Improving Attacks on Speck32/64 using Deep Learning

Aron Gohr

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

Abstract. This paper presents a very practical key recovery attack on Speck32/64 reduced to 11 rounds based on a novel type of differential distinguisher using machine learning. These distinguishers exceed distinguishers based on the entire differential distribution table of Speck32/64 in accuracy, specificity and sensitivity. We show that they obtain significant gain from features of the output distribution that are invisible to the differential distribution table. The key recovery attack has been completely verified empirically and has an average runtime of approximately three minutes on a desktop computer with a fast graphics card or about 30 minutes on the same machine when not using the graphics card. This corresponds to roughly 41 bits of remaining security for 11-round Speck32/64, which is a substantial improvement over previous literature. The average data complexity of our attack is slightly lower than the best previous attack on the same number of rounds.
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 [BCB14, WSC+16] and autonomous driving [CSKX15] to playing various abstract board games at superhuman level [CS15, SHM+16, SHS+17]. In cryptography, practical work using machine learning techniques has mostly focused on side-channel analysis [MPP16, PHG16, PSK+18]. 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 [Riv91]. 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.

As far as regards cryptanalysis, few authors have conducted practical experiments to test the applicability of machine learning to cryptanalytic problems. Indeed, a straightforward, black-box application of machine learning is unpromising even for many tasks that are cryptanalytically trivial. For instance, it should usually be practically impossible to learn to simulate a cryptographically strong cipher if only given black box access to the cipher (although there are counter-examples to this, e.g. the one-time pad provides perfect secrecy but is easy to learn in this setting). The fact that a human cryptographer understands the design of the primitive they want to attack certainly gives a large advantage to handcrafted solutions.

However, this does not mean that machine learning cannot be used successfully to solve nontrivial cryptographic problems. The logical structure of a cryptographic problem

can be used at various steps of the development process of a machine learned solution, e.g. in the design of the initial model, by pretraining the model on related problems or by the choice of presentation and preprocessing of training data. The cryptographer can in addition use their understanding of the mechanism under study by designing a subproblem that can possibly be attacked by machine learning.

This paper therefore looks at the possibility of using machine learning as a tool for the rapid prototyping of parts of cryptanalytic attacks and uses for this the exploitation of differential properties of the Speck32/64 block cipher as a test case. This is mainly motivated by the fact that Speck is easy to implement and has a fairly large amount of published cryptanalysis. We use the smallest version of Speck primarily because for 32-bit Speck we can optimize classical differential distinguishers very effectively. This allows us to benchmark the performance of our machine learning models against very strong classical baselines.

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 traditional cryptanalytic techniques. These are based on computing and using the complete difference distribution induced by a given input difference for the effort of distinguishing Speck output from random data. 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 seven rounds past our chosen input difference.

Our neural distinguishers outperform these very strong baselines in terms of classifier accuracy, sensitivity and specificity. We further investigate this 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, reinforcing the observation that the neural distinguishers use features not represented in the differential distribution table.

However, the evaluation of a deep residual neural network is fairly costly compared to traditional distinguishers used in differential cryptanalysis, and the real test of a novel distinguisher is whether it can be used to improve key recovery attacks. In order to test cryptographic usefulness, we 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 $2^{13.6}$ chosen-plaintext queries at a computational complexity equivalent to about 2^{41} Speck encryptions; expected average wall time to recovery of the last two subkeys on a desktop computer with a single GTX 1080 Ti graphics card is about three minutes in our implementation. The closest comparison to this in the literature might be the attack on Speck32/64 reduced to 11 rounds presented in [Din14], which needs an expected 2^{14} chosen plaintexts to recover a Speck key with a computational effort of about 2^{46} reduced Speck evaluations. A summary may be found in Table 3.

While other authors have tried to use neural networks for cryptanalytic tasks (see e.g. [AEA15, DH14, Ala12, LMSV07, CLC12, dMX18] 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

Table 1: Summary of the main distinguishing attacks presented in this paper for the conventional setting of random or real distinguishing tasks. Accuracies given are best available empirical estimates on a test set containing an equal number of examples for both distributions.

ID	Type	Rounds	Accuracy	Reference
D5	Multiple differential real or random distinguisher	5	0.911	Section 3
D7-	Multiple differential real or random distinguisher using an empirically computed differential distribution table (sample size: 10^{11})	7	0.541	Section 3
NC5	Deep residual neural network solving the same task as D5	5	0.929	Section 4
NC7	Deep residual neural network solving the same task as D7	7	0.611	Section 4
Search	Perfect differential real or random distinguisher solving the D5 task based on optimized key search used as a final benchmark.	5	≈ 0.95	Section 3

Table 2: Summary of the main distinguishing attacks presented in this paper in the setting of the real differences experiment. In the real differences experiment, the adversary must distinguish execution of a primitive F based on seeing two output blocks with a known input difference from $F' = F \oplus C$, where C is a random value of appropriate bit size.

ID	Type	Rounds	Accuracy	Reference
Add	Perfect real differences distinguisher for addition modulo 2^n	-	0.864 (for $n = 16$)	Section 5
NAdd	Neural real differences distinguisher for addition modulo 2^n	-	0.865 (for $n = 16$)	Section 5
NC5 retrained	Same as for the D5 task	5	0.761	Section 5
Search_RD	Search modified to solve the five round real differences task for Speck32/64	5	0.81	Section 5

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.

The most important distinguishing attacks developed in this paper are summarised in Tables 1 and 2.

Structure of the paper In section 2, we give a short overview of the Speck family of block ciphers and discuss some basic cryptographic properties. Nothing in this section is new, but it is useful to explicitly state the mentioned properties to understand both some aspects and possible extensions of the manually developed distinguishers of section 3. In our machine learning experiments, input of prior knowledge into the learning system is restricted to knowledge of the word-oriented structure of the cipher, the notion that

Table 3: 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	[Din14]
Neural multiple differential	2^{41}	$2^{13.6}$ CP	This paper, section 4

bit-sliced functions might be important, and the use of a fixed input difference in sample generation.

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 [AL12]. These distinguishers are based on calculating for Speck reduced to 5 and 6 rounds the probability of observed differences based on the transition probabilities of all relevant differential paths. For seven rounds, we use empirically determined frequencies of differential states based on 10^{11} observations. This gives distinguishers with overall classification accuracy of 91.1 percent for five rounds, 75.5 percent for six rounds, and 54.1 percent for seven rounds, given only two blocks of ciphertext for an unknown plaintext with the chosen input difference. To illustrate the possible benefit of considering the full ciphertext pair distribution instead of the distribution of observed output differences, we also develop a distinguisher in the chosen-ciphertext setting that uses a simple property of Speck to this end as well as a five-round chosen-plaintext distinguisher based on key search. Data on our five- and seven round distinguishers is summarised in Table 1.

In section 4, we first revisit the distinguishing problems on 5,6, and 7 rounds by training deep residual neural networks to solve them. We describe in detail the learning pipeline, data preprocessing and model structure for our best model. Our best models show a distinguishing power that exceeds the all-in-one purely differential attacks outlined in section 3, giving overall classification accuracies of approximately 93, 79 and 61 percent for the five, six and seven round distinguishing tasks discussed previously; sensitivity and specificity of the neural classifiers also exceed baseline. We further show that the internal input representation of our neural networks can be used to rapidly develop distinguishers for round numbers and input differences different from the ones seen in training. For instance, a few examples of the output distribution induced by the input difference used in our main attacks for six rounds usually suffice to train a fairly strong distinguisher for six-round Speck if we already have a neural distinguisher trained to distinguish three-round Speck with a fixed random input difference (see Figure 2). The necessary retraining can be done in roughly a millisecond on our machine. We use this 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. We perform retraining in the five-round example and find that this significantly improves performance, although a significant gap remains compared to a manually designed benchmark that employs partial key search. The predictive performance of the key search based benchmark comes, however, at a large price

the form of required computing power. We also discuss a concrete example of a ciphertext pair that is misclassified by traditional differential distinguishers.

As a toy example, we analyze the differential properties of addition modulo 2^n with regards to xor-addition in the real differences experiment, deriving an optimal distinguisher with running time linear in the size of the input data. We then train a recurrent neural network using the long short term memory framework [HS97] to solve the same task for 16-bit integers, finding close to optimal performance of the neural model on a test dataset. We test how the neural model scales with increasing input bitsize and find that scaling is without any retraining likewise similar to the optimal distinguisher, proving that the result is not substantially due to overfitting on the training dataset with relatively low bitsize examples.

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

1.3 Related work

1.3.1 Related cryptographic work

Speck has since its publication [BTCS+15] received a fair amount of analysis, see e.g. [BSS+15] 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 [ALLW14]. They constructed efficient differential trails for round-reduced versions of all members of the Speck family of ciphers and showed how to use these trails for key recovery. For Speck32/64, the round 3 difference of their 9-round trail is used in the present work as the input difference required by our differential distinguishers.

In [Din14], Dinur improved the analysis given in [ALLW14] 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 their 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 their 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 [BVLC16] a framework for the automatic search for optimal single differential trails of Speck and further improved on the differential trails found in [ALLW14]. 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 [AB16, LFW+16].

Multiple differential cryptanalysis as an extension of truncated differential cryptanalysis was first studied by Blondeau and Gerard in [BG11]. 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 [AL12] 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.

1.3.2 Prior work on machine learning and data-driven techniques in cryptography

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.

Laskari, Meletiou, Stamatiou and Vrahatis [LMSV07] 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. They also reported some evidence of successful learning (but not of any advantage over traditional techniques) in neural networks trained to solve various problems from classical public key cryptography with extremely small parameter sizes.

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 [CLC12], who perform some experiments along these lines and give a review of the literature.

Rivest in [Riv91] 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 [Gre17].

Purely data driven attacks have been used with good success e.g. against RC4 by Paterson, Poettering and Schuldt [PPS14]. 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

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

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 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. [AK18]).

As introduced by Lai, Massey and Murphy [LMM91], 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 differential distribution. The most general form of differential attack that has been formally discussed in the literature are *multiple differential attacks* [BG11], 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*. When it is useful to distinguish between the two, we will 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 [BTCS+15] for the NSA with the aim of building a cipher efficient in software implementations in IoT devices [BSS+15]. 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 [BTCS+15], 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:

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

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 (22 rounds for 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 round function 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 [BTCS+15] for reference.

2.3 Basic cryptographic properties

The only non-linear operation in the Speck round function is the modular addition used in the computation of L_{i+1}, and the only key-dependent operation is the subkey addition. The subkey addition happens after the modular addition, i.e. the cipher operation is completely predictable until this first subkey addition. This leads to the following observation:

Observation 1. *Differential and linear transitions through a single round of Speck do not depend on the key but only on the plaintext value supplied as input. In a chosen plaintext setting, the adversary can freely choose the difference at the output of round one. In a known plaintext setting, any intermediate ciphertext differences at the round 1 output can be computed from the plaintext.*

We note that a conceptually simple way to achieve e.g. the prediction of round-1 ciphertext differences is just to take the two plain texts P_0, P_1 in question and compute $F(0, P_0) \oplus F(0, P_1)$.

While Observation 1 is not new and quite basic, we mention it explicitly because it is used in the manually designed distinguisher with ID CC2. This is the only handcrafted distinguisher in this paper that can claim a substantial advantage in both speed and accuracy over neural distinguishers solving the same problem.

In [Din14], a method is given to complete any differential trail for two-round Speck when only input and output differences are given. We summarize their method as follows:

Observation 2. *Let $\Delta_{in}, \Delta_{out} \in \mathbb{F}_2^n$, with $\Delta_{in} = (L_{in}, R_{in})$ and $\Delta_{out} = (L_{out}, R_{out})$. If the differential transition $\Delta_{in} \to \Delta_{out}$ is possible for two-round Speck, then the round 1 difference for the corresponding differential trail must have been $(X \oplus (R_{in} \ll \beta), X)$ with $X := (R_{out} \oplus L_{out}) \gg \beta$.*

Note that combining this with Observation 1, it is straightforward to derive all intermediate differences even for three-round Speck if the full plaintext pair and the ciphertext differences are given.

3 Multiple differential attacks on Speck32/64

3.1 Pure differential distinguishers

Setting Multiple differential attacks [BG11] 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 In this section, all of our differential distinguishers start with the round 3 difference of the differential trail given in Table 7 of [ALLW14], i.e. with the difference $\Delta = 0x0040/0000$. We then use algorithm 2 in [LM01] to compute the differential behaviour of the nonlinear component of Speck32/64, which is simply modular addition modulo 2^{16}. A meet-in-the-middle approach is used to calculate arbitrary entries of the difference distribution table for five and six round Speck. By performing a short tree search, we obtain a table T of all possible state differences

Table 4: Accuracy of differential distinguishers processing two blocks of ciphertext for Speck reduced to Nr rounds. Error bars reported are for 2σ confidence intervals. The precision of reported figures was truncated to three decimal places. Each test set contained exactly the same number of examples drawn from the real and the random distribution respectively. Values reported are overall classification accuracy as well as accuracy on samples taken from the real and the random distribution respectively.

Nr	ID	Accuracy	True Positive Rate	True Negative Rate	Size test set
5	D5	$0.913 \pm 1.26 \cdot 10^{-3}$	$0.879 \pm 2.06 \cdot 10^{-3}$	$0.947 \pm 1.42 \cdot 10^{-3}$	$2 \cdot 10^5$
6	D6	$0.755 \pm 9.97 \cdot 10^{-3}$	$0.673 \pm 1.33 \cdot 10^{-2}$	$0.837 \pm 1.04 \cdot 10^{-2}$	10^4
7	D7-	$0.541 \pm 2.23 \cdot 10^{-3}$	$0.433 \pm 3.33 \cdot 10^{-3}$	$0.649 \pm 3.02 \cdot 10^{-3}$	$2 \cdot 10^5$

and their associated transition probabilities three rounds past our chosen input difference. Given a ciphertext pair to classify, we then enumerate all differential trails leading to one of the intermediate states stored in T. When computing the probabilities of individual trails in the transition graph, we assume that at each round, an intermediate ciphertext pair with difference δ that is to be further processed has uniformly and independently from previous round transitions been drawn from all ciphertext pairs with difference δ. This means in particular that when computing the probability of a differential trail, the transition probabilities for subsequent rounds can be multiplied.

This calculation works only if the cipher under study does not deviate too strongly from the Markov property. We have therefore tested the validity of this model by also testing the performance of distinguishers against five and six rounds of Speck based on empirically approximating the difference distribution table by observing a sample of 10^{11} differential transitions. The distinguishers so obtained had lower accuracy than the Markov model, indicating that model error is relatively benign.

For 7-round Speck, the calculation of exact differential probabilities using the full Markov model for Speck was not practical on the machine used in this study. We instead opted to estimate the transition probabilities for 7-round Speck empirically. To this end, we first produced a size 200000 sample of real and random ciphertext pairs to classify. We then attached counters initialized to zero to all the ciphertext differences in our sample, generated a larger sample of 100 billion ciphertext pairs, and incremented each counter whenever the corresponding difference appeared in this bigger sample.

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. Since we have good approximations of the probability of the observed ciphertext difference in the real distribution, standard Bayesian inference can be used to obtain a classification that exploits differential non-uniformity perfectly up to model and double precision arithmetic rounding error.

Results The performance of the distinguishers so obtained was measured by empirical testing on a random test dataset generated for each of the cases. The test dataset consisted of equal numbers of random and real examples. During example generation, each example was computed with a fresh randomly generated key and plaintext pair. The results are summarised in Table 4.

Table 5: Accuracy of chosen-ciphertext differential distinguishers processing two blocks of ciphertext for Speck reduced to 5 rounds. Error bars reported are for 2σ confidence intervals. The precision of reported figures was truncated to three decimal places.

ID	Accuracy	True Positive Rate	True Negative Rate	Size test set
CC1	$0.873 \pm 1.49 \cdot 10^{-3}$	$0.878 \pm 2.07 \cdot 10^{-3}$	$0.869 \pm 2.13 \cdot 10^{-3}$	$2 \cdot 10^5$
CC2	$0.995 \pm 9.97 \cdot 10^{-4}$	$0.997 \pm 9.94 \cdot 10^{-4}$	$0.993 \pm 9.96 \cdot 10^{-4}$	$2 \cdot 10^4$

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 case study: chosen ciphertext distinguishers As Observation 1 suggests that in the case of chosen-ciphertext attacks one can easily access the differential distribution of Speck one round prior to the observed output, we have as a first test performed a simple experiment on five-round Speck decryption. In this experiment, we evaluated the efficiency of two differential distinguishers for Speck32/64. Both of these are testing decryption of random ciphertext pairs with difference $0x8000/8000$ for five rounds. The first of these distinguishers (which we will call *CC1*) simply uses the differential distribution calculated by the same methods as used for five-round Speck in the forward direction. The other (henceforward called *CC2*) first re-encrypts the plaintext pair received for one round under the all-zero subkey and then compares the difference so obtained to the differential distribution *four* rounds after the constant input difference. The results of this experiment are summarised in Table 5.

A perfect differential distinguisher for Speck32/64 reduced to five rounds We have also 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 given a uniformly random choice of input plaintexts and keys and a block cipher E of block size b and key size k is given by

$$P(C|\text{real}) = \frac{\text{Card}(\{(P,K) \in \{0,1\}^{b+k} \text{ with } E_K(P) = C_0, E_K(P \oplus \Delta) = C_1\})}{2^{b+k}},$$

and the numerator 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 [Din14] 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 (due to the first differential transition in the trail used being deterministic) 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. While this distinguisher is fairly slow in our implementation, we tested it on a small sample of 320 examples and found that this achieves a classification accuracy of about 95 percent. We also tried a simplified version of this distinguisher which uses key search only on the final two rounds and estimates the number of solutions for the two remaining subkeys using the differential table, i.e. by using the approximation $N_{\text{keys}} \approx p_{\text{transition}} \cdot \text{Card}(\mathcal{K})$, where $\mathcal{K}$ denotes the set of subkeys for the rounds in question and where $p_{\text{transition}}$ is the probability of the differential transition of the sub-trail under study; this achieved an overall accuracy of 94.3 percent on a sample of size 10000.

4 Neural distinguishers for 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. Many other choices of architecture yield results that are also superior to the distinguishers presented in the previous section.

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 [HZRS16] for image recognition and have been successful since in a number of other applications, for instance strategic board games [SSS+17, SHS+17]. The results reported in this paper were obtained with a residual tower of depth 10 followed by a densely connected prediction head with two hidden layers and one output cell.

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 /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. It takes a few seconds to generate a data set of size 10^7 in our implementation.

Training Training was run for 200 epochs on the dataset of size 10^7. The datasets were processed in batches of size 2048, except for the NCC distinguisher, which used a batch size of 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 [KB14] with default parameters in Keras [C+15]. A cyclic learnrate schedule was used, setting the learnrate 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.

Training cost A single epoch of training according to this 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 differential distribution table can be obtained in less than fifteen minutes after starting the training cycle in all the examples considered.

4.4 Results

Test set accuracy We summarize our findings in Table 6. The neural distinguishers achieve higher accuracies than the purely differential baselines discussed in the previous section on all tasks. The accuracy of the CC2 and 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.

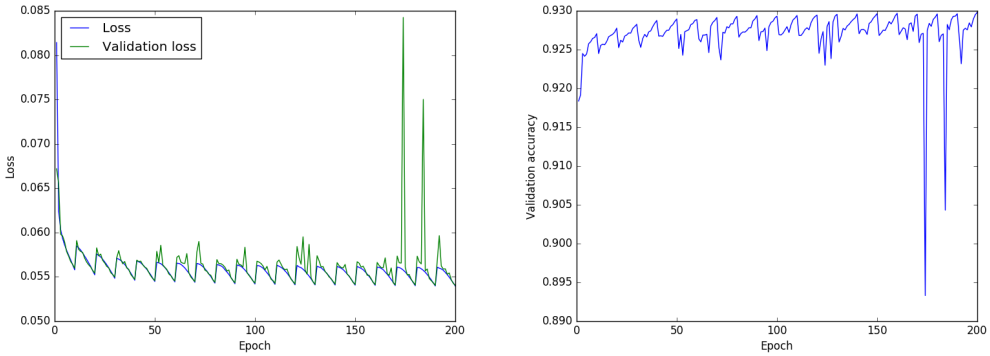

Figure 1: Training a neural network to distinguish Speck output for the input difference $\Delta = 0x0040/0$ from random data. *(left)* Training and validation loss by epoch. *(right)* Validation accuracy. *(both)* The weak validation performance at Epoch 174 does not represent overfitting. Performance on the training set for this network is in fact equally weak. This is not apparent from the data on training loss since it represents an average over the loss observed in all training steps.

Table 6: Empirically measured test set accuracies for neural distinguishers on 5, 6, and 7 rounds of Speck32/64 as well as for a 5-round neural chosen-ciphertext distinguisher. The reported accuracies were obtained on a randomly generated test set of size 10^6 containing approximately 500000 positive and negative examples each.

Network	Accuracy	True Positive Rate	True Negative Rate
NC5	$0.929 \pm 5.13 \cdot 10^{-4}$	$0.904 \pm 8.33 \cdot 10^{-4}$	$0.954 \pm 5.91 \cdot 10^{-4}$
NC6	$0.788 \pm 8.17 \cdot 10^{-4}$	$0.724 \cdot 1.26 \cdot 10^{-3}$	$0.853 \pm 1.00 \cdot 10^{-3}$
NC7	$0.613 \pm 8.5 \cdot 10^{-4}$	$0.551 \pm 1.41 \cdot 10^{-3}$	$0.674 \pm 1.33 \cdot 10^{-3}$
NCC	$0.971 \pm 3.35 \cdot 10^{-4}$	$0.975 \pm 4.41 \cdot 10^{-4}$	$0.967 \pm 5.05 \cdot 10^{-4}$

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. Inference speed can be increased to about one million ciphertext pairs per second at a small cost in accuracy by using a simple form of knowledge distillation [HVD15]. We generate a size 9 million training data set as described before, then predict each example using a pre-trained ten-block network, and use the predictions as training targets for a one-block student network. Training was performed for 30 epochs, with the learnrate being lowered from 0.001 to 0.0001 after epoch 20. For the 7-round case, the distinguisher so obtained was more or less equivalent in quality to the original ten-block distinguisher; for 6 rounds, a very small loss in accuracy was observed on a size one million test set. For five rounds, a more substantial but still mild loss in prediction quality was found on the test set.

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. For each ciphertext pair c in the test set, brute force key search was performed on the last round key and the resulting partially-decrypted ciphertext pairs c_k were fed through a six-round neural distinguisher. The scores z_k returned by the neural distinguisher were heuristically treated as estimates for the probability that c_k was sampled from the real distribution in our real-vs-random experiment. Setting $z'_k := z_k/(1 - z_k)$ we obtained the corresponding likelihood ratios. These were averaged for all keys to obtain an average score z_{av} and a score z for the ciphertext pair c was obtained by applying the transformation $z := z_{\text{av}}/(1 + z_{\text{av}})$. c was classified as real if $z > 0.5$ and as random otherwise. This procedure did indeed slightly improve prediction accuracy: ground truth was matched in 62.7 percent of the test sample. See Algorithm 1 for a summary of the algorithm used.

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

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 for 30 epochs with a single learnrate drop from 0.001 to 0.0001 at epoch 20. The solution so trained was tested on a separately generated test set and was found to be our strongest neural distinguisher for seven rounds of Speck, with a global accuracy of 61.6 percent. Compared to the original ten-block distinguisher, this gain is mostly due to increased specificity, at the cost of losing some sensitivity, i.e. a higher rejection rate for samples from the real distribution.

Disagreement with differential 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 differential 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 differential 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. This shows that the neural predictor provides a considerably clearer signal than the differential distribution table on this subset of cases.

However, exploitation of information that can be obtained reliably from the differential 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/0x5dbc, 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 differential distribution table assigns a 98 percent probability to this output pair being from the real distribution. In an empirical test of this prediction, we found 44 good pairs for this differential transition in 10^{10} trials, which is substantially lower than predicted, but still within an order of magnitude of the expected value. However, the neural distinguisher returns an almost zero likelihood of (c_0, c_1) being produced by the real distribution. This turns out to be correct both by comparison to the ground truth as given by the labelling of the test sample and more importantly by comparison with the result of key search. The latter shows that this output pair is in fact impossible to produce by the real distribution.

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 $L : \mathbb{R}^{64} \rightarrow \mathbb{R}$ 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** $L \circ N'$

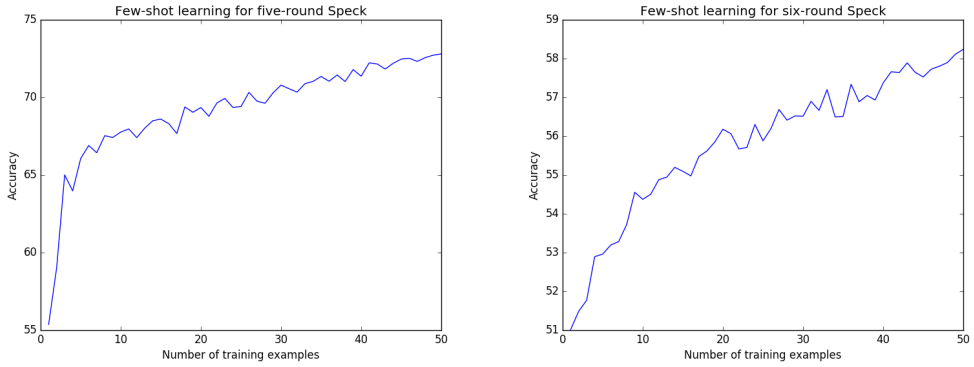

Figure 2: Few-shot learning on the D5 and D6 tasks using a pre-trained classifier to preprocess the input data. Algorithm 2 was used with a fixed auxiliary network trained to distinguish Speck32/64 reduced to three rounds with a random fixed input difference. The number of training examples supplied 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.

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.

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}

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

Remark on data augmentation In the context of few-shot-learning, it is natural to ask whether symmetries of the problem could be used to improve the process. For instance, it is a priori known that the ciphertext pairs C_0, C_1 and C_1, C_0 have the same probability of appearing in the real or the random distribution. Also, it is easy to calculate from C_0, C_1 all ciphertext pairs that would result if the last subkey used in encryption where exchanged for a different subkey. However, in our testing these data augmentation methods seemed not to be helpful. The reason for this is that the auxiliary network used in transfer learning apparently most of the time knows about these symmetries already and maps inputs randomized in this way to nearby internal representations.

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 NC7 and NC6 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 [Din14]. 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 [Din14], 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^{41} Speck evaluations if it is executed on a CPU. Its data complexity is slightly lower than that of the attack in [Din14].

Basic attack idea 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 [BC04] 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 Our neural distinguishers return for each ciphertext pair C processed a score z between 0 and 1. This score can heuristically be treated as the probability that in a real-vs-random distinguishing experiment, this ciphertext pair has been drawn from the real distribution. The ratio between the probabilities p_{real} and p_{rand} that the ciphertext pair C will be generated by the real and the random distribution respectively can then be calculated by the relation $p_{\text{real}}/p_{\text{rand}} = z/(1-z)$. To calculate a score for a structure of ciphertexts that should either all be from the random distribution or all from the real distribution, we take dual logarithms of these likelihood ratios and sum up the results to obtain a score for the structure.

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.

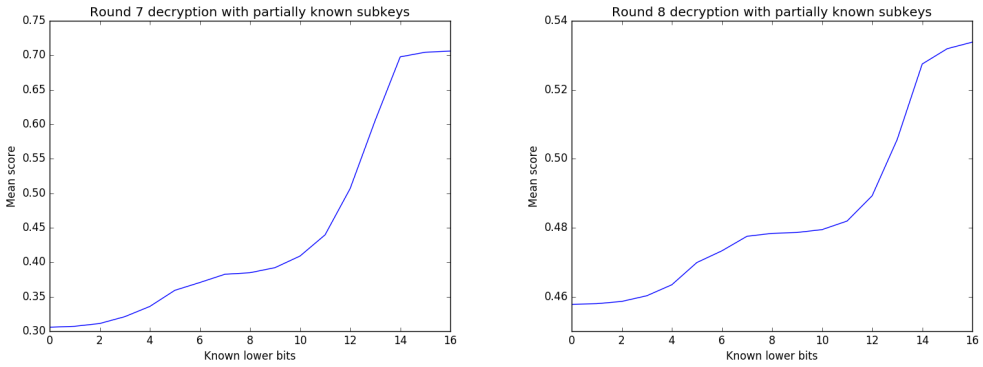

Figure 3: Decryption with a partially known subkey. 10^7 ciphertext pairs with the input difference $0x0040/0000$ were encrypted for 7 (left) or 8 (right) rounds of Speck using randomly generated keys. Then, the most significant bits of the final subkey were zeroized and the messages partially decrypted using this changed subkey were processed by the appropriate neural distinguisher. The average scores returned by the neural network in each case are shown.

Refinements This basic attack can be accelerated in various ways. We have implemented the following enhancements:

1. Our neural distinguishers can recognize decryptions by partially correct subkeys, as can be seen from Figure 3. We use this by not performing brute force search on the last round-key. Instead, we implement a guess-and-determine attack where an initial set of key candidates is tried and ranked, the most promising keys are kept, expanded by one bit, and the process is repeated until all bits of the remaining key candidates are fixed.

2. Likewise, in a preliminary step we can rank the ciphertext structures by performing a partial last-round key search in the described manner using only half the ciphertexts for each ciphertext structure. This reduces data usage and saves time since we do not need to query the encryption oracle further for ciphertext structures that have already been found to likely not have passed the initial differential based on looking at the first half of their members.

3. With these changes, keys are often found that are at hamming distance one or two of the real subkeys. To fix this, a short verification search is done around hamming weight two of the final best key guess, using the six-round distinguisher to rank the key guesses in this verification search.

A conceptually simple way to use the key division property exhibited in Figure 3 is to use Algorithm 3 to optimize the combined score returned by a ciphertext structure as a function of the last subkey. We have tested this and it does in fact work reasonably well, but it is not the most effective attack we tried.

The basic building block of the improved key recovery attack is Algorithm 4. It takes as input a ciphertext structure (i.e. a set of ciphertexts that are expected to have passed the initial differential together) as well as a set of final subkey candidates to be considered. It outputs a reduced set of key guesses and scores for the structure.

In our implementation, we slightly deviate from Algorithm 4 by additionally varying the most significant bits of each candidate key when decrypting the ciphertext structure. Conceptually, this means that we consider some bits of a key candidate k as undefined and use different settings of these bits when decrypting different ciphertexts within one

Algorithm 4 RemoveKeyCandidates: Narrow down a set of key candidates given a ciphertext structure

Require: Ciphertext structure $C = C_0, \dots C_n$, set of candidate subkeys $S = \{sk_0, \dots, sk_m\}$, selectivity parameter q, neural distinguisher N

1: $P_{i,k} \leftarrow Decrypt(C_i, k)$ for all $k \in S$.
2: $v_{i,k} \leftarrow N(P_{i,k})$ for all i,k
3: $v_{i,k} \leftarrow \log_2(v_{i,k}/(1 - v_{i,k}))$ for all i,k
4: $v_k \leftarrow \sum_{i=1}^{n} v_{i,k}$ for all k
5: $v, S' \leftarrow$ sort S and the v_k together in descending order by v_k
6: $c \leftarrow \lfloor q \cdot m \rfloor$
7: **return** $S'[0, \dots, c], v[0, \dots, c]$

structure. However, the scores obtained from all of these keys will be fused into the score of key k as described. This slightly improves key space coverage by ensuring that *some* ciphertexts within the structure to be tested get decrypted by the correct key or an almost correct key if the bits of the key that are considered fixed are correct.

A high-level view of the entire attack can be given as follows: for each ciphertext structure, we first create a set of key guesses that covers all values of the 10 least significant bits of the last subkey. We apply RemoveKeyCandidates to these with $q = 1/2$, extend the remaining subkeys by one bit, and repeat until all subkey bits have been fixed and we are left with a set of 512 candidates for the last subkey and corresponding network scores. We throw away all subkey candidates with a score less than a cutoff parameter c_1. For each of the remaining candidates, a trial decryption is performed. We then create a set of candidates for the second subkey that covers all values of the least significant 9 bits and repeatedly apply RemoveKeyCandidates and extend the key candidates as described before. We use the more aggressive pruning parameter $q = 1/3$ this time, as the six-round distinguisher yields a very strong signal.

If this process finds a pair of subkey candidates that score higher than a second cutoff c_2 after the application of the second round key, we decide to output a key guess but finish processing all key candidates for the current ciphertext structure before doing so. In the whole process, we keep track of the current best candidates for the two round keys. Whenever a new best pair of round keys is found, the list of candidates for the last round key is extended by adding all keys within hamming distance two of the best guess.

Before a key guess is output, a short verification search is done with hamming radius two around each of the two round keys to be recovered. The verification search is repeated if it yields a score improvement. The pair of keys with the best score returned by the six round network is finally given as the best guess for the key values.

Pre-sorting of ciphertext structures applies the same procedure as generating the candidate lists for the last round key but uses more aggressive pruning ($q = 1/3$) and only half of the data in each ciphertext structure. The key candidates generated are thrown away and the ciphertext structures are sorted in descending order by best score returned.

Results In the trials subsequently described, we use the cutoff parameters $c_1 = 5$ and $c_2 = 10$. Given a hundred ciphertext structures, our implementation outputs a key guess in approximately two minutes on average (measured average in 100 trials: 132 seconds) on a computer with a GTX 1080 Ti graphics card. This key guess is not always correct, but if it is not, this is easily apparent from the scores returned. On a single core of one CPU of the same system, we expect an average runtime under the same conditions of roughly two hours. Efficiency slightly increases when more data is available, since pre-sorting then has a larger impact. With five hundred ciphertext structures of data and aggressive selectivity settings, an average time to solution of about a minute can be reached.

One hundred ciphertext structures correspond to 12800 chosen plaintexts. In one hundred trials, our attack queried on average $2^{12.9}$ chosen plaintexts under these conditions. 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 one of the real key. Under these conditions, the attack was successful in 63 out of 100 trials; recovery of the first round key was successful in 66 cases and in all of these cases, the second round key guess was within hamming weight two of the real key. For comparison, the attack presented in [Din14] is expected to succeed with the same data complexity in about 55 percent of all trials. Our success rate and data usage when using 100 ciphertext structures correspond to an expected data usage of $2^{13.6}$ chosen plaintexts until success in the simple model where in case of failure we request ciphertext values for another 100 plaintext structures.

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 three hours on average to execute the key recovery algorithm on a single core of our system. This yields an estimated computational attack complexity of 2^{41} Speck encryptions until a solution is found. The additional effort needed for full-key recovery can be trivially upper bounded by $2^{36.1}$ Speck encryptions assuming that the last round key has been guessed correctly and that the second-to-last round key guess is correct up to a single bit flip.

On a fast GPU, we expect that brute force key search can be massively sped up. The GTX 1080 Ti, for instance, has 3584 CUDA cores and runs at a clock rate of 1582 Mhz. Speck encryption needs ten basic arithmetic operations per round including the key schedule. Since Speck32/64 operates on 16-bit words, it seems reasonable to assume that data level parallelism may be exploited to perform four basic 16-bit integer arithmetic operations per CUDA core per cycle. This suggests that the GTX 1080 Ti could be used to perform $2^{37.5}$ Speck encryptions per second. Three minutes of wall time per attack until solution then suggest an attack complexity of 2^{45} Speck encryptions.

It is not unusual that an attack that is more naturally run on a GPU than on CPU has an apparently worse complexity if it is compared to brute force key search on the same platform [SBK+17]. We believe that both figures likely understate the computational complexity advantage of our attack compared to other attacks in the literature when both are run fully utilizing the capabilities of modern CPUs or GPUs. For instance, it does not seem obvious that the attack in [Din14] is able to effectively use data-level parallelism or a GPU-type execution model, while this is obvious both for brute force key search and for our attack.

Tradeoffs There are various tradeoffs available in this attack between data usage, selectivity of the search, and search cutoff criteria. These have a real impact on the complexity of the attack. For instance, with slightly higher data usage, we can construct an attack that has roughly half the runtime claimed here. Fundamentally improving the search, however, is likely to have higher impact than parameter tuning. We leave this to further research.

5 The real differences experiment

5.1 Speck32/64

Motivation We have seen in the previous section that our best neural distinguishers are better at recognizing Speck32/64 reduced to five and six rounds than a distinguisher based

on the full differential table. In principle, there are two main reasons that could explain this observation:

1. The Markov model used to calculate differential probabilities for Speck is wrong and the neural distinguisher exploits the available differential information better because it is unaffected by the error of our mathematical model of the differential transitions of Speck.

2. The neural distinguishers exploit features of the ciphertext pair distribution that are not visible to purely differential distinguishers.

In this section, we give evidence supporting the second hypothesis. 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. We also briefly report on an experiment to train a neural network to solve a toy problem in this setting in a fairly general way.

Experimental setup First, 10^6 samples were drawn from the real distribution for the D5, D6 and D7 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 five, six or seven rounds for classification as random or real. No further training took place.

For reference, the *Search* 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, we calculate

$$A_{\text{real}} := 2^{32} \cdot \sum_{\delta \in \mathcal{D}_{\text{mid}}} P(\delta) \cdot N_\delta(C),$$

where $\mathcal{D}_{\text{mid}}$ is the set of possible round-3 differences, $P(\delta)$ denotes for one such difference the probability of observing this round 3 difference given the fixed input difference of our trail, and $N_\delta(C)$ denotes the number of solutions for the final two subkeys that decrypt our observed ciphertext pair C to the round 3 difference δ.

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

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.

Table 7: Performance of the NC5, NC6 and NC7 networks in the real differences experiment. For comparison, the performance of a key search based distinguisher and a version of the NC5 network retrained to this task are also given. Test set size is 10^6 for the neural distinguishers and 10^4 for Search.

Nr	Network ID	Accuracy
5	NC5	$0.707 \pm 9.10 \cdot 10^{-4}$
6	NC6	$0.606 \pm 9.77 \cdot 10^{-4}$
7	NC7	$0.548 \pm 9.95 \cdot 10^{-4}$
5	Search	$0.810 \pm 3.92 \cdot 10^{-3}$
5	NC5 retrained	$0.762 \pm 8.51 \cdot 10^{-4}$

Results Our best five, six and seven round 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 7 for details.

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 pairs (0x58e0bc4, 0x85a4ff6c) and (0xaa12b7f2, 0x2a38435a). Both have the difference 0x802a/f4a8, and the transition 0x0040/0000 $\rightarrow$ 0x802a/f4a8 is of fairly high likelihood for five round Speck (roughly $2^{-15.3}$ according to the full Markov model, which matches empirical trials well here). However, the former ciphertext pair decrypts to the starting difference of this transition with a much lower likelihood, around 2^{-36} according to our calculations.

5.2 Real differences of addition

Motivation Even in the case of five-round Speck32/64, it seems difficult to achieve perfect exploitation of the nonuniformity of the ciphertext pair distribution present in the real differences experiment. We therefore briefly looked at a simpler situation, namely the real differences of addition. In the sequel, we give a brief account of our results.

Setting We assume that we are given a differential equation of addition, i.e. a problem of the form

$$(x \oplus a) \boxplus (y \oplus b) = (x \boxplus y) \oplus c,$$

where $a, b, c \in \mathbb{F}_2^n$ are known. We assume further that the adversary is additionally given $z := x \boxplus y$ or a blinded version $z := (x \boxplus y) \oplus K$, where K is randomly chosen for each sample. The adversary has to distinguish whether they have been given a blinded (random) or not blinded (real) problem. For problem generation, a, b, x, y, K are chosen uniformly at random from $\mathbb{F}_2^n$. z and c are set to the values $z := x \boxplus y$ and $c := (x \boxplus y) \oplus ((x \oplus a) \boxplus (y \oplus b))$ so that the resulting problem always has at least one solution.

Exact solution The probability p_{rand} of observing the tuple (a, b, c, z) in the blinded distribution can be readily calculated by the methods given in the literature on differential equations of addition, see e.g. [LM01, SG13]. To calculate also the probability p_{real}, we designed a finite-state machine $S(a, b, c, z)$ with $64n + 1$ states that traverses a proposed solution $x, y, z, x' = (x \oplus a), y' = (y \oplus b), z' = (z \oplus c)$ in a bit-sliced manner from least to

Table 8: Number of correctly classified examples for real differences of addition test sets of size 10^5, with problem bit sizes 16 and 128. *Add* here denotes classification by counting all solutions to the given problem, while *NAdd* is a neural distinguisher based on a recurrent neural network. The same test set was used for both solutions. The handcrafted solution is superior outside experimental error margins for the 128 bit problem size.

Problem size	Add	NAdd
16	86438	86582
128	99751	99273

most significant bit and which checks whether the proposed solution is in fact a solution, entering a special non-accepting state otherwise. We then calculate the number of accepting end-states by multiplying the transition matrices for each time step. To process an n-bit string, this requires the construction of at most 64 transition matrices and $n-1$ multiplications of matrices of size 64×64. p_{real} can then be obtained from the number of solutions multiplying by 2^{-6n}. The given data is finally classified as real if and only if $p_{\text{real}} > p_{\text{rand}}$.

Neural network We also trained a neural network to solve the same problem. The input data is presented as a sequence of 4-bit vectors representing (a, b, c, z) in the natural way. The neural network consists of a recurrent layer with 16 LSTM units with rectifier output activations followed by a dense output layer with a single unit using a sigmoid activation. Training was performed in batches of size 1000 against binary crossentropy loss on a data set of 10^6 samples of length 16 bit for 100 epochs using the Adam optimizer and the same cyclic learnrate schedule as described previously for the Speck distinguisher networks. The last 10^5 samples of the training set were withheld for validation testing. The best network by validation loss was retained for further study. Finally, the solution was evaluated on two test sets of size 100000 corresponding to problem sizes (bit length of the DEA problem) $n = 16$ and $n = 128$ respectively. No regularization was used, as overfitting did not seem to be a problem.

Results The overall performance of the machine-learned and the manually designed solution were found to be comparable, with the latter slightly superior for the 128 bit problem size. Scaling with bit size was similar for both solutions. Quantitative results of the final evaluation can be seen from Table 8.

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 differential 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 differential 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 differential 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 also in the specific cases considered 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 differential table as an additional input, but this was only marginally helpful.

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.

References

[AB16] 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.

[AEA15] Wasan Shaker Awad and ES El-Alfy. Computational intelligence in cryptology. *Improving Information Security Practices through Computational Intelligence*, pages 28–45, 2015.

[AK18] 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.

[AL12] 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.

[Ala12] Mohammed M Alani. Neuro-cryptanalysis of des and triple-des. In *International Conference on Neural Information Processing*, pages 637–646. Springer, 2012.

[ALLW14] 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.

[BC04] Eli Biham and Rafi Chen. Near-collisions of sha-0. In *Annual International Cryptology Conference*, pages 290–305. Springer, 2004.

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[BG11] Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: theory and practice. In *International Workshop on Fast Software Encryption*, pages 35–54. Springer, 2011.

[BSS+15] 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.

[BTCS+15] 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.

[BVLC16] 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.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[CLC12] 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.

[CS15] Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play go. In *International Conference on Machine Learning*, pages 1766–1774, 2015.

[CSKX15] 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.

[DH14] 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.

[Din14] Itai Dinur. Improved differential cryptanalysis of round-reduced speck. In *International Workshop on Selected Areas in Cryptography*, pages 147–164. Springer, 2014.

[dMX18] Flavio L de Mello and José AM Xexéo. Identifying encryption algorithms in ecb and cbc modes using computational intelligence. *Journal of Universal Computer Science*, 24(1):25–42, 2018.

[Gre17] Sam Greydanus. Learning the enigma with recurrent neural networks. *arXiv preprint arXiv:1708.07576*, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HVD15] Geoffry Hinton, Orio Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *arXiv preprint: arXiv 1503.02531*, 2015.

[HZRS16] 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.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[LFW+16] 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.

[LM01] 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.

[LMM91] 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.

[LMSV07] Elena C Laskari, Gerasimos C Meletiou, Yannis C Stamatiou, and Michael N Vrahatis. Cryptography and cryptanalysis through computational intelligence. In *Computational Intelligence in Information Assurance and Security*, pages 1–49. Springer, 2007.

[MPP16] 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.

[PHG16] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Template attack vs bayes classifier. Technical report, Cryptology ePrint Archive, Report 2017/531, 2017, 2016.

[PPS14] 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.

[PSK+18] 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.

[Riv91] Ronald L Rivest. Cryptography and machine learning. In *International Conference on the Theory and Application of Cryptology*, pages 427–439. Springer, 1991.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full sha-1. In *Annual International Cryptology Conference*, pages 570–596. Springer, 2017.

[SG13] Ernst Schulte-Geers. On ccz-equivalence of addition mod 2^n. *Designs, codes and cryptography*, 66(1-3):111–127, 2013.

[SHM+16] 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.

[SHS+17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[SSS+17] 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.

[WSC+16] 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.