Assessing Block Cipher Security using Linear and Nonlinear Machine Learning Models

Ting Rong Lee[a], Je Sen Teh[a,*], Jasy Liew Suet Yan[a], Norziana Jamil[b], Jiageng Chen[c]

[a]*School of Computer Sciences, Universiti Sains Malaysia*
[b]*Department of Computing, College of Computing and Informatics, Universiti Tenaga Nasional*
[c]*School of Computer, Central China Normal University*

Abstract

In this paper, we investigate the use of machine learning classifiers to assess block cipher security from the perspective of differential cryptanalysis. The models are trained using the general block cipher features, making them generalizable to an entire class of ciphers. The features include the number of rounds, permutation pattern, and truncated differences whereas security labels are based on the number of differentially active substitution boxes. Prediction accuracy is further optimized by investigating the different ways of representing the cipher features in the dataset. Machine learning experiments involving six classifiers (linear and nonlinear) were performed on a simplified generalized Feistel cipher as a proof-of-concept, achieving a prediction accuracy of up to 95%. When predicting the security of *unseen* cipher variants, prediction accuracy of up to 77% was obtained. Our findings show that nonlinear classifiers outperform linear classifiers for the prediction task due to the nonlinear nature of block ciphers. In addition, results also indicate the feasibility of using the proposed approach in assessing block cipher security or as machine learning distinguishers[1].

Keywords: Classifier, cryptanalysis, encryption, machine learning, security

*Corresponding author
Email address: `jesen_teh@usm.my` (Je Sen Teh)

[1]Part of this work was presented at CRYPTOLOGY 2020 [1]

1. Introduction

Block ciphers are symmetric-key encryption algorithms, using a single secret key for both encryption and decryption tasks. A plaintext undergoes multiple rounds of key-dependent transformations to produce the resulting ciphertext. Block ciphers are designed using a variety of well-studied and security proven structures such as substitution-permutation networks (SPN), generalized Feistel structure (GFS) and Addition-Rotation-XOR (ARX). Recently, the design of compact lightweight block ciphers has become the focus of the cryptographic community due to the prevalence of highly constrained Internet of things devices [2, 3]. Block cipher security is usually evaluated on a *trial-by-fire* basis, whereby newer ciphers will be subjected to various attacks by cryptanalysts to ascertain their security levels. Resistance against differential cryptanalysis has become one of the de facto requirements when it comes to block cipher security. Cryptanalysts use searching algorithms [4] or mathematical solvers [5] to identify differential trails that occur with sufficiently high probability. These trails are then used as distinguishers in a key recovery attack. However, these algorithms or solvers become more computationally intensive as the number of rounds or block size increases.

As an alternative, researchers have explored the use of machine learning models for cryptanalytic purposes. Early applications mainly consist of training machine learning models to emulate the behaviour of ciphers given the assumption of a fixed secret key. For example in [6], a neural network was trained to encrypt data as simplified DES (SDES). Then, the cryptanalyst would be able to extract secret key information given plaintext-ciphertext pairs. A similar attempt using neural networks was used to perform known plaintext attacks on DES and Triple-DES in [7], whereby the neural networks were capable of decrypting ciphertexts without knowledge of the secret key. However, this approach has limited practicality as the neural networks were trained using plaintexts and ciphertexts corresponding to a specific key. If a different key is used, the model would have to be retrained using a separate dataset.

The same approach was used to cryptanalyze lightweight block ciphers, FeW and PRESENT [8, 9] with limited success. Neural networks were trained, validated and tested using plaintexts, ciphertexts and intermediate round data all generated using the same encryption key. The trained networks were unable to learn the behaviour of the block ciphers, achieving an accuracy of approximately 50%. Generally, the use of machine learning algorithms to cryptanalyze ciphers in a straightforward manner were only successful in older, classical ciphers. As an example, [10] trained a neural network to extract the encryption keys of Caesar, Vignere poly-alphabetic and substitution ciphers. Generative adversarial networks were also used to crack these classical ciphers in [11]. Both approaches achieved accuracies of up to 100%.

A more practical approach is the use of machine learning algorithms as cryptographic distinguishers. The classification capabilities of machine learning algorithms have been used to identify cryptographic algorithms from ciphertexts [12]. Classifiers were trained using known ciphertexts generated from a set of five commonly used cryptographic algorithms. A high identification rate of 90% was achieved if the same key was used for both training and testing ciphertexts. Another approach compared the performance of five different machine learning algorithms when distinguishing encrypted from unencrypted traffic [13]. They found that the C4.5 decision tree-based classification algorithm performed the best, achieving a detection rate of up to 97.2%.

In [14], a neural network was used to distinguish between right and wrong subkey guesses, similar to how a differential or linear distinguisher would be used for key recovery in traditional cryptanalysis. When the neural network is trained using plaintext-ciphertext pairs generated from a wrong key guess, it will produce random outputs that greatly differ from a cipher's actual outputs, whereas training using data generated from a correct key guess will lead to outputs with fewer errors. This allows a cryptanalyst to distinguish between right and wrong key guesses. The approach was tested on a hypothetical Feistel cipher as a proof of concept. [15] later introduced an attack on Speck32/64 using deep learning. A neural network model was trained using input and output differences corresponding to random keys, then used as a distinguisher. The proposed method outperforms existing differential attacks in terms of time complexity. However, it is unknown if the inclusion of other block cipher features could make the attack more efficient.

So far, the machine learning approaches are cipher-specific rather than being generalizable. To overcome this problem, we propose a generalizable approach to assess a block cipher's resistance against differential cryptanalysis using machine learning[2]. Rather than predicting or extracting key information, we investigate the capability of linear and nonlinear machine learning classifiers in determining if a block cipher is secure or insecure based on the number of active s-boxes. These classifiers are trained using various cipher features that include truncated input and output differences, permutation pattern, and the number of rounds. Data was generated using a modified Matsui's branch-and-bound algorithm [4]. Apart from determining the most suitable machine learning model and hyperparameters for the security prediction task, we also look into how data representation can affect prediction accuracy. Experiments are performed on a 4-branch GFS cipher, making the proposed approach generalizable to an entire class of block ciphers rather than a specific one. An in-depth comparison of six classifiers (linear and nonlinear) is performed. Our findings show that nonlinear classifiers outperform linear classifiers due to the nonlinear transformation involved in block ciphers, achieving a prediction accuracy of up to 95% when predicting *seen* cipher variants and up to 77% when predicting *unseen* cipher variants.

The rest of this paper is structured as follows: Section 2 introduces preliminary information required to understand the proposed work. Section 3 then provides the detailed steps and experimental setup involved in the proposed work. This is followed by experimental results in Section 3. Section 4 provides a discussion of our findings and its significance. The paper is concluded in

[2]Supplementary code for this paper is available at `https://github.com/trlee/ml-block-cipher`

Section 5 which includes some future directions of this work.

2. Preliminaries

2.1. Differential Cryptanalysis and Active S-boxes

$$\Delta X = X' \oplus X\text{"} \tag{1}$$

$$\Delta X = [\Delta X_0, \Delta X_1, ..., \Delta X_{i-1}], \tag{2}$$

where X' and X" are two individual plaintexts. An output difference is similarly defined where Y' and Y'' are the corresponding ciphertexts. The pair, $\{\Delta X, \Delta Y\}$ is known as a differential pair. For an ideal cipher, given any particular input difference ΔX, the probability of any particular ΔY occurring will be exactly $\frac{1}{2^b}$ where b is the block size. A successful differential attack require a differential, $\Delta X \rightarrow \Delta Y$ with a probability far greater than $\frac{1}{2^b}$.

An s-box is defined to be *differentially active* if its input is a non-zero difference. Rather than computing the concrete differential probability for a given differential pair, resistance against differential cryptanalysis can be estimated by calculating the number of active s-boxes. The estimated probability that input differences will be mapped to output differences can then be calculated based on the s-box's differential distribution table. The mapping of differences holds with a certain probability, 2^{-p}. By taking into consideration the best-case (from the attacker's perspective) s-box differential probability, a block cipher is considered to be secure if $2^{AS \times p} \geq 2^b$, where AS denotes the total number of active s-boxes. Figure 1 depicts an example of s-box activation for a 4-branch GFS cipher, whereby the left s-box is active.

An interesting property of differential cryptanalysis that we leverage upon in this work is the effect of round keys, rk_i being negated through the use of differences. Any random key can be used to generate differential pairs, thus the resulting dataset for machine learning experiments is not catered to a specific secret key. In addition, we are also able to generate an exhaustive dataset by

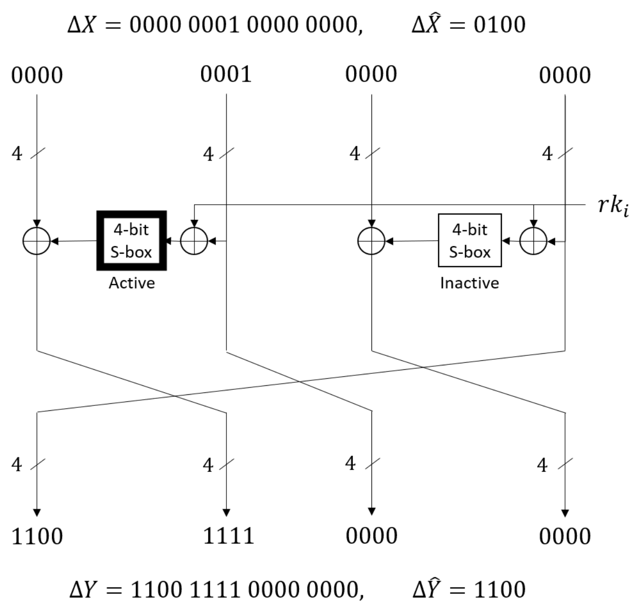

Figure 1: 1 round of a 4-branch GFS with 4-bit s-box

taking advantage of truncated differentials [16]. We can truncate the input differences based on the size of the s-box. For example, plaintext or ciphertext differences for a b-bit block cipher with s-bit s-boxes can be truncated to t-bit differences, where ($t = \frac{b}{s}$). Thus, each bit in the truncated difference denotes a non-zero difference corresponding to each s-bit word in the plaintext (or ciphertext) block. An example of how a differential pair $(\Delta X, \Delta Y)$ is mapped to a truncated differential pair $(\Delta \hat{X}, \Delta \hat{Y})$ is shown in Figure 1. However, the use of such a truncated difference would only be applicable to block ciphers that use a word-based permutation rather than bitwise permutation.

2.2. Matsui's Branch-and-Bound Differential Search

Matsui's branch-and-bound is an algorithm used for deriving the best differential or linear paths for differential and linear cryptanalysis. It is applicable to block ciphers that have s-box-like tables. The algorithm goes through all possible iterations of the differential paths, then prunes paths that have probabilities less than $\overline{B_n}$. $\overline{B_n}$ is defined as the best probability the running algorithm has found so far. An initial value has to be set for $\overline{B_n}$ and it should be as close to the actual probability B_n as possible to eliminate more non-promising paths earlier on. The $\overline{B_n}$ is constantly updated according to the best probability of the paths found so far which effectively reduces the potential search space. The process is repeated until the all the possible paths with respect to the branching rules and bounding criteria have been enumerated.

In the proposed work, we use a variant of Matsui's algorithm as described in [4]. We further simplify the algorithm as we only need the number of differentially active s-boxes rather than the concrete differential probability for our experiments. This greatly increases the speed of the search, which allows us to remove all bounding restrictions to generate large datasets for training and testing purposes.

2.3. 4-branch GFS Cipher

As a proof-of-concept, our proposed work is applied to a 4-branch GFS cipher, similar to the one in Figure 1. By using a GFS cipher with a word-based permutation, we can use truncated differences in our experiments. The 4-branch GFS effectively represents ultralightweight block ciphers with 16- or 32-bit blocks depending on whether 4-bit or 8-bit s-boxes are used. Regardless, experiments can be conducted under the assumption that both the 4-bit and 8-bit s-box would have a best differential probability of 2^{-2}. For example, PRESENT and AES s-boxes both have the best differential probability of 2^{-2}.

2.4. Machine Learning Classifiers

The proposed work investigates the performance of linear and nonlinear classifiers when predicting the security of block ciphers. Essentially, the goal is to have the classifiers learn the best hypothesis function (i.e. linear or nonlinear) to segregate the secure and insecure classes. A machine learning model refers to a trained classifier with specific features, machine learning algorithm and hyperparameters. This section describes the three linear and nonlinear classifiers used in our experiments.

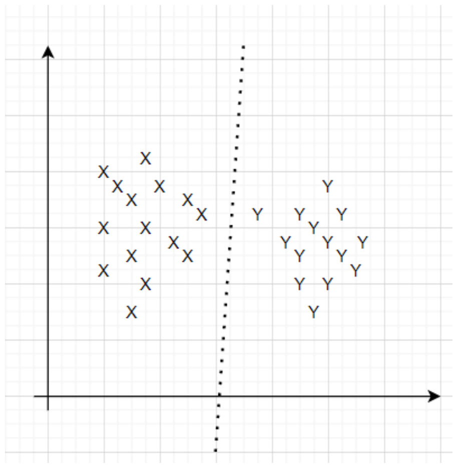

Figure 2: Linear Classification Problem

2.4.1. Linear Classifiers

As its name suggests, linear classifiers solve classification tasks based on a linear combination of features. In Figure 2 where X and Y are sample classes, the goal of linear classifiers is to segregate, as accurately as possible, the training data into their respective classes using a linear function (i.e., a straight line). We utilize three linear classifiers in our experiments: Tensorflow (TF) Linear classifier, and sklearn's logistic regression and single-layer perceptron.

Linear models predicts the probability of a discrete value/label, otherwise known as class, given a set of inputs. For the context of binary classification, the possible labels for the problem will only be 0 or 1. The linear model computes the input features with weights and bias. The weights indicate the direction of the correlation between the input features and the output label, whereas the bias acts as the offset in determining the final value of the label, should its conditions be fulfilled.

Logistic regression models the probabilities of an observation belonging to each class using linear functions and is generally considered more robust than regular linear classifiers. Unlike a linear function used by a linear classifier, the logistic regression model uses what is referred to as a sigmoid function (Fig. 3),

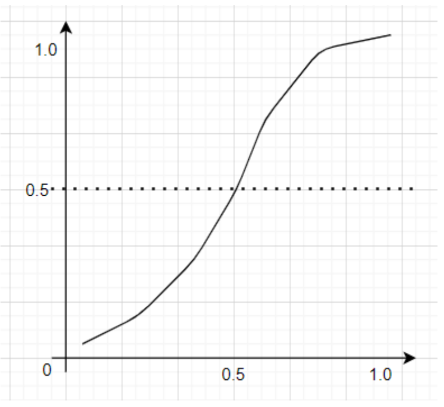

Figure 3: Sigmoid Function

and maps any real value of a problem into another value between the boundary of 0 and 1. For the case of machine learning, sigmoid functions are typically used for mapping the predictions of a model to probabilities. This structure is shared by both **TF's linear classifier** and **sklearn's logistic regression** models. Both models differ in terms of how the data is represented and used for training. In TF's linear classifier, training samples are pooled from the training dataset randomly in batches, and steps are defined by the total number of batch sampling that has to be performed before moving to the next epoch while sklearn's logistic regression model fits the data directly and trains its model throughout the epochs.

Single-layer perceptron is a linear classifier based on a threshold function

$$f(x) = w(x) + b, \tag{3}$$

where $f(x)$ is the output value, x is a real-valued input vector, w is the weight of the vector and b is the bias. When it comes to a binary classification task, the threshold function classifies x as either a positive or negative instance, with the weight and vector being the primary variable in determining the label, and bias is an additional paramter that can possibly adjust the label.

All aforementioned linear classifiers are tuned with respect to the following hyperparameters for optimal performance. The range of hyperparameter values

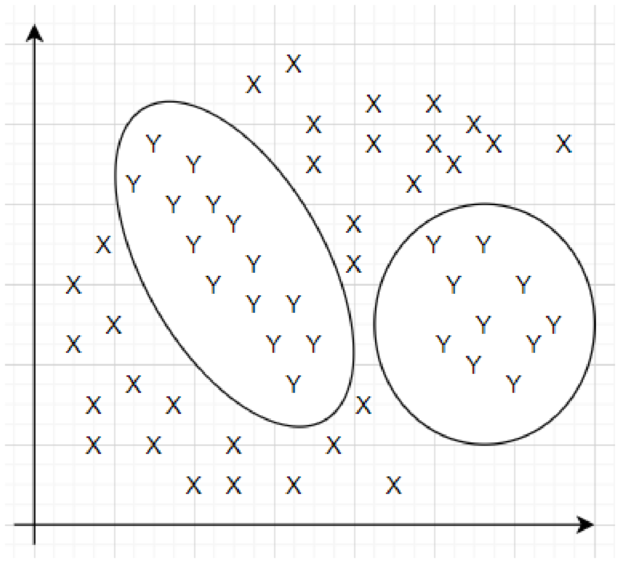

Figure 4: Nonlinear Classification Problem

that were tested is also denoted:

- *Stopgap* (Value range: [250,1000]): The total number of iterations that the model needs to undergo with no improvements before stopping the training process early.
- *Epochs* (Value range: [500,1000]): The total number of passes the model has to undergo throughout the training data batches.

2.4.2. Nonlinear Classifiers

Not all data can be segregated naturally using a straight line as shown in Figure 2. A nonlinear classifier allows the machine learning model to learn a nonlinear function or decision boundary to best separate the training data into two classes. The nonlinear classifiers used in this study are sklearn's k-nearest neighbors, decision tree and multi-layer perceptron.

k-nearest neighbor (KNN) is a type of instance-based learning that classifies new data based on majority voting of k number of training instances closest to it. Hyperparameters that can be tuned to optimize performance include:

- *NN* (Value range: {2,3,4}): The value of k as explained earlier. *NN* refers to the number of neighbors to be used for the k-neighbors query.

- *Distance*: This is measure used to determine the distance between two neighbours. The default Minkowski distance is used for all experiments.

- *Algo* (*kd_tree* or *ball_tree*): Algorithm used to compute the nearest neighbors for the model.

- $Leaf_S$ (Value range: {20,40}): Leaf size passed to the KD or Ball Tree, which can affect the speed of the tree construction and query, as well as memory required.

Decision tree classifiers are used to predict a class or value of the target variable by learning simple decision rules inferred from the training data. The model operates on the basis of "branching" from one decision node to another one deeper down until it finally reaches its desired output. Its parameters include:

- *Split*: The strategy used to choose the split on each node, which can be either *best* or *random*.

- $Leaf_N$ (Value range: [1000,5000] or *unlimited*): Maximum number of leaf nodes.

- *Sample Split*: Minimum amount of samples required to split an internal node. A default value of 2 is used.

Multi-layer perceptron (MLP) is a derivation of the perceptron model as described in Section 2.4.1, with added functions such as error functions and backpropagation to further improve performance of the model. The hyperparameters that are tuned to optimize the model are as follows:

- *Stopgap* (Value range: [250,1000])

- *Epochs* (Value range: [500,1000])

- *Activation*: The function that determines the outputs of the nodes. The default rectified linear function is used for all experiments.
- *Hidden layers* (Value range: [2,4]): The number of hidden layers of the neural network.
- *Nodes per hidden layer*: The number of nodes per hidden layer. We use a default value of 100 nodes per hidden layer for all experiments.

3. Experimental Setup

3.1. Dataset Generation

Each sample in the dataset used to train the machine learning classifiers consist of block cipher-related features. They are labelled as secure or insecure depending on the number of active s-boxes associated with the particular sample. For the target 4-branch GFS cipher, the features include the truncated input difference $\hat{X}$, truncated output difference $\hat{Y}$, number of rounds, r and a word-based permutation pattern, P, $\hat{X}$, $\hat{Y}$ and r are features shared by any block cipher whereas P is commonly used in GFS ciphers. Each training sample is essentially describes a truncated differential trail from $\hat{X}$ to $\hat{Y}$ for r number of rounds that goes through a GFS cipher with P permutation pattern. In our experiments, we use all $4! = 24$ possible permutation patterns for a 4-branch GFS. This also implies that there are 24 possible variants of the GFS cipher. Each cipher variant can generate a large set of data samples which consists of its truncated differential paths for different number of rounds.

We utilize the branch-and-bound algorithm described in Section 2.2 to automatically generate the dataset. The output of the branch-and-bound algorithm is the number of active s-boxes, AS which will be used alongside a security margin threshold, α to calculate the data labels (secure - 1, insecure - 0). If $AS > r\alpha$, the input sample is considered to be secure (labelled as 1) whereas if $AS \leq r\alpha$, the input sample is considered to be insecure (labelled as 0). In other words, α dictates the minimum number of active s-boxes per round for a block cipher to be considered secure. α can be configured based on the desired security margin that the cryptanalyst or designer requires.

We want to ensure that α is selected to be as strict as possible, while still allowing us to generate a balanced dataset for training purposes. $\alpha = 1$ is a loose bound, whereby a 16-bit and 32-bit cipher will require at least 8 rounds and 16 rounds respectively to be considered secure. On the other hand if $\alpha = 2$, a 16-bit and 32-bit cipher will require at least 4 rounds and 8 rounds respectively to be considered secure. Having $\alpha = 2$ is too restrictive as it requires all s-boxes to be active in every round. Thus, to ensure that the security bound is sufficiently strict while capable of generating a balanced dataset, we have selected $\alpha = 1.5$. Some samples from the dataset are shown in Table 1 (note that AS is not actually used for training).

Our experiments can be divided into three main phases: baseline setup, permutation feature representation, and generalization. In Phase 1, a balanced dataset (50:50) of 500000 samples are generated from all 24 variants of the GFS cipher. Note that the 500000 examples are randomly sampled from the exhaustive dataset using a single integer to represent the permutation pattern. We denote this method of representation as rep_1.

We compare the effect of the permutation representation on model performance in Phase 2. rep_1 is compared with rep_2 which represents the permutation as four separate features (one integer to map each truncated difference bit). As an example, the permutation pattern shown in Figure 1 can be represented by $rep_1 = \{1230\}$ or by $rep_2 = \{1, 2, 3, 0\}$. For Phase 2, we use the same 500000 samples from all 24 variants of the GFS cipher but created a second version of the dataset with permutation feature transformed into rep_2.

The third phase involves generalizing to *unseen* cipher variants. This phase reflects upon the capability of the trained machine learning classifiers to predict the security level of these *unseen* ciphers. We define an *unseen* cipher variant as a block cipher whose data was not used to train the machine learning classifiers. Thus, predicting the security of these *unseen* ciphers is analogous to predicting the security of newly proposed ciphers. In Phase 3, we test the classifiers'

Table 1: Sample Dataset where $\alpha = 1.5$

$\hat{X}$	$\hat{Y}$	P	r	AS	Label
1010	1010	0123	8	16	Secure
0111	1101	1203	11	17	Secure
1111	0100	3021	12	9	Secure
0010	0010	0123	5	5	Insecure
1111	0101	3021	12	6	Insecure
1101	1100	3120	11	6	Insecure

performance on three different *unseen* block cipher variants, which we denote as BC_1, BC_2 and BC_3. For each of these block ciphers, we generate a dataset consisting of 80000 samples, each. The difference between these datasets is the ratio of the number of secure to insecure samples (1:0). The ratios are summarized as:

- BC_1 - 1:3 (20000 to 60000)
- BC_2 - 1:1 (40000 to 40000)
- BC_3 - 3:1 (60000 to 20000)

BC_1 represents an insecure block cipher design, BC_2 represents a moderately secure block cipher design whereas BC_3 represents a secure block cipher design. In order to generate sufficient samples that fulfil these ratios, four block cipher variants are used, $P = \{0321, 1320, 2013, 3012\}$. Thus, the training dataset consists of 500000 samples generated from only 20 out of the 24 variants of the GFS cipher.

3.2. Experiments

Assessing block cipher security based on its features is a supervised learning problem which we framed as a binary classification task (1 for secure, 0 for insecure). We limit the scope of this paper to linear and nonlinear classifiers, where Tensorflow's (TF) linear classifier model, sklearn's single-layer perceptron and logistic regression models were selected as linear classifiers, and KNN, decision tree and MLP were selected as nonlinear classifiers. To optimize performance, various parameter combinations for each classifier will be tested. We also investigate the effect of data representation on prediction accuracy, specifically how the permutation patterns are represented. The experiments can be divided into three main phases which are detailed as follows:

- **Phase 1 - Baseline Setup** - The goal of this phase is to identify classifiers that are best suited for the prediction task. An 80:20 train-test split is performed on the dataset. Apart from the six classifiers, we also include a dummy classifier as a baseline model for performance comparison. Intuitively, the dummy classifier should have a prediction accuracy of 50% as it is a randomly guessing model that does not have any advantage in predicting security margins. For all classifiers, we investigate various hyperparameter combinations to maximize prediction performance. rep_1 is used as the permutation representation.

- **Phase 2 - Permutation Feature Representation** - In this phase, we investigate the effect of rep_1 and rep_2 on prediction accuracy. We select the best performing linear and nonlinear models (along with the optimal hyperparameter values) from Phase 1 and repeat the train-test procedure using the dataset generated from rep_2.

- **Phase 3 - Generalizability to *Unseen* Cipher Variants** - This phase consists of three separate experiments. In each one, we first train the machine learning classifiers using 500000 samples from the 20 *seen* cipher variants. Then, we separately test the performance of the models using the dataset from BC_1, BC_2 and BC_3. Unlike Phase 1, the training dataset will not contain a single sample from these *unseen* cipher variants. Thus, the test results will indicate if the classifiers are able to generalize to "new" ciphers with varying levels of security. For this experiment, the type of permutation representation will be selected based on results obtained in Phase 2.

Let S, TP, TN, FP, and FN represent the total number of samples, true positives, true negatives, false positives and false negatives respectively. The following metrics are used to evaluate the performance of each classifier in which secure is the positive class and insecure is the negative class:

- **Accuracy (Acc)**: The sum of true positives and true negatives divided by the total number of samples, $\frac{TP+TN}{S}$. Accuracy refers to the fraction of predictions that the model has correctly made.
- **Precision (Pre)**: True positives divided by the sum of true and false positives, $\frac{TP}{TP+FP}$. Precision refers to the percentage of correctly classified samples out of the total number of predictions made. We record the precision for both positive and negative classes as they are both equally important from the cryptographic perspective.
- **Recall (Rec)**: True positives divided by the sum of true positives and false negatives, $\frac{TP}{TP+FN}$. It represents the percentage of correctly classified samples out of the total number of actual samples that belong to a particular class. Similar to precision, we record the recall for both positive and negative classes.
- **F1 score (F1)**: The harmonic mean of precision and recall, $F1 = 2 \times \frac{Pre \times Rec}{Pre+Rec}$. It is an accuracy measure that takes both precision and recall into consideration.

We analyze the performance of the proposed models based on accuracy and F1 score. Accuracy reflects upon how well the models generally perform in the prediction task whereas the F1 scores for each of the classes provide deeper insights into prediction bias.

4. Results and Discussion

4.1. Baseline Setup

The prediction accuracy of the dummy classifier (50%) is used as a baseline to determine which models have truly learnt to perform the classification task. In general, all classifiers outperformed the dummy classifier with nonlinear classifiers outperforming linear ones. The majority of classifiers performed well, achieving accuracy values ranging from 66% to 98%. Although TF linear classifier managed to achieve an accuracy of up to 66%, a closer examination of its precision and recall values indicate that the classifier makes highly biased predictions, i.e. predicting all samples as insecure as indicated by a zero F1 score for secure predictions. Although TF linear classifier and logistic regression are both based on the same machine learning algorithm, the difference in their data sampling methods lead to a significant difference in prediction results. As for nonlinear classifiers, decision tree and KNN have less biased predictions as compared to MLP, which is biased towards the insecure class.

Overall, the best performing models are logistic regression for linear classifiers, and KNN and decision tree for nonlinear classifiers. A summary of the results is shown in Table 2 for which the optimal hyperparameters are listed below:

- All linear classifiers:
  $Stopgap = 1000$
  $Epochs = 1000$

- MLP:
  $Stopgap = 1000$
  $Epochs = 1000$
  $HiddenLayers = 4$

- Decision Tree Classifier:
  $Split = random$
  $Leaf_N = unlimited$
  $SampleSplit = 2$

- KNN:
  $NN = 2$
  $Algo = kd\_tree$

$Leaf_S = 40$

Table 2: Baseline Setup Results

Model	F1 (Insecure)	F1 (Secure)	Accuracy
Dummy Classfier	0.50	0.50	0.50
TF Linear Classifier	0.78	0	0.66
Logistic Regression	0.89	0.66	0.84
Single-layer Perceptron	0.79	0.49	0.71
MLP	0.82	0.42	0.72
Decision Tree Classifier	0.97	0.85	0.93
KNN Classifier	0.94	0.93	0.92

4.2. Permutation Feature Representation

Table 3: Comparison results for permutation feature representation

Model	Perm	F1 (Insecure)	F1 (Secure)	Accuracy
KNN	rep_1	0.95	0.93	0.92
	rep_2	0.95	0.93	0.92
Decision Tree	rep_1	0.95	0.85	0.93
	rep_2	0.95	0.85	0.93
MLP	rep_1	0.82	0.42	0.72
	rep_2	0.91	0.72	0.87
Logistic Regression	rep_1	0.89	0.66	0.84
	rep_2	0.89	0.66	0.84

Experiments were conducted on the best linear classifier (logistic regression) and all nonlinear classifiers. The same set of optimal hyperparameter values described in Phase 1 were used. Results in Table 3 show that only MLP classifier has visible improvements when using rep_2 rather than rep_1. We conjecture that the use of rep_2 improves upon the performance MLP due to its sensitivity to feature scaling. rep_2 reduces the scale of the feature to a single integer in the range of [1,4] (although the number of features is increased), allowing MLP to converge faster and avoid being stuck in a local minimum. KNN, decision tree and logistic regression were able to achieve optimal performance regardless of how the permutations were presented. Based on this finding, Phase 3 uses rep_2 as it has the potential to improve the performance of certain classifiers without having an adverse effect on the rest.

4.3. Generalizability to Unseen Cipher Variants

The third phase is the most important one as it reflects upon the practicality of the proposed approach. First, we expect the classifiers to perform better when predicting *unseen* cipher variants that are insecure compared to secure ones. Second, we expect the classifiers to generally perform poorer at making security predictions on *unseen* cipher variants compared to cipher variants encountered in Phase 1. As expected, all classifiers do not perform as well as in the baseline experiments in Phase 1. Although linear classifiers seem to be as accurate as nonlinear classifiers, a closer inspection of the F1 scores indicate that the predictions made by linear classifiers are highly biased. In fact, all of the linear classifiers predict nearly every sample as insecure, showing that linear classifiers cannot generalize well to *unseen* block ciphers.

As for nonlinear classifiers, decision tree and KNN have the most unbiased results when predicting all *unseen* cipher variants but their performance is inversely proportionate to the cipher's security level. Both decision tree and KNN have similar accuracy for BC_1 (75% vs 77%) and BC_2 (66% vs 61%) but decision tree falters when it comes to BC_3 (46% vs 57%). Based on the F1 scores, the nonlinear classifiers are also biased towards predicting samples as insecure. This would also explain why prediction accuracy drops when the cipher's security level increases (BC_1 has 3 times more insecure than secure samples).

Overall, the best classifier for predicting the security of an *unseen* cipher variant is KNN. A summary of the results is shown in Table 4 for which all models use the same hyperparameter settings in Phase 1 except for the single-layer perceptron and decision tree. The optimal hyperparameters for the single-

Table 4: Generalization Results

Cipher	Model	F1 (Insecure)	F1 (Secure)	Accuracy
BC_1	TF Linear Classifier	0.86	0	0.76
	Logistic Regression	0.86	0	0.76
	Single-layer Perceptron	0.84	0.11	0.72
	MLP	0.86	0	0.76
	Decision Tree	0.85	0.46	0.75
	KNN	0.85	0.53	0.77
BC_2	TF Linear Classifier	0.75	0	0.61
	Logistic Regression	0.75	0	0.61
	Single-layer Perceptron	0.72	0.13	0.57
	MLP	0.75	0	0.61
	Decision Tree	0.77	0.36	0.66
	KNN	0.66	0.52	0.61
BC_3	TF Linear Classifier	0.44	0	0.28
	Logistic Regression	0.44	0	0.28
	Single-layer Perceptron	0.46	0.18	0.35
	MLP	0.44	0	0.28
	Decision Tree	0.52	0.39	0.46
	KNN	0.51	0.62	0.57

layer perceptron and decision tree classifiers in Phase 3 are as follows:

- Single-layer Perceptron:

  $Stopgap = 750$

  $Epochs = 750$

- Decision Tree Classifier:

  $Split = best$

  $Leaf_N = unlimited$

  $SampleSplit = 2$

4.4. Discussion, Practical Applications and Future Work

Overall, the experimental results proves the feasibility of the proposed approach whereby classifiers were able to learn the relationship between block cipher features and its security. More specifically, results show that nonlinear classifiers are better suited for assessing the security of block ciphers compared to linear classifiers. Linear classifiers such as logistic regression can still be used if security assessment is performed on *seen* block cipher variants but it cannot generalize well to *unseen* ones. In general, we recommend the use of nonlinear classifiers, specifically KNN as it was able to achieve a 92% prediction accuracy for *seen* cipher variants. KNN was still able to generalize to *unseen* cipher variants with an accuracy of 77%, 61% and 57% for BC_1, BC_2 and BC_3, respectively.

As the proposed approach can achieve a high accuracy (up to 92%) when predicting the security of *seen* cipher variants, it can be used to quickly identify good differential pairs for cryptanalysis. Although searching algorithms or mathematical solvers can also be used for this reason, determining the strength of each differential pair requires reasonable computational effort especially for large block sizes or number of rounds. In contrast, machine learning algorithms can perform this prediction near-instantaneously albeit with longer preprocessing time. Apart from that, high accuracy when predicting *seen* cipher variants implies that additional cipher features such as permutation pattern can potentially be used to improve existing machine learning-based distinguishers [15] for key recovery attacks.

The trained machine learning models can be used to quickly assess the security margin of new block cipher designs. This capability is showcased in Phase 3 when the classifiers were used to predict the security of *unseen* cipher variants. Although the best performing KNN classifier was only able to achieve prediction accuracy in the range of 57% to 77%, a closer inspection of F1 score indicates that it is biased towards predicting samples as insecure rather than secure. From another perspective, this means that KNN is more likely to classify a cipher as insecure, and will do so more accurately. This behaviour is more desirable than the inverse (higher likelihood to classify ciphers as secure) as it is essentially a stricter filter. This is useful for block cipher designers who wish to quickly discard poor designs without having to run computationally intensive searching algorithms or mathematical solvers.

The proposed work is not without its limitations. As of now, it remains to be seen if the same approach can be applicable to other block cipher structures such as SPN and ARX. For these structures, the use of truncated differentials may not be feasible as these ciphers may involve bitwise permutations. Thus, generating an exhaustive dataset for training will be more time consuming. Apart from that, the use of a single threshold value α is restrictive and may not accurately reflect the security requirements of different ciphers. With a more dynamic or flexible threshold, the performance of the models may be improved. The proposed approach sets a precedence for future work which includes:

- Exploring the use of deep learning to maximize the prediction accuracy for *unseen* cipher variants
- Investigating the use of (and different representations of) other features such as s-box probability or diffusion properties of the permutation pattern to further optimise prediction accuracy
- Prediction of differential probability or the number of active s-boxes using regression techniques
- Improving the accuracy of existing machine learning-based distinguishers using additional cipher features
- Training a machine learning algorithm to predict the security of a larger block cipher using data from smaller block ciphers with the same structure
- Predicting the security of other block cipher structures such as SPN or ARX

5. Conclusion

In this paper, we propose a different approach in applying machine learning for cryptanalysis. Rather than being used to directly cryptanalyze block ciphers to recover secret keys, we train machine learning classifiers using generic block cipher features to predict if a block cipher is secure or insecure based on the notion of differentially active s-boxes. Thus, the proposed approach is not specific to a particular block cipher nor secret key which is the case for the majority of existing methods. As a proof-of-concept, we perform experiments on a 4-branch GFS cipher which can be considered an ultralightweight 16-bit or 32-bit block cipher depending on the s-box size. By using truncated differentials, we are able to exhaustively generate the training and testing datasets by using a modified version of Matsui's branch-and-bound algorithm. We tested our approach by using 3 linear and 3 nonlinear classifiers. Experimental results concluded that nonlinear classifiers were better suited for the security prediction task, with KNN depicting optimal performance. When predicting *seen* cipher variants, KNN was able to achieve a prediction accuracy of up to 92% whereas when generalizing to *unseen* cipher variants, it achieved an accuracy of up to 77% depending on the security level of the targeted cipher. These results not only depict the feasibility of the proposed approach but also implies that the trained models can be used in practice to identify strong differential pairs for cryptanalysis and also to assess the security of new block cipher designs.

Acknowledgements

This work was supported in part by the Ministry of Education Malaysia under the Fundamental Research Grant Scheme (FRGS) no. FRGS/1/2019/ICT05/USM/02/1 and by the Uniten BOLD research grant under Grant No. 10463494/B/2019117.

References

[1] T. R. Lee, J. S. Teh, J. L. S. Yan, N. Jamil, W.-Z. Yeoh, A machine learning approach to predicting block cipher security, in: Proceedings of

the 7th International Cryptology and Information Security Conference 2020 (CRYPTOLOGY), UPM, 2020.

[2] P. Li, S. Zhou, B. Ren, S. Tang, T. Li, C. Xu, J. Chen, Efficient implementation of lightweight block ciphers on volta and pascal architecture, in: Journal of Information Security and Applications, Vol. 47, Elsevier BV, 2019, pp. 235–245. doi:10.1016/j.jisa.2019.04.006.

[3] P. Dey, R. S. Rohit, A. Adhikari, Single key MITM attack and biclique cryptanalysis of full round Khudra, in: Journal of Information Security and Applications, Vol. 41, Elsevier BV, 2018, pp. 117–123. doi:10.1016/j.jisa.2018.06.005.

[4] J. Chen, J. Teh, Z. Liu, C. Su, A. Samsudin, Y. Xiang, Towards accurate statistical analysis of security margins: New searching strategies for differential attacks, IEEE Transactions on Computers 66 (10) (2017) 1763–1777. doi:10.1109/tc.2017.2699190.

[5] R. Ankele, S. Kölbl, Mind the gap - a closer look at the security of block ciphers against differential cryptanalysis, in: Selected Areas in Cryptography – SAC 2018, Springer International Publishing, 2019, pp. 163–190. doi:10.1007/978-3-030-10970-7_8.

[6] K. Alallayah, M. Amin, W. AbdElwahed, A. Alhamamii, Applying neural networks for simplified data encryption standard (SDES) cipher system cryptanalysis, in: The International Arab Journal of Information Technology, 2012, pp. 163–169.

[7] M. M. Alani, Neuro-cryptanalysis of DES and triple-DES, in: Neural Information Processing, Springer Berlin Heidelberg, 2012, pp. 637–646. doi:10.1007/978-3-642-34500-5_75.

[8] A. Jain, G. Mishra, Analysis of lightweight block cipher FeW on the basis of neural network, in: Harmony Search and Nature Inspired Op-

timization Algorithms, Springer Singapore, 2018, pp. 1041–1047. doi:10.1007/978-981-13-0761-4_97.

[9] G. Mishra, S. V. S. S. N. V. G. K. Murthy, S. K. Pal, Neural network based analysis of lightweight block cipher PRESENT, in: Harmony Search and Nature Inspired Optimization Algorithms, Springer Singapore, 2018, pp. 969–978. doi:10.1007/978-981-13-0761-4_91.

[10] R. Focardi, F. L. Luccio, Neural cryptanalysis of classical ciphers, in: ICTCS, 2018.

[11] A. N. Gomez, S. Huang, I. Zhang, B. M. Li, M. Osama, L. Kaiser, Unsupervised cipher cracking using discrete gansarXiv:http://arxiv.org/abs/1801.04883v1.

[12] C. Tan, Q. Ji, An approach to identifying cryptographic algorithm from ciphertext, in: 2016 8th IEEE International Conference on Communication Software and Networks (ICCSN), IEEE, 2016. doi:10.1109/iccsn.2016.7586649.

[13] R. Alshammari, A. N. Zincir-Heywood, Machine learning based encrypted traffic classification: Identifying SSH and skype, in: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, IEEE, 2009. doi:10.1109/cisda.2009.5356534.

[14] A. Albassal, A.-M. Wahdan, Neural network based cryptanalysis of a feistel type block cipher, in: International Conference on Electrical, Electronic and Computer Engineering, 2004. ICEEC '04., IEEE. doi:10.1109/iceec.2004.1374430.

[15] A. Gohr, Improving attacks on round-reduced speck32/64 using deep learning, in: Advances in Cryptology – CRYPTO 2019, Springer International Publishing, 2019, pp. 150–179. doi:10.1007/978-3-030-26951-7_6.

[16] L. R. Knudsen, Truncated and higher order differentials, in: Fast Software Encryption, Springer Berlin Heidelberg, 1995, pp. 196–211. doi:10.1007/3-540-60590-8_16.