learned by the 1D-CNN. The output of the first block is connected to the input and added to the output of the subsequent layer in the residual block (see Figure 3).

In the residual blocks (Blocks 2-i), both 1D-CNNs have a kernel of size 3, a padding of size 1 and a stride of size 1 which make the temporal dimension invariant across layers. At the end of each layer, the output is connected to the input and added to the output of the subsequent layer to prevent the relevant input signal from being wiped out across layers. The output of a residual block is a (32×16) feature tensor (see Figure 4).

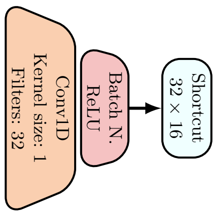


Fig. 3: Initial convolution block (Block 1).

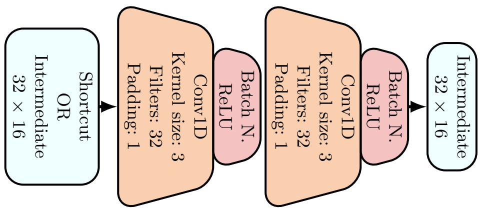


Fig. 4: The residual block (Blocks 2-i).

The final classification block takes as input the flattened output tensor of the residual block. This 512×1 vector is passed into three perceptron layers (Multi-Layer Perceptron or MLP) with batch normalization and ReLU activation functions for the first two layers and a final sigmoid activation function performs the binary classification (see Figure 5).

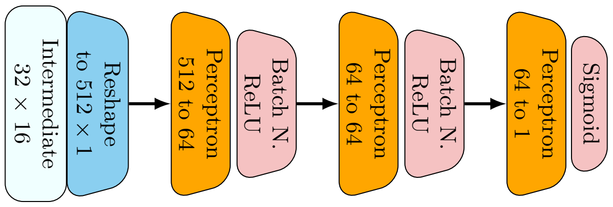


Fig. 5: The classification block (Block 3).

Accuracy and efficiency of the neural distinguishers. For each pair, the neural distinguishers outputs a real-valued score between 0 and 1. If this score is greater than or equal to 0.5, the sample is classified as a real pair, and as a random pair otherwise. The results given by Gohr are presented in Table 1. Note that N_7 and N_8 are trained using some sophisticated methods (we refer to [11] for more details on the training). We remark that Gohr's neural distinguisher has about 100,000 floating parameters, which is size efficient considering the accuracies obtained.

Table 1: Accuracies of neural distinguishers for 5, 6, 7 and 8 rounds (taken from Table 2 of [11]). TPR and TNR denote true positive and true negative rates respectively.

Rds	**Distinguisher**	**Accuracy**	**TPR**	**TNR**
5	D_5	0.911	0.877	0.947
	N_5	$0.929 \pm 5.13 \times 10^{-4}$	$0.904 \pm 8.33 \times 10^{-4}$	$0.954 \pm 5.91 \times 10^{-4}$
6	D_6	0.758	0.680	0.837
	N_6	$0.788 \pm 8.17 \times 10^{-4}$	$0.724 \pm 1.26 \times 10^{-3}$	$0.853 \pm 1.00 \times 10^{-3}$
7	D_7	0.591	0.543	0.640
	N_7	$0.616 \pm 9.7 \times 10^{-4}$	$0.533 \pm 1.41 \times 10^{-3}$	$0.699 \pm 1.30 \times 10^{-3}$
8	D_8	0.512	0.496	0.527
	N_8	$0.514 \pm 1.00 \times 10^{-3}$	$0.519 \pm 1.41 \times 10^{-3}$	$0.508 \pm 1.42 \times 10^{-3}$

Real differences experiment. The neural distinguishers performed better than the distinguishers using the full DDT, indicating that the neural distinguishers may learn something more than pure differential cryptanalysis. Gohr explores this effect with the *real differences experiment*. In this experiment, instead of distinguishing a real pair from a random pair, the challenge is to distinguish real pairs from masked real pairs, computed as $(C \oplus M, C' \oplus M)$, where M is a random 32-bit value. These experiments use the N_{nr} distinguishers directly, without retraining them for this new task. Table 2 shows the accuracies of these distinguishers. Notice that this operation does not affect

$\Delta_{out} = C \oplus C' = (C \oplus M) \oplus (C' \oplus M)$ and thus the output difference distribution. However, the neural distinguishers are still able to distinguish real pairs from masked pairs even without re-training for this particular purpose, which shows that they do not just rely on the difference distribution.

Table 2: Accuracies of various neural distinguishers in the real differences experiment.

Rds	**Distinguisher**	**Accuracy**
5	N_5	$0.707 \pm 9.10 \times 10^{-4}$
6	N_6	$0.606 \pm 9.77 \times 10^{-4}$
7	N_7	$0.551 \pm 9.95 \times 10^{-4}$
8	N_8	$0.507 \pm 1.00 \times 10^{-3}$

4 Interpretation of Gohr's Neural Network: a Cryptanalysis Perspective

Interpretability of neural networks remains a highly researched area in machine learning, but the focus has always been on improving the model and computational efficiency. We will discuss more about the interpretability in a machine learning sense in Section 5. In this section, we want to find out why and how the neural distinguishers work in a cryptanalysis sense. In essence, we want to answer the following question:

What type of cryptanalysis is Gohr's neural distinguisher learning?

If the neural distinguisher is learning some currently-unknown form of cryptanalysis, then we would like to extrapolate the additional statistics that it exploits. If not, then we want to find out what causes Gohr's neural distinguishers to perform better than pure differential attacks, and even improve state-of-the-art attacks. With this question in mind, we perform a series of experiments and analyses in order to come up with a reasonable guess, later validated by the creation of a pure cryptanalysis-based distinguisher that matches the accuracy of Gohr's one.

Gohr's neural distinguishers are able to correctly identify approximately 90.4%, 68.0% and 54.3% of the real ciphertext pairs (given by the true positive rates) for 5, 6 and 7 rounds of SPECK-32/64 respectively (see Table 1). We will try to find out what these ciphertext pairs are if there are any common patterns and see whether we are able to identify and isolate them.

Choice of input difference. As a start, we looked into Gohr's choice of input difference: `0x0040/0000`. This difference is part of a 9-round differential characteristics from Table 7 of [1]. The reason given by Gohr is that this difference deterministically transits to a difference with low Hamming weight after one round. Using constraint programming and techniques similar to [10], we found that the best differential characteristics with a **fixed** input difference of `0x0040/0000` for 5 rounds is `0x0040/0000` $\rightarrow$ `0x802a/d4a8`, with probability of 2^{-13}. In contrast, when we do not restrict the input difference, the best differential characteristics for 5 rounds is `0x2800/0010` $\rightarrow$ `0x850a/9520`, with probability of 2^{-9}. However, when we trained the neural distinguishers to recognize ciphertext pairs with the input difference of `0x2800/0010`, the neural distinguishers performed worse (an accuracy of 75.85% for 5 rounds). This is surprising as it is generally natural for a cryptanalyst to maximize the differential probability when choosing a differential characteristic. We believe this is explained by the fact that `0x0040/0000` is the input difference maximizing the differential probability for 3 or 4 rounds of SPECK-32/64 (verified with constraint programming), which has the most chances to provide a biased distribution one or two rounds later. Generally, we believe that when using such neural distinguisher, a good method to choose an input difference is to simply use the input difference leading to the highest differential probability for $nr - 1$ or $nr - 2$ rounds.

Changing the inputs to the neural network. Gohr's neural distinguishers are trained using the actual ciphertext pairs (C, C') whereas the pure differential distinguishers are only using the difference between the two ciphertexts $C \oplus C'$. Thus, it is unfair to compare both as they are not exploiting the same amount of information. To have a fair comparison of the capability of neural distinguishers and pure differential distinguishers, we trained new neural distinguishers using $C \oplus C'$, instead of (C, C'). The results are an accuracy of 90.6% for 5 rounds, 75.4% for 6 rounds and 58.3% for 7 rounds. This shows us that when the neural distinguishers are restricted to only have access to the difference distribution, they do not perform as well as their respective N_{nr}, and similarly to D_{nr}[4] as can be seen in Table 1. Therefore, this is another confirmation (on top of the real differences experiment conducted in [11]), that Gohr's neural distinguishers are learning more than just the distribution of the differences on the ciphertext. With that information, we therefore naturally looked beyond just the difference distribution at round nr.

4.1 Analyzing Ciphertext Pairs

In this section, we limit and focus the discussions and results mostly to 5 rounds of SPECK-32/64. We recall that the last layer of the neural distinguisher is a sigmoid activation function. Thus, its output is a value between 0 and 1. When the score is 0.5 or more, the neural distinguisher predicts it as a real pair or otherwise, random pair.

The closer a score is to 0.5, the least certain the neural distinguisher is on the classification. In order to know what are the traits that the neural distinguisher is looking for, we segregate the ciphertext pairs that yield extreme scores, *i.e.* scores that are either less than 0.1 (bad score) or more than 0.9 (good score). For the

[4] Note that the new neural distinguishers are trained with 10^7 pairs, the same number as in [11]

rest of this section, we label the ciphertext pairs as "bad" and "good" ciphertext pairs and refer to the sets as B and G respectively. As we were experimenting with them, we kept the keys (unique to each pair) that are used to generate the ciphertext pairs. The goal now is to find similarities and differences in these two groups separately.

As we believe that most of the features the neural distinguishers learned is differential in nature, we focus on the differentials of these ciphertext pairs. To start, we did the following experiment (Experiment A):

1. Using 10^5 real 5-round SPECK-32/64 ciphertext pairs, extract the set G.
2. Obtain the differences of the ciphertext pairs and sort them by frequency
3. For each of the differences δ:
   (a) Generate 10^4 random 32-bit numbers and apply the difference, δ to get 10^4 different ciphertext pairs.
   (b) Feed the pairs to the neural distinguisher N_5 to obtain the scores.
   (c) Note down the number of pairs that yield a score ≥ 0.5

In Table 3, we show the top 25 differences for 5 rounds of SPECK-32/64 with their respective score from the above experiment. Out of the first 1000 differences, each records about 75% of the pairs scoring more than 0.5. Also, there exist multiple pairs of differences such that one is more probable than the other, and yet, it has a lower number of pairs classifying as real (*e.g.* No. 21 in Table 3). Thus, there is little evidence showing that if a difference is more probable, then the neural distinguisher is necessarily more likely to recognize it.

Table 3: The top 25 differences (5 rounds of SPECK-32/64) in G with their respective results for Experiment A as a percentage of how many pairs having a score of ≥ 0.5 out of 10^4 pairs. Cnt refers to the number of differences obtained in G.

No.	**Difference**	**Cnt**	**Percent.**	**No.**	**Difference**	**Cnt**	**Percent.**
1	`0x802a/d4a8`	116	75	14	`0x883a/dcb8`	45	75
2	`0x802e/d4ac`	81	76	15	`0x801e/d49c`	45	75
3	`0x803a/d4b8`	73	74	16	`0xa026/f4a4`	42	75
4	`0x8e2a/daa8`	73	75	17	`0xbe1a/ea98`	41	75
5	`0x822a/d6a8`	72	75	18	`0x821a/d698`	41	76
6	`0xb82a/eca8`	67	75	19	`0xbe26/eaa4`	41	75
7	`0x882a/dca8`	65	75	20	`0x83ea/d768`	40	75
8	`0x801a/d498`	62	75	21	`0x8626/caa4`	40	38
9	`0xa02a/f4a8`	62	75	22	`0x886a/dce8`	40	75
10	`0xbe2a/eaa8`	62	75	23	`0xa06a/f4e8`	40	75
11	`0x806a/d4e8`	59	74	24	`0x8e1a/da98`	39	75
12	`0x8e26/daa4`	47	75	25	`0x8226/cea4`	38	37
13	`0x8026/d4a4`	46	74				

Since the neural distinguishers outperform the ones with just the XOR input, we started to look beyond just the differences at 5 rounds. We decided to partially decrypt the ciphertext pairs from G for a few rounds and re-run Experiment A on these partially decrypted pairs: for each pair, we compute the difference and for each difference, we created 10^4 random plaintext pairs with these differences and encrypted them to round nr using random keys. The results are very intriguing, as compared to that of Table 3: almost all of the (top 1000) unique differences obtained in this experiment achieved 99% or 100% of ciphertext pairs having a score of ≥ 0.5.

We can see that the differences at rounds 3 and 4 (after decrypting 2 and 1 round respectively) start to show some strong biases. In fact, for all of the top 1000 differences at rounds 3 and 4, all 10^4 pairs $\times$ 1000 differences returned a score of ≥ 0.5[5]. With that, we conduct yet another experiment (Experiment B):

1. For all the ciphertext pairs in G, decrypt i rounds with their respective keys and compute the corresponding difference. Denote the set of differences as $\texttt{Diff}_{\texttt{5-i}}$.
2. Generate 10^5 plaintext pairs with a difference of `0x0040/0000` with random keys, encrypt to 4 rounds
3. If the pair's difference is in $\texttt{Diff}_{\texttt{5-i}}$, keep the pair. Otherwise, discard.
4. Encrypt the remaining pairs to 5 rounds and evaluate them using N_5.

When $i = 2$, we obtain 1669 unique differences with a dataset size of 89,969. 97.86% of these ciphertext pairs yielded a score ≥ 0.5 (*i.e.* by this method, we can isolate 88.04% of the true positive ciphertexts pair). Using $i = 1$, we have 128,039 unique differences and the size of the dataset is 74,077. While we could get a cleaner set with 99.98% of these ciphertext pairs obtaining a score of ≥ 0.5, we only managed to isolate 74.06% of the true positive pairs. Comparing with the true positive rate of N_5 from Table 1, which is $0.904 \pm 8.33 \times 10^{-4}$, the case when $i = 2$ seems to be closer.

We also looked into the bias of the difference bits (the j^{th} difference bit refers to the j^{th} bit index of $C_{5-2} \oplus C'_{5-2}$ where C_{nr-i} refers to the nr round ciphertext decrypted by i rounds. Table 4 shows the difference bit biases of the first 1000 (most common) unique differences of ciphertext pairs in G and B after decrypting two rounds. We assume that the neural distinguisher is able to identify some bits at these rounds because they are significantly more biased, though both the set B and G are from the real distribution.

Now, we state the assumption required for our conjecture, which we will verify experimentally in Section 4.3.

Assumption 1 *Given a 5-round* SPECK*-32/64 ciphertext pair, N_5 is able to determine the difference of certain bits at rounds 3 and 4 with high accuracy.*

Conjecture 1. Given a 5-round SPECK-32/64 ciphertext pair, N_5 finds the difference of certain bits at round 3 and decides if the ciphertext pair is real or random.

Interestingly, the difference bit biases after decrypting 1 and 2 rounds are very similar (in their positions). We will provide an explanation in Section 4.2.

[5] the differences were obtained experimentally.

Table 4: Difference bit bias of ciphertext pairs in G and B after decrypting 2 rounds. A negative (resp. positive) value indicates a bias towards '0' (resp. '1').

bit position	**31**	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
G	$\underline{0.476}$	$\underline{-0.454}$	-0.355	-0.135	0.045	0.084	-0.009	$\underline{0.487}$	$\underline{-0.473}$	-0.426	-0.300	-0.050	0.006	0.019	$\underline{0.500}$	$\underline{-0.500}$
B	-0.002	0.018	0.008	-0.011	0.044	0.002	0.023	-0.022	0.010	-0.002	0.013	-0.004	0.006	-0.005	0.103	0.072

bit position	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
G	$\underline{0.476}$	$\underline{-0.454}$	-0.142	-0.006	0.025	0.084	-0.009	$\underline{0.487}$	$\underline{-0.473}$	-0.426	0.165	0.094	-0.006	0.019	$\underline{-0.500}$	$\underline{-0.500}$
B	0.031	-0.009	-0.015	-0.007	-0.014	-0.024	0.025	0.026	0.034	-0.005	-0.018	-0.021	0.006	0.009	0.079	-0.065

The exact truncated differentials are ($*$ denotes no specific constraint, while 0 or 1 denotes the expected bit difference):

$$\text{3 rounds: } 10*****00*****00\ 10*****00*****10$$
$$\text{4 rounds: } 10*****10*****10\ 10*****10*****00$$

We refer to these particular truncated differential masks as TD_3 and TD_4 for the following discussion. Using constraint programming, we evaluate that the probabilities for these truncated differentials are 87.86% and 49.87% respectively. In order to verify how much the neural distinguisher is relying on these bits, we perform the following experiment (Experiment C):

1. Generate 10^6 plaintext pairs with initial difference `0x0040/0000` and 10^6 random keys.
2. Encrypt all 10^6 plaintext pairs to $5 - i$ rounds. If a plaintext pair satisfies the TD_{5-i}, then we keep it. Otherwise, it will be discarded.
3. Encrypt the remaining pairs to 5 rounds and evaluate them using N_5.

Table 5: Results of Experiment C with TD_3 and TD_4. Proport. refers to the number of true positive ciphertext pairs captured by the experiment.

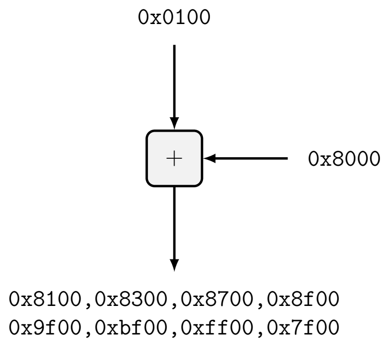
5-i	**Trunc. Diff.**	**Dataset size**	**Acc.**	**Proport.**
3	TD3	87741	99.277%	87.11%
4	TD4	50063	99.996%	50.06%

Table 5 shows the statistics of the above experiment with 5 rounds of SPECK-32/64. The true positive rates for ciphertext pairs that follow these are closer to that of Gohr's neural distinguisher. Now, there remains about 3% of the ciphertext pairs yet to be explained (comparing the results of TD_{5-2} with N_5). The important point to note here is that the pairs we have identified are exactly the ones verified by the neural distinguisher as well, by the nature of these experiments. In other words, we managed to find what the neural distinguisher is looking for and not just another distinguisher that would achieve a good accuracy by identifying a different set of ciphertext pairs.

4.2 Deriving TD_3 and TD_4

With an input difference of `0x0040/0000`, which has a deterministic transition to `0x8000/8000` in round 1, the difference will only start to spread after round 1 due to the modular addition in the SPECK-32/64 round function. The inputs to the modular addition at round 2 are `0x0100` and `0x8000` (cf. Figure 6). While there are two active bits, only one of them will propagate the carry (as the other is the MSB), resulting in multiple differences. Assuming a uniform distribution, the carry has a probability $\frac{1}{2}$ of propagating to the left. This causes the probability of the various differentials to reduce by $\frac{1}{2}$ as the carry bit propagates until b_{31} (bit position 31) is reached and any further carry will be removed by the modular addition.

Fig. 6: The distribution of the possible output differences after passing through the modular addition operation.



Out. Diff.	Prob.
0x8100	2^{-1}
0x8300	2^{-2}
0x8700	2^{-3}
0x8f00	2^{-4}
0x9f00	2^{-5}
0xbf00	2^{-6}
0xff00	2^{-7}
0x7f00	2^{-7}

In Figure 7 and Figure 8, we show how the bits evolve along the most probable differential path from round 1 (`0x8000/8000`) to round 4 (`0x850a/9520`). As it passes through the modular addition operation, we highlight the bits that have a relatively higher probability of being different from the most probable differential. The darker the color, the higher the probability of the difference being toggled.

Figure 7 and Figure 8 show us why TD_3 is important at round 3, and how the active bits shift in SPECK-32/64 when we start with the input difference of `0x0040/0000`. In every round, b_{31}, (the leftmost bit) has a high probability of staying active. This bit is then rotated to b_{24} before it goes into the modular addition operation. In each round, b_{26} has a $\frac{1}{2}$ chance of switching from $1 \rightarrow 0$ or the other way round. b_{27} and b_{28} have a $\frac{1}{4}$ and $\frac{1}{8}$ chance respectively of switching. This makes them highly volatile and therefore, unreliable. On the other hand, the right part of SPECK-32/64 rotates by 2 to the left at the end of each round. Because of the high rotation value in the left part of SPECK-32/64, low rotation value of the right part of SPECK-32/64, and the fact that the left part is added into the right part after the rotation, it takes about 3 to 4 rounds for the volatile and unreliable bits to spread.

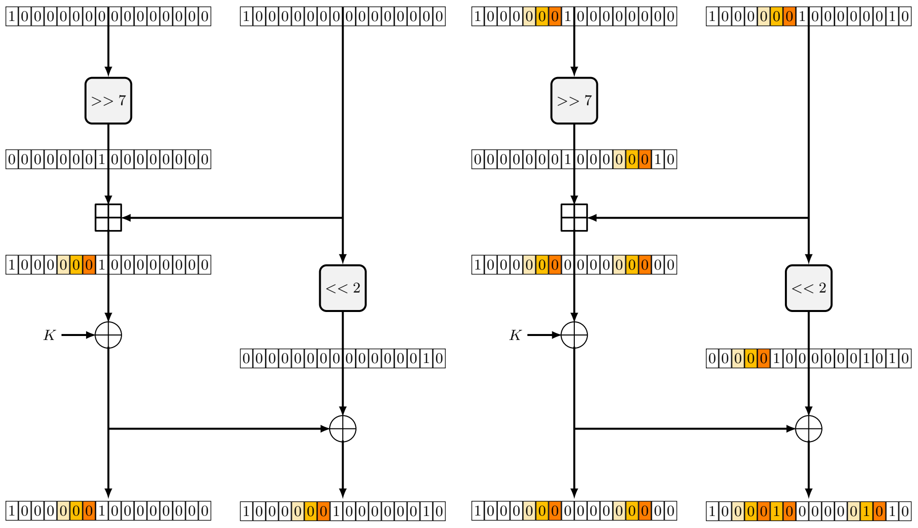


Fig. 7: The left (resp. right) part shows how the active bit from difference `0x8000/8000` (resp. `0x8100/8102`) propagates to difference `0x8100/8102` (`0x8000/820a`). The darker the color, the higher the probability ($\geq \frac{1}{4}$) that it has a carry propagated to.

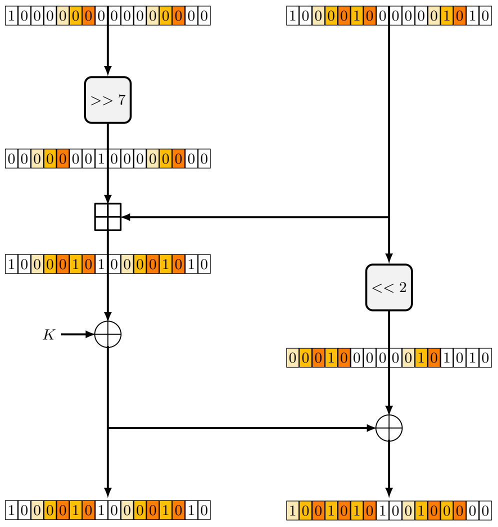


Fig. 8: Showing how the active bit from difference `0x8000/820a` propagates to difference `0x850a/9520`. The darker the color, the higher the probability ($\geq \frac{1}{4}$) that it has a carry propagated to.

4.3 Verifying Assumption 1

To verify if Gohr's neural distinguisher is able to recognize the truncated differential, we retrain the neural distinguisher with a slight difference (Experiment D):

1. Generate 10^7 plaintext pairs such that about $\frac{1}{2}$ of the pairs satisfy TD_3 (these are the positive pairs)
2. Encrypt the plaintext pairs for two rounds
3. Train the neural network to distinguish the two distributions, and validate with the same hyper-parameters as in [11], with a depth of 1 in the residual block.

After retraining, the neural distinguisher has an accuracy of 96.57% (TPR: 99.95%, TNR: 93.19%) This shows that the neural distinguisher has the capabilities to actually recognize the truncated differential with an outstanding accuracy.

4.4 SPECK-32/64 Reduced to 6 Rounds

We perform Experiments C for 6 rounds of SPECK-32/64 as well. Table 6 shows the comparison of the true positive results of rounds 5 and 6. While the results are not as obvious as for the case of 5 rounds, we can still observe a similar trend for 6 rounds.

6-i rds	Truncated Differential	Size of dataset	Accuracy	Proport.
4	10*****10*****10 10*****10*****00	49902	99.41%	49.6%
5	10*****00*****10 11*****01*****00	6884	99.927%	6.88%

Table 6: Results of SPECK-32/64 reduced to six rounds for Experiment C. Proportion refers to the number of true positive ciphertext pairs captured by the experiment.

4.5 Average Key Rank Differential Distinguisher

Taking into consideration the observations we presented in this section, we introduce a new average key rank distinguisher that is not based on machine learning and almost matches the accuracy as Gohr's neural network for 5, 6 and 7 rounds of SPECK-32/64. Here are the key considerations used in our distinguisher:

– The training set of Gohr's neural network consists of 10^7 ciphertext pairs. Thus, we restrict our distinguisher to only use 10^7 ciphertext pairs as well.
– If we do an exhaustive key search for two rounds, the time complexity will be extremely high. Instead, we may need to limit ourselves to only one round to match the complexity of the neural distinguishers.
– If we know the difference at round i, the $i-1$ round difference for the right part is known as well, since $r_{i-1} = (l_i \oplus r_i) \ggg 2$

With those pointers in mind, we created a distinguisher that uses an approximated DDT (aDDT); that is, a truncated DDT that is experimentally constructed based on n ciphertext pairs. In this distinguisher, we use $n = 10^7$ to ensure that both our distinguisher and the neural distinguishers have the same amount of information. The idea of the distinguisher is to decrypt the last round, nr, using all possible subkey bits that are relevant to the bits we are interested in. Then, we compute the average of the probabilities of all partial decryptions for a given pair, read from $aDDT(nr - 1)$, to get a score. If the score is greater than that of the random distribution, the distinguisher will return 1 (Real) and 0 (Random) otherwise. The bits we are interested in can be represented as an AND mask, that is, a mask that has '1' in the bit positions that we want to consider the bit and '0' for those we want to ignore. The mask value we have chosen is `0xff8f/ff8f` rather than the expected `0xc183/c183` as we believe the truncated differential they are detecting is at $nr - 2$ rounds. Thus, other than the bits that are identified earlier in this section, we decided to include more bits to improve the accuracy. With the look-up table to the aDDT, we do not just only match the data complexity (of the offline training) of the Gohr's neural distinguishers, but at the same time, include the correlations between bits as well.

The pseudocodes for creating the aDDT and the average key rank distinguisher can be found in Algorithm 1 and Algorithm 2 given in Supplementary Materials. We applied the distinguisher for 5, 6 and 7 rounds of SPECK-32/64 and the results are given in Table 7. It shows that our distinguisher closely matches the accuracies of Gohr's neural distinguishers.

Degree of closeness. We now study the similarity between our distinguishers and Gohr's neural distinguishers. In particular, we are interested in whether the classifications of the ciphertext pairs are the same for both distinguishers. To verify this, we gave a set of 10^5 5-round ciphertext pairs (approx. 50,000 from real and random distribution each) to both our average key rank distinguisher and N_5, and measured how many times did they have the same output. The results for $nr = 5$ are shown in Table 8. We can see that about 97.6% of the ciphertext pairs tested have the same classification in both distinguishers. For $nr = 6$, we achieved 94.98% of the pairs with the same classification.

Complexity comparison. In our average key rank distinguisher, for each pair, we perform the partial decryption of two ciphertexts, and a table lookup in aDDT. In the partial decryption, we enumerate the 2^{12} keys affecting the rightmost 13 bits of δl_{nr-1} covered by our mask. Therefore, the complexity of our distinguisher is 2^{13} one-round SPECK-32/64 decryptions, and 2^{13} table lookups. Comparing its complexity with Gohr's distinguishers is not trivial, as the operations involved are different. Gohr evaluates the complexity of his neural key recovery by their runtime and an estimation of the number of speck encryptions that could be performed at the same time on a GPU implementation. We propose to use the number of floating point multiplications performed by the neural network instead. Let I and O respectively denote the number of inputs and outputs to one layer. The computational cost of going through a dense layer is $I \cdot O$ multiplications. For 1D-CNN with kernel size $ks = 1$, a null padding, a stride equal to 1 and F filters, with input size (I, T) the cost is computed as $I \cdot F \cdot T$ multiplications. With the same input but with kernel size $ks = 3$, a padding equal to 1, the cost is $I \cdot ks \cdot F \cdot T$ Applying these formulas to Gohr's neural network, we obtain a total of $137280 \approx 2^{17.07}$ multiplications. Note that we do not account for batch normalizations and additions, which are dominated by the cost of the multiplications. Using this estimation, it seems that our distinguisher is slightly better in terms of complexity.

Table 7: Accuracy of the average key rank distinguisher with a mask value of `0xff8f/ff8f`.

N_5	**Accuracy**	**TPR**	**TNR**
5	92.98%	90.76%	95.22%
6	78.79%	72.53%	85.07%
7	60.28%	55.31%	65.24%

Table 8: Closeness of the outputs of N_5 and average key rank distinguisher.

		N_r output	
		≥ 0.5	**< 0.5**
AKR Dist	**1**	46.6%	1.48%
	0	0.953%	51.0%

4.6 Discussion

Even though Gohr trained a neural distinguisher with a fixed input difference, it is unfair to compare the accuracy of neural distinguisher to that of a pure differential cryptanalysis (with the use of DDT), since there are alternative cryptanalysis methods that the neural distinguisher may have learned. The experiments performed indicate that while Gohr's neural distinguishers did not rely much on difference at the nr round, they rely strongly on the differences at round $nr - 1$ and even more strongly at round $nr - 2$. These results support the hypothesis that the neural distinguisher may learn differential-linear cryptanalysis [13] in the case of SPECK. While we did not present any attacks here, using the MILP model shown in [9], we verified that there are indeed many linear relations with large biases for 2 to 3 rounds.

Unlike traditional linear cryptanalysis, which usually use independent characteristics or linear hull involving the same plaintext and ciphertext bits, a well-trained neural network is able to learn and exploit several linear characteristics while taking into account their dependencies and correlations.

We believe that neural networks find the easiest way to achieve the best accuracy. In the case of SPECK, it seems that differential-linear cryptanalysis would be a good fit since it requires less data and the truncated differential has a very high probability. Thus, we think that neural networks have the ability to efficiently learn short but strong differential, linear or differential-linear characteristics for small block ciphers for a small number of rounds.

4.7 Application to AES-2-2-4 [7]

We are also interested in the capabilities of the neural distinguishers on a Substitution-Permutation Network (SPN) cipher. We chose a small scale variant of AES from [7] with the parameters: $r = 2, c = 2, e = 4$. We chose this cipher as it has a small state size, which could be exhaustively searched through. AES-2-2-8 would be a good choice as it also has a state size of 32-bit, however, our distinguishers are not able to learn anything significant. We trained AES-2-2-4 with 2^{15} pairs, starting with an input difference of $(1, 0, 0, 1)$. This input difference was chosen such that only after two rounds, all S-boxes will be active. We trained them for 3 rounds and obtained an accuracy of 61.0%. In contrast, we use the same number of pairs, we trained an aDDT distinguisher and we obtained an accuracy of 62.3%.

To show the possibilities of relying purely on differences, we perform an experiment similar to Experiment A. With the trained neural distinguisher, we exhaust all possible 16-bit differences and we generate 100 random pairs for each difference. Next, we feed the pairs to the neural distinguisher and count the number of pairs in each basket of score: $[0.0 - 0.1), [0.1 - 0.2), ..., [0.9 - 1.0]$. Our result shows that for each differential, the 100 random pairs form a cluster about a center similar to a Gaussian distribution. These results seem to suggest the nature of the neural distinguisher for AES-2-2-4 is one that relies fully on differential: giving a confidence interval based on just the difference.

5 Interpretation of Gohr's Neural Network: a Machine Learning Perspective

In this section, we are exploring the following practical question:

Can Gohr's neural network be replaced by a strategy inspired by both differential cryptanalysis and machine learning?

We will demonstrate here that this is possible. First of all, it should be emphasized that DNNs often outperform mathematical modeling or standard machine learning approaches in supervised data-driven settings, especially on high-dimensional data. It seems to be the case because correlations found between input and output pairs during DNN training lead to more relevant characteristics than those found by experts. In other words, Gohr's neural distinguisher seems to be capable of finding a property $\mathcal{P}$ currently unknown by cryptanalysts. One may ask if we could experimentally approach this unknown property $\mathcal{P}$ that encodes the neural distinguisher behavior, using both machine learning and cryptanalysis expertise. With this question in mind, we propose our best estimate with a focus on 5 and 6 SPECK-32/64 rounds where the DNN achieves accuracies of 92.9% and 78.8% in a real/random distinction setting and where the full DDT approach can achieve accuracies of 91.1% and 75.8%. In our best setting, we reach accuracy values of 92.3% and 77.9%.

Section 3 discusses in detail how Gohr's neural distinguisher is modeled in three blocks. Our objective here is to replace each of these individual blocks by a more interpretable one, coming either from machine learning or from the cryptanalysts' point of view. This work is thus the result of the collaboration between two worlds addressing the open question of deep learning interpretability. In the course of the study, we set forth and challenged four conjectures to estimate the property $\mathcal{P}$ learned by the DNN as detailed below.

5.1 Four Conjectures

Conjectures 2 & 3 aim to uncover Block 3 behavior. Conjecture 4 tackles Block 1 while Conjecture 5 concerns Block 2-i.

The DNN can not be entirely replaced by another machine learning model. Ensemble-based machine learning models such as random forests [4] and gradient boosting decision trees [8] are accurate and easier to interpret than DNNs [14]. Nevertheless, DNNs outperform ensemble-based machine learning models for most tasks on high-dimensional data such as images. However, with only 64 bits of input, we could legitimately wonder whether the DNN could be replaced by another ensemble-based machine learning model. Despite our small size problem, our experiments reveal that other models significantly decrease the accuracy.

Conjecture 2. Gohr's neural network outperforms other non-neuronal network machine learning models.

Experiment. To challenge this conjecture, we tested multiple machine learning models, such as Random Forest (RF), Light Gradient Boosting Machine (LGBM), Multi-Layer Perceptron (MLP), Support Vector Machine (SVM) and Linear Regression (LR). They all performed equally. For the rest of this paper, we will only consider LGBM [12] as an alternative ensemble classifier for DNN and MLP. LGBM is an extension of Gradient Boosting Decision Tree (GBDT) [8] and we fixed our choice on it because it is accurate, interpretable and faster to train than RF or GBDT. In support of our conjecture, we established that the accuracy for the LGBM model is significantly lower than the one of the DNN when the inputs are (C_l, C_r, C'_l, C'_r), see third column of Table 9.

Table 9: A comparison of the neural distinguisher and LGBM model for 5 round, for 10^6 samples generated of type (C_l, C_r, C'_l, C'_r).

N_5	D_5	**LGBM as classifier for the original input**	**LGBM as classifier for the 512-feature**	**LGBM as classifier for the 64-feature**
92.9%	91.1%	76.34% ± 2.62	91.49% ± 0.09	92.36% ± 0.07

The final MLP block is not essential. As described above, we can not replace the entire DNN with another non-neuronal machine learning model that is easier to interpret. However, we may be able to replace the last block (Block

3) of the neural distinguisher performing the final classification, by an ensemble model.

Conjecture 3. The MLP block of Gohr's neural network can be replaced by another ensemble classifier.

Experiment. We successfully exchanged the final MLP block for a LGBM model. The reasons for choosing LGBM as a non-linear classifier were detailed in the previous experiment paragraph. The first attempt is a complete substitution of Block 3, taking the 512-dimension output of Block 2-10 as input. In the fourth column of Table 9, we observe that this experiment leads to much better results than the one from Conjecture 2, and even better results than the classical DDT method D_5 (+0.39%). To further improve the accuracy, we implemented a partial substitution, taking only the 64-dimension output of the first layer of the MLP as input. As can be seen in the fifth column from Table 9, the accuracy with those inputs is now much closer to the DNN accuracy. In both cases, the accuracy is close to the neural distinguisher, supporting our conjecture. At this point, in order to grasp the unknown property $\mathcal{P}$, one needs to understand the feature vector at the residuals' output.

The linear transformation on the inputs. We saw in Section 3 that Block 1 performs a linear transformation on the input. By looking at the weights of the DNN first convolution, we observe that it contains many opposite values. This indicates that the DNN is looking for differences between the input features. Consequently, we propose the following conjecture.

Conjecture 4. The first convolution layer of Gohr's neural network transforms the input (C_l, C_r, C'_l, C'_r) into $(\varDelta L, \varDelta V, V_0, V_1)$ and a linear combination of those terms.

Experiments. As the inputs of the first convolution are binary, we could formally verify our conjecture. By forcing to one all non-zero values of the output of this layer, we calculated the truth-table of the first convolution. We thus obtained the boolean expression of the first layer for the 32 filters. We observed that eight filters were empty and the remaining twenty-four filters were simple. The filter expressions are provided in Table 14 in the Supplementary Materials.

However, one may argue that setting all non-zero values to one is an over-simplified approach. Therefore, we replaced the first ReLU activation function by the Heaviside activation function, and then we retrained the DNN. Since the Heaviside function binarizes the intermediate value (as in [28]), we can establish the formal expression of the first layer of the retrained DNN. This second DNN had the same accuracy as the first one and almost the same filter boolean expression.

Finally, we trained the same DNN with the following entries $(\varDelta L, \varDelta V, V_0, V_1)$. Using the same method as before, we established the filters' boolean expressions. This time, we obtained twenty five null filters and seven non-null filters, with the following expressions: ΔL, $\overline{V_0} \wedge V_1$, $\overline{\Delta L}$, ΔL, $\overline{V_0} \wedge \overline{V_1}$, $\Delta L \wedge \Delta V$, $\overline{\Delta L} \wedge \overline{\Delta V}$. These observations support conjecture 4. Therefore, we kept only $(\Delta L, \Delta V, V_0, V_1)$ as inputs for our pipeline.

The masked output distribution table. With regards to the remaining residual block replacement, our first assumption is that the DNN calculates a shape close to the DDT in that residual block. However, two major properties of the neural distinguisher prevent us from assuming that it is a DDT in the classical sense of the term. The first property, as explained in Section 3, is that the neural distinguisher does not only rely on the difference distribution to distinguish real pairs as presented in Table 2. The second specificity is that the DNN has only approximately 100,000 floating parameters to perform classification, which can be considered as size efficient. Our second assumption is therefore that the DNN is able to compress the distribution table. We introduce the following definitions.

Output Distribution Table (ODT). We propose to compute a distribution table on the values $(\Delta L, \Delta V, V_0, V_1)$ directly, instead of doing so on the difference of the ciphertext pair $(C_l \oplus C'_l, C_r \oplus C'_r)$. We call this new table an Output Distribution Table (ODT) and it can be seen as a generalization of the DDT. The entries of the ODT are 64 bits, which is not tractable for 10^7 samples. Also, the DNN has only 100,000 parameters. The DNN is therefore able to compress the ODT.

Masked Output Distribution Table (M-ODT). A compressed ODT means that the input is not 64 bits, but instead hw bits, where hw represents the Hamming weight of the mask. Let us consider a mask $M \in \mathcal{M}_{hw}$ with $\mathcal{M}_{hw}$ the ensemble of 64-bits masks with Hamming weight hw and $M = (M_1, M_2, M_3, M_4)$, with M_i a 16-bit mask. Compressing the ODT therefore means applying the M mask to all inputs. In our case, with $I = (\Delta L, \Delta V, V_0, V_1)$, we get $I_M = (\Delta L \wedge M_1, \Delta V \wedge M_2, V_0 \wedge M_3, V_1 \wedge M_4) = I \wedge M$, before computing the ODT. By calculating that way, the number of ODT entries per mask decreases. It becomes a function that depends only on hw and on the bit positions in the masks. It is therefore a more compact representation of the complete ODT. However, it turns out that if we consider only one mask, we get only one value per sample to perform the classification: $P(\texttt{Real}|I_M)$, while the DNN has a final vector size of 512. We considered several masks. Thus, by defining the ensemble $R_M \in \mathcal{M}_{hw}$, the set of relevant masks of $\mathcal{M}_{hw}$, we can calculate for a specific input $I = (\Delta L, \Delta V, V_0, V_1)$ the probability $P(\texttt{Real}|I_M), \forall M \in R_M$. Then, we concatenate all the probabilities into a feature vector of size $m = |R_M|$. We get the feature F for the input I: $F = \begin{pmatrix} P(\texttt{Real}|I_{M1}) & P(\texttt{Real}|I_{M2}) & \cdots & P(\texttt{Real}|I_{Mm}) \end{pmatrix}^T$. We are now able to propose the final conjecture.

Conjecture 5. The neural distinguisher internal data processing of Block 2-i can be approached by:

1. Computing a distribution table for input $(\Delta L, \Delta V, V_0, V_1)$.

2. Finding several relevant masks and applying them to the input in order to compress the output distribution table.

We abbreviate M-ODT this Masked-Output Distribution Table. Thus, the feature vector of the DNN can be replaced by a vector where each value represents the probability stored in the M-ODT for each mask.

This approach enables us to replace Block 2-i of the DNN. Though, we still need to clarify how to get the R_M ensemble.

Extracting masks. Based on local interpretation methods, we can extract these masks from the DNN. Indeed, these methods consist of highlighting the most important bits of the entries for classification. Thus, by sorting the entries according to their score and by applying these local interpretation methods, we can obtain the relevant masks.

5.2 Approximating the Expression of the Property $\mathcal{P}$

From our conjectures, we hypothesized that we can approximate the unknown property $\mathcal{P}$ that encodes the neural distinguisher behavior by the following:

- Changing (C, C') into $I = (\Delta L, \Delta V, V_0, V_1)$.
- Changing the 512-feature vector of the DNN by the feature vector of probabilities $F = \left(P(\texttt{Real}|I_{M1}) \; P(\texttt{Real}|I_{M2}) \cdots P(\texttt{Real}|I_{Mm}) \right)^T$.
- Changing the final MLP block by the ensemble machine learning model LGBM.

These points stand respectively for Block 1, Block 2-i and Block 3.

5.3 Implementation

In this section and based on the verified conjectures, we are describing the stepwise implementation of our method. We consider that we have a DNN formed with 10^7 data of type $(\Delta L, \Delta V, V_0, V_1)$ for 5 and 6 rounds of SPECK-32/64. We developed a three-step approach:

1. Extraction of the masks from the DNN with a first dataset.
2. Construction of the M-ODT with a second dataset.
3. Training of the final classifier from the probabilities stored in the M-ODT with a third dataset.

Mask extraction from the DNN. We first ranked 10^4 real samples according to DNN score, as described in Section 4.1, in order to estimate the masks from these entries. We used multiple local interpretation methods: Integrated Gradients [26], DeepLift [22], Gradient Shap [15], Saliency maps [23], Shapley Value [5], and Occlusion [27]. These methods score each bit according to their importance for the classification. Following averaging by batch and by method, there were two possible ways to move forward. We could either assign a Hamming weight or else set a threshold above which all bits would be set to one. After a wide range of experiments, we chose the first option and set the Hamming weight to sixteen and eighteen (which turned out to be the best values in our testing). This approach allowed us to build the ensemble R_M of the relevant masks.

Implementation details. We used the captum library[6] which brings together multiple methods on local interpretation. The dataset is divided into batches of size about 2,500 and grouped by scores. The categories we used were: scores from 1 to 0.9 (about 2,000 samples), scores from 0.9 to 0.5 (about 500 samples), scores from 1 to 0.8 (about 2,100 samples) and scores from 1 to 0.5 (about 2,500 samples). This way, one score per method could be derived for each bit of each sample. We then proposed several methods to average these importance scores by bit of category: the sum of absolute values, the median of absolute values and the average of absolute values. Then, we took the sixteen and eighteen best values and we obtained a mask. There is one mask per score, one per local interpretation method and one per averaging method. On average, for 5,000 samples we generate about 100 relevant masks. Finally, with the methods available in scikit-learn [20], we ranked the features and so the masks according to their performance. After multiple repetitions of mask generation and selection at every time, we obtained 50 masks that are effective: they are provided in Table 15 in the Supplementary Materials. The final ensemble of masks is the addition of those 50 effective masks and the generated relevant masks.

Constructing the M-ODT. Once the ensemble R_M of relevant masks is determined, we compute the M-ODT. Algorithm D (in Supplementary Materials) describes our construction method which is similar to that of the DDT. The inputs of the algorithm include a second dataset composed of $n = 10^7$ real samples of type $I = (\Delta L, \Delta V, V_0, V_1)$, and the set of relevant masks R_M. The output is the M-ODT dictionary with the mask as first key, the masked input as second key, and $P(\texttt{Real}|I \wedge M) = P(\texttt{Real}|I_M)$ as value.

The M-ODT dictionary is constructed as follow: first, for each mask M in R_M, we compute the corresponding masked-dataset $\mathcal{D}_M$ which is simply the operation $I_M = I \wedge M$ for all I in $\mathcal{D}$. Secondly we compute a dictionary U with key the element of D_M and with value the occurrences number of that element in D_M. Then, we compute for all element I_M in $\mathcal{D}_M$ the probability:

$$P(\texttt{Real}|I_M) = \frac{P(I_M|\texttt{Real})P(\texttt{Real})}{P(I_M|\texttt{Real})P(\texttt{Real}) + P(I_M|\texttt{Random})P(\texttt{Random})}$$

[6] https://github.com/pytorch/captum

with $P(\texttt{Real}) = P(\texttt{Random}) = 0.5$, $P(I_M|\texttt{Random}) = 2^{-HW(M)}$, $HW(M)$ being the Hamming weight of M and $P(I_M|\texttt{Real}) = \frac{1}{n} \times U[I_M]$. Finally we update M-ODT as follow: M-ODT$[M][I_M] = P(\texttt{Real}|I_M)$.

Training the classifier on probabilities. Upon building the M-ODT, we can start training the classifier. Given a third dataset $\mathcal{D} = \{(input_0, y_0)... (input_n, y_n)\}$, with $input_j$ a sample of type (C, C'), transformed into $(\Delta L, \Delta V, V_0, V_1)$ and the label $y_j \in [0,1]$, with $n = 10^6$, we first compute the feature vector $F_j = \begin{pmatrix} P(\texttt{Real}|I_j \wedge M1) & P(\texttt{Real}|I_j \wedge M2) \cdots P(\texttt{Real}|I_j \wedge Mm) \end{pmatrix}^T$ for all inputs and for $m = |R_M|$. Next, we determined the optimal θ parameters for the g_θ model according to Equation 1, with L being the square loss. Here, the g_θ classifier is Light Gradient Boosting Machine (LGBM) [12].

Implementation details. Feature vectors are standardized. Model hyper-parameters fine-tuning has been achieved by grid search. Results were obtained by cross-validation on 20% of the train set and the test set had 10^5 samples. Finally, results are obtained on the complete pipeline for three different seeds, five times for every seed.

5.4 Results

The M-ODT pipeline was implemented with numpy, scikit-learn [20] and pytorch [19]. The project code can be found at this URL address[7]. Our work station is constituted of a GPU Nvidia GeForce GTX 970 with 4043 MiB memory and four Intel core i5-4460 processors clocked at 3.20GHz.

General results. Table 10 shows accuracies of the DDT, the DNN and our M-ODT pipeline on 5 and 6-round reduced SPECK-32/64 for 1.1×10^7 generated samples. When compared to DNN and DDT, our M-ODT pipeline reached an intermediate performance right below DNN. The main difference is the true positive rate which is higher in our pipeline (this can be explained by the fact that our M-ODT preprocessing only considers real samples). All in all, our M-ODT pipeline successfully models the property $\mathcal{P}$.

Matching. Table 11 summarizes the results of the quantitative correspondence studies for the prediction between the two models. We compared the DNN trained on samples type $(\Delta L, \Delta V, V_0, V_1)$ to our M-ODT pipeline. On 5 rounds, we obtained a rate of 97.5% identical predictions. In addition, 91.3% were both identical and equal to the label. On 6 rounds, matching prediction reduces down to 93.1%.

We thus demonstrated that our method advantageously approximates the performance of the neural distinguisher. With an initial linear transformation

[7] https://github.com/AnonymousSubmissionEuroCrypt2021/A-Deeper-Look-at-Machine-Learning-Based-Cryptanalysis

Table 10: A comparison of Gohr's neural network, the DDT and our M-ODT pipeline accuracies for around 150 masks generated each time, with input $(\Delta L, \Delta V, V_0, V_1)$, LGBM as classifier and 1.1×10^7 samples generated in total. TPR and TNR refers to true positive and true negative rate respectively.

Rd	**Distinguisher**	**Accuracy**	**TPR**	**TNR**
5	D_5	91.1%	87.7%	94.7%
	N_5	$92.9\% \pm 0.05$	$90.4\% \pm 0.08$	$95.4\% \pm 0.06$
	M-ODT (Ours)	$92.3\% \pm 0.08$	$95.5\% \pm 0.09$	$89.1\% \pm 0.2$
6	D_6	75.8%	68.0%	83.7%
	N_6	$78.8\% \pm 0.08$	$72.4\% \pm 0.01$	$85.3\% \pm 0.1$
	M-ODT (Ours)	$77.9\% \pm 0.1$	$85.2\% \pm 0.1$	$70.6\% \pm 0.2$

on the inputs, computing a M-ODT for a set of masks extracted from the DNN and then classifying the resulting feature vector with LGBM, we achieved an efficient yet more easily interpretable approach than Gohr distinguishers. Indeed, DNN obscure features are simply approached in our pipeline by $F = \left(P(\texttt{Real}|I_{M1})\ P(\texttt{Real}|I_{M2}) \cdots P(\texttt{Real}|I_{Mm}) \right)^T$. Finally, we interpret the performance of the classifier globally (i.e. retrieving the decision tree) and locally (i.e. deducing which feature played the greatest role in the classification for each sample) as in [14]. Those results are not displayed as they are beyond the scope of the present work, but they can be found in the project code.

Table 11: A comparison of Gohr's neural network predictions and our M-ODT pipeline predictions for around 150 masks generated each time, with input $(\Delta L, \Delta V, V_0, V_1)$, LGBM as classifier and 1.1×10^7 samples generated in total.

Nr	**Model**	**Accuracy**	**Matching**	**Matching & equal to label**
5	N_5	92.9%	97.5% ± 0.06	91.3% ± 0.08
	M-ODT (Ours)	92.3%		
6	N_6	78.8%	93.1% ± 0.07	75.3% ± 0.11
	M-ODT (Ours)	77.9%		

5.5 Application to SIMON Cipher

In order to check whether our approach could be generalized to other cryptographic primitives, we evaluated our M-ODT method on 8 rounds of SIMON-32/64 block cipher. Implementing the same pipeline, we enjoyed a 82.2% accuracy for the classification, whereas the neural distinguisher achieves 83.4% accuracy. In addition, the matching rate between the two models was up to

92.4%. The slight deterioration in the results of our pipeline for SIMON can be explained by the lack of efficient masks as introduced in Section 5.3 for SPECK.

5.6 Discussions

From the cryptanalysts' standpoint, one important aspect of using the neural distinguisher is to uncover the property $\mathcal{P}$ learned by the DNN. Unfortunately, while being powerful and easy to use, Gohr's neural network remains opaque.

Our main conjecture is that the 10-layer residual blocks, considered as the core of the model, are acting as a compressed DDT applied on the whole input space. We model our idea with a Masked Output Distribution Table (M-ODT). The M-ODT can be seen as a distribution table applied on masked outputs, in our case $(\varDelta L, \varDelta V, V_0, V_1)$, instead of only the difference $(C_l \oplus C'_l, C_r \oplus C'_r)$. By doing so, features are no longer abstract as in the neural distinguisher. In our pipeline, each one of the features is a probability for the sample to be real knowing the mask and the input. In the end, with our M-ODT pipeline, we successfully obtained a model which has only -0.6% difference accuracy with the DNN and a matching of 97.3% on 5 rounds of SPECK-32/64. Additional analysis of our pipeline (e.g. masks independence, inputs influence, classifiers influence) are available into the project code. To the best of our knowledge, this work is the first successful attempt to exhibit the underlying mechanism of the neural distinguisher. However, we note that a minor limitation of our method is that it still requires the DNN to extract the relevant masks during the preparation of the distinguisher. Since it is only during preparation, this does not remove anything with regards to the interpretability of the distinguisher. Future work will aim at computing these masks without DNN. All in all, our findings represent an opportunity to guide the development of a novel, easy-to-use and interpretable cryptanalysis method.

6 Improved Training Models

While in the two previous sections we focused on understanding how the neural distinguisher works, here we will explain how one can outperform Gohr's results. The main idea is to create batches of ciphertext inputs instead of pairs.

We refer to batch input of size B, a group of B ciphertexts that are constructed from the same key. Here, we can distinguish two ways to train and evaluate the neural distinguisher pipeline with batch input. The straightforward one is to evaluate the neural distinguisher score for each element of the batch and then to take the median of the results. The second is to consider the whole batch as a single input for a neural distinguisher. In order to do so, we used 2-dimensional CNN (2D-CNN) where the channel dimension is the features $(\varDelta L, \varDelta V, V_0, V_1)$. We should point out that, for sake of comparability with Gohr's work, we maintained the product of the training set size by the batch size to be equal to 10^7. Both batch size-based challenging methods yielded similar accuracy values (see Table 12). Notably, in both cases, we enjoyed 100% accuracy on 5 and 6 rounds with batch sizes 10 and 50 respectively.

Table 12: Study of the batch size methods on the accuracies with ($\mathit{\Delta} L$, $\mathit{\Delta} V$, V_0, V_1) as input for 5 and 6 rounds.

Rounds	5			6			
Batch input size	1	5	10	1	5	10	50
Averaging Method	92.9%	99.8%	100%	78.6%	95.41%	99.0%	100%
2D-CNN Method	-	99.4%	100%	-	93.27%	97.7%	100%

Considering these encouraging outcomes, we extended the method to 7 rounds. As the 7-round training is more sophisticated and the two previous methods are equivalent, we decided to only apply the first method (the averaging one), because it requires to train only one neural distinguisher. Results given in Table 13 confirm our previous findings: with a batch size of 100, we obtain 99.7% accuracy on 7 rounds. This remarkable outcome demonstrates the major improvement of our batch strategy over those from earlier Gohr's work.

Table 13: Study of the averaging batch size method on the 7-round accuracies with ($\mathit{\Delta} L$, $\mathit{\Delta} V$, V_0, V_1) as input.

Batch input size	1	5	10	50	100
Averaging Method	61.2%	73.5%	80.8%	96.7%	99.7%

Conclusion

In this article, we proposed a thorough analysis of Gohr's deep neural network distinguishers of SPECK-32/64 from CRYPTO'19. By carefully studying the classified sets, we managed to uncover that these distinguishers are not only basing their decisions on the ciphertext pair difference, but also the internal state difference in penultimate and antepenultimate rounds. We confirmed our findings by proposing pure cryptanalysis-based distinguishers on SPECK-32/64 that match Gohr's accuracy. Moreover, we also proposed a new simplified pipeline for Gohr's distinguishers, that could reach the same accuracy while allowing a complete interpretability of the decision process. We finally gave possible directions to even improve over Gohr's accuracy.

Our results indicate that Gohr's neural distinguishers are not really producing novel cryptanalysis attacks, but more like optimizing the information extraction with the low-data constraints. Many more distinguisher settings, machine learning pipelines, types of ciphers should be studied to have a better understanding of what machine learning-based cryptanalysis might be capable of. Yet, we foresee that such tools could become of interest for cryptanalysts and designers to easily and generically pre-test a primitive for simple weaknesses.

Our work also opens interesting directions with regards to interpretability of deep neural networks and we believe our simplified pipeline might lead to better interpretability in other areas than cryptography.

Acknowledgements

The authors are grateful to the anonymous reviewers for their insightful comments that improved the quality of the paper. The authors are supported by the Temasek Laboratories NTU grant DSOCL17101. We would like to thank Aron Gohr for pointing out that the differential characteristics mentioned in the attacks of Dinur's [6] have been extended by one free round, thus, our previous suggestion of extending Dinur's attack by one round is invalid.

References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential cryptanalysis of round-reduced simon and speck. In: Fast Software Encryption - FSE 2014. LNCS, vol. 8540, pp. 525–545. Springer (2014)
2. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The SIMON and SPECK families of lightweight block ciphers. IACR Cryptol. ePrint Arch. 2013, 404 (2013), `http://eprint.iacr.org/2013/404`
3. Biryukov, A., Roy, A., Velichkov, V.: Differential analysis of block ciphers SIMON and SPECK. In: Cid, C., Rechberger, C. (eds.) Fast Software Encryption - FSE 2014. LNCS, vol. 8540, pp. 546–570. Springer (2014)
4. Breiman, L.: Random forests. Machine learning 45(1), 5–32 (2001)
5. Castro, J., Gómez, D., Tejada, J.: Polynomial calculation of the shapley value based on sampling. Computers & Operations Research 36(5), 1726–1730 (2009)
6. Dinur, I.: Improved differential cryptanalysis of round-reduced speck. In: Selected Areas in Cryptography - SAC 2014. pp. 147–164 (2014)
7. Duan, X., Yue, C., Liu, H., Guo, H., Zhang, F.: Attitude tracking control of small-scale unmanned helicopters using quaternion-based adaptive dynamic surface control. IEEE Access 9, 10153–10165 (2021), `https://doi.org/10.1109/ACCESS.2020.3043363`
8. Friedman, J.H.: Greedy function approximation: a gradient boosting machine. Annals of statistics pp. 1189–1232 (2001)
9. Fu, K., Wang, M., Guo, Y., Sun, S., Hu, L.: Milp-based automatic search algorithms for diff erential and linear trails for speck. IACR Cryptol. ePrint Arch. 2016, 407 (2016)
10. Gerault, D., Minier, M., Solnon, C.: Constraint programming models for chosen key differential cryptanalysis. In: Principles and Practice of Constraint Programming. pp. 584–601. Springer (2016)
11. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. In: Advances in Cryptology - CRYPTO 2019. LNCS, vol. 11693, pp. 150–179. Springer (2019)
12. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in neural information processing systems. pp. 3146–3154 (2017)
13. Langford, S.K., Hellman, M.E.: Differential-linear cryptanalysis. In: Advances in Cryptology - CRYPTO '94. LNCS, vol. 839, pp. 17–25. Springer (1994)
14. Lundberg, S.M., Erion, G., Chen, H., DeGrave, A., Prutkin, J.M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., Lee, S.I.: Explainable ai for trees: From local explanations to global understanding. arXiv preprint arXiv:1905.04610 (2019)

15. Lundberg, S.M., Lee, S.I.: A unified approach to interpreting model predictions. In: Advances in neural information processing systems. pp. 4765–4774 (2017)
16. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Security, Privacy, and Applied Cryptography Engineering - SPACE 2016. pp. 3–26 (2016)
17. Mouha, N., Preneel, B.: A proof that the ARX cipher salsa20 is secure against differential cryptanalysis. IACR Cryptol. ePrint Arch. 2013, 328 (2013), http://eprint.iacr.org/2013/328
18. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Information Security and Cryptology - Inscrypt 2011. pp. 57–76 (2011)
19. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. In: Advances in neural information processing systems. pp. 8026–8037 (2019)
20. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in python. the Journal of machine Learning research 12, 2825–2830 (2011)
21. Rivest, R.L.: Cryptography and machine learning. In: Advances in Cryptology - ASIACRYPT '91. pp. 427–439 (1991)
22. Shrikumar, A., Greenside, P., Kundaje, A.: Learning important features through propagating activation differences. arXiv preprint arXiv:1704.02685 (2017)
23. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv preprint arXiv:1312.6034 (2013)
24. Song, L., Huang, Z., Yang, Q.: Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: Information Security and Privacy - ACISP 2016. pp. 379–394 (2016)
25. Sun, S., Gérault, D., Lafourcade, P., Yang, Q., Todo, Y., Qiao, K., Hu, L.: Analysis of aes, skinny, and others with constraint programming. IACR Trans. Symmetric Cryptol. 2017(1), 281–306 (2017)
26. Sundararajan, M., Taly, A., Yan, Q.: Axiomatic attribution for deep networks. arXiv preprint arXiv:1703.01365 (2017)
27. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: European conference on computer vision. pp. 818–833. Springer (2014)
28. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. CoRR abs/1606.06160 (2016), http://arxiv.org/abs/1606.06160

Supplementary Materials

A Pseudocode for aDDT and Average Key Rank Distinguisher

Algorithm 1 Function used to build the approximated DDT from a given number of random samples and a bitmask.

```
1: function BuildApproxDDT($nr, np, m$)    ▷ where $nr$ - Number of rounds, $np$ - No. of pairs, $m$ - bitmask
2:     $aDDT \leftarrow ZEROS(2^{32})$
3:     for $i = 1$ to $np$ do
4:         $p \leftarrow rand(0, 2^{32} - 1)$
5:         $p' \leftarrow p \oplus$ 0x0040/0000
6:         $key \leftarrow rand(0, 2^{64} - 1)$
7:         $l_{nr}, r_{nr} \leftarrow ENC_{key}(p, nr)$;
8:         $l'_{nr}, r'_{nr} \leftarrow ENC_{key}(p', nr)$
9:         $\delta_l \leftarrow l_{nr} \oplus l'_{nr}$;
10:        $\delta_v \leftarrow l_{nr} \oplus r_{nr} \oplus l'_{nr} \oplus r'_{nr}$
11:        $aDDT[(\delta_l || \delta_v) \wedge m] = aDDT[(\delta_l || \delta_v) \wedge m] + \frac{1}{n}$
12:    end for
13:    return $aDDT$
14: end function
```

Algorithm 2 Pseudocode for the average key rank differential distinguisher.

```
1: function AKRDD($nr, C, C', m$)    ▷ where $nr$ - Number of rounds, $C$, $C'$ - two ciphertexts, $m$ - bitmask
2:     $aDDT \leftarrow BuildApproxDDT(nr - 1, 10^7, m)$
3:     $score \leftarrow 0$
4:     for $key = 0$ to $2^{12} - 1$ do
5:         $l_{nr-1}, r_{nr-1} \leftarrow DEC_{key}(C, 1)$;    ▷ Decrypt the last round with the subkey $key$
6:         $l'_{nr-1}, r'_{nr-1} \leftarrow DEC_{key}(C', 1)$
7:         $\delta_l \leftarrow l_{nr-1} \oplus l'_{nr-1}$;
8:         $\delta_v \leftarrow l_{nr-1} \oplus r_{nr-1} \oplus l'_{nr-1} \oplus r'_{nr-1}$
9:         $score += aDDT[\delta_l || \delta_v]$
10:    end for
11:    return $(\frac{score}{2^{12}} > 2^{-HW(m)})$    ▷ HW(x) refers to the Hamming weight of x
12: end function
```

B Expression of the Filter After First 1D-CNN According to Methods from Section 5.1

Table 14: Expression of the filter after first 1D-CNN for type input (C_l, C_r, C'_l, C'_r).

Filter number	**Boolean expression**
1	$(\overline{C_l} \wedge \overline{C'_l})$ OR $(C_r \wedge \overline{C_l} \wedge \overline{C'_r})$ OR $(C_r \wedge \overline{C'_l} \wedge \overline{C'_r})$
2	$C_l \wedge C'_r \wedge \overline{C'_l}$
3	$(C_r \wedge \overline{C'_l} \wedge \overline{C'_r})$ OR $(\overline{C_l} \wedge \overline{C'_l} \wedge \overline{C'_r})$
4	$(C_l \wedge C_r \wedge \overline{C'_l})$ OR $(C_l \wedge C_r \wedge \overline{C'_r})$ OR $(C_l \wedge \overline{C'_l} \wedge \overline{C'_r})$ OR $(C_r \wedge \overline{C'_l} \wedge \overline{C'_r})$
5	Null
6	$(C'_l \wedge \overline{C_l} \wedge \overline{C'_r})$ OR $(C'_l \wedge \overline{C_r} \wedge \overline{C'_r})$
7	$(C'_r \wedge \overline{C_l} \wedge \overline{C'_l})$ OR $(\overline{C_l} \wedge \overline{C_r} \wedge \overline{C'_l})$
8	$(\overline{C_l} \wedge \overline{C_r} \wedge \overline{C'_l})$ OR $(\overline{C_l} \wedge \overline{C'_l} \wedge \overline{C'_r})$ OR $(\overline{C_r} \wedge \overline{C'_l} \wedge \overline{C'_r})$
9	$(C'_l \wedge C'_r)$ OR $(C'_l \wedge \overline{C_l})$ OR $(C_r \wedge C'_r \wedge \overline{C_l})$
10	Null
11	$(C'_l \wedge C'_r \wedge \overline{C_l})$ OR $(C'_l \wedge C'_r \wedge \overline{C_r})$ OR $(C'_l \wedge \overline{C_l} \wedge \overline{C_r})$
12	Null
13	$(C_l \wedge C'_l)$ OR $(C_l \wedge C_r \wedge \overline{C'_r})$ OR $(C_r \wedge C'_l \wedge \overline{C'_r})$
14	$(C_l \wedge \overline{C_r} \wedge \overline{C'_l})$ OR $(C_l \wedge \overline{C_r} \wedge \overline{C'_r})$ OR $(C_l \wedge \overline{C'_l} \wedge \overline{C'_r})$ OR $(\overline{C_r} \wedge \overline{C'_l} \wedge \overline{C'_r})$
15	$(C_l \wedge \overline{C'_l})$ OR $(C_l \wedge C_r \wedge C'_r)$ OR $(C_r \wedge C'_r \wedge \overline{C'_l})$
16	Null
17	$C_l \wedge C_r \wedge \overline{C'_l}$
18	$(C_l \wedge C'_l)$ OR $(C'_l \wedge C'_r)$ OR $(C_l \wedge C_r \wedge C'_r)$
19	$C_l \wedge C'_l \wedge \overline{C_r}$
20	$(C_r \wedge C'_l \wedge \overline{C_l})$ OR $(C_r \wedge C'_l \wedge \overline{C'_r})$ OR $(C_r \wedge \overline{C_l} \wedge \overline{C'_r})$ OR $(C'_l \wedge \overline{C_l} \wedge \overline{C'_r})$
21	Null
22	$(C_l \wedge C'_l \wedge C'_r)$ OR $(C_l \wedge C'_l \wedge \overline{C_r})$ OR $(C_l \wedge C'_r \wedge \overline{C_r})$
23	$C_l \wedge \overline{C'_l}$
24	$C_l \wedge \overline{C_r} \wedge \overline{C'_l}$
25	Null
26	$\overline{C_l} \wedge \overline{C'_l}$
27	$(C'_r \wedge \overline{C'_l})$ OR $(\overline{C_l} \wedge \overline{C'_l})$ OR $(C_r \wedge C'_r \wedge \overline{C_l})$
28	$(C_r \wedge C'_l \wedge \overline{C_l})$ OR $(C_r \wedge C'_l \wedge \overline{C'_r})$ OR $(C_r \wedge \overline{C_l} \wedge \overline{C'_r})$ OR $(C'_l \wedge \overline{C_l} \wedge \overline{C'_r})$
29	$C'_l \wedge \overline{C_l}$
30	Null
31	$\overline{C_l} \wedge \overline{C_r}$
32	Null

C Expression of the 50 Efficient Masks for the M-ODT Pipeline

Table 15: Expression (in hexadecimal notation) of the 50 efficient masks for the M-ODT pipeline for samples generated of type $(\mathit{\Delta L}, \mathit{\Delta V}, V_0, V_1)$.

Name masks	**Mask 1**	**Mask 2**	**Mask 3**	**Mask 4**
(3844-15384-7692-7692)	0F04	3C18	1E0C	1E0C
(7939-31804-15360-15360)	1F03	7C3C	3C00	3C00
(3842-16020-516-7684)	0F02	3E94	0204	1E04
(1551-15420-12-12)	060F	3C3C	000C	000C
(7693-30768-14336-14352)	1E0D	7830	3800	3810
(25569-12039-0-0)	63E1	2F07	0000	0000
(6409-26738-10240-10240)	1909	6872	2800	2800
(1543-7708-3072-3072)	0607	1E1C	0C00	0C00
(7693-7228-24-24)	1E0D	1C3C	0018	0018
(6150-30780-12312-12312)	1806	783C	3018	3018
(6159-14396-6168-2072)	180F	383C	1818	0818
(7692-31792-11264-12312)	1E0C	7C30	2C00	3018
(7683-21564-15360-15360)	1E03	543C	3C00	3C00
(6159-15420-8196-4124)	180F	3C3C	2004	101C
(30768-24624-0-0)	7830	6030	0000	0000
(11280-8208-0-0)	2C10	2010	0000	0000
(33155-34691-12-12)	8183	8783	000C	000C
(3852-14384-7184-7184)	0F0C	3830	1C10	1C10
(527-5692-2068-2068)	020F	163C	0814	0814
(5647-14396-4112-6168)	160F	383C	1010	1818
(772-256-41066-41066)	0304	0100	A06A	A06A
(5644-0-7186-7186)	160C	0000	1C12	1C12
(7-7740-1038-1038)	0007	1E3C	040E	040E
(779-7958-2048-3588)	030B	1F16	0800	0E04
(49496-45811-3-0)	C158	B2F3	0003	0000
(7680-30780-7168-7168)	1E00	783C	1C00	1C00
(7680-30780-6144-6144)	1E00	783C	1800	1800
(15-7740-1030-1030)	000F	1E3C	0406	0406
(7692-30768-14336-12304)	1E0C	7830	3800	3010
(7695-31804-0-28)	1E0F	7C3C	0000	001C
(37304-21475-0-12)	91B8	53E3	0000	000C
(2823-32566-0-4)	0B07	7F36	0000	0004
(1025-13923-4611-9795)	0401	3663	1203	2643
(3855-32316-0-0)	0F0F	7E3C	0000	0000
(7951-32308-0-0)	1F0F	7E34	0000	0000
(3840-13372-3076-3076)	0F00	343C	0C04	0C04
(3842-7196-9216-9216)	0F02	1C1C	2400	2400
(54568-192-4632-4632)	D528	00C0	1218	1218
(7692-28728-6144-6144)	1E0C	7038	1800	1800
(37304-21475-2-140)	91B8	53E3	0002	008C
(3840-14384-3076-3076)	0F00	3830	0C04	0C04
(28704-24608-0-0)	7020	6020	0000	0000
(2569-13360-273-273)	0A09	3430	0111	0111
(7693-30768-14336-13328)	1E0D	7830	3800	3410
(7174-14396-2560-5144)	1C06	383C	0A00	1418
(1-47891-4865-785)	0001	BB13	1301	0311
(5292-1-51-51)	14AC	0001	0033	0033
(2831-16182-0-2048)	0B0F	3F36	0000	0800
(7692-30768-12288-13328)	1E0C	7830	3000	3410
(7-31731-0-6674)	0007	7BF3	0000	1A12

D Algorithm for Computing the M-ODT

Algorithm 3 Function used to construct the masked-output distribution table (M-ODT) from a dataset and a set of relevant masks.

```
 1: function BUILD M-ODT($\mathcal{D}, R_M$)
 2:     M-ODT = {}
 3:     $n = |\mathcal{D}|$
 4:     $P(\texttt{Real}) = 0.5$
 5:     $P(\texttt{Random}) = 0.5$
 6:     for all $M$ in $R_M$ do
 7:         $N_h(M)$ = number of 1 in $M$
 8:         $\mathcal{D}_M = \mathcal{D} \wedge M$                                   ▷ $\forall I \in \mathcal{D}, I_M = I \wedge M$
 9:         U = Unique($\mathcal{D}_M$)    ▷ Return a dictionary with key the element of $D_M$ and
    with value the number occurrence of that element in $D_M$
10:         for all $I_M$ in $\mathcal{D}_M$ do
11:             $P(I_M|\texttt{Random}) = 2^{-N_h(M)}$
12:             $P(I_M|\texttt{Real}) = \frac{1}{n} \times U[I_M]$
13:             $P(\texttt{Real}|I_M) = \frac{P(\texttt{Real})P(I_M|\texttt{Real})}{P(I_M|\texttt{Real})P(\texttt{Real})+P(I_M|\texttt{Random})P(\texttt{Random})}$
14:             M-ODT$[M][I_M] = P(\texttt{Real}|I_M)$
15:         end for
16:     end for
17:     return M-ODT
18: end function
```