Following-up On Machine Learning Assisted Differential Distinguishers

Anubhab Baksi[1], Jakub Breier[2], Vishnu Asutosh Dasu[3], Xiaoyang Dong[4], and Chen Yi[4]

[1] Nanyang Technological University, Singapore

[2] Silicon Austria Labs, Graz, Austria

[3] TCS Research and Innovation, India

[4] Tsinghua University, Beijing, PR China

anubhab001@e.ntu.edu.sg, jbreier@jbreier.com, vishnu.dasu1@tcs.com, xiaoyangdong@tsinghua.edu.cn, chenyi19@mails.tsinghua.edu.cn

Abstract. Machine Learning (ML) is almost ubiquitously used in multiple disciplines nowadays, though its usage in cryptography so far is somewhat limited. In this regard, recent work done by Baksi, Breier, Dong and Yi (Eprint'20) attempts to find a generic application of ML in a form of a differential distinguisher. We revisit the framework here and show new results based on it. While our work follows the framework, we show extensions. In particular, we show a Support Vector Machine based distinguisher for 3-round Ascon; Multi-layer Perceptron based distinguishers 9-round Simeck-32 and 14-round Simeck-64. Each of our results work with trivial complexity.

Keywords: ascon, simeck, distinguisher, machine learning, differential

1 Introduction

In order to extend the application of Machine Learning (ML) in symmetric key cryptography, we revisit the concept of ML based differential distinguishers presented in [2]. We apply the concept of [2] to new ciphers, namely `ASCON` [4], `SIMECK` [9]. Further, we explore Support Vector Machine (SVM) which supplements the Multi-layer Perceptron (MLP) used in [2].

1.1 Motivation

As in [2], our work augments the classical differential distinguisher model by incorporating ML tools. 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 (reduced round) cipher. Then the 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 the (reduced round) cipher is substituted by a random source, then the attacker would be successful in distinguishing the (reduced cipher) from random.

The modelling of probability distribution for $\delta \rightsquigarrow \Delta$ is done through various methods, such as the wide trail strategy [1, Chapter 1.4] or Mixed Integer Linear Programming (MILP) [7] in the classical differential distinguisher.

One may note that, the attacker discards all the output differences which do not match Δ. This happens due to the very nature of the classical distinguisher. However, the assumption that the attacker will necessarily do this acts as a hindsight, since this underestimates the attackers capabilities.

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 a random source, this acts as a differential distinguisher. In this way, it is possible to reduce the complexity of the differential distinguisher drastically, even to the cube root of what is required for the classical case [2].

1.2 Machine Learning Basics

The Multi Layer Perceptron (MLP) An MLP [5] is a supervised learning algorithm which is a type of a feedforward Artificial Neural Network (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 feedforward 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 [3] are supervised learning algorithms which are predominantly used for classification problems with two classes. An SVM constructs a set hyperplanes 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.

Tools Used Similar to [2], for MLP, we use TensorFlow[1] back-end with Keras[2] API, with Adam optimizer [6]. For SVM, we use ThunderSVM[3]. All these tools are open-source.

1.3 Our Contributions

Our results, which are detailed in Section 3, can be summarized as follows:

- In Section 3.1, we present results on 3-round `ASCON`. These are obtained by using a linear-kernel SVM.

[1] `https://www.tensorflow.org/`
[2] `https://keras.io/`
[3] `https://github.com/Xtra-Computing/thundersvm`

- In Section 3.2, we show results on 9-round `SIMECK-32` and 14-round `SIMECK-64`, obtained using MLPs.

Similar to [2], all our results are practical and take less than an hour to perform on a modern computer. Further, it is to be noted that we do not assume any more power to the attacker than allowed in the classical differential distinguisher model. The only new ability the attacker has comes from how she analyses the information collected.

2 Machine Learning Based Distinguisher

As already mentioned, our model is adopted from that of [2]. Here, Eve chooses t (≥ 2) distinct input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$ and creates t differentials. In doing so, 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 $i = 0(1)t-1$.

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`[4]) 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 ($= \texttt{CIPHER}(P)$); then for each δ_i, she computes the output differences ($C \oplus C_i$ where $C_i = \texttt{CIPHER}(P \oplus \delta_i)$); 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 $\texttt{ORACLE}(P) \oplus \texttt{ORACLE}(P \oplus \delta_i)$ should be classified as belonging to class i). This way, she is able to measure the accuracy during testing. If $\texttt{ORACLE} = \texttt{CIPHER}$, then the testing accuracy should match that of the training phase, which is $> \frac{1}{t}$. Otherwise, i.e., if $\texttt{ORACLE} = \texttt{RANDOM}$, 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.

Training and Testing the Model With the algorithmic description given in Algorithm 1, the basic work-flow is described here (also adopted from [2]):

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

Algorithm 1: Differential distinguisher with machine learning (adopted from [2])

```
1: procedure OFFLINE PHASE (Training)
2:     $TD \leftarrow (\cdot)$                                ▷ Training data
3:     Choose random $P$
4:     $C \leftarrow \texttt{CIPHER}(P)$
5:     for $i = 0; i \leq t-1; i \leftarrow i+1$ do
6:         $P_i \leftarrow P \oplus \delta_i$
7:         $C_i \leftarrow \texttt{CIPHER}(P_i)$
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:     $C \leftarrow \texttt{ORACLE}(P)$
5:     for $i = 0; i \leq t-1; i \leftarrow i+1$ do
6:         $P_i \leftarrow P \oplus \delta_i$
7:         $C_i \leftarrow \texttt{ORACLE}(P_i)$
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:        $\texttt{ORACLE} = \texttt{CIPHER}$
13:    else                                            ▷ $a' = \frac{1}{t}$
14:        $\texttt{ORACLE} = \texttt{RANDOM}$
15:    Repeat from Step 3 if required
```

Training (Offline).

1. Select t (≥ 2) non-zero input differences $\delta_0, \delta_1, \ldots, \delta_{t-1}$.
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 $\texttt{ORACLE} = \texttt{CIPHER}$, 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 $\texttt{ORACLE} = \texttt{RANDOM}$, 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 Results on Round-Reduced Ciphers

3.1 ASCON

The result for the rate part of ASCON[5] [4], 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 not make it easy. 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. Due to space constraints, the expression is omitted here, and is made available online together with other resources[6].

3.2 SIMECK

Here we describe our findings for SIMECK-32 and SIMECK-64 [9]. 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 1 (Table 1(a) for SIMECK-32, Table 1(b) for SIMECK-64).

4 Conclusion and Future Directions

Following the concept of [2], we show that it is possible to reduce the search complexity of the classical differential distinguisher by employing ML tools. Our work covers distinguishers with trivial complexity for round reduced ciphers, ASCON [4] and SIMECK [9].

We supplement the work in [2] by SVM as an ML tool. In the process, we note the following:

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

[6]https://entuedu-my.sharepoint.com/:f:/g/personal/anubhab001_e_ntu_edu_sg/EgEiFUUgzGNGm6zuuEGalZwBE2QLOMFPdrKBvqH84jkowQ?e=KDCWlq.

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

$$
\begin{aligned}
&+0.06524x_{0}+0.25818x_{1}-0.07127x_{2}-0.02698x_{3}-0.00589x_{4}-0.32018x_{5}+0.00419x_{6}\\
&+0.10561x_{7}-4.89209x_{8}-0.07874x_{9}-0.23816x_{10}-0.01899x_{11}-0.03706x_{12}\\
&+0.00224x_{13}-0.13761x_{14}+0.03035x_{15}-0.01552x_{16}-1.70353x_{17}-0.32852x_{18}\\
&+0.16048x_{19}-0.02296x_{20}-0.03522x_{21}-0.02862x_{22}-0.01690x_{23}-0.32018x_{24}\\
&-0.04786x_{25}+0.00340x_{26}-0.13893x_{27}-0.05532x_{28}+0.16708x_{29}-0.06691x_{30}\\
&-0.02850x_{31}-0.06942x_{32}-0.03979x_{33}+0.08352x_{34}-0.12548x_{35}+0.95676x_{36}\\
&+0.00000x_{37}-0.14355x_{38}-0.06691x_{39}-0.03362x_{40}-0.11080x_{41}-0.07196x_{42}\\
&+0.19412x_{43}-0.00180x_{44}-0.00503x_{45}+0.27334x_{46}+0.04656x_{47}+0.05862x_{48}\\
&+0.01036x_{49}-0.22783x_{50}+0.00008x_{51}-0.10638x_{52}-0.02959x_{53}+0.09513x_{54}\\
&-0.05866x_{55}-0.02052x_{56}-0.06191x_{57}+0.10620x_{58}+0.11661x_{59}+0.04581x_{60}\\
&+0.57142x_{61}+0.00000x_{62}-1.00000x_{63}+0.00708x_{64}-0.02973x_{65}-0.02207x_{66}\\
&-0.00509x_{67}-0.02888x_{68}-0.28811x_{69}+0.07271x_{70}+0.01869x_{71}-0.10360x_{72}\\
&-0.01156x_{73}-0.31847x_{74}-0.06710x_{75}+0.02993x_{76}-0.00578x_{77}-0.18291x_{78}\\
&+0.09424x_{79}+0.84935x_{80}+0.00000x_{81}+0.08682x_{82}+0.39318x_{83}+0.13964x_{84}\\
&-1.05348x_{85}+0.03237x_{86}-0.12471x_{87}+0.16543x_{88}+0.08003x_{89}+0.07077x_{90}\\
&+0.02339x_{91}-0.00371x_{92}-0.03341x_{93}+0.13572x_{94}+0.20409x_{95}+0.01148x_{96}\\
&-0.04107x_{97}+0.14575x_{98}-0.30807x_{99}-0.00354x_{100}-0.69512x_{101}+0.86495x_{102}\\
&-0.06458x_{103}+0.02611x_{104}+0.34864x_{105}-0.02176x_{106}-0.02630x_{107}+0.58935x_{108}\\
&-0.02643x_{109}+0.00852x_{110}-0.06558x_{111}-0.00644x_{112}-0.05778x_{113}+0.52099x_{114}\\
&+0.00206x_{115}+0.03979x_{116}-0.01654x_{117}+0.01060x_{118}+0.00693x_{119}+0.07832x_{120}\\
&-0.10912x_{121}+0.00012x_{122}+0.16375x_{123}+0.18298x_{124}-0.97580x_{125}+0.28003x_{126}\\
&-0.81702x_{127}-0.03862
\end{aligned}
$$

1. The ML tools, so far, are basically used as black-boxes (by calling the high level APIs). It would be interesting if one can interpret the results in terms of the underlying mathematical structure. It would reveal more information and will likely yield more powerful distinguishers. Area of AI tackling this problem is called *explainable deep learning* [8].
2. One may note that, (linear-kernel) SVMs express the distinguisher as a linear inequality. Apart from allowing a visualisation, it could potentially lead to the following interesting research works (which could be hard to attain with MLPs):
   (a) *Key recovery.* As the distinguisher would still work if a round key is XORed, it may be possible to retrieve (part of) the round key using the linear inequality returned by SVM.
   (b) *Combination of differentials.* Separate SVMs can be mounted to generate the set of linear inequalities for a certain round. This may be combined to generate a possibly advanced distinguisher that may work with higher accuracy for that round.
   (c) *Higher order differential distinguisher.* We are only using the first order differentials so far. Using the SVM inequalities, it may be possible to combine multiple distinguishers; so that, it works as a higher order differential distinguishers, and possibly cover more rounds.
3. The hyper-parameters of the MLPs as well as the input differences are somewhat arbitrarily chosen. An in-depth study those topics may be conducted so that the performance of the distinguisher can be improved.

Table 1: 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

References

1. Avanzi, R.: A salad of block ciphers. Cryptology ePrint Archive, Report 2016/1171 (2016) `https://eprint.iacr.org/2016/1171`. 1
2. Baksi, A., Breier, J., Dong, X., Yi, C.: 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
3. Cortes, C., Vapnik, V.: Support-vector networks. In: Machine Learning. (1995) 273–297 2
4. 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, 5
5. Haykin, S.: Neural Networks and Learning Machines (third edition). Pearson (2008) 2
6. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014) 2
7. 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 1
8. Xie, N., Ras, G., van Gerven, M., Doran, D.: Explainable deep learning: A field guide for the uninitiated. arXiv preprint arXiv:2004.14545 (2020) 6
9. 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, 5