2025 International Conference on Intelligent Computing
July 26-29, Ningbo, China
https://www.ic-icc.cn/2025/index.php

Enhancing the Identification of Related-Key Neural Differential Distinguishers for SPECK32/64

Wanqing Wu[1,2], and Mengxuan Cheng[1]

[1] School of Cyber Security and computer, Hebei University, Baoding, Hebei 071002
[2] Key Laboratory on High Trusted Information System in Hebei Province (Hebei University), Baoding, Hebei 071002
cmxuan117@163.com

Abstract. Lightweight encryption algorithms play a vital role in securing communications for resource-constrained devices. As a prominent lightweight cipher, SPECK has attracted extensive security analyses. At ASIACRYPT 2023, Bao et al. introduced a related-key neural-network differential distinguisher capable of effectively distinguishing 9-round SPECK32/64 ciphertexts and integrated it into a (1+s+r+1) key-recovery framework to attack 14-round SPECK32/64. Inspired by their work, this paper presents a new related-key neural differential distinguisher for SPECK32/64, built upon a novel related-key processing method and an alternative network architecture, which significantly boosts the accuracy of distinguishing 10-round ciphertexts. Within the same (1+s+r+1) key-recovery framework, we employed our trained distinguisher to recover the key of 15-round SPECK32/64. The specific contributions are as follows: First, this paper introduces a novel related-key processing method, generating correlated subkey pairs for encrypting samples containing 64 plaintext pairs. Second, a related-key neural differential distinguisher was constructed based on the Inception module from GoogleNet and the DenseNet architecture. Experimental results demonstrate that the trained distinguisher achieves a recognition accuracy of 97.24% for 10-round ciphertexts, surpassing Bao et al.'s results by extending the recognizable rounds by one. Finally, leveraging the 10-round neural distinguisher, this paper successfully executed a key recovery attack on 15-round SPECK32/64. Analysis of error-bit distributions revealed a correct key recovery success rate of 98.67%.

Keywords: SPECK, Key Recovery Attack, Neural Differential Distinguisher, Related Key.

1 Introduction

With the rapid development of the Internet of Things (IoT) and embedded systems, traditional encryption algorithms such as DES and AES struggle to function effectively under constrained storage and computational resources. Consequently, lightweight encryption algorithms have emerged. The SPECK cipher, designed by the U.S. National Security Agency (NSA) [1], is a lightweight block cipher that exhibits strong performance in both hardware and software implementations. However, as modern

cryptographic attack techniques continue to evolve, evaluating the security of the SPECK algorithm has become critically important. In block cipher research, differential cryptanalysis remains one of the most effective analytical methods.

Differential cryptanalysis, proposed by Biham and Shamir [2] in 1991 to crack the DES cipher [3], is a primary method for attacking symmetric encryption algorithms. It infers key information by analyzing the relationship between input and output differences. Subsequent studies have explored SPECK variants and SPECK differential trails [4-6]. At CRYPTO 2019, Gohr [7] introduced neural differential distinguishers, combining neural networks (ResNet [8]) with differential cryptanalysis. For reduced-round SPECK32/64, neural distinguishers were successfully trained for 5–7 rounds. Compared to classical differential distinguishers, neural distinguishers capture richer feature information and achieve higher key recognition accuracy. The 7-round distinguisher achieved 61.6% accuracy, a 2.5% improvement over the classical distinguisher's 59.1%. This advancement propelled research on neural network-based cryptanalysis. In 2021, Chen et al. [9] extended the input plaintext pairs from one pair to multiple pairs, organizing them into input matrices based on bit positions. This approach improved Gohr's [7] 7-round neural distinguisher accuracy by 3.3%, reaching 64.9%. In 2022, Zhang et al. [10] proposed an extended data format $(X_r, X'_r, Y_r, Y'_r, Y_{r-1}, Y'_{r-1})$, incorporating information from the previous round into the neural network input. This modification increased the 7-round distinguisher's accuracy to 89.58%, with the 8-round distinguisher achieving 58.53%. In 2023, Yue et al. [11] introduced a new data structure $(Y_{r-1}, Y'_{r-1}, \mathrm{Z}, X_r, X'_r, Y_r, Y'_r)$, further boosting the 7-round neural distinguisher's accuracy to 97.13%. The same year, Liu et al. [12] proposed a novel input data generation method and two specialized input models, constructing distinguishers capable of identifying 8-round SPECK32/64 ciphertext differential features, achieving 65.02% accuracy for the 8-round distinguisher.

Subsequently, researchers integrated related-key differential attacks with neural networks to develop related-key neural differential distinguishers. Tcydenova et al. [13] utilized key differences to build related-key neural distinguishers, improving 8-round distinguisher accuracy to 84.84% and achieving 59.32% accuracy for a 9-round distinguisher. Bao et al. [14] analyzed high-probability differential characteristics of related keys, encrypting plaintext pairs with related keys conforming to specific differences, and using the resulting ciphertexts as inputs for neural distinguishers. Their work enhanced the 9-round distinguisher's accuracy to 77.26% and successfully trained a 10-round neural distinguisher with 56.43% accuracy.

Inspired by these advancements, this paper proposes a novel key processing method that integrates features of related-key and single-key scenarios to generate correlated subkey pairs for encrypting plaintext samples. For the neural network architecture, we design a new model based on DenseNet [15] and Inception modules [16]. The core module employs three parallel convolutional layers with varying kernel sizes, enabling the neural differential distinguisher to capture richer feature information from ciphertext samples, thereby achieving higher accuracy. Experimental results demonstrate related-key neural distinguishers for 9-round and 10-round SPECK with accuracies of 99.99% and 97.24%, respectively. Finally, we validate the correctness of the 10-round SPECK neural distinguisher by performing a key recovery attack on 15-round SPECK.

2 Preliminaries

2.1 SPECK Algorithm

SPECK is a lightweight block cipher designed by the U.S. National Security Agency (NSA) in 2013. SPECK employs a Feistel structure and utilizes four core operations: modular addition (⊞), XOR (⊕), right rotation (⋙), and left rotation (⋘).This paper primarily focuses on SPECK32/64, which consists of 22 round functions, with a block size of 32 bits, a key size of 64 bits, and a word length of 16 bits.

For the round function, given the i-th round input (X_i, Y_i) and subkey k_i, the corresponding output is (X_{i+1}, Y_{i+1}), where $0 \le i \le 21$. The detailed process is as follows:

$$\begin{cases} X_{i+1} = \left((X_i \ggg 7) \boxplus Y_i\right) \oplus k_i \\ Y_{i+1} = (Y_i \lll 2) \oplus X_{i+1} \end{cases}$$

The subkey generation algorithm aligns closely with the SPECK round function. The round function is illustrated in Fig. 1.

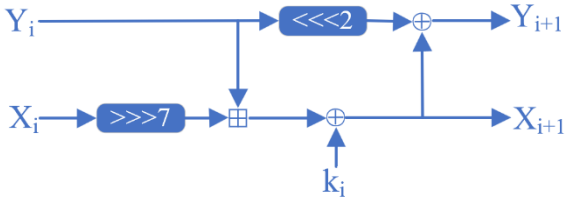

Fig. 1. Round function for SPECK cipher.

2.2 DenseNet

Deep neural networks face challenges such as vanishing gradients, inefficient feature reuse, and parameter redundancy. To address these issues, Huang et al. [15] proposed DenseNet, which introduces dense connectivity: each layer connects directly to all subsequent layers. This mechanism enhances gradient flow, maximizes feature reuse, and improves parameter efficiency and training stability.

DenseNet's dense connectivity mechanism can be divided into feature concatenation and feature reuse. Feature concatenation can be expressed mathematically. If the output formula of a traditional neural network at layer i is represented as:

$$L_i = H(L_{i-1})$$

, then the output formula of DenseNet at layer i can be expressed as:

$$L_i = H([L_0, L_1, \ldots, L_{i-1}])$$

where H(·) denotes a nonlinear transformation function, comprising a series of operations including Batch Normalization (BN), ReLU activation, Pooling, and Convolution

(Conv). The symbol [·] represents concatenation along the channel dimension, and L_i denotes the output of the *i*-th H function mapping from all preceding layers.

Feature reuse refers to the direct access of feature maps from all previous layers by each subsequent layer, facilitating the fusion of low-level and high-level features and thereby reducing redundant computations.

The core components of DenseNet's architecture are the Dense Block and Transition Layer. The workflow of a Dense Block is illustrated in Fig. 2.

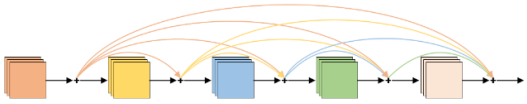

Fig. 2. Dense Block

Within a Dense Block, multiple densely connected layers are included. Each layer can directly access the feature maps from all preceding layers. The spatial dimensions of the feature maps remain unchanged, while the number of channels increases incrementally at a predefined growth rate.

The Transition Layer, positioned between two Dense Blocks, compresses the feature map dimensions and reduces the number of channels to prevent parameter explosion.

2.3 Inception Module

The Inception module first appeared in GoogleNet and serves as its core component, proposed by Christian Szegedy [16]. By introducing multi-scale convolution and pooling operations, this module enables the network to perform feature extraction across varying receptive fields, thereby enhancing the model's recognition capability and efficiency. The Inception module executes parallel convolutional operations with kernels of different sizes (e.g.,1×1, 3×3, and 5×5). The 1×1 convolution is used for dimensionality reduction to minimize parameters and computational costs, while the 3×3 and 5×5 convolutions capture features at different scales. These features are concatenated to form the input for the subsequent layer.

3 A New Method for Generating Related Keys

3.1 Analysis of Single-Key and Related-Key Encryption

Single-Key Encryption.
When encrypting plaintext pairs with a single key, the diffusion of differential characteristics in ciphertext pairs arises solely from the modular addition operations during encryption. The differential value calculations are shown in Equations (1) and (2). The differential value after the *i*-th round of encryption is:

$$\begin{aligned} X_{i+1} \oplus X'_{i+1} &= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus k_i \oplus \left((X'_i \ggg 7) \boxplus Y'_i\right) \oplus k_i \\ &= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus \left((X'_i \ggg 7) \boxplus Y'_i\right) \end{aligned} \tag{1}$$

$$\begin{aligned} Y_{i+1} \oplus Y'_{i+1} &= (Y_i \lll 2) \oplus X_{i+1} \oplus (Y'_i \lll 2) \oplus X'_{i+1} \\ &= \left((Y_i \oplus Y'_i) \lll 2\right) \oplus (X_{i+1} \oplus X'_{i+1}) \\ &= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus \left((X'_i \ggg 7) \boxplus Y'_i\right) \oplus \left((Y_i \oplus Y'_i) \lll 2\right) \end{aligned} \tag{2}$$

As the number of rounds increases, the diffusion of differential characteristics intensifies, making it progressively harder for neural networks to recognize ciphertext differentials. A schematic of ciphertext feature diffusion in single-key encryption is shown in Fig. 3.

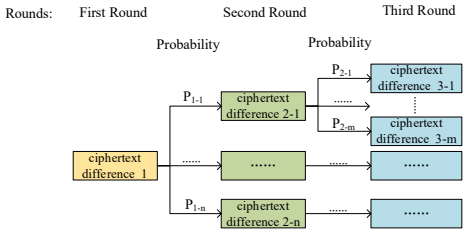

Fig. 3. Thumbnail of Ciphertext Feature Diffusion in Single-Key Encryption

Related-Key Encryption

Encrypting identical plaintexts with related subkeys (with a non-zero differential) produces ciphertext pairs whose differential values are influenced solely by the key differences. The differential value after the first round is calculated as follows:

$$\begin{aligned} X_{i+1} \oplus X'_{i+1} &= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus k_i \oplus \left((X_i \ggg 7) \boxplus Y_i\right) \oplus k'_i \\ &= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus \left((X_i \ggg 7) \boxplus Y_i\right) \oplus k_i \oplus k'_i \\ &= k_i \oplus k'_i \end{aligned} \tag{3}$$

$$\begin{aligned} Y_{i+1} \oplus Y'_{i+1} &= (Y_i \lll 2) \oplus X_{i+1} \oplus (Y_i \lll 2) \oplus X_{i+1} \\ &= \left((Y_i \oplus Y_i) \lll 2\right) \oplus (X_{i+1} \oplus X_{i+1}) \\ &= k_i \oplus k'_i \end{aligned} \tag{4}$$

However, in subsequent rounds of ciphertext pairs, the differential characteristics of the ciphertext propagate due to the modular addition operations applied to the differential values inherited from previous rounds. Simultaneously, the diffusion of differential characteristics in the related subkey generation process further amplifies this effect. Consequently, the differential values of the ciphertext pairs are influenced both by the modular addition operations and the differentials of the subkeys, as demonstrated in Equations (5) and (6). Under related-key encryption, the diffusion speed of ciphertext differential characteristics is significantly faster compared to single-key encryption.

$$\begin{aligned} X_{i+1} \oplus X'_{i+1} &= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus k_i \oplus \left((X'_i \ggg 7) \boxplus Y'_i\right) \oplus k_i \\ &= \left(\left((X_i \ggg 7) \boxplus Y_i\right) \oplus \left((X'_i \ggg 7) \boxplus Y'_i\right)\right) \oplus (k_i \oplus k'_i) \end{aligned} \tag{5}$$

$$Y_{i+1} \oplus Y'_{i+1} = (Y_i \lll 2) \oplus X_{i+1} \oplus (Y'_i \lll 2) \oplus X'_{i+1}$$
$$= \left((X_i \ggg 7) \boxplus Y_i\right) \oplus \left((X'_i \ggg 7) \boxplus Y'_i\right) \oplus ((Y_i \oplus Y'_i) \lll 2) \oplus (k_i \oplus k'_i) \quad (6)$$

When encrypting plaintext pairs with a zero differential using a master key pair with a differential value of Δ=0x0040/0000/0000/0000, the subkey generation algorithm ensures that the first three rounds of subkey differentials are zero. Combined with the zero plaintext differential, neither key nor ciphertext characteristics propagate during the first three encryption rounds. This design partially mitigates the accelerated diffusion of differential characteristics in subsequent rounds, allowing the neural differential distinguisher to effectively identify ciphertext features up to 8 rounds [15]. A schematic of ciphertext differential characteristic diffusion under related-key encryption is illustrated in Fig. 4.

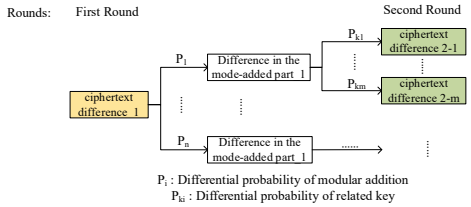

Fig. 4. Thumbnail of Ciphertext Feature Diffusion in Related-Key Encryption

3.2 New Subkey Generation Method

Building on the above analysis, this paper proposes a new method for generating related subkeys. A 64-bit master key is randomly selected, and a related master key is derived using the differential Δ=0x0040/0000/0000/0000. These two master keys form a related master key pair. Using the key schedule algorithm, the first three rounds of subkeys generated from this pair have zero differentials. A non-zero differential first appears in the fourth-round subkeys. After encrypting four rounds with the fourth-round subkey pair, the fifth-round ciphertext pair becomes the first to exhibit a differential. For the fifth and subsequent rounds, only one set of subkeys (generated from one master key) is retained and assigned to both subkey sequences. This ensures that subkey differentials from the fifth round onward are eliminated, thereby slowing the diffusion of ciphertext differentials under related-key conditions.

The specific process for generating related subkeys is as follows:

1. Set the differential value Δ=0x0040/0000/0000/0000. Randomly and uniformly select a 64-bit string as the master key K .
2. Compute the related master key K' using $K \oplus K' = \triangle$.
3. Apply the key schedule algorithm F to K for r rounds (r≥5), generating the corresponding subkeys $k_i (i = 0,1,2, \dots, r-1)$.
4. Apply the key schedule algorithm to K' for 4 rounds, generating subkeys $k'_i (i = 0,1,2,3)$. Starting from the 5th round, assign $k'_i = k_i (\mathrm{i} = 4,5,6, \dots, \mathrm{r}-1)$.

5. Organize the subkey pairs $(k_i, k_i')(i = 0,1,2,\ldots,r-1)$ to obtain the related master key pair(K, K') and its associated subkey sequences.

The complete related key generation workflow is illustrated in Fig. 5.

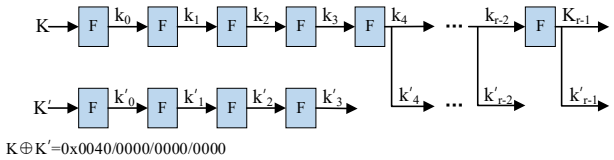

Fig. 5. Key Processing Procedure

4 Constructing Neural Differential Distinguishers

4.1 Building Input Data for Neural Differential Distinguishers

This paper adopts a multi-ciphertext-pair data structure as the input for neural differential distinguishers.

1. Randomly generate a training dataset containing 2^n samples. Each sample consists of mm uniformly distributed random plaintext pairs (P, P') with a differential value of 0, where P is 32 bits.
2. Assign a binary label vector T to the sample set. Randomly label half of the samples in the set as positive samples, denoted by T=1; label the other half as negative samples, denoted by T=0. In the negative samples, replace P' with other different randomly generated data.
3. Divide P evenly into two 16-bit parts, with the left part denoted as X_0 and the right part as Y_0. Apply the same process to the plaintext P'; thus, the plaintext pair (P, P') can be represented as (X_0, Y_0, X_0', Y_0'). Therefore, one sample can be expressed as in formula (7):

$$\begin{matrix}(X_{0\_1}, Y_{0\_1}, X'_{0\_1}, Y'_{0\_1}) \\ (X_{0\_2}, Y_{0\_2}, X'_{0\_2}, Y'_{0\_2}) \\ \cdots \\ (X_{0\_m}, Y_{0\_m}, X'_{0\_m}, Y'_{0\_m})\end{matrix} \tag{7}$$

   In the formula, the subscript of X_{i_j} denotes that i indicates encryption up to round i and j indicates the j-th plaintext/ciphertext pair in the sample.
4. According to the related subkey generation method proposed in Section 3, randomly generate 2^n related keys(K, K'), where each related key generates corresponding r-round related subkey pairs $(k_i, k_i')(i = 0,1,2,\ldots,r-1)$.
5. Introduce the r-round related subkey pairs (k_i, k_i') to encrypt the plaintext samples and obtain r rounds of ciphertext data. Then, based on the data structure proposed in [11], perform transformation operations on the ciphertext data to obtain the final input data samples for the neural differential discriminator, as shown in formula (8):

$$
\begin{aligned}
&(Y_{r-1\_1}, Y'_{r-1\_1}, \mathrm{Z\_1}, X_{r\_1}, Y_{r\_1}, X'_{r\_1}, Y'_{r\_1}) \\
&(Y_{r-1\_2}, Y'_{r-1\_2}, \mathrm{Z\_2}, X_{r\_2}, Y_{r\_2}, X'_{r\_2}, Y'_{r\_2}) \\
&\cdots \\
&(Y_{r-1\_m}, Y'_{r-1\_m}, \mathrm{Z\_m}, X_{r\_m}, Y_{r\_m}, X'_{r\_m}, Y'_{r\_m})
\end{aligned} \tag{8}
$$

The overall construction process of the input data structure is shown in the Fig. 6:

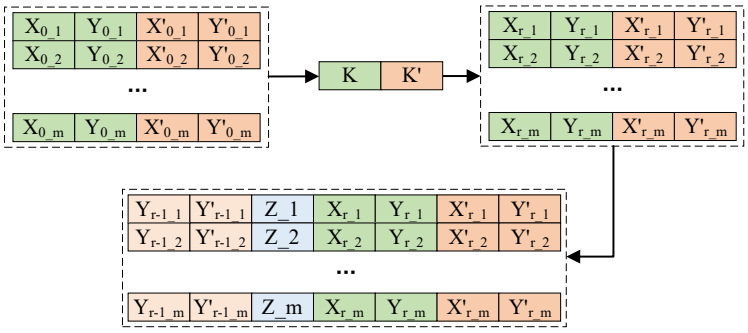

Fig. 6. Data structure

4.2 Network Architecture

This paper proposes a novel neural network architecture inspired by DenseNet and the Inception module from GoogleNet. The network consists of an input layer, an initial convolutional layer, Dense Blocks, and a prediction head. The overall framework is depicted in Fig. 7.

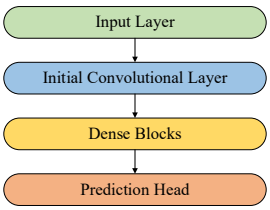

Fig. 7. Neural network architecture

Input Layer: This layer performs preliminary processing on the data input into the neural network. The samples are input in the form of one-dimensional data. The input layer receives these data and converts them into two-dimensional data with a shape of [W∗M, m], and then passes these two-dimensional data to the Initial Convolutional Layer. In [W∗M, m], W represents the bit-size of the basic data in the sample, M represents the number of basic data in the sample, and m represents the number of plaintext pairs in a sample.

Initial Convolutional Layer: This layer is used to extract low-level features from the ciphertext data. After obtaining the two-dimensional data from the Input Layer, a convolution with a 1×1 kernel is first performed. Then, the convolved results are batch

normalized, followed by processing with the "relu" activation function to obtain the output, which is then passed to the subsequent Dense Blocks. The number of filter channels corresponding to the Initial Convolutional Layer is 3Nf.

Dense Blocks: First, three convolutional layers with the same number of channels Nf and kernel sizes of 1×1, 3×3, and 8×8 are applied to the input data. Then, the results of these three convolutions are concatenated along the channel dimension to obtain merged data with 3Nf channels. The merged data is then processed by batch normalization and the "relu" activation function to yield intermediate data. This intermediate data is passed through a convolutional layer with a kernel size of 3×3 and 3Nf channels, followed by a normalization layer and a "relu" activation layer to obtain the final data. Finally, the input data of the Dense Block is concatenated with the final data obtained within the Dense Block to form the output of the Dense Block. The Dense Block is executed five times, with the output of the last block being passed to the subsequent module.

Prediction Head: Once the final output with differential features from the Dense Blocks is obtained, it is transformed into a one-dimensional format using the flatten function. This data is then input into two fully connected layers, each containing 1024 neurons, followed by normalization and "relu" activation. A dropout function is incorporated to discard some neurons, preventing overfitting and enhancing model robustness. Finally, a fully connected layer with a single neuron, activated by a sigmoid function, produces the final result.

The complete neural network process is shown in Fig. 8.

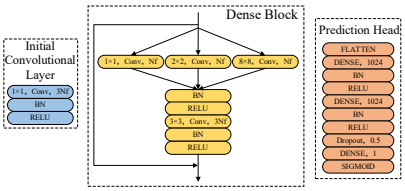

Fig. 8. Complete neural network

The network employs the Mean Squared Error (MSE) loss function and the ADAM optimizer. The learning rate is updated cyclically after each epoch according to Equation (9):

$$L_i = \alpha + \frac{(t - i) \bmod (t + 1)}{n} (\beta - \alpha) \tag{9}$$

4.3 Specific Parameter Settings

The following are the specific parameters used in the input data and the neural network.

5 Experimental Results

The experiments in this paper were conducted on the Ubuntu 20.04 operating system, utilizing Python 3.8 and TensorFlow 2.10.0 for code implementation. The hardware setup includes a machine equipped with two NVIDIA A4500 16GB GPUs, a 16-core Intel(R) Xeon(R) Gold 5222 CPU running at 3.80GHz, and 60GB of RAM.

Table 1. Specific Parameter Settings

Parameter	Symbol	Value
Number of samples	2^n	2^{20}
Plaintext pairs per sample	m	64
Bit size of basic data elements	W	16
Number of basic data elements	M	7
Filter channels	Nf	32
Minimum learning rate	α	10^{-4}
Maximum learning rate	β	4×10^{-3}
Cycle parameter	t	9

5.1 Experimental Results of the Neural Differential Distinguisher

Based on the methodology described in Section 4, a total of 2^{20} training samples and 2^{17} testing samples were randomly generated for the 9-round encrypted SPECK cipher, with each sample containing m=64 ciphertext pairs. These samples were subsequently fed into the 9-round neural differential distinguisher for training, the same procedure was applied for ten rounds. The results are illustrated in Fig. 9.

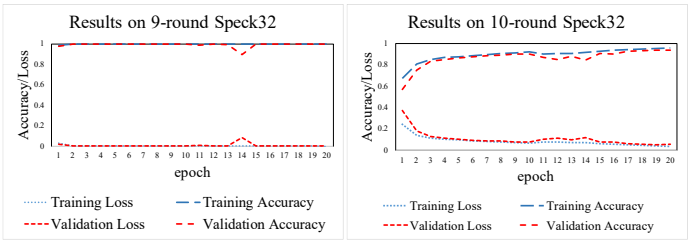

Fig. 9. Experimental results of the nine and ten rounds neural differential distinguisher

Fig. 9 depict the training performance of the 9-round and 10-round neural differential distinguishers. In these figures, the x-axis represents the epochs, while the y axis shows the accuracy and loss rates during training and validation. The lines illustrate the changes in accuracy and loss as the number of epochs increases. In the left figure, it is evident that the validation accuracy of the 9-round neural differential distinguisher reached 99.99%, with a corresponding loss rate of only 0.02%. In the right figure, the validation accuracy of the 10-round neural differential distinguisher achieved 97.24%, while the validation loss rate was 5.51%.

The final results of Bao et al. and this paper are summarized in Table 2. The comparison indicates an improvement in accuracy with the proposed method. Specifically, the accuracy of the 9-round distinguisher in this paper reached 99.99%, compared to 77.26% in Bao et al.'s findings. Additionally, the accuracy for the 10-round distinguisher was 97.24%, representing a 40.81% increase over Bao et al.'s result of 56.43%. While this paper trained an 11-round neural differential distinguisher, the results were less favorable, with an accuracy of only 53.64%.

Table 2. Comparison of the end result

SPECK's round	Accuracy of Bao's differentiators [14]	The accuracy of the differentiators in this paper
9	77.26%	99.99%
10	56.43%	97.24%
11	-	53.65%

5.2 Comparison of Different Encryption Methods

To demonstrate the advantages of the related key generation method proposed in this paper, we compared the impact of four encryption approaches on neural differential distinguisher accuracy. Using identical data structures and network architectures across experiments ensured that performance differences solely reflected encryption method variations. The evaluated approaches include:

1. Single-key encryption with plaintext differential ΔP=0x0040/0000.
2. Related-key encryption with ΔP=0 and key differential Δ=0x0040/0000/0000/0000.
3. Related-key encryption with ΔP=0 using keys conforming to specific differential trails (Bao et al. [14]).
4. The proposed method: related-key encryption with ΔP=0 using fully random keys generated via our approach.

The input data generated by these four methods were used to train the neural differential distinguisher, and the results are presented in Table 3.

Table 3. Results of 10 Rounds of Training for Different Encryption Schemes

Input Data Encryption Mode	10-Round Distinguisher Accuracy
Single-key [7]	50.52%
Related-key [13]	50.19%
Bao [14]	74.14%
Ours	97.24%

From Table 3, it can be observed that data encrypted using either single-key or conventional related-key methods fails to exhibit detectable differential features for the 10-round neural distinguisher, with accuracies close to random guessing (approximately 50%). While Bao et al.'s encryption method achieves a 74.14% accuracy for the 10-round neural distinguisher by employing related keys adhering to specific differential trails, this approach narrows the key space by relying on not fully random key selection,

limiting its applicability to constrained scenarios. In contrast, our method generates fully random keys while achieving a 10-round neural distinguisher accuracy of 97.24%. This demonstrates that the keys generated in this work offer broader applicability, more effectively mitigate ciphertext differential diffusion, and enable neural distinguishers to identify high-round ciphertext features with greater reliability.

5.3 Comparison of Different Data Structures

In Bao et al.'s experiments, the data format was $(Y_{r-1}, Y'_{r-1}, X_r, X'_r, Y_r, Y'_r)$, while this paper adopts $(Y_{r-1}, Y'_{r-1}, \mathrm{Z}, X_r, X'_r, Y_r, Y'_r)$. To evaluate the impact of data structures on neural distinguisher accuracy, a comparative experiment was conducted for 10-round neural distinguishers using three structures: the proposed structure, Bao et al.'s structure, and Gohr's structure (X_r, X'_r, Y_r, Y'_r). All experiments utilized the neural network proposed in this paper, with other parameters held constant. Results are shown in Table 4.

Table 4. Results of 10 rounds of training for different data formats

Data Structures	Source	Accuracy of neural differential distinguishers
$(Y_{r-1}, Y'_{r-1}, \mathrm{Z}, X_r, X'_r, Y_r, Y'_r)$	ours	97.24%
$(Y_{r-1}, Y'_{r-1}, X_r, X'_r, Y_r, Y'_r)$	Bao [14]	88.76%
(X_r, X'_r, Y_r, Y'_r)	Gohr [7]	50.26%

Experimental results confirm that the parameter Z in the proposed structure preserves features from the differential values of X_{r-1} and X'_{r-1} unaffected by subkey k [11]. This design achieves the highest accuracy 97.24% among the three structures, surpassing Bao et al.'s method by 8.48%. In contrast, Gohr's structure achieved only 50.26% accuracy, failing to identify meaningful features. Thus, the proposed data structure demonstrates superior performance in enhancing neural distinguisher accuracy.

5.4 Comparison of Different Neural Networks

To evaluate the impact of neural network architectures on the performance of neural differential distinguishers, this paper compares three networks under identical experimental conditions. All networks use the data processing method proposed in this paper, adopt the same data structure, and adjust network parameters to optimize performance. Comparative experiments for 10-round neural distinguishers were conducted against the networks proposed by Bao et al. [14] and Yue et al. [11]. The accuracy results are summarized in Table 5

As shown in Table 5, the proposed network achieves 97.24% accuracy for 10-round distinguishers, outperforming the other two networks by at least 5%. This demonstrates that the proposed architecture significantly enhances the recognition of ciphertext differential features. For scenarios demanding higher accuracy, the proposed network is the superior choice.

Through these comparative experiments, the novel related-key generation method and network architecture proposed in this paper collectively advance the state-of-the-art in neural differential distinguishers. The recognition accuracy for 10-round SPECK32/64 reaches 97.24%, surpassing the previously highest reported 9-round accuracy of 77.26% [14]. This represents a one-round improvement in attack capability while achieving a substantial accuracy gain.

Table 5. Results of 10 rounds of training for different neural networks

Source	Accuracy of neural differential distinguishers
Bao[14]	93.57%
Yue[11]	94.22%
ours	97.24%

6 Key Recovery Attack

To validate the effectiveness of the related-key neural differential distinguisher described in Section 5, this chapter conducts a 15-round key recovery experiment using the trained 10-round neural distinguisher. The following table defines key symbols used in the key recovery process:

Table 6. Symbol Notation in Key Recovery

Symbol	Explanation
n	Sample size
m	Number of plaintext pairs per sample
2^{-p}	Propagation probability of the related-key differential trail: Δ=0x0200/0080/0011/4a00→Δ=0x0040/0000/0000/0000
2^{-q}	Propagation probability of the classical differential ΔS→ΔP
c	A positive constant
△S	Input differential for the classical distinguisher: Δ=0x02a1/4001
△P	Input differential for the neural distinguisher: Δ=0x0000/0000
△k	Differential of the first-round related subkey pair: Δ=0x0000/4a00

6.1 Key Recovery Attack on Reduced-Round SPECK32/64

This experiment uses a (1 + s + r + 1)-round key recovery attack framework, which consists of one free round, *s* rounds of classical distinguisher, *r* rounds of neural differential distinguisher, and one round of key guessing. Under the related-key condition, it is necessary to determine an appropriate key differential trail and specify the key differentials for the initial rounds in the experiment. The key differential trail is shown in Table 7.

In the first free round, since the encryption process in the first round is influenced by the related key, the decryption key is not (0, 0) but instead (0,0⊕△k) in order to

recover the plaintext. This adjustment effectively eliminates the impact of the first-round subkey pair on the ciphertext differential of the first round.

Table 7. Related-key differential trails

Round(r)	Differential in Key	log2 Pr
0	(0200,0080,0011,4a00)	
1	(2800,0200,0080,0001)	-4
2	(0000,2800,0200,0004)	-1
3	(0000,0000,2800,0010)	-1
4	(0040,0000,0000,0000)	-2
5	(0000,0040,0000,0000)	0
6	(0000,0000,0040,0000)	0
7	(8000,0000,0000,8000)	0

Following the free round, there are *s* rounds of a classical distinguisher, where *s* is set to 3. The inclusion of the classical distinguisher extends the overall number of rounds in the key-recovery process. The classical distinguisher employs related subkey pairs that adhere to the differential trajectory for rounds 1–3 as specified in Table 7, with an input ciphertext differential of ΔS and an output ciphertext differential of ΔP.

For the *r* rounds corresponding to the related-key neural differential distinguisher, the first four rounds conform to the key differential trajectory for rounds 4–7 in Table 7, while also satisfying the requirements of the proposed key generation algorithm for the first four rounds. By integrating this algorithm, the keys for subsequent rounds and the associated ciphertext data can be derived.

Finally, in the last guessing round, a guessed key is used to perform one round of decryption on the generated ciphertext data. The decryption results are then fed into the neural differential distinguisher, which, based on its judgment, yields a guessed key that is either the closest to the correct key or is the correct key outright.

Complete workflow of key recovery attack:

1. Create a candidate subkey score list variable $d \leftarrow$(None). The list *d* records the cumulative scores for each candidate subkey during key recovery, with a size of 2^{16}. The index corresponds to the subkey value.
2. Uniformly sample n × $(2^p + c)$ random related master key pairs with differential values △=0x0200/0080/0011/4a00. Filter *n* pairs satisfying the differential trail in Table 7 to serve as related keys for encryption.
3. For the *n* master key pairs, execute the key scheduling algorithm to generate related subkey pairs for the first four rounds. Using the method proposed in this paper, extend these to produce related subkey pairs for rounds 5–15, yielding full 15-round related subkey pairs.
4. Uniformly sample $n \times m \times (2^q + c)$ random data pairs with input differential ΔS. Perform single-round decryption using the related subkeys $(0,0 \oplus \triangle k)$ to derive plaintext pairs.

5. Encrypt the plaintext pairs for four rounds using the related subkey pairs from Step 3. Filter $n \times m$ pairs with ciphertext difference result (0,0). Group every consecutive m pairs into a sample, yielding n plaintext samples for key recovery.
6. Encrypt the n plaintext samples using the n related subkey pairs to generate n ciphertext samples.
7. For j from 0 to 2^{16}:
   a. Decrypt ciphertext samples for one round using subkey j.
   b. Apply linear transformation to decrypted results to obtain input for the distinguisher.
   c. Feed samples into the related-key differential neural distinguisher to compute prediction scores.
   d. Sum all sample scores as the total score for subkey j, recorded as $d[j]$.
8. Sort list d in descending order and output the top 10 candidate subkeys with highest scores.

6.2 Analysis of Key Recovery Results

In total, 150 key recovery experiments were conducted, with each experiment involving 128 plaintext pairs. If the correct key was found among the top ten candidates, it was considered a successful guess. Out of 150 attempts, all were successful, resulting in a 100% success rate (with 94.67% of guesses in the top five), as detailed in Table 8. Given that the criteria for key guessing success are not fixed, the corresponding success rate may vary. Thus, the success rate in this section serves merely as a reference for the effectiveness of the key recovery algorithm.

Table 8. Ranking of the correct subkeys

Rankings	Number of experiments	Rankings	Number of experiments
1	31	6	4
2	46	7	2
3	34	8	1
4	26	9	1
5	5	others	0

Additionally, this section analyzes the accuracy of each bit in all results, as well as the number of erroneous bits and their distribution in the individual guessed subkeys. The bits are designated from right to left as k0 to k15. The candidate subkey with the highest score across all results is selected as the final guessed key. If the number of incorrect bits is fewer than three, the key guess is deemed successful. The final count of erroneous bits can be eliminated through exhaustive methods. The analysis results are presented in Tables 9 and 10.

Table 9 demonstrates that the differential neural network distinguisher proposed in this paper successfully identifies 148 subkeys over 15 encryption rounds with 98.67% accuracy, allowing no more than two erroneous bits in subkey guesses.

Table 10 reveals that only four bits positions–k_7, k_8, k_{14}, k_{15}–exhibit accuracy rates below 100%. Specifically, k_7 and k_8 show accuracy rates exceeding 95%, while k_{14} and k_{15} have accuracy rates between 45% and 50%.

Table 9. The number of error bits in the guessed subkey

Number of error bits	Number of experiments
0	31
1	76
2	41
others	2

Table 10. The bit accuracy rate of the guessed subkey

Subkey bits	Accuracy	Subkey bits	Accuracy
k_{15}	48%	k_7	96%
k_{14}	47%	k_6	1
k_{13}	1	k_5	1
k_{12}	1	k_4	1
k_{11}	1	k_3	1
k_{10}	1	k_2	1
k_9	1	k_1	1
k_8	99%	k_0	1

Combining the insights from Tables 9 and 10 suggests that the one to two erroneous bits in the guessed subkey predominantly occur at k_{14} and k_{15}, with some instances at k_7 and k_8. Consequently, further analysis of the distribution of erroneous bits in the guessed subkey is presented in Fig. 10.

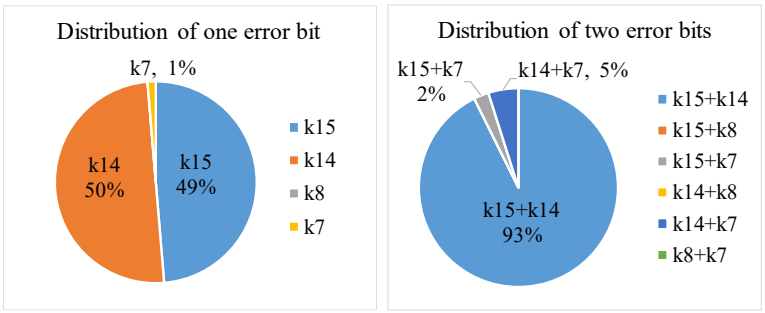

Fig. 10. The distribution of error bits in the guessed subkey

Fig.10 indicates that in the guessed subkeys with one erroneous bit, the errors are primarily concentrated between k15 and k14, accounting for 49% and 50% of errors, respectively, totaling 99% of all errors. In the cases of guessed subkeys with two erroneous bits, only three combinations of errors were observed: k15 + k14, k15 + k7 and k14 + k7, with k15 + k14 comprising the majority at 93%.

In summary, during the final exhaustive phase of key recovery attacks, it is advantageous to prioritize the high probability cases of k_{15}, k_{14}, $k_{15} + k_{14}$ to efficiently identify the correct subkey.

7 Conclusion

This paper proposes a novel key processing procedure and a neural network model. The proposed key generation methodology integrates characteristics of both related-key and single-key scenarios, eliminating the need to separately analyze plaintext differentials or key differentials. Compared to existing network architectures, our model captures more primitive and discriminative features. A series of comparative experiments systematically validates the advantages of the proposed key processing mechanism and neural architecture in training neural distinguishers. Furthermore, we successfully applied the trained related-key differential neural distinguisher to perform a key recovery attack on 15-round SPECK32/64. By integrating candidate key ranking and error bit analysis, the achieved key recovery correctness rate reaches 98.67%.

Throughout the experimental process, variations in the data structure also impact the final outcomes. In future work, we will explore new data structures and develop more efficient neural network training techniques to further boost the accuracy of the neural differential distinguisher, and we will also investigate additional lightweight ciphers.

References

1. Beaulieu R, Shors D, Smith J, et al. The simon and speck families of lightweight block ciphers[J]. IACR Cryptology ePrint Archive,2013,404. https://eprint.iacr.org/2013/404.pdf
2. Biham E, Shamir A. Differential cryptanalysis of DES-like cryptosystems[J]. Journal of CRVPTOLOGY, 1991,4(1):3-27
3. FIPS PUB. Data Encryption Standard (DES); NIST, 1999. Available online: https://csrc.nist.gov/csrc/media/publications/fips/46/3/archive/1999-10-25/documents/fips46-3.pdf (accessed on 27 April 2023).
4. Abed F, List E, Lucks S, et al. Differential cryptanalysis of round-reduced simon and speck[C]. International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, 2014: 525-545
5. Biryuköv A, Roy A, Velichkov V. Differential analysis of block ciphers simon and speck[C]. International Workshop on Fast Software Encryption. Springer, Berlin, Heidelberg, 2014: 546-570.
6. Kölbl S, Leander G, Tiessen T. Observations on the simon block cipher family[C]. Annual Cryptology Conference. Springer, Berlin, Heidelberg, 2015: 161-185
7. Ghor A. Improving attacks on round reduced speck32/64 using deep learning[C]. Advances in Cryptology—CRYPTO 2019, Part II. Springer Cham, 2019: 150–179
8. Kaiming He, xiangyu Zhang, et al. Deep Residual Learning for Image Recognition[C]. Conference on Computer Vision and Pattern Recognition, 2016.
9. Yi Chen, Yantian Shen, Hongbo Yu, et al. A new neural distinguisher considering features derived from multiple ciphertext pairs[J]. Cryptology ePrint Archive, Paper 2021/310, 2021.

10. Liu Zhang, Zilong Wang, Baocang Wang, et al. Improving Differential-Neural Cryptanalysis with Inception Blocks[J]. Cryptology ePrint Archive, 2022.
11. Xiaoteng Yue, Wanqing Wu. Improved Neural Differential Distinguisher Model for Lightweight Cipher Speck[J]. Applied Sciences, 2023.
12. JiaShuo Liu, JiongJiong Ren, et al.Improved neural distinguishers with multi-round and multi-splicing construction[J], Journal of Information Security and Applications, 2023:2214-2126
13. Erzhena Tcydenova, Byoungjin Seok, et al. Related-key Neural Distinguisher on Block Ciphers SPECK-32/64, HIGHT and GOST[J]. Journal of Platform Technology, 2023:72-84
14. Zhenzhen Bao, Jinyu Lu, et al. More Insight on Deep Learning-aided Cryptanalysis[C]. Advances in Cryptology – ASIACRYPT 2023,2023:436-467
15. Gao Huang, Zhuang Liu, et al. Densely Connected Convolutional Networks[J]. Conference on Computer Vision and Pattern Recognition (CVPR),2017.
16. Christian Szegedy, Wei Liu, et al. Going Deeper with Convolutions[C]. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9