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

Aron Gohr

Bundesamt für Sicherheit in der Informationstechnik (BSI), Germany,
aron.gohr@bsi.bund.de

Abstract. This paper has four main contributions.[1] First, we calculate the predicted difference distribution of Speck32/64 with one specific input difference under the Markov assumption completely for up to eight rounds and verify that this yields a globally fairly good model of the difference distribution of Speck32/64. Secondly, we show that contrary to conventional wisdom, machine learning can produce very powerful cryptographic distinguishers: for instance, in a simple low-data, chosen plaintext attack on nine rounds of Speck, we present distinguishers based on deep residual neural networks that achieve a mean key rank roughly five times lower than an analogous classical distinguisher using the full difference distribution table. Thirdly, we develop a highly selective key search policy based on a variant of Bayesian optimization which, together with our neural distinguishers, can be used to reduce the remaining security of 11-round Speck32/64 to roughly 38 bits. This is a significant improvement over previous literature. Lastly, we show that our neural distinguishers successfully use features of the ciphertext pair distribution that are invisible to all purely differential distinguishers even given unlimited data.
While our attack is based on a known input difference taken from the literature, we also show that neural networks can be used to rapidly (within a matter of minutes on our machine) find good input differences without using prior human cryptanalysis.

Keywords: Deep Learning · Differential Cryptanalysis · Speck

1 Introduction

1.1 Motivation and Goals of This Paper

Deep Learning has led to great improvements recently on a number of difficult tasks ranging from machine translation [7, 40] and autonomous driving [13] to

[1] Supplementary code and data for this paper is available at `https://github.com/agohr/deep_speck`

playing various abstract board games at superhuman level [16, 37, 38]. In cryptography, practical work using machine learning techniques has mostly focused on side-channel analysis [31, 34, 35]. On a theoretical level, it has long been recognized that cryptography and machine learning are naturally linked fields, see e.g. the survey of the subject given in [36]. Many cryptographic tasks can be naturally framed as learning tasks and consequently cryptographic hardness assumptions may for instance yield examples for distributions that are by design difficult to learn. However, not much work as been done on machine-learning based cryptanalysis. This paper is the first to show that neural networks can be used to produce attacks quite competitive to the published state of the art against a round-reduced version of a modern block cipher.

1.2 Contributions and Structure of This Paper

Main Results This paper tries to teach neural networks to exploit differential properties of round-reduced Speck. To this end, we train neural networks to distinguish the output of Speck with a given input difference from random data. To test the strength of these machine-learned distinguishers, we first calculate the expected efficiency of some multiple-differential distinguishers for round-reduced Speck32/64 that use the full Markov model of Speck32/64, i.e. all differential characteristics following a given input difference. This is the strongest form of differential distinguishing attack known that does not involve key search and to the best of our knowledge, the efficiency of distinguishing attacks of this kind has not been studied before for any Speck variant. A fairly high detection efficiency is achieved for up to about eight rounds past our chosen input difference.

Our neural distinguishers achieve better overall classification accuracy than these very strong baselines (see Table 2 for details). As an additional performance metric, we construct a simple partial key recovery attack on nine rounds of Speck using only 128 chosen plaintexts where the two types of distinguisher can be directly compared. In this test, we try to recover one subkey. The mean rank of this subkey is roughly five times lower with the neural distinguishers than with the difference distribution table. We explore this further by designing a cryptographic task in which the adversary has to distinguish two ciphertext pair distributions that have exactly the same ciphertext difference distribution. We find that our neural distinguishers perform fairly well in this game without any retraining, reinforcing the observation that the neural distinguishers use features not represented in the difference distribution table.

In order to allow for a direct comparison to existing literature, we also construct a partial key recovery attack against 11 (out of 22) rounds of Speck32/64 based on a lightweight version of our neural distinguishers. The attack is expected to recover the last two subkeys after $2^{14.5}$ chosen-plaintext queries at a computational complexity equivalent to about 2^{38} Speck encryptions; expected average wall time to recovery of the last two subkeys on a desktop computer under single-threaded CPU-only execution is about 15 minutes in our proof of concept implementation. The closest comparison to this in the literature might be the attack on Speck32/64 reduced to 11 rounds presented in [19], which needs

an expected 2^{14} chosen plaintexts to recover a Speck key with a computational effort of about 2^{46} reduced Speck evaluations. For a summary, see Table 1.

All experiments reported in this paper have been performed with a full implementation of Speck32/64, i.e. including the real key schedule. However, there is no evidence that the neural distinguishers use any properties of the key schedule. In particular, our 11-round key recovery attack has been tested also against reduced Speck32/64 with the free key schedule (independent and uniformly distributed subkeys), with no difference in performance compared to the real key schedule.

While other authors have tried to use neural networks for cryptanalytic tasks (see e.g. [6,15,17,18,26,28] and the references cited therein), this paper is to the best of our knowledge the first work that compares cryptanalysis performed by a deep neural network to solving the same problems with strong, well-understood conventional cryptanalytic tools. It is also to the best of our knowledge the first paper to combine neural networks with strong conventional cryptanalysis techniques and the first paper to demonstrate a neural network based attack on a symmetric cryptographic primitive that improves upon the published state of the art.

The comparison with traditional techniques serves in this paper both a benchmarking purpose and heuristically also as an additional safeguard against possible flaws in experimental setup. In the examples here considered, the performance of our deep neural networks is competitive with results obtained classically.

Table 1. Summary of key recovery attacks on 11 round Speck32/64. Computational complexity is given in terms of Speck evaluations on a modern CPU, i.e. assuming full utilisation of SIMD parallelism for fast key search.

Type	Complexity	Data	Source
Single-trail differential	2^{46}	2^{14} CP	[19]
Neural multiple differential	2^{38}	$2^{14.5}$ CP	This paper, section 4

Structure of the Paper In section 2, we give a short overview of the Speck family of block ciphers and fix some notations.

In section 3, we systematically develop new high-gain random-or-real differential distinguishers for round-reduced Speck based on an approach similar to that used on KATAN32 in [3]. For five to eight rounds of Speck32/64, we calculate for the first time the full distribution of differences within the Markov model for Speck induced by the input difference $0x0040/0000$ up to double precision rounding error. See Table 2 for details.

Section 4 contains our main results on using neural networks for cryptanalysis: we develop strong neural distinguishers against Speck reduced to up to eight rounds and show key recovery attacks competitive with classical methods for 9 and 11 rounds. We further show that using few shot learning techniques, fairly strong distinguishers against up to six rounds of Speck can be trained from very

small data sets (see Figure 2) and with very little computation. We use this further to automatically *find* good input differences for Speck32/64 without using prior human cryptanalysis.

In section 5, we further investigate the capabilities of our networks by introducing a differential cryptanalytic task which we call the *real differences experiment* where the distinguishers of section 3 are made useless. In this model, the adversary has to distinguish a real ciphertext $C = (C_0, C_1)$ obtained by encrypting two blocks of data P_0, P_1 with a known plaintext difference Δ from ciphertext that has additionally been bitwise-added with a random masking value $K_{out} \in \{0,1\}^b$, where b is the block size of the primitive considered.

We show that our best neural models for the main distinguishing task discussed in section 4 have discovered ways to win in this experiment significantly more often than random guessing without any retraining, although for the five-round case retraining is found to be quite helpful in extending this advantage. We also discuss a concrete example of a ciphertext pair that is misclassified by traditional differential distinguishers.

In section 6 we discuss our results and possible extensions of this work.

1.3 Related Work

Related Cryptographic Work Speck has since its publication [9] received a fair amount of analysis, see e.g. [8] for a review. We focus only on those works that are most relevant to the present paper.

The differential cryptanalysis of Speck was first studied by Abed, List, Lucks and Wenzel in [2]. They constructed efficient differential characteristics for round-reduced versions of all members of the Speck family of ciphers and showed how to use these for key recovery. For Speck32/64, the round 3 difference of their 9-round characteristic is used in the present work as the input difference required by our differential distinguishers.

In [19], Dinur improved the analysis given in [2] by using a two-round guess and determine attack to speed up key recovery and extend the number of rounds that can be attacked. The two-round guess and determine stage of this attack takes as input a bitwise input difference and the cipher output two rounds later and returns all possible solutions for the subkeys used in these final rounds. In section 3, we use this two-round attack to construct a practical distinguisher for Speck reduced to five rounds that exploits the nonuniformity of the ciphertext pair distribution perfectly in the setting where only the input difference but not the input values to the cipher are known.

Biryukov and Velichkov proposed in [11] a framework for the automatic search for optimal single differential characteristics of Speck and further improved on the differential characteristics found in [2]. They also showed that Speck is not a Markov cipher. This latter finding is reinforced by our finding that neural distinguishers on Speck can for a nontrivial number of rounds outperform in terms of prediction accuracy all purely differential distinguishers.

Differential attacks are most useful in the chosen plaintext setting, as the adversary needs to see the output of the primitive under study given plaintext

inputs with a particular chosen difference. The most powerful attacks against round-reduced Speck that have been put forward in the known plaintext setting come from linear cryptanalysis [5, 30].

Multiple differential cryptanalysis as an extension of truncated differential cryptanalysis was first studied by Blondeau and Gerard in [12]. A multiple-differential attack framework for block ciphers with small block size which (under the assumption that the relevant differential transition probabilities can be calculated correctly) exploits the difference between the wrong-key and right-key difference distributions perfectly was developed by Albrecht and Leander [3] and used to provide new cryptanalytic results on the KATAN32 block cipher, significantly extending the number of rounds that can be shown to be attackable.

Prior Work on Machine Learning and Data Driven Techniques in Cryptanalysis A number of works have explored the use of machine learning and broadly applicable statistical techniques for cryptanalytic purposes previously. We give a brief review here.

For the purpose of this review, precomputation attacks are generally viewed as not being machine learning. Likewise, side channel attacks and other ways of exploiting the implementation of a mechanism are considered not to be cryptanalysis in the sense here discussed.

Laskari, Meletiou, Stamatiou and Vrahatis [28] reported some success (in terms of search tree size, not necessarily in terms of execution time) compared to the baseline given by unoptimized brute force search in applying evolutionary computing methods to the problem of recovering additional subkey bits in four- and six round reduced DES subsequent to a classical differential attack.

Klimov, Mityagin and Shamir used genetic algorithms and neural networks to break a proposed public-key scheme itself based on neural networks [25]. The same protocol was broken in the same paper also using two other methods.

A few authors have looked at the possibility of using machine learning directly to distinguish between or to otherwise attack unreduced modern ciphers. From a cryptographic point of view, this is clearly expected to be impossible, at least unless mode-of-operation or other implementation issues make it feasible. This is e.g. also the conclusion reached by Chou, Lin and Chen [15], who perform some experiments along these lines and give a review of the literature.

Gomez, Huang, Zhang, Li, Osama and Kaiser [20] used unsupervised learning using neural networks to achieve code book recovery for short-period Vigenere ciphers in a setting in which neither parallel text nor information on the enciphering mechanism was available to the network during training. Their motivation was primarily to work towards unsupervised learning techniques for machine translation.

Abadi and Andersen [1] trained two neural networks to protect their communications from a third network that was trying to read their traffic. They showed that the two networks were in this setting able to use a pre-shared secret to shut out the adversary. However, neither analysis of humanly designed primitives nor

human cryptanalysis of the communication method developed by the networks was performed.

Rivest in [36] reviewed various connections between machine learning and cryptography. He also suggested some possible directions of research in cryptanalytic applications of machine learning.

Greydanus reported that recurrent neural networks can in a black box setting learn to simulate a restricted version of Enigma [21].

Purely data driven attacks have been used with good success e.g. against RC4 by Paterson, Poettering and Schuldt [32]. They basically learn from a very large amount of RC4 keystream examples a Bayesian model of single-byte and two-byte biases of RC4. This model is then used to derive some plaintext data given on the order of millions of encryptions of the same plaintext.

2 The Speck Family of Block Ciphers

2.1 Notations and Conventions

Bitwise addition will in the sequel be denoted by $\oplus$, modular addition modulo 2^n by $\boxplus$, and bitwise rotation of a fixed-size word by $\ll$ for rotation to the left and $\gg$ for rotation to the right. Here, k will be the word size of the primitive in question, which in the case of Speck32/64 is 16.

In this paper, differential cryptanalysis will always mean cryptanalysis with regards to bitwise differences in the adversary-controlled input to the cipher under study. Let hence $F : \{0,1\}^n \to \{0,1\}^m$ be a map. Then, a *differential transition* for F is a pair $(\Delta_{\text{in}}, \Delta_{\text{out}}) \in \{0,1\}^n \times \{0,1\}^m$. The probability $P(\Delta_{\text{in}} \to \Delta_{\text{out}})$ of the differential transition $F : \Delta_{\text{in}} \to \Delta_{\text{out}}$ is defined as

$$P(\Delta_{\text{in}} \to \Delta_{\text{out}}) := \frac{\text{Card}(\{x \in \{0,1\}^n : F(x) \oplus F(x \oplus \Delta_{\text{in}}) = \Delta_{\text{out}}\})}{2^n}. \quad (1)$$

In the description of differential attacks, it is sometimes necessary to specify specific ciphertext or plaintext differences or ciphertext/plaintext states. A single Speck block (or the difference between two blocks, depending on context) will in this paper be described by a pair of hexadecimal numbers. For instance, for Speck32/64, a state difference in which only the most significant bit is set will be written as $0x8000/8000$.

For a primitive iteratively constructed by repeated application of a simpler building block (i.e. a round function), a *differential characteristic* or *differential trail* will be a sequence of differential transitions, given by a sequence of differences $\Delta_0, \Delta_1, \ldots, \Delta_n$. When the same concepts are applied to key-dependent function families (e.g. block ciphers), any key dependence of the differential probability will usually be suppressed, although such key-dependencies can make a difference for security evaluation and although they are known to exist in ARX primitives (see e.g. [4]).

As introduced by Lai, Massey and Murphy [27], a *Markov cipher* is an iterated block cipher in which the probability of the individual differential transitions is independent of the concrete plaintext values if the subkeys applied to

each round are chosen in a uniformly random manner. It is common to suppress the effect of initial or final keyless permutations on the assessment of the Markov property, because the details of message modifying the data that goes into these initial or final permutations are outside the scope of differential cryptanalysis. In the case of Speck, the first round up to and excluding the first subkey addition is for instance a fixed initial permutation on the plaintext.

A *differential attack* is any cryptographic attack that uses nonrandom properties of the output of a cryptographic primitive when it is being given input data with a known difference distribution. The most general form of differential attack that has been formally discussed in the literature are *multiple differential attacks* [12], where information from an arbitrary set of differential transitions is exploited in order to maximise the gain of the resulting attack.

In this paper, we will see both attacks that only use the information contained in observed ciphertext *differences* and the full information contained in output ciphertext *pairs*. We will in the sequel call the former *purely differential* attacks and the latter *general differential* attacks.

A *distinguisher* is a classifier $\mathcal{C}$ that accepts as input d data sampled independently from a finite event space Ω according to one of n probability distributions $\mathcal{D}_i, i = 1, \ldots, n$, and outputs a guess of i for the submitted input item d. Here, i is chosen at each trial with a probability p_i from the set $\{1, 2, \ldots, n\}$. The selection method for i together with the distributions $\mathcal{D}_i$ is known in advance and is in this paper called an *experiment*.

2.2 A Short Description of Speck

Speck is an iterated block cipher designed by Beaulieu, Treatman-Clark, Shors, Weeks, Smith and Wingers [9] for the NSA with the aim of building a cipher efficient in software implementations in IoT devices [8]. It is an ARX construction, meaning that it is a composition of the basic functions of modular addition (mod 2^k), bitwise rotation, and bitwise addition applied to k-bit words. In [9], various versions of Speck were proposed, which differ from each other by the values of some rotation constants, the number of rounds suggested, as well as by the block and key sizes used. Generally, Speckn/m will denote Speck with n bit block size and m bits key size.

The round function $F : \mathbb{F}_2^k \times \mathbb{F}_2^{2k} \to \mathbb{F}_2^{2k}$ of Speck is very simple. It takes as input a k-bit subkey K and a cipher state consisting of two k-bit words (L_i, R_i) and produces from this the next round state (L_{i+1}, R_{i+1}) as follows:

$$L_{i+1} := ((L_i \gg \alpha) \boxplus R_i) \oplus K, R_{i+1} := (R_i \ll \beta) \oplus L_{i+1}, \tag{2}$$

where α, β are constants specific to each member of the Speck cipher family ($\alpha = 7, \beta = 2$ for Speck32/64 and $\alpha = 8, \beta = 3$ for the other variants).

The round function is applied a fixed number of times (for 22 rounds in the case of Speck32/64) to produce from the plaintext input the ciphertext output. The subkeys for each round are generated from a master key by a non-linear key schedule that uses as its main building block also this round function. Some

details of the key schedule differ between different versions of Speck due to the different number of words in the master key. The key schedule will not be analyzed in this paper and we therefore refer to [9] for reference.

3 Multiple Differential Attacks on Speck32/64

3.1 Pure Differential Distinguishers

Setting Multiple differential attacks [12] build cryptographic distinguishers by using a set $\mathcal{S}$ of differential transitions for some cryptographic function F to characterise its behaviour. The basic idea is that each transition $\Delta_i \to \delta_j$ in $\mathcal{S}$ has associated with it a probability p_{ij} of being observed given the experimental setting the cipher is being studied in and another probability $\tilde{p}_{ij}$ in some situation that is being distinguished against. Given some observed data $\mathcal{O}$ from the experiment, Bayesian inference can then be used to determine e.g. if the observed data comes from the real or the random experiment.

Calculating Differential Transition Probabilities We use algorithm 2 in [29] to compute the differential behaviour of the nonlinear component of Speck32/64, which is simply modular addition modulo 2^{16}. This gives us an efficient way to access arbitrary entries of the single-round differential transition matrix $A \in \mathbb{R}^{2^{32} \times 2^{32}}$ of Speck. Given an input difference distribution $v_i \in \mathbb{R}^{2^{32}}$ for round i of Speck, we calculate the distribution at the input of round $i+1$ by setting $v_{i+1} := Av_i$.

Starting from the input difference $\Delta = 0x0040/0000$, i.e. the round 3 difference of the differential characteristic given in Table 7 of [2], we have calculated the full predicted induced output distribution of Speck32/64 for up to 8 rounds in this way. The required sparse matrix-vector multiplications and the on-the-fly calculation of the relevant matrix entries took around 300 core-days of computing time in our implementation and produced about 34 gigabytes of distribution data for each round, which was saved to disk for further study. The input difference Δ is used in most distinguishers developed in the remainder of this paper. It transitions deterministically to the low-weight difference $0x8000/8000$ and has been chosen for being a very good starting point for truncated differential cryptanalysis in a low-data setting.

Cryptographic Tasks In this section, we set out to distinguish reduced-round Speck output with the input difference Δ from random data. Our distinguishers will use the full predicted output difference distribution for the number of rounds considered. We will denote by Di the resulting distinguisher for i rounds. Hence, D5 will e.g. be the resulting five-round distinguisher and the corresponding distinguishing problem will be referred to as the *D5 task*.

Classification To distinguish between examples of real ciphertext pairs and examples generated at random, we assume that random ciphertext pair differences are distributed according to the uniform distribution on nonzero ciphertext blocks. We classify an observed output difference δ as real if the predicted

probability of observing it in the real distribution is $> 1/(2^{32}-1)$ and as random otherwise. This exploits the non-uniformity of the output difference distribution perfectly if our prediction of this distribution does not contain errors. The reported true positive rates and accuracies for the distinguishers defined by the predicted output distribution were calculated under the assumption that the true output distribution is the predicted one.

Sources of Error This kind of calculation works only if the cipher under study does not deviate too strongly from the Markov property. Also, in our calculation we used double-precision arithmetic, which introduces rounding errors. We have therefore tested the validity of this model in three ways:

1. We checked empirically that the highest-probability transition found by our model for eight rounds ($0x0040/0000 \rightarrow 0x0280/0080$) is empirically observed with the expected probability of $2^{-26.015}$. This was found to be the case.
2. We checked empirically that the predicted true positive rates of our differential distinguishers match observed values on a size 10^6 test set from the real distribution. This was also the case within experimental error margins. The corresponding experiment for true negative rates was not performed, as the random distribution is a priori known exactly, so given the distinguisher, there is no error in predicting its accuracy on random samples.
3. We approximated the true difference distribution of Speck32/64 also empirically using 100 billion samples in each case. The resulting distinguishers were clearly inferior to our theoretical model.

These experiments indicate that our model captures the difference distribution of round-reduced Speck32/64 for the considered input difference quite well.

Distinguisher Accuracy and Key Rank Accuracy as well as true positive and true negative rate results are summarised in the next section, specifically in Table 2. Computing the full difference distribution table of Speck32/64 yields fairly strong distinguishers for at least up to eight rounds (better results than presented here may be possible with other input differences). Statistics on key ranking in the context of a simple key recovery attack can be found in Table 3.

3.2 Differential Distinguishers Using the Full Distribution of Ciphertext Pairs

Setting and Motivation The distinguishers so far considered in this paper observe a ciphertext pair that has been generated from a known input difference (but unknown input plaintext pairs) and try to guess based on the ciphertext difference whether the observed pair has been generated by reduced Speck encryption or randomly chosen. It is clear that one could improve on this by considering not only the difference data for an observed ciphertext pair, but the entire data

observed. However, this is more difficult, because calculating the full distribution of ciphertext pairs for the real distribution is not feasible. The goal of this section is to determine, for differential distinguishers on five-round Speck with the input difference used in the previously studied distinguishers, how much of an advantage the adversary might gain in still exploiting this additional information.

A Perfect Differential Distinguisher for Speck32/64 Reduced to Five Rounds We have developed a perfect distinguisher for the D5 task. Given an observed ciphertext pair $C := (C_0, C_1)$ and an input difference Δ, the likelihood $P(C|\text{real})$ that we would observe (C_0, C_1) under the real distribution for a block cipher E of block size b and key size k given uniformly random key and plaintext data is given by $2^{-(b+k)}N$, where N is the number of key and plaintext pairs (K, P) such that $E_K(P) = C_0$ and $E_K(P \oplus \Delta) = C_1$. But N is just the number $N_{\text{keys}}(C)$ of keys that decrypt C into a plaintext pair with difference Δ. On the other hand, $P(C|\text{random}) = 1/(2^{2b} - 2^b)$, so applying Bayes' theorem again, for perfect classification we need to determine whether $N_{\text{keys}}(C) > 2^{b+k}/(2^{2b} - 2^b) \approx 2^{b+k-2b}$ or not. For Speck32/64, we hence check whether $N_{\text{keys}} > 2^{32}$.

For the D5 task, it is possible to do this in practice by enumerating the possible round-3 differential states and then launching the two-round attack from [19] for each of these intermediate differences, enumerating the subkeys sk_5 and sk_4 used in rounds 4 and 5. After obtaining candidate round 3 output, we note that the round 1 output difference is known (the input difference transitions deterministically to 0x8000/8000) and use the two round attack again to recover the first two subkeys. We stop after $2^{32} + 1$ solutions have been found or the key space has been exhausted, whichever comes first. We tested this distinguisher on a test set of 10000 examples. 9456 of these were correctly classified, for an overall accuracy of about 95 percent. Replacing key search on the first two rounds with a (much faster) estimate of the number of solutions based on the 3-round difference distribution table did not lead to a statistically significant loss in performance.

4 Neural Distinguishers for Reduced Speck32/64

4.1 Overview

In this section, we will use neural networks to develop distinguishing attacks that try to solve the same problems as those presented previously. We only report results on our best neural models. The computational effort used in searching for a good architecture was not excessive; all machine learning experiments here reported were performed on a single workstation. Some other choices of architecture yield results that are also comparable or superior to the distinguishers presented in the previous section. One example of a simpler network architecture with still reasonable performance is shown in the github repository.

4.2 Network Structure

Input Representation A pair (C_0, C_1) of ciphertexts for Speck32/64 can be written as a sequence of four sixteen-bit words (w_0, w_1, w_2, w_3), mirroring the word-oriented structure of the cipher. In our networks, the w_i are directly interpreted as the row-vectors of a 4×16-matrix and the input layer consists of 64 units likewise arranged in a 4×16 array.

Overall Network Structure Our best network is a residual tower of two-layer convolutional neural networks preceded by a single bit-sliced convolution and followed by a densely connected prediction head. Deep residual networks were first introduced in [22] for image recognition and have been successful since in a number of other applications, for instance strategic board games [38, 39]. The results reported for five and six rounds use a depth-10 residual tower; for seven and eight round Speck, our final models use just a single residual block.

Initial Convolution The input layer is connected in channels-first mode to one layer of bit-sliced, e.g. width 1, convolutions with 32 output channels. Batch normalization is applied to the output of these convolutions. Finally, rectifier nonlinearities are applied to the outputs of batch normalization and the resulting 32x16 matrix is passed to the main residual tower.

Convolutional Blocks Each convolutional block consists of two layers of 32 filters. Each layer applies first the convolutions, then a batch normalization, and finally a rectifier layer. At the end of the convolutional block, a skip connection then adds the output of the final rectifier layer of the block to the input of the convolutional block and passes the result to the next block.

Prediction Head The prediction head consists of two hidden layers and one output unit. The first and second layer are densely connected layers with 64 units. The first of these layers is followed by a batch normalization layer and a rectifier layer; the second hidden layer does not use batch normalization but is simply a densely connected layer of 64 relu units. The final layer consists of a single output unit using a sigmoid activation.

Rationale The use of the initial width-1 convolutional layer is intended to make the learning of simple bit-sliced functions such as bitwise addition easier. The number of filters in the initial convolution is meant to expand the data to the format required by the residual tower. The choice of the input channels is motivated by a desire to make the word-oriented structure of the cipher known to the network. The use of a densely connected prediction head reflects the fact that for a nontrivial number of rounds, we do not expect the input data to show strong spatial symmetries, so any attempt to extract local features from the data using a spatially symmetric pooling layer of some sort is probably futile. The size of the layers was determined by experiment, although we tried only a few settings. The depth of the residual tower was chosen so as to allow for integration of input

data over the whole input string within the convolutional layers. However, even a design with just one residual block achieves reasonably good (clearly superior to a purely differential distinguisher) results.

4.3 Training Real vs Random Classifiers

Data Generation Training and validation data was generated by using the Linux random number generator (/dev/urandom) to obtain uniformly distributed keys K_i and plaintext pairs P_i with the input difference $\Delta = 0x0040/0000$ as well as a vector of binary-valued real/random labels Y_i. To produce training or validation data for k-round Speck, the plaintext pair P_i was then encrypted for k rounds if Y_i was set, while otherwise the second plaintext of the pair was replaced with a freshly generated random plaintext.

In this way, data sets consisting of 10^7 samples were generated for training. Preprocessing was performed to transform the data so obtained into the format required by the network. Data generation is very cheap. On a standard PC, it takes a few seconds to generate a data set of size 10^7 in our implementation.

Basic Training Pipeline Training was run for 200 epochs on the dataset of size 10^7. The datasets were processed in batches of size 5000. The last 10^6 samples were withheld for validation. Optimization was performed against mean square error loss plus a small penalty based on L2 weights regularization (with regularization parameter $c = 10^{-5}$) using the Adam algorithm [24] with default parameters in Keras [14]. A cyclic learning rate schedule was used, setting the learning rate l_i for epoch i to $l_i := \alpha + \frac{(n-i) \bmod (n+1)}{n} \cdot (\beta - \alpha)$, with $\alpha = 10^{-4}$,$\beta = 2 \cdot 10^{-3}$ and $n = 9$. The networks obtained at the end of each epoch were stored and the best network by validation loss was evaluated against a test set of size 10^6 not used in training or validation.

Improving the Distinguishers by Key Search We tested whether the distinguishers obtained can be improved by key search. To this end, a size one million test set for Speck reduced to seven rounds was generated as previously described. Each ciphertext pair c in the test set was then evaluated by performing brute force key search on the last round, grading the resulting partial decryptions using a six-round neural distinguisher, and combining the results into a score for the ciphertext pair c by transforming the scores into real-vs-random likelihood ratios and averaging. Algorithm 1 gives details on the method used.

Using key search as a teacher for a fast neural distinguisher The size one million sample set so obtained was further used as a training target for a single-block distinguisher against seven rounds of Speck. Training was performed from a randomly initialized network state for 300 epochs at batch size 5000 with a single learning rate drop from 0.001 to 0.0001 at epoch 200. Data on the resulting distinguisher can be found in Table 2 and Table 3.

Algorithm 1 KeyAveraging: Deriving a differential distinguisher against a block cipher E^{r+1} reduced to $r+1$ rounds for input difference Δ from a corresponding distinguisher $\mathcal{D}$ against E^r. A sample is predicted to come from the real distribution if and only if the output value of the algorithm is ≥ 0.5.

Require: Observed output ciphertext pair $C_0, C_1 \in \{0,1\}^b$
1: $D_i \leftarrow$ [DecryptOneRound(C_i, k) for $k \in$ Subkeys]
2: $v_k \leftarrow \mathcal{D}(D_0[k], D_1[k])$ for all $k \in$ Subkeys
3: $v_k \leftarrow v_k/(1-v_k)$ for all $k \in$ Subkeys
4: $v \leftarrow$ Average$([v_k, k \in$ Subkeys$])$
5: $v \leftarrow v/(1+v)$
6: **return** v

Training an 8-Round Distinguisher For 8 rounds, the training scheme described above fails, i.e. the model does not learn to approximate any useful function. We still succeeded in training an 8-round distinguisher slightly superior to the difference distribution table by using several stages of pre-training. First, we retrained our best seven-round distinguisher to recognize 5-round Speck32/64 with the input difference $0x8000/840a$ (the most likely difference to appear three rounds after the input difference $0x0040/0000$). This was done on 10^7 examples for ten epochs with a batch size of 5000 and a learning rate of 10^{-4}. Then, we trained the distinguisher so obtained to recognize 8-round Speck with the input difference $0x0040/0000$ by processing 10^9 freshly generated examples once with batch size 10000, keeping the learning rate constant. Finally, learning rate was dropped twice to 10^{-5} and finally to 10^{-6} after processing another 10^9 fresh examples each, again with a batch size of 10000.

Training Cost A single epoch of training according to the basic training schedule for one of our ten-block networks takes about 150 seconds on a single GTX 1080 Ti graphics card at batch size 5000. A full training cycle can therefore be run in less than a day, and results superior to the difference distribution table can be obtained in less than fifteen minutes after starting the training cycle for our neural distinguishers against 5 to 7 rounds.

4.4 Results

Test Set Accuracy We summarize data on our best models in Table 2. N5 and N6 are networks with ten residual blocks trained using the basic training method. N7 was trained to predict output of the KeyAveraging algorithm used with a six-round single-block neural distinguisher derived by knowledge distillation from N6 (see the paragraph on inference speed below for further details). N8 was derived from N7 using the staged training method described in section 4.3. The neural distinguishers achieve higher accuracies than the purely differential baselines discussed in the previous section on all tasks. The accuracy of the key search based distinguishers was not matched, as expected. Validation losses were only slightly lower than training losses at the end of training, suggesting that

only mild overfitting took place. An example learning history for a five-round network is shown in Figure 1. Algorithm 1 was tried on the seven round problem and did slightly improve prediction accuracy: ground truth was matched in 62.7 percent of the test sample.

If encryption is performed with fixed keys, a mild key dependency of distinguisher performance is observed, in line with previous work on Speck [4]. For instance, with 100 random keys, we found that true positive rates for the 7-round distinguisher empirically varied between 57.1 and 49.7 percent. See the github repository for code and data on this.

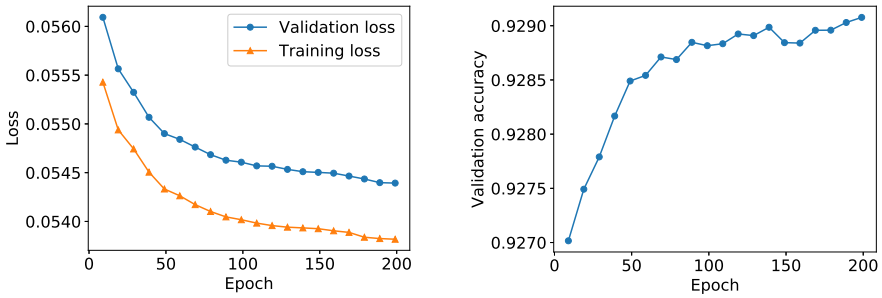

Fig. 1. Training a neural network to distinguish 5-round Speck32/64 output for the input difference $\Delta = 0x0040/0$ from random data. *(left)* Training and validation loss by epoch. *(right)* Validation accuracy. *(both)* Only data for epochs with lowest learning rate is shown. Intermediate epochs contained excursions to low performance. Full learning history for this run is available from supplementary data.

Key Ranking We can extend all of the distinguishers here discussed by one round at no additional cost by using the fact that the first subkey addition happens after the first application of nonlinearity in Speck. An adversary in the chosen-plaintext setting can easily inject plaintext differences of their choosing into the output of the first round of Speck. A simple attack on 9-round Speck can then be performed as follows:

1. Request encryptions for n chosen plaintext pairs $P_1, \ldots, P_n$ such that the output difference of the first round will be $\Delta = 0x0040/0000$. Obtain the corresponding ciphertext pairs $C_1, \ldots, C_n$.
2. For each value of the final subkey k, decrypt the C_i under k to get C_i^k. Let δ_i^k be the difference of the ciphertext pair C_i^k.
3. Use a 7-round differential distinguisher to get scores Z_i^k for each partially decrypted ciphertext pair.
4. For each k, combine the scores Z_i^k into one score v_k.
5. Sort the keys in descending order according to their score v_k.

Table 2. Accuracy of various distinguishers against Speck32/64 using two blocks of ciphertext with chosen plaintext difference $0x0040/0000$ for Nr rounds. D5-D8 are classical differential distinguishers that use the entire difference distribution table of Speck32/64 (calculated under the Markov assumption). N5-N8 are neural distinguishers solving the same distinguishing task. The accuracies of the D5-D8 distinguishers are theoretical predictions based on the assumption that they correctly predict the difference distribution, but have been empirically confirmed within 2σ error margins on size 10^6 test sets. The figures for the neural distinguishers were obtained by testing on size 10^6 test sets containing approximately 500000 positive and negative examples each. N5 and N6 are networks with ten residual blocks, while N7 and N8 are smaller networks with only one block.

Nr	Distinguisher	Accuracy	True Positive Rate	True Negative Rate
5	D5	0.911	0.877	0.947
5	N5	$0.929 \pm 5.13 \cdot 10^{-4}$	$0.904 \pm 8.33 \cdot 10^{-4}$	$0.954 \pm 5.91 \cdot 10^{-4}$
6	D6	0.758	0.680	0.837
6	N6	$0.788 \pm 8.17 \cdot 10^{-4}$	$0.724 \pm 1.26 \cdot 10^{-3}$	$0.853 \pm 1.00 \cdot 10^{-3}$
7	D7	0.591	0.543	0.640
7	N7	$0.616 \pm 9.7 \cdot 10^{-4}$	$0.533 \pm 1.41 \cdot 10^{-3}$	$0.699 \pm 1.30 \cdot 10^{-3}$
8	D8	0.512	0.496	0.527
8	N8	$0.514 \pm 1.00 \cdot 10^{-3}$	$0.519 \pm 1.41 \cdot 10^{-3}$	$0.508 \pm 1.42 \cdot 10^{-3}$

We have implemented this attack both with the 7-round distinguisher derived from the difference distribution of 7-round Speck with the given input difference and with our 7-round neural distinguisher. In the case of the neural distinguisher, we used the formula

$$v_k := \sum_{i=1}^{n} \log_2(Z_i^k/(1 - Z_i^k)) \tag{3}$$

to combine the scores of individual decrypted ciphertext pairs into a score for the key; in the case of the difference distribution table, we set

$$v_k := \sum_{i=1}^{n} \log_2(P(\delta_i^k)), \tag{4}$$

where $P(\delta_i^k)$ is the probability according to the difference distribution table of observing the output difference δ_i^k in the output of Speck32/64 reduced to seven rounds given the input difference Δ. This is comparable, since in both cases we can up to a constant multiplicative factor heuristically treat the summed terms as logarithms of real-vs-random likelihood ratios[2].

We chose $n = 64$ for this experiment. In this setting, we found that the neural distinguishers achieved much better key ranking (Table 3).

It is worth noting the following:

[2] As an implementation remark, note that with the neural networks *used in this paper*, the individual terms in the sum of Equation 3 are up to a scale factor just the neural network outputs before application of the final sigmoid activation.

Table 3. Statistics on a key recovery attack on 9-round Speck32/64. The same attack using 128 chosen plaintexts is executed using both a distinguisher based on the difference distribution table and a neural distinguisher against Speck32/64 reduced to 7 rounds. All values reported are based on 1000 trials of the respective attacks. Reported error bars around the mean are for a 2σ confidence interval, where σ is calculated based on the observed standard deviation of the key rank. The rank of a key is defined as the number of subkeys ranked higher, i.e. rank zero corresponds to successful key recovery. When several keys were ranked equally, the right key was assumed to be in a random position among the equally ranked keys. Key rank data on all runs as well as data on runs with 64 and 256 chosen plaintexts is available from the github repository.

Distinguisher	Mean of key rank	Median key rank	Success rate
D7	263.9 ± 77.7	9.0	0.13
N7	52.1 ± 34.7	1.0	0.358

Proposition 1. *Assume that E is any Speck variant with a free key schedule and that $\mathcal{A}$ is an attack that tries to recover the Speck key used using purely differential methods, i.e. assume that it gets as input plaintext differences $P_0 \oplus P_1, P_0 \oplus P_2, \ldots, P_0 \oplus P_n$ as well as ciphertexts $C_0, C_1, \ldots, C_n$. Then full key recovery can never be successful with a success rate beyond 50 percent.*

Proof. To see this, consider any pair of ciphertexts C_0, C_1 and a Speck subkey k. Suppose that $E_k^{-1}(C_0) \oplus E_k^{-1}(C_1) = \delta$, where E_k denotes single-round encryption under the subkey k. Flip the most significant bit of k and call the resulting new subkey k'. Then it is straightforward to verify that $E_{k'}^{-1}(C_0) \oplus E_{k'}^{-1}(C_1) = \delta$ as well. The proposition follows by applying this reasoning to the first round of E.

Hence, purely differential distinguishers for Speck always produce pairs (or larger groups) of equally ranked subkeys until the key schedule can be used to rule out candidate keys.

However, already with 256 chosen plaintexts our 9-round attack does in fact yield a success rate of about 70 percent if a neural distinguisher is used. This is only possible because our neural distinguishers are not purely differential distinguishers. See Section 5 for further evidence of this.

Inference Speed The deep residual architecture described yields networks that are still reasonably efficient to evaluate. On a single machine using a GTX 1080 Ti graphics card we were able to process roughly 200000 ciphertext pairs per second with our deeper networks (10 residual blocks). For single-block networks, we reach about one million ciphertext pairs per second on the same hardware. For seven and eight rounds of Speck, the best networks here presented are single-block networks; for five and six rounds, one can produce networks that are almost as good as the ten-block architecture by using a simple form of knowledge distillation [23]. For instance, a single-block network with 92.7 percent accuracy on the D5 task can be obtained by training against the output of our ten-block network for 30 epochs on a size 10^7 training dataset with a single learning rate drop from 0.001 to 0.0001 at epoch 20.

Disagreement with Difference Distribution Table For five-round Speck, we generated a size one million test set and calculated for each example both the relevant entry of the difference distribution table for Speck32/64 using the Markov model of Speck, and the output of a five-round one-block neural predictor. Exactly half of the test sample was generated using the real distribution, with the other half being drawn at random. We used this data set to study disagreements between the neural predictor and the difference distribution table.

Disagreement between both predictors was observed in 48826 samples, of which the majority was from the random distribution (about 57 percent). Our neural network chose the classification corresponding to ground truth in 67 percent of these cases of disagreement.

However, exploitation of information that can be obtained reliably from the difference distribution table was not perfect. For instance, 1549 of our samples were found to correspond to impossible differential transitions. Two of these were misclassified by the neural network as coming from the real distribution, although in both cases the confidence level returned by the neural network output was low (56 percent and 53 percent respectively).

On the other hand, the neural network also successfully identified output pairs that could not have appeared in the real distribution. For instance, the lowest neural network score on the set of disagreements was obtained for the output pair $(c_0, c_1) := (0\text{xc65d2696}, 0\text{xa6a37b2a})$. This corresponds to an output difference of 0x60fe/5dbc, and the transition 0x0040/0000 $\rightarrow$ 0x60fe/5dbc for five rounds has a transition probability of about 2^{-26} according to the Markov model. Accordingly, the predictor based on the difference distribution table assigns a 98 percent probability to this output pair being from the real distribution, whereas our neural distinguisher returns an almost zero score. Performing optimized key search, we found that there is in fact no key that links this output pair to a possible intermediate difference in round 3. Indeed, the sample had come from the random distribution in our test set.

Few-Shot Learning of Cryptographic Distributions Few-shot learning is the ability of people (and sometimes machines) to learn to recognize objects of a certain category or to solve certain problems after having been shown only a few or even just *one* example. We tested if our neural networks can successfully perform few-shot learning of a cryptographic distribution given knowledge of another related distribution by performing the following experiment: a fresh neural network with one residual block was first trained to recognize Speck reduced to three rounds with a fixed but randomly chosen input difference. Training consisted of a single epoch of 2000 descent steps with batch size 5000, which on our hardware corresponds to about a minute of training time. We then accessed the output of the second-to-last layer of this network, treating it as a representation of the input data. We generated small samples (only real examples, specifically between 1 and 50 of them) of the output distribution for six rounds of Speck with the chosen input difference of our main distinguishers. Each example set so created was complemented by the same number of samples drawn from the random distribution. The resulting example set S was sent through the neural network to

obtain the corresponding set $S' \subset \mathbb{R}^{64}$ of internal representation vectors. Ridge regression (with regularization parameter $\alpha = 1$) was used to create from this small training set a linear predictor $L : \mathbb{R}^{64} \to \mathbb{R}$ for the six-round distribution minimizing the squared error between labels and predictions on S'. We classified an example $x \in S'$ as real if $L(x) > 0.5$ and as random otherwise. This predictor was then tested on a size 50000 test set to determine its accuracy. This worked well even with a single-figure number of examples. Figure 2 gives detailed results, Algorithm 2 summarizes the algorithm used.

Algorithm 2 TrainByTransfer: Training a distinguisher for a block cipher with block size b reduced to r rounds E^r with input difference δ by transfer learning given an auxiliary neural distinguisher N for input difference Δ and E^s.

Require: N, r, δ, n

1: $X_0 \leftarrow n$ samples drawn from the real output distribution of E^r with input difference δ.
2: $Y_0 \leftarrow (1, 1, ..., 1) \in \mathbb{R}^n$
3: $X_1 \leftarrow n$ samples drawn uniformly at random from $\{0, 1\}^{2b}$.
4: $Y_1 \leftarrow 0 \in \mathbb{R}^n$.
5: $N' \leftarrow N[-2]$, where $N[-2]$ denotes the output of the second-to-last layer of N.
6: $Z, Y \leftarrow N'(X_0||X_1), Y_0||Y_1$
7: $L \leftarrow \text{RidgeRegression}(Z, Y)$
8: **return** $L \circ N'$

Training a new distinguisher using Algorithm 2 is very efficient. For instance, retraining on a thousand example training set takes about a millisecond on our platform.

Deriving Good Input Differences Without Human Knowledge This few-shot learning capability allows us to very quickly derive a rough lower bound for the effectiveness of truncated differential distinguishers for Speck for a given input difference and a given number of rounds. Concretely, given a pre-trained network for three-round Speck and a random input difference δ, we can quickly train a distinguisher for another random input difference δ' and evaluate its accuracy on a small test set. Starting with a random δ', we then use Algorithm 3 to optimize δ' for test set accuracy of the resulting distinguishers. Using $\alpha = 0.01, t = 2000$, and test and training datasets of size 1000 for each new input difference to be tested, we need less than two minutes of computing time for the training of the initial three-round distinguisher and about 15 seconds for each full run of Algorithm 3 on our platform. The initial difference of our main neural distinguishers is usually found within a few random restarts of the greedy optimizer and extending the number of rounds to be attacked, one can then easily show using transfer learning that at least six rounds of Speck can be distinguished with fairly high gain.

It seems likely that other generic optimization algorithms, e.g. suitable variants of Monte Carlo Tree Search, might work even better.

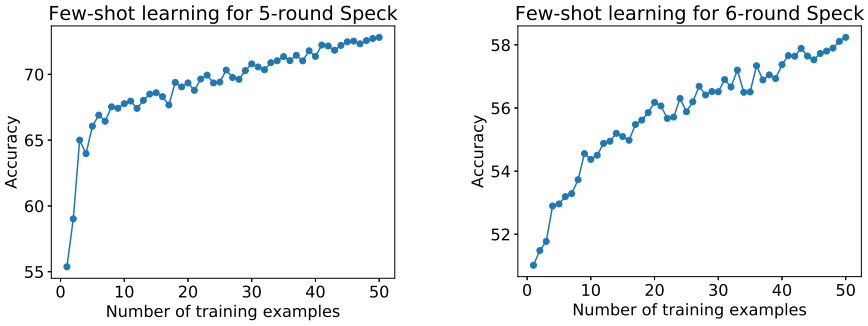

Fig. 2. Few-shot learning on the D5 and D6 tasks using a pre-trained classifier to preprocess the input data. Algorithm 2 was used with a fixed auxiliary network trained to distinguish Speck32/64 reduced to three rounds with a random fixed input difference. The number of training examples supplied was varied from 1 to 50. The accuracy figures shown are an average over 100 runs for each training set size, where for each training run a fresh training set of the indicated size was generated on the fly. Accuracy was measured against a fixed test set of size 50000. Measured accuracy is above guessing at 2σ significance level even for a single training example.

Algorithm 3 GreedyOptimizerWithExplorationBias: Given a function $F : \{0,1\}^b \to R$, try to find $x \in \{0,1\}^b$ which maximises F.

Require: F, number t of iterations, exploration factor α, input bit size b

$x \leftarrow \text{Rand}(0, 2^b - 1)$
$v_{\text{best}} \leftarrow F(x)$
$x_{\text{best}} \leftarrow x$
$v \leftarrow v_{\text{best}}$
$H \leftarrow$ hashtable with default value 0
for $i \in \{1, \ldots, t\}$ **do**
  $H[x] \leftarrow H[x] + 1$
  $r \leftarrow \text{Rand}(0, b - 1)$
  $x_{new} \leftarrow x \oplus (1 \ll r)$
  $v_{\text{new}} = F(x_{\text{new}})$
  if $v_{\text{new}} - \alpha \log_2(H[x_{\text{new}}]) > v - \alpha \log_2(H[x])$ **then**
    $v, x \leftarrow v_{\text{new}}, x_{\text{new}}$
  end if
  if $v_{\text{new}} > v_{\text{best}}$ **then**
    $v_{\text{best}}, x_{\text{best}} \leftarrow v, x$
  end if
end for
return x_{best}

4.5 Key Recovery Attack

To showcase the utility of our neural distinguishers as research tools, we have constructed a partial-key recovery attack based on the N7 and N6 distinguisher that is competitive to the best attacks previously known from the literature on Speck32/64 reduced to 11 rounds, i.e. in particular to the 11-round attack of [19]. The attack proposed by Dinur has a computational complexity approximately equivalent to 2^{46} Speck evaluations. The attack is expected to succeed after querying 2^{13} chosen-plaintext pairs and obtaining the corresponding ciphertexts. Attacks on 12 to 14 rounds were also proposed in [19], naturally with substantially larger computational and data complexities.

Our eleven-round attack, in contrast, is expected to succeed with a computational complexity of roughly 2^{38} Speck evaluations if it is executed on a CPU. Its data complexity is slightly higher than that of the attack in [19]; however, computational complexity is reduced by a factor of more than 200.

Basic Attack Idea

Overview The idea of our attack is to extend our neural 7-round distinguisher to a 9-round distinguisher by prepending a two-round differential transition $\delta \rightarrow$ 0x0040/0000 that is passed as desired with a probability of about 1/64. The 9-round distinguisher is then extended by another round at no additional cost by asking for encryptions of ciphertext pairs P_0, P_1 that encrypt to the desired input difference δ after one round of Speck encryption; this is easy, since no key addition happens in Speck before the first nonlinear operation.

The signal from this distinguisher will be rather weak. We therefore boost it by using k (probabilistic) neutral bits [10] to create from each plaintext pair a plaintext structure consisting of 2^k plaintext pairs that are expected to pass the initial two-round differential together. For each plaintext structure, we decrypt the resulting ciphertexts under all final subkeys and rank each partially decrypted ciphertext structure using our neural distinguisher. If the resulting score is beyond a threshold c_1, we attempt to decrypt another round and grade the resulting partially-decrypted ciphertexts using a six-round neural distinguisher. A key guess is returned if the resulting score for the partially decrypted ciphertext structure then exceeds another threshold c_2.

Ranking a Partial Decryption To combine scores returned for individual ciphertext pairs in a ciphertext structure into a score for the structure, we use Equation (3) as in the previously described 9-round attack.

Attack Parameters This basic idea can be turned into a practical key recovery attack on 11-round Speck. The initial difference $(0x211/0xa04)$ and the neutral bits set consisting of bits 14,15,20,21,22,23 of the cipher state work well, even though bits 14,15 and 23 are not totally neutral. Using $c_1 = 15, c_2 = 100$ one obtains an attack that succeeds on average within about 20 minutes of computing time on a machine equipped with a GTX 1080 Ti graphics card, or in about 12

hours on a single core of a modern CPU. In one hundred trials, a key guess was output after processing on average $2^{13.2}$ ciphertext pairs. Recovery of both true last subkeys was successful in 81 cases; the final subkey was correctly guessed in 99 cases. In the one remaining case, the second-to-last subkey was correct and the guess for the last subkey was incorrect in one bit.

Improved Attack This basic attack can be accelerated in various ways. Here, we focus on the following ideas:

1. The wrong key randomization hypothesis does not hold when only one round of trial decryption is performed, especially in a lightweight cipher. We use this to introduce an efficient key search policy using a generic optimization algorithm.
2. It is inefficient to spend the same amount of computation on every ciphertext structure. We use a generic method (an automatic exploitation versus exploration tradeoff based on upper confidence bounds) to focus our key search on the most promising ciphertext structures.

With these improvements, we can build an attack that recovers the final two subkeys of Speck32/64 reduced to 11 rounds with a success probability of about 50 percent from ciphertext corresponding to 12800 chosen plaintexts in about 8 minutes running in single-threaded mode on a single CPU core.[3]

Bayesian Optimization Bayesian optimization [33] is a method that is commonly used for the optimization of black box functions f that are expensive to evaluate. Examples are found in many domains; the tuning of hyperparameters of machine learning models is one common example. It uses prior knowledge about the function to be optimized to construct a probabilistic model of the function that is easy to optimize. Knowledge about the model parameters is adjusted to accomodate input from function evaluations $f(x_0), f(x_1), \ldots, f(x_n)$. An *acquisition function* is then used to decide which points of the function to query next in order to improve in the most effective way possible knowledge about the maximum.

In this work, we use Bayesian optimization to build an effective key search policy for reduced-round Speck. This key search policy drastically reduces the number of trial decryptions used by our basic attack, at the cost of a somewhat expensive optimization step. The basic idea of our key search policy is that the expected response of our distinguisher upon wrong-key decryption will depend on the bitwise difference between the trial key and the real key. This *wrong-key response profile* can be captured in a precomputation. Given some trial decryptions, the optimization step then tries to come up with a new set of key hypotheses to try. These new key hypotheses are chosen such that they maximize the probability of the observed distinguisher responses.

[3] Running the same code with different parameters, other attacks can be obtained. The code repository, for instance, contains parameters for a 12-round attack that is practical on a single PC (with the parameters used, average runtime is under an hour on a GeForce GTX 1080 Ti GPU and success rate is ≈ 40 percent).

Model Assumptions Let C_0, C_1 be a ciphertext pair and let k be the real subkey used in the final round of encryption. Let $\delta \in \mathbb{F}_2^{16}$ and let $k' = k \oplus \delta$ be a wrong key. Denote the response of our distinguisher D to decryption by the key k' by $R_{D,\delta}(C_0, C_1) := D(E_{k'}^{-1}(C_0), E_{k'}^{-1}(C_1))$. We can then view $R_{D,\delta}$ as a random variable depending on δ induced by the ciphertext pair distribution and compute its mean μ_δ and standard deviation σ_δ. If we average the distinguisher response over all elements of a ciphertext structure of size n as used in our attack, the average can be expected to approximately follow a normal distribution[4] with mean μ_δ and standard deviation $\sigma_\delta/\sqrt{n}$.

Wrong Key Randomization We calculated the wrong key response profile for our six- and seven round distinguishers for Speck32/64. To calculate the $r+1$-round wrong key response profile, we generated 3000 random keys and message input pairs P_0, P_1 for each δ and encrypted for $r+1$ rounds to obtain ciphertexts C_0, C_1. Denoting the final subkey of each encryption operation by k, we then performed single-round decryption to get $E_{k\oplus\delta}^{-1}(C_0), E_{k\oplus\delta}^{-1}(C_1)$ and had the resulting partially decrypted ciphertext pair rated by an r-round neural distinguisher. μ_δ and σ_δ were then calculated as empirical mean and standard deviation over these 3000 trials. The wrong key response profile for seven rounds is shown in Figure 3. A lot of non-random structure is evident. The shape of the curves for σ_δ and for six rounds is similar.

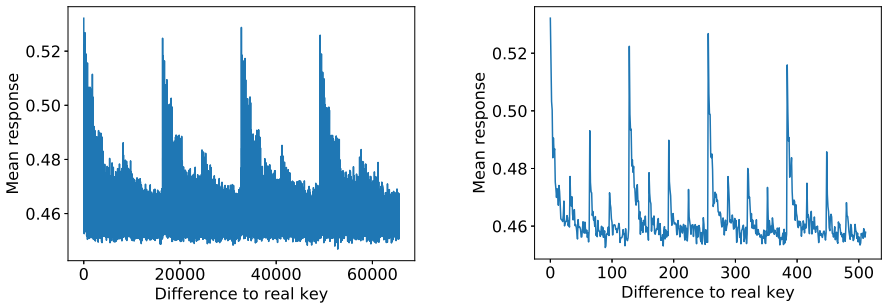

Fig. 3. Wrong key response profile (only μ_δ shown) for 8-round Speck32/64 and our 7-round neural distinguisher. For each difference δ between trial key and right key, 3000 ciphertext pairs with the input difference $0x0040/0000$ were encrypted for 8 rounds of Speck using randomly generated keys and then decrypted for one round using a final subkey at difference δ to the right key. Differences are shown on the x-axis, while mean response over the 3000 pairs tried is shown in the y-axis.

[4] Note that for our neural networks, this argument can be slightly strengthened if the final sigmoid activation is removed, since then distinguisher output on an individual ciphertext pair is just a linear combination of 64 somewhat independent intermediate network units.

Using the Wrong-Key Response Profile for Key Search Given our model assumptions and observations of the distinguisher response $r_0, r_1, \ldots, r_{n-1}$ for keys $k_0, k_1, \ldots, k_{n-1}$, we can view the r_i as values obtained from an n-dimensional normal distribution. The parameters of this normal distribution depend on the bitwise differences of the k_i to the real last subkey k, specifically on $\mu_{k \oplus k_i}$ and $\sigma_{k \oplus k_i}$. It is easy to see that the probability density at the observed values is maximised by minimizing the weighted euclidean distance $\sum_{i=0}^{n-1} (m_i - \mu_{k \oplus k_i})^2 / \sigma^2_{k \oplus k_i}$. Our algorithm first generates a set of random key candidates, then scores those keys by decrypting the ciphertext structure currently under study, then calculates the average distinguisher response on the tried keys, and finds a new set of key candidates that bring the precomputed wrong key response profile in line with the observed values as well as possible. This is iterated for a few cycles. Algorithm 4 sums up the algorithm.

Algorithm 4 BayesianKeySearch: efficiently find a list of plausible key candidates given a ciphertext structure satisfying the initial differential of our attack.

Require: Ciphertext structure $C = C_0, \ldots C_{m-1}$, neural distinguisher N, number of candidates to be generated n, number of iterations l.

$S := \{k_0, k_1, \ldots, k_{n-1}\} \leftarrow$ choose at random without replacement from the set of all subkey candidates.
$L \leftarrow \{\}$
for $j \in \{0, 1, \ldots, l-1\}$: **do**
    $P_{i,k} \leftarrow Decrypt(C_i, k)$ for all $i \in \{0, 1, \ldots, m-1\}$, $k \in S$.
    $v_{i,k} \leftarrow N(P_{i,k})$ for all i,k
    $w_{i,k} \leftarrow \log_2(v_{i,k}/(1 - v_{i,k}))$ for all $i \in \{0, \ldots, m-1\}, k \in S$
    $w_k \leftarrow \sum_{i=1}^{n} v_{i,k}$for all$k \in S$
    $L \leftarrow L || [(k, w_k)$ for $k \in S]$
    $m_k \leftarrow \sum_{i=0}^{n-1} v_{i,k}/n$ for $k \in \{k_0, \ldots, k_{n-1}\}$
    $\lambda_k \leftarrow \sum_{i=0}^{n-1} (m_{k_i} - \mu_{k_i \oplus k})^2 / \sigma^2_{k_i \oplus k}$ for $k \in \{0, 1, \ldots, 2^{16} - 1\}$:
    $S \leftarrow \text{argsort}_k(\lambda)[0 : n-1]$
end for
return L

All keys tried and their scores w_k on the current ciphertext structure are stored. Keys that obtain a score above a cutoff threshold c_1 are expanded by repeating the process for one further round, i.e. Algorithm 4 is used with a six-round neural distinguisher and its associated wrong key response profile. If one of the resulting key candidates scores above another threshold c_2, we determine that the search will be terminated, but the processing of the current search node is finished before the best pair of subkeys found for the last two rounds is returned.

Before we return a key, we perform a small verification search with hamming radius two around the two subkey candidates that are currently best. This removes remaining bit errors in the key guess. If the verification search yields an improvement, it is repeated with the new best key guess.

Given t ciphertext structures, our algorithm is first tried on each structure. If no solution is found, since Algorithm 4 is probabilistic, we continue for a preset number of iterations it before returning the highest-scoring pair of subkeys for the last two rounds. During these additional iterations, we have to actively decide which ciphertext structures we will spend our computational budget on. We treat this as a multi-armed bandit problem and solve it using a standard exploration-exploitation technique, namely Upper Confidence Bounds (UCB). The order of the ciphertext structures to be tested in this phase depends on the highest distinguisher score obtained in the last-key search for the structures so far and on the number of visits they have received in our search. Specifically, denote by $w^i_{\max}$ the highest distinguisher score obtained so far for the ith ciphertext structure, by n_i the number of previous iterations in which the ith ciphertext structure has been selected, and by j the number of the current iteration. We calculate a priority score

$$s_i := w^i_{\max} + \alpha \cdot \sqrt{\log_2(j)/n_i} \tag{5}$$

and pick the ciphertext structure with the highest priority score for further processing. The visit count and the best result for this ciphertext structure are updated after the iteration has finished. We set α to $\sqrt{n_c}$, where n_c is the number of ciphertext structures available.

Results In the trials subsequently described, we use 100 ciphertext structures of 64 chosen plaintext pair encryptions each, the cutoff parameters $c_1 = 5$, $c_2 = 10$, the UCB exploration term $\alpha = 10$, an iteration count for the Bayesian key search policy of $l = 5$ and candidate number $n = 32$, and an iteration budget for the main loop $\mathrm{it} = 500$. Given a hundred ciphertext structures, our implementation outputs a key guess in approximately eight minutes on average (measured average in 100 trials: 500.68 seconds) when running on a single thread of our machine with no graphics card usage. This key guess is not always correct, but if it is not, this is easily apparent from the scores returned. When a fast graphics card is used, performance of our proof of concept implementation is not limited by the speed of neural network evaluation, but by the key search policy. The key search policy tries with the settings mentioned only 160 keys when processing a ciphertext structure.

We count a key guess as successful if the last round key was guessed correctly and if the second round key is at hamming distance at most two of the real key. Under these conditions, the attack was successful in 521 out of 1000 trials; recovery of the first round key was successful in 521 cases and in all of these cases, the second round key guess was wrong for at most two bits within the most significant nibble. For comparison, the attack presented in [19] is expected to succeed with the same data complexity in about 55 percent of all trials. In the simple model where in case of failure we request ciphertext values for another 100 plaintext structures, we expect that this attack will on average use $2^{14.5}$ chosen plaintexts until success, slightly more than [19].

Computational Complexity We estimate that a highly optimized, fully SIMD-parallelized implementation of Speck32/64 could perform brute force key search on our system at a speed of about 2^{28} keys per second per core. Adjusting for the empirically measured success rate of our attack we expect to need about 1000 seconds on average to execute the key recovery algorithm on a single core of our system. This yields an estimated computational attack complexity of 2^{38} Speck encryptions until a solution is found. The additional effort needed for full-key recovery is negligible, since at that point a good ciphertext structure has been found and the same attack can be launched on that single ciphertext structure with distinguishers for less rounds of Speck.

5 The Real Differences Experiment

5.1 Summary

In this section, we design a cryptographic experiment in which the adversary has to do differential cryptanalysis in a setting where the random and the real difference distribution are the same. We show that our neural distinguishers are successful in this experiment and compare their efficiency to solving the same problem by key search in the case of five-round Speck.

5.2 Experiment

Motivation We have seen in the previous section that our best neural distinguishers are better at recognizing Speck32/64 reduced up to eight rounds than a distinguisher based on the full difference distribution table. We have also seen that the Markov model at least predicts its own distinguishing success rate fairly well and have seen some evidence that the neural distinguishers exploit features outside the difference distribution table, e.g. from the fact that our neural distinguishers break Proposition 1. This section looks at that topic in isolation.

To this end, we introduce a differential cryptographic distinguishing task in which perfect knowledge of the differential distribution of a primitive under study does not in itself allow the adversary to do better than random guessing.

Experimental Setup First, 10^6 samples were drawn from the real distribution for the D5, D6, D7 and D8 tasks. Then, half of these samples were randomized in the following way: for an output ciphertext pair $C = (C_0, C_1) \in \mathbb{F}_2^{2b}$ to be randomized, a blinding value $K \in \mathbb{F}_2^b$ was generated uniformly at random by reading from /dev/urandom. This value was bitwise-added to both ciphertexts to produce the randomized ciphertext $\tilde{C} = (C_0 \oplus K, C_1 \oplus K)$.

The resulting 10^6 samples of randomized or non-randomized ciphertext pairs were preprocessed as previously described and the results were passed to the appropriate pretrained neural network for classification as random or real. No further training took place.

Rationale The distribution of difference values is clearly the same in both the random and the real sample in this experiment. On the other hand, in the random sample any information about the ciphertext other than the difference between the two ciphertext blocks given is perfectly hidden, as the blinding makes the results of the random sampling uniformly distributed on the hyperplane given by each possible difference.

Adapting Key Search For reference, the key search based distinguisher on five rounds from section 3 was modified to work in the real differences setting. In this setting, an exact solution by counting the keys leading to a decryption with the desired input difference seems infeasible, as both random and real examples are expected to regularly have a very high number of solutions. We therefore calculate two approximations to N_{keys}.

First, we calculate $A_{\text{rand}} := 2^{64} \cdot DP(\Delta C)$, where $DP(\Delta C)$ is the differential probability of observing the output difference of the ciphertext pair C as given by the Markov model of Speck.

Second, denoting by $\mathcal{D}_{\text{mid}}$ the set of possible round 3 differences, by $P(\delta)$ the probability of observing the round 3 difference δ, and by $N_\delta(C)$ the number of solutions for the final two subkeys that decrypt our observed ciphertext pair C to the round 3 difference δ, we compute $A_{\text{real}} := 2^{32} \cdot \sum_{\delta \in \mathcal{D}_{\text{mid}}} P(\delta) \cdot N_\delta(C)$.

C is then predicted as real if $A_{\text{real}} > A_{\text{rand}}$ and as random otherwise.

Results Our best networks were found to solve the real differences task measurably better than random guessing without ever having explicitly been trained for it. Training on the real differences task was tried in the five-round case and expanded this advantage considerably. Predictably, however, key search yielded clearly superior distinguishing power. See Table 4 for details.

Table 4. Performance of neural distinguishers in the real differences experiment. For comparison, the performance of a key search based distinguisher and a version of the N5 network retrained to this task are also given. Test set size is 10^6 for the neural distinguishers and 10^4 for Search.

Nr	Distinguisher	Accuracy
5	N5	$0.707 \pm 9.10 \cdot 10^{-4}$
6	N6	$0.606 \pm 9.77 \cdot 10^{-4}$
7	N7	$0.551 \pm 9.95 \cdot 10^{-4}$
8	N8	$0.507 \pm 1.00 \cdot 10^{-3}$
5	Search	$0.810 \pm 7.84 \cdot 10^{-3}$
5	N5 retrained	$0.762 \pm 8.51 \cdot 10^{-4}$

These tests show that ciphertext pairs are not evenly distributed within their respective difference equivalence classes. Indeed, using neural distinguishers as a search tool it is easy to find examples of ciphertext pairs with relatively high-likelihood differences which have very little chance of appearing in the ciphertext

pair distribution of reduced Speck. One such example has already been discussed in section 4.4. For another, consider the output pair (0x58e0bc4, 0x85a4ff6c). It has the ciphertext difference 0x802a/f4a8, which is a high probability output difference for five round Speck given our input difference ($p \approx 2^{-15.3}$ according to the full Markov model, which matches empirical trials well here). However, the pair decrypts to our input difference with a much lower likelihood, around $2^{-35.3}$ according to our calculations.

Constraining the Additional Signal If we constrain the blinding values used in the real differences experiment to be of the form aa, where a is any 16-bit word, our distinguishers fail. This is consistent with the observation that the distinguishers do not exploit the key schedule, as addition of a blinding value of this form is equivalent to changing the final subkey used in encryption. It also suggests that the networks exploit a strictly more fine-grained partition of the output pairs into equivalence classes than the difference equivalence classes. We find that intra-class variance of distinguisher output is very low for these equivalence classes and that also the input representations generated by the penultimate network layer show tight clustering.

6 Conclusions

We have tested in this paper whether neural networks can be used to develop statistical tests that efficiently exploit differential properties of a symmetric primitive that has been weakened sufficiently by round reduction to allow for attacks to be carried out in a low data setting. In the setting considered, this works reasonably well: our distinguishers offer classification accuracy superior to the difference distribution table for the primitive in question and use less memory, even if inference speed is of course low compared to the simple memory lookup needed with a precomputed difference distribution table. We consider it interesting that this much knowledge about the differential distribution of round-reduced Speck can be extracted from a few million examples by black-box methods.

The time needed to train a network from the ground up to an accuracy level beyond the difference distribution table is on the order of minutes in our trials when a single fast graphics card is available. Our networks start training with no cryptographic knowledge beyond the word-structure of the cipher, making our approach fairly generic. The transfer learning capabilities shown in this paper demonstrate that finding good input differences from scratch is likewise possible using our networks with minimal input of prior cryptographic knowledge. Our distinguishers have various novel properties, most notably the ability to differentiate between ciphertext pairs within the same difference class.

In the context of this study, it certainly helped that Speck32/64 is a small blocksize, lightweight primitive. However, this is true both for the optimisation of conventional attacks and for the application of machine learning.

Given that the present work is an initial case study, we would not be surprised if our results could be improved. Various directions for further research suggest

themselves. For instance, it would be interesting if a reasonably generic way were found to give the model to be trained more prior knowledge about the cipher or to enable the researcher to more easily extract knowledge from a trained model.

Small improvements to network performance are also completely expected to be possible within the architecture and setting given by this paper.

It would be interesting to see the effect of giving the network cryptographic knowledge in the form of precomputed features. We did some tests along these lines, for instance by giving the prediction head a classification derived from the difference distribution table as an additional input, but this was only marginally helpful.

The use of Bayesian optimization and related methods for key search could be of more general interest whenever an attack exploits a statistical distinguisher with high evaluation cost. This could be a neural network, but for instance ordinary statistical distinguishers that need to be evaluated on very large sets of ciphertexts to be effective may also be examples.

We do not think that machine learning methods will supplant traditional cryptanalysis. However, we do think that our results show that neural networks can learn to do cryptanalysis at a level that is interesting for a cryptographer and that ML methods can be a useful addition to the cryptographic evaluators' tool box. We expect that similar to other general-purpose tools used in cryptography such as SAT solvers or Groebner basis methods, machine learning will not solve cryptography but usefully complement and support conventional dedicated methods of doing research on the security of symmetric constructions.

Acknowledgments The author wishes to thank the anonymous reviewers for their questions and comments, as they helped him to improve the present paper.

References

1. Martin Abadi and David G Andersen. Learning to protect communications with adversarial neural cryptography. *arXiv preprint arXiv:1610.06918*, 2016.
2. Farzaneh Abed, Eik List, Stefan Lucks, and Jakob Wenzel. Differential cryptanalysis of round-reduced Simon and Speck. In *International Workshop on Fast Software Encryption*, pages 525–545. Springer, 2014.
3. Martin R Albrecht and Gregor Leander. An all-in-one approach to differential cryptanalysis for small block ciphers. In *International Conference on Selected Areas in Cryptography*, pages 1–15. Springer, 2012.
4. Ralph Ankele and Stefan Kölbl. Mind the Gap – A Closer Look at the Security of Block Ciphers against Differential Cryptanalysis. In *Proceedings SAC 2018*, 2018.
5. Tomer Ashur and Daniël Bodden. Linear cryptanalysis of reduced-round SPECK. In *Proceedings of the 37th Symposium on Information Theory in the Benelux*. Werkgemeenschap voor Informatie-en Communicatietheorie, 2016.
6. Wasan Shaker Awad and El-Sayed El-Alfy. Computational intelligence in cryptology. *Improving Information Security Practices through Computational Intelligence*, pages 28–45, 2015.
7. Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

8. Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block Ciphers for the Internet of Things. *IACR Cryptology ePrint Archive*, 2015:585, 2015.
9. Ray Beaulieu, Stefan Treatman-Clark, Douglas Shors, Bryan Weeks, Jason Smith, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*, pages 1–6. IEEE, 2015.
10. Eli Biham and Rafi Chen. Near-collisions of SHA-0. In *Annual International Cryptology Conference*, pages 290–305. Springer, 2004.
11. Alex Biryukov, Vesselin Velichkov, and Yann Le Corre. Automatic search for the best trails in ARX: Application to block cipher Speck. In *International Conference on Fast Software Encryption*, pages 289–310. Springer, 2016.
12. Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: theory and practice. In *International Workshop on Fast Software Encryption*, pages 35–54. Springer, 2011.
13. Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 2722–2730. IEEE, 2015.
14. François Chollet et al. Keras. `https://keras.io`, 2015.
15. Jung-Wei Chou, Shou-De Lin, and Chen-Mou Cheng. On the effectiveness of using state-of-the-art machine learning techniques to launch cryptographic distinguishing attacks. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 105–110. ACM, 2012.
16. Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play go. In *International Conference on Machine Learning*, pages 1766–1774, 2015.
17. Moisés Danziger and Marco Aurélio Amaral Henriques. Improved cryptanalysis combining differential and artificial neural network schemes. In *Telecommunications Symposium (ITS), 2014 International*, pages 1–5. IEEE, 2014.
18. Flavio de Mello and José Xexéo. Identifying Encryption Algorithms in ECB and CBC Modes Using Computational Intelligence. *Journal of Universal Computer Science*, 24(1):25–42, 2018.
19. Itai Dinur. Improved differential cryptanalysis of round-reduced Speck. In *International Workshop on Selected Areas in Cryptography*, pages 147–164. Springer, 2014.
20. Aidan N. Gomez, Sicong Huang, Ivan Zhang, Bryan M. Li, Muhammad Osama, and Lukasz Kaiser. Unsupervised cipher cracking using discrete GANs. In *International Conference on Learning Representations*, 2018.
21. Sam Greydanus. Learning the enigma with recurrent neural networks. *arXiv preprint arXiv:1708.07576*, 2017.
22. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
23. Geoffry Hinton, Orio Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv preprint: arXiv 1503.02531*, 2015.
24. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
25. Alexander Klimov, Anton Mityagin, and Adi Shamir. Analysis of neural cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 288–298. Springer, 2002.

26. Linus Lagerhjelm. Extracting Information from Encrypted Data using Deep Neural Networks. Master's thesis, Umeå University, 2018.
27. Xuejia Lai, James L Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 17–38. Springer, 1991.
28. Elena Laskari, Gerasimos Meletiou, Yannis Stamatiou, and Michael Vrahatis. Cryptography and cryptanalysis through computational intelligence. In *Computational Intelligence in Information Assurance and Security*, pages 1–49. Springer, 2007.
29. Helger Lipmaa and Shiho Moriai. Efficient algorithms for computing differential properties of addition. In *International Workshop on Fast Software Encryption*, pages 336–350. Springer, 2001.
30. Yu Liu, Kai Fu, Wei Wang, Ling Sun, and Meiqin Wang. Linear cryptanalysis of reduced-round SPECK. *Information Processing Letters*, 116(3):259–266, 2016.
31. Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.
32. Kenneth G Paterson, Bertram Poettering, and Jacob CN Schuldt. Big bias hunting in amazonia: Large-scale computation and exploitation of RC4 biases. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 398–419. Springer, 2014.
33. Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. BOA: The Bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*, pages 525–532. Morgan Kaufmann Publishers Inc., 1999.
34. Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template attack vs bayes classifier. Technical report, Cryptology ePrint Archive, Report 2017/531, 2017, 2016.
35. Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 157–176. Springer, 2018.
36. Ronald L Rivest. Cryptography and machine learning. In *International Conference on the Theory and Application of Cryptology*, pages 427–439. Springer, 1991.
37. David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
38. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
39. David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
40. Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.