Theoretical Explanation and Improvement of Deep Learning-aided Cryptanalysis

Weixi Zheng[a,b,*], Liu Zhang[a,b,*], Zilong Wang[a,b,**]

[a] *School of Cyber Engineering, Xidian University, Xi'an, China*
[b] *State Key Laboratory of Cryptology, P.O.Box 5159, Beijing 100878, China*

Abstract

At CRYPTO 2019, Gohr demonstrated that differential-neural distinguishers ($\mathcal{DND}$s) for Speck32/64 can learn more features than classical cryptanalysis's differential distribution tables (DDT). Furthermore, a non-classical key recovery procedure is devised by combining the Upper Confidence Bound (UCB) strategy and the BayesianKeySearch algorithm. Consequently, the time complexity of 11-round key recovery attacks on Speck32/64 is significantly reduced compared with the state-of-the-art results in classical cryptanalysis. This advancement in deep learning-assisted cryptanalysis has opened up new possibilities. However, the specific encryption features exploited by $\mathcal{DND}s$ remain unclear.

In this paper, we begin by analyzing the features learned by $\mathcal{DND}$ based on the probability distribution of a ciphertext pair. Our analysis reveals that $\mathcal{DND}$ not only learns the differential features of the ciphertext pair but also captures the XOR information of the left and right branches of the ciphertext pair. This explains why the performance of $\mathcal{DND}$ can outperform DDT in certain cases. For other ciphers, we can also predict whether deep learning methods can achieve superior results to classical methods based on the probability distribution of the ciphertext pair. Next, we modify the input data format and network structure based on the specific features that can be learned to train $\mathcal{DND}$ specifically. With these modifications, it is possible to reduce the size of their parameters to only $1/16$ of their previous networks while maintaining high precision. Additionally, the training time for the $\mathcal{DND}$s is significantly reduced. Finally, to improve the efficiency of deep learning-assisted cryptanalysis, we introduce Bayes-UCB to select promising ciphertext structures more efficiently. We also introduce an improved BayesianKeySearch algorithm to retain guessed keys with the highest scores in key guessing. We use both methods to launch 11-round, 12-round, and 13-round key recovery attacks on Speck32/64. The results show that under the same conditions, the success rate of 11-round key recovery attacks has increased from Gohr's 36.1% to 52.8%, the success rate of 12-round key recovery attacks has increased from Gohr's 39%

*These authors contributed to the work equally and should be regarded as co-first authors
**Corresponding author
Email address: zlwang@xidian.edu.cn (Zilong Wang)

to 50%, and the success rate of 13-round key recovery attacks has increased from Zhang *et al.*'s 21% to 24%. In addition, the time complexity of these experiments is also significantly reduced.

Keywords: Probability Distribution, Neural Network, Parameter Amount, Bayes-UCB, Key Recovery Attack

1. Introduction

In CRYPTO 2019, Gohr [1] proposed the idea of differential-neural cryptanalysis. The differential-neural distinguisher ($\mathcal{DND}$), trained by the neural network, is introduced as the underlying distinguisher to distinguish whether ciphertexts are encrypted by plaintexts that satisfy a specific input difference or by random numbers. However, the current differential-neural distinguisher seems only effective for limited rounds of ciphertexts. Therefore, a short high-probability classical differential ($\mathcal{CD}$) is prepended before the differential-neural distinguisher to increase the number of rounds for key recovery attacks. In the process, BayesianKeySearch algorithm, Wrong Key Response Profile (WKRP), and UCB speed up key recovery attacks with higher efficiency compared to classical differential cryptanalysis.

From the perspective of the components used in differential-neural cryptanalysis, there are two ways to improve the performance of differential-neural cryptana-lysis. One is to increase the number of rounds or accuracy of $\mathcal{DND}$. There has been a lot of work to improve the performance of $\mathcal{DND}$ by modifying the structure of the neural network, changing the format of input data and hyperparameter optimization [2],[3],[4],[5],[6],[7]. In addition, Bacuieti *et al.* [8] used the lottery ticket hypothesis to prune the $\mathcal{DND}$ by removing parameters that have little influence on the results. This reduced the size of the neural network and constructed a smaller, more efficient $\mathcal{DND}$. The other is to increase the length of $\mathcal{CD}$. Bao *et al.* [7] generalized the concept of neutral bits and searched for (conditional) simultaneous neutral bit-set with a higher probability for more rounds of the $\mathcal{CD}$. Thus, Bao *et al.* improved the rounds of key recovery attacks for Speck32/64 by increasing the length of the $\mathcal{CD}$.

In addition, the $\mathcal{DND}$ surpasses the performance of the pure differential distinguisher built with differential distribution tables (DDT) for Speck32/64 in [1]. The result indicates that the $\mathcal{DND}$ learns more than pure differential cryptanalysis and has been verified with the Real-Differences-Experiments. More generally, interpretability for deep neural networks has been considered as a very complex problem. It is hard to interpret the additional differential captured by the $\mathcal{DND}$. In EUROCRYPT 2021, Benamira *et al.* [9] proposed that Gohr's $\mathcal{DND}$ is inherently building an excellent approximation of the DDT during the learning phase and using that information to classify ciphertext pairs directly. Moreover, they found that the $\mathcal{DND}$ generally relies on the differential distribution of the ciphertext pair and the differential distribution in the penultimate and antepenultimate rounds for Speck32/64. However, they do not

come up with a specific form of the additional information the $\mathcal{DND}$ utilizes. Meanwhile, in AICrypt 2023, Gohr [10] proved that the $\mathcal{DND}$ for SIMON32/64 can only utilize differential features and achieve approximately the same distinguishing quality as purely differential ones.

Bao *et al.* [11] extended the concept of difference information, identified additional features used by differential-neural distinguishers for SPECK32/64, and applied these features to classical differential distinguishers, which gained improved performance. Different from directly modifying the input data and verifying the features used by the neural distinguisher through the results, we directly analyze the encryption distribution of different ciphertext pairs to obtain the general features of the neural distinguishers with different ciphers and then conduct experiments to verify that these features do exist. In addition, we have also done some research on the improvement of neural network structure and optimization of acceleration methods for key recovery attacks. The contributions of this work include the following:

- We analyzed the feature information learned by the $\mathcal{DND}$ trained on SIMON32/64 and SPECK32/64 in different input data forms from the perspective of the probability distribution of a ciphertext pair and verified these inferences through improved Real-Difference-Experiments. The most important conclusion is that the $\mathcal{DND}$ not only learns the differential features from the ciphertext pair of SPECK32/64 but also learns the XOR information of the left and right branches of the ciphertext pair, which explains why the performance of the ciphertext pair can surpass DDT.

- Based on the above inferences, we figure out some features exploited by the neural distinguishers of different ciphers. According to these features, we modify the input data format and remove the initial convolutional layer to combine ciphertext pairs linearly. We train a $\mathcal{DND}$ using the modified neural network structure and show that the modified network reduces the number of parameters and training time of the model with little degradation in distinguisher performance, demonstrated in Talbe 5 and 6.

- After an in-depth analysis of the key recovery attack process, we found that using the UCB to calculate the priority of the ciphertext structure sometimes got stuck in the wrong ciphertext structure which was difficult to escape. Therefore, we propose to use the Bayes-UCB strategy to select the correct ciphertext structure. At the same time, we modified the initial guess key selection strategy in the BAYESIANKEYSEARCH algorithm and proposed the Key-Aided Bayesian Key Search algorithm. After several comparative experiments, it is shown that the two improved methods can effectively improve the success rate and reduce complexity of key recovery attacks, directed in Table 9.

Organization. The rest of this paper is organized as follows: Section 2 introduces the design of SPECK32/64 and SIMON32/64, the network structure, and an initial introduction to differential-neural cryptanalysis. Section 3 introduces

some lemma about the probability distribution of the ciphertext pair from different ciphers. Section 4 introduces the modified network structure. An improved key recovery attack method is introduced in Section 5. Finally, we conclude the paper in Section 6.

2. Preliminary

2.1. Notations

Denote by n the word size in bits and $2n$ the state size in bits. Let (C_L^i, C_R^i) be a state's left and right branches after encryption of i rounds, k^i the subkey of i rounds. Denote the bitwise XOR by $\oplus$, the addition modulo 2^n by $\boxplus$, the bitwise AND by $\odot$, the bitwise right/left rotation by $\ggg$ / $\lll$.

2.2. Brief Description of Speck*32/64 and* Simon*32/64*

Speck32/64 and Simon32/64 are members in the lightweight block cipher family Speck and Simon respectively [12]. The i-round Speck32/64 takes a 16-bit subkey k^i and a state consisting of two 16-bit words (C_L^i, C_L^i) as input. The state of the next round (C_L^{i+1}, C_R^{i+1}) is computed as follows:

$$C_L^{i+1} := ((C_L^i \ggg 7) \boxplus C_R^i) \oplus k^i, C_R^{i+1} := (C_R^i \lll 2) \oplus C_L^{i+1}.$$

The i-round (out of 32) Simon32/64 takes a 16-bit subkey k^i and a state consisting of two 16-bit words $(C_L^i, C_L^i))$ as input. The next round state (C_L^{i+1}, C_R^{i+1}) is computed as follows:

$$C_L^{i+1} := (C_L^i \lll 1) \odot (C_L^i \lll 8) \oplus (C_L^i \lll 2) \oplus C_R^i \oplus k^i, C_R^{i+1} := C_L^i.$$

2.3. Differential Scenario

Given an encryption function $E : \mathbb{F}_2^{n_p} \times \mathbb{F}_2^{n_k} \to \mathbb{F}_2^{n_c}$, where n_p represents the number of plaintext bits, n_k represents the number of key bits, and n_c represents the number of ciphertext bits. The task of a differential distinguisher is to distinguish the encryption sample and the random sample:

- encryption sample: $(E(p_1, k), E(p_2, k))$ if $p_1 \oplus p_2 = \Delta$
- random sample: $(E(p_1, k), E(p_2, k))$ if $p_1 \oplus p_2 \neq \Delta$

where Δ is a fixed input difference and k and p are independent and identically distributed over $\mathbb{F}_2^{n_k}$ and $\mathbb{F}_2^{n_p}$, respectively. Using the neural network to train the difference-neural distinguisher instead of the differential distinguisher as the underlying distinguisher, we select encryption samples as positive samples and random samples as negative samples. The primary purpose of the differential-neural distinguisher is to distinguish encryption samples from random samples.

2.4. The Neural Network for Training the Differen-tial-Neural Distinguisher

In [1], a $\mathcal{DND}$ was built for SPECK32/64, which was trained by the neural network shown in Figure 1. The neural network's input layer consisting of ciphertext pairs (encryption samples or random samples) is arranged in a $[n, 4]$ array and transposed in the Preprocess module. Module 1 is the initial with-1 convolution layer that intends to learn bit-sliced functions such as bitwise addition. Module 2 is the Residual Network. $Conv$ stands for one-dimensional convolution $Conv1D$ with N_f filters, and k_s is the size of the convolution kernel. The number of modules 2 is determined experimentally. The prediction head comprises modules 3, 4, and the output layer. FC is a fully connected layer with d_1 or d_2 neurons. BN is the batch normalization layer. $Relu$ and $Sigmoid$ are two different activation functions. If the score exceeds 0.5, the input ciphertext pair is regarded as a positive sample; otherwise, it is viewed as a negative sample.

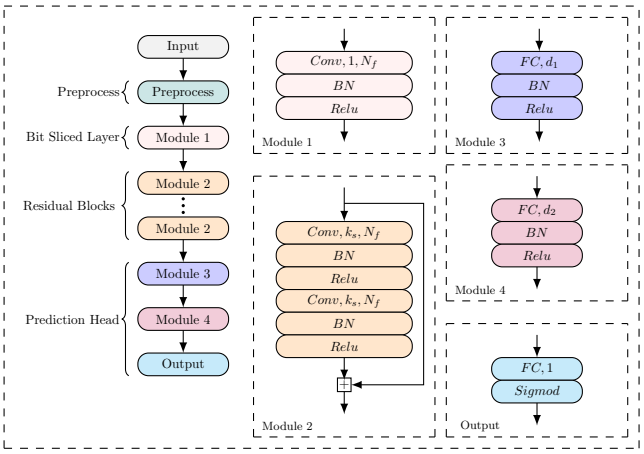

Figure 1: Structure of neural network used in [1].

2.5. Key Recovery Attack of Differential-neural Cryptanal-ysis

Gohr [1] proposed a framework for differential-neural cryptanalysis dedicated to recovering the last two rounds of subkeys for SPECK32/64. A key recovery attack is deemed successful if the last round subkey is guessed correctly and the Hamming distance between the penultimate round subkey and the real subkey is no more than two. In the improved key recovery attack, a short s-round $\mathcal{CD}$ ($\delta \rightarrow \Delta$ with probability 2^{-p}) is prepended before the r-round $\mathcal{DND}$. Since no key addition occurs in SPECK32/64 until the first non-linear operation, the plaintext pairs satisfying the difference δ can be extended for one round without additional cost. Thus, $(1+s+r+1)$-round key recovery attack can be launched. We need to use r-round main $\mathcal{DND}$ and $(r-1)$-round helper $\mathcal{DND}$. In these two $\mathcal{DND}$s' training processes, the plaintext pairs corresponding to the ciphertext pairs need to satisfy the input difference Δ. Figure 2 illustrates the components of the key recovery attack.

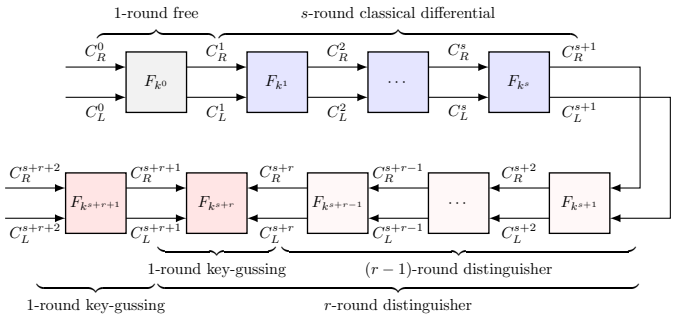

Figure 2: $(1 + s + r + 1)$-round key recovery attack of differential-neural cryptanalysis in [1]

In terms of ciphertext generation, first, we randomly generate approximately $c \times 2^p$ (denoted as n_{cts}) data pairs with an input difference δ, where c is a small constant. Second, each data pair is extended to a data structure containing n_b data pairs using the neutral bits of s-round $\mathcal{CD}$. Then, the obtained n_{cts} data structures are decrypted one round with 0 as the subkey to obtain the plaintext structure. Finally, all plaintext structures are encrypted for $1 + s + r + 1$ rounds to obtain the corresponding ciphertext structure.

When only one round of trial decryption is performed, the wrong key randomization hypothesis does not hold, especially for lightweight ciphers. Based on this observation, an efficient Bayesian key search strategy for accelerating the key recovery attack was proposed by Gohr[1]. The expected response of the distinguisher upon wrong-key decryption depends on the bitwise difference between the subkey k_i and the real subkey k. A new set of candidate subkeys can be obtained by minimizing the weighted Euclidean distance using the precomputed WKRP as a criterion in BayesianKeySearch algorithm.

As the number of encryption rounds increases, the accuracy of the $\mathcal{DNN}$ decreases. To reduce the impact of the misjudgment of the single prediction of the distinguisher, Gohr used the combined response of the $\mathcal{DNN}$ in ciphertext structure with the same distribution, which can be satisfied by neutral bits. The responses $v_{i,k}$ from the $\mathcal{DNN}$ on ciphertext pairs in the ciphertext structure (of size n_b) are combined using the formula $s_k = \sum_{i=0}^{n_b-1} \log_2 \left(\frac{v_{i,k}}{1-v_{i,k}} \right)$ and s_k is considered as the score of a recommended subkey. The score s_k plays a decisive role in the execution time and success rate of the attack. The number of samples with the same distribution should be sufficiently large to enhance the distinguishing ability of the low-accuracy $\mathcal{DNN}$.

Since the $\mathcal{CD}$ is probabilistic, the plaintext structure satisfying the difference δ has only a probability of 2^{-p} to propagate to the ciphertext structure with the difference Δ after s-round differential propagation. Most ciphertext structures do not satisfy the expected distribution, so spending the same computation on each is inefficient. To accelerate the key recovery attack, Gohr proposed to treat the selection of ciphertext structure as a multi-armed bandit problem and solve it using standard exploration-exploitation techniques known as the Upper

Confidence Bound (UCB). The ultimate goal of the algorithm is to determine the ciphertext structure with the highest distinguisher score and regard it as the correct ciphertext structure for a key recovery attack. By calculating a priority $priority_p(i) = X_{\max}(i) + \gamma \times \frac{\sqrt{\log 2(p)}}{\sqrt{N_p(i)}}$ to select the ciphertext structure with the highest priority score for the next key guessing, where $X_{\max}(i)$ is the maximum score ever obtained of ciphertext structure i before the p-th iteration, $N_p(i)$ is the number of times the ciphertext structure i was used before the p-th iteration, and the weights $\gamma = \sqrt{n_{cts}}$. Each ciphertext structure is prioritised based on the highest combined score of recommended subkeys and the number of times it was selected to concentrate computational resources on the most promising ciphertext structures.

3. Features Learned by the Differential-neural Distinguisher

By the Real-Difference-Experiment, Gohr shows the $\mathcal{DND}$ learns only differential features from the ciphertext pair of SIMON, Skinny, Present, and Katan, i.e. features present in the difference distribution table (DDT). However, the $\mathcal{DND}$ learns additional features from the ciphertext pair of SPECK and ChaCha. It seems that the $\mathcal{DND}$ can surpass the classical differential cryptanalysis if there are more features than DDT in the encryption sample.

Key Addition Removes Potential Dependency Between Bits. The encryption sample is of the form $(c_1, c_2) = (E(p,k), E(p \oplus \Delta), k)$ for $k \in \mathbb{F}_2^{n_k}$ and $p \in \mathbb{F}_2^{n_p}$ drawn independently and uniformly at random, which has to be distinguished from the random ones. In general, we would think that if $E(\cdot, k)$ is bijective, choosing p at random is equivalent to choosing c_1 or c_2 at random. Hence, in terms of ciphertext, only taking c_1(or c_2) into account makes no difference from the random ones. After the key addition, the bits would be uniformly distributed. The distinguisher cannot directly obtain information from these bits to distinguish between encryption data and random samples.

Probability Distribution of DDT. When we consider c_1 and c_2 simultaneously, it can be seen from the following lemma that the differences between the ciphertext pair eliminate the randomness generated by key addition.

Lemma 1. *Let $X_1^1, \ldots, X_n^1$, $X_1^2, \ldots, X_n^2$, and $K_1, \ldots, K_n$ be Bernoulli-distributed random variables, and $(x_1^1, \ldots, x_n^1)$, $(x_1^2, \ldots, x_n^2)$ as well as $(\delta_1, \ldots, \delta_n)$ be elements of $\mathbb{F}_2^n$, where $(x_1^2, \ldots, x_n^2) = (x_1^1, \ldots, x_n^1) \oplus (\delta_1, \ldots, \delta_n)$. Let further (for every i in K) K_i be 1 with probablity $\frac{1}{2}$ and independent of $X_1^1, \ldots, X_n^1$, $X_1^2, \ldots, X_n^2$, and $K_1, \ldots, K_{i-1}, K_{i+1}, \ldots, K_n$. Then*

$$\Pr[X_i^1 = x_i^1 \oplus K_i, X_i^2 = x_i^2 \oplus K_i \forall i] = \Pr[X_i^1 = X_i^2 \oplus \delta_i \forall i]$$

Proof. Let $(z_1, \ldots z_m), (w_1, \ldots, w_m)$ as well as $(\delta_1, \ldots, \delta_m)$ be elements of $\mathbb{F}_2^m$. Due to key addition, we can have

$$\Pr\left[V_j = v_j \mid X_i^1 = v_i, X_i^2 = v_i \oplus \delta_i \forall i \neq j\right] = \frac{1}{2}$$

hold for all $(v_1, ..., v_n) \in \mathbb{F}_2^n$ and all $V_j \in \{X_i^1, X_i^2\}$. Then we have that

$$\Pr\left[X_i^1 = z_i, X_i^2 = z_i \oplus \delta_i \forall i\right] = \Pr\left[X_i^1 = w_i, X_i^2 = w_i \oplus \delta_i \forall i\right],$$

i.e., the probability is independent of the concrete choice of $z_1, \ldots z_m$[10].

$$\begin{aligned}
&\Pr[X_i^1 = x_i^1 \oplus K_i, X_i^2 = x_i^2 \oplus K_i \forall i] \\
=&\sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^2 \oplus k_i \forall i] \cdot \Pr[K_i = k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^2 \oplus k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^1 \oplus \delta_i \oplus k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^1 = X_i^2 \oplus \delta_i \forall i] \\
=&\Pr[X_i^1 = X_i^2 \oplus \delta_i \forall i]
\end{aligned}$$

□

This means that as long as the key addition to the ciphertext is not involved in a non-linear operation, the probability distribution only depends on the differences between X^1 and X^2 and, therefore, it can be equivalent to DDT.

3.1. Distribution of Encryption Sample from Reverted Simon*32/64*

From Figure 3, even without knowing the key, part of the last round is always reversible for Simon32/64, i.e. based on the ciphertext (C_L^i, C_R^i), we are able to calculate $(C_L^{i-1}, C_R^{i-1} \oplus K^1)$ in the case of Simon32/64. We take $(Y, X) = (C_L^{i-1}, C_R^{i-1} \oplus K^1)$ as known data, and next we analyze the probability distribution of encryption sample (X^1, Y^1, X^2, Y^2).

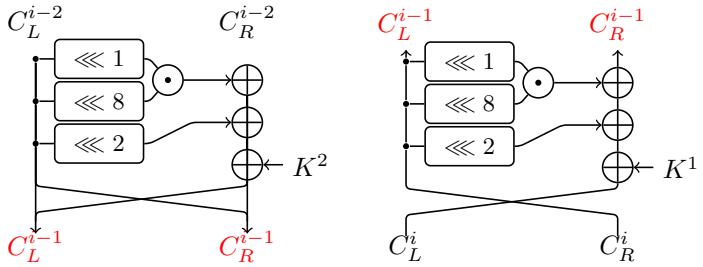

Figure 3: The Round Function of The Reverted Simon32/64

$$
\begin{aligned}
&X = C_R^{i-1} \oplus K^1 \\
=&(C_L^i \oplus (C_R^i \lll 1) \odot (C_R^i \lll 8) \oplus (C_R^i \lll 2)) \oplus K^1 = x \oplus K^1 \\
&Y = C_L^{i-1} \\
=&(C_R^{i-2} \oplus (C_R^{i-1} \lll 1) \odot (C_R^{i-1} \lll 8) \oplus (C_R^{i-1} \lll 2)) \oplus K^2 = y \oplus K^2
\end{aligned}
$$

Lemma 2. *Let $X_1^1, \ldots, X_n^1$, $Y_1^1, \ldots, Y_n^1$, $X_1^2, \ldots, X_n^2$, $Y_1^2, \ldots, Y_n^2$, $K_1^1, \ldots, K_n^1$, and $K_1^2, \ldots, K_n^2$, be Bernoulli-dis-tributed random variables. Let $(x_1^1, \ldots, x_n^1)$, $(x_1^2, \ldots, x_n^2), (\delta_1^1, \ldots, \delta_n^1), (y_1^1, \ldots, y_n^1)$, $(y_1^2, \ldots, y_n^2)$ as well as $(\delta_1^2, \ldots, \delta_n^2)$ be elements of $\mathbb{F}_2^n$, where $(x_1^2, \ldots, x_n^2) = (x_1^1, \ldots, x_n^1) \oplus (\delta_1^1, \ldots, \delta_n^1)$ and $(y_1^2, \ldots, y_n^2) = (y_1^1, \ldots, y_n^1) \oplus (\delta_1^2, \ldots, \delta_n^2)$. Let (for every i in K^1) K_i^1 be 1 with probablity $\frac{1}{2}$ and independent of $X_1^1, \ldots, X_n^1$, $Y_1^1, \ldots, Y_n^1$, $X_1^2, \ldots, X_n^2$, $Y_1^2, \ldots, Y_n^2$, $K_1^1, \ldots, K_{i-1}^1, K_{i+1}^1, \ldots, K_n^1$ and $K_1^2, \ldots, K_n^2$. Let (for every j in K^2) K_j^2 be 1 with probablity $\frac{1}{2}$ and independent of $X_1^1, \ldots, X_n^1, Y_1^1, \ldots, Y_n^1$, $X_1^2, \ldots, X_n^2$, $Y_1^2, \ldots, Y_n^2$, $K_1^1, \ldots, K_n^1$ and $K_1^2, \ldots, K_{j-1}^2, K_{j+1}^2, \ldots, K_n^2$. Then*

$$
\begin{aligned}
&\Pr[X_i^1 = x_i^1 \oplus K_i^1, X_i^2 = x_i^2 \oplus K_i^1 \forall i, Y_j^1 = y_j^1 \oplus K_j^2, Y_j^2 = y_j^2 \oplus K_j^2 \forall j] \\
=&\Pr[X_i^1 = X_i^2 \oplus \delta_i^1 \forall i, Y_j^1 = Y_j^2 \oplus \delta_j^2 \forall j]
\end{aligned}
$$

Proof. Similarly, due to key addition, we can have

$$
\begin{aligned}
&\Pr\left[V_j = v_j \mid X_i^1 = v_i, Y_i^1 = v_i \oplus \delta_i^1 \forall i \neq j\right] = \frac{1}{2} \\
&\Pr\left[W_j = w_j \mid X_i^2 = w_i, Y_i^2 = w_i \oplus \delta_i^2 \forall i \neq j\right] = \frac{1}{2}
\end{aligned}
$$

hold for all $(v_1, ..., v_n)$, $(w_1, ..., w_n) \in \mathbb{F}_2^n$ and all $V_j \in \{X_j^1, Y_j^1\}$,$W_j \in \{X_j^2, Y_j^2\}$

$$
\begin{aligned}
&\Pr[X_i^1 = x_i^1 \oplus K_i^1, X_i^2 = x_i^2 \oplus K_i^1 \forall i, Y_j^1 = y_j^1 \oplus K_j^2, Y_j^2 = y_j^2 \oplus K_j^2 \forall j] \\
=&\sum_{(k_1^1, \ldots, k_n^1) \in \mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i^1, X_i^2 = x_i^2 \oplus k_i^1 \forall i, Y_j^1 = y_j^1 \oplus K_j^2, Y_j^2 = y_j^2 \oplus K_j^2 \\
&\qquad \forall j] \cdot \Pr[K_i^1 = k_i^1 \forall i] \\
=&2^{-n} \cdot \sum_{(k_1^1, \ldots, k_n^1) \in \mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i^1, X_i^2 = x_i^2 \oplus k_i^1 \forall i, Y_j^1 = y_j^1 \oplus K_j^2, Y_j^2 = y_j^2 \oplus \\
&\qquad K_j^2 \forall j] \\
=&2^{-n} \cdot \sum_{(k_1^1, \ldots, k_n^1) \in \mathbb{F}_2^n} \sum_{(k_1^2, \ldots, k_n^2) \in \mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i^1, X_i^2 = x_i^2 \oplus k_i^1 \forall i, Y_j^1 = y_j^1 \oplus k_j^2, \\
&\qquad Y_j^2 = y_j^2 \oplus k_j^2 \forall j] \cdot \Pr[K_j^2 = k_j^2 \forall j] \\
=&2^{-n} \cdot 2^{-n} \cdot \sum_{(k_1^1, \ldots, k_n^1) \in \mathbb{F}_2^n} \sum_{(k_1^2, \ldots, k_n^2) \in \mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i^1, X_i^2 = x_i^2 \oplus k_i^1 \forall i, Y_j^1 = y_j^1 \\
&\qquad \oplus k_j^2, Y_j^2 = y_j^2 \oplus k_j^2 \forall j]
\end{aligned}
$$

$$=2^{-n}\cdot 2^{-n}\cdot \sum_{(k_1^1,\ldots,k_n^1)\in\mathbb{F}_2^n}\sum_{(k_1^2,\ldots,k_n^2)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i^1 \forall i, Y_j^1 = y_j^1 \oplus k_j^2 \forall j, X_i^2 = x_i^1 \oplus \delta_i^1 \oplus k_i^1 \forall i, Y_j^2 = y_j^1 \oplus \delta_j^2 \oplus k_j^2 \forall j]$$

$$=2^{-n}\cdot 2^{-n}\cdot \sum_{(k_1^1,\ldots,k_n^1)\in\mathbb{F}_2^n}\sum_{(k_1^2,\ldots,k_n^2)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i^1, X_i^2 = X_i^1 \oplus \delta_i^1 \forall i, Y_j^1 = y_j^1 \oplus k_j^2, Y_j^2 = Y_j^1 \oplus \delta_j^2 \forall j]$$

$$=\Pr[X_i^1 = X_i^2 \oplus \delta_i^1 \forall i, Y_j^1 = Y_j^2 \oplus \delta_j^2 \forall j]$$

□

We could see that for the form of the ciphertext pair $(Y^1 = y^1 \oplus K^2, X^1 = x^1 \oplus K^1, Y^2 = y^2 \oplus K^2, X^2 = x^2 \oplus K^1)$, where Y, X represent the left and right parts of the ciphertext with key addition respectively and K^1 is independent of K^2, the encryption sample only implies the differential features of ciphertext pairs, which means $\mathcal{DND}$ can only learn differential features from encryption samples. This also means that the accuracy of the $\mathcal{DND}$ will not exceed the accuracy of DDT when the input difference is the same.

3.2. The Real-Difference-Experiment for Reverted Simon*32/64*

To verify whether the $\mathcal{DND}$ for reverted Simon32/64 learns only the features contained in the ciphertext pair derived in Lemma 2, we did some Real-Difference-Experiments, and the results are listed in Table 1. First, we must train a $\mathcal{DND}$. The train and test set include 2×10^7 and 2×10^6 samples. The dataset contains half of the positive samples and half of the negative samples, respectively. The dataset of a 9-round $\mathcal{DND}$ for reverted Simon32/64 is shown below:

- Positive sample: encryption sample.
- Negative sample: random sample.

The accuracy (Acc) and true negative rate (TNR) of $\mathcal{DND}$ are 66.1% and 73.6%, respectively. The Acc indicates the performance of the $\mathcal{DND}$ in distinguishing between the encryption sample and the random sample, and TNR indicates the ability of the $\mathcal{DND}$ to identify a random sample.

Experiment 1: We use a test set of 2×10^6 and put it into the $\mathcal{DND}$ to predict, and the positive and negative samples of the new test set are as follows:

- Positive sample: encryption sample.
- Negative sample: blind encryption sample.

The form of the encryption sample is (Y^1, X^1, Y^2, X^2). Using the random value R to blind different parts of the ciphertexts, the blind encryption sample has $2^4 - 1 = 15$ cases of counterexamples, as shown in Table 1. We can observe changes in the accuracy (Acc_{real}) and true negative rate (TNR_{real}) of the $\mathcal{DND}$. This will help us analyze and verify what features are learned by the $\mathcal{DND}$. If the TNR_{real} is closer to the TNR, it suggests that the blinded encryption sample would look like the random sample to the $\mathcal{DND}$.

Table 1: The Result of Experiment 1

No.	The Form of Negative Sample	Acc_{real}	TNR_{real}
1	$X^1 \oplus R, Y^1, X^2 \oplus R, Y^2$	50.0%	41.5%
2	$X^1, Y^1 \oplus R, X^2, Y^2 \oplus R$	50.0%	41.6%
3	$X^1 \oplus R, Y^1, X^2, Y^2 \oplus R$	66.0%	73.4%
4	$X^1, Y^1 \oplus R, X^2 \oplus R, Y^2$	66.0%	73.5%
5	$X^1 \oplus R, Y^1 \oplus R, X^2, Y^2$	66.0%	73.4%
6	$X^1, Y^1, X^2 \oplus R, Y^2 \oplus R$	66.0%	73.4%
7	$X^1 \oplus R, Y^1 \oplus R, X^2 \oplus R, Y^2 \oplus R$	50.0%	41.4%
8	$X^1, Y^1 \oplus R, X^2 \oplus R, Y^2 \oplus R$	63.8%	69.1%
9	$X^1 \oplus R, Y^1, X^2 \oplus R, Y^2 \oplus R$	66.0%	73.4%
10	$X^1 \oplus R, Y^1 \oplus R, X^2, Y^2 \oplus R$	63.9%	69.2%
11	$X^1 \oplus R, Y^1 \oplus R, X^2 \oplus R, Y^2$	66.0%	73.4%
12	$X^1 \oplus R, Y^1, X^2, Y^2$	63.8%	69.1%
13	$X^1, Y^1 \oplus R, X^2, Y^2$	66.0%	73.5%
14	$X^1, Y^1, X^2 \oplus R, Y^2$	63.8%	69.0%
15	$X^1, Y^1, X^2, Y^2 \oplus R$	66.0%	73.5%

- No.1-2: although $X^1 \oplus R$ and $X^2 \oplus R$ are randomized, $X^1 \oplus X^2$ and $Y^1 \oplus Y^2$ keep unchanged. From the Acc_{real}, it can be observed that the $\mathcal{DND}$ could not distinguish two kinds of samples. A similar explanation for No.2.

- No.3-6: In No.3, due to X^1 and Y^2 being randomized by R, $(X^1 \oplus R) \oplus X^2$ and $Y^1 \oplus (Y^2 \oplus R)$ are random, i.e. the differences between the ciphertext pair are randomized. In view of the $\mathcal{DND}$, the negative samples are no different from random samples. A similar explanation for No.4-6.

- No.7: each word of the ciphertext pair is blinded by a random value R, but the differences between the ciphertext pair do not change. Therefore, the $\mathcal{DND}$ will not be able to distinguish between the blinded encryption samples and the encryption samples. The results show that the $\mathcal{DND}$ does not capture any feature more than differential features.

- No.8-15: since the partial randomization of ciphertext pairs destroys the differences between the ciphertext pair, they are equivalent to a random sample in terms of the $\mathcal{DND}$.

In summary, the $\mathcal{DND}$ for the reverted SIMON32/64 did not learn additional features other than differential features. Through No.8-15, randomizing the left and right differences of the ciphertext pair has different effects on the Acc_{real}. Compared with randomizing X, the Acc_{real} is higher when randomizing Y, which means the blind encryption sample at this time is closer to the random sample. In other words, the differences between Y^1 and Y^2 reveal more information.

3.3. Distribution of Encryption Sample from SIMON*32/64 (Reverted* SPECK*32/64)*

From the right half part of Figure 4, we can calculate $C_R^{i-1} = (C_L^i \oplus C_R^i) \ggg 2$ based on the ciphertext (C_L^i, C_R^i) in the case of SPECK32/64, taking $(X, Y) = (C_L^i, C_R^{i-1})$ as known data; For SIMON32/64, we take $(X, Y) = (C_L^i, C_R^i)$ as known data; The round functions of SIMON32/64 and SPECK32/64 show that X is involved in the key addition, but Y is not. Therefore, we can uniformly analyze the probability distribution of encryption samples from SIMON32/64 and reverted SPECK32/64.

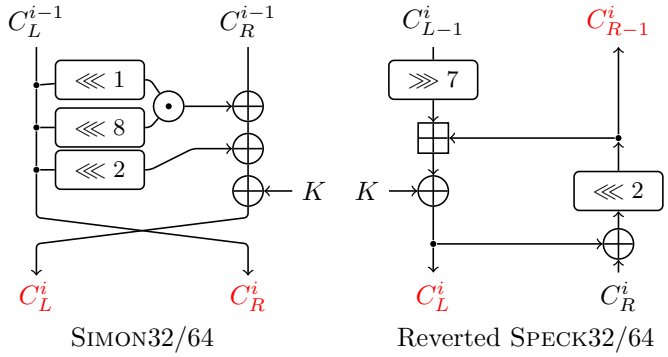

Figure 4: The Round Function of SIMON32/64 Reverted SPECK32/64

Lemma 3. *Let* $X_1^1, \ldots, X_n^1$, $Y_1^1, \ldots, Y_n^1$, $X_1^2, \ldots, X_n^2$, $Y_1^2, \ldots, Y_n^2$, *and* $K_1, \ldots, K_n$, *be Bernoulli-distributed random variables. Let* $(x_1^1, \ldots, x_n^1)$, $(x_1^2, \ldots, x_n^2)$ *as well as* $(\delta_1^1, \ldots, \delta_n^1)$ *be elements of* $\mathbb{F}_2^n$, *where* $(x_1^2, \ldots, x_n^2) = (x_1^1, \ldots, x_n^1) \oplus (\delta_1^1, \ldots, \delta_n^1)$. *Let further (for every i)* K_i *be 1 with probablity* $\frac{1}{2}$ *and independent of* $X_1^1, \ldots, X_n^1$, $Y_1^1, \ldots, Y_n^1$, $X_1^2, \ldots, X_n^2$, $Y_1^2, \ldots, Y_n^2$ *and* $K_1, \ldots, K_{i-1}, K_{i+1}, \ldots, K_n$. *Then*

$$\begin{aligned}&\Pr[X_i^1 = x_i^1 \oplus K_i, X_i^2 = x_i^2 \oplus K_i \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j] \\ =&\Pr[X_i^1 = X_i^2 \oplus \delta_i^1 \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j]\end{aligned}$$

Proof. Similarly, due to key addition, we can have

$$\Pr\left[V_j = v_j \mid X_i^1 = v_i, X_i^2 = v_i \oplus \delta_i^1 \forall i \neq j\right] = \frac{1}{2}$$

hold for all $(v_1, ..., v_n) \in \mathbb{F}_2^n$ and all $V_j \in \{X_i^1, X_i^2\}$.

$$
\begin{aligned}
&\Pr[X_i^1 = x_i^1 \oplus K_i, X_i^2 = x_i^2 \oplus K_i \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j] \\
=&\sum_{(k_1,\cdots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^2 \oplus k_i \forall i, Y_j^1 = y_j^1 \forall j, Y_j^2 = y_j^2 \forall j] \cdot \Pr[K_i = \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\cdots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^2 \oplus k_i \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j] \\
=&2^{-n} \cdot \sum_{(k_1,\cdots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^1 \oplus \delta_i^1 \oplus k_i \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j] \\
=&2^{-n} \cdot \sum_{(k_1,\cdots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = X_i^1 \oplus \delta_i^1 \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j] \\
=&\Pr[X_i^1 = X_i^2 \oplus \delta_i^1 \forall i, Y_j^1 = y_j^1, Y_j^2 = y_j^2 \forall j]
\end{aligned}
$$

□

We can see that for r-round ciphertexts of the form ($X^1 = x^1 \oplus K$, Y^1, $X^2 = x^2 \oplus K$, Y^2), where K is the key independent of (X^1, Y^1, X^2, Y^2), the encryption sample implies not only the r-round differential features of the ciphertext pair but also right half values of the ciphertext pair, which means $\mathcal{DND}$ maybe learn additional features from encryption samples. Note that in some cases, such as for Simon32/64, since the ciphertext pair can also obtain $(r-1)$-round differential features without knowing the r-round key, the accuracy of DND_r with the ciphertext pair as input is almost the same as that of DDT_{r-1}[7].

3.4. The Real-Difference-Experiment for Simon*32/64*

We conducted the Real-Difference-Experiment to more comprehensively verify the features used by the $\mathcal{DND}$ for Simon32/64, and the experimental results are shown in Table 2. Because experimental analysis results are also similar between Simon32/64 and reverted Speck32/64, the Real-Difference-Experiment for reverted Speck32/64 is omitted to avoid duplication of content. Consequently, we only did the Real-Difference-Experiment on Simon32/64. First, we need to train a $\mathcal{DND}$. The train and test set include 2×10^7 and 2×10^6 samples. The dataset contains half of the positive (encryption) samples and half of the negative (random) samples, respectively. The Acc and TNR of $\mathcal{DND}$ are 66.1% and 73.6%, respectively.

Experiment 2: The form of the encryption sample is (X^1, Y^1, X^2, Y^2). Using the random value R to blind different parts of the ciphertexts, there are 15 cases of counterexamples. The new test set of size 2×10^6 includes the positive samples (encryption samples) and negative samples (blind encryption samples), We put the new test set into the $\mathcal{DND}$ to predict, observing changes in the Acc_{real} and TNR_{real} of the $\mathcal{DND}$. This will help us analyze and verify what features are learned by the $\mathcal{DND}$.

Table 2: The Result of Experiment 2

No.	The Form of Negative Samples	Acc_{real}	TNR_{real}
1	$X^1 \oplus R, Y^1, X^2 \oplus R, Y^2$	50.0%	41.8%
2	$X^1, Y^1 \oplus R, X^2, Y^2 \oplus R$	64.6%	70.8%
3	$X^1 \oplus R, Y^1, X^2, Y^2 \oplus R$	65.6%	73.5%
4	$X^1, Y^1 \oplus R, X^2 \oplus R, Y^2$	65.7%	73.6%
5	$X^1 \oplus R, Y^1 \oplus R, X^2, Y^2$	66.0%	73.7%
6	$X^1, Y^1, X^2 \oplus R, Y^2 \oplus R$	66.2%	73.9%
7	$X^1 \oplus R, Y^1 \oplus R, X^2 \oplus R, Y^2 \oplus R$	64.4%	70.9%
8	$X^1, Y^1 \oplus R, X^2 \oplus R, Y^2 \oplus R$	66.2%	74.0%
9	$X^1 \oplus R, Y^1, X^2 \oplus R, Y^2 \oplus R$	66.0%	73.9%
10	$X^1 \oplus R, Y^1 \oplus R, X^2, Y^2 \oplus R$	65.9%	73.9%
11	$X^1 \oplus R, Y^1 \oplus R, X^2 \oplus R, Y^2$	65.9%	73.7%
12	$X^1 \oplus R, Y^1, X^2, Y^2$	66.1%	74.3%
13	$X^1, Y^1 \oplus R, X^2, Y^2$	65.9%	73.7%
14	$X^1, Y^1, X^2 \oplus R, Y^2$	65.7%	73.6%
15	$X^1, Y^1, X^2, Y^2 \oplus R$	66.0%	73.9%

- No.1: although X^1 and X^2 were randomized, the differences between X^1 and X^2 are unchanged, and neither did Y^1 nor Y^2. According to the Lemma 4, this blinded encryption sample is identical to the encryption sample, and the $\mathcal{DND}$ should not be able to distinguish the two kinds of samples. The Acc_{real} is consistent with our inference.

- No.2: we obtained negative samples by randomizing Y^1 and Y^2 from the encryption samples. The inference of Lemma 4 is destroyed, thus the Acc_{real} is greater than 0.5. For SIMON32/64, Y^1 and Y^2 contain the key addition of the previous round. According to Lemma 3, the neural distinguisher can also directly utilize the differential features of the current round, which also explains why Acc_{real} accuracy of the $\mathcal{DND}$ is slightly reduced compared to Acc. Therefore, the $\mathcal{DND}$ indeed learn the feature other than differential features.

- No.3-6: not only is $(X^1 \oplus R) \oplus X^2$ randomized, but $Y^2 \oplus R$ is also randomized in No.3. Therefore, The inference of Lemma 4 are destroyed. The $\mathcal{DND}$ can effectively distinguish blinded encryption samples and encryption samples. A similar explanation for No.4-6.

- No.7: X^1, X^2, Y^1, and Y^2 are randomized simultaneously by bitwise XORing the random value R. Although the value of Y^1 and Y^2 is randomized, the differences between the ciphertext pair remain the same. Therefore, Acc_{real} accuracy of the $\mathcal{DND}$ is slightly reduced compared to Acc. The fact that the $\mathcal{DND}$ can effectively distinguish encryption samples from blind encryption samples shows that the $\mathcal{DND}$ has learned features more than differential features.

- No.8-15: $\frac{3}{4}$ or $\frac{1}{4}$ of a sample are randomized. The differences between X^1 and X^2 being randomized or Y^1 and Y^2 being randomized both cause the blind encryption samples to be more biased towards the random samples, so the $\mathcal{DND}$ can distinguish between the encryption samples and the blind encryption samples.

The above results show that the $\mathcal{DND}$ learns not only the differential features but also the features of the right half of the ciphertext pair.

3.5. Distribution of Encryption Sample from SPECK*32/64*

We take $(X, Y) = (C_L^i, C_R^i)$ as known data. From the round functions of SPECK32/64 in Figure 5, it can be seen that both X and Y involve an XOR operation of the same key K. Accordingly, we analyze the probability distribution of encryption sample (X^1, Y^1, X^2, Y^2).

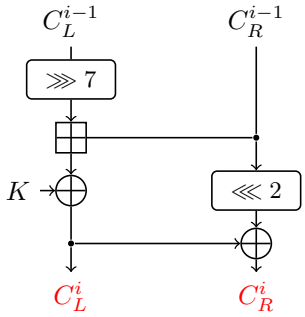

Figure 5: The Round Function of SPECK32/64

Lemma 4. *Let* $X_1^1, \ldots, X_m^1, Y_1^1, \ldots, Y_m^1, X_1^2, \ldots, X_m^2, Y_1^2, \ldots, Y_m^2$, *and* $K_1, \ldots, K_m$, *be Bernoulli-distributed random variables. Let* $(x_1^1, \ldots, x_n^1)$, $(x_1^2, \ldots, x_n^2)$, $(y_1^1, \ldots, y_n^1)$, $(y_1^2, \ldots, y_n^2)$, $(\delta_1^1, \ldots, \delta_n^1)$, $(\delta_1^2, \ldots, \delta_n^2)$ *as well as* $(\delta_1^3, \ldots, \delta_n^3)$ *be elements of* $\mathbb{F}_2^n$, *where* $(x_1^2, \ldots, x_n^2) = (x_1^1, \ldots, x_n^1) \oplus (\delta_1^1, \ldots, \delta_n^1)$, $(y_1^1, \ldots, y_n^1) = (x_1^1, \ldots, x_n^1) \oplus (\delta_1^2, \ldots, \delta_n^2)$, $(y_1^2, \ldots, y_n^2) = (x_1^1, \ldots, x_n^1) \oplus (\delta_1^3, \ldots, \delta_n^3)$. *Let further (for every i)* K_i *be 1 with probability* $\frac{1}{2}$ *and independent of* $X_1^1, \ldots, X_m^1$, $Y_1^1, \ldots, Y_m^1$, $X_1^2, \ldots, X_m^2$, $Y_1^2, \ldots, Y_m^2$ *and* $K_1, \ldots, K_{i-1}, K_{i+1}, \ldots, K_m$. *Then*

$$\begin{aligned}
&\Pr[X_i^1 = x_i^1 \oplus K_i, X_i^2 = x_i^2 \oplus K_i, Y_i^1 = y_i^1 \oplus K_i, Y_i^2 = y_i^2 \oplus K_i \forall i] \\
=&\Pr[X_i^2 = X_i^1 \oplus \delta_i^1, Y_i^1 = X_i^1 \oplus \delta_i^2, Y_i^2 = X_i^1 \oplus \delta_i^3 \forall i] \qquad (1) \\
=&\Pr[X_i^1 = X_i^2 \oplus \delta_i^1, Y_i^1 = X_i^2 \oplus \delta_i^2 \oplus \delta_i^1, Y_i^2 = X_i^2 \oplus \delta_i^1] \qquad (2)
\end{aligned}$$

Proof. Similarly, due to key addition, we can have

$$\Pr[V_j = v_j \mid X_i^1 = v_i, X_i^2 = v_i \oplus \delta_i^1, Y_i^1 = v_i \oplus \delta_i^2, Y_i^2 = v_i \oplus \delta_i^3 \forall i \neq j] = \frac{1}{2}$$

hold for all $(v_1, ..., v_n) \in \mathbb{F}_2^n$ and all $V_j \in \{X_j^1, Y_j^1, X_j^2, Y_j^2\}$

$$\begin{aligned}
&\Pr[X_i^1 = x_i^1 \oplus K_i, X_i^2 = x_i^2 \oplus K_i, Y_i^1 = y_i^1 \oplus K_i, Y_i^2 = y_i^2 \oplus K_i \forall i] \\
=&\sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, Y_i^1 = y_i^1 \oplus k_i, X_i^2 = x_i^2 \oplus k_i, Y_i^2 = y_i^2 \oplus k_i \forall i] \cdot \\
&\qquad\qquad \Pr[K_i = k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, Y_i^1 = y_i^1 \oplus k_i, X_i^2 = x_i^2 \oplus k_i, Y_i^2 = y_i^2 \oplus k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = x_i^1 \oplus \delta_i^1 \oplus k_i, Y_i^1 = x_i^1 \oplus \delta_i^2 \oplus k_i, Y_i^2 \\
&\qquad\qquad = x_i^1 \oplus \delta_i^3 \oplus k_i \forall i] \\
=&2^{-n} \cdot \sum_{(k_1,\ldots,k_n)\in\mathbb{F}_2^n} \Pr[X_i^1 = x_i^1 \oplus k_i, X_i^2 = X_i^1 \oplus \delta_i^1, Y_i^1 = X_i^1 \oplus \delta_i^2, Y_i^2 = X_i^1 \oplus \\
&\qquad\qquad \delta_i^3 \forall i] \\
=&\Pr[X_i^2 = X_i^1 \oplus \delta_i^1, Y_i^1 = X_i^1 \oplus \delta_i^2, Y_i^2 = X_i^1 \oplus \delta_i^3 \forall i]
\end{aligned}$$

□

Similarly, we can obtain inference (2), and the same process also applies to Y^1 or Y^2. We can see that for ciphertexts of the form ($X^1 = x^1 \oplus K$, $Y^1 = y^1 \oplus K$, $X^2 = x^2 \oplus K$, $Y^2 = y^2 \oplus K$), where K is the key independent of (X^1, Y^1, X^2, Y^2), the encryption sample implies not only the difference feature of the ciphertext pair but also XOR information between the left and right branches of a ciphertext pair.

3.6. The Real-Difference-Experiment for SPECK*32/64*

From Lemma 4, the ciphertext pair contains more features than differential features. Next, we conduct the Real-Difference-Experiment to more comprehensively verify the features used by the $\mathcal{DND}$ for SPECK32/64, and the experimental results are shown in Table 3. First, we need to train a $\mathcal{DND}$. The train and test set include 2×10^7 and 2×10^6 samples. The dataset contains half of the positive (encryption) and half of the negative (random) samples. The Acc and TNR of $\mathcal{DND}$ for 6-round SPECK32/64 are 78.5% and 71.9%, respectively.

Experiment 3: The form of the encryption sample is (X^1, Y^1, X^2, Y^2). Using the random value R to blind different parts of the ciphertexts, there are 15 cases of counterexamples. The new test set of size 2×10^6 includes the positive samples (encryption sample) and negative samples (blind encryption samples), We put the new test set into the $\mathcal{DND}$ to predict, observing changes in the Acc_{real} and TNR_{real} of the $\mathcal{DND}$. This will help us analyze and verify what features are learned by the $\mathcal{DND}$.

- No.1-2: X^1 and X^2 are randomized in No.1. Although the differences between (X^1, X^2) remain unchanged, the of ($X^1 \oplus R$, Y^1) and ($X^1 \oplus R$, Y^2)

Table 3: The Result of Experiment 3

No.	The Form of Negative Samples	Acc_{real}	TNR_{real}
1	$X^1 \oplus R, Y^1, X^2 \oplus R, Y^2$	60.1%	48.3%
2	$X^1, Y^1 \oplus R, X^2, Y^2 \oplus R$	60.4%	48.7%
3	$X^1 \oplus R, Y^1, X^2, Y^2 \oplus R$	76.8%	81.7%
4	$X^1, Y^1 \oplus R, X^2 \oplus R, Y^2$	76.7%	81.6%
5	$X^1 \oplus R, Y^1 \oplus R, X^2, Y^2$	76.9%	81.9%
6	$X^1, Y^1, X^2 \oplus R, Y^2 \oplus R$	77.1%	82.1%
7	$X^1 \oplus R, Y^1 \oplus R, X^2 \oplus R, Y^2 \oplus R$	50.1%	27.6%
8	$X^1, Y^1 \oplus R, X^2 \oplus R, Y^2 \oplus R$	78.4%	85.1%
9	$X^1 \oplus R, Y^1, X^2 \oplus R, Y^2 \oplus R$	78.7%	85.2%
10	$X^1 \oplus R, Y^1 \oplus R, X^2, Y^2 \oplus R$	78.7%	85.2%
11	$X^1 \oplus R, Y^1 \oplus R, X^2 \oplus R, Y^2$	78.3%	84.9%
12	$X^1 \oplus R, Y^1, X^2, Y^2$	78.7%	85.2%
13	$X^1, Y^1 \oplus R, X^2, Y^2$	78.5%	84.8%
14	$X^1, Y^1, X^2 \oplus R, Y^2$	78.7%	85.3%
15	$X^1, Y^1, X^2, Y^2 \oplus R$	78.4%	84.8%

are randomized. Thus, the $\mathcal{DND}$ can distinguish the encryption and blind encryption samples. It is difficult for the $\mathcal{DND}$ to distinguish between the two kinds of samples, so the Acc_{real} is relatively low. A similar explanation for No.2.

- No.3-4: X^1 and Y^2 are randomized in No.3, the of (X^1, X^2) and (X^1, Y^1) are randomized. Thus, the $\mathcal{DND}$ can distinguish the encryption and blind encryption samples. Therefore, the Acc_{real} is slightly lower than Acc. A similar explanation for No.4.

- No.5-6: X^1 and Y^1 are randomized in No.5, the of (X^1, X^2) and (X^1, Y^2) are randomized. Thus, the $\mathcal{DND}$ can distinguish the encryption and blind encryption samples. Therefore, the Acc_{real} is slightly lower than Acc. A similar explanation for No.6.

- No.7: although X^1, Y^1, X^2, and Y^2 are randomized, the differences between the ciphertext pair and the differences between different branches of the ciphertext pair don't change. This means that the distribution of the blind encryption sample is the same as that of the encryption distribution. Therefore, the $\mathcal{DND}$ cannot distinguish between these two samples.

- No.8-15: the blind encryption sample destroys the differences between the ciphertext pair and the left and right branches of the ciphertext pair, making the blind encryption sample equal to a random sample. As a result, the Acc_{real} almost equals the Acc.

The above experiments prove that the $\mathcal{DND}$ has indeed learned the differences between the ciphertext pair and the XOR information between different

branches of the ciphertext pair. When the differences between the ciphertext pair are destroyed, the Acc_{real} of the $\mathcal{DND}$ is greatly affected compared with the XOR information between different branches of the ciphertext pair. This means that the $\mathcal{DND}$ mainly learns the differences between ciphertext pairs.

4. Modified Neural Network Structure

In the neural network designed by Gohr [1], the initial convolutional layer with a kernel size of 1 is for learning simple bit-slicing functions such as bitwise addition. Furthermore, Benamira *et al.* [9] proved that the initial convolutional layer realizes the linear combination of different parts of the ciphertext pair. Through the analysis of Sect. 3, we already know the probability distribution of a ciphertext pair and the features that the $\mathcal{DND}$ can learn from a ciphertext pair. Therefore, we can remove the initial convolutional layer from the network structure proposed in [1]. Specifically, we modify the form of the input data according to the Lemma 2, 3, 4 as follows:

- SIMON32/64: $(X^1 \oplus X^2, Y^1, Y^2)$
- Reverted SIMON32/64: $(X^1 \oplus X^2, Y^1 \oplus Y^2)$
- SPECK32/64: $(X^1 \oplus X^2, X^1 \oplus Y^1, X^1 \oplus Y^2)$
- Reverted SPECK32/64: $(X^1 \oplus X^2, Y^1, Y^2)$

Network Structure: The structure of the modified neural network is shown in Figure 6. We modify the form of input data to the neural network according to the probability distribution of the ciphertext pair. Compared to the network structure presented in [1], the initial convolutional layer is removed. The input data of the neural network is a sequence of $N_w \times n$ bits, where N_w is the number of words, which is reshaped to an array of (N_w, n) and transposed in the Preprocess module. Module 2 is the Residual Network. $Conv$ stands for one-dimensional convolution $Conv1D$ with N_f or N_w filters, and k_s is the size of the convolution kernel. In the two-layer residual block, the number of input filters must equal the number of output filters, so the number of filters increases from N_w to N_f and then decreases to N_w. The experiment determines the number of modules 2 (*depth*). It should be noted that the circular convolution (*cconv*) listed in Table 4 refers to whether $Conv1d$ is replaced by a circular convolution layer used in [10]. The prediction head comprises modules 3, 4, and the output layer. Please refer to Table 4 for the value of some parameters in the network structure.

The Training of $\mathcal{DND}$: We conducted the training for 80 epochs in the dataset where the size of the train set is 10^7 and test set 10^6. The batch size is denoted by B_s. Optimization was performed against mean square error loss plus a small penalty $L2$ using the Adam algorithm. A cyclic learning rate schedule was applied, setting the learning rate l_i for epoch i to $l_i = \alpha + \frac{(n-i) \mod (n+1)}{n} \cdot (\beta - \alpha)$

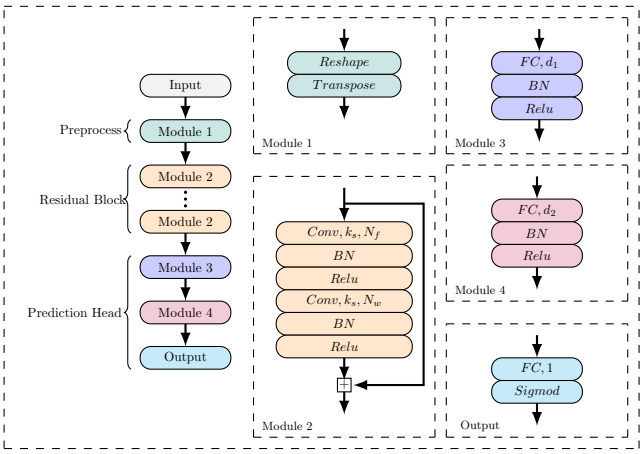

Figure 6: Structure of Modified Neural Network.

with $n = 9$. The model obtained at the end of each epoch was stored, and the best model by validation loss was evaluated against a test set. Please refer to Table 4 for the value of some training parameters.

Table 4: Hyperparameters of the modified neural network

	SIMON32/64	Reverted SIMON32/64	SPECK32/64	Reverted SPECK32/64
B_s	5000	5000	2000	2000
β (10^{-3})	3	4	3.5	4
α (10^{-4})	1	1	1	1
$depth$	10	4	10	5
d_1	64	64	64	64
d_2	64	64	64	64
k_s	7	7	3	3
N_f	48	32	32	16
N_w	3	2	3	3
$L2(10^{-7})$	40	10	5	20
$cconv$	True	True	False	False

The Accuracy of $\mathcal{DND}$: We summarize the accuracy of $\mathcal{DND}$ compared to previous work in Table 5. On the one hand, when we remove the initial convolutional layer from the neural network and modify the format of the input data according to the derived lemmas, the accuracy of the $\mathcal{DND}$ does not decrease significantly, which shows the correctness of our lemmas. On the other hand, the $\mathcal{DND}$ trained for reverted SIMON32/64/reverted SPECK32/64 is not higher than that of the $\mathcal{DND}$ trained for SIMON32/64/SPECK32/64, which indicates

that although the $\mathcal{DND}$ learns the distribution of the value of a ciphertext pair, it may not improve the performance of the $\mathcal{DND}$.

Table 5: The Acc of $\mathcal{DND}$ trained by modified neural network. For comparison, the Acc is listed in descending order.

Cipher	r	Acc	Ref.
SIMON32/64	9	66.1%	[10]
		66.0%	This work
		65.9%	[7]
		$< 63.0\%$	[13]
		62.8%	[14]
Reverted SIMON32/64	9	65.9%	This work
SPECK32/64	6	78.8%	This work
		78.8%	[1]
		78.4%	[2]
	7	61.7%	[10]
		61.4%	This work
		60.7%	[2]
Reverted SPECK32/64	6	78.6%	This work
	7	61.4%	This work

The Parameters Amount of Neural Network: The primary objective of modifying the network structure is to reduce network size, decrease the model file size, and accelerate model training. This is accomplished by removing initial convolutional layers and reducing the number of neurons in the fully connected layer. Consequently, the modified neural network substantially reduces overall parameters, as Table 6 illustrates. The reduction in model parameters directly contributes to shorter training times. Moreover, the reduction in model parameters also reduces forward propagation time during the prediction process. This reduction in time may potentially decrease the complexity of key recovery attacks to some extent.

Table 6: The total amount of parameters and train time each epoch (on an NVIDIA GeForce RTX 3070 graphics card)

Cipher	Parameter Amount	Train Time Each Epoch(s)	Ref.
SIMON32/64	449425	1300	[7]
	30583	120	This work
Reverted SIMON32/64	184417	150	[10]
	11113	45	This work
SPECK32/64	44865	72	[10]
	15383	130	This work
Reverted SPECK32/64	9788	73	This work

5. Improvement of Key Recovery Attack

We have made some improvements to the key recovery attack proposed by Gohr [1] and first introduce the experimental environment[1] and complexity calculation method. We consider the key guess successful if the last round subkey is guessed correctly and the Hamming distance between the penultimate round subkey and the real subkey is no more than two. The data complexity of the experiment is calculated by the formula $n_b \times \min(n_{ct}, n_{used}) \times 2$, where n_{used} is the number of ciphertext structures actually used and n_b is the number of ciphertext pairs in each ciphertext structure. The time complexity calculation formula in our experiment is $n_{kg} \times 2^{28} \times rt$, under the assumption that one second equals the time of 2^{28} fully SIMD-parallel executions of SPECK32/64 or SIMON32/64 on a CPU[1], rt is the average running time of multiple experiments, and n_{kg} is the number of possible values for the bits of the first round subkey k^0, on which the conditions depend.

5.1. Improved Selection Strategy of Ciphertext Structure

In differential-neural cryptanalysis, a $\mathcal{CD}$ is employed on top of the $\mathcal{DND}$ to enhance the round of key recovery attacks. However, the s-round $\mathcal{CD}$ ($\delta \to \Delta$) operates probabilistically. Given a plaintext structure that satisfies the difference δ that is encrypted after s rounds, if the corresponding ciphertext structure exhibits a difference that satisfies Δ, it is referred to as the correct ciphertext structure; otherwise, it is considered the wrong ciphertext structure. Gohr [1] utilized the UCB (Upper Confidence Bound) strategy to select the correct ciphertext structure as follows:

$$priority_p(i) = X_{\max}(i) + \gamma \times \frac{\sqrt{\log 2(p)}}{\sqrt{N_p(i)}},$$

[1]The experiment is conducted by Python 3.7.15 and Tensorflow 2.5.0 in Ubuntu 20.04. The device information is Intel(R) Xeon(R) Gold 6226R*2 with 2.90GHz, 256GB RAM, and NVIDIA RTX2080Ti 12GB*5.

In the formula, $X_{\max}(i)$ denotes the maximum score achieved for ciphertext structure i prior to the p-th iteration, $N_p(i)$ represents the number of times ciphertext structure i has been used until the p-th iteration, and the weight γ is defined as $\sqrt{n_{cts}}$. This strategy estimates the priority of each ciphertext structure through multiple sampling and utilizes these sampled results to select an appropriate ciphertext structure in subsequent iterations.

Due to the term $\gamma \times \sqrt{\log 2(p)}$, the early stage of UCB allows for the sequential selection of all ciphertext structures. Bring the obtained distinguisher score into the above formula, and the computed value is used as the initial estimation of the ciphertext structure's priority, which is continuously updated during the selection process. As a ciphertext structure is repeatedly selected, $N_p(i)$ keeps increasing, potentially decreasing its priority. This strategy enables escaping from such a ciphertext structure and searching for a more promising one, effectively avoiding local optima.

In addition, the value of weights γ plays a significant role in balancing the trade-off between "finding a new ciphertext structure" and "utilizing the ciphertext structure with the highest priority". In our experiments, we find that the UCB algorithm sometimes cannot get rid of incorrect ciphertext structures. For instance, if we mistakenly choose a ciphertext structure and the guessed subkey with a small Hamming distance from the real subkey, the $\mathcal{DND}$ score can be higher. This could potentially result in the score of the wrong ciphertext structure being higher than that of the correct ciphertext structure decrypted using an incorrect subkey. Consequently, a wrong ciphertext structure might be selected, leading to an impact on efficiency. The selection of an appropriate weight γ proves to be a challenging task. Hence, we aim to identify a more efficient and convenient strategy that does not rely on weight selection.

In [15], the authors propose a general bandit strategy called Bayesian Upper Confidence Bound (Bayes-UCB). Instead of using the frequency-based UCB strategy, Bayes-UCB assigns each ciphertext structure a parameter that follows a prior distribution. This strategy aims to achieve an average performance across all potential problem instances, taking the prior distribution of the parameters into account. The agent is confronted with a set of n_{cts} ciphertext structures, each associated with an unknown parameter. The agent selects the ciphertext structure (I_p) for the p-th time based on the cumulative score of each ciphertext structure.

Assuming that the agent selects the ciphertext structure I_p at the p-th iteration according to a given policy, we denote the score of $\mathcal{DND}$ at the j-th selection as X_j. For the ciphertext structure i, if $I_p = i$, we represent it as $\mathbb{1}(I_p = i) = 1$; otherwise, $\mathbb{1}(I_p \neq i) = 0$. The cumulative score of ciphertext structure i before the p-th selection is denoted as $s_p(i)$. We use $Q(1-\nu, \rho)$ to refer to the quantile function associated with the distribution ρ, and $\mathcal{T}(d)$ represents the t-distribution with d degrees of freedom. By employing Bayes-UCB, the priority of the ciphertext structure is calculated as follows:

$$priority_p(i) = \frac{s_p(i)}{N_p(i)} + \sqrt{w_p(i)} \times \frac{Q(1-\frac{1}{p}, \mathcal{T}(N_p(i)-1))}{\sqrt{N_p(i)}},$$

where

$$s_p(i) = \sum_{j=1}^{p-1} \mathbb{1}(I_j = i) X_j, w_p(i) = \frac{(\sum_{j=1}^{p-1} \mathbb{1}(I_j = i) X_j^2) - s_p^2(i)/N_p(i)}{N_p(i) - 1}$$

for each ciphertext structure, and the ciphertext structure with the highest priority is selected for further processing. Compared with the UCB, Bayes-UCB has the following differences:

- To prevent the strategy from getting trapped in a local optimum caused by occasional "good scores" of the ciphertext structure, we opt for utilizing the average score $\frac{s_p(i)}{N_p(i)}$ instead of the maximum score $X_{\max}(i)$.
- Using the quantile function, denoted as $Q(1 - \frac{1}{p}, \mathcal{T}(N_p(i) - 1))$, instead of $\sqrt{log2(p)}$ in the UCB is a more favourable approach for identifying the correct ciphertext structure. In a specific selection scenario, where the ciphertext structure i is initially chosen according to the strategy and continuously selected in the following iterations, $Q(1 - \frac{1}{p}, \mathcal{T}(N_p(i) - 1))$ of other ciphertext structures tends to increase as the number of iterations p increases. Consequently, this increased value makes these alternative ciphertext structures more likely to be selected. Additionally, as the frequency of selecting ciphertext structure i(denoted as $N_P(i)$) continuously increased, the strategy becomes more prone to breaking away from the local optimum of ciphertext structure i. This, in turn, accelerates the process of escaping the local optimum by the strategy and promotes faster convergence towards the optimal solution.
- Additionally, the initial priorities of the ciphertext structures are significantly higher than their actual priorities due to the factor $Q(1 - \frac{1}{p}, \mathcal{T}(N_p(i) - 1))$. As a result, all ciphertext structures can be sequentially selected in the early stages of the algorithm, and the calculated value estimates the score distribution of the ciphertext structures. To achieve a better balance between "exploration" and "utilization" without relying on weights, we introduce a dynamic value $\sqrt{w_p(i)}$ to replace the weights $\gamma = \sqrt{n_{cts}}$.

We conducted 11-round key recovery attacks on SPECK32/64 using Bayes-UCB, leveraging the $\mathcal{DND}$s trained in [1], to evaluate its effectiveness. To ensure a fair comparison, we reproduced the results of [1] on our equipment, and the results are presented in Table 7. The experimental results demonstrate that using Bayes-UCB for key recovery attacks improves time complexity and success rate.

5.2. Improved BAYESIANKEYSEARCH Algorithm

During the process of recovering the key, we observed that utilizing the correct key with an incorrect ciphertext structure sometimes leads to a higher

Table 7: The result of key recovery attack using the Bayes-UCB Algorithm

r	Conf.	c_1	c_2	n_{cts}	n_{it}	n_{kg}	N_e	rt(s)	sr	Time	Ref.
11	1+2+7+1	5	10	100	500	2^0	100	227.07	51%	$2^{35.83}$	[1]
								182.57	55%	$2^{35.52}$	Bayes-UCB

1. c_1 and c_2 represent the cutoffs with respect to the scores of the recommended last subkey and second to last subkey, respectively.

2. The success rate sr is calculated as the number of successful recoveries divided by the total number of experiments (N_e).

3. n_{it} represents the maximum number of iterations in key recovery attacks.

score. As a result, we introduce an improved BayesianKeySearch Algorithm 1, which retains the key with the highest score during key recovery and employs it to generate a new candidate key for each Bayesian key search. Furthermore, we conducted key recovery attacks to showcase the effectiveness of the improved BayesianKeySearch Algorithm and obtained Table 8.

Table 8: The result of key recovery attack using the improved BayesianKeySearch Algorithm

r	Conf.	c_1	c_2	n_{cts}	n_{it}	n_{kg}	N_e	rt(s)	sr	Time	Ref.
11	1+2+7+1	5	10	100	500	2^0	100	227.07	51%	$2^{35.83}$	[1]
								158.84	52%	$2^{35.32}$	Alg. 1

From the above experimental results, it can be seen that when the improved BayesianKeySearch algorithm is used for key recovery attacks, the sr is almost not improved, but the time complexity is reduced compared with that of UCB.

To verify the effectiveness of Bayes-UCB and improved BayesianKeySearch algorithm at the same time, we launched 11-round, 12-round and 13-round key recovery attacks on Speck32/64 using these improved algorithms, with the help of $\mathcal{DND}$s of Speck32/64 trained in [1]. For a fairer comparison, the results of [1] and [7] are reproduced on our equipment, and the results are listed in Table 9.

Algorithm 1: Improved BayesianKeySearch Algorithm

Input: Ciphertext structures $\mathcal{C} := C_0, \ldots, C_{n_b-1}$, the number of candidates to be generated within each iteration n_{cand}, the key K_{best} with the highest score during key recovery, the number of iterations l, a $\mathcal{DND}$, and its wrong key response profile μ and σ

Output: The list L of tuples of recommended keys and their scores

if $K_{best} = None$ **then**
  $S := \{k_0, k_1, \ldots, k_{n_{cand}-1}\} \leftarrow$ choose n_{cand} scores at random without replacement from the set of all subkey candidates.;
else
  $S := \{k_0, k_1, \ldots, k_{n_{cand}-2}\} \leftarrow$ choose $n_{cand} - 1$ scores at random except K_{best} without replacement from the set of all subkey candidates.;
  $S := K_{best}||S$;
end
$L \leftarrow \{\}$;
for $t = 1$ *to* l **do**
  for $\forall k_i \in S$ **do**
    for $j = 0$ *to* $n_{cts} - 1$ **do**
      $C'_{j,k_i} \leftarrow Decrypt(C_j, k_i)$
      $z_{j,k_i} = \mathcal{DND}(C'_{j,k_i})$
      $z_{j,k_i} = \log_2 \frac{z_{j,k_i}}{1-z_{j,k_i}}$
    end
    $s_{k_i} = \sum_{j=0}^{n_b-1} z_{j,k_i}$ `/* the combined score of` k_i `using neutral bits. */`
    $L \leftarrow L||(k_i, s_{k_i})$
    $m_{k_i} = \sum_{j=0}^{n_b-1} z_{j,k_i}/n_{cts}$
    for $k \in \{0, 1, \ldots, 2^{16} - 1\}$ **do**
      $\lambda = \sum_{i=0}^{n_{cand}-1} (m_{k_i} - \mu_{k_i \oplus k})^2 / \sigma^2_{k_i \oplus k}$
    end
  end
  $S \leftarrow argsort_k(\lambda)[0 : n_{cand} - 1]$; `/* Select the` n_{cand} `keys with the` n_{cand} `smallest score as the new candidate keys` S `*/`
end
return L

Table 9: Summary of key-recovery attacks on SPECK32/64

r	Conf.	c_1	c_2	n_{cts}	n_{it}	n_{kg}	N_e	rt(s)	sr	Time	Data	Ref.
11	1+2+7+1	5	10	100	500	2^0	1000	216.61	36.1%	$2^{35.76}$	$2^{13.49}$	[1]
								168.07	52.8%	$2^{35.40}$	$2^{13.50}$	This work
		10	10	256	500	2^0	1000	69.56	65.2%	$2^{34.12}$	$2^{14.66}$	[1]
								68.50	65.4%	$2^{34.10}$	$2^{14.61}$	This work
12	1+3+7+1	7	10	2^{12}	2^{13}	2^1	1000	437.63	100.0%	$2^{37.78}$	$2^{18.58}$	[7]
								484.52	100.0%	$2^{37.92}$	$2^{18.58}$	This work
	1+2+8+1	20	500	500	2000	2^0	100	1842.43	39.0%	$2^{38.85}$	$2^{22.92}$	[1]
								1786.86	50.0%	$2^{38.81}$	$2^{22.89}$	This work
13	1+3+8+1	8	-500	2^{11}	2^{12}	2^4	100	17011.22	21%	$2^{46.05}$	2^{27}	[16]
								16145.12	24%	$2^{45.97}$	2^{27}	This work

From the above experimental results, it can be seen that when using the combination of the Bayes-UCB and improved BAYESIANKEYSEARCH Algorithm for key recovery attacks, although the time complexity of the attack is slightly reduced, the sr is significantly increased.

Remark 1. *The sr of correct key guessing in the last round using Gohr's key recovery attack algorithm is as high as* 55.9%*, but the final sr is far lower than this value when* $c_1 = 5$ *and* $c_2 = 10$*. In our experiment, the sr of the last round of key guessing is only* 52.9%*, but the final sr hardly decreases. The main reason for this phenomenon is that the threshold* c_1 *and* c_2 *are not correctly selected, and the UCB strategy cannot determine the optimal ciphertext structure well. Suppose the algorithm guesses the correct key while the wrong ciphertext structure is selected. In that case, this will lead to a situation where the priority of the wrong ciphertext structure is too high, and the inappropriate threshold cannot filter the wrong ciphertext structure very well. As a result, the sr is significantly reduced when recovering the penultimate round subkey. In contrast, using Bayes-UCB and improved* BAYESIANKEYSEARCH *algorithm solves this problem.*

6. Conclusion

In this paper, we first analyze the probability distribution of the ciphertext pair generated by various ciphers and investigate the learned features of the differential-neural distinguisher. Subsequently, we validated our findings with improved Real-Difference-Experiments. Based on the features exploited by the neural distinguisher, we primarily remove the initial convolutional layers and modify the input data format to reduce the number of network parameters and training time while maintaining high accuracy. Additionally, we present some enhancements to the key recovery attack procedure. We use Bayes-UCB to select

an appropriate ciphertext structure and propose an improved algorithm called Key-Aided Bayesian Key Search. These enhancements significantly increase the success rate and reduce the time complexity of key recovery attacks.

References

[1] A. Gohr, Improving attacks on round-reduced speck32/64 using deep learning, in: CRYPTO (2), Vol. 11693 of Lecture Notes in Computer Science, Springer, 2019, pp. 150–179.

[2] Y. Chen, Y. Shen, H. Yu, S. Yuan, A new neural distinguisher considering features derived from multiple ciphertext pairs, The Computer Journal 66 (6) (2023) 1419–1433.

[3] L. Zhang, Z. Wang, B. Wang, Improving differential-neural cryptanalysis with inception blocks, IACR Cryptol. ePrint Arch. (2022) 183.

[4] Z. Hou, J. Ren, S. Chen, Improve neural distinguisher for cryptanalysis, IACR Cryptol. ePrint Arch. (2021) 1017.

[5] J. Lu, G. Liu, Y. Liu, B. Sun, C. Li, L. Liu, Improved neural distinguishers with (related-key) differentials: Applications in SIMON and SIMECK, IACR Cryptol. ePrint Arch. (2022) 30.

[6] H. Su, X. Zhu, D. Ming, Polytopic attack on round-reduced simon32/64 using deep learning, in: Inscrypt, Vol. 12612 of Lecture Notes in Computer Science, Springer, 2020, pp. 3–20.

[7] Z. Bao, J. Guo, M. Liu, L. Ma, Y. Tu, Conditional differential-neural cryptanalysis, IACR Cryptol. ePrint Arch. (2021) 719.

[8] N. Băcuieți, L. Batina, S. Picek, Deep neural networks aiding cryptanalysis: A case study of the speck distinguisher, in: Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings, Springer, 2022, pp. 809–829.

[9] A. Benamira, D. Gerault, T. Peyrin, Q. Q. Tan, A deeper look at machine learning-based cryptanalysis, in: Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I 40, Springer, 2021, pp. 805–835.

[10] A. Gohr, G. Leander, P. Neumann, An assessment of differential-neural distinguishers, Cryptology ePrint Archive.

[11] Z. Bao, J. Lu, Y. Yao, L. Zhang, More insight on deep learning-aided cryptanalysis, https://eprint.iacr.org/2023/1391 (2023).

[12] R. Beaulieu, D. Shors, J. Smith, S. Treatman-Clark, B. Weeks, L. Wingers, The SIMON and SPECK families of lightweight block ciphers, IACR Cryptol. ePrint Arch. (2013) 404.

[13] Z. Hou, J. Ren, S. Chen, Cryptanalysis of round-reduced simon32 based on deep learning, Cryptology ePrint Archive.

[14] H.-C. Su, X.-Y. Zhu, D. Ming, Polytopic attack on round-reduced simon32/64 using deep learning, in: Information Security and Cryptology: 16th International Conference, Inscrypt 2020, Guangzhou, China, December 11–14, 2020, Revised Selected Papers, Springer, 2021, pp. 3–20.

[15] E. Kaufmann, O. Cappé, A. Garivier, On bayesian upper confidence bounds for bandit problems, in: Artificial intelligence and statistics, PMLR, 2012, pp. 592–600.

[16] L. Zhang, Z. Wang, B. Wang, et al., Improving differential-neural cryptanalysis with inception, Cryptology ePrint Archive.