Machine Learning Assisted Differential Distinguishers For Lightweight Ciphers

Anubhab Baksi*, Jakub Breier†, Yi Chen‡, Xiaoyang Dong‡
*Nanyang Technological University, Singapore
anubhab001@e.ntu.edu.sg
†Silicon Austria Labs, TU-Graz SAL DES Lab, Austria
Graz University of Technology, Austria
jbreier@jbreier.com
‡Tsinghua University, Beijing, PR China
chenyi19@mails.tsinghua.edu.cn, xiaoyangdong@tsinghua.edu.cn

***Abstract*—At CRYPTO 2019, Gohr first introduces the deep learning based cryptanalysis on round-reduced SPECK. Using a deep residual network, Gohr trains several neural network based distinguishers on 8-round SPECK-32/64. The analysis follows an 'all-in-one' differential cryptanalysis approach, which considers all the output differences effect under the same input difference.**

Usually, the all-in-one differential cryptanalysis is more effective compared to the one using only one single differential trail. However, when the cipher is non-Markov or its block size is large, it is usually very hard to fully compute. Inspired by Gohr's work, we try to simulate the all-in-one differentials for non-Markov ciphers through machine learning.

Our idea here is to reduce a distinguishing problem to a classification problem, so that it can be efficiently managed by machine learning. As a proof of concept, we show several distinguishers for four high profile ciphers, each of which works with trivial complexity. In particular, we show differential distinguishers for 8-round Gimli-Hash, Gimli-Cipher and Gimli-Permutation; 3-round Ascon-Permutation; 10-round Knot-256 permutation and 12-round Knot-512 permutation; and 4-round Chaskey-Permutation. Finally, we explore more on choosing an efficient machine learning model and observe that only a three layer neural network can be used. Our analysis shows the attacker is able to reduce the complexity of finding distinguishers by using machine learning techniques.

***Index Terms*—gimli, ascon, knot, chaskey, distinguisher, machine learning, differential**

I. Introduction

Machine Learning (ML) tools have made a great progress in the last decades. At present, these techniques are widely used in various fields, such as computer vision [1], machine translation [2], [3], autonomous driving [4], to name a few. In the aspect of cryptography however, usage of ML is mainly confined in the context of side channel analysis, such as [5], [6], to the best of our knowledge.

Therefore, the applicability of ML techniques in classical cryptanalysis is not much explored. It seems the community is rather skeptical regarding this approach. For example, the authors of [7] comment, "Neural networks are generally not meant to be great at cryptography. Famously, the simplest neural networks cannot even compute XOR, which is basic to many cryptographic algorithms".

An extended version of this paper can be found at [25].

Although ML techniques have been used against legacy ciphers, like the Enigma (the German cipher used during second world war) [8], it was not until the work by Gohr [9] at CRYPTO'19 that this line of research finally got its exposure. The idea here is applied to key recovery attacks on round-reduced `SPECK` using ML.

This work focuses on extending the commonly used model of differential distinguisher by using ML techniques. In the case of differential distinguisher, the attacker Eve XORs a chosen input difference δ to the input of the state of the (reduced round) cipher and watches for a particular output difference Δ, with randomly chosen inputs. If the (δ, Δ) pair occurs with a probability significantly higher for the (reduced round) cipher than what it should be for a random case, the (reduced round) cipher can be distinguished from the random case. This probability distribution of $\delta \rightarrow \Delta$ is modeled by differential branch number [10] or by automated tools like Mixed Integer Linear Programming (MILP) [11]. We extend the modeling for differential distinguisher by incorporating machine learning algorithms.

We argue that the usual modeling with branch number or MILP underestimates attacker's power. Having collected the output differences for the chosen input differences, the attacker can use any technique to distinguish the cipher from the random case. In such a situation, machine learning based techniques can reduce the search complexity estimated by existing methods. In fact, we observe that ML can reduce the search complexity to the cube root of the previously estimated bound.

Detailed discussion of machine learning is out of scope of this work, interested readers may refer to standard textbooks such as [12] for the same. We use TensorFlow[1] back-end with Keras[2] API. The optimizer function is based on Adam algorithm [13].

Our Contributions

In this work, we construct machine learning models to simulate the *all-in-one differentials* from [14]. We consider the classical distinguisher game: Given $\texttt{ORACLE} \overset{\$}{\leftarrow} \{\texttt{CIPHER}, \texttt{RANDOM}\}$,

[1]https://www.tensorflow.org/
[2]https://keras.io/

the attacker is to identify whether ORACLE = CIPHER with probability significantly $> \frac{1}{2}$ and with sufficiently small number of queries.

The core of our analysis lies in the actual problem of distinguishing the CIPHER from RANDOM into a classification problem. For this purpose, we propose two models (details are given in Section II). We choose the most common ML tool, *Multi-Layer Perceptron* (MLP) as a starting point (we have also tested with other ML tools, as described in Section IV).

As for transforming the differential distinguisher to a classification problem, we propose two models here. We apply the first model (described in Section II-A) to round-reduced GIMLI [15], ASCON [16] and KNOT [17]. All of the ciphers are the 2nd round candidates in the ongoing NIST Lightweight Cryptography (LWC) competition[3]. Further, we also show the effectiveness of the second model (described in Section II-B) over the lightweight MAC CHASKEY [18].

Results are described in Section III and Section IV, which can be summarized as follows:

- For GIMLI, we obtain 8-round practical distinguishers on GIMLI-HASH and GIMLI-CIPHER in nonce respecting case in Section III-A. The distinguishing complexity is $2^{17.6}$ offline data to train the model and $2^{14.3}$ online data to distinguish the cipher. Note that the authors of GIMLI [15] proved that the optimal 8-round differential trail is with probability of 2^{-52}. If we use this trail to distinguish 8-round GIMLI, we need at least 2^{52} online data. Thus, we are able to perform a differential distinguisher in around cube root complexity.
- For the ciphers, ASCON [16] and KNOT [17], we show practical distinguishers for reduced round versions of the underlying permutations in Section III-B. For the 320-bit ASCON-PERMUTATION, we show two separate distinguishers that work until 3-rounds with 2^{19} training data. For the 256-bit permutation of KNOT (KNOT-256), we show distinguishers for up to 10 rounds; and for the 512-bit permutation (KNOT-512), we show the same for up to 12-rounds. All results on KNOT are obtained with 2^{19} training data.
- For CHASKEY [18], [19], we present our results in Section III-C. With trivial training/testing complexity (2^{23} for training and $2^{14.3}$ for testing), we show the existence of a 4-round distinguisher. This contradicts the authors' claim that no 4-round differential distinguisher of complexity 2^{37} searches exists.

In Section IV, we discuss effects of choosing different neural network architectures with respect to 8-round GIMLI-PERMUTATION as the target cipher. We show that even a shallow three layer neural network works well for our purpose. We also report few differential distinguishers for 8-round GIMLI-PERMUTATION in the process.

With regard to our analysis, we emphasize on the following points:

- **Generality and practicality.** It is to be mentioned that our ML assisted model is generic in nature and can be adopted to any symmetric key setting. All the results presented are practical and can be carried out in around an hour on a modern computer.
- **Compatibility with differential distinguisher.** We would like to point out that we use the same attack model as that of the pre-existing differential distinguisher. The only distinction is made during the analysis, as ours is done by machine learning (instead of the branch number or automated tools like MILP). We do not fix any output difference a priori, instead all the differentials are fed to the ML.
- **Effect of truncation.** The results (Section III-A) for GIMLI-HASH and GIMLI-CIPHER are with truncated versions of the state, whereas the results for GIMLI-PERMUTATION (Section IV) are with the full state. Moreover, we argue that the truncation of the state does not fall outside the perceived model. Having collected the differentials, the attacker can employ any method (including truncating a part) based on the attacker's preference for analysis. This is also noted (with an example) in [20].
- **Extensibility.** The results presented in our work do not constitute the theoretical upper bound for the ML assisted distinguishers. By using a more sophisticated ML model and/or more training/testing data, it is likely that one can cover more rounds[4].

II. Machine Learning Based Distinguishers

A. Model 1: Multiple Input Differences

Under this model, the attacker chooses t (≥ 2) input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$. In the training (offline) phase, the attacker tries to learn whether there is any pattern in the CIPHER outputs that the machine learning tool is capable of finding. To test that, Eve creates t differentials with those input differences and feeds all the data to the machine learning. If the accuracy (a) of ML training is higher than what it should be for random data (i.e., $\frac{1}{t}$), the attacker is able extract pattern from the CIPHER outputs and proceeds to the online phase. Otherwise (if the training accuracy is $\frac{1}{t}$), she aborts.

During the testing (online) phase, the attacker would check if the sequence of classes predicted by the already-trained machine learning model matches the expected sequence (in which she has queried the ORACLE) with the same probability as training. Since she queries in a specific sequence to the ORACLE, the classes predicted by the machine learning would follow the specific sequence with the same probability as training if ORACLE = CIPHER; or would be arbitrary if ORACLE = RANDOM. Since the ORACLE can only choose between CIPHER and RANDOM, the machine learning produced sequence (of predicted classes) would either match the attacker's pre-perceived sequence with the same probability as

[3]https://csrc.nist.gov/projects/lightweight-cryptography

[4]New functionalities like higher order differential or key recovery can be incorporated too.

she has observed during training ($> \frac{1}{t}$); or that sequence would be arbitrary and hence would match the attacker's pre-perceived sequence with probability $\frac{1}{t}$. Therefore, if the accuracy of predicting the classes (a') is same as a, we infer the chosen ORACLE = CIPHER. On the other hand, if $a' = \frac{1}{t}$, we conclude ORACLE = RANDOM. A top-level description of the model is given in Algorithm 1.

B. Model 2: One Input Difference

While the first model (described earlier in Section II-A) can work with an arbitrary number of input differences, $t \geq 2$, here we propose a different model that can work with only one input difference. A top-level view can be found in Algorithm 2. As it can be seen, this model actually converts one input difference to a problem of classification with two classes. The case for ORACLE = RANDOM would result in training accuracy of $\frac{1}{2}$.

III. Results on Round-Reduced Ciphers

A. GIMLI *(Model 1)*

Using a SAT/SMT-based approach, the authors of GIMLI present the optimal differential trails up to 8 rounds [15], which are given in Table I. If we use the optimal 8-round differential trail to distinguish GIMLI-PERMUTATION, we need more than 2^{52} input pairs. Now, based on the ML assisted model presented in Section II-A (the first model), we present differential distinguishers till 8 rounds of GIMLI-HASH and GIMLI-CIPHER, each with training data of $2^{17.6}$ samples (and testing data of size $2^{14.3}$ samples). In both the cases, we choose only two input differences, so $t = 2$. A summary of results can be found in Table II.

TABLE I: Optimal differential trails for the round-reduced GIMLI-PERMUTATION

Rounds	1	2	3	4	5	6	7	8
Weight	0	0	2	6	12	22	36	52

We use the same MLP network for distinguishing 8-round GIMLI-HASH and GIMLI-CIPHER. The network has 5 middle layers with numbers of neurons as $(296, 258, 207, 112, 160)$. The activation function is ReLU, and the number of epochs is set to 20. We observe that our ML model is able to train the data collected with accuracy > 0.5 till 8-rounds (the accuracy drops to 0.5 from 9 rounds onward) for both GIMLI-HASH and GIMLI-CIPHER.

1) GIMLI-HASH.*:* For GIMLI-HASH, we focus on the processes of absorbing the last block of message M and squeezing the first 128-bit hash value h. Suppose the full message M is just 127 bytes (denote the i^{th} byte as $M[i]$), then after padded with a zero byte, the 128-bit block is absorbed into the sponge function. We generate the training data by flipping the least significant bit of byte $M[4]$ and $M[12]$. In other words, we process the message pairs with difference 1 in the 4th byte (as δ_0) and 12th byte (as δ_1). Then we compute the first 128-bit hash values pairs accordingly and collect the difference of hash values as training samples.

2) GIMLI-CIPHER.*:* For GIMLI-CIPHER, we assume there is only one associated data block, so at least 2 GIMLI permutations (48 rounds) are involved until the first output of the ciphertext block. Of 48 rounds, here we take the reduced version, namely up to 8 rounds to show our distinsguisher. For the data collection, we generate uniformly distributed 256-bit keys and 128-bit nonce pairs with difference 1 in the 4th byte (as δ_0) or 12th byte (as δ_1) of the nonces (similar to GIMLI-HASH). We set the associated data and the first message block m_0 to be zero. Then compute the ciphertext c_0 and the differences of it.

TABLE II: Accuracy of ML training on round-reduced GIMLI-HASH and GIMLI-CIPHER

Rounds	Accuracy	
	GIMLI-HASH	GIMLI-CIPHER
6	0.9689	0.9528
7	0.7229	0.6340
8	0.5219	0.5099

B. ASCON *and* KNOT *(Model 1)*

To show genericness of our methodology, here we present differential distinguishers on ASCON[5] [16] and KNOT [17], using the first model (Section II-A) for this purpose.

ASCON-PERMUTATION works with 320-bit state; and we only take 256 and 512-bit versions of KNOT-PERMUTATION. Our results show existence of differential distinguishers for up to 3 rounds of ASCON-PERMUTATION; up to 10 rounds of KNOT-256, and up to 12 rounds KNOT-512 with trivial complexity. Table III shows results for ASCON with 1–3 reduced rounds. Here, we XOR a mask value to the 64-bit register x_0 to get δ_0, and XOR the same mask value to the register x_1 to get δ_1. The results with the mask value 1000 are presented in Table III(a), and the same for the mask value 10001 are presented in Table III(b). For both the variants of KNOT, the mask value 1 is XORed to the 0th and 1st bytes of the state to generate the input differences, respectively. Similarly, Table IV shows the results for KNOT-PERMUTATION. In particular, Table IV(a) shows the results for KNOT-256 with 6–10 reduced rounds, and Table IV(b) shows that of KNOT-512 with 8–12 reduced rounds.

The size of neurons for the middle layers of the MLP used for ASCON and KNOT-256 is $(128, 1024,\ 1024,\ 1024)$, and for KNOT-512 it is $(256, 1024, 1024,\ 1024)$. Activation function in the hidden layers is taken as ReLU. Training is run for 20 epochs with 2^{19} training data (validation is done with testing data of equal size) samples in all the cases.

C. CHASKEY *(Model 2)*

Using the Lipmaa-Moriai formula, the authors of CHASKEY present some differential trails up to 8 rounds [18], which are given in Table V. Using the notion of pre-existing all-in-one differential, if we use the 4-round differential trail to distinguish CHASKEY, we need at least 2^{37} input pairs.

[5]We denote the latest version, ASCONv1.2, as ASCON for simplicity.

Algorithm 1: Model 1 (multiple input differences) for 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
```

Algorithm 2: Model 2 (one input difference) for differential distinguisher with machine learning

```
1:  procedure OFFLINE PHASE (Training)
2:      $TD \leftarrow (\cdot)$                                    ▷ Training data
3:      Choose random $P_0, P_1$ $(\neq P_0 \oplus \delta)$
4:      $P_2 = P_1 \oplus \delta$
5:      $C_i \leftarrow$ CIPHER$(P_i)$, for $i = 0, 1, 2$
6:      Append $TD$ with:
    $(0, C_0 \| C_1)$,                     ▷ $C_0 \| C_1$ is from class 0
    $(1, C_0 \| C_2)$                      ▷ $C_0 \| C_2$ is from class 1
7:      Repeat from Step 3 if required
8:      Train ML model with $TD$
9:      ML training reports accuracy $a$
10:     if $a > \frac{1}{2}$ then
11:         Proceed to Online phase
12:     else                                               ▷ $a = \frac{1}{2}$
13:         Abort
```

```
1:  procedure ONLINE PHASE (Testing)
2:      $TD' \leftarrow (\cdot)$                                   ▷ Testing data
3:      Choose random $P_0, P_1$ $(\neq P_0 \oplus \delta)$
4:      $P_2 = P_1 \oplus \delta$
5:      $C_i \leftarrow$ ORACLE$(P_i)$, for $i = 0, 1, 2$
6:      Append $TD'$ with $C_0 \| C_1$ and $C_0 \| C_1$ in order
7:      Test ML model with $TD'$ to get $\mathcal{C}$
                                 ▷ $\mathcal{C}$ is sequence of classes by ML
8:      $a'$ = probability that $\mathcal{C}$ matches $(0, 1)$
9:      if $a' = a > \frac{1}{2}$ then
10:         ORACLE = CIPHER
11:     else                                               ▷ $a' = \frac{1}{2}$
12:         ORACLE = RANDOM
13:     Repeat from Step 3 if required
```

TABLE III: Accuracy of ML training for reduced round ASCON-PERMUTATION

(A) Mask: 1000

Rounds	Accuracy
1	0.7499
2	0.9847
3	0.9861

(B) Mask: 10001

Rounds	Accuracy
1	0.8749
2	0.9990
3	0.8314

TABLE IV: Accuracy of ML training for reduced round KNOT-256 and KNOT-512

(A) KNOT-256

Rounds	Accuracy
6	0.9508
7	0.8984
8	0.8293
9	0.7096
10	0.5912

(B) KNOT-512

Rounds	Accuracy
8	0.9999
9	0.9989
10	0.9824
11	0.8472
12	0.6032

TABLE V: Differential trails for the round-reduced CHASKEY-PERMUTATION

Rounds	1	2	3	4	5	6	7	8
Weight	0	4	16	37	73	133	208	293

Here, we apply the second model of machine learning based differential distinguisher (described in Section II-B) on 4-round CHASKEY-PERMUTATION. The ML model from Gohr's [9] is used here, except the number of epochs is set to 10.

First, we extend the 4-round differential given by the designers [18] to get a new 5-round trail presented in Table VI, which shows the complexity for distinguishing 4-round CHASKEY-PERMUTATION would be at minimum 2^{37} input pairs. Next for our analysis, first two inputs P_0, P_1 are randomly generated. Thereafter, we obtain two pairs of samples: $P_0 \| P_1$, and $P_0 \| P_1 \oplus (00008400 \| 00000400 \| 00000000 \| 00000000)$. Then we follow the methodology as described in Section II-B. With training data size of 2^{23}, (validation is done with $2^{14.3}$ data) we observe the accuracy of ML is 0.616 for 4-round

CHASKEY-PERMUTATION.

TABLE VI: Differential trail for 5-round CHASKEY-PERMUTATION

Round(s)	$\triangle v_0$	$\triangle v_1$	$\triangle v_2$	$\triangle v_3$	Weight
0	c0240100	44202100	0c200008	0c200000	–
1	00008400	00000400	00000000	00000000	15
2	80000000	00000000	00000000	80000000	2
3	80008080	00000040	00000000	80109080	2
4	10409000	00542800	08400010	12408210	18
5	42828202	48540a0d	0a000090	10c0cb52	35

TABLE VII: Accuracy of ML training on reduced round CHASKEY-PERMUTATION

Rounds	Differential Probability	Training Accuracy
$0 \rightarrow 4$	2^{-37}	0.61618899

IV. Choice of Machine Learning Model

For finding a distinguisher, we opt for machine learning techniques that can reveal the hidden structures in the data without explicit feature selection. Our problem is treated as a classification problem where one tries to check whether a particular differential is inserted at the input to the (round-reduced) cipher or not. Results in this section are obtained by taking the 8-round distinguisher of GIMLI-PERMUTATION as the benchmark. For training, we use 2^{18} data samples. Number of epochs is set to 20 as for higher numbers the models tend to overfit.

When using machine learning algorithms, one has to make a choice on hyperparameters that would perform the best on the given problem. These parameters are normally chosen in an empirical way, by testing various network architectures and following best practices. There are automated techniques to tune the hyperparameters [21], [22]; however, these require significant resources which can be hard to emulate. Here, we report the outcome from the manual architecture search next. Results in this section are obtained by using Intel Xeon Silver 4215 processor with 256 GB of RAM.

We have tried several different neural network types, including the basic MLP, *Convolutional Neural Network* (CNN), and *Long Short-Term Memory Network* (LSTM). We have varied the width (number of neurons per layer) and the depth (number of layers) of these to find out the best accuracy and speed of learning. We have also tried several different types of activation functions.

Here, we choose our first model (Section II-A), with two differences, $\delta_0 = 1$ and $\delta_1 = 2$ (i.e., the least significant and the second least significant bits are flipped, respectively). The input layer size of the model is 128 (first 128-bits are squeezed), and the output layer size, $t = 2$. Our findings, which contain several distinguishers for 8-round GIMLI-PERMUTATION (highlighted), are stated in Table VIII. Architecture denotes number of neurons per layer starting from the input layer. Activation function denotes the function that was used in the hidden layers, as the output layer always used softmax. A summary can be given as follows:

TABLE VIII: Results for architecture search with 8-round GIMLI-PERMUTATION

Network	Architecture	Activation Function	Number of Parameters	Training Time (s)	Accuracy
MLP I	128, 296, 258, 207, 112, 160, 2	ReLU	226,633	1267.39	0.5588
MLP II	128, 1024, 2	ReLU	150,658	1081.67	0.5569
MLP III	128, 1024, 1024, 2	ReLU	1,200,256	1162.21	0.5652
MLP IV	128, 256, 128, 64, 2	LeakyReLU	90,818	510.9	0.5478
MLP V	128, 1024, 2	LeakyReLU	150,658	934.89	0.5516
MLP VI	128, 1024, 1024, 2	LeakyReLU	1,200,256	1778.5	0.5509
MLP VII	128, 1024, 1024, 1024, 2	ReLU	2,249,858	2410.1	0.5689
CNN I	128, 128, 128, 100, 2	ReLU	128,046	2951.7	0.5000
CNN II	128, 1024, 128, 128, 100, 2	ReLU	604,206	11503.0	0.5000
LSTM I	128, 256, 128, 2	tanh/sigmoid	444,162	50460.7	0.5316
LSTM II	128, 200, 100, 128, 2	tanh/sigmoid	313,170	39825.9	0.5325

- **CNNs** are not suitable for the purpose of finding a distinguisher. We have tried several architectures, and the accuracy was always 0.5. This is expected result, as CNNs are aimed at recognizing patterns in input data, which helps in image recognition or natural language processing, but does not work for cipher input where the bits are not related in any way. In fact, we observe distinguisher for 8-round GIMLI-PERMUTATION only except these models.
- **LSTMs** perform better than CNNs, but worse than fine-tuned MLP. The main drawback of LSTMs was the training speed – as they have recurrent layers, these have high memory requirements and more computations are required.
- **MLPs** provide the best accuracy, and can be tuned to train in very fast time. Our best result is achieved by MLP with 3 hidden layers and 1024 neurons per hidden layer (MLP VII in Table VIII), respectively. One can notice that in some cases we used Leaky ReLU as activation function [23], which allows a small, positive gradient when the unit is not active (e.g. input is negative), and therefore is considered more “balanced.” This is an advantage for smaller networks compared to normal ReLU, but for the network with 1.2M parameters, its performance is slightly worse.

V. Conclusion and Outlook

In this work, we propose two novel methods on finding distinguishers on symmetric key primitives by using machine learning. Our methods work with any number (non-zero) input differences. At the core, we use multiple differentials and convert the problem of distinguishing CIPHER from RANDOM into a classification problem, which is then tackled by a machine learning technique.

Overall, we generalize the classical (all-in-one) differential distinguisher [14], [24]. Unlike the usual differential distinguisher that relies on a rigorous observation and specifics of the target cipher, both of our approaches are much simpler and only rely on analyzing a set of input difference-output difference by machine learning. Thus, we propose the first proof of concept on how machine learning can be used as a generic tool in symmetric key cryptanalysis.

Our methods are also efficient as they can be carried out in around an hour by a modern computer. On the non-Markov ciphers GIMLI [15], ASCON [16], KNOT [17] and CHASKEY

[18], [19]; we show drastic reduction of search complexity reported by the designers for the round-reduced versions (typically of the order of a cube root).

A summary of advantages and limitations of our work can be given as:

+ The attack model is simple. In terms of neural network, our method works with as simple as a three layer neural network.
+ For round-reduced ciphers, we show that the complexity of differential cryptanalysis can be reduced to around a cube root of the claimed complexity. For example, the designers of `GIMLI` [15] claim the complexity of mounting a differential distinguisher by existing modeling methods would be at least 2^{52} data. However we are able to find the same with the complexity of around $2^{17.6}$ data (details can be found in Section III-A and Section IV).
+ Our work is generic, and can be applied to any symmetric key primitive where differential cryptanalysis can be applied. Here we target non-Markov ciphers as these are typically more complicated to analyze.
– So far, we are not able to extend our model to cover further rounds.
– For the time being, our model does not have a key recovery functionality.

That being said, we would like to emphasize that our work does not indicate the theoretical limit of the application of machine learning in symmetric key cryptography. Future works in this direction will likely cover more rounds with advanced ML modelling and/or more training/testing data. As noted in [20], new functionalities like key recovery, higher order differential can be achieved by switching to a *Support Vector Machine* (SVM)[6].

References

[1] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014. 1

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014. 1

[3] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016. 1

[4] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "Deepdriving: Learning affordance for direct perception in autonomous driving," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2722–2730. 1

[5] H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques," in *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, 2016, pp. 3–26. [Online]. Available: https://doi.org/10.1007/978-3-319-49445-6_1 1

[6] E. Cagli, C. Dumas, and E. Prouff, "Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing," in *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, 2017, pp. 45–68. [Online]. Available: https://doi.org/10.1007/978-3-319-66787-4_3 1

[6]It is possible to get the same functionalities with an artificial neural network, but it is computationally expensive compared to a support vector machine.

[7] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," *CoRR*, vol. abs/1610.06918, 2016. [Online]. Available: http://arxiv.org/abs/1610.06918 1

[8] S. Greydanus, "Learning the enigma with recurrent neural networks," *arXiv preprint arXiv:1708.07576*, 2017. 1

[9] A. Gohr, "Improving attacks on round-reduced speck32/64 using deep learning," in *Advances in Cryptology – CRYPTO 2019*, A. Boldyreva and D. Micciancio, Eds. Cham: Springer International Publishing, 2019, pp. 150–179. 1, 4

[10] J. Daemen and V. Rijmen, *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer-Verlag Berlin Heidelberg, 2002. [Online]. Available: https://www.springer.com/gp/book/9783540425809 1

[11] N. Mouha, Q. Wang, D. Gu, and B. Preneel, "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, pp. 57–76. [Online]. Available: https://doi.org/10.1007/978-3-642-34704-7_5 1

[12] S. Haykin, *Neural Networks and Learning Machines (third edition)*. Pearson, 2008. 1

[13] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 1

[14] M. R. Albrecht and G. Leander, "An all-in-one approach to differential cryptanalysis for small block ciphers," in *Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers*, 2012, pp. 1–15. [Online]. Available: https://doi.org/10.1007/978-3-642-35999-6_1 1, 5

[15] D. J. Bernstein, S. Kölbl, S. Lucks, P. M. C. Massolino, F. Mendel, K. Nawaz, T. Schneider, P. Schwabe, F. Standaert, Y. Todo, and B. Viguier, "Gimli," 2019. [Online]. Available: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/gimli-spec-round2.pdf 2, 3, 5, 6

[16] C. Dobraunig, M. Eichlseder, F. Mendel, and M. Schläffer, "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. 2, 3, 5

[17] W. Zhang, T. Ding, B. Yang, Z. Bao, Z. Xiang, F. Ji, and X. Zhao, "Knot: Algorithm specifications and supporting document," Submission to NIST, 2019, https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/knot-spec-round.pdf. 2, 3, 5

[18] N. Mouha, B. Mennink, A. V. Herrewege, D. Watanabe, B. Preneel, and I. Verbauwhede, "Chaskey: An efficient MAC algorithm for 32-bit microcontrollers," in *Selected Areas in Cryptography - SAC 2014 - 21st International Conference, Montreal, QC, Canada, August 14-15, 2014, Revised Selected Papers*, 2014, pp. 306–323. [Online]. Available: https://doi.org/10.1007/978-3-319-13051-4_19 2, 3, 4, 6

[19] N. Mouha, "Chaskey: a MAC algorithm for microcontrollers - status update and proposal of chaskey-12 -," *IACR Cryptology ePrint Archive*, vol. 2015, p. 1182, 2015. [Online]. Available: http://eprint.iacr.org/2015/1182 2, 6

[20] A. Baksi, J. Breier, V. A. Dasu, X. Dong, and C. Yi, "Following-up on machine learning assisted differential distinguishers," *SILC Workshop – Security and Implementation of Lightweight Cryptography*, 2021. [Online]. Available: https://www.esat.kuleuven.be/cosic/events/silc2020/wp-content/uploads/sites/4/2020/10/Submission4.pdf 2, 6

[21] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural computation*, vol. 12, no. 8, pp. 1889–1900, 2000. 5

[22] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012. 5

[23] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3. 5

[24] E. Biham and A. Shamir, "Differential cryptanalysis of des-like cryptosystems," in *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, 1990, pp. 2–21. [Online]. Available: https://doi.org/10.1007/3-540-38424-3_1 5

[25] A. Baksi, J. Breier, X. Dong, and C. Yi, "Machine learning assisted differential distinguishers for lightweight ciphers," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 571, 2020. [Online]. Available: https://eprint.iacr.org/2020/571 1