New Results on Machine Learning Based Distinguishers

Anubhab Baksi[1], Jakub Breier[2], Vishnu Asutosh Dasu[3], Xiaolu Hou[4], Hyunji Kim[5], and Hwajeong Seo[5]

[1] Nanyang Technological University, Singapore; anubhab001@e.ntu.edu.sg
[2] Silicon Austria Labs, Austria; jakub.breier@gmail.com
[3] Pennsylvania State University, Pennsylvania, USA; vishnu98dasu@gmail.com
[4] Slovak University of Technology in Bratislava, Slovakia; xiaolu.hou@stuba.sk
[5] Hansung University, Seoul, South Korea; khj1594012@gmail.com, hwajeong84@gmail.com

Abstract. Machine Learning (ML) is almost ubiquitously used in multiple disciplines nowadays. Recently, we have seen its usage in the realm of differential distinguishers for symmetric key ciphers. In this work, we explore the possibility of a number of ciphers with respect to various ML-based distinguishers.
We show new distinguishers on the unkeyed and round reduced version of SPECK-32, SPECK-128, ASCON, SIMECK-32, SIMECK-64 and SKINNY-128. We explore multiple avenues in the process. In summary, we use neural network as well as support vector machine in various settings (such as varying the activation function), apart from experimenting with a number of input difference tuples. Among other results, we show a distinguisher of 8-round SPECK-32 that works with practical data complexity (most of the experiments take a few hours on a personal computer).

Keywords: speck · ascon · simeck · skinny · distinguisher · machine learning · differential

1 Introduction

Machine learning (ML) is becoming ubiquitous in multiple research areas in computer science. Naturally, there have been a number of attempts to use of ML in cryptography, particularly fitting it to work with the well-known differential attack model. In fact, ML tools typically have native support for classification problem, which is similar to the distinguisher model where one attempts to classify the `CIPHER` from `RANDOM`. Particularly after Gohr's work on `SPECK-32` [23], the proper application of ML seems to be growing quite fast, as many research works including (but not limited to) [3,11,15,16,18,19,22,26,28,30] have taken interest in this.

In this work, we humbly attempt to extend the ML-assisted differential attack model. As the starting point, we adopt the differential distinguishing model presented in [7, Section 3.1]. We apply the concept of [7] to new ciphers, namely `ASCON` [21], `SPECK` [12], `SKINNY` [13], and `SIMECK` [31]. While `SPECK-32` has been the major, if not the only, focus of the previous works (a trend initiated/popularised by [23]); rest ciphers have never been analyzed with respect to ML-assisted attacks, to the best of our knowledge.

We carry out experiment with the two major Neural Network (NN) library, PyTorch and TensorFlow/Keras. Further, we explore the applicability of the Support Vector Machine (SVM), thus supplementing the NN which is the only ML tool used in the existing literature up to this point. More details on ML are deferred till Section 2.2.

1.1 Contributions

We argue the traditional analysis of the differential distinguisher (that does not involve ML tools), in all likelihood, has been underestimating the attacker's true power, who is free to use ML tools. Unlike some of the recent works (most notably, by Gohr [23]), where it is assumed the attacker is an expert in machine learning (thus is capable of designing a special purposed ML architecture), here we assume the other way around. We show, how the attacker is able to achieve the task of distinguishing cipher by using very simple ML tools – the parameters of which are decided arbitrarily. Even with that, we easily beat the non-ML based analysis, and yield same (if not better) result compared to a specialized ML architecture.

Our results, which are detailed in Section 5, can be summarized as follows:

- In Section 5.1, we present distinguishers on up to 8-round SPECK-32 and 7-round SPECK-128, using MLPs. We experiment with various options for the choice of the input differences (contrasting, e.g., Gohr's work [23]) where only one such option is considered.
- In Section 5.2, we present results on 3-round ASCON. These are obtained by using a linear-kernel SVM.
- In Section 5.3, we show results on 9-round SIMECK-32 and 14-round SIMECK-64, obtained using MLPs.
- In Section 5.4, we present distinguishers on 5-round SKINNY-128 reduced to 7 rounds, using SVMs (linear, RBF and polynomial kernels).

Note that our data generation method is similar to that of Gohr's [23], i.e., un-keyed permutation. In our case, t (> 1) input differences are used to create a t-class classification problem; whereas one input difference is used together to create a 2-class classification problem in [23]. The way the un-keyed permutations are considered, it is inherently considered that the full round keys are XORed at each round. Also, our analysis seems to question SKINNY's security claim made by the designers [14].

1.2 Novelty and Advancement of State-of-the-art

Reflection on ML-assisted Results on SPECK-32.

Number of Rounds. To the best of our knowledge, Gohr (CRYPTO'19 in [23]) reports the maximum number of rounds of SPECK-32 attacked by ML-based distinguisher as 8. To the best of our finding, the follow-up works fail to extend to increase the number of rounds from 8 [3, 11, 16]. We achieve the same number of rounds (Section 5.1), with simpler models and with lower data complexity (thus requiring much less time).

Simplicity of ML Model. Our model for distinguishger is adopted from [7, Model 1 in Section 3.1]. Thus, the number of neurons at the input layer is same as the state size of the cipher. This contrasts with the model used in [3, 11, 16, 23], where the number of neurons is double at the input layer. Further, we need less number of epochs (≤ 20), whereas Gohr's model requires much more (such as, 200) epochs. Thus, in some sense, our NN model is simpler. Apart from that we only use MLP for its simplicity, this is not intrinsic; thus other NN models can be used instead. For instance, the Convolutional Neural Network (CNN), which seems to the exclusive choice [3, 11, 16, 23], can also be used. More relevant discussion can be found in Section 3.3.

Level of Significance. As only 2-class classification is for the most part in this work, any case with training/testing accuracy of > 0.5 can be potentially taken as a distinguisher. In this work, we only consider those cases with training and testing accuracy both matching and > 0.51.

Since the inner working of a machine learning is typically poorly understood, it might happen that this minute deviation (i.e., less than 0.01) is caused due to some artefact of the tool (cf. the performance of Tensorflow/Keras and PyTorch discussed in Section 5.1), rather than being a true indicator of deviation from randomness. Until this minute deviation is confirmed otherwise (such as, some other method that does not involve ML), there is an off-chance that it may not hold up in the future (say, with an updated version of the same ML tool sometime in the future). Thus, we keep 0.01 as the threshold for detection.

We have noticed certain distinguishers achieve (marginally) accuracy of > 0.5 for higher rounds in some experiments with 9-round SPECK-32. This hints that it may be possibly that the distinguishers follow through more rounds than reported in this work. We do not immediately claim any confirmation about 9-round distinguisher (since the gap of accuracy from the RANDOM case is very similar), though it is an interesting case to study.

Apart from 2-class classification, we also use 3-class and 32-class classification, where we want to distinguish accuracy of 0.33333 and 0.03125, respectively. Here the threshold for distinguisher detection is kept at 0.01 and 0.001, respectively.

Practicality. All our results are practical and take in the ballpark of a couple of hours (except for the SVM which seems to take longer, though it may be possible to reduce run time by tweaking some parameters) to perform on a modern computer (without high-performance computing compatible hardware). Compared to the previous works (e.g., [23]), ours probably takes the least amount of time.

It can be further mentioned that we do not assume any more power to the attacker, Eve, than allowed in the classical differential distinguisher model. The only new ability the attacker has comes from how she analyses the information collected.

New Methods and Ciphers. We experiment with PyTorch and TensorFlow/Keras. As far as we are concerned, there has not been any attempt to study the impact of the choice of the NN library. While it is true that in a typical application these tools perform almost identically (where the accuracy is near-perfect), it may not be the case for the current situation where a meager 0.51 accuracy is considered a success. As a matter of fact, it seems that PyTorch outperforms TensorFlow/Keras; as the former can distinguisher up to 7-round `SPECK-32` (Table 3a) but the latter only works up to 6-round (Table 3b); despite using the same parameters, training/testing data and default options (though further experimentation is needed).

We employ SVM to study its impact on ML-based differential distinguishers. One major comment in [16] is about interpreting ML-based distinguishers by using equivalent representation that do not involve ML-specific terminology. In this regard, (linear kernel) SVM is a natural choice, since it gives an interpretation which can be readily interpreted. In particular, the linear kernel SVM gives a linear expression.

To top it all off, we show ML-assisted distinguishing results on some ciphers probably for the first time.

Low Data Complexity. All of our experiments (with same parameters) have been conducted multiple times independently. Therefore, even though the data complexity per experiment is relatively low, the possibility that the results are statistical outliers can be excluded.

Second Order Differential. Since the Model 1 from [7] naturally supports multiple differences, it is possible to realize higher order differential (see Table 5d for second order differential analysis on `SPECK-32`). To the best of our knowledge, this is the first time this is used in the literature.

2 Background

2.1 Motivation

In the classical differential distinguisher, the attacker, Eve chooses an input difference δ and XORs it the to the input of the state of the (possibly round reduced) `CIPHER`. Then `CIPHER` is run multiple times with randomly chosen inputs. The attacker finds the output differences for each run. Eve is also able to deduce a pre-calculated output difference Δ (which is a constant) with a certain probability at which the (δ, Δ) pair appears. When this probability is significantly more than what would be expected if (possibly round reduced) `CIPHER` is substituted by a random source, then the attacker would be successful in distinguishing (possibly reduced roud version of `CIPHER`) from `RANDOM`.

The modelling of probability distribution for $\delta \rightsquigarrow \Delta$ is done through various methods, such as the wide trail strategy [2, Chapter 1.4] or some tool [4, 27, 29] in the classical differential distinguisher.

One may note that, the attacker discards all the output differences which do not match Δ in the classical setting. This happens due to the very nature of the classical distinguisher. However, the assumption that the attacker will necessarily do this, possibly acts as a hindsight, since this may underestimate the attacker's capability.

Instead of discarding any output difference, we feed all to a suitable ML model. However, for this purpose, we need at least 2 input differences. Therefore, we consider the general case with t distinct input differences which are denoted as $\delta_0, \delta_1, \ldots, \delta_{t-1}$. When the accuracy of the ML model exceeds what is to be expected for `RANDOM`, this acts as a differential distinguisher. Thus, at its core an ML-assisted differential distinguisher model works by distinguishing between (possibly round-reduced) `CIPHER` from `RANDOM`; by formulating the challenger–adversary game to a suitable classification problem [7,8,9], for which native support is available. It is sometimes possible to reduce the complexity of the differential distinguisher drastically, even to the cube root of what is required for the classical case [7].

One point to note here is that we use the testing data for validation. This is generally not recommended in typical ML applications, due to the problem of overfitting. However, this is not a problem in our case, as there is only one test case (i.e., the testing data which is either generated from `RANDOM` or from `CIPHER`).

Another interesting question that may come to one's mind is about the usage of a differential model; since on the surface it appears that a classification problem can be formulated by setting, Class 0: `CIPHER` and Class 1: `RANDOM`; and watch out for accuracy > 0.5. However, the caveat is that this model (generally) returns accuracy of 0.5, even if a constant datum is used in case of `CIPHER`.

2.2 Machine Learning Basics

Multi Layer Perceptron (MLP). An MLP [24] is a supervised learning algorithm which is a type of a feed-forward NN (also called, Artificial Neural Network, which is abbreviated as ANN). An MLP consists of three or more layers of neurons (which is the basic unit of computation in a neural network). The first and the last layers are called the *input layer*, and the *output layer*, respectively, while all the middle layers are called the *hidden layers*. One characteristic of an MLP is that each neuron in a layer is connected to every neuron in the subsequent layer. Based on a rule, known as activation (where non-linear functions can be used), each neuron may fire with a different intensity. The back-propagation algorithm is used for training of feed-forward neural networks with the usage of gradient descent optimization method to update the weights of the neuron connections between each layer.

Support Vector Machine (SVM). SVMs [20] are supervised learning algorithms which are predominantly used for classification problems with two classes. An SVM constructs a set hyper-planes to separate the classes. The points from the two classes which are closest to the hyper-plane are known as support vectors. The distance between the hyper-plane and the support vectors are called margins. In order to find the hyper-plane that best divides the classes, an SVM tries to maximize the margin. Thus an SVM can be thought as an optimization problem. The classes need to be linearly separable to construct the optimal hyper-plane. If the classes are not linearly separable, the original space is mapped to a higher dimensional space, where separation of the classes is possible with a linear boundary. The data in each class are then defined in terms of a *kernel* function, which the SVM uses to compute the optimal hyper-plane. The usage of SVM in cryptography is not new, one may refer to [25].

3 Machine Learning Based Distinguisher

3.1 Basic Idea and Overall Description

As already mentioned, our model is adopted from that of [7, Section 3.1] (or [5, Chapter 6.4.1]). Here, Eve chooses t (≥ 2) distinct input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$ and creates t differentials. In the process, she converts the problem of distinguisher to the problem of classification, which can be efficiently tackled by ML tools. More specifically, she assumes the output differences corresponding to the input difference δ_i belong to class i, for $i = 0, 1, \ldots, t-1$.

As for the actual attack procedure, we assume the following set-up. The `ORACLE` tosses an unbiased coin, and chooses either `RANDOM` (a random source; which can be emulated, for example, with

/dev/random[1]) or CIPHER, depending on the outcome of the coin toss. Which output between RANDOM and CIPHER is chosen is kept secret from the attacker, and she has to find it out with probability significantly $> \frac{1}{2}$. For that purpose, she can query the ORACLE with inputs of her choice as many times she wants (but it has to be significantly less than that of the exhaustive search) and the ORACLE will return the output from either RANDOM or CIPHER.

In our context, she first builds the ML model during the training (offline) phase with sufficient training data. This is possible as she knows the specification of the CIPHER. Essentially, she chooses a random input P, computes the corresponding output C ($=$ CIPHER(P)); then for each δ_i, she computes the output differences ($C \oplus C_i$ where $C_i =$ CIPHER$(P \oplus \delta_i)$); and finally labels the output differences as belonging from class i. If the accuracy for training is $> \frac{1}{t}$ (measurement of the training accuracy is possible as she knows which output difference belongs from which class), she proceeds to the testing (online) phase.

In the online phase, she chooses random inputs P and queries it to ORACLE. Then she queries with $P \oplus \delta_i$ for $i = 0(1)t-1$, and computes the output differences corresponding to each input difference. However, it is to be noted that she is not able to measure the testing accuracy, as it is not known which output difference belongs to which class. To overcome this issue, we propose to use the ordering of the input differences. Therefore, she queries in the sequence: $P \oplus \delta_0, P \oplus \delta_1, \ldots, P \oplus \delta_{t-1}$. In doing so, she can now expect which output difference should belong to which class (i.e., the output difference ORACLE$(P) \oplus$ ORACLE$(P \oplus \delta_i)$ should be classified as belonging to class i). This way, she is able to measure the accuracy during testing. If ORACLE $=$ CIPHER, then the testing accuracy should match that of the training phase, which is $> \frac{1}{t}$. Otherwise, i.e., if ORACLE $=$ RANDOM, then the ML model would arbitrarily predict the classes for the output differences, hence the testing accuracy would be $\frac{1}{t}$. This constitutes the distinguisher.

Algorithm 1: Differential distinguisher with machine learning

1: **procedure** OFFLINE PHASE (Training)
2: $TD \leftarrow (\cdot)$ ▷ Training data
3: Choose random P
4: $C \leftarrow$ CIPHER(P)
5: **for** $i = 0; i \leq t-1; i \leftarrow i+1$ **do**
6: $P_i \leftarrow P \oplus \delta_i$
7: $C_i \leftarrow$ CIPHER(P_i)
8: Append TD with $(i, C_i \oplus C)$
▷ $C_i \oplus C$ is from class i
9: Repeat from Step 3 if required
10: Train ML model with TD
11: ML training reports accuracy a
12: **if** $a > \frac{1}{t}$ **then**
13: Proceed to Online phase
14: **else** ▷ $a = \frac{1}{t}$
15: Abort

1: **procedure** ONLINE PHASE (Testing)
2: $TD' \leftarrow (\cdot)$ ▷ Testing data
3: Choose random P
4: $C \leftarrow$ ORACLE(P)
5: **for** $i = 0; i \leq t-1; i \leftarrow i+1$ **do**
6: $P_i \leftarrow P \oplus \delta_i$
7: $C_i \leftarrow$ ORACLE(P_i)
8: Append TD' with $C_i \oplus C$
9: Test ML model with TD' to get $\mathcal{C}$
▷ $\mathcal{C}$ is sequence of classes by ML
10: $a' =$ probability that $\mathcal{C}$ matches $(0, 1, \ldots, t-1)$
11: **if** $a' = a > \frac{1}{t}$ **then**
12: ORACLE $=$ CIPHER
13: **else** ▷ $a' = \frac{1}{t}$
14: ORACLE $=$ RANDOM
15: Repeat from Step 3 if required

3.2 Training and Testing the Model

With the algorithmic description given in Algorithm 1, the basic work-flow is described here (also adopted from [7, Section 3.1]):

Training (Offline).

1. Select t (≥ 2) non-zero input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$.

[1] https://man7.org/linux/man-pages/man7/random.7.html

2. For each input difference δ_i, generate (an arbitrary number of) input pairs $(P, P_i = P \oplus \delta_i)$. Run the (unkeyed) permutation the input pairs to get the output pairs: $C \leftarrow \texttt{CIPHER}(P)$, $C_i \leftarrow \texttt{CIPHER}(P_i)$ for all i. Then XOR the outputs within a pair to generate the output difference $(C_i \oplus C)$. The output difference together with its label i (i.e., this sample belongs from class i) form a training sample.
3. Check if the training accuracy is $> \frac{1}{t}$. Otherwise (i.e., if accuracy $= \frac{1}{t}$), the procedure is aborted.

Testing (Online).

1. Generate the input pairs in the same way as training. In other words, randomly generate an input P. With the same input differences chosen during training $\delta_0, \delta_1, \ldots, \delta_{t-1}$; generate new inputs $P_i = P \oplus \delta_i$ for all $i = 0(1)t-1$.
2. Collect the outputs C and C_i's by querying `ORACLE` with input P and P_i's in order, for all $i = 0(1)t-1$.
3. Generate the testing data as $C \oplus C_i$ for all i and in order.
4. Get the predicted classes from the trained model with the testing data.
5. Find the accuracy of class prediction. In other words, tally the classes returned by the trained ML with the sequence: $(0, 1, \ldots, t-1, 0, 1, \ldots, t-1, \ldots, 0, 1, \ldots, t-1)$, and find the probability that both match.
6. (a) If `ORACLE` = `CIPHER`, the ML would predict the class for $C \oplus C_i$ as i wtih the same probability as training. Therefore in this case, the accuracy for class prediction (in Step 5) would be same (or, close to) the accuracy observed during training, i.e., $> \frac{1}{t}$.
   (b) If `ORACLE` = `RANDOM`, the ML would arbitrarily predict the classes. Therefore the accuracy for predicting classes by the trained ML (in Step 5) would be equal to (or, close to) $\frac{1}{t}$.

3.3 Comparison of Machine Learning Models

At this place, it is perhaps worth noting the differences with our ML model with the previous ones, most notably with Gohr's [23] (other works like [3,11,16] use some variation of that model).

The following points can be noted:

1. In Gohr's model, the input layer neuron size is doubled.
2. It seems that Gohr's model intrinsically requires CNN (some justification on CNN is given in [23, Section 4.2]) at least for the time being; which is relaxed in our case, any ML tool that supports classification (including an SVM) can be used.
3. Gohr does not use dropout layers, while we do. Instead he uses L2 regularization for the dense and convolution layers.
4. Gohr's model requires high number of epochs (like, 200), whereas we use much less (not more than 20). Our model takes considerably less time.
5. Gohr uses sigmoid activation (as the problem is always about binary classification) in the last layer while we use softmax (as we need support for multiple classes).
6. Gohr uses mean squared error (MSE) as the loss function while we use cross-entropy.
7. Gohr trains 8-round `SPECK-32` classifier using a *transfer learning* approach, as the regular (directly observed) distinguisher stops working after 7 rounds[2]. In contrast, our model can directly observe up to 8-round of `SPECK-32` with training/testing accuracy > 0.51 without any transfer learning.
8. Compared to Gohr, we need one-third entropy (for 2-class classification).
9. Due to the way the classification problem is formulated by Gohr, the cipher query complexity is double of the data complexity. In our case, the number of queries to the cipher is same as the total data complexity[3].

[2] While the efficacy of this approach is clear, we remark that transfer learning is typically used where there is a dearth of data (the reasoning for choosing the transfer learning approach is not clear given the context).

[3] To avoid any possible ambiguity, we count the data complexity as the total amount data (not the amount of data per class) in our results.

10. In Gohr's model, it can be argued that "RANDOM" is a misnomer, since it is actually a pair of ciphertexts which are obtained by encrypting two random plaintexts.

Note that the choices (such as, loss function, batch size, number of epochs) made in our architecture are mostly arbitrary, since we want to emulate an attacker who has merely a basic understanding of ML. For this reason, we consider the most basic neural network, MLP (apart from SVM). In any case, we would like to emphasize that, all these choices (including the choice of MLP) are no way linked to the basic ML-based distinguisher model — any design option can be considered in conjunction with/instead of our design choices. More specialized architecture that follow the same basic model can be expected to give better accuracy for a given round of a cipher than this work, and more importantly it can be expected to cover more round than this work.

Having said all that, one may note that, there is no inherent incompatibility from the ML model used here (adopted from [7, Section 3.1] with that of Gohr's (one can be converted to-and-from another if needed). For reference, in Gohr's model, one class corresponds to the ciphertext pair coming from input difference δ, and the other class corresponds to ciphertext pair coming from random plaintext pair.

4 Cipher Description in Brief

4.1 SPECK

SPECK [12] is a lightweight block cipher family, based on Feistel structure, designed by the National Security Agency (NSA) in 2013. There are 10 variants of SPECK, of which only two with state size of 32 and 128 bits are considered here. That one with state size of 32-bits runs for 22 rounds in the full version; and the other one runs for 32, 33, or 34 rounds depending on the key size. The round function of SPECK divides the input value into l and r, and rotation, modular addition, and xor are performed as follows: $l_i = (\mathrm{ROR}_7(l_i) \boxplus r_i) \oplus k_i$ and $r_i = \mathrm{ROL}_3(r_i) \oplus l_i$.

4.2 ASCON

ASCON [21] is a well-known lightweight authenticated encryption with associated data (AEAD). ASCON uses a 320-bit permutation, that runs for 12 rounds. It consists of Addition of Round Constants, Nonlinear Substitution Layer, and Linear Diffusion Layer, and operates by dividing it into 5 64-bit words $(x_0, x_1, x_2, x_3, x_4)$. The round constants are XORed of byte-1 of x_2 during Addition of Round Constants. In Nonlinear Substitution Layer, 5-bit SBox is applied. In Linear Diffusion Layer, rotation operation is applied to each word as follows: $\Sigma(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28), \Sigma(x_1) = x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39), \Sigma(x_2) = x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 6), \Sigma(x_3) = x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17), \Sigma(x_4) = x_4 \oplus (x_4 \ggg 7) \oplus (x_4 \ggg 41)$.

4.3 SIMECK

The lightweight block cipher family SIMECK [31] allows three (state size/key size) variants: 32/64 (32 rounds), 48/96 (36 rounds) and 64/128 (44 rounds). The round function and the key schedule are based on the Feistel architecture. Round function of SIMECK is similar to SIMON. Before the round function, the input plaintext is divided by l_0 (plaintext to be encrypted) and r_0 (plaintext to be encrypted). The round function(i^{th}) is as follows: $R_{k_i}(l_i, r_i) = (r_i \oplus f(l_i) \oplus k_i, l_i)$. The number of rotations of SIMECK round function are (0, 5, 1), and ROL_i means rotation left operation (i^{th} bit). The following f is used in the round function R_{k_i} update: $f(x) = x \wedge \mathrm{ROL}_5(x) \oplus \mathrm{ROL}_1(x)$.

4.4 SKINNY

SKINNY is a tweakable block cipher family which is introduced in CRYPTO 2016 targeting lightweight application scenario [13]. It supports 64-bit and 128-bit block sizes. The internal state is composed of a 4×4 array of cells according to the block size (each cell consists of a 4-bit cell in the case of a 64-bit block, and an 8-bit cell in the case of a 128-bit block size).

5 Results on Round-Reduced Ciphers

Set-ups

The following tools are used:

- For experiment on SPECK (Section 5.1, except those with the fixed input difference 28000010), ASCON (Section 5.2) and SIMECK (Section 5.3); our platform consists of 16× Intel Xeon E7-8880 CPUs, and 1× Nvidia Tesla-P100 16GB GPU accelerator (CUDA-10.2); and runs Ubuntu-18.04 (shared among multiple users); with Python-3.6.9 and Numpy-1.16.4.
- For experiment on SPECK (Section 5.1 with the fixed input difference of 28000010), and SKINNY (Section 5.4); our platform consists of an Apple M1 Pro 16GB with 10-core CPU, 16-core GPU, and 16-Neural Engine; with Python-3.8.9, Numpy-1.23.1 and Pytorch 1.12.0.
- We use TensorFlow-2.1.6[4] back-end with Keras-2.1.6[5] API, and PyTorch-0.4.1[6]. Among the ML models, only MLP is used throughout, with Adam as the optimizer.

Implementation of SPECK-32 and SPECK-128 unkeyed permutations are taken from a publicly available repository[7]. For the rest ciphers, the implementation provided by its designers' are used. Unless otherwise mentioned, by time, we indicate the total time (data generation + training + testing time).

5.1 SPECK

Arbitrary/Ad-hoc Input Differences. The results in this part are obtained from an MLP with TensorFlow/Keras that runs for 5 epochs. The size of the input to the MLP is same as the state size and the hidden layers have $(128, 256, 256, 256, 128)$ neurons respectively. A dropout layer (of rate 0.2) is included after the input layer to reduce the possibility of overfitting. The activation function for all the layers, save for the output layer, is ReLU. We use 2^{15} data for training and the same amount of data for testing. The batch size is kept at default, 32.

For 5-round SPECK-32 and 7-round SPECK-128, the results are summarized in Table 1, where the valid distinguishers (i.e., the accuracy is significantly $> \frac{1}{t}$) are marked for better readability. While considering more the two input differences together, it appears that the input difference 100000 has a greater impact on SPECK-32. Including this input difference in a previous set of input differences (for which a valid distinguisher is not found) yields a valid distinguisher. More research would be needed to explain the observation.

We also describe distinguishers for SPECK-32 and SPECK-128 for smaller rounds. The outcomes are given in in Table 2 (Table 2a for SPECK-32, Table 2b for SPECK-128). The results for SPECK-32 are done for the input differences (79042080, 1000000), and that for SPECK-128 are done for the input differences (1000000, 1).

[4] https://www.tensorflow.org/
[5] https://keras.io/
[6] https://pytorch.org/
[7] https://github.com/inmcm/Simon_Speck_Ciphers/blob/master/Python/simonspeckciphers/speck/speck.py

Table 1: Accuracy of ML training for 5-round SPECK-32 and 7-round SPECK-128 (TensorFlow/Keras)

	Input Differences	Accuracy
SPECK-32 5-round	79042080, 100000	0.5416
	79042080, 100000, 52030701	0.3595
	79042080, 100000, 52030701, 8710609	0.2729
	20400040, 52030701, 8710609	0.3333
SPECK-128 7-round	1000000, 1	0.8266
	1240004000000000801042004000000, 1	0.7580

Table 2: Accuracy of ML training for reduced round SPECK-32 and SPECK-128 (TensorFlow/Keras)

(a) SPECK-32

Rounds	Accuracy
3	0.83
4	0.68

(b) SPECK-128

Rounds	Accuracy
5	0.99
6	0.96

One-bit Input Differences. We apply the concept of choosing the 1-bit input differences, which is inspired from [10]. While the choice of such input differences in [10] is proposed to find the location of the differential fault attack (DFA) [6, Section 5.1], we notice that it can be linked to the classical differential distinguisher (for a systematic method to generate the input differences).

Taken from [10], the input differences in this category are all possible 1-Hamming weight cases. In other words, given the state size of the cipher n, we choose n input differences; where the bit at location i is set to 1 and the rest are 0, $\forall i \in \{0, 1, \ldots, n-1\}$. Thus, the input differences are chosen systematically (instead of those in Section 5.1, which are chosen in arbitrary or in ad-hoc manner). SPECK-32, having state size of 32; the number of classes is 32 in this category, and the accuracy for RANDOM is 0.03125.

The average time in seconds for training and validation (not counting the time taken for data generation) per round is indicated in Table 3a (PyTorch) and Table 3b (TensorFlow/Keras). It may be noted that, PyTorch (with default options) can possibly distinguish 7-rounds of SPECK-32, but TensorFlow/Keras (with default options) cannot go beyond 6 rounds; even though size of training/testing data and hyper-parameters are kept the same. Although, the choice of the activation function appears to drastically affect the accuracy/coverage. More experiments are needed to understand the observations fully.

Results from PyTorch. The results from PyTorch over various settings are given in Table 3a (rest settings are kept at default). The `in_features` size of the first linear layer and the `out_features` size of the last linear layer is 32[8]. The rest `in_features`/`out_features` are as indicated. A dropout layer of rate 0.2 is applied after the first linear layer. No separate SoftMax layer is used at the output layer for PyTorch, as the `CrossEntropyLoss`[9] combines LogSoftMax. Differential distinguishers can be observed till 6 rounds of SPECK-32 with all the activation functions tested, except for the TanhShrink activation function which does not seem to find any distinguisher even at 1-round (not included in Table 3a). On top, a strong indication that the distinguisher follows through the 7^{th} round can be noted with activation functions ReLU; as well as with its general forms, PReLU, RReLU, ReLU6 and LeakyReLU[10].

Results from TensorFlow/Keras. Results for round-reduced SPECK-32 from TensorFlow/Keras are given in Table 3b. A SoftMax layer is applied at the output layer with neuron size 32, which is not included for

[8] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear

[9] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#torch.nn.CrossEntropyLoss

[10] Although the deviation of accuracy for the RANDOM case is small, the same deviation is observed through repeated trials of the same experiment.

Table 3: Results for one-bit input differences for round reduced SPECK-32
(a) PyTorch

SPECK-32	Architecture (MLP)				Data Size		Accuracy		Average
Rounds	Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
5	32, 128, 256, 112, 96, 128, 256, 128, 64	PReLU	32	12	$2^{23.2604}$	$2^{23.0211}$	0.37183	0.35452	36457.5
6							0.09460	0.09638	
7							0.03439	0.03490	
5		ReLU					0.27554	0.23028	21748.3
6							0.08190	0.07742	
7							0.03375	0.03395	
5	128, 256, 112, 256, 128, 64	ReLU	32	10	$2^{21.5049}$	$2^{20.9881}$	0.35440	0.37222	4279.6
6							0.09169	0.09386	
7							0.03406	0.03442	
5		RReLU	128				0.32060	0.33570	1591.6
6							0.08538	0.08615	
7							0.03407	0.03386	
5		ReLU6					0.32116	0.33948	1502.4
6							0.08596	0.08760	
7							0.03476	0.03367	
5		LeakyReLU	32				0.35932	0.37501	4828.3
6							0.09220	0.09365	
7							0.03404	0.03386	
5		SELU					0.36505	0.37348	5558.9
6							0.09592	0.09746	
7							0.03172	0.03112	
5		HardTanh					0.37314	0.37872	5533.0
6							0.09111	0.09250	
7							0.03135	0.03118	
5		Tanh					0.37794	0.38250	6162.5
6							0.09481	0.09516	
7							0.03131	0.03123	
5		LogSigmoid	128				0.26537	0.27944	2142.7
6							0.06671	0.06753	
7							0.03135	0.03124	

the sake of brevity. Note that the differential distinguisher works till the 6^{th} round, with the activation functions ELU, SELU and ReLU. No indication for it to follow at the 7^{th} round is observed.

28000010 as a Fixed Input Difference. One notable contribution of [16, Section 4], is to find an interesting input difference for SPECK-32, 28000010. Gohr's idea is to choose the input difference 400000 (taken from [1, Table 7]), the idea here is to use an input difference with a low Hamming weight. When this constraint of low Hamming weight is lifted, as per the authors of [16], the best input difference turns out to be 28000010. Interestingly (or not), this input difference does not bode well when used with Gohr's distinguisher [23]. As the authors put it [16, Section 4]:

> In contrast, when we do not restrict the input difference, the best differential characteristics for 5 rounds is 0x2800/0010 → 0x850a/9520, with probability of 2^{-9}. However, when we trained the neural distinguishers to recognize ciphertext pairs with the input difference of 0x2800/0010, the neural distinguishers performed worse (an accuracy of 75.85% for 5 rounds). This is surprising as it is generally natural for a cryptanalyst to maximize the differential probability when choosing a differential characteristic.

(b) TensorFlow/Keras

SPECK-32	Architecture (MLP)				Data Size		Accuracy		Average
Rounds	Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
5	128, 256, 112, 256, 128, 64	Sigmoid	128	10	$2^{21.5049}$	$2^{20.9881}$	0.31239	0.31239	4087.1
5		Tanh	32				0.23050	0.24706	14040.7
5		ELU					0.34536	0.35649	15462.6
6							0.07299	0.07551	
5		SELU					0.33400	0.33844	15214.0
6							0.06105	0.06585	
5		ReLU	128		$2^{23.2558}$	$2^{23.0156}$	0.38132	0.39083	12400.5
6							0.07923	0.08075	

Not to be deterred by this revelation, we decide to have a try with the model from [7, Section 3.1][11]. As the model from [7, Section 3.1] requires at least one more input difference, we choose some other difference(s) along with 28000010 and give it a try with some arbitrarily chosen MLP. Results for the smaller (5-, 6- and 7-rounds) are given in Table 4. Since the results are promising at the first glance (the ML accuracy for 5-rounds is typically higher than that is reported in [16]), we decide to attempt larger (≥ 8) rounds As. luck would have it, basically everything paired up with 28000010 works as a distinguisher up to 8-round SPECK-32 with accuracy > 0.51. In some cases, the accuracy for 9-round is > 0.5, and very close to 0.51 (but < 0.51), but we refrain from counting those. As the icing on the cake, we choose some of the input differences reported in [11], and show the 8-round distinguisher.

Table 4: Results for SPECK-32 smaller (5, 6 and 7) rounds with 28000010 as an input difference (PyTorch)
(a) Large batch size and epochs (pair of differences)

SPECK-32	Input Difference	Architecture (MLP)				Data Size		Accuracy		Average
Round	(with 28000010)	Middle Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
5	1	128, 128, 128, 128, 64	ReLU	5000	150	$2^{23.7389}$	$2^{22.9315}$	0.81429	0.80833	18604.1
6								0.64266	0.64436	
7								0.57286	0.57229	
5	8000001							0.77971	0.78283	
6								0.63979	0.64164	
7								0.57277	0.57153	
5	02110A04							0.77721	0.77898	
6								0.64165	0.64328	
7								0.57289	0.57243	
5	1000000		PReLU					0.85139	0.85337	18182.3
6								0.66785	0.66860	
7								0.57372	0.57248	
5	80604101							0.79591	0.79705	
6								0.63662	0.63640	
7								0.57336	0.57221	

The results with 28000010 as a fixed input difference are consolidated in Table 5. As it can be seen, we adopt PyTorch and try with pairing 28000010 with various input differences. Overall, we use various MLPs (by varying number of layers/neuron size and activation function), with varying batch sizes and epochs; and with varying training/testing data sizes. We present 8-round distinguisher for SPECK-32, with some indication that it follows through the 9^{th} round as well. More specifically; Table 5a uses larger batch size (5000) and epochs (150) with pairs of input differences; Table 5b uses smaller batch size (32) and epochs (20) with pairs of input differences; Table 5c uses triplets of input differences; and

[11] The model from [7, Section 3.1] was already in released in public at the time (or before) of publication of [16], it is not clear why the authors of [16] did not attempt their newly found input difference with that model.

(a) Small batch size and epochs (pair of differences)

SPECK-32 Round	Input Difference (with 28000010)	Architecture (MLP) Middle Layers	Activation	Batch	Epochs	Data Size Training	Data Size Testing	Accuracy Training	Accuracy Testing	Average Time (s)
5	1000000	128, 128, 128, 128, 64	ReLU	32	20	$2^{23.7389}$	$2^{22.9315}$	0.84429	0.84646	9913.7
6								0.66208	0.66523	
7								0.56956	0.56909	
5	02110A04	256, 128, 64, 32, 16						0.78481	0.78772	10064.3
6								0.64322	0.64537	
7								0.57167	0.57249	
5	4000000		PReLU					0.86837	0.87020	9042.3
6								0.69521	0.69672	
7								0.57334	0.57315	
5	1000080		Tanh					0.77490	0.77520	9665.8
6								0.63378	0.63409	
7								0.56608	0.56594	

Table 5: Results for SPECK-32 8-round with 28000010 as an input difference (PyTorch)
(a) Large batch size and epochs (pair of differences)

Input Difference (with 28000010)	Architecture (MLP) Middle Layers	Activation	Batch	Epochs	Data Size Training	Data Size Testing	Accuracy Training	Accuracy Testing	Average Time (s)
1	128, 128, 128, 128, 64	ReLU	5000	150	$2^{23.7389}$	$2^{22.9315}$	0.51946	0.51039	19628.9
8000001							0.51965	0.51158	
4000000							0.52005	0.51160	
1000000							0.51948	0.51041	
400000							0.51983	0.51055	
02110A04							0.52055	0.51315	
80204101							0.51877	0.51065	
80604101							0.51936	0.51069	
80214101							0.51915	0.51075	
80614101							0.51922	0.51069	
10004440							0.51961	0.51027	
1000000		PReLU					0.52425	0.51322	21719.5
02110A04							0.52406	0.51151	
80204101							0.52337	0.51013	
80604101							0.52511	0.51413	

Table 5d uses triplets of input differences which form the second order differential. Some more results corresponding to possible distinguishers of 9-round SPECK-32 are shown in Table 6.

We remark that, using more epochs may tend to overfit. For instance, with respect to 9-round SPECK-32; (28000010, 1, 28000011), (28000010, 8000001, 20000011), (28000010, 4000000, 2C000010), (28000010, 1000000, 29000010), (28000010, 400000, 28400010); the training accuracy is less than 0.34 for 4000 batch size (and 20 epochs), but more than 0.34 for 5000 batch 150 epochs (shown in Table 6).

Thus, it is owing to the hard work of the authors of [16], we could ultimately find the 8-round distinguishers of SPECK-32 (i.e., by fixing 28000010 as an input difference. This is in a way ironic, since the best result from [16] covers only till 7 rounds of SPECK-32. Before that, our best result (with accuracy of about 0.59), was up to 7-rounds, with the following pairs of input differences: (1, 400000), (2, 400000), (8, 400000), (40, 400000), (200, 400000), (800, 400000), (1000, 400000), (10000, 400000), (400000, 20000000). Note that 400000 is common – this is the same input difference is used by Gohr [23] – though, in our case it is found by individually trying with all $\binom{32}{2}$ input difference pairs of Hamming weight 1 and thereafter choosing the best pairs. Granted, our 7-round distinguishers take only a few

(b) Small batch size and epochs (pair of differences)

Input Difference	Architecture (MLP)				Data Size		Accuracy		Average
(with 28000010)	Middle Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
1	128, 128, 128, 128, 64	ReLU	32	20	$2^{23.7389}$	$2^{22.9315}$	0.51342	0.51270	8196.0
804002							0.51333	0.51321	
80604101							0.51341	0.51303	
A604205							0.51356	0.51273	
28000011							0.51336	0.51343	
2000000							0.51298	0.51279	
4800020							0.51335	0.51306	
80000000							0.51358	0.51354	
800000							0.51321	0.51322	
1000000							0.51417	0.51413	
20000000							0.51321	0.51286	
8000							0.51357	0.51313	
1000080	256, 128, 64, 32, 16						0.51483	0.51369	8092.3
800001							0.51445	0.51334	
4000000							0.51428	0.51313	
10000							0.51281	0.51290	
40000000							0.51319	0.51314	
8000000							0.51308	0.51312	
2000							0.51331	0.51297	
4002							0.51320	0.51305	
4000							0.51334	0.51309	
2110A04							0.51607	0.51556	
80214101							0.51409	0.51261	
A204205							0.51497	0.51335	
80204101							0.51456	0.51343	
80614101							0.51455	0.51367	
400000							0.51373	0.51380	
1000000	128, 128, 128, 128, 64	PReLU					0.51322	0.51307	10084.5
1000080	256, 128, 64, 32, 16						0.51395	0.51256	7902.3
800001							0.51407	0.51279	
4000000							0.51526	0.51450	
2110A04							0.51639	0.51502	
1000080		Tanh					0.51380	0.51330	9103.2

minutes (thus considerably faster than probably all the competitors), still we would want to increment the number of rounds; this arguably would not have been possible without the assistance from [16].

(c) Triplet of differences

Input Differences	Architecture (MLP)				Data Size		Accuracy		Average
(with 28000010)	Middle Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
1, 8000001	128, 128, 128, 128, 64	ReLU	5000	150	$2^{24.3238}$	$2^{23.5165}$	0.35035	0.34454	31331.2
1, 4000000							0.34802	0.34068	
1, 1000000							0.34835	0.34093	
1, 400000							0.35285	0.34866	
1, FF8FFF8F							0.34975	0.34425	
1, 400000							0.35083	0.34542	
1, 80008000							0.34827	0.34082	
1, 850A9520							0.34814	0.34082	
4000000, 800001							0.34835	0.34073	
4000000, 1000000							0.34878	0.34077	
4000000, 400000							0.34856	0.34121	
800001, 400000							0.35308	0.34990	
800001, 1000000							0.34838	0.34110	
1, 4000000		PReLU					0.35068	0.34071	34340.4
1, 1000000							0.35086	0.34055	
1, 400000							0.35109	0.34035	
1, FF8FFF8F							0.35149	0.34143	
1, 80008000							0.35109	0.34046	
4000000, 800001							0.35135	0.34089	
4000000, 1000000							0.35088	0.34068	
800001, 1000000							0.35088	0.34071	

(d) Second order differential (triplet of differences)

Input Differences	Architecture (MLP)				Data Size		Accuracy		Average
(with 28000010)	Middle Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
1, 28000011	128, 128, 128, 128, 64	ReLU	4000	20	$2^{24.3238}$	$2^{23.5165}$	0.34299	0.34202	5131.6
8000001, 20000011							0.34304	0.34242	
4000000, 2C000010							0.34266	0.34214	
1000000, 29000010							0.34285	0.34231	
400000, 28400010							0.34267	0.34176	
1, 28000011			5000	150			0.34389	0.34094	35028.5
8000001, 20000011							0.34855	0.34102	
4000000, 2C000010							0.34851	0.34135	
1000000, 29000010							0.34992	0.34429	
400000, 28400010							0.34871	0.34111	
1, 28000011		PReLU					0.35283	0.34366	32816.8
8000001, 20000011							0.35049	0.34007	
4000000, 2C000010							0.35255	0.34379	
1000000, 29000010							0.35141	0.34166	

Table 6 shows some potential candidates for 9-round SPECK-32 distinguisher.

Table 6: Results for SPECK-32 9-round with 28000010 as an input difference (PyTorch; no confirmed distinguisher)

Input Difference(s)	Architecture (MLP)				Data Size		Accuracy		Average
(with 28000010)	Middle Layers	Activation	Batch	Epochs	Training	Testing	Training	Testing	Time (s)
1000080	128, 128, 128, 128 128, 128, 64	ReLU	32	20	$2^{23.7389}$	$2^{22.9315}$	0.49992	0.50000	9348.6
28000011							0.50014	0.50000	
1000000					$2^{24.2534}$	$2^{23.2534}$	0.50003	0.50000	15673.0
1	128, 128, 128, 128, 128, 64		128		$2^{23.7389}$	$2^{22.9315}$	0.51400	0.50026	3811.9
800001							0.50315	0.49996	
4000000			4000				0.50552	0.50015	3287.6
1000000							0.50473	0.50018	
400000							0.50466	0.49992	
1			5000	150			0.51529	0.49995	21621.2
800001							0.51506	0.50021	
4000000							0.51549	0.50011	
1000000							0.51517	0.49999	
400000							0.51055	0.49997	
02110A04							0.51581	0.50042	
80204101							0.51599	0.50016	
80604101							0.51503	0.50017	
80214101							0.51566	0.50010	
80614101							0.51540	0.50035	
10004440							0.51547	0.49995	
1, 28000011					$2^{24.3238}$	$2^{23.5165}$	0.34520	0.33346	30006.0
8000001, 20000011							0.34493	0.33367	
4000000, 2C000010							0.34484	0.33333	
1000000, 29000010							0.34532	0.33345	
400000, 28400010							0.34537	0.33355	
1, 800001							0.34537	0.33346	
1, 4000000							0.34532	0.33342	
1, 1000000							0.34548	0.33316	
1, 400000							0.34532	0.33360	
1, FF8FFF8F							0.34494	0.33347	
1, 400000							0.34552	0.33329	
1, 80008000							0.34502	0.33355	
1, 850A9520							0.34512	0.33374	
4000000, 800001							0.34530	0.33361	
4000000, 1000000							0.34547	0.33345	
4000000, 400000							0.34514	0.33342	
800001, 400000							0.34550	0.33369	
800001, 1000000							0.34498	0.33325	
1		PReLU			$2^{23.7389}$	$2^{22.9315}$	0.51647	0.49981	20667.9
800001							0.51794	0.50001	
4000000							0.51782	0.50018	
1000000							0.51839	0.50018	
400000							0.51808	0.49980	
02110A04							0.51681	0.50010	
80204101							0.51647	0.49992	
80604101							0.51783	0.50033	
1, 28000011					$2^{24.3238}$	$2^{23.5165}$	0.34640	0.33325	31879.6
8000001, 20000011							0.34694	0.33363	
4000000, 2C000010							0.34712	0.33351	
1000000, 29000010							0.34637	0.33350	
400000, 28400010							0.34734	0.33334	
1, 800001							0.34616	0.33312	
1, 4000000							0.34670	0.33347	
1, 1000000							0.34605	0.33355	
1, 400000							0.34656	0.33330	
1, FF8FFF8F							0.34603	0.33329	
1, 400000							0.34665	0.33340	
1, 80008000							0.34744	0.33337	
1, 850A9520							0.34595	0.33356	
4000000, 800001							0.34622	0.33340	
4000000, 1000000							0.34653	0.33330	
4000000, 400000							0.34652	0.33340	
800001, 400000							0.34642	0.33329	
800001, 1000000							0.34676	0.33344	

5.2 ASCON

The result for the rate part of ASCON[12] [21], which is the first 128-bits, is presented in Expression (1). For simplicity, each coefficient is rounded off to 5-decimal places. This acts a 3-round distinguisher which works with accuracy of 0.916. It is obtained using linear-kernel SVM where all the hyper-parameters are kept at its default value (except for `kernel` which is set to `linear` instead of the default `rbf`) with around $2^{14.96}$ training data and validated with $2^{12.96}$ testing data. For the input differences, we choose the mask value `1000`. We then XOR it with the 64-bit register x) to get δ_0, and XOR the same mask value with the register x_1 to get δ_1. For a given output difference, if the expression results as < 0, then it belongs to class 0 (i.e., corresponds to input difference δ_0). Otherwise, i.e., if the expression results as ≥ 0, then it belongs to class 1 (i.e., corresponds to input difference δ_1).

Effect of Truncation. Note that, taking only the rate part (128-bits, instead of the full state of 320-bits) does not give us any extra leverage. After collecting the output differences the attacker can employ any method, including truncating a part of it. Therefore, this falls within the model. Apart from that; so far our experiments suggest that when taking the full state, the same distinguisher always works with the same/higher accuracy than that of the truncated case. Thus, we believe that truncating part of the state may make the attacker's job more difficult, but will definitely not make it easy than it currently is. Indeed, with the full state of ASCON (320-bits) and keeping everything as-is, the accuracy increases to 1.00. This is obtained for around $2^{14.96}$ training data and the model is validated with around $2^{12.96}$ testing data.

Expression 1 SVM distinguisher for 3-round ASCON (rate/128-bits, accuracy 0.916)

$$
\begin{aligned}
&+ 0.06524x_0 + 0.25818x_1 - 0.07127x_2 - 0.02698x_3 - 0.00589x_4 - 0.32018x_5 + 0.00419x_6 \\
&+ 0.10561x_7 - 4.89209x_8 - 0.07874x_9 - 0.23816x_{10} - 0.01899x_{11} - 0.03706x_{12} \\
&+ 0.00224x_{13} - 0.13761x_{14} + 0.03035x_{15} - 0.01552x_{16} - 1.70353x_{17} - 0.32852x_{18} \\
&+ 0.16048x_{19} - 0.02296x_{20} - 0.03522x_{21} - 0.02862x_{22} - 0.01690x_{23} - 0.32018x_{24} \\
&- 0.04786x_{25} + 0.00340x_{26} - 0.13893x_{27} - 0.05532x_{28} + 0.16708x_{29} - 0.06691x_{30} \\
&- 0.02850x_{31} - 0.06942x_{32} - 0.03979x_{33} + 0.08352x_{34} - 0.12548x_{35} + 0.95676x_{36} \\
&+ 0.00000x_{37} - 0.14355x_{38} - 0.06691x_{39} - 0.03362x_{40} - 0.11080x_{41} - 0.07196x_{42} \\
&+ 0.19412x_{43} - 0.00180x_{44} - 0.00503x_{45} + 0.27334x_{46} + 0.04656x_{47} + 0.05862x_{48} \\
&+ 0.01036x_{49} - 0.22783x_{50} + 0.00008x_{51} - 0.10638x_{52} - 0.02959x_{53} + 0.09513x_{54} \\
&- 0.05866x_{55} - 0.02052x_{56} - 0.06191x_{57} + 0.10620x_{58} + 0.11661x_{59} + 0.04581x_{60} \\
&+ 0.57142x_{61} + 0.00000x_{62} - 1.00000x_{63} + 0.00708x_{64} - 0.02973x_{65} - 0.02207x_{66} \\
&- 0.00509x_{67} - 0.02888x_{68} - 0.28811x_{69} + 0.07271x_{70} + 0.01869x_{71} - 0.10360x_{72} \\
&- 0.01156x_{73} - 0.31847x_{74} - 0.06710x_{75} + 0.02993x_{76} - 0.00578x_{77} - 0.18291x_{78} \\
&+ 0.09424x_{79} + 0.84935x_{80} + 0.00000x_{81} + 0.08682x_{82} + 0.39318x_{83} + 0.13964x_{84} \\
&- 1.05348x_{85} + 0.03237x_{86} - 0.12471x_{87} + 0.16543x_{88} + 0.08003x_{89} + 0.07077x_{90} \\
&+ 0.02339x_{91} - 0.00371x_{92} - 0.03341x_{93} + 0.13572x_{94} + 0.20409x_{95} + 0.01148x_{96} \\
&- 0.04107x_{97} + 0.14575x_{98} - 0.30807x_{99} - 0.00354x_{100} - 0.69512x_{101} + 0.86495x_{102} \\
&- 0.06458x_{103} + 0.02611x_{104} + 0.34864x_{105} - 0.02176x_{106} - 0.02630x_{107} + 0.58935x_{108} \\
&- 0.02643x_{109} + 0.00852x_{110} - 0.06558x_{111} - 0.00644x_{112} - 0.05778x_{113} + 0.52099x_{114} \\
&+ 0.00206x_{115} + 0.03979x_{116} - 0.01654x_{117} + 0.01060x_{118} + 0.00693x_{119} + 0.07832x_{120} \\
&- 0.10912x_{121} + 0.00012x_{122} + 0.16375x_{123} + 0.18298x_{124} - 0.97580x_{125} + 0.28003x_{126} \\
&- 0.81702x_{127} - 0.03862
\end{aligned}
$$

[12]The latest version, `ASCONv1.2` is used here and denoted as ASCON for simplicity.

5.3 SIMECK

Here we describe our findings for SIMECK-32 and SIMECK-64 [31]. We take $\delta_0 = 1$ and $\delta_1 = 2$ for all the cases. All the results are from an MLP model with the hidden layers having $(128, 256, 256, 256, 128)$ neurons respectively, and run for 5 epochs. We achieve accuracy of 0.526 for 9-round SIMECK-32, and 0.55 for 14-round SIMECK-64, both with 2^{15} training data (validation is done with equal amount of testing data). More information regarding earlier rounds can be found in Table 7 (Table 7a for SIMECK-32, Table 7b for SIMECK-64).

Table 7: Accuracy of ML training for reduced round SIMECK-32 and SIMECK-64

(a) SIMECK-32

Rounds	Accuracy
8	0.683
9	0.526
10	0.500

(b) SIMECK-64

Rounds	Accuracy
11	0.83
12	0.75
13	0.64
14	0.55
15	0.50

5.4 SKINNY

Similar to ASCON SVM (Section 5.2), we show the results for SKINNY-128 unkeyed permutation [13]. Table 8 shows the summarized results (only training data size and training accuracy are shown), with two input difference pairs. Further, Expression (2) shows an example for input difference pair (1, 00005900000000000000000000000000) for 6-round SKINNY-128 with linear kernel SVM (that works with accuracy of 0.54556). For a particular test case, if it results as < 0 then it belongs to class 0 (i.e., corresponds to input difference $\delta_0 = 1$), otherwise it belongs to class 1 (i.e., corresponds to input difference $\delta_1 =$ ffffffffffffffffffffffffffffffff).

Table 8: Results for SKINNY-128 with SVM

(a) 6-round

Input differences	Kernel	Data size		Accuracy	
		Training	Testing	Training	Testing
1, ffffffffffffffffffffffffffffffff	Linear	$2^{14.28771}$	$2^{13.7732}$	0.54865	0.5535
	RBF			0.9997	0.9895
	Polynomial			1.0	0.9912
1, 80000000000000000000000000000080	Linear			0.56725	0.5562
	RBF			0.9992	0.9875
	Polynomial			1.0	0.9897

(b) 7-round

Input differences	Kernel	Data size		Accuracy	
		Training	Testing	Training	Testing
1, 59000000000000000000000000000000	Linear	$2^{19.93157}$	$2^{19.194605}$	0.5049	0.5050
	RBF			0.7745	0.5396
	Polynomial			0.7639	0.5456

Expression 2 SVM distinguisher for 6-round `SKINNY-128` (accuracy 0.54556)

$$
\begin{aligned}
&- 0.000002458x_0 - 0.00000453x_1 - 0.00001726x_2 - 0.00002808x_3 + 0.00001783x_4 - 0.00000363x_5 - 0.00002760x_6 \\
&- 0.00001677x_7 - 0.00003956x_8 - 0.00000467x_9 - 0.00001278x_{10} + 0.00000166x_{11} + 0.00001986x_{12} \\
&+ 0.00001712x_{13} + 0.00000503x_{14} + 0.00001689x_{15} + 0.00002077x_{16} - 0.00003732x_{17} - 0.00001814x_{18} \\
&- 0.00002409x_{19} - 0.00003847x_{20} - 0.00006143x_{21} - 0.00000067x_{22} + 0.00000691x_{23} - 0.00000597x_{24} \\
&+ 0.00000018x_{25} + 0.00000000x_{26} + 0.00002974x_{27} - 0.00000331x_{28} - 0.00008007x_{29} + 0.00001104x_{30} \\
&- 0.00000219x_{31} - 0.00000038x_{32} - 0.00005310x_{33} - 0.00004009x_{34} - 0.00002686x_{35} - 0.00000967x_{36} \\
&- 0.00024361x_{37} - 0.00004561x_{38} - 0.00007616x_{39} - 0.00003045x_{40} + 0.00000026x_{41} - 0.00001561x_{42} \\
&- 0.00000510x_{43} - 0.00000569x_{44} - 0.00001290x_{45} + 0.00000030x_{46} - 0.000000976x_{47} - 0.00000400x_{48} \\
&+ 0.00006144x_{49} - 0.00003996x_{50} + 0.00000411x_{51} - 0.00004234x_{52} - 0.00000999x_{53} - 0.00001662x_{54} \\
&- 0.00001821x_{55} + 0.00002785x_{56} + 0.00016537x_{57} + 0.00001928x_{58} + 0.00001700x_{59} - 0.00006496x_{60} \\
&- 0.00011006x_{61} + 0.00000138x_{62} - 0.00006339x_{63} - 0.00005156x_{64} + 0.00003192x_{65} - 0.00001398x_{66} \\
&+ 0.00001874x_{67} - 0.00012107x_{68} - 0.00010488x_{69} - 0.00005654x_{70} - 0.00005476x_{71} + 0.00000765x_{72} \\
&+ 0.00004549x_{73} + 0.00001019x_{74} - 0.00000517x_{75} - 0.00001394x_{76} - 0.00022932x_{77} + 0.00001376x_{78} \\
&- 0.00002833x_{79} - 0.00000946x_{80} - 0.00000643x_{81} - 0.00000823x_{82} + 0.00001040x_{83} - 0.00003902x_{84} \\
&- 0.00001667x_{85} - 0.00000758x_{86} + 0.00003016x_{87} - 0.00003748x_{88} - 1.99938367x_{89} - 0.00004061x_{90} \\
&- 0.00002421x_{91} - 0.00001997x_{92} - 0.00008293x_{93} - 0.00011033x_{94} + 0.00004228x_{95} + 0.00000025x_{96} \\
&+ 0.00002680x_{97} + 0.00000691x_{98} - 0.00001166x_{99} - 0.00003569x_{100} - 0.00000056x_{101} + 0.00001540x_{102} \\
&- 0.00000333x_{103} + 0.00001192x_{104} + 0.00000612x_{105} - 0.00001477x_{106} - 0.00001475x_{107} + 0.00000492x_{108} \\
&+ 0.00000779x_{109} + 0.00001762x_{110} + 0.00001734x_{111} + 0.00000166x_{112} + 0.00001838x_{113} + 0.00003240x_{114} \\
&- 0.00000428x_{115} - 0.00001534x_{116} - 0.00003687x_{117} + 0.00001803x_{118} + 0.00000481_{119} + 0.00003978x_{120} \\
&+ 0.00001102x_{121} + 0.00004022x_{122} - 0.00000616x_{123} - 0.00000095x_{124} - 0.00007751x_{125} - 0.00001126x_{126} \\
&- 0.00001836x_{127} + 1.00047451
\end{aligned}
$$

6 Conclusion and Future Directions

Similar to its predecessor [7], we anticipate this work too would be useful to the community. It shows the machine learning based differential distinguishers for the (round-reduced version of) unkeyed permutations — `SPECK-32` and `SPECK-128` [12], `ASCON` [21], `SIMECK` [31] and `SKINNY` [13].

For reduced-round version of the ciphers, we show how an attacker equipped with moderate understanding of machine learning tools can severely lower down the search complexity which would otherwise be expected for distinguishing it from the random scenario, while staying completely within the by classical differential attack model. Indeed, much of our work relies on some ad-hoc ML architecture and arbitrarily chosen input differences; still we are able to out-compete the otherwise computed complexity, and match the same number of rounds as the currently best-known ML-based attacks (that use more specialized/sophisticated architecture than ours) on `SPECK-32`.

It is possible to extend our existing model to do one/more of the followings:

1. Concatenate multiple ML distinguishers to cover more rounds.
2. Combine with traditional differential distinguishers (i.e., that does not involve ML).
3. Use specialized ML architecture (our ML-based distinguisher models are not tied to any particular ML architecture). For instance, one may use *reinforcement learning* to generate such architecture. In that case, Gohr's model would require twice as much neuron in the input layer. (as Optuna is used in [26]).
4. Support key recovery.

In this work, however, we keep the focus on the basic observation on how the state-of-the-art methods in traditional differential distinguishers might have underestimated the power of an attacker who has only basic knowledge of ML.

In response to the 8-round distinguisher on GIMLI reported in [7], the designers of GIMLI have commented in [17] that, "*it is not possible to extend their model to cover further rounds*". This assumption, in reality, is not true. There is no inherent limitation of the model that blocks itself from covering more rounds.

In the long run, we expect the bound for our ML assisted model can be increased with further research, as our results do not constitute the upper limit. The coverage of rounds could likely be extended with more training/testing data, deeper network, different choice of hyper-parameters, various activation functions etc. Also, the choice of the input differences play an important role. Therefore, it may be possible to increase the coverage only by choosing suitable input differences.

References

1. Abed, F., List, E., Lucks, S., Wenzel, J.: Differential cryptanalysis of round-reduced simon and speck. In Cid, C., Rechberger, C., eds.: Fast Software Encryption - 21st International Workshop, FSE 2014, London, UK, March 3-5, 2014. Revised Selected Papers. Volume 8540 of Lecture Notes in Computer Science., Springer (2014) 525–545 10
2. Avanzi, R.: A salad of block ciphers. Cryptology ePrint Archive, Report 2016/1171 (2016) https://eprint.iacr.org/2016/1171. 3
3. Bacuieti, N., Batina, L., Picek, S.: Deep neural networks aiding cryptanalysis: A case study of the speck distinguisher. In Ateniese, G., Venturi, D., eds.: Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Volume 13269 of Lecture Notes in Computer Science., Springer (2022) 809–829 1, 2, 6
4. Baksi, A.: New insights on differential and linear bounds using mixed integer linear programming (full version). Cryptology ePrint Archive, Report 2020/1414 (2020) https://eprint.iacr.org/2020/1414. 3
5. Baksi, A.: Classical and Physical Security of Symmetric Key Cryptographic Algorithms. Springer (2022) https://link.springer.com/book/10.1007/978-981-16-6522-6. 4
6. Baksi, A., Bhasin, S., Breier, J., Jap, D., Saha, D.: Fault attacks in symmetric key cryptosystems. Cryptology ePrint Archive, Report 2020/1267 (2020) 9
7. Baksi, A., Breier, J., Chen, Y., Dong, X.: Machine learning assisted differential distinguishers for lightweight ciphers. Cryptology ePrint Archive, Report 2020/571 (2020) https://eprint.iacr.org/2020/571. 1, 2, 3, 4, 5, 7, 11, 18, 19
8. Baksi, A., Breier, J., Dasu, V.A., Dong, X., Yi, C.: Following-up On Machine Learning Assisted Differential Distinguishers. SILC Workshop – Security and Implementation of Lightweight Cryptography (2021) https://www.esat.kuleuven.be/cosic/events/silc2020/wp-content/uploads/sites/4/2020/10/Submission4.pdf. 4
9. Baksi, A., Breier, J., Dasu, V.A., Hou, X.: Machine Learning Attacks On SPECK. SILC Workshop – Security and Implementation of Lightweight Cryptography (2021) https://www.esat.kuleuven.be/cosic/events/silc2020/wp-content/uploads/sites/4/2021/09/Submission10.pdf. 4
10. Baksi, A., Sarkar, S., Siddhanti, A., Anand, R., Chattopadhyay, A.: Fault location identification by machine learning. IACR Cryptology ePrint Archive (2020) 717 https://eprint.iacr.org/2020/717. 9
11. Bao, Z., Guo, J., Liu, M., Ma, L., Tu, Y.: Conditional differential-neural cryptanalysis. Cryptology ePrint Archive, Paper 2021/719 (2021) https://eprint.iacr.org/2021/719. 1, 2, 6, 11
12. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: Simon and speck: Block ciphers for the internet of things. Cryptology ePrint Archive, Report 2015/585 (2015) https://eprint.iacr.org/2015/585. 1, 7, 18
13. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. IACR Cryptology ePrint Archive **2016** (2016) 660 1, 8, 17, 18
14. Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II. (2016) 123–153 2
15. Bellini, E., Rossi, M.: Performance comparison between deep learning-based and conventional cryptographic distinguishers. Cryptology ePrint Archive, Paper 2020/953 (2020) https://eprint.iacr.org/2020/953. 1
16. Benamira, A., Gerault, D., Peyrin, T., Tan, Q.Q.: A deeper look at machine learning-based cryptanalysis. Cryptology ePrint Archive, Paper 2021/287 (2021) https://eprint.iacr.org/2021/287. 1, 2, 3, 6, 10, 11, 12, 13
17. Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli: Nist lwc second-round candidate (status update) (September 2021) https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/status-update-sep2020/gimli_update.pdf. 19
18. Chen, Y., Shen, Y., Yu, H., Yuan, S.: A new neural distinguisher considering features derived from multiple ciphertext pairs. Cryptology ePrint Archive, Paper 2021/310 (2021) https://eprint.iacr.org/2021/310. 1
19. Chen, Y., Yu, H.: Bridging machine learning and cryptanalysis via edlct. Cryptology ePrint Archive, Paper 2021/705 (2021) https://eprint.iacr.org/2021/705. 1

20. Cortes, C., Vapnik, V.: Support-vector networks. In: Machine Learning. (1995) 273–297 4
21. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. Submission to NIST (2019) https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/ascon-spec-round2.pdf. 1, 7, 16, 18
22. Ebrahimi, A., Regazzoni, F., Palmieri, P.: Reducing the cost of machine learning differential attacks using bit selection and a partial ml-distinguisher. Cryptology ePrint Archive, Paper 2021/1479 (2021) https://eprint.iacr.org/2021/1479. 1
23. Gohr, A.: Improving attacks on round-reduced speck32/64 using deep learning. Cryptology ePrint Archive, Paper 2019/037 (2019) https://eprint.iacr.org/2019/037. 1, 2, 3, 6, 10, 12
24. Haykin, S.: Neural Networks and Learning Machines (third edition). Pearson (2008) 4
25. Heuser, A., Zohner, M.: Intelligent machine homicide. In Schindler, W., Huss, S.A., eds.: Constructive Side-Channel Analysis and Secure Design, Berlin, Heidelberg, Springer Berlin Heidelberg (2012) 249–264 4
26. Kimura, H., Emura, K., Isobe, T., Ito, R., Ogawa, K., Ohigashi, T.: Output prediction attacks on block ciphers using deep learning. Cryptology ePrint Archive, Paper 2021/401 (2021) https://eprint.iacr.org/2021/401. 1, 18
27. Mouha, N., Wang, Q., Gu, D., Preneel, B.: Differential and linear cryptanalysis using mixed-integer linear programming. In: Information Security and Cryptology - 7th International Conference, Inscrypt 2011, Beijing, China, November 30 - December 3, 2011. Revised Selected Papers. (2011) 57–76 3
28. Pal, D., Mandal, U., Chaudhury, M., Das, A., Chowdhury, D.R.: A deep neural differential distinguisher for arx based block cipher. Cryptology ePrint Archive, Paper 2022/1195 (2022) https://eprint.iacr.org/2022/1195. 1
29. Sun, L., Wang, W., Wang, M.: Accelerating the search of differential and linear characteristics with the sat method. Cryptology ePrint Archive, Report 2021/213 (2021) https://eprint.iacr.org/2021/213. 3
30. Yadav, T., Kumar, M.: Differential-ml distinguisher: Machine learning based generic extension for differential cryptanalysis. Cryptology ePrint Archive, Paper 2020/913 (2020) https://eprint.iacr.org/2020/913. 1
31. Yang, G., Zhu, B., Suder, V., Aagaard, M.D., Gong, G.: The simeck family of lightweight block ciphers. Cryptology ePrint Archive, Report 2015/612 (2015) https://eprint.iacr.org/2015/612. 1, 7, 17, 18