Assessing Lightweight Block Cipher Security using Linear and Nonlinear Machine Learning Classifiers

Ting Rong Lee1, Je Sen Teh1, Jasy Liew Suet Yan1, Norziana Jamil2, Jiageng Chen3

1School of Computer Sciences, Universiti Sains Malaysia
2Department of Computing, College of Computing and Informatics, Universiti Tenaga Nasional
3School of Computer, Central China Normal University

Received: date / Accepted: date

Abstract In this paper, we investigate the use of machine learning classifiers to assess block cipher security from the perspective of differential cryptanalysis. These classifiers were trained using common block cipher features (number of rounds, permutation pattern, truncated input and output differences), making our approach generalizable to an entire class of ciphers. Each data sample represents a truncated differential path, for which the level of security is labelled as secure or insecure by the trained classifier based on the number of differentially active S-boxes. We trained six machine learning classifiers (linear and nonlinear) to perform the security prediction task using a dataset generated from a small-scale generalized Feistel structure (GFS) cipher as a proof-of-concept. Prediction accuracy was further refined by determining the best way to represent features in the dataset during training. We then studied how well these classifiers perform the prediction tasks on ciphers that they were trained on (seen) and those that they were not (unseen). When applied on seen ciphers, the classifiers achieved prediction accuracy results of up to 93% whereas for unseen cipher variants, accuracy results of up to 71% were obtained. Our findings indicate that nonlinear classifiers are better suited for the prediction task. Next, we applied the proposed approach to a full-scale lightweight GFS block cipher, TWINE. By training the best performing nonlinear classifiers (k-nearest neighbour and decision tree classifiers) using data from five other GFS ciphers, we obtained an accuracy of up to 74% when labelling data from TWINE. In addition, the trained classifiers could generalize to a larger number of rounds of TWINE despite being trained using data obtained from fewer rounds. These findings showcase the feasibility of using machine learning classifiers, notably nonlinear variants, as a tool to assess block cipher security.

Keywords Block cipher · Classifier · Cryptanalysis · Encryption · Machine Learning · Security

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 [1,2]. 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 [3] or mathematical solvers [4] 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 [9], 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 [6], 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 [7, 8] 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, [9] trained a neural network to extract the encryption keys of Caesar, Vignere polyalphabetic and substitution ciphers. Generative adversarial networks were also used to crack these classical ciphers in [10].

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 [11]. 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 [12]. They found that the C4.5 decision tree-based classification algorithm performed the best, achieving a detection rate of up to 97.2%.

In [13], 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. [14] introduced a recurrent neural network-based approach for identifying differentials for Serpent by modelling the search as a multi-level weighted directed graph. [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, most machine learning approaches have been cipher-specific rather than generalizable. A cipher-specific approach is one that would require the entire training process to be repeated if a different cipher needs to be analyzed. To overcome this problem, we propose a generalizable approach to assess a block cipher’s resistance against differential cryptanalysis using machine learning. 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 were 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 [3]. Apart from determining the most suitable machine learning classifier and hyperparameters for the security prediction task, we also look into how data representation can affect prediction accuracy. Preliminary experiments were performed on 4-branch GFS ciphers to showcase the generalizability of the proposed approach to an entire class of block ciphers, rather than a specific one. An in-depth comparison of six classifiers (linear and nonlinear) was performed.

Our findings show that nonlinear classifiers outperform linear classifiers due to the nonlinear transforms involved in block ciphers, achieving a prediction accuracy of up to 93% when predicting seen cipher variants and up to 71% when predicting unseen cipher variants. We then apply the best-performing classifiers to predict or label data from full-scale (16-branch) lightweight GFS ciphers. We train two nonlinear classifiers (k-nearest neighbour and decision tree) using data from five other GFS ciphers. When labelling data samples from ciphers that the models have seen before, they were able to achieve an accuracy of up to 97%. When assessing the lightweight cipher, TWINE which was not seen during training, the best performing classifier achieved an accuracy of up to 93%.

1Supplementary code for this paper is available at https://github.com/trlee/ml-block-cipher
accuracy of up to 74%. The classifiers were also able to accurately label data obtained from the 9th round of TWINE despite being trained with data from round 1-8 of the five GFS ciphers. This indicates that the trained classifier was able to generalize to a larger number of rounds than it has been trained for.

The rest of this paper is structured as follows: Section 2 introduces preliminary information required to understand the proposed work. Sections 3 and 4 then provide the detailed steps, experimental setup and results for the small-scale (4-branch) and full-scale (16-branch) GFS experiments respectively. Section 5 provides a discussion of our findings and their significance. The paper is concluded in Section 6 which includes some future directions of this work.

2 Preliminaries

2.1 Differential Cryptanalysis and Active S-boxes

Differential cryptanalysis observes the propagation of an XOR difference of a pair of plaintexts (input difference) through a cipher to produce a corresponding pair of ciphertexts with a specific XOR difference (output difference). We define an input difference as

\[ \Delta X = X' \oplus X'' \]  
\[ \Delta X = [\Delta X_0, \Delta X_1, ..., \Delta X_{i-1}] \]  

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 \( \Delta X \), the probability of any particular \( \Delta Y \) occurring will be exactly \( \frac{1}{2^b} \) where \( b \) is the block size. A successful differential attack requires 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 with 4-bit S-box.

![Fig. 1 1 round of a 4-branch GFS with 4-bit S-box](image)

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 \( B_n \). \( B_n \) is defined as the best probability the running algorithm has found so far. An initial value has to be set for
classes. A machine learning model refers to a trained
ear or nonlinear) to segregate the secure and insecure
classifiers learn the best hypothesis function (i.e. lin-
arity of block ciphers. Essentially, the goal is to have the
ear and nonlinear classifiers when predicting the secu-

In the proposed work, we use a variant of Mat-
sui’s algorithm as described in [3]. We further simplify
the algorithm as we only need the number of differen-
tially active S-boxes rather than the concrete differen-
tial probability for our experiments. This greatly in-
creases the speed of the search, which allows us to re-
move all bounding restrictions to generate large datasets
for training and testing purposes. The dataset gener-
ated for the current study can be reproduced using the
algorithm available at [https://github.com/jesenteh
16b-gfs-as-search](https://github.com/jesenteh
16b-gfs-as-search).

2.3 Generalized Feistel Structure

GFS is the generalization of the Feistel structure that
was first used in the block cipher Lucifer, the prede-
cessor to DES. It divides an input into \( d \) blocks, where
\( d > 2 \). As a proof-of-concept, our proposed work is ap-
plied to a 4-branch GFS cipher (\( d = 4 \)), similar to the
one in Figure 1. We then extend our work to full-scale
16-branch (\( d = 16 \)) GFS ciphers. By using a GFS cipher
with a word-based permutation, we can use truncated
differences in our experiments. A 4-branch GFS effec-
tively represents ultralightweight block ciphers with 16-
or 32-bit blocks depending on whether 4-bit or 8-bit S-
boxes are used whereas a 16-branch GFS can represent
a lightweight 64-bit block cipher such as TWINE or a
128-bit block cipher. Regardless of which, security anal-
ysis based on the number of active S-boxes is usually
performed based on the highest differential probability
for a given S-box. For example, TWINE and AES S-
boxes have the best differential probabilities of \( 2^{-2} \) and
\( 2^{-6} \) respectively.

2.4 Machine Learning Classifiers and the Security
Prediction Task

The proposed work investigates the performance of lin-
ear and nonlinear classifiers when predicting the secu-
rity of block ciphers. Essentially, the goal is to have the
classifiers learn the best hypothesis function (i.e. lin-
ear or nonlinear) to segregate the secure and insecure
classes. A machine learning model refers to a trained
classifier with specific features, machine learning algo-
rithm and hyperparameters.

Three linear and three nonlinear classifiers were used
in our experiments. The three linear classifiers are the
Tensorflow (TF) Linear classifier, and scikit-learn’s lo-
gistic regression and single-layer perceptron, whereas
the three nonlinear classifiers include scikit-learn’s k-
nearest neighbors (KNN), decision tree, and multi-layer
perceptron (MLP). To achieve optimal prediction ac-
curacy, the aforementioned linear classifiers are tuned
with respect to the following hyperparameters:

- **Stopgap**: The total number of iterations that the
  model needs to undergo with no improvements be-
  fore stopping the training process early.
- **Epochs**: The total number of passes the model has
to undergo throughout the training data batches.

Each of the nonlinear classifiers have different hyper-
parameters that need to be tuned separately. KNN’s
hyperparameters that can be tuned to optimize perfor-
ance include:

- **NN**: NN refers to the number of neighbors to be
  used for the k-neighbors query.
- **Distance**: This is measure used to determine the dis-
tance between two neighbors. The default Minkowski
distance is used for all experiments.
- **Algo**: Algorithm used to compute the nearest neigh-
bors for the model. Three options include KDTree,
BallTree or brute force.
- **LeafS**: Leaf size passed to the KDTree or BallTree,
which can affect the speed of the tree construction
and query, as well as memory required.

The decision tree classifier’s parameters include:

- **Split**: The strategy used to choose the split on each
  node, which can be either best or random.
- **LeafS**: Maximum number of leaf nodes.
- **Sample Split**: Minimum amount of samples required
to split an internal node.

In addition to Stopgap and Epoch, MLP’s hyperpara-
eters are as follows:

- **Activation**: The function that determines the out-
puts of the nodes. The default rectified linear func-
tion is used for all experiments.
- **Hidden layers**: 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 4-branch GFS Experiments

#### 3.1 Experimental Setup

All experiments were performed on computer with an Intel i5 2.4GHz CPU and 16GB RAM using Python 3.6.7, scikit-learn 0.22.2 and TensorFlow 2.2. 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, scikit-learn’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, we perform hyperparameter tuning for each classifier. We also investigate the effect of data representation on prediction accuracy, specifically how the permutation patterns are represented.

To investigate the feasibility of the proposed approach, we first perform preliminary experiments on smaller-scale, 4-branch GFS ciphers before proceeding to their 16-branch counterparts. This allows us to generate a large amount of training/testing data within a practical amount of time for all possible permutation patterns. Each sample in the dataset used to train the machine learning classifiers consists 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 ciphers, 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 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 a 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, $\alpha$ 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, $\alpha$ dictates the minimum number of active S-boxes per round for a block cipher to be considered secure. $\alpha$ can be configured based on the desired security margin that the cryptanalyst or designer requires. We want to ensure that $\alpha$ 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 actual values of $AS$ are not 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 all examples are randomly sampled from an exhaustive dataset. A single integer is used to represent the entire 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 where $rep_1$ is compared with $rep_2$ which represents the permutation as 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 with the permutation feature transformed into $rep_2$. Figure 2 depicts the experimental flow for both Phase 1 and Phase 2.

The third phase depicted in Figure 3 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

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$P$</th>
<th>$r$</th>
<th>$AS$</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>1010</td>
<td>0123</td>
<td>8</td>
<td>16</td>
<td>Secure</td>
</tr>
<tr>
<td>0111</td>
<td>1101</td>
<td>1203</td>
<td>11</td>
<td>17</td>
<td>Secure</td>
</tr>
<tr>
<td>1111</td>
<td>0100</td>
<td>3021</td>
<td>12</td>
<td>9</td>
<td>Secure</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
<td>0123</td>
<td>5</td>
<td>5</td>
<td>Insecure</td>
</tr>
<tr>
<td>1111</td>
<td>0101</td>
<td>3021</td>
<td>12</td>
<td>6</td>
<td>Insecure</td>
</tr>
<tr>
<td>1101</td>
<td>1100</td>
<td>3120</td>
<td>11</td>
<td>6</td>
<td>Insecure</td>
</tr>
</tbody>
</table>
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' performance on three different unseen block ciphers denoted 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 fulfill these ratios, four block cipher variants (or equivalently, four permutation patterns) 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. A summary of the three main phases are 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, \( F_1 = 2 \times \frac{\text{Pre} \times \text{Rec}}{\text{Pre} + \text{Rec}} \). 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.

### 3.2 Experimental Results

#### 3.2.1 Baseline Results

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 69% to 93%. TF linear classifier underperformed (56% accuracy) with a distinct bias towards predicting samples as insecure. Although TF linear classifier and logistic regression are both based on the same machine learning algorithm, the difference in their data sampling methods leads 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:

- **TF Linear Classifier:**
  
  \[
  \text{Stopgap} = 350 \\
  \text{Epochs} = 750 \\
  \]

- **Other linear classifiers:**

  \[
  \text{Stopgap} = 1000 \\
  \text{Epochs} = 1000 \\
  \]

- **MLP:**

  \[
  \text{Stopgap} = 1000 \\
  \text{Epochs} = 1000 \\
  \text{Hidden Layers} = 4 \\
  \text{Neurons per hidden layer} = 100 \\
  \]

- **Decision Tree Classifier:**

  \[
  \text{Split} = \text{random} \\
  \]

#### Table 2 Baseline Setup Results (0-insecure, 1-secure)

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 (0)</th>
<th>F1 (1)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dummy Classifier</td>
<td>0.50</td>
<td>0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>TF Linear Classifier</td>
<td>0.71</td>
<td>0.15</td>
<td>0.56</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>0.66</td>
<td>0.72</td>
<td>0.69</td>
</tr>
<tr>
<td>Single-layer Perceptron</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>MLP</td>
<td>0.73</td>
<td>0.75</td>
<td>0.74</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.95</td>
<td>0.85</td>
<td>0.93</td>
</tr>
<tr>
<td>KNN</td>
<td>0.95</td>
<td>0.83</td>
<td>0.92</td>
</tr>
</tbody>
</table>

#### 3.2.2 Permutation Feature Representation

To study the impact of feature representation on prediction accuracy, we perform experiments on the best linear classifier (single-layer perceptron) 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 of 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 and decision tree were able to achieve optimal performance regardless of how the permutations were presented, while single-layer perceptron saw a slight improvement. Based on these results, Phase 3 will rely on \( rep_2 \) as it has the potential to improve the performance of certain classifiers without having an adverse effect on the rest.

#### Table 3 Comparison results for permutation feature representation (0-insecure, 1-secure)

<table>
<thead>
<tr>
<th>Model</th>
<th>Perm</th>
<th>F1 (0)</th>
<th>F1 (1)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single-layer Perceptron</td>
<td>( rep_1 )</td>
<td>0.71</td>
<td>0.71</td>
<td>0.71</td>
</tr>
<tr>
<td>MLP</td>
<td>( rep_1 )</td>
<td>0.73</td>
<td>0.75</td>
<td>0.74</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>( rep_1 )</td>
<td>0.95</td>
<td>0.85</td>
<td>0.93</td>
</tr>
<tr>
<td>KNN</td>
<td>( rep_1 )</td>
<td>0.95</td>
<td>0.83</td>
<td>0.92</td>
</tr>
</tbody>
</table>
### 3.2.3 Generalizability to Unseen Cipher Variants

The third phase is the most important one as it reflects upon the practicality of the proposed approach. We expect the classifiers to perform better when predicting unseen cipher variants that are insecure compared to secure ones. We also expect the classifiers to generally perform poorer at making security predictions on unseen cipher variants as compared to the ones that they have. 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. Generally, KNN outperforms decision tree in all scenarios: 71% vs 69% for BC1, 62% vs 58% for BC2, and 56% vs 51% for BC3. We can conclude that 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 as in Phase 1, except for decision tree classifier (\(\text{Split} = \text{best}, \text{Leaf}_N = \text{unlimited}, \text{SampleSplit} = 2\)).

<table>
<thead>
<tr>
<th>Cipher</th>
<th>Model</th>
<th>F1 (0)</th>
<th>F1 (1)</th>
<th>Accu.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TF Linear Classifier</td>
<td>0.86</td>
<td>0</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>Logistic Regression</td>
<td>0.86</td>
<td>0</td>
<td>0.76</td>
</tr>
<tr>
<td></td>
<td>Single-layer Perceptron</td>
<td>0.77</td>
<td>0.25</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.83</td>
<td>0.20</td>
<td>0.71</td>
</tr>
<tr>
<td></td>
<td>Decision Tree</td>
<td>0.69</td>
<td>0.64</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.82</td>
<td>0.26</td>
<td>0.71</td>
</tr>
<tr>
<td>BC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TF Linear Classifier</td>
<td>0.68</td>
<td>0</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>Logistic Regression</td>
<td>0.68</td>
<td>0</td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>Single-layer Perceptron</td>
<td>0.73</td>
<td>0.18</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.69</td>
<td>0.16</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>Decision Tree</td>
<td>0.77</td>
<td>0.36</td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.66</td>
<td>0.52</td>
<td>0.62</td>
</tr>
<tr>
<td>BC2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>TF Linear Classifier</td>
<td>0.44</td>
<td>0</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>Logistic Regression</td>
<td>0.44</td>
<td>0</td>
<td>0.28</td>
</tr>
<tr>
<td></td>
<td>Single-layer Perceptron</td>
<td>0.29</td>
<td>0.43</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>MLP</td>
<td>0.43</td>
<td>0.54</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>Decision Tree</td>
<td>0.46</td>
<td>0.48</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>KNN</td>
<td>0.51</td>
<td>0.62</td>
<td>0.56</td>
</tr>
<tr>
<td>BC3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 4.1 Experimental Setup

All experiments were performed on the same computer with an Intel i5 2.4GHz CPU and 16GB RAM using Python 3.6.7, scikit-learn 0.22.2 and TensorFlow 2.2. The computational time required to generate sufficient training data for 16-branch GFS ciphers is exponentially higher than that of 4-branch ciphers. It is also not practical to generate data for every possible permutation pattern (\(16! \approx 2 \times 10^{13}\) possibilities). Thus, we have selected six 16-branch GFS ciphers for our experiments. Apart from TWINE itself, which is the target cipher for generalization experiments, five others were selected based on permutation patterns with optimal cryptographic properties (full diffusion in 8 rounds and a minimum of 40 AS after 20 rounds). The six permutation patterns for the chosen GFS ciphers are shown in Table 5 with naming conventions for the permutations taken from [17]. The same modified branch-and-bound search is used to generate data samples. Due to their underlying permutation patterns, these ciphers already achieve full diffusion in 8 rounds. Thus, we limit the number of rounds to 8 to ensure that data can be generated in an exhaustive manner within a practical amount of time (approximately 1-2 days for 8 rounds). Generating the data in an exhaustive manner allows us to perform random sampling without imposing any limits to the inputs nor bounding criteria for the branch-and-bound search. For each cipher, we generate 100000 samples, whereby 12500 samples are taken from each round of the cipher. To determine if the machine learning models are able to generalize to more rounds than they have been trained with, we generate an additional 100000 samples from the 9th round of TWINE. In total, we have three datasets that can be summarized as follows:

- \(\text{GFS}_{(1,8)}\) - 500000 samples from round 1-8 of five GFS ciphers (excluding TWINE)
- \(\text{TW}_{(1,8)}\) - 100000 samples from round 1-8 of TWINE
- \(\text{TW}_9\) - 100000 samples from round 9 of TWINE

<table>
<thead>
<tr>
<th>Name</th>
<th>Permutation Pattern, (P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. 5</td>
<td>5,2,9,4,11,6,15,8,3,12,1,10,7,0,13,14</td>
</tr>
<tr>
<td>No. 7</td>
<td>1,2,11,4,3,6,7,8,15,12,5,14,9,0,13,10</td>
</tr>
<tr>
<td>No. 9</td>
<td>1,2,11,4,9,6,15,8,5,12,7,14,3,0,9,13,10</td>
</tr>
<tr>
<td>No. 10</td>
<td>7,2,13,4,11,8,3,6,15,9,9,10,1,14,5,12</td>
</tr>
<tr>
<td>No. 12</td>
<td>1,2,11,4,15,8,3,6,7,9,12,5,14,13,10</td>
</tr>
<tr>
<td>TWINE</td>
<td>5,0,1,4,7,12,3,8,13,6,9,2,15,10,11,14</td>
</tr>
</tbody>
</table>

| Table 5 16-branch permutation patterns |
Fig. 4 Experiment 2 - Phase 1 flow

The format of each data sample is similar to Table 1 but the input and output truncated differences as well as the permutation are 16 words rather than 4. In terms of feature representation, we found that using $rep_2$ for both permutation pattern and truncated differences led to better results. As the maximum number of $AS$ per round for a 16-branch GFS is 8, the security margin threshold is set to half, $\alpha = 4$. For our experiments, we chose the KNN and decision tree classifiers as they were the two best performing models based on our findings in Section 3. The experiments are divided into two main phases:

- **Phase 1 - Baseline Setup** - The goal of this phase (depicted in Figure 4) is to determine if machine learning classifiers are able to perform security predictions for seen 16-branch block ciphers. The $GFS_{1,8}$ dataset is used, to which an 80:20 train-test split is performed (400000 training samples, 100000 test samples). Hyperparameter tuning is performed to obtain the best performing models.

- **Phase 2 - Generalizability to TWINE** - The goal of this phase (depicted in Figure 5) is to determine if machine learning models can be used for security prediction for an actual unseen lightweight cipher, TWINE after being trained using data from the five other GFS ciphers. The $GFS_{1,8}$ dataset is used for training whereas $TW_{1,8}$ dataset is used for testing. We also compare the performance of the classifiers when performing prediction for more rounds than they have been trained for. This is performed by training the models using the $GFS_{1,8}$ dataset and using the $TW_3$ dataset for testing. Hyperparameter tuning is performed again to obtain the best performing models.

We analyze the performance of the proposed models using the same accuracy and F1 metrics as in Section 3. Additionally, we also compare the models based on the area under the receiver operating characteristic (AUROC). As the $TW_3$ dataset is highly imbalanced (more secure samples as compared to insecure samples), AUROC will provide better performance insights.

4.2 Experimental Results

4.2.1 Baseline Results

The best performing decision tree and KNN models achieved an accuracy of 97% and 96% respectively. They were able to perform predictions with minimal biases for both the secure and insecure classes as shown in Table 6. These results also indicate that the machine learning models better at security prediction for 16-branch GFS ciphers as compared to 4-branch ciphers. This can be attributed to the larger number of features involved during training, 49 features (Input difference - 16, Output difference - 16, Permutation - 16, Number of Rounds - 1) features as compared to 7 features (Input difference - 1, Output difference - 1, Permutation - 1, Number of Rounds - 1). Although decision tree slightly outperforms KNN in terms of accuracy, KNN is more accurate in predicting the secure class as depicted in the ROC curve shown in Figure 6. The optimal hyperparameters for both models are listed below:

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

- KNN:

  $NN = 2$
  
  $Algo = KDTree$
  
  $Leaf_S = 100$

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 (0)</th>
<th>F1 (1)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>0.97</td>
<td>0.97</td>
<td>0.97</td>
</tr>
<tr>
<td>KNN</td>
<td>0.97</td>
<td>0.96</td>
<td>0.96</td>
</tr>
</tbody>
</table>

Table 6 Baseline Setup Results for 16-branch GFS (0-insecure, 1-secure)
4.2.2 Generalizability to TWINE

This phase is an important one as it reflects upon the feasibility of the proposed approach to be used in actual cryptanalytic settings. Naturally, we expect the nonlinear classifiers to make more accurate predictions for the five GFS ciphers that they have already seen as compared to TWINE. The results in Table 7 confirm this notion as both decision tree and KNN did not perform as well as in the baseline experiments when labelling data from TWINE. However, both models were still able to generalize well to TWINE, with KNN and decision tree achieving accuracy scores of 74% and 67% respectively. The ROC curve in Figure 7 clearly depicts that KNN discriminates between secure and insecure classes better than decision tree. The prediction results for TWINE in terms of both accuracy and bias were also better than the generalization results for the unseen 4-branch ciphers, BC1, BC2 and BC3.

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 (0)</th>
<th>F1 (1)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>0.72</td>
<td>0.60</td>
<td>0.67</td>
</tr>
<tr>
<td>KNN</td>
<td>0.79</td>
<td>0.68</td>
<td>0.74</td>
</tr>
</tbody>
</table>

GFS ciphers with strong permutation patterns such as TWINE will achieve full diffusion after 8 rounds. Thus, a dataset generated entirely from the 9th round will consist of mostly secure samples. This is the case for the TW9 dataset, which has 99918 secure samples but only 82 insecure samples. Due to the highly imbalanced nature of this dataset, a comparison of accuracy scores shown in Table 8 cannot reliably depict performance. However, the AUROC scores still indicate that KNN greatly outperforms decision tree (0.818 vs 0.659) in terms of correctly predicting the secure class. The ROC curve for the 9-round TWINE generalization experiment is shown in Figure 8. We can conclude that the best classifier for predicting the security of an unseen 16-branch cipher is KNN, even for a larger number of rounds than it has been trained for. These results were obtained after a second round of hyperparameter tuning which resulted in the same hyperparameter values for decision tree but different values for KNN (NN = 8, Algo = KDTree, Leafs = 250).

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 (0)</th>
<th>F1 (1)</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Tree</td>
<td>0.01</td>
<td>0.88</td>
<td>0.78</td>
</tr>
<tr>
<td>KNN</td>
<td>0.02</td>
<td>0.97</td>
<td>0.94</td>
</tr>
</tbody>
</table>
5 Discussion, Practical Applications and Future Work

Overall, the experimental results showcased the feasibility of the proposed approach whereby classifiers were able to learn the relationship between block cipher features and security (with respect to differential cryptanalysis). More specifically, results showed that nonlinear classifiers are better suited for assessing the security of block ciphers as 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 71%, 62% and 56% for BC1, BC2 and BC3, respectively.

Contrary to intuition, the trained models (specifically decision tree and KNN) actually performed better when applied to 16-branch GFS ciphers. We conjecture that this is a result of the increased number of features being used for training (7x more features as compared to the 4-branch ciphers). Investigating the impact of specific features and the number of features will be left to future work. Our findings indicate that the prior recommendation of using KNN for the prediction task still holds valid. KNN was able to achieve 96% accuracy when performing predictions for the five seen GFS ciphers, and could generalize well to the unseen GFS cipher, TWINE with an accuracy of 74%. As compared to decision tree, KNN can better discriminate between secure and insecure classes based on its higher AUROC scores. In addition, KNN was able to make accurate predictions for 9 rounds of TWINE despite being trained with only round 1-8 data from the five GFS ciphers. This depicts the capability of KNN to generalize to more rounds than it has been trained for.

As the proposed approach can achieve high accuracy (up to 96%) 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 a longer training time. This is an efficiency trade-off between the online phase of an attack and its pre-processing phase. 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 for key recovery attacks.

The trained machine learning models can be used to quickly assess the security margin of unseen block ciphers. In practice, these unseen block ciphers can be new designs or any other block cipher that the model was not trained with. This capability was depicted when the trained nonlinear classifiers were used on TWINE. The best performing KNN classifier achieved a prediction accuracy of 74%. A closer inspection of the F1 scores indicates that KNN is more likely to classify a cipher as insecure (F1 = 0.79) rather than secure (F1 = 0.68), and will do so more accurately. This implies that the predictions made by the classifier are more conservative (favouring insecure rather than secure), which is desirable in a practical setting. Its high AUROC scores (0.818-0.934) shows that it is also proficient at classifying secure samples correctly. These results support the reliability of the proposed model’s predictions. The trained models will be 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 applied on 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 the use of deep learning to maximize prediction accuracy, the use of other cryptographic features, the use of regression techniques to predict the actual differential probability, improvement of existing machine learning-based distinguishers using the findings from this work, and prediction for other block cipher structures.

6 Conclusion

In this paper, we proposed an alternative 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
cylinder 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 performed experiments on 4-branch GFS ciphers. By using truncated differentials, we were 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 three linear and three nonlinear classifiers. Experimental results concluded that nonlinear classifiers were better suited for the security prediction task, with decision tree and KNN depicting optimal performance. When predicting seen cipher variants, the decision tree classifier was able to achieve a prediction accuracy of up to 93% as compared to 92% for KNN. KNN outperformed decision tree when generalizing to unseen cipher variants, achieving an accuracy of up to 71% depending on the security level of the targeted cipher. We then applied the proposed approach on 16-branch GFS ciphers, including the lightweight block cipher, TWINE. We found that the decision tree and KNN classifiers were highly adept at making predictions for seen ciphers, achieving accuracy results ranging between 96-97%. When generalizing to an unseen block cipher (TWINE), KNN not only outperformed decision tree (74% versus 67%), there were also minimal biases as compared to predictions made for the smaller-scale ciphers. KNN could also make accurate predictions (accuracy of 94%, AUROC score of 0.934) for 9-round TWINE despite being trained using data obtained from only round 1-8 of the five GFS ciphers. These results not only depict the feasibility of the proposed approach but also implies that the trained models can be used in practice to filter 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 and by the Unite BOLD research grant under Grant No. 10463494/B/2019117.

7 Conflict of Interest

The authors declare that they have no conflict of interest.

References