An Assessment of Differential-Neural Distinguishers

Aron Gohr[1], Gregor Leander[2] and Patrick Neumann[2]

[1] Independent Researcher, aron.gohr@gmail.com
[2] Ruhr University Bochum, Bochum, Germany, firstname.lastname@rub.de

Abstract. Since the introduction of differential-neural cryptanalysis, as the machine learning assisted differential cryptanalysis proposed in [Goh19] is coined by now, a lot of followup works have been published, showing the applicability for a wide variety of ciphers. In this work, we set out to vet a multitude of differential-neural distinguishers presented so far, and additionally provide general insights.
Firstly, we show for a selection of different ciphers how differential-neural distinguishers for those ciphers can be (automatically) optimized, also providing guidance to do so for other ciphers as well. Secondly, we explore a correlation between a differential-neural distinguisher's accuracy and a standard notion of difference between the two underlying distributions. Furthermore, we show that for a whole (practically relevant) class of ciphers, the differential-neural distinguisher can use differential features only. At last, we also rectify a common mistake in current literature, and show that, making use of an idea already presented in the foundational work[Goh19], the claimed improvements from using multiple ciphertext-pairs at once are at most marginal, if not non-existent.

Keywords: Deep Learning · Differential Cryptanalysis · Simon · Speck · Skinny · Present · Katan · ChaCha

1 Introduction

Evaluating the security of a cipher is of great importance. In symmetric cryptography this is, for the most time, done by proving resilience against already known types of attacks and by the absence of (known) attacks against the full version of the cipher, often including some security margin. For most types of attacks, automated search strategies for the underlying properties were developed. Examples of this are finding differential and/or linear characteristics with highest probability/correlation for a certain number of rounds[MWGP11, MP13, SHW$^+$14], or to automatically find integral attacks[Tod15, XZBL16]. Such tools often reduce the underlying problem to, for example, a SAT or MILP instance, which can be solved using already existing solvers.

Not too long ago, a new, machine learning assisted, type of attack on the block cipher Speck was proposed in [Goh19]. Having its foundations in differential cryptanalysis, the idea is to distinguish ciphertext-pairs that belong to a fixed plaintext difference from random ones. For this, neuronal networks are employed, leading to an 11-round key recovery attack on Speck32/64 that is at least competitive with *classical* ones.

While it is clear that machine learning will not completely substitute *classical* cryptanalysis, especially since we will see that, at least for the problem at hand, they tend to need a quite pronounced difference in the distributions of the learning problem, the question arises how generic this approach is and to which extend it can complement the work of a cryptanalyst. In other words, can we see machine learning as a tool assisting cryptanalysis, similar to how SAT and MILP solvers, among others, are seen by now?

While the successful application of differential-neural cryptanalysis to a wide variety of different ciphers[ZW22, BBCD21, CSYY22, JKM20, LLL$^+$22, ZWW22, HRC21a, HRC21b, BGL$^+$21, SZM20] already shows the generality of this technique, we explore on the basis of six ciphers, representing five different cipher designs, if (and by how much) a differential-neural distinguisher, as the machine learning based distinguisher from [Goh19] is coined by now, can be optimized further. Despite our distinguishers leading to better result than previously published ones, see Table 1 for a comparison[1], the increase in distinguishing performance over the best previously known distinguisher is not that big, and is disproportionate to the explored search space, making it quite likely that bigger improvements may not be possible. We will indeed see that, under some common assumptions, our results for Simon are nearly optimal. Based on this observation and the fact that the most important hyper-parameters appear to be identical for all six ciphers, we also provide guidance on how to change the neural networks hyper-parameters for other ciphers, allowing for automatic optimizations.

As another contribution, we show that the accuracy of a differential-neural distinguisher is strongly correlated with the mean absolute distance of the ciphertext-difference distribution and the uniform one, which, under some common assumptions, completely explains the features that the neural network is able to utilize for a whole class of ciphers. Still, for ciphers of a different class, it is already known from [Goh19] that more than purely differential features are used, and we provide additional example where this is likely to be the case, too. Using Simon as an example where only differential features can be used, we show that the differential-neural distinguisher is able to reach the same distinguishing quality as a purely differential distinguisher, while we are also able to show for toy versions of Speck that, similar to Speck32/64, the differential-neural distinguishers are able to surpass the purely differential ones. In both cases, the differential-neural distinguishers are able to learn a good approximation of the ciphertext-pair distribution, which is effectively the ciphertext-difference distribution in the case of Simon. If we consider that the ciphertext-difference distribution is classically computationally expensive to calculate, if not infeasible, such an approximation could be useful for cryptanalysis, especially since it is obtainable in a matter of hours on consumer grade hardware. Hence, this, the generality and the competitiveness of attacks relying on this technique show that, despite the already known limitations of differential-neural cryptanalysis such as limited explainability and number of rounds directly covered by the differential-neural distinguisher, it can be a valuable addition to the cryptanalyst's toolkit.

Finally, we also set out to straighten up some claims about distinguisher performance. It has become quite common in works following [Goh19] to use more than just one ciphertext-pair at once, see for example [CSYY22, ZWW22, ZW22, LLL$^+$22, HRC21b]. While this approach has the potential to use dependencies between the ciphertext-pairs that come from using the same key, current literature fails to put the results into the right perspective. Based on an idea already used in the underlying work[Goh19], we show how a fair comparison could be done and rectify some of the claims made in those works. Most crucially, some of the works that claim a huge improvement in distinguishing power are even worse than the original neural distinguishers from [Goh19], while the improvements of others are at most marginal, despite the high claims of the authors, see Table 2. This is especially unfortunate since it is exactly this idea that is used to (implicitly) turn the single-pair distinguisher into a multi-pair one for the original attacks against Speck32/64.

Related Work Following the introduction of differential-neural cryptanalysis[Goh19], many different directions have been explored. For instance, Baksi *et al.*[BBCD21] and Jain

[1]Note that the input differences for Present differ. But even using the same input difference as in [CSYY22], the accuracies of our distinguishers (without any further adjustments) are 0.662 and 0.550 for 6 and 7 rounds respectively.

Table 1: Comparison of neural distinguishers. For convenience of the reader, the results for each round-reduced cipher are ordered by *accuracy*.

Cipher	Full Rounds	Rounds	Accuracy	Source
SIMON32/64	32	9	0.598	[HRC21b]
			0.628	[SZM20]
			<0.63	[HRC21a]
			0.659	[BGL+21]
			0.661	This Work
		10	$0.501^{\dagger}$	[SZM20]
			0.566^{*}	[BGL+21]
			0.567^{*}	This Work
		11	0.517^{*}	[BGL+21]
			0.520^{*}	This Work
SPECK32/64	22	7	0.607	[CSYY22]
			0.616	[Goh19]
			0.617	This Work
		8	0.514^{*}	[Goh19]
SKINNY64/64	32	7	0.937	This Work
PRESENT	31	6	0.658	[CSYY22]
			0.712	This Work
		7	0.549	[CSYY22]
			0.563	This Work
KATAN32	254	61	0.536	This Work
		63	0.522	This Work
		65	0.514	This Work
		66	0.505	This Work
CHACHA	8/12/20	3	0.989	This Work

†: As far as we understand, the accuracy is evaluated using 10^6 samples, which means that the probability that this is just random noise is around 3.6%.
∗: Results need a more elaborate training procedure; there is no known way to obtain them by simple variations of direct training.

Table 2: Comparison of multi-pair differential-neural distinguishers. *Combined* means that the corresponding single pair distinguisher was used by combining the scores under independence assumption. For this, 10^6 samples, each consisting of the given number of pairs, were used to evaluating the accuracy.

			Pairs						
Cipher	Rounds	Neural Distinguisher	1	2	4	8	16	32	64
Simon32/64	9	[HRC21b]	0.591	0.586	0.623	–	–	0.823	–
		[SZM20]	0.628	0.637†	–	–	–	–	–
		[LLL+22]	–	–	–	–	–	–	0.993
		[ZWW22]	–	0.724	0.810	0.896	0.963	–	–
		Our (combined)	0.661	0.730	0.811	0.896	0.963	0.994	1.00
	10	[LLL+22]	–	–	–	–	–	–	0.861
		[SZM20]	0.501	0.502†	–	–	–	–	–
		[HRC21b]	–	–	–	–	–	0.611	–
		[ZWW22]	–	0.591	0.634	0.690	0.761	–	–
		Our (combined)	0.567	0.598	0.637	0.692	0.761	0.842	0.922
	11	[LLL+22]	–	–	–	–	–	–	0.596
		[ZWW22]	–	0.524	0.539	0.559	0.588	–	–
		Our (combined)	0.520	0.529	0.543	0.563	0.589	0.625	0.675
	12	[ZWW22]	–	–	–	0.515	0.523	–	–
Speck32/64	5	[CSYY22]	0.926	0.974	0.991	0.999	1.000	–	–
		[CSYY22] (combined)	0.926	0.978	0.997	1.000	1.000	1.000	1.000
		[Goh19] (combined)	0.929	0.978	0.997	1.000	1.000	1.000	1.000
	6	[CSYY22]	0.784	0.867	0.936	0.953	0.979	–	–
		[CSYY22] (combined)	0.784	0.873	0.947	0.989	0.999	1.000	1.000
		[ZWW22]	–	0.877	0.950	0.990	0.999	–	–
		[Goh19] (combined)	0.788	0.877	0.950	0.990	1.000	1.000	1.000
	7	[CSYY22]	0.607	0.640	0.685	0.701	0.649	–	–
		[HRC21b]	–	–	–	–	–	0.889	–
		[CSYY22] (combined)	0.607	0.650	0.706	0.779	0.861	0.938	0.985
		[ZWW22]	–	0.665	0.728	0.811	0.896	–	–
		[Goh19] (combined)	0.616	0.661	0.722	0.796	0.877	0.948	0.988
		Our (combined)	0.617	0.663	0.725	0.801	0.883	0.954	0.991
	8	[HRC21b]	–	–	–	–	–	0.565	–
		[ZWW22]	–	–	0.543	0.556	0.585	–	–
		[Goh19] (combined)	0.514	0.521	0.530	0.543	0.560	0.585	0.621
	9	[ZWW22]	–	–	–	0.502	0.505	–	–
Present	6	[CSYY22]	0.658	0.720	0.795	0.831	0.826	–	–
		[CSYY22] (combined)	0.658	0.730	0.815	0.902	0.967	–	–
		[ZW22]	–	0.733	0.820	0.907	0.970	–	–
		Our (combined)	0.712	0.799	0.889	0.960	0.993	1.000	–
	7	[CSYY22]	0.549	0.550	0.585	0.579	0.582	–	–
		[CSYY22] (combined)	0.549	0.573	0.605	0.650	0.711	–	–
		[ZW22]	–	0.572	0.607	0.656	0.721	–	–
		Our (combined)	0.563	0.593	0.635	0.691	0.765	0.849	–
Skinny	7	Our (combined)	0.937	0.992	1.000	1.000	1.000	1.000	1.000
Katan	66	Our (combined)	0.505	0.509	0.516	0.524	0.536	0.551	0.571
ChaCha	3	Our (combined)	0.989	0.999	1.000	1.000	1.000	1.000	1.000

†: Result is based on a difference-polytope that consists of 3 differences (i. e. 4 plain-/ciphertexts are used per sample).

et al. [JKM20] classify input differences instead of distinguishing between real and random samples. In addition, the work of Benamira *et al.*[BGPT21] explores the explainability of the results in [Goh19]. Also, similar approaches to [Goh19] have been applied to other ciphers, such as ASCON[BBCD21], Chaskey[ZW22, BBCD21, CSYY22], DES[ZW22, CSYY22], GIMLI[BBCD21], KNOT[BBCD21], Present[CSYY22, ZW22, JKM20], Simon[LLL+22, ZWW22, HRC21a, HRC21b, BGL+21, SZM20] and Simeck[LLL+22]. Furthermore, Bacuieti *et al.*[BBP22] explored the direction of finding smaller, and therefore more efficient, neural distinguishers, which, despite inferior distinguishing performance, have the potential to lower the attack complexity.

Outline. We start in Section 2 with introducing some basic notation, the general setting of our distinguishers and the structure of the employed neuronal network, as well as the ciphers under scrutiny. Afterwards, we perform a case study of optimizing the original differential-neural distinguisher for six ciphers, following five different design approaches, in Section 3 and provide some general guidance on which hyper-parameters should be considered. This is followed up by pointing out some manual optimizations that can be done for the ciphers at hand in Section 4. After that, we show in Section 5 that the distinguishing performance of a differential-neural distinguisher is strongly correlated to the mean absolute distance of the two underlying distributions, which is essentially the accuracy a purely differential distinguisher is able to achieve, as well as that for a class of ciphers it is impossible for any distinguisher to use more than differential features. This is followed up by conducting the real difference experiment proposed in [Goh19] for all ciphers under scrutiny in Section 6. We then show in Section 7 that it is rather likely for differential-neural distinguisher to require a quite pronounced skewness of the ciphertext-pair distribution in order to be able to learn, followed by a rectification of distinguisher comparisons for using multiple ciphertext-pairs at once in Section 8[2]. At last, we conclude our work in Section 9.

2 Preliminaries

2.1 Notation

In this work, $\mathbb{F}_2$ denotes the finite field consisting of the two elements $\{0,1\}$, and $\mathbb{F}_2^n$ the n-dimensional vector space over this field. For convenience, we will implicitly identify $(x_n,...,x_1) \in \mathbb{F}_2^n$ with the integer $\sum_{i=1}^n x_i 2^i$, and the other way around, based on the context. We will therefore call x_1 the least significant bit (lsb) and x_n the most significant bit (msb). Furthermore, $\oplus, \boxplus, \odot$ denote XOR, modular addition and bitwise product, and $\lll, \ggg$ denote left and right rotation respectively. Additionally, $\gcd(a,b)$ will denote the greatest common divisor of the two integers a and b.

At last, let us introduce some basic notation about differentials. For this, let $|S|$ denote the cardinality of the set S.

Definition 1 (Differential Probability (DP))**.** Let $F : \mathbb{F}_2^n \to \mathbb{F}_2^n$, as well as $\gamma, \delta \in \mathbb{F}_2^n$. Then we call $p_\delta^\gamma(F) := 2^{-n}|\{x \in \mathbb{F}_2^n | F(x) \oplus F(x \oplus \gamma) = \delta\}|$ the DP of the differential (γ, δ). If γ and F are clear from the context, we will just write p_δ.

As all ciphers under scrutiny are of an iterative nature, we will also recall the DP under Markov assumption.

[2]The source code for our experiments and the neural distinguishers can be found at `https://github.com/differential-neural/An-Assessment-of-Differential-Neural-Distinguishers`

Corollary 1 (DP under Markov Assumption)**.** *Let $F_1, ..., F_r : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, as well as $\gamma_1, ..., \gamma_{r+1} \in \mathbb{F}_2^n$. Then, under Markov assumption,*

$$p_{\gamma_{r+1}}^{\gamma_1}(F_r \circ ... \circ F_1) = \sum_{\gamma_2,...,\gamma_r \in \mathbb{F}_2^n} \prod_{i=1}^{r} p_{\gamma_{i+1}}^{\gamma_i}(F_i).$$

We are mainly interested in the maximal DP, either over the choice of output difference or both, input and output difference.

Definition 2 (Maximal DP)**.** Let $F : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$. Then the maximal DP for a fixed input difference $\gamma \in \mathbb{F}_2^n$ is $\max_{\delta \in \mathbb{F}_2^n \setminus \{0\}} p_\delta^\gamma(F)$. In addition, the maximal DP is $\max_{\gamma, \delta \in \mathbb{F}_2^n \setminus \{0\}} p_\delta^\gamma(F)$.

2.2 Differential Scenario

While cryptographers will most likely have some intuition on what we may mean by a differential distinguisher, let us provide a bit more details. Given an encryption function $E : \mathbb{F}_2^{n_k} \times \mathbb{F}_2^{n_p} \rightarrow \mathbb{F}_2^{n_c}$, where n_k denotes number of key bits, n_p the number of plaintext bits and n_c the number of ciphertext bits, the task of a differential distinguisher in this work will be to distinguish samples of the form $(E(k,p), E(k, p \oplus \Delta))$, for a fixed $\Delta \in \mathbb{F}_2^n$, as well as independent and identical distributed $k \in \mathbb{F}_2^{n_k}$ and $p \in \mathbb{F}_2^{n_p}$, which we shall call encryption samples, from independent and identical distributed $(c, c') \in (\mathbb{F}_2^{n_c})^2$, which we shall simply call random samples[3]. We will also refer to the distributions those samples are drawn from as the encryption and random distribution respectively.

This approach is motivated by the wrong-key-randomization-hypothesis, which states that a wrong (round-)key guess should lead to something that is indistinguishable from a random sample. While it is well known that, especially for the ciphers under scrutiny, this hypothesis does not hold, this assumption is still typical for attacks, and results like the ones from [Goh19] show that such a distinguisher can be very useful in an attack.

2.3 Structure of the Neural Network

Let us quickly recall the structure of the neural network used in [Goh19], which is shown in Fig. 1. Given a ciphertext-pair (c_1, c_2) the neural network first splits the ciphertexts up into words (e. g. left and right side in the case of Speck) in what we shall call the preprocessing layer. The result then goes through the bit sliced layer, where a convolution with a filter size of one and 32 filters is applied, followed by batch normalization and a Rectified Linear Unit (ReLU) activation layer. After that, 10 residual blocks are applied[4], each consisting of two iterations of convolutional layer (each using a filter size of 3 and 32 filters), batch normalization and (ReLU) activation, and a final addition of the input. At last, the output of the last residual block is processed by the prediction head, which consists of two iterations of a densely connected layer (using 64 neurons), batch normalization and activation, and a final densely connected layer, which uses the sigmoid activation function and one neuron only.

For training, the Adam optimizer is used, together with a cyclic Learning Rate (LR). The loss is calculated as the Mean Squared Error (MSE) between the prediction and the class label, together with a L2 regularization penalty.

[3]Since ciphers are usually bijective mappings if the key is fixed, strictly speaking the case that $c = c'$ should be excluded. But given that n_c is at least 32 for the ciphers under scrutiny, the probability of this happening is negligible.

[4]Note that less blocks were used to produce the final distinguishers in [Goh19].

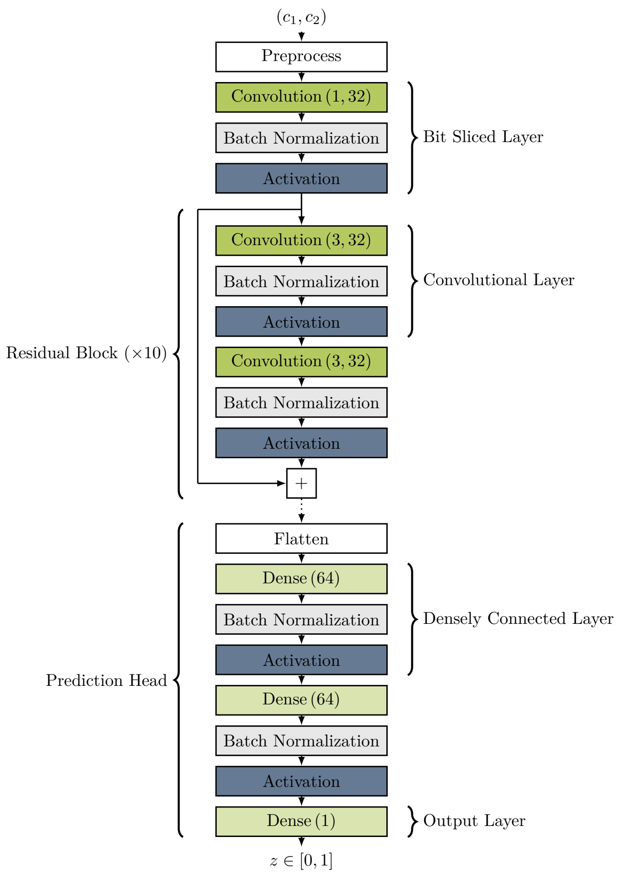
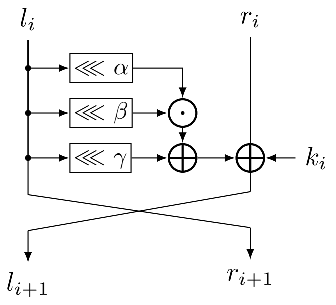
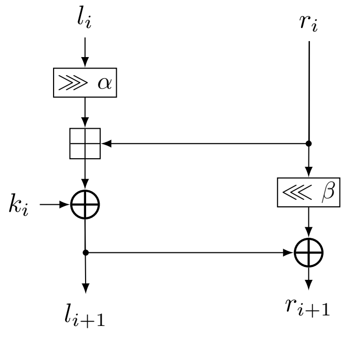


Figure 1: Structure of neural network used in [Goh19].



(a) Simon



(b) Speck

Figure 2: Round functions of Simon and Speck.

2.4 Ciphers under Scrutiny

Let us quickly recall the ciphers Speck, Skinny, Present, Katan and ChaCha that we will analyze in this work.

The Ciphers Simon and Speck Simon and Speck are families of lightweight block cipher designed by the NSA[BSS+13]. For the most part, we will concentrate on the variant using a 32-bit state and a 64-bit key, which we shall denote as Simon32/64 and Speck32/64 respectively. During encryption, the state is split into two equal sized words, which are then processed by Feistel-like round functions depicted in Fig. 2. The rotational constants for Simon32/64 are $\alpha = 1$, $\beta = 8$ and $\gamma = 2$, while the ones for Speck32/64 are $\alpha = 7$ and $\beta = 2$.

Note that, even without knowing the key, part of the last round can always be reverted, i. e. based on a ciphertext (l_i, r_i) we are able to calculate $(l_{i-1}, r_{i-1} \oplus k_{i-1})$ in the case of Simon and (l_i, r_{i-1}) in the case of Speck. We will refer to this as calculating the ciphertexts back.

For more details on Simon and Speck, the reader may consult [BSS+13].

The Cipher Skinny Skinny[BJK+16] is a tweakable block cipher and was designed by Beierle *et al.* It uses a round function, which is depicted in Fig. 3 and inspired by the Advanced Encryption Standard (AES). We will only consider the smaller version that uses a 64-bit block size and a (tweak-)key of the same size. Furthermore, we will call each of the 16 (4-bit) cells a word of the state. To be in line with the framework from [Goh19] we will ignore the tweak and use the tweakkey as an ordinary key.

Again, part of the round function can be reverted, without any knowledge about the key. This includes reverting the linear part (MixColumns and ShiftRows), as well as the addition of the round constants (AC) and the application of the S-box (SC) to the last two rows. In other words, it is possible to calculate the green cells (Fig. 3) within the last round.

More details on Skinny can be found in [BJK+16].

The Cipher Present Present[BKL+07], a cipher designed by Bogdanov *et al.*, also has a 64-bit block size, and we focus on the version that uses a 80-bit key. Its round function is shaped like a classical Substitution-Permutation Network (SPN), as can be seen in Fig. 4. Note that after the last round a final round key addition takes place, as otherwise most of the last round could trivially be reverted. This means that whenever we refer to the number of rounds for Present, this should be understood as the number of applications

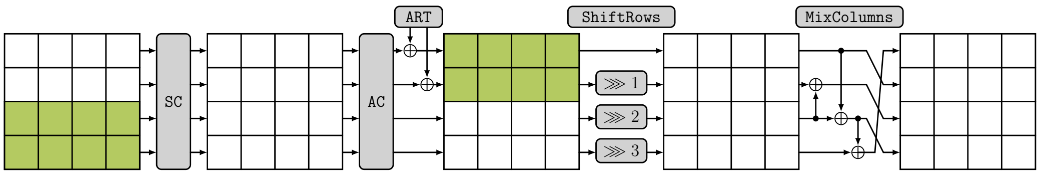


Figure 3: Round function of Skinny [Jea16]. The 16 green cells indicate to which point the last round function can be reverted without knowing the key.

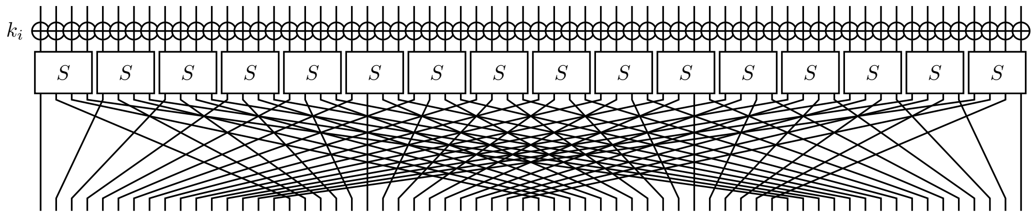


Figure 4: Round function of Present (modified version of [Jea15]).

of the round function, and the number of key additions is the number of rounds increased by one. See [BKL+07] for more details on Present.

While for Present, the bit-permutation right before the last key addition is, by nature, a linear operation, and we could revert this part, similar to how we revert non-key dependent operations for the other ciphers under scrutiny, we will not do so, since we do not expect changing the order of the bits to make a big difference.

The Cipher Katan Katan[DDK09] is a block cipher that uses shift registers and non-linear feedback functions to manipulate the state during en-/decryption. Here, we only consider the 32-bit version depicted in Fig. 5. For more details on Katan we refer to its specification[DDK09].

As for most of the other ciphers, we will also consider reverting non-key dependent parts of the round function. Since for Katan, this means reverting 17 (partial) rounds, we provide the details of this in Appendix B.

The Cipher ChaCha The last cipher we consider here is ChaCha[Ber08]. While ChaCha is a stream cipher, we will handle a keystream block as a ciphertext, and nonce and counter (i. e. the input data that may be controlled by an adversary) as the plaintext. This enables us to use similar techniques for all of the ciphers under scrutiny.

The key stream generation works as follows. First, an initial state, which is usually

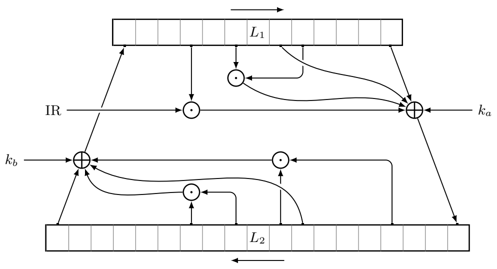


Figure 5: Structure of Katan (modified version of [Ava17]).

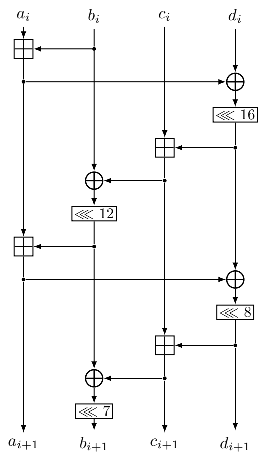


Figure 6: Quarter round of ChaCha (modified version of [Bre17]).

depicted as a 4×4 matrix of 32-bit words, is constructed by setting the first row to a constant, the next two rows to the 256 key bits, and the last row to nonce and counter. This state is then modified by an iterative application of the round function, which applies the quarter round function depicted in Fig. 6 to the columns resp. the diagonals of the matrix in parallel in even and odd rounds respectively. At the end, the initial state is added using modular addition. For more details, we refer to the design paper [Ber08].

For backwards calculation, we only consider reverting the modular addition of the two known rows of the initial state. While, in addition, the two outer outputs of the quarter round (a_i and d_i) are known, calculating d_i back to $d_{i-1} \oplus (a_{i-1} \boxplus b_{i-1})$ could make it harder to recognize any dependencies between d_i and the other three outputs. One possibility to prevent this would be to just provide $d_{n-1} \oplus (a_{n-1} \boxplus b_{n-1})$ as additional data. But this would increase the size of one sample (ciphertext-pair) from 1024 bits to 1280 bits. Also, we would hope that the neural network would be able to revert one simple XOR and two rotations itself. As we will see later, the accuracy of our differential neural-distinguisher is already 0.989. Hence, we do not provide this additional data.

3 A Case Study on Various Ciphers

In [Goh19] it was shown that for Speck neural networks can be used as distinguishers that lead to attacks that are at least competitive with *classical* ones. Naturally, this raises the following question.

1. Is the presented technique of differential-neural cryptanalysis generic? In other words, is the general idea applicable to a wide variety of ciphers? Considering that followup works for the ciphers Ascon[BBCD21], Chaskey[ZW22, BBCD21, CSYY22], DES[ZW22, CSYY22], GIMLI[BBCD21], Knot[BBCD21], Simon[LLL$^+$22, ZWW22, HRC21a, HRC21b, BGL$^+$21, SZM20], Simeck[LLL$^+$22] and Present[CSYY22, ZW22, JKM20, CSYY22] exist, this question can be answered positively.

2. How generic is the underlying neural network, i. e. can it be used to learn a multitude of different ciphers? For this, it should be noted that not all of the works cited above employ the same network architecture as in [Goh19], which could be understood as this architecture not being suitable for all of the ciphers.

3. Is it possible to improve the neural network form [Goh19] any further? While it is to be expected that for Speck32/64 the neural network may already be optimal, this

Table 3: Overview of used input differences. Differences without a reference were chosen such that diffusion in the first few rounds is limited and were found to perform better than ones from literature.

Cipher	Difference
Speck	0x0040 0000[Goh19, ALLW15]
Simon	0x0000 0040[KLT15, ALLW13]
Skinny	0x0000 0000 0000 2000
Present	0x0000 0000 00d0 0000
Katan	0x00002000
ChaCha	0x00000000 00000000 00000000 00008000

expectation should still be tested. In addition, assuming that the neural network can also be used for other ciphers than Speck, can we optimize it further for each specific cipher, and which of the hyper-parameters are important?

In the following of this section, we will set out to answer the last two (still open) questions. To this end, we will train and further optimize the neural network from [Goh19] for six different ciphers, following five different design approaches, namely Simon and Speck, representing Feistel networks, Present and Skinny, representing a SPN using a simple and a more involved (AES-like) linear layer respectively, Katan, that is based on non-linear feedback shift registers, as well as ChaCha, which is in contrast to the other ciphers not a block cipher, but a stream cipher. As we will see, using a bit of (automatic) optimization/customization for each cipher leads to a quite promising differential-neural distinguisher for a not insignificant number of rounds, which further supports the generality of the approach and of the employed neural network. In addition, we show that we can slightly improve over the best previously known differential-neural distinguishers, but despite considering a great amount of hyper-parameters, no groundbreaking improvements could be found.

3.1 On the used Input Difference

As described in Section 2.2, the encryption distribution requires an input difference Δ. We will see later that this difference has a major impact on the accuracy a (purely differential) distinguisher is able to achieve. Also, training the neural network from [Goh19] using a variety of differences shows that the choice of difference has a big impact on the accuracy of the neural network, too. It is therefore crucial to choose a reasonable good input difference for each cipher.

Our approach to this problem is as follows. First, we try differences that belong to differential characteristics of high probability reported in literature. In addition, we also try differences which should lead to low diffusion in the first few rounds. The later approach is motivated by the conjecture that low diffusion may help the neural network to better learn the encryption distribution.

Note that we first chose the input difference and then try to optimize the used hyper-parameters of the neural network. Even though the network is optimized for Speck, initial experiments show that it is also able to learn for other ciphers, be it with a lower accuracy and/or number of rounds compared to a cipher-optimized version.

We summarize the differences that led to the best results in our experiments, and that we will use in the rest of this paper (if not stated otherwise), in Table 3.

3.2 Cipher Specific Optimizations

A neural network, as the one used in [Goh19], exhibits a certain structure, which must be set in advance. This structure is defined by the so-called *hyper-parameters*, like the number of layers or neurons in each layer, which can have a big impact on the networks ability to learn. Hence, it is common in machine learning to search for hyper-parameters that lead to the best results by trying different combinations. We did so for all the ciphers under scrutiny.

The Search Space We started with a quite broad search space for both, Simon and Speck, which we than reduced to a more reasonable one for the other ciphers based on the results for those two ciphers.

The first search space is shown in Table 4. The last column gives the values used in [Goh19][Section 4.2][5]. The concrete search space for each hyper-parameter was determined by two factors, hardware limitations[6] and/or degradation in distinguishing power. For more details on those hyper-parameter, we refer to Appendix A.

Out of all those parameters, only batch size, the use of circular convolutions, the number of neurons in the densely connected layers, the number of filters, the filter size, the LR (while using the cyclic LR schedule), the number of residual blocks and the L2 penalty seem to have a significant (positive) effect. Hence, we restricted the search space for the other ciphers to only those hyper-parameters. The overall results are shown in Table 5 (hyper-parameters not shown are as in [Goh19][Section 4.2]). Out of all the hyper-parameters we tested, the ones that appear to have the biggest impact are LR, L2 penalty and especially the filter size. For Simon the higher filter size of 7 leads to an accuracy that is multiple percent points higher than with a filter size of 3. One possible explanation for this is that the neuronal network most likely (effectively) reverts deterministic transformations at the end of the encryption process, which in the case of Simon is one complete round (up to key addition). Hence, reverting such calculations in advance (i. e. before providing the samples to the neural network), which we shall denote as calculating the ciphertexts back, can be seen as one additional hyper-parameter of our search space. Doing so indeed improved the results for half of the ciphers in our experiments. For Speck, on the other hand, we do not see any significant improvement, which may be due to the neural network already being optimized to perform this calculation itself, and the same appears to be the case for Simon if a filter size of at least 7 is used.

We want to highlight that at least a small hyper-parameter search, with different LRs, L2 penalties and filter sizes, should be conducted if one plans to use a neural distinguisher for another cipher. But given that such a search can be automated using a tool like Optuna[ASY+19], the results in Table 5 show that the neural network from [Goh19] is quite generic.

Comparison to Literature As we mentioned before, there already exist results for Simon, Speck and Present in literature. If we take a look at Table 1, we see that, for the same number of rounds, our results are always superior (in terms of accuracy), even if not by much in the cases of Simon and Speck (and also Present if we use the same input difference). Given the size of the search space, it is likely that much greater improvements are simply not possible. We will see in Section 5 that at least for Simon this is most certainly the case.

Note that we naturally tried to increase to number of rounds, but did not find more rounds to be learnable without any changes to the training data and/or process. This

[5]Note that we use 10 residual blocks as our default/baseline despite the fact that less blocks were used to produce the final distinguishers in [Goh19].

[6]We used a Nvidia RTX 2080 Ti and 32GB of RAM for most of our experiments.

Table 4: Considered Hyper-Parameters.

Hyper-Parameter	Options	[Goh19]
Optimizer	Adadelta, Adagrad, Adam, Adamax, Ftrl, Nadam, RMSprop, SGD	Adam
LR Schedule	constant, cyclic, cyclic to constant, exponential decay, inverse time decay, linear cyclic, picewise constant decay, polynomial decay	cyclic
Loss Function	Kullback-Leibler Divergence, Mean Absolute Error, Mean Absolute Percentage Error, MSE, Mean Squared Logarithmic Error, Binary Crossentropy, Hinge, LogCosh, Poisson, squared Hinge, Huber	MSE
Activation Function	ELU, ReLU, Leaky ReLU, SELU, Softplus, Swish	ReLU
Batch Size	10^3 to $5 \cdot 10^4$	$5 \cdot 10^3$
# Densely Connected Neurons	48 to 96 per layer	64
# Densely Connected Layers	0 to 4	2
# Residual Blocks (depth)	0 to 48	10
Residual Block Design	See Fig. 14	Fig. 14a
# Convolutions in Residual Block	1, 2, 3	2
# Filters	8 to 256	32
Filter Size	1 to 11	3
Filter Size Bit Sliced	1, 3	1
Inception[SLJ+15] instead of Convolution	true, false	false
Circular Convolutions[SNPP19]	true, false	false
Dropout Rate	0 and from 0.02 to 0.0025	0
Batch Normalization	true, false	true
Regularization	L1, L2, L1 and L2	L2
Regularization Factor	$2.5 \cdot 10^{-3}$ to $8 \cdot 10^{-8}$	10^{-5}
Residual Scaling	1 to 0.1	1
Bit Encoding	0/1, 1/2, -1/1	0/1
Label Smoothing*	0 to 0.3	0

∗: Only for using Binary Crossentropy as the loss function.

Table 5: Overview of differential-neural distinguishers from hyper-parameter search. Parameters not mentioned are as in [Goh19]. The accuracy is evaluated usingl 10^7 samples, i. e. an accuracy above 0.5005 reaches a significance level of 0.001.

Cipher	Speck	Simon	Skinny	Present	Katan	ChaCha
Full Rounds	22	32	32	31	258	20
Round	7	9	7	7	66	3
Accuracy (Acc.)	0.617	0.661	0.937	0.563	0.505	0.989
High LR ($/10^{-3}$)	3.5	2.7	1.1	3	1.7	2
Low LR ($/10^{-5}$)	22	20	4.5	28	6	14
Neurons	80	64	64	64	64	64
Filter Size	3	7	3	1	7	7
Number Filters	16	32	32	32	32	32
L2 ($/10^{-8}$)	84.9	220	4.3	62	600	1400
Circular Convolution	false	true	false	false	false	false
Calculate Back	false	false	true	false	true	true

seems to be in line with literature, as the results for more rounds are either not free from doubt, or use the more elaborate training processes introduced in [Goh19].

4 Manual Optimizations

In the previous section, we studied the performance of a generic neural network based cryptanalytic training pipeline on a wide variety of different ciphers. In this section, we will show that the presented results can be significantly improved with some limited guidance from a cryptanalyst.

4.1 Types of Manual Optimization

Cryptanalytic attacks on block ciphers can usually be roughly divided into two parts, namely a statistical distinguisher against the block cipher and a key extraction phase. Typically, the distinguisher will recognize some event that happens in the middle of the cipher's execution and which is triggered (with some probability) by sending well-chosen input values into the cipher. The key extraction phase then uses the distinguisher to verify guesses of the last round keys and, sometimes, also to guide the search for these subkey values. In this article, this second step is not of interest.

When designing a machine learning based attack, the cryptanalyst will aim to capture a stronger distinguishing signal than would be available using classical techniques and to bridge as many rounds as possible using an automatic techniques, while retaining a simple interface to classical pre- and post-processing steps and a reasonably high level of distinguisher efficiency. In other words, ideally they would want to reliably detect an event deep inside the cipher that can be controlled well by classical chosen input techniques.

In the context of optimizing the neural part of neural-differential attacks, this naturally opens up the following ideas for optimization of existing distinguishers:

1. We can improve the choice of input differences. In principle, this does not necessarily mean picking input differences that generate *deeper* distinguishers; it is perfectly conceivable that improvements to an attack can be obtained by trading a slightly weaker machine-learned distinguisher for more control over the event that it detects. However, in this work, we will solely focus on optimizing input differences to obtain stronger distinguishers.

2. The designer can also try to obtain neural distinguishers that work for more rounds by modifying the distinguisher to cover more rounds. In general, the most natural approach to do this are *staged training* or *extension by key-averaging* as introduced in [Goh19]; in this article, we will see that for PRESENT, *simply using the same distinguisher for more rounds* works to a significant extent.

4.2 Optimizations for Simon

Before diving into evaluating the PRESENT distinguishers on more rounds, we would like to quickly explain how we trained the differential-neural distinguishers for 10- and 11-round SIMON32/64, that we report of in Tables 1 and 2. Similar to the training of the 11-round distinguisher in [BGL+21], we applied the staged training method introduced in [Goh19]. We started with training 8-round distinguishers based on our 9-round distinguisher, but for the input differences `0x0180 0040` and `0x0440 0100`, which are the most likely differences to occur after 2 and 3 rounds respectively. Based on those two distinguishers, we trained the 10 and 11 round distinguisher (using 10^8 samples) for one epoch with a learning rate of 10^{-4}, and another two/nine epochs using a learning rate of 10^{-5}.

The results actually slightly surpass the ones reported in [BGL+21], as our 10-round distinguisher achieves an accuracy of 0.567 instead of 0.566 and the 11-round distinguisher achieves one of 0.520 instead of 0.517. Note that the increase in accuracy, despite being small, is still significant.

4.3 Optimizations for Present

For PRESENT, we investigated whether staged training leads to learning of additional rounds; doing so, we found at first that our distinguisher was adapting extremely quickly to higher round numbers and then realized that it simply kept working *without* retraining, albeit at reduced distinguishing advantage. We also found that the same distinguisher works well for several input differences.

Evaluating the Advantage of a Neural Distinguisher To the best of our knowledge, in all of the literature since [Goh19], the distinguishing advantage of neural distinguishers has so far been characterised by two measures: their accuracy on the one hand and the key-ranking advantage obtained in attacks on the other hand. Additionally, the performance of these distinguishers is implicitly being measured by the loss functions used during training; mean square error loss is the most common choice in the literature.

All of these measures of success for a distinguisher encounter problems when we try to determine how well a given neural distinguisher works for output distributions other than the one it was trained for:

1. Accuracy collapses the output values of the distinguisher into binary choices, which loses a lot of information. This loss of information is usually not too critical for distinguishers that we can obtain by training on the output distribution, as the biases discovered by training will usually be large enough to lead to statistically significant accuracy results on test samples of the same order of magnitude as the training data (or much smaller). However, for distinguishers obtained by transferring an existing distinguisher to a different output pair distribution, one would be interested in being able to show the existence of very small biases.

2. Mean square error loss is not necessarily easy to interpret for poorly calibrated distinguishers. For instance, a given neural distinguisher D and the same distinguisher with binary thresholding will by definition reach the same accuracy level, but the mean square error of the latter will usually be far larger than even for the baseline that outputs a value of 0.5 for every sample.

Table 6: Present distinguisher evaluated (using 10^7 samples) on more rounds than it was trained on. Every result other than the ones for 10 rounds passes a significance test with significance level 0.001.

Encryption Rounds	6-round Distinguisher	7-round Distinguisher
6	0.658	–
7	0.557	0.563
8	0.509	0.512
9	0.502	0.502
10	0.500	0.500

Table 7: Present distinguisher evaluated (using 10^7 samples from the real distribution and a fixed set of 10^8 samples from the random distribution) on more rounds than it was trained on. All reported results except the one on 11 rounds are significant at the 3σ level or more. The 11-round result becomes significant when 10^8 real labels are used, at an empirical deviation of $\approx 5\sigma_0$.

Encryption Rounds	7-round Distinguisher, $(\mu_1 - \mu_0)/\sigma_0$
8	229
9	34.8
10	4.8
11	0.7

A natural alternative approach is to measure distinguisher performance by sampling the output distribution (without thresholding) of a distinguisher on real and on random test data and applying standard hypothesis testing to see whether the hypothesis that both distributions are the same can be rejected. In principle, there are a variety of tests that can be used for this purpose, but we will in this work settle for a very simple method.

If we denote the distinguisher output for sample i with X_i and assume that we have n samples chosen independently at random either all from the real or all from the random distribution, then the X_i are independent and identically distributed random variables with range $[0, 1]$. As such, they have a mean μ_j (where index $j = 1$ is the real case and $j = 0$ indicates the random distribution) and standard deviation σ_j. The mean and standard deviation of the empirical mean $\frac{1}{n} \sum i = 1^n X_i$ are μ_j and $1/\sqrt{n}\sigma_j$ and by the central limit theorem, the distribution of the empirical mean is well approximated by a normal distribution with these parameters.

In general, it is sufficient to show that sampling from the real and random output distribution of a cryptographic distinguisher yields *recognisably different* results for these two cases. In the case of Present, we will, however, assume that $\mu_1 \geq \mu_0$. Since the random distribution does not depend on number of rounds or input difference considered, empirical estimates of the parameters of the random distinguisher output distribution can be computed once on a large sample for a given distinguisher and then reused across experiments. This allows us to be fairly certain about the parameters for the distinguisher output distribution on the random samples. In Table 7, we report an advantage at more than three standard errors $s = \frac{\mu_1 - \mu_0}{\sigma_0}$ of the random distribution for up to ten rounds; this advantage can be shown to still exist at 11 rounds, although in our trials we saw it reliably only with significantly more real ciphertext pairs (roughly 10^8 were clearly sufficient). For comparison, Table 6 shows the observed empirical accuracy values for two different distinguishers against Present as measured by the standard test of running the distinguisher on 10^7 samples collected from the real and the random distribution in roughly equal proportion.

4.4 Optimizations for Katan

A similar phenomenon as described in the previous section for Present was also evident for Katan. While we did not find that our neural distinguishers were *directly* applicable to more rounds than we had trained the distinguishers in question for, we found that fine-tuning the weights of the last network layer of the distinguisher provided a quite effective way to adapt them for higher round numbers. To this end, we used one of our ordinary distinguishers for Katan as the base model and removed the final sigmoid output layer. The resulting model outputs for any observed Katan ciphertext pair an internal representation vector consisting of 64 floating-point numbers. To attack $n + k$ rounds, we then performed linear regression to find a linear decision surface that would classify $n + k$-round output as real or random as accurately as possible. Using our 60-round distinguisher as the base model, applying the same backwards calculation as for that model, and training the final linear distinguisher on $5 \cdot 10^6$ samples, we obtained a distinguisher against 71-round Katan with an accuracy of 50.15 percent; applying the same test as in the last section for Present, the mean response of the resulting fine-tuned distinguisher to real samples was estimated more than 12 times the standard error of the observed random samples above the mean for the random samples.

5 Relations to *Classical* Differential Attacks

While it is common in differential cryptanalysis to use just a single differential to mount an attack, it can actually be beneficial to use all possible output differences (or more precise their probability), see for example [AL13]. Of course, this is often infeasible as it would require the calculation of one row of the Difference Distribution Table (DDT). But at least for Simon32/64 and Speck32/64 it is, under Markov assumption, feasible using the methods provided in [KLT15] and [LM02] respectively, and was already done in [BGL+21] for Simon32/64 and in [Goh19] for Speck32/64.

The Accuracy of a Purely Differential Distinguisher Given the probability p_δ for every output difference $\delta \in \mathbb{F}_2^n$ (and a fixed input difference) a distinguisher could just use $p_\delta > \frac{1}{2^n-1}$ as a decision rule, i. e. the ciphertext-pair is coined to belong to the encryption distribution if the probability of the observed ciphertext difference is greater in the encryption case than the probability of the same difference to appear in the random case[7]. With this, the accuracy of such a distinguisher would be

$$\frac{1}{2}\sum_{p_\delta > \frac{1}{2^n-1}} p_\delta + \frac{1}{2}\sum_{p_\delta \leq \frac{1}{2^n-1}} \frac{1}{2^n-1},$$

assuming that both, the encryption and the random case, appear with probability $\frac{1}{2}$, as it is the case for training the neural network (and evaluating its accuracy). Note that if both cases are equiprobable, this is actually the best that can be done if only the ciphertext-differences of single samples are considered.

Interestingly, this accuracy is basically the same as the mean absolute distance of the ciphertext-difference distribution and the uniform one.

Lemma 1. *The accuracy of a purely differential distinguisher is the same (up to constants) as the mean absolute distance of the ciphertext-difference distribution and the uniform one.*

[7]Note that the zero difference cannot appear, which means that (in the random case) the differences can be seen as drawn uniformly from $\mathbb{F}_2^n \setminus \{0\}$.

Proof. Since both p_δ and $\frac{1}{2^n-1}$ sum to 1 over all possible $\delta \in \mathbb{F}_2^n \setminus \{0\}$, we have

$$\sum_{p_\delta > \frac{1}{2^n-1}} p_\delta + \sum_{p_\delta \le \frac{1}{2^n-1}} \frac{1}{2^n-1} = 1 - \sum_{p_\delta \le \frac{1}{2^n-1}} p_\delta + \sum_{p_\delta \le \frac{1}{2^n-1}} \frac{1}{2^n-1} = 1 + \sum_{p_\delta \le \frac{1}{2^n-1}} \left| \frac{1}{2^n-1} - p_\delta \right|,$$

as well as

$$\sum_{p_\delta > \frac{1}{2^n-1}} p_\delta + \sum_{p_\delta \le \frac{1}{2^n-1}} \frac{1}{2^n-1} = \sum_{p_\delta > \frac{1}{2^n-1}} p_\delta + 1 - \sum_{p_\delta > \frac{1}{2^n-1}} \frac{1}{2^n-1} = \sum_{p_\delta > \frac{1}{2^n-1}} \left| p_\delta - \frac{1}{2^n-1} \right| + 1.$$

Combining both, we get

$$\begin{aligned} &\frac{1}{2} \sum_{p_\delta > \frac{1}{2^n-1}} p_\delta + \frac{1}{2} \sum_{p_\delta \le \frac{1}{2^n-1}} \frac{1}{2^n-1} \\ =&\frac{1}{4} \left(1 + \sum_{p_\delta \le \frac{1}{2^n-1}} \left| \frac{1}{2^n-1} - p_\delta \right| + \sum_{p_\delta > \frac{1}{2^n-1}} \left| p_\delta - \frac{1}{2^n-1} \right| + 1 \right) \\ =&\frac{1}{2} + \frac{1}{4} \sum_{\delta \in \mathbb{F}_2^n \setminus \{0\}} \left| p_\delta - \frac{1}{2^n-1} \right|. \end{aligned}$$

□

This means that maximizing the accuracy of such a distinguisher (over the chosen input difference) may not lead to the same input difference as if one tries to maximize the probability of a differential, as the former maximizes the *mean absolute* distance, while the later maximizes the *maximal* distance.

5.1 On the Ciphertext-Pair Distribution

While it was shown in [Goh19] that the neural distinguisher for SPECK is able to use more than just differential features, the same is unlikely to be true for ciphers like SIMON and PRESENT. To see why this is the case, let us recall that a sample from the encryption distribution is of the form $(c_1, c_2) = (E(k, p), E(k, p \oplus \Delta))$ for $k \in \mathbb{F}_2^{n_k}$ and $p \in \mathbb{F}_2^{n_p}$ drawn independently and uniformly at random, which has to be distinguished from the two ciphertexts being drawn independently and uniformly at random. If $E(k, .)$ is bijective, choosing p uniform at random is the same as choosing c_1 or c_2 uniform at random. Hence, looking only at c_1 (or only at c_2) there is no difference to the uniform distribution. So, a distinguisher needs to look at (parts of) c_1 and (parts of) c_2 at once in order to distinguish the encryption distribution from the uniform one.

Knowing that c_1 and c_2 are uniform distributed also tells us that the marginal distribution of each bit is a Bernoulli-distribution with probability $\frac{1}{2}$ for it to be 1 (or 0). Obviously, adding independent key bits from a key chosen uniform at random removes any potential dependency between bits.

Lemma 2. *Let $X_1, ..., X_m$, $Y_1, ..., Y_l$ and $K_1, ..., K_l$ be Bernoulli-distributed random variables. Let further (for every i) K_i be 1 with probability $\frac{1}{2}$ and independent of $X_1, ..., X_m$, $Y_1, ..., Y_l$ and $K_1, ...K_{i-1}, K_{i+1}, K_l$. Then*

$$\Pr\left[X_i = x_i \,\forall i, Y_j = y_j \oplus K_j \,\forall j\right] = 2^{-l} \Pr\left[X_i = x_i \,\forall i\right].$$

Proof.

$$\begin{aligned}
&\Pr\left[X_i = x_i\,\forall i, Y_j = y_j \oplus K_j\,\forall j\right]\\
=&\sum_{(k_1,\ldots,k_l)\in\mathbb{F}_2^l} \Pr\left[X_i = x_i\,\forall i, Y_j = y_j \oplus k_j\,\forall j\right]\cdot \Pr\left[K_j = k_j\,\forall j\right]\\
=&2^{-l}\sum_{(k_1,\ldots,k_l)\in\mathbb{F}_2^l} \Pr\left[X_i = x_i\,\forall i, Y_j = y_j \oplus k_j\,\forall j\right]\\
=&2^{-l}\Pr\left[X_i = x_i\,\forall i\right]
\end{aligned}$$

□

By interpreting the X_i as bits from c_1 and Y_i as bits from c_2 this means that if we add independent key bits (more precisely key bits that are independent of the bits from c_1 and c_2 we are looking at) to c_2 we would get

$$\begin{aligned}
&\Pr\left[X_1 = x_1, \ldots, X_m = x_m, Y_1 = y_1 \oplus K_1, \ldots, Y_l = y_l \oplus K_l\right]\\
&= 2^{-l}\Pr\left[X_1 = x_1, \ldots, X_m = x_m\right] = 2^{-m-l},
\end{aligned}$$

i. e. the bits would look uniform distributed. But there is also a different interpretation. If we divide the bits from c_1 and c_2 that we are looking at in a group of bits that get added by the same key bits (or no key bits at all), i. e. we have dependencies between the key bits that get added to the bits, and interpret them as $X_1, \ldots, X_m$ (after key addition), and interpret the rest of the bits, i. e. the ones where we add independent key bits, as $Y_1 \oplus K_1, \ldots, Y_l \oplus K_l$, we can see that we only need to consider the group corresponding to the dependent key bits.

For illustration let us look at a Feistel cipher with round function $F : \mathbb{F}_2^{2n} \to \mathbb{F}_2^{2n}$, $F_k(l, r) = (r \oplus k \oplus f(l), l)$ and Feistel function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$. Let us consider the two ciphertexts $c_1 = (s_1^{(l)}, \ldots, s_n^{(l)}, s_1^{(r)}, \ldots, s_n^{(r)})$ and $c_2 = (q_1^{(l)}, \ldots, q_n^{(l)}, q_1^{(r)}, \ldots, q_n^{(r)})$. If we would look at $s_1^{(l)}, \ldots, s_n^{(l)}$ and $q_1^{(r)}, \ldots, q_n^{(r)}$ we would see no dependency since the last round key got added to $s_1^{(l)}, \ldots, s_n^{(l)}$ but is independent of $q_1^{(r)}, \ldots, q_n^{(r)}$, assuming independent round keys. Analog, there can also be no dependency between $s_1^{(l)}, \ldots, s_{\lfloor n/2\rfloor-1}^{(l)}$ and $q_{\lceil n/2\lceil}^{(l)}, \ldots, q_n^{(l)}$.

DDT vs. Ciphertext-Pair Distribution While it is clear that it is possible to calculate the DDT knowing the probability distribution of (c_1, c_2), it is interesting to note that the reverse is also true under some conditions.

Lemma 3. *Let $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_m$ be Bernoulli-distributed random variables and $(z_1, \ldots z_m)$, $(w_1, \ldots, w_m)$ as well as $(\delta_1, \ldots, \delta_m)$ be elements of $\mathbb{F}_2^m$. Let further*

$$\Pr\left[V_j = v_j \mid X_i = v_i, Y_i = v_i \oplus \delta_i\,\forall i \neq j\right] = \frac{1}{2} \tag{1}$$

hold for all $(v_1, \ldots, v_m) \in \mathbb{F}_2^m$ and all $V_j \in \{X_j, Y_j\}$. Then we have that

$$\Pr\left[X_i = z_i, Y_i = z_i \oplus \delta_i\,\forall i\right] = \Pr\left[X_i = w_i, Y_i = w_i \oplus \delta_i\,\forall i\right],$$

i. e. the probability is independent of the concrete choice of $z_1, \ldots z_m$.

Proof. Let us assume that $(z_1, \ldots z_m) \neq (w_1, \ldots, w_m)$ as otherwise the lemma would already be true. This means that there exists an j such that $z_j \neq w_j$. We will show that the probability stays the same when replacing z_j with $w_j = z_j \oplus 1$. For this let us look at

$$\begin{aligned}
&\Pr\left[X_i = z_i, Y_i = z_i \oplus \delta_i\,\forall i\right]\\
=&\Pr\left[X_j = z_j, Y_j = z_j \oplus \delta_j \mid X_i = z_i, Y_i = z_i \oplus \delta_i\,\forall i \neq j\right]\cdot \Pr\left[X_i = z_i, Y_i = z_i \oplus \delta_i\,\forall i \neq j\right].
\end{aligned}$$

By using Eq. (1) we get that

$$\begin{aligned}
&\Pr\left[X_j = z_j, Y_j = z_j \oplus \delta_j \mid X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i \neq j\right] \\
=&\frac{1}{2} - \Pr\left[X_j = z_j, Y_j = z_j \oplus 1 \oplus \delta_j \mid X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i \neq j\right] \\
=&\frac{1}{2} - \left(\frac{1}{2} - \Pr\left[X_j = z_j \oplus 1, Y_j = z_j \oplus 1 \oplus \delta_j \mid X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i \neq j\right]\right) \\
=&\Pr\left[X_j = w_j, Y_j = w_j \oplus \delta_j \mid X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i \neq j\right].
\end{aligned}$$

Thus, we have

$$\Pr\left[X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i\right] = \Pr\left[X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i \neq j, X_j = w_j, Y_j = w_j \oplus \delta_j\right].$$

The lemma follows now from repeating this for all $z_i \neq w_i$. □

For us this means that as long as Eq. (1) holds the probability distribution does only depend on the difference between c_1 and c_2 and can therefore be calculated using the DDT. This can easily be seen by expressing the probability of a difference as the sum of every possible bit configuration

$$\Pr\left[X_i = Y_i \oplus \delta_i \,\forall i\right] = \sum_{(z_i,\ldots,z_m)} \Pr\left[X_i = z_i, Y_i = z_i \oplus \delta_i \,\forall i\right].$$

Since the lemma tells us that all configurations have the same probability, calculating them from the probability of the difference is just a matter of dividing by the number of possible configurations. Hence, if we want to see whether it could be possible to learn additional features aside from the ones provided by the DDT, we can check if Eq. (1) does not hold.

For example, let us assume we have a cipher with round function of the from $R : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$, $R(k,x) = f(x) \oplus k$ for some $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and independent round key k. Then we know from Lemma 2 that Eq. (1) holds. Therefore, we cannot expect to learn additional information from the distribution of ciphertext-pairs than we could already learn from the DDT[8]. More general, we have shown the following.

Theorem 1. *If for a (bijective) cipher there exist a transformation of the ciphertexts independent of the key such that the transformed ciphertexts have the from $c' \oplus k'$, where k' are independent key bits, then the encryption distribution and the ciphertext-difference distribution (where the difference is calculated after applying the key independent transformation) contain the same information.*

Hence, in order to show that the encryption distribution does not provide any additional information to the DDT we have to show that we can represent ciphertexts as $c' \oplus k'$ for independent key bits k'. Obviously, this is the case for Present if we assume independent round keys, as well as all other key-alternating ciphers.

Additionally, this is also the case for Feistel ciphers. To see that, let us denote the round function of such a cipher as $F_k(l,r) = (r \oplus k \oplus f(l), l)$ for some function $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ and round key k. Obviously, we can see the key addition as the first operation in the round and calculate one round back up to the key addition. Hence, instead of looking at ciphertexts of the from $(r_i \oplus k_i \oplus f(l_i), l_i)$ we can look at ones of the form $(r_i \oplus k_i, l_i) = (r_i \oplus k_i, r_{i-1} \oplus k_{i-1} \oplus f(l_{i-1})) = (r_i, r_{i-1} \oplus f(l_{i-1})) \oplus (k_i, k_{i-1})$. Since r_i, r_{i-1}, l_{i-1} do not depend on the round keys k_i, k_{i-1} (assuming independent round keys)

[8]That is if we know the real DDT, i. e. the model used to calculate the DDT does not contain any assumptions that are not met such as the Markov assumption that is often used even though it does not hold in most cases. Otherwise, we would only have an approximation of the DDT, and therefore of the distribution of ciphertext-pairs, and could still hope to find a better one.

Table 8: Accuracy comparison between neural distinguisher and DDT based one (after reverting one round up to key addition) for Simon32/64.

Distinguisher	Rounds		
	9	10	11
Neural	0.661	0.567	0.520
DDT[BGL+21]	0.663	0.568	0.520

we know that we can calculate the probability distribution of the ciphertexts (calculated back up to key addition) by using the DDT. In other words, Simon is another example where only differential features can be used, again, assuming independent round keys. But for Feistel ciphers, such as Simon, we first need to revert one round, up to key addition.

Knowing that, and given that Bao *et al.*[BGL+21] already calculated the accuracy a DDT based distinguisher is able to achieve, we can actually judge the quality of the approximation the differential-neural distinguisher is able to produce for Simon32/64. As can be seen in Table 8, the accuracies of the two distinguishers are almost identical, supporting the conjecture that the results we have seen in Section 3 may be (almost) optimal. We will later see that the same is true for toy versions of Simon, which makes it reasonable to assume that this observation can be extrapolated to the 48-, 64-, 96- and 128-bit block versions of Simon, too. Note that, while calculating the DDT for the 32-bit version of Simon is possible, at least under Markov assumption, it is already computationally expensive, and one can expect it to become infeasible for greater block sizes. Hence, the use of differential-neural distinguishers could be even more valuable for greater block sizes.

A Chance to Still Use More Than Differential Features While the results above show that for single ciphertext-pairs only their difference may provide information, one option to still improve over purely differential distinguishers could be to use multiple ciphertext-pairs at once, making use of dependencies between the pairs, especially if the key is fixed. But as we will see in Section 8, the dependencies are either not that strong, or they simply do not get used much by current state of the art multi-pair distinguishers.

5.2 Influence of the Key Schedule

Applying Theorem 1 to Simon and Present has the caveat of assuming independent round keys. While this assumption is not uncommon, it is conceivable that a differential-neural distinguisher may use (potential) weaknesses within the key schedule. In the end, this could as well be the reason why the differential-neural distinguisher in [Goh19] is able to perform better than the purely differential one. To answer this question we conducted a simple experiment, replacing the original round keys derived using the corresponding key schedule by independent and identical distributed round keys (i. e. using a free key schedule) and evaluating the differential-neural distinguisher on the so created samples. For all six ciphers, doing so does not lead to any significant difference in accuracy. Hence, we can conclude that the key schedule has no (notable) impact on the distinguishing performance, at least for the assessed ciphers.

5.3 Experiments using Small Scale Variants of Simon and Speck.

To analyze the relation between differential-neural and *classical* cryptanalysis further, we conducted experiments on small scale variants of Simon and Speck. First, we calculated the whole DDT, under Markov assumption, using the methods from [KLT15, LM02]. While doing so for the 32-bit variants is feasible, and was already done in [BGL+21, Goh19] for a fixed input difference, it is still computationally expensive. To make our experiment more economic, and since it is unlikely that the results would look significantly different for bigger block sizes, we will consider 16-bit versions of the two ciphers instead.

But since 16-bit versions of SIMON and SPECK are not specified, it is not clear which rotational constants to use[9]. Hence, we will first reduce the amount of rotational constants by considering reasonable choices (with respect to purely differential attacks) only.

Small Scale Simon Starting with SIMON, the authors of [KLT15, Bei16] already provide constraints for choosing reasonable rotational constants α, β and γ (using the notation from Fig. 2a). The most obvious one is that α and β can be switched and, of course, $\alpha \neq \beta$ should hold. Also, it should hold that $\gcd(|\alpha - \beta|, n) = 1$, where $n = 8$ is the word size, in order to keep the differential probabilities low. For the same reason $\gamma \notin \{\alpha, \beta\}$ should hold, too. In addition, Kölbl *et al.* also show in [KLT15] that multiplying all rotational constants by $s \in \mathbb{Z}_n^*$ results in the same cipher apart from bit permutations (and changes within the key schedule). We will adopt their notation of structural equivalence for this.

Definition 3 (Structural equivalence[KLT15])**.** If two ciphers are identical apart from bit permutations of the input, output and round key bits we will call them structural equivalent.

Furthermore, we want to note that the following also holds.

Lemma 4. *For SIMON, adding $n/2$ to all rotational constants, where n denotes the word size, leads to a structural equivalent cipher.*

Proof. Let
$$f(x)_{(\alpha,\beta,\gamma)} = ((x \lll \alpha) \odot (x \lll \beta)) \oplus (x \lll \gamma)$$
be the Feistel function of SIMON. It is easy to see that, for any integer ρ, $f(x)_{(\alpha+\rho,\beta+\rho,\gamma+\rho)} = f(x)_{(\alpha,\beta,\gamma)} \lll \rho$ holds. If we denote by
$$R^{(k)}_{(\alpha,\beta,\gamma)}(l, r) = \big(r \oplus f(l)_{(\alpha,\beta,\gamma)} \oplus k, l\big)$$
the round function of SIMON, then by using $\rho = n/2$ it is easy to see that
$$R^{(k)}_{(\alpha+n/2,\beta+n/2,\gamma+n/2)}(l, r \lll n/2) = \big((r \oplus f(l)_{(\alpha,\beta,\gamma)} \oplus (k \lll n/2)) \lll n/2, l\big)$$
and
$$R^{(k)}_{(\alpha+n/2,\beta+n/2,\gamma+n/2)}(l \lll n/2, r) = \big(r \oplus f(l)_{(\alpha,\beta,\gamma)} \oplus k, l \lll n/2\big)\,.$$
Hence, using (α, β, γ) and $(\alpha + n/2, \beta + n/2, \gamma + n/2)$ produces structural equivalent ciphers. □

Therefore, we have classes of equivalent rotational constants, leading to structural equivalent ciphers. For selecting one combination of rotational constants of each of these equivalence classes we make use of the following lemma.

Lemma 5. *Let n be the word size. If n is even, $n \geq 4$ and $\gcd(|\alpha - \beta|, n) = 1$, then there exist unique and non-negative integers $b < n/2$ (even) and $c < n - 1$ such that, for SIMON, using the constants (α, β, γ) is structural equivalent to using the constants $(1, b, c)$.*

[9]For SIMON it is also not clear how the key schedule should be adapted, which is why we use independent rounds keys instead. For SPECK, on the other hand, the key schedule makes use of the round function only, which makes adaptation of the key schedule straight forward.

Table 9: Maximal DP and accuracy (based on DDT) for 7-round small scale Simon and different choices of rotational constants, together with the input difference which leads to those results, and ordered by maximal DP. Rotational constant α is fixed to 1.

β	γ	Difference	log(max DP)	Difference	max Acc.
2	3	0x00 0b	-11.3	0x00 01	0.566
2	5	0x01 20	-11.0	0x00 01	0.547
2	7	0x01 80	-11.0	0x00 01	0.587
0	7	0x01 80	-11.0	0x00 01	0.688
0	5	0x01 20	-11.0	0x00 01	0.546
0	6	0x00 01	-10.4	0x00 01	0.546
0	2	0x00 01	-10.4	0x00 01	0.574
0	3	0x00 05	-10.3	0x00 01	0.547
2	6	0x00 01	-8.7	0x00 01	0.694
0	4	0x00 01	-7.4	0x00 01	0.701
2	4	0x00 01	-7.2	0x00 01	0.619
2	0	0x00 01	-6.8	0x00 01	0.737

Proof. Since $gcd(|\alpha-\beta|, n) = 1$ and n is even, $\alpha-\beta$ must be odd. This means that one of them (α or β) has to be even and the other one odd. Hence, by multiplying all constants with $s \in \mathbb{Z}_n^*$ we can bring the odd one to 1 while the even one remains even. For the sake of simplicity let us assume that $\alpha = 1$. Now we can choose b as either β if $\beta < n/2$ already holds or as $\beta + n/2$ by simply adding $n/2$ to all constants (and reducing modulo n). Of course, this means that we could change α from 1 to $1 + n/2$, but by multiplying all constants with $1 + n/2$ we can return α to $(1 + n/2)^2 = 1 \bmod n$ (since $n \geq 4$ even) and also leave $b < n/2$ untouched since $b \cdot (1 + n/2) = b + n \cdot \frac{b}{2} = b \bmod n$ as β and therefore b is even. The uniqueness follows from the fact that adding $n/2$ and multiplying by $s \in \mathbb{Z}_n^*$ is commutative and every combination of those can be written as a single multiplication by s and at most one addition of $n/2$. Adding $n/2$ or multiplying by s alone would violate the first constant being 1 while doing both would require $1 \cdot s + n/2 = 1$ for the first constant to remain 1, which means that s has to be $n/2 + 1$, i.e.

$$b \cdot s + n/2 = b \cdot (n/2 + 1) + n/2 = b + n/2 > n/2$$

since b is even and $b < n/2$. Hence, b and c are unique. □

This leaves us with the rotational constants

$$(\alpha, \beta, \gamma) \in \{(1, b, c) | b \in \{0, 2\}, \gamma \in \{0, 2, 3, 4, 5, 6, 7\} \backslash \{b\}\}. \quad (2)$$

We calculated the DDT for 7 rounds by making use of Theorem 3 from [KLT15] for those constants. Note that since the round function is invariant under rotations (i.e. rotation both words of the input by d is the same as doing so at the output) we only need to consider input differences up to rotational equivalence. Table 9 provides the maximal DP, as well as the maximal accuracy of a purely differential distinguisher, together with the input differences leading to those results. Here, it becomes already apparent that choosing the rotational constants such that the maximal DP is minimized, as it is normally done[10], does not necessarily minimize the accuracy a purely differential distinguisher is able to achieve.

We already know from Section 5.1 that an r-round distinguisher would use the DDT for $r-1$ rounds. Hence, we trained differential-neural distinguishers[11] on 8 rounds using

[10]Of course, other kinds of attacks then differential ones would also be considered.

[11]The hyper-parameters were chosen to be as in [Goh19][Section 4.2], i.e. no cipher specific hyper-parameter optimization was done.

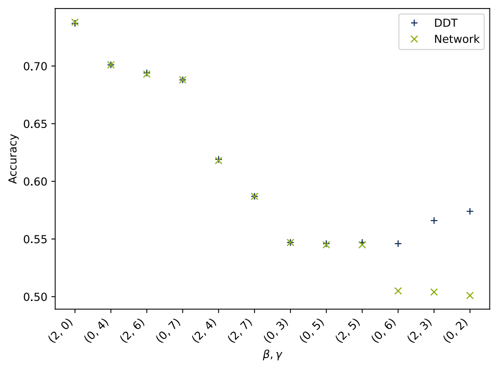


Figure 7: Network performance for small scale SIMON and different choices of rotational constants when using the input difference `0x00 01` and one additional round, i. e. 8 rounds, while the DDT is still calculated for 7 rounds. α is fixed to 1 and the results are ordered by the accuracy of the neural network.

the input difference `0x00 01`, which leads to the highest accuracies for the DDT-based distinguisher, no matter the rotational constants. As can be seen in Fig. 7, if the neural network did learn at all, the accuracies are virtually identical.

Note that the input difference `0x00 01`, which is found to be the best (in terms of accuracy), is rotational equivalent to every other input difference with only one active bit that is in the right words. In other words, the input difference we have chosen for SIMON32/64, namely `0x0000 0040`, is rotational equivalent to `0x0000 0001`, which is a reasonable candidate for the best input difference for the 32-bit version, based on the results we have seen here.

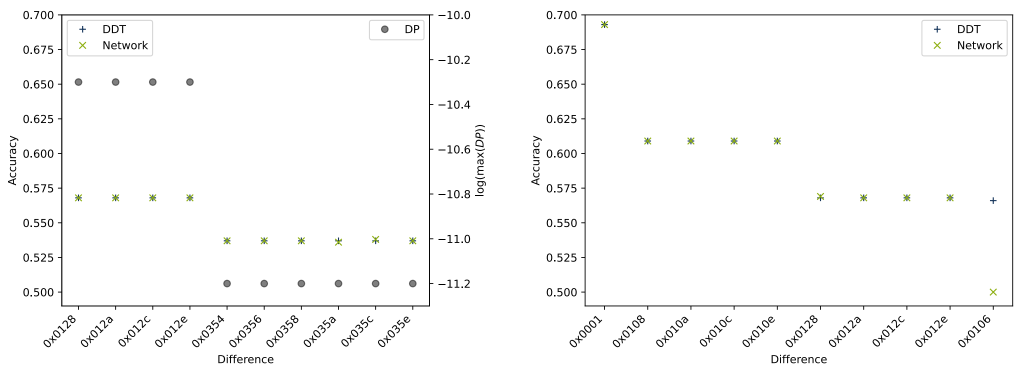
Next, let us fix the rotational constants to $(\alpha, \beta, \gamma) = (1, 2, 3)$, i. e. the ones that lead to the lowest maximal DP. Since 7 rounds seem to be harder for the neural network to learn, we will use 6 rounds instead, in order to have more meaningful results.

Starting with the input differences that lead to the maximal DP, it can be seen in Fig. 8a that both accuracies, the one based on the DDT and the one of the neural network, are again virtually identical. The same can be seen for input differences leading to maximal accuracy (based on the DDT), see Fig. 8b. Therefore, we conclude that for SIMON the neural network seems to be able to find a good approximation of the DDT, which may not be that interesting for the small scale variant, but, as we have already seen, also seems to be the case for SIMON32/64, and is therefore also likely to be true for variants with greater block sizes, where calculating the DDT becomes infeasible. Hence, such an approximation, especially for larger block sizes, could get quite useful.

Small Scale Speck For SPECK, we, again, need to chose the rotational constants. For this, we can make similar arguments. Once more, we need α and β to be not both divisible by 2 in order to achieve better diffusion. Also, $\alpha = n - \beta \bmod n$ (n being the word size) should be avoided, as the rotations would otherwise be identical. In addition, we incorporate the rational of the THREEFISH designers[FLS+10] and ignore the rotational constants -1, 0 and 1 since diffusion in adjacent bits should already be provided by the modular addition.

The best input differences for purely differential distinguishers on 6-round SPECK and different choices of rotational constants are given in Table 10. Note that, once more, the best input difference in terms of DP and the best in terms of accuracy based on the DDT are not identical.

In contrast to small scale SIMON, where the input difference `0x00 01` always lead to



(a) Top ten differences based on DP (b) Top ten differences based on accuracy (DDT)

Figure 8: Network performance for small scale Simon, using rotational constants $(\alpha, \beta, \gamma) = (1, 2, 3)$, for the top ten input differences (up to rotational equivalence) based on and ordered by DP (for 7 rounds) and accuracy (DDT) respectively. The network was trained on 7 rounds, but the accuracy based on the DDT is reported for 6 rounds.

the highest accuracy and is the natural counterpart to `0x0000 0040` (up to rotational equivalence), the choice of input difference is not as clear cut for the different variants of small scale Speck. But if we take a closer look at the input difference used for Speck32/64 in [Goh19], we notice that at the first modular addition only the msb of the left word is active. This leads to a deterministic first round transition and can easily be adapted for other choices of rotational constants.

Using such differences, we can see once more that the accuracy based on the DDT is a good approximation of the one of the neural distinguisher, see Fig. 9a. But as already noted in [Goh19], the neural distinguisher is able to surpass the purely differential one in most of the cases, and as can be seen in Fig. 9b, just optimizing the size of the filters enables the neural distinguisher to at lest catch up to the purely differential one in the cases where the purely differential distinguisher was superior before.

We would like to note that trying different filter sizes for small scale Simon did not make any significant difference. In other words, the version of Fig. 7 optimized for filter size would not look any different.

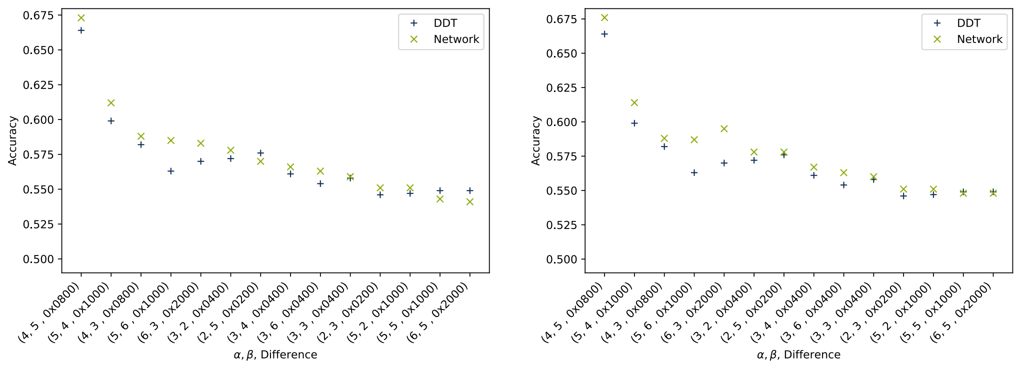
Given that the rotational constants $\alpha = 5$ and $\beta = 2$ lead to the lowest maximal DP, let us now fix those constants. We can see the top ten input differences, as well as the corresponding maximal DP and accuracies for both, the neural network, as well as the DDT, in Fig. 10a. Here, we would like to point out two things. Firstly, if the neural network is able to learn at all, it either reaches the same accuracy as the distinguisher based on the DDT or it surpasses it. Secondly, the figure definitely shows that there is no clear relation between the accuracy (based on the DDT) and the maximal DP, which is not surprising as we already know that one corresponds to the *mean* and the other to the *maximal* distance of the distributions.

Moving on to the top ten input differences according to the accuracy of the purely differential distinguisher, we can see in Fig. 10b that the accuracy based on the DDT is once more a good approximation for the one of the neural distinguisher, and, again, the neural distinguisher is at least as good as the purely differential one and even surpasses it in most cases.

If we take a closer look at the input differences that lead to the highest accuracies for small scale Speck, we see that the counterpart to the input difference used for Speck32/64 (i. e. `0x10 00`) does not lead to the highest accuracy. Hence, the question arises if we are able to find a better input difference based on those results. While it is not completely

Table 10: Maximal DP and accuracy (based on DDT) for 6-round small scale Speck and different choices of rotational constants together with the input difference which leads to those results, ordered by maximal DP.

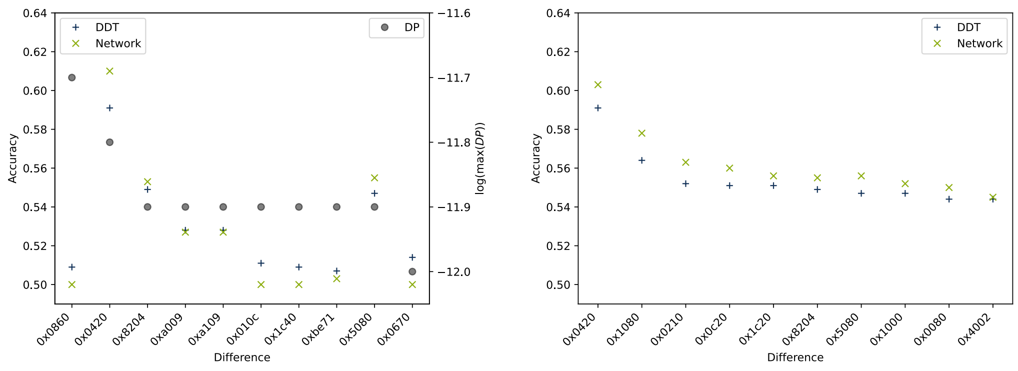
α	β	Difference	log(max DP)	Difference	max Acc.
5	2	0x08 60	-11.7	0x04 20	0.591
3	2	0x28 01	-11.7	0x01 20	0.580
3	6	0x84 98	-11.6	0x10 02	0.596
6	5	0x20 00	-11.0	0x01 04	0.581
5	6	0x20 01	-11.0	0x10 00	0.563
2	5	0x48 10	-11.0	0x10 04	0.587
6	3	0x28 a9	-11.0	0x04 10	0.580
5	4	0x12 91	-10.0	0x10 80	0.644
3	3	0x34 02	-10.0	0x24 80	0.592
5	5	0xd0 94	-10.0	0x12 80	0.598
2	3	0x02 80	-9.7	0x40 10	0.559
4	3	0x28 32	-9.0	0x48 80	0.599
3	4	0x01 20	-8.4	0x04 80	0.615
4	5	0x84 40	-8.0	0x08 00	0.664



(a) Without hyper-parameter optimization.

(b) Using filter sizes 3, 5 or 7.

Figure 9: Network performance for 6-round small scale Speck and different choices of rotational constants when using the input difference such that only the msb of the left side is active right before the modular addition in the first round, extended by the accuracy calculated using the DDT (for the same number of rounds). The results are ordered by the accuracy of the neural network in the left image.



(a) Top ten differences based on DP (b) Top ten differences based on accuracy (DDT)

Figure 10: Network performance for 6-round small scale Speck, using rotational constants $(\alpha, \beta) = (5, 2)$, for the top ten input differences based on and ordered by DP and accuracy (DDT) respectively.

clear how one can find a counterpart of one difference for a different variant of Speck, we will still try to do so based on the word size and the rotational constants. For the difference 0x04 20 for example, one can notice that after rotation the left side by α the two sides are identical, i. e. the differences in both words are the same right before the first modular addition. Furthermore, rotating the right side by β yields a difference in the msb only. The counterpart of 0x04 20 is therefore likely to be 0x0010 2000. But for Speck32/64 this difference leads to an accuracy of 0.595, which is slightly worse than the original input difference, but still the best result of all differences such that the two sides are identical up to rotating the left side by α. It hence looks reasonable that the choice of input difference in [Goh19] could already be optimal, at least if we consider the neural distinguisher on its own.

6 Real-Difference-Experiment

In [Goh19] the Real-Difference-Experiment is proposed to show that the neural distinguisher for Speck is not a purely differential distinguish. For this, 10^7 samples[12] of the form (c_1, c_2) are drawn from the encryption distribution. Then half of the samples get blinded as $(c_1 \oplus b, c_2 \oplus b)$ by independent blinding values b chosen uniformly at random. The resulting samples get judged by the distinguisher and the accuracy of distinguishing the blinded from the un-blinded samples is evaluated using the distinguishers trained for the scenario described in Section 2.2.

As explained in [Goh19], the rationale behind this is that the blinded samples contain the same ciphertext difference, but all additional information is lost. Hence, if only differential features would be used by a distinguisher the accuracy would be around 0.5. If the accuracy is significantly higher, then more than just the ciphertext difference is used.

Actually, the distinguishing performance on the non-blinded encryption samples will stay the same, as a sample labeled as encryption sample is still an encryption sample, while one labeled as random sample is now a blinded encryption sample. This means we could just compare the performance on random and blinded encryption samples, both labeled as random samples, which is typically called the True Negative Rate (TNR) in both, the default network evaluation, as well as the Real-Difference-Experiment. If the TNR would be the same in both cases, then the blinded encryption samples would look

[12]We increased the amount of samples from 10^6 to 10^7 in order to get more precise estimates for Katan.

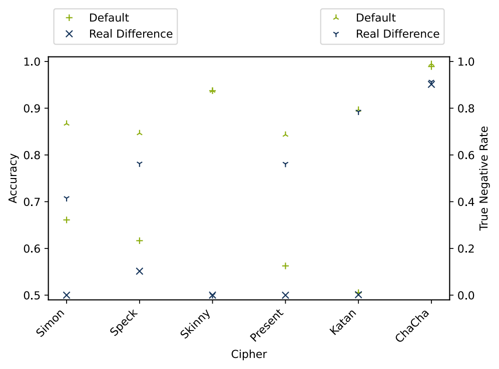


Figure 11: Results of Real-Difference-Experiment using the neural distinguisher from Table 5, except for Simon where we used a version of the neural distinguisher trained on samples where the last round was reverted, given the results from Section 5.1.

like random samples to the network. If it would go down, then at least some differential features get used.

Still, we think that the accuracy of the Real-Difference-Experiment is the more intuitive metric, which is why we will simply report both. If we take a look at the results in Fig. 11, we see that for Simon32/64, Skinny and Present the accuracy drops to 0.5 in the Real-Difference-Experiment, indicating that the neural distinguisher only makes use of the ciphertext difference (or more precise, the difference after a deterministic transformation of the ciphertexts in the cases of Simon and Skinny). While this was to be expected for Simon and Present, given the results from Section 5.1, for Skinny it is not as clear cut. Still, the experiment shows that for Skinny only differential features were used, too. Whether this is because other features do not exist or because the neural network was unable to learn them is not clear.

While the results for Katan look like also only differential features are used, a closer look on the accuracy of 0.501 reveals that, as it was evaluated using 10^7 samples, it still deviates significantly from 0.5, which means that more than purely differential features were used. Similar can be seen for ChaCha and Speck32/64, where the accuracy drops are not too high.

We would like to point out that while the results for Simon, Present and Skinny clearly indicate that only differential features are used, we cannot conclude the opposite for Katan and ChaCha, as we only considered one (natural) representation. If we, for example, look at Simon32/64 without backwards calculation, we see that the accuracy in the Real-Difference-Experiment would still be 0.645 (with a TNR of 0.708). In other words, we would need to repeat the Real-Difference-Experiment for all possible (deterministic) transformations of the ciphertexts (up to linear equivalence), if we would want to make such a conclusion plainly based on the Real-Difference-Experiment. Nevertheless, the results show that using the given representation, more than just differential features can be used. Additionally, at least for Speck, it was already show in [Goh19], and also in Section 5.3, that the neural distinguisher uses more than just differential features.

All in all, neural distinguishers are able to make use of more than just differential feature, even in a setting that is differential by nature (i. e. using a fixed input difference). It would therefore be interesting to classically explain such features, as this may have the potential to lead to new classes of attacks, and hence improve current security estimations as well as provide new guidance for cipher design.

7 Empirical Evaluation of Differential-Linear Biases

Next, we would like to distinguish the encryption distribution using statistical means. Given the complexity of this distribution, the idea is to reduce it to the distribution of one bit only. As it turns out, the found one-bit biases for the same amount of rounds as the differential-neural distinguishers cover are quite pronounced.

From Lemma 2 it is clear that reducing our analysis to (parts of) one ciphertext only would reduce the distribution to a uniform one. Therefore, we will look at one bit of the difference of the two ciphertexts of a ciphertext-pair[13]. In other words, we will empirically evaluate the differential-linear biases for each possible one bit linear mask, together with the input difference we used for our neural distinguishers.

For each of the six ciphers under scrutiny here, we show in Fig. 12 the differential-linear biases that pass a significance test with significance level 0.01 (after Bonferroni correction for the number of linear masks evaluated). Interestingly, the biases for Present are all negative, which is related to the fact that the output differences are biased towards lower weight. In addition, the biases for Skinny are clearly clustered into four bit blocks, which happen to align with the word structure of Skinny.

Unsurprisingly, the biases seem to be strongly related to the accuracy the neural disitnguishers are able to achieve. Both, the ChaCha and Skinny distinguisher achieve accuracies above 0.9, and have many high absolute biases. For Present, Simon and Speck the biases get lower and/or less frequent, and the accuracies of the neural distinguishers are between 0.56 and 0.66. For Katan, the biases are the lowest, as is the accuracy.

We would like to note that the lowest absolute bias passing the significance test would be around 2^{-11}. Since the amount of samples used here is the same as we train the neural networks with (i. e. 10^7), this means that a lower bias would be unlikely to even be noticeable during training. In contrast, such an absolute bias is not disproportionally low for it to be used in an attack. Of course, depending on the cipher there exist information not visible in the differential-linear biases, as it is the case for Speck, given that the neural distinguisher uses more than just differential features[Goh19], but our experiments still show a strong correlation between the biases and the networks performance.

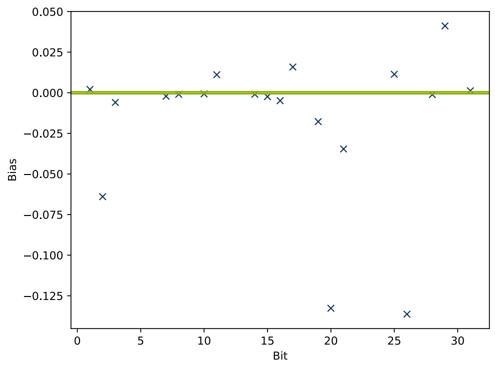
Given that we used the highest number of rounds for which we were able to directly[14] train a neural distinguisher, it looks like there needs to exist a certain degree of skewness in the encryption distribution for it to be distinguishable from the uniform one by neural networks as the one from [Goh19]. While this should be no surprise, the biases in Fig. 12 indicate that the degree of skewness may need to be relatively high. In other words, it is unlikely that neural distinguishers will be useful for breaking any reasonable (full round) cipher, but we still think that they can be a useful tool to evaluate its security, as they can be expected to procure good approximations of the DDT and/or ciphertext-pair-distribution for non-trivial numbers of rounds.

8 Using Multiple Ciphertext-Pairs

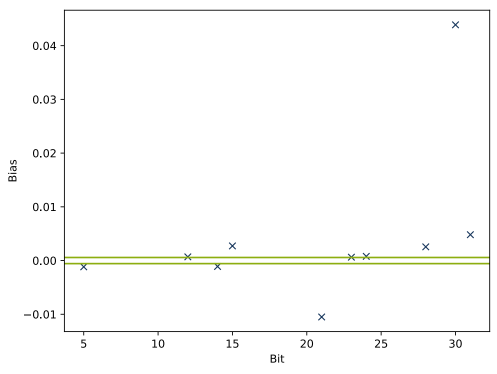
One current trend in the vicinity of machine learning assisted cryptanalysis is the use of multiple ciphertext-pairs per sample. While this could be a valid approach to improve upon using single pairs due to possible dependencies between the pairs, and especially could help to surpass *classical* distingiushers even for ciphers where this is not possible using single pairs (see Section 5), current literature fails to put the results into the right perspective, which we would like to chance.

[13]Strictly speaking, we revert deterministic parts of the last round function before calculating the ciphertext difference (except for present, as this would just result in a permutation of the bit positions), which is motivated by Section 5.1

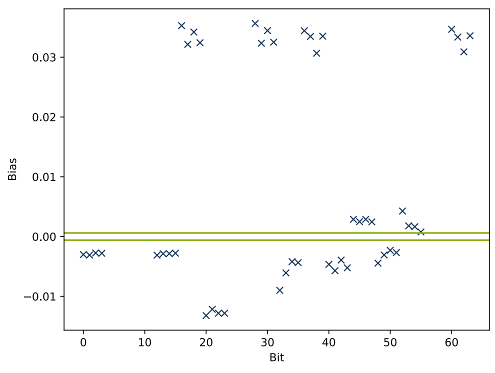
[14]I. e. without any form of pre-training or more elaborate training process.



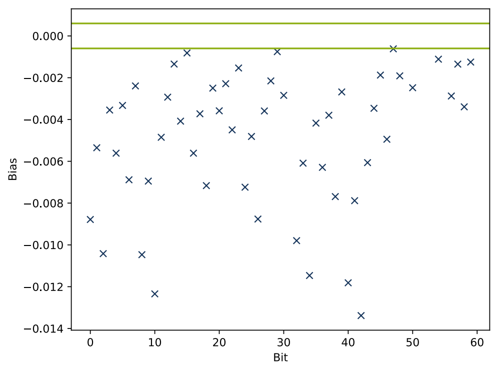
(a) 9-round Simon32/64



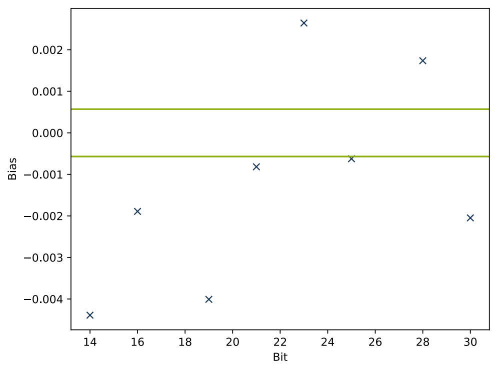
(b) 7-round Speck32/64



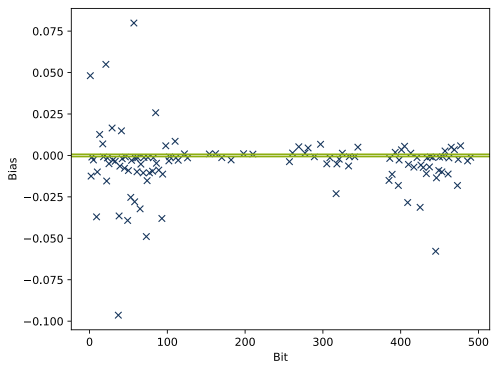
(c) 7- round Skinny



(d) 7-round Present



(e) 66-round Katan



(f) 3-round ChaCha

Figure 12: Differential-linear biases when using the usual input differences as well as one-bit linear masks, empirically evaluated using 10^7 samples. Results that do not pass a Bonferroni corrected significance test with significance level 0.01 are not shown. The two critical values for the significance test are shown by the two horizontal lines.

For this, it is important to note that differential-neural distinguishers, like the one from [Goh19], actually produces a score between 0 and 1 for each sample, representing its confidence for the sample to belong to the encryption class (instead of the random class). To evaluate the accuracy, we reduce this confidence into a hard decision by simply comparing it to 0.5, i. e. we loose information. But if we simply interpret each score as the probability of the sample to belong to the encryption class, and if we assume independence, we can actually make use of the whole score.

Corollary 2. *Let us assume that we have m samples, each drawn from the same distribution (encryption or random). Let us denote the probability that the i-th samples is an encryption samples as p_i. If the probabilities $p_1, ..., p_m$ are independent, then the probability that all m samples are drawn from the encryption distribution is*

$$\frac{1}{1 + \prod_{i=1}^{m} \frac{1-p_i}{p_i}}.$$

Note that this was already implicitly used in [Goh19], as a ranking of m ciphertext-pairs based on this combined probability is the same as one by

$$\sum_i \log\left(\frac{p_i}{1-p_i}\right),$$

which is how the keys are ranked in [Goh19] when using multiple ciphertext-pairs during trial decryption. In other words, [Goh19] already used multi-pair distinguishers, even if only implicitly.

Using this rule of combining probabilities/distinguisher responses, we are able to explicitly turn a distinguisher for one ciphertext-pair into one for an arbitrary amount of ciphertext-pairs. A comparison of such distinguishers and results from literature is shown in Table 2[15]. As can be seen, the claimed improvement is at best non-existent in most cases. In the case of [CSYY22], the multi-pair results are even worse than just using their single-pair distinguishers. Only [ZWW22] manage to improve over the combined distinguishers, but not as much as they reported, and in the case of 7- to 9-round Speck32/64, as well as 12-round Simon32/64 only.

To be fair to the authors, our distinguishers were not available for them to compare to, but at least comparisons to published distinguishers, such as the ones for Speck32/64 from [Goh19], or even to their own single pair distinguishers, would have been desirable.

In addition to the results in Table 2, Benamira *et al.*[BGPT21, Section 6] claim to improve the distinguisher from [Goh19] by averaging over the score of multiple (single-pair) samples or combining them using a neural network, effectively creating a multi-pair distinguisher. Unfortunately, they fail to notice that the key recovery attacks in [Goh19] effectively combine the scores as detailed above. Despite their claims, they do not manage to improve the results from [Goh19] at all, see Table 11.

Given the results for using multiple ciphertext-pairs when training a neural distinguisher, one could come to the conclusion that this may be the best way to learn more rounds. Indeed, if we look for example at the accuracies the authors of [ZWW22] achieved for 9-round Speck32/64 when using at least 8 pairs, it seems unlikely that we could tell an accuracy for one pair apart from 0.5 using 10^6 (or even 10^7) samples. In other words, it could be that single-pair distinguishers for 9-round Speck32/64 already were trained, but were not identified as being useful due to the noise that comes from experimentally evaluating the accuracy. A good example for this are our results for 66-round Katan.

[15]To avoid dividing by zero, we will use the number of times that $p_i = 0$ or $p_i = 1$ holds as a decision rule in case that at least one of the p_i is zero. Also, note that we used independent keys for each ciphertext-pair, but do not expect the results to look significantly different if the same key were to be used for all pairs contained in one sample.

Table 11: Accuracy (for Speck32/64) when using the score combination method implicitly used in [Goh19] compared to the ones proposed in [BGPT21].

Rounds	Pairs	[Goh19]	Averaging[BGPT21]	2D-CNN[BGPT21]
5	5	0.999	0.998	0.994
	10	1.000	1.000	1.000
6	5	0.967	0.954	0.933
	10	0.995	0.990	0.977
	50	1.000	1.000	1.000
7	5	0.743	0.735	–
	10	0.821	0.808	–
	50	0.977	0.967	–
	100	0.997	0.997	–

While a single pair leads to an accuracy of 0.505 only, which one could easily mistake for random noise, using (for instance) 8 pairs leads to an accuracy of 0.524, for which it is out of the gate more believable to be better than random guessing, but which is still way higher than the accuracy of 0.502 (reported in [ZWW22] for 9-round Speck32/64 and 8 pairs).

We would like to emphasise that we do think that using more than just one ciphertext-pair could be a viable way to improve on already existing distinguishers, as there should exist dependencies between the pairs that could be used, but we would like to ask for a fair comparison, for example as we did here. Especially, a fair comparison does not include using the same total amount of plain-/ciphertexts when evaluating the accuracy, but keeping the amount used for one sample constant. It should be clear that the amount of samples/trials only effects the quality of the accuracy estimation, but not the distinguishing performance itself.

9 Conclusion

We have seen that differential-neural distinguishers are quite generic and can be applied to a wide variety of ciphers and cipher designs. Furthermore, we provide guidance on how the network hyper-parameters should be chosen in order to produce potent differential-neural distinguishers. Despite the size of the explored search space, only a few hyper-parameters needed to be changed for the six ciphers under scrutiny, which makes it plausible that the same may be true for other ciphers as well.

In addition, we have seen that the best input difference with respect to the probability of a differential is not necessarily the best for differential-neural distinguishers, and also that there is a pronounced correlation between the accuracy a differential-neural distinguisher is able to achieve and the mean absolute distance between the output difference distribution and the uniform one, which is essentially the accuracy of a DDT-based distinguisher.

For a whole class of ciphers, we were able to see that only differential features can be used, assuming independent round keys. Furthermore, we have seen examples of ciphers not belonging to this class, where it is most likely that more than differential features are used by the differential-neural distinguishers presented in this work.

While the skewness of the encryption distribution most likely needs to be quite pronounced for the neural network to learn, this does not diminish the fact that differential-neural distinguishers seem to be able to learn a good approximation of the ciphertext-difference distribution, which, especially for greater block sizes, should be much faster than calculating the corresponding part of the DDT.

At last, we were able to rectify results from literature for using multiple ciphertext-pairs at once, showing that the claimed improvements are at best marginal, if not non-existent. This is especially important as the approach of combining distinguisher responses is by no means new, but is already used in the underlying work[Goh19], be it only implicitly. We hope that going forward, this gives authors a straightforward way of comparing such distinguishers to single-pair ones.

Discussion One downside of neural network based cryptanalysis is without doubt the explainability of the used features. It would be desirable to understand the neural-distinguishers decision in order to gain insights for (classical) cryptanalysis and cipher design. While there does exist work in this direction (e. g. [BGPT21]), it is (so far) distinguisher (and therefor cipher) specific. On the other hand, we were able to show that for a great number of ciphers any distinguisher in the differential setting we discussed here can at most make use of the ciphertext-difference distribution[16], which completely explains the features a neural network could use. For such ciphers, we can even see differential-neural distinguishers as a tool providing (in a reasonable amount of time) a lower bound on the mean absolute distance of the ciphertext-difference distribution and the uniform one, but with the drawback that it may be unclear how tight this bound is.

In addition, even if unknown features get used by a machine learning based distinguisher, knowing of the existence of such features is at least a first step in gaining insights, and is preferable over being unaware of such features.

Future Work We see multiple directions in the area of differential-neural cryptanalysis that could be worth exploring. For one, we were, despite a quite significant effort, not able to improve the distinguisher performance for Speck32/64 from [Goh19] by much, and also show that the ones for Simon are most likely almost optimal. It is therefore conceivable that the differential-neural distinguishers cannot be improved much further. But one way to still improve current machine learning based attacks is to make the distinguisher evaluations more efficient, for example by using a smaller neural network as in [BBP22]. During our hyper-parameter search for 7-round Speck32/64 we incidentally found a differential-neural distinguisher that has only 29,921 instead of 44,321[Goh19] overall parameters[17], but still achieves an accuracy of 0.616. Given that we did not try to optimize the number of parameters at all, it would be unreasonable to assume that this is as low as we can go.

Another approach could be to try to find better input differences. While our experiments using toy versions of Simon and Speck suggest that the input differences we use may already by optimal (with regards to distinguishing performance), it would be unreasonable to assume that the same is true for the other ciphers under scrutiny here.

On the other hand, it could be more beneficial to chose the input difference with the classical part in mind. In this work, we focused plainly on distinguishing performance, while [Goh19] augments their neural distinguisher using a classical differential. This is, of course, somewhat limited by the chosen input difference the neural distinguisher was trained with, especially if neutral bits for the differential transition should be used. It is therefore quite likely that the optimal input difference with regards to the distinguishing performance may not be the best for an overall attack.

Despite our findings that published multi-pair distinguisher do not perform much better than single pair ones, we still think that there should be a way to make use of the dependencies between the pairs. It would therefore be interesting to see if such

[16]If we assume independent round keys, which is a standard assumption in symmetric cryptanalysis.

[17]We provide this distinguisher as part of our source code, too, see https://github.com/differential-neural/An-Assessment-of-Differential-Neural-Distinguishers

distinguishers can indeed lead to greatly increased distinguishing performance, but it could be that the dependencies would need to be more pronounced for this to be possible.

Also interesting could be to try to find the best input difference by classical means. While it is possible that this is as hard as computing the DDT, given that the best input difference for a differential-neural distinguisher seems to be the one that maximizes the mean absolute distance between the ciphertext-difference distribution and the uniform one, this could also prove useful for general security estimations, as it would especially be a nice addition to just knowing the maximal distance, i. e. the maximal probability of a differential, since an adversary is by no means limited to one ciphertext-difference only.

At last, for ciphers like Simon32/64, where differential-neural distinguishers are unlikely to exceed a distinguisher based on the DDT, and where it is feasible to compute the needed part of the DDT, it could be interesting to see if an attack that basically replaces the differential-neural distinguisher by one based on the DDT in the attack from [Goh19] can be competitive with existing attacks. While evaluating the distinguisher could be a problem for the overall time complexity, making use of all possible output differences should be superior to using just a single one.

References

[AL13] Martin R. Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In Lars R. Knudsen and Huapeng Wu, editors, *SAC 2012*, volume 7707 of *LNCS*, pages 1–15. Springer, Heidelberg, August 2013.

[ALLW13] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential and linear cryptanalysis of reduced-round Simon. Cryptology ePrint Archive, Report 2013/526, 2013. https://eprint.iacr.org/2013/526.

[ALLW15] Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced Simon and Speck. In Carlos Cid and Christian Rechberger, editors, *FSE 2014*, volume 8540 of *LNCS*, pages 525–545. Springer, Heidelberg, March 2015.

[ASY$^+$19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In Ankur Teredesai, Vipin Kumar, Ying Li, Rómer Rosales, Evimaria Terzi, and George Karypis, editors, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*, pages 2623–2631. ACM, 2019.

[Ava17] Roberto Avanzi. Structure of KATAN/KTANTAN. https://www.iacr.org/authors/tikz/, 2017.

[BBCD21] Anubhab Baksi, Jakub Breier, Yi Chen, and Xiaoyang Dong. Machine learning assisted differential distinguishers for lightweight ciphers. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 176–181. IEEE, 2021.

[BBP22] Nicoleta-Norica Bacuieti, Lejla Batina, and Stjepan Picek. Deep neural networks aiding cryptanalysis: A case study of the speck distinguisher. In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings*, volume 13269 of *Lecture Notes in Computer Science*, pages 809–829. Springer, 2022.

[Bei16] Christof Beierle. Pen and paper arguments for SIMON and SIMON-like designs. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pages 431–446. Springer, Heidelberg, August / September 2016.

[Ber08] D. J. Bernstein. Chacha, a variant of salsa20, 2008. http://cr.yp.to/chacha/chacha-20080128.pdf.

[BGL+21] Zhenzhen Bao, Jian Guo, Meicheng Liu, Li Ma, and Yi Tu. Conditional differential-neural cryptanalysis. Cryptology ePrint Archive, Report 2021/719, 2021. https://eprint.iacr.org/2021/719.

[BGPT21] Adrien Benamira, David Gérault, Thomas Peyrin, and Quan Quan Tan. A deeper look at machine learning-based cryptanalysis. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 805–835. Springer, Heidelberg, October 2021.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153. Springer, Heidelberg, August 2016.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466. Springer, Heidelberg, September 2007.

[Bre17] Jakub Breier. ChaCha Quarter-Round. https://www.iacr.org/authors/tikz/, 2017.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. https://eprint.iacr.org/2013/404.

[CSYY22] Yi Chen, Yantian Shen, Hongbo Yu, and Sitong Yuan. A New Neural Distinguisher Considering Features Derived From Multiple Ciphertext Pairs. *The Computer Journal*, 03 2022. bxac019.

[DDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN - a family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, Heidelberg, September 2009.

[FLS+10] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, and Jesse Walker. The skein hash function family, 2010.

[Goh19] Aron Gohr. Improving attacks on round-reduced Speck32/64 using deep learning. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 150–179. Springer, Heidelberg, August 2019.

[HRC21a] Zezhou Hou, Jiongjiong Ren, and Shaozhen Chen. Cryptanalysis of round-reduced SIMON32 based on deep learning. Cryptology ePrint Archive, Report 2021/362, 2021. https://eprint.iacr.org/2021/362.

[HRC21b] Zezhou Hou, Jiongjiong Ren, and Shaozhen Chen. Improve neural distinguisher for cryptanalysis. Cryptology ePrint Archive, Report 2021/1017, 2021. https://eprint.iacr.org/2021/1017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[Jea15] Jérémy Jean. PRESENT block cipher. https://www.iacr.org/authors/tikz/, 2015.

[Jea16] Jérémy Jean. Skinny Round Function. https://www.iacr.org/authors/tikz/, 2016.

[JKM20] Aayush Jain, Varun Kohli, and Girish Mishra. Deep learning based differential distinguisher for lightweight cipher PRESENT. Cryptology ePrint Archive, Report 2020/846, 2020. https://eprint.iacr.org/2020/846.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 161–185. Springer, Heidelberg, August 2015.

[LLL+22] Jinyu Lu, Guoqiang Liu, Yunwen Liu, Bing Sun, Chao Li, and Li Liu. Improved neural distinguishers with (related-key) differentials: Applications in SIMON and SIMECK. Cryptology ePrint Archive, Report 2022/030, 2022. https://eprint.iacr.org/2022/030.

[LM02] Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In Mitsuru Matsui, editor, *FSE 2001*, volume 2355 of *LNCS*, pages 336–350. Springer, Heidelberg, April 2002.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. Cryptology ePrint Archive, Report 2013/328, 2013. https://eprint.iacr.org/2013/328.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and linear cryptanalysis using mixed-integer linear programming. In Chuankun Wu, Moti Yung, and Dongdai Lin, editors, *Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers*, volume 7537 of *Lecture Notes in Computer Science*, pages 57–76. Springer, 2011.

[SHW+14] Siwei Sun, Lei Hu, Meiqin Wang, Peng Wang, Kexin Qiao, Xiaoshuang Ma, Danping Shi, Ling Song, and Kai Fu. Towards finding the best characteristics of some bit-oriented block ciphers and automatic enumeration of (related-key) differential and linear characteristics with predefined properties. Cryptology ePrint Archive, Report 2014/747, 2014. https://eprint.iacr.org/2014/747.

[SKSS16] Anish Shah, Eashan Kadam, Hena Shah, and Sameer Shinde. Deep residual networks with exponential linear unit. *CoRR*, abs/1604.04112, 2016.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[SNPP19] Stefan Schubert, Peer Neubert, Johannes Pöschmann, and Peter Protzel. Circular convolutional neural networks for panoramic images and laser data. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 653–660, 2019.

[SZM20] Heng-Chuan Su, Xuan-Yong Zhu, and Duan Ming. Polytopic attack on round-reduced simon32/64 using deep learning. In Yongdong Wu and Moti Yung, editors, *Information Security and Cryptology - 16th International Conference, Inscrypt 2020, Guangzhou, China, December 11-14, 2020, Revised Selected Papers*, volume 12612 of *Lecture Notes in Computer Science*, pages 3–20. Springer, 2020.

[Tod15] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 287–314. Springer, Heidelberg, April 2015.

[XZBL16] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 648–678. Springer, Heidelberg, December 2016.

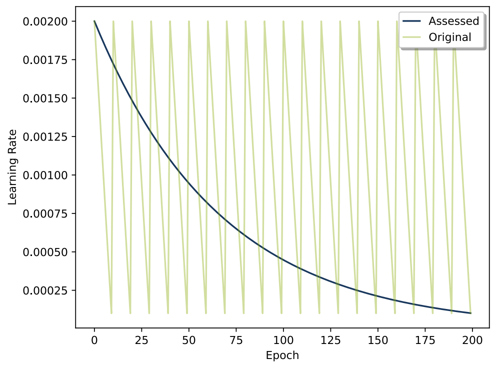
[ZW22] Liu Zhang and Zilong Wang. Improving differential-neural distinguisher model for DES, chaskey and PRESENT. Cryptology ePrint Archive, Report 2022/457, 2022. https://eprint.iacr.org/2022/457.

[ZWW22] Liu Zhang, Zilong Wang, and Boyang Wang. Improving differential-neural cryptanalysis with inception blocks. Cryptology ePrint Archive, Report 2022/183, 2022. https://eprint.iacr.org/2022/183.

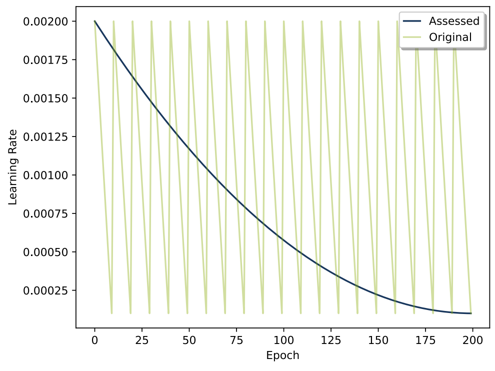
A Neural Network Hyper-Parameters

While most of the hyper-parameters we refer to in Section 3.2 should be clear to people familiar with neural networks, others are a bit ambiguous and require some explanation. Therefore, let us go through the list of considered hyper-parameters (Table 4) and give a bit more detail.

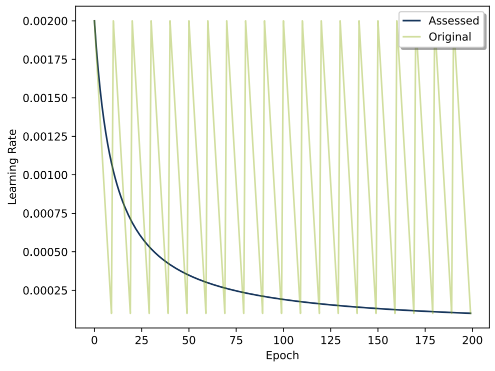
Learning Rates and Learning Rate Schedules In [Goh19] a cyclic LR schedule, which starts at $2 \cdot 10^{-3}$ (to which we refer as high LR), then linearly drops to 10^{-4} (low LR) over 10 epochs, and jumps back to $2 \cdot 10^{-3}$ completing the cycle, is used. Looking at the validation accuracy and loss during training, we had the conjecture that returning to the high LR of $2 \cdot 10^{-3}$ may prevent the neural network from refining the weights, as it seems to regularly get out of a (local and approximated) minimum (of the loss function). While this can be useful to find a better minimum, it could be beneficial to commit to one minimum at some point. With this reasoning, we tried the LR schedules depicted in Fig. 13, in addition to constant ones. In the end, we did not find the choice of a different LR schedule to give any significant improvement.



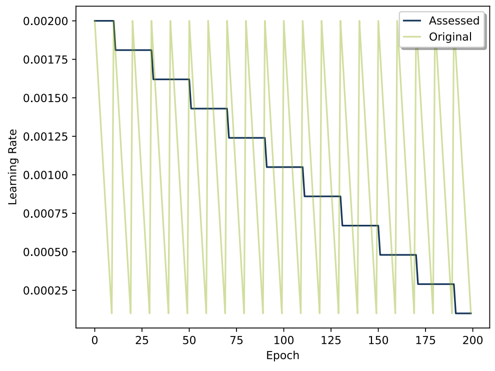
(a) Exponential Decay



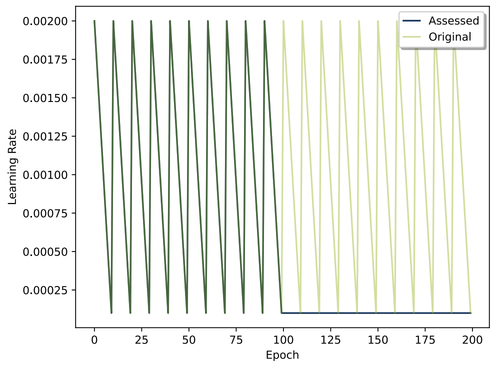
(b) Polynomial Decay



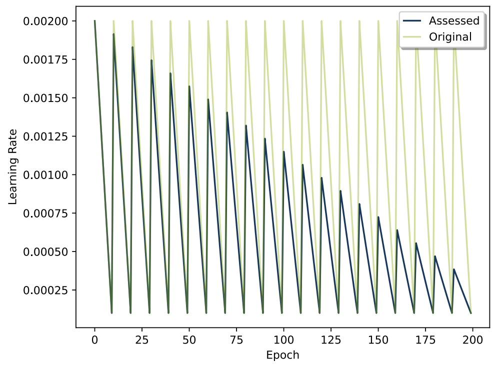
(c) Inverse Time Decay



(d) Piecewise Constant Decay

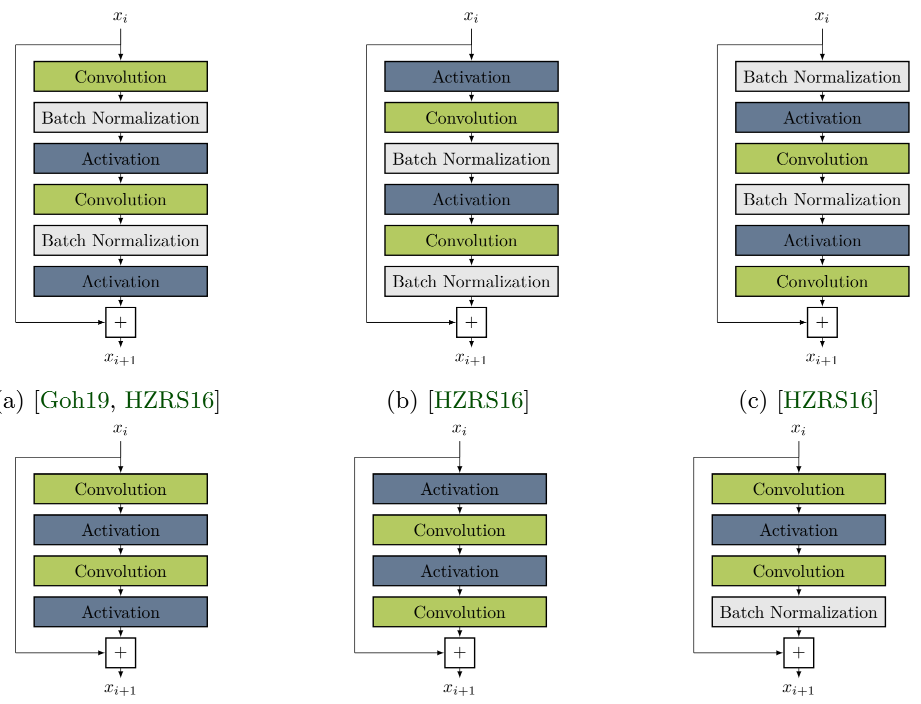


(e) Cyclic to Constant



(f) Linear Cyclic

Figure 13: Examples of tried LR schedules, compared to the cyclic one used in [Goh19] (*original*).



(a) [Goh19, HZRS16] (b) [HZRS16] (c) [HZRS16]

(d) [SKSS16] (e) [SKSS16] (f) [SKSS16]

Figure 14: Tried residual block designs.

Residual Block Design We tested a variety of different residual blocks, see Fig. 14, all using two convolutional layers. In addition, we also tried in-/decreasing the number of convolutional layers using the original design shown in Fig. 14a. Here, the number of convolutions actually refers to the number of applications of the composition of convolutional layer, batch normalization and activation.

Inception Given the importance of the filter size, we also tried to use Inception[SLJ$^+$14], which allows the use of multiple filter sizes at the same time, in a way enabling the network to learn the filter size. A graphic representation of our implementation can be found in Fig. 15. Since calculating multiple convolutional layers with different filter sizes instead of one convolutional layer with a fixed filter size is computationally more expensive, the authors of [SLJ$^+$14] proposed to reduce the number of filters for the layers with a filter size greater than one by adding additional convolutional layers with filter size one and less filters before these layers.

Given the increase in training time, but not in accuracy, we found it to be more efficient to directly search for the optimal filter size using a hyper-parameter search.

Circular Convolutions Based on the fact that many of the ciphers under scrutiny make use of some sort of bit cyclic shift (i. e. rotation), it seems reasonable that a cyclic representation would make more sense. To achieve this, Schubert *et al.*[SNPP19] propose to pad the input using data from the opposing side, instead of with zero. We adapted this approach to the neural network from [Goh19], and found it to make small improvements. But in total, those improvements also seem possible by tweaking the remaining hyper-parameters.

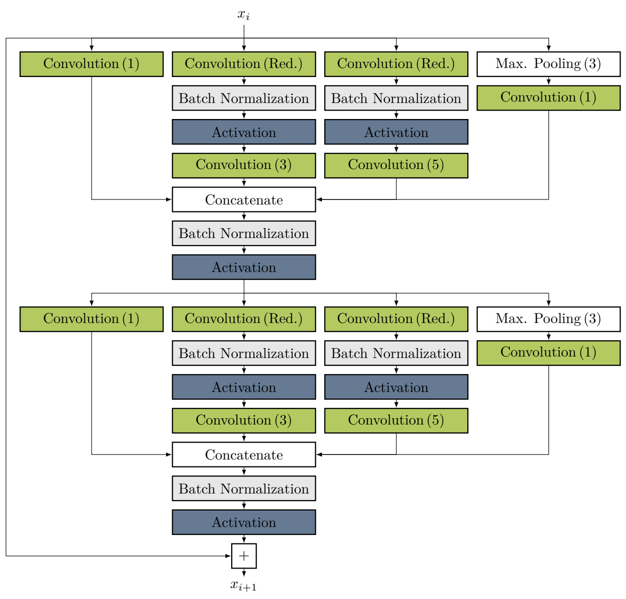


Figure 15: Residual block using the Inception Model. Convolutional layers marked with *(Red.)* are used for filter reduction and use filter size of one. All other convolutional layers have their filter size written in parentheses. For max. pooling, the number in parentheses represent the used pool size.

Dropout As an additional form or regularization, we also considered dropping out neurons in the densely connected layers, but did not find this to be helpful.

Batch Normalization The original network from [Goh19] already uses batch normalization. We therefore wanted to find out which effect it has on the performance of the neural distinguisher. As batch normalization after convolutions is already part of the residual block design, we tested removing batch normalization after the densely connected layers only, and found it to be quite essential.

Residual Scaling The residual block takes some input x, calculated some function F on it and adds the input to the result. While this can enable deeper neural networks in the first place, our idea is to control the impact of F by scaling it accordingly, i. e. instead of calculating $F(x) + x$, we calculate $\lambda \cdot F(x) + x$ for a scalar λ. This was found to have only a limited effect.

Bit Encoding In [Goh19] the samples are represented as arrays of bits, encoded as zero and one. But note that most layers of the neural network just perform matrix multiplication over real numbers. Therefore, using zero as one possible encoding could have the drawback that every multiplication with zero gives zero. Based on this observation, we tried encoding the bits as -1 and 1, as well as 1 and 2, but did not find this to make any difference.

B Reverting Deterministic Parts of the Katan Round Function

Let us take a look at how we calculate ciphertexts back for the cipher KATAN. Let $(x_0^r, \ldots, x_{31}^r) \in \mathbb{F}_2^{32}$ be the state of KATAN after round r, where $(x_0^r, \ldots, x_{12}^r)$ is the content of the first shift register and $(x_{13}^r, ..., x_{31}^r)$ the content of the second one. If we look at the structure of KATAN in Fig. 5, we know that

$$\begin{aligned} x_i^{r-j} &= x_{i+j}^r \text{ for } 0 \leq i \leq 12 - j \text{ and} \\ x_i^{r-j} &= x_{i+j}^r \text{ for } 13 \leq i \leq 31 - j. \end{aligned}$$

Let further $(k_a^r, k_b^r) \in \mathbb{F}_2^2$ be the round key and $IR^r \in \mathbb{F}_2$ the round constant of round r. Hence

$$\begin{aligned} x_{12}^{r-1} &= x_{13}^r \oplus x_8^{r-1} \cdot x_5^{r-1} \oplus x_7^{r-1} \oplus x_3^{r-1} \cdot IR^r \oplus k_a^r \text{ and} \\ x_{31}^{r-1} &= x_0^r \oplus x_{16}^{r-1} \cdot x_{21}^{r-1} \oplus x_{20}^{r-1} \oplus x_{23}^{r-1} \cdot x_{25}^{r-1} \oplus k_b^r. \end{aligned}$$

Therefore, we know every variable except the round keys for the last four rounds, which means we that we can calculate

$$\begin{aligned} x_{9+i}^{r-4} \oplus k_a^{r-i} &\text{ for } 0 \leq i \leq 3 \text{ and} \\ x_{28+i}^{r-4} \oplus k_b^{r-i} &\text{ for } 0 \leq i \leq 3. \end{aligned}$$

In other words, we know the state of four rounds before (up to the eight key bits). But since we only know $x_9^{r-4} \oplus k_a^r$ and not x_9^{r-4} itself, we can no longer calculate x_{12}^{r-5} as it would require us to know $x_8^{r-5} = x_9^{r-4}$. On the other hand, it is still possible to calculate at least parts of some more rounds backwards. While we do not know x_8^{r-5} and therefore cannot revert the addition of $x_8^{r-5} \cdot x_5^{r-5}$ to x_{12}^{r-5}, we can still revert the addition of $x_7^{r-5} \oplus x_3^{r-5} \cdot IR^{r-4}$. Furthermore, we can still calculate $x_{31}^{r-5} \oplus k_b^{r-4}$ and $x_{31}^{r-6} \oplus k_b^{r-5}$, before $x_{23}^{r-7} \cdot x_{25}^{r-7}$ becomes unknown. Hence, partly reverting six rounds gives

$$\begin{aligned} &x_i^{r-6} && \text{for } 0 \leq i \leq 6, \\ &x_i^{r-6} \oplus k_a^{r+7-i} && \text{for } 7 \leq i \leq 10, \\ &x_i^{r-6} \oplus x_8^{r+6-i} \cdot x_5^{r+6-i} \oplus k_a^{r+7-i} && \text{for } 11 \leq i \leq 12, \\ &x_i^{r-6} && \text{for } 13 \leq i \leq 25 \text{ and} \\ &x_i^{r-6} \oplus k_b^{r+26-i} && \text{for } 26 \leq i \leq 31. \end{aligned}$$

As already mentioned, we now no longer are able to calculate $x_{23}^{r-7} \cdot x_{25}^{r-7}$. Despite that, we can still continue to calculate part of the rounds backwards. After all, it takes another four rounds for x_{12}^{r-10} to become unknown (up to key addition), because we cannot calculate $x_3^{r-10} \cdot IR^{r-9}$. Also, $x_7^{r-10} = x_{11}^{r-6}$ which is not only masked by the round key k_a^{r-4}, but also by $x_8^{r-5} \cdot x_5^{r-5}$. Therefore, calculating nine rounds (partially) backwards gives

$$\begin{aligned} &x_i^{r-9} && \text{for } 0 \leq i \leq 3, \\ &x_i^{r-9} \oplus k_a^{r+4-i} && \text{for } 4 \leq i \leq 7, \\ &x_i^{r-9} \oplus x_8^{r+3-i} \cdot x_5^{r+3-i} \oplus k_a^{r+4-i} && \text{for } 8 \leq i \leq 12, \\ &x_i^{r-9} && \text{for } 13 \leq i \leq 22, \\ &x_i^{r-9} \oplus k_b^{r+23-i} && \text{for } 23 \leq i \leq 28 \text{ and} \\ &x_i^{r-9} \oplus x_{23}^{r+22-i} \cdot x_{25}^{r+22-i} \oplus k_b^{r+23-i} && \text{for } 29 \leq i \leq 31, \end{aligned}$$

but after that we can only revert the addition of $x_{16}^{r-10} \cdot x_{21}^{r-10}$ for the next round, whereas the one of x_{20}^{r-i} can still be reverted (up to addition of the round key) up to $i = 17$. Hence, partially reverting ten rounds gives

$$\begin{array}{ll}
x_i^{r-10} & \text{for } 0 \leq i \leq 2, \\
x_i^{r-10} \oplus k_a^{r+3-i} & \text{for } 3 \leq i \leq 6, \\
x_i^{r-10} \oplus x_8^{r+2-i} \cdot x_5^{r+2-i} \oplus k_a^{r+3-i} & \text{for } 7 \leq i \leq 11, \\
x_i^{r-10} \oplus x_8^{r-10} \cdot x_5^{r-10} \oplus x_7^{r-10} \oplus x_3^{r-10} \cdot IR^{r-9} \oplus k_a^{r-9} & \text{for } i = 12 \\
x_i^{r-10} & \text{for } 13 \leq i \leq 21, \\
x_i^{r-10} \oplus k_b^{r+22-i} & \text{for } 22 \leq i \leq 27 \text{ and} \\
x_i^{r-10} \oplus x_{23}^{r+21-i} \cdot x_{25}^{r+21-i} \oplus k_b^{r+22-i} & \text{for } 28 \leq i \leq 31.
\end{array}$$

We now can continue reverting the addition of x_{20}^{r-i} for three additional rounds until all values within the first register are masked by a key, i. e.

$$\begin{array}{ll}
x_i^{r-13} \oplus k_a^{r-i} & \text{for } 0 \leq i \leq 3, \\
x_i^{r-13} \oplus x_8^{r-1-i} \cdot x_5^{r-1-i} \oplus k_a^{r-i} & \text{for } 4 \leq i \leq 8, \\
x_i^{r-13} \oplus x_8^{r-1-i} \cdot x_5^{r-1-i} \oplus x_7^{r-1-i} \oplus x_3^{r-1-i} \cdot IR^{r-i} \oplus k_a^{r-i} & \text{for } 9 \leq i \leq 12, \\
x_i^{r-13} & \text{for } 13 \leq i \leq 18, \\
x_i^{r-13} \oplus k_b^{r+19-i} & \text{for } 19 \leq i \leq 24, \\
x_i^{r-13} \oplus x_{23}^{r+18-i} \cdot x_{25}^{r+18-i} \oplus k_b^{r+19-i} & \text{for } 25 \leq i \leq 28 \text{ and} \\
x_i^{r-13} \oplus x_{16}^{r+18-i} \cdot x_{21}^{r+18-i} \oplus x_{23}^{r+18-i} \cdot x_{25}^{r+18-i} \oplus k_b^{r+19-i} & \text{for } 29 \leq i \leq 31.
\end{array}$$

As already mentioned, the next obstacle is that $x_{20}^{r-18} = x_{25}^{r-13}$ is not known (even not up to key addition). But we can still (partially) revert 17 rounds.

$$\begin{array}{ll}
x_i^{r-17} \oplus x_8^{r-5-i} \cdot x_5^{r-5-i} \oplus k_a^{r-4-i} & \text{for } 0 \leq i \leq 4, \\
x_i^{r-17} \oplus x_8^{r-5-i} \cdot x_5^{r-5-i} \oplus x_7^{r-5-i} \oplus x_3^{r-5-i} \cdot IR^{r-4-i} \oplus k_a^{r-4-i} & \text{for } 5 \leq i \leq 12, \\
x_i^{r-17} & \text{for } 13 \leq i \leq 14, \\
x_i^{r-17} \oplus k_b^{r+15-i} & \text{for } 15 \leq i \leq 20, \\
x_i^{r-17} \oplus x_{23}^{r+14-i} \cdot x_{25}^{r+14-i} \oplus k_b^{r+15-i} & \text{for } 21 \leq i \leq 24, \\
x_i^{r-17} \oplus x_{16}^{r+14-i} \cdot x_{21}^{r+14-i} \oplus x_{23}^{r+14-i} \cdot x_{25}^{r+14-i} \oplus k_b^{r+15-i} & \text{for } 25 \leq i \leq 27 \text{ and} \\
x_i^{r-17} \oplus x_{16}^{r+14-i} \cdot x_{21}^{r+14-i} \oplus x_{23}^{r+14-i} \cdot x_{25}^{r+14-i} \oplus k_b^{r+15-i} \oplus k_a^{r+28-i} & \text{for } 28 \leq i \leq 31
\end{array}$$

From now on, we cannot even revert parts of the round function. Hence, this is the state we are referring to if we talk about ciphertexts calculated back.