Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 16, No. 1, 2025

The Implementation of Neural Network Encryption Algorithms for Side-Channel Attack Protection

Bo Chen, Yi Wang*

Chengdu University of Information Science and Technology, Chengdu, 610000, Sichuan, China
* **Corresponding author**: Yi Wang (Email: wangyi1177@126.com)

Abstract: With the rapid development of deep learning in the field of side-channel analysis, neural network-trained encryption algorithms have demonstrated numerous advantages, providing novel ideas for resisting side-channel attacks. We implemented a bit neural network-based block encryption scheme resistant to side-channel attacks. Experimental verification shows that the complete algorithm combined with this scheme exhibits correctness, reliability, efficiency, and resistance to side-channel attacks. This scheme has two significant advantages: First, bit networks can achieve functions that multilayer perceptrons (MLPs) cannot perform. For example, using the AES encryption algorithm, we successfully reduced the column mixing network loss during MLP training from 0.25 to 0. Second, bit networks can integrate with MLPs without intermediate value leakage issues. Once combined with MLPs, the generalization capability of the AES round operation model is significantly enhanced, while ensuring that the number

Keywords: BitNet; Multilayer Perceptron; Block Cipher Algorithm.

1. Introduction

Side-Channel Analysis (SCA)[1][2], proposed in the mid-1990s, has long been used by attackers as a crucial means to crack encryption system keys. It exploits physical signals (such as power consumption and electromagnetic radiation) leaked during system execution to compute intermediate values related to keys and ciphertexts, thereby recovering secret data like keys.[3][4][5] The main countermeasures against side-channel attacks are hiding and masking. The former reduces side-channel leakage by lowering the signal-to-noise ratio, while the latter introduces random masks to render intermediate values obtained by attackers invalid.

With the development of deep learning in side-channel analysis, although most research focuses on attacks, countermeasures against side-channel attacks continue to evolve [6] [7]. Picek et al. [8] [9] proposed a method to mislead neural network models into generating incorrect predictions by adding noise to attack samples. However, accurately distinguishing between training samples and attack samples remains an open question. Meanwhile, Krautter et al. [10] used deep neural networks to train substitution functions for AES S-boxes, ensuring constant computation cycles during neural network forward propagation to enhance resistance against correlation power analysis attacks, offering new insights for defending against side-channel attacks.

Inspired by Krautter et al., directly training encryption algorithms using neural networks presents multiple advantages. First, this approach maintains constant computation cycles during forward propagation, effectively reducing risks from Correlation Power Analysis (CPA) attacks. Second, the black-box nature of neural networks [8] ensures attackers cannot infer correct keys even if intermediate values are obtained, further enhancing system security. However, directly training encryption networks remains impractical in real-world scenarios.

For years, researchers have focused on enabling neural network inputs, weights, and outputs to be converted from continuous values to simpler fixed-point or binary representations. Abadi et al. [10] proposed a mapping method using trigonometric functions to transform (0;1) values to (0;π) before remapping them back to (0;1) via inverse functions and rounding operations. This method excels in single-byte XOR operations, generating compact networks with 100% accuracy. However, accuracy declines with multi-byte XOR operations, and it cannot handle nonlinear operations like S-box substitutions. Courbariaux et al. [11] introduced weight binarization, mapping parameters to [-1,1] while using continuous weights during backpropagation and parameter updates.

Kim et al. [12] proposed Bitwise Neural Networks (BNN), replacing floating/fixed-point operations with efficient bitwise operations. Their training process involves two phases: real-value training using continuous values constrained to (-1,1) via tanh functions, followed by discretizing outputs and trinarizing weights to (-1,0,1) using sign functions while maintaining continuous gradients and updates. For an L-hidden-layer BNN, forward propagation processes input data through linear transformations Z_L and sign(x) functions across layers to generate outputs A_L. Its forward propagation process can be expressed as:

$$Z_L = W_L A_{L-1} + b_L \tag{1}$$

$$A_L = \sigma(Z_L) = \mathrm{sign}(Z_L) \tag{2}$$

$$A_0 = X \tag{3}$$

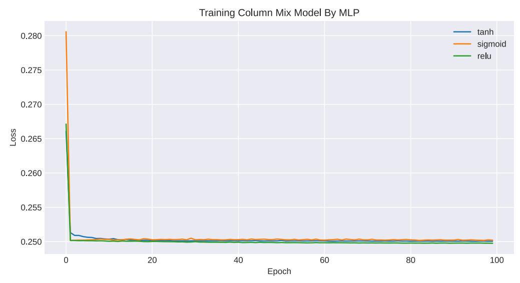
Among them, X and A_L represent the input data matrix and the output matrix of the L-th neural network layer respectively. W_L and b_L denote the weight matrix and bias vector of the L-th layer, which are implemented using 0/1 binary numbers during computation. The activation value A_L is equivalent to determining whether the linear computed value Z_L exceeds half of the input unit count plus one.

The BNN real-value computation phase employs Multilayer Perceptron (MLP) as the primary model architecture. Although MLP inherits strong generalization capabilities, certain inherent limitations remain evident, including the sensitivity of gradient descent and error backpropagation mechanisms to initial parameter selection. Particularly in our study of training column mixing functions in AES algorithms using MLP and Bitnet, we observed that even though column mixing operation is inherently linear (where input data multiplied by specific weight matrices directly yields mixing results), MLP still fails to effectively learn this process. We hypothesize that this may be due to current common initialization methods like Xavier and He initialization failing to adequately meet the requirements of column mixing training in neural networks.

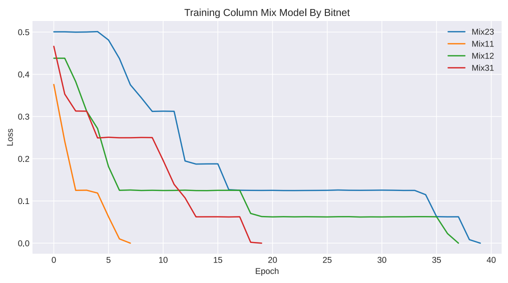
Additionally, we found that the implementation methods of various functional sub-networks within MLP architectures cannot be conveniently combined to form complete cryptographic networks. This reveals existing limitations of MLP networks in handling complex cryptographic algorithm structures. Therefore, we propose the necessity to further explore more effective initialization methods for column mixing operations and seek more suitable neural network architectures to build efficient cryptographic networks.

Leveraging the black-box nature of neural networks, we propose a reliable symmetric encryption scheme that effectively resists side-channel intermediate value leakage attacks. This scheme abandons traditional gradient-based error propagation in favor of a novel voting mechanism: it learns and simulates symmetric encryption algorithms by minimizing the voting values of labels on neural network hidden layers. Taking AES as an example, we validated this method's effectiveness, demonstrating that trained networks ensure encryption reliability and correctness while preventing intermediate value leakage. Specifically, we trained functional networks for each major step of symmetric encryption (e.g., AES S-box substitution, row shifting, column mixing, and round key addition), then skillfully combined them to construct a complete encryption algorithm neural network that prevents intermediate value leakage.

We propose a deep neural network training method without gradient descent that achieves functional networks unattainable by gradient-based methods, such as bitwise neural networks for AES column mixing. The core of this method lies in using voting and weight flipping for network training. Although weight flipping introduces randomness, the final results remain deterministic. Considering the model characteristics introduced by activation functions, bitwise networks demonstrate higher computational efficiency and unique generalization capabilities compared to standard MLP. Moreover, bitwise network layer weights can be combined with MLP layers, facilitating the integration of various functional networks in encryption algorithms.



(1a)



(1b)

Figure 1. These two experiments demonstrate the limitations of multilayer perceptrons (MLPs). In Figure (1a), we defined a 5-hidden-layer MLP and trained the column-mixing function model using different activation functions. In Figure (1b), our defined Bitnet successfully trained the column-mixing function model.

2. Bit Network

2.1. Section Headings

2.1.1. Sub-section Headings

In this section, we only use bit networks to train 0/1 bit datasets with linear mapping relationships, achieving functional networks that MLPs cannot train. Taking the AES algorithm as an example, we explain how to define bit neural networks to train column mixing functional networks. By comparing the difference between labels and bit network output values to vote on hidden layer weights, then flipping weights and propagating vote counts backward based on voting results, we call this process the voting mechanism. Through this mechanism, we successfully trained four required functional sub-networks for the column mixing functional network, all of whose weights are 0/1.

2.2. Bit Network Forward Propagation

As shown in Figure 2, bit networks significantly reduce model storage space and improve operational efficiency through bitwise operations by representing both weights and activation values as 1-bit binary numbers (0 or 1).

However, unlike BNNs, bit networks employ the modulo-2 function as the activation function. This choice ensures hidden layer activations remain strictly 0 or 1 while also forming the core definition of bit networks, endowing them with critical properties. For instance, enabling the combination of multiple small-function network weights into large-function network weights. These characteristics will be proven in Section 3.2.

$$\sigma(Z) = Z \bmod 2.$$

2.3. Bit Network Voting Mechanism

In order to achieve bit network back propagation to reduce errors and release its unique generalization potential, the traditional gradient descent algorithm is no longer applicable. This paper proposes a voting-based error back propagation mechanism. As shown in Figure 2, this mechanism uses a unique error function

$$\text{Loss} = 2\left| y_{\text{label}} - A_L \right| - 1.$$

Assume that the bit network we define has L hidden layers $T_i(*):\mathbb{B}^{n_{i-1}}\to\mathbb{B}^{n_i}$, so the activation value of the Lth layer is expressed as $A_L \in \mathbb{B}^{n_i \times m}$, where $i \in \{1,2,\dots,L\}$, m represents the number of samples. We calculate the error value of the output layer through the Loss function and use it as the voting result of the last layer 〖$\mathrm{Grad}_L \in \{-1,1\}^{m\times n_i}$, which are divided into two categories: approval (1) and opposition (-1). A sample can only vote on the weight of the current layer when the output value of the previous layer is 1. This is because when the output value is 0, the result of multiplication with the weight matrix has no effect on the result of the current layer. Finally, we count the voting results of all samples for each layer $V_L^{(*)} \in \mathbb{B}^{n_{i-1}\times n_i}$, and take the minimum number of votes as the learning goal:

$$V_L^{(*)} = \underset{V_L}{\operatorname{argmin}} \sum_{k=0}^{m} A_{L-1}^{(i,k)} \mathrm{Grad}_L^{(k,j)} \quad . \qquad (4)$$

Where m is the number of samples, $A_{L-1}^{(i,k)}$ and $\mathrm{Grad}_L^{(k,j)}$ represent the activation value and back propagation error value of the k-th sample at the i-th node and j-th node on the corresponding layer respectively, and their product is composed of {0,-1,1}, which represents the voting result of the weight matrix element $W_L^{(i,j)}$. The parameter update process will flip the weight 0-1 according to the voting result $V_L^{(*)}$. The flipped weight $W_L^{new} \in \mathbb{B}^{n_{i-1}\times n_i}$ can be expressed as

$$W_L^{new} = W_L^{old} \oplus \mathrm{Mask}$$

$$\mathrm{Mask} = \begin{cases} 1 \text{ if } V_L^{(i,j)} \text{is the one of } \max(0, V_L) \text{ for middle layers} \\ 1 \text{ if } V_L^{(i,j)} \text{is the one of } \max\left(0, V_L^{(:,j)}\right) \text{ for last layer} \\ 0 \text{ otherwise} \end{cases}$$

Among them, the $\mathrm{Mask} \in \mathbb{B}^{n_{i-1}\times n_i}$ matrix is a mask matrix composed of 0 and 1, which is used to flip one or more weight values of the weight matrix W_L^{old} before updating from 0 to 1. For the discussion of the number of weights updated in the middle layer, we compared the training of the same XOR network training data set, analyzed the training of updating one weight and updating an odd number of weights, and found that the two have little effect on the number of iterations. For the convenience of subsequent encoding, we fix the number of updated weights to 1.

The flipped new weight W_L^{new} will be applied to the back propagation of the previous layer. That is, if and only if the new weight is 1, the vote of the sample in the current layer will be passed to the previous layer. This is because the result of multiplying the 0 weight with the activation value of the previous layer has no effect on the result of the next layer. Therefore, the back propagation gradient calculation formula is

$$\mathrm{Grad}_{L-1} = \mathrm{Grad}_L W_L^{new}$$

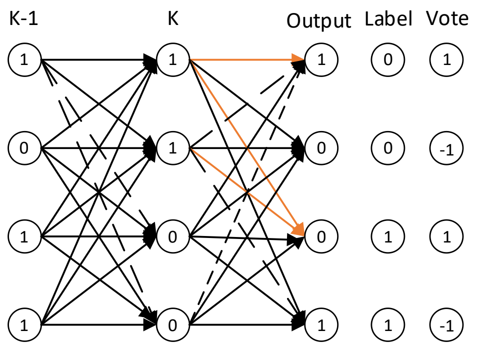


Figure 2. Bit network

2.4. Bit Network Implementation of Column Mixing

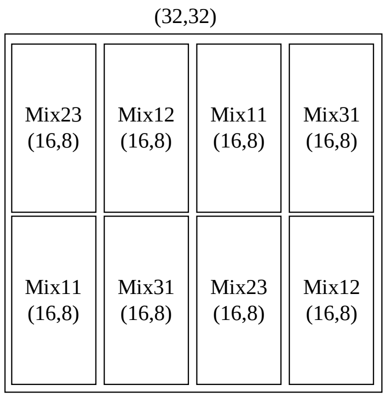
Column mixing transformation is a basic operation of the AES encryption algorithm. It performs a linear transformation on each column of the state matrix so that each byte is related to other bytes in the column, thereby increasing the obfuscation effect. Column mixing transformation can be implemented by multiplying a 4×4 constant matrix with each column of the state matrix, where the multiplication and addition involved in matrix multiplication are all performed on the finite field $GF(2^8)$.

$$\begin{bmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{bmatrix} \otimes \begin{bmatrix} B_1 & B_5 & B_9 & B_{13} \\ B_2 & B_6 & B_{10} & B_{14} \\ B_3 & B_7 & B_{11} & B_{15} \\ B_4 & B_8 & B_{12} & B_{16} \end{bmatrix} = \begin{bmatrix} B_1' & B_5' & B_9' & B_{13}' \\ B_2' & B_6' & B_{10}' & B_{14}' \\ B_3' & B_7' & B_{11}' & B_{15}' \\ B_4' & B_8' & B_{12}' & B_{16}' \end{bmatrix}$$

Unfortunately, we have shown through many experiments that it is not possible to directly use MLP networks and bit networks to implement column mixing transformations. However, through observation, we found that under the premise that the state matrix is unknown, by reasonably assuming that the state matrix follows a random distribution, we can use random variables (B_i, B_j) to represent the two bytes of 16-bit binary numbers in a column of the state matrix. We found that when the constant matrix is multiplied by the state matrix, there are four cases of modulo 2 multiplication:

$$\begin{bmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{bmatrix} \otimes \begin{bmatrix} B_i \\ B_j \end{bmatrix}$$

（3a）



(3b)

Figure 3. By decomposing the column mixing constant matrix function into 4 independent sub-functions. In Figure (3a), we have marked the 4 possible product situations. In Figure (3b), we have spliced a column of plaintext through the sub-function network weight matrix to perform the column mixing operation.

Therefore, we can exhaustively enumerate all possible two-byte (B_i, B_j) byte 16-bit binary numbers of the state matrix as input samples, and use the modulo-2 multiplication results of (B_i, B_j) and 4 constant vectors as output values for training, and obtain 4 sub-function bit network models when 2 bytes perform $GF(2^8)$ operations. According to the activation value after splicing the bit network weights, it is equivalent to performing an XOR operation, and further splicing the column mixing parameter matrix Mix= (Mix23, Mix11, Mix12, Mix31) of any column of plaintext (4 bytes) in the state matrix.

For the discussion of the correctness of the column mixing model, we can further combine the parameter matrix of the column mixing operation process of 4 columns of plaintext (16 bytes). In order to more intuitively show the operation process of the state matrix when performing column mixing, we draw the following operation flow chart:

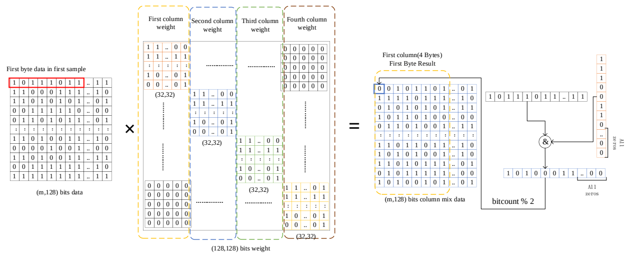


Figure 4. Column mixing matrix operation

2.5. Comparison between Multilayer Perceptron and BitNet

Compared with the traditional multilayer perceptron (MLP), the basic operations of BitNet during training are all fixed-point calculations, so it has obvious computational advantages. In addition, since BitNet uses modulo 2 function as activation function, the model trained by this network can be functionally spliced to achieve more complete network functions (this feature can be used to prove the reliability of the model, and the reliability of the large function model can be deduced by proving the reliability of the small function model). At the same time, the trained model can merge any two adjacent layers of weights into a network layer, which is conducive to model compression. Finally, the BitNet layer can be used in combination with the traditional MLP layer, which greatly expands the application scope of the BitNet. (The proof of relevant characteristics will be explained in Section 4.1)

However, since the weights of the BitNet and the output values of the intermediate layer are both 0/1, the BitNet can only be trained on data sets with a certain linear relationship, which limits the application scenarios of the BitNet. However, since the BitNet layer can be used in combination with the MLP layer, its potential application value is still worth exploring.

3. Assembling encryption algorithms

In this section, we will demonstrate how to combine the bit network with the multi-layer perceptron (MLP) to build a complete encryption algorithm. Taking the AES algorithm as an example, we train the round key encryption operation and the S-box replacement operation together into a network to ensure that the intermediate calculation process is not affected by the side channel leakage attack. In addition, we also combine the bit network model with the MLP model to ensure that the MLP model further expands the application scope of the bit network while training the column obfuscation function.

3.1. Joint training of bit network weights and MLP

Since the first XOR process of the plaintext and the key in the AES encryption process is usually the hardest hit area by attacks, we need to encrypt the XOR operation. On the one hand, in order to realize a single-byte S-box replacement subnetwork, we use the MLP neural network and use all possible single-byte 8-bit binary numbers as input samples and the corresponding new bytes replaced by the S-box as the output value for training, so as to obtain a reliable S-box replacement function network. On the other hand, we use the bit network to train an XOR network model with two intermediate layers and split the two weight matrices into three blocks, as shown in Figure 2 Xor network parameter structure.

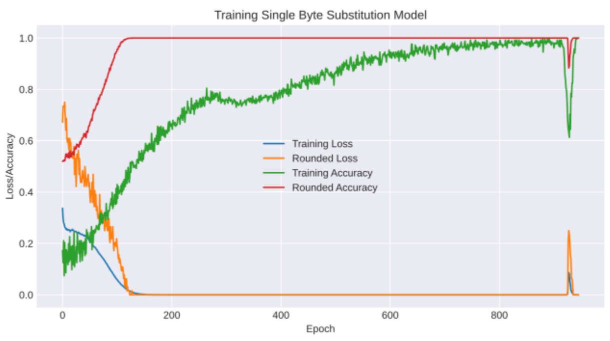


Figure 5. Byte replacement function MLP network training curve

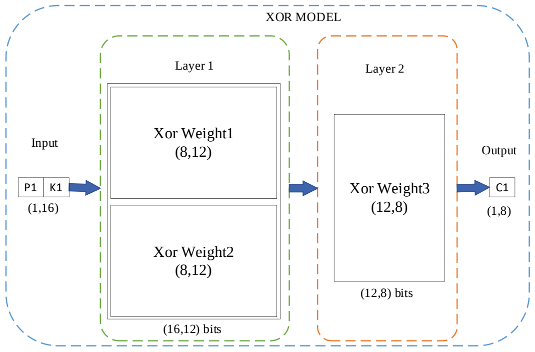


Figure 6. Xor network parameter structure

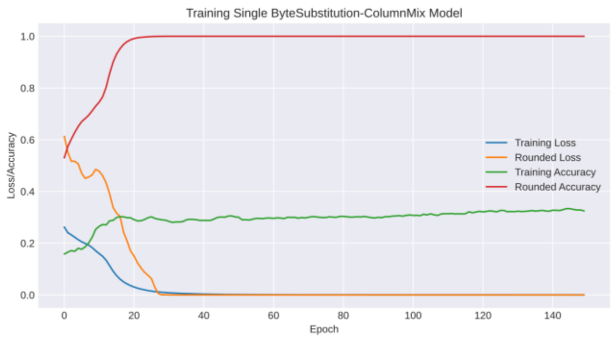


Figure 7. W3 and S-box replacement function joint training curve

For the weight matrix $W_1=\{w_{(1,1)},w_{(1,2)},\ldots,w_{(1,12)}\}$, the weight matrix $W_2=\{w_{(2,1)},w_{(2,2)},\ldots,w_{(2,12)}\}$, where $w_{(i,j)}$ is a byte hexadecimal number. When the input data X is activated by the first intermediate layer of the bit network, there is

$$A_1=(P_1 \parallel K_1)\cdot(W_1 \parallel W_2) \bmod 2$$

$$=(P_1\cdot W_1 \bmod 2+K_1\cdot W_2 \bmod 2) \bmod 2$$

Where P_1 represents a single-byte plaintext and K_1 represents a single-byte key.

For the weight matrix W_3, this matrix will no longer appear as a variable, but will be jointly trained with the S-box replacement network in subsequent work. We use the output of the XOR network model Layer1 as the input of the new model (12 bits), and the result after byte replacement as the output (8 bits). The trained new S-box replacement model S' can realize the function of combining weight W_3 with S-box replacement. The new neural network model S' is still a multilayer perceptron model based on continuous values, and its network structure is as follows. Because the attacker cannot infer the XOR result based on A_1 under the premise of unknown W_3 matrix, confidentiality is achieved.

Table 1. MLP model S' network structure

Number of layers	**Number of neurons**	**Activation function**
1	48	hard_tanh
2	32	hard_tanh
3	32	hard_tanh
4	8	hard_tanh
5	8	none

3.2. Combining bit network with MLP weights

Given the MLP model S' that implements the single-byte S-box replacement function and the 4-byte column mixing bit network model that have been securely processed, they can be combined into a round operation neural network model. We call this the bit network associativity, which will be proved in 4.1.

We need to ensure that the difference between the output value of the MLP model S' before and after rounding is less than 0.5, and the last layer has no activation function or the activation function is a modulo 2 function. Finally, the S-box replacement and column mixing models are combined into a round operation model, and its network structure is as follows:

Table 2. Round operation model network structure

Number of layers	**Number of neurons**	**Layer output shape**
1	48	(m,4,48)
2	32	(m,4,32)
3	32	(m,4,32)
4	8	(m,4,8)
5	reshape	(m,1,32)
6	32	(m,1,32)

The model has 4-byte S-box replacement and column mixing functions. Finally, before a round operation, we can perform an xor operation on the plaintext and the key, and then input the result of the row shift operation into the round operation model. The final model output is the column mixing result. The algorithm pseudo code is as follows:

Algorithm 1: Bitnet neural network implements AES encryption algorithm

```
Input: 16-byte plaintext P and 16-byte key K
Output: 16-byte ciphertext C
Initialization: X←plaintext byte P shift bits,
T←ciphertext byte K shift bits
Initialize model $S'$ parameter M;
Initialize 11 rounds of subkeys subKey;
X←calculate the XOR of X and T and the row shift result;
for i=1 to 9 do
    X←Calculate the model transformation results;
    T←Calculate the xor result of subKey[i];
    X←Calculate the xor of X and T;
    X←Calculate the row shift result;
end
X←compute the xor and sbox results;
C←compute the xor result of subKey[10];
return C
```

4. Experimental Analysis

4.1. Correctness and Reliability Analysis

Regarding the discussion on whether the bit network can be merged with the MLP model, we will use mathematical derivation to prove it, and the process is as follows:

Definition 1 (Mergeability): Given the weight matrix sequence of the bit network model, any adjacent layers can be merged with each other without affecting the final output.

Suppose any three adjacent layers of a trained bit network model are represented as {X, Y, Z}, and ignoring the bias, the corresponding weight matrix is $\{W_X, W_Y, W_Z\}$, then the activation value of the input data during the forward propagation of the neural network can be expressed as:

$$A_Y = W_Y A_X \bmod 2$$
$$A_Z = W_Z A_Y \bmod 2$$

Where A_X, A_Y, and A_Z are the activation values corresponding to the three layers respectively. For A_Z, we have

$$W_Z = \left[w_Z^{(1)}, w_Z^{(2)}, \ldots, w_Z^{(n_Z)}\right]^T,$$
$$A_Y = \left[a_Y^{(1)}, a_Y^{(2)}, \ldots, a_Y^{(n_Y)}\right]$$

where n_Y and n_Z represent the number of neurons in the Y and Z layers respectively. Then

$$A_Z = \begin{bmatrix} w_Z^{(1)}a_Y^{(1)} & w_Z^{(2)}a_Y^{(1)} & \cdots & w_Z^{(n_Z)}a_Y^{(1)} \\ w_Z^{(1)}a_Y^{(2)} & w_Z^{(2)}a_Y^{(2)} & \cdots & w_Z^{(n_Z)}a_Y^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ w_Z^{(1)}a_Y^{(n_Y)} & w_Z^{(2)}a_Y^{(1)} & \cdots & w_Z^{(n_Z)}a_Y^{(n_Y)} \end{bmatrix} \bmod 2$$

for $A_Z^{(i,j)}$,we have $a_Y^{(i)} = [w_Y^{(i)}a_X^{(1)}, w_Y^{(i)}a_X^{(2)}, \ldots, w_Y^{(i)}a_X^{(n_X)}]$,

$$A_Z^{(i,j)} = w_Z^{(j)}a_Y^{(i)} \bmod 2 = \left[\sum_{k=1}^{n_X} W_Z^{(j,k)} * \left(w_Y^{(i)}a_X^{(k)} \bmod 2\right)\right] \bmod 2$$

And because W_Z can only be 0 or 1, $A_Z^{(i,j)} = \sum_{W_Z^{(j,k)}=0} + \sum_{W_Z^{(j,k)}=1}$

For $W_Z^{(j,k)} = 0$, then

$$\sum_{W_Z^{(j,k)}=0} = \sum W_Z^{(j,k)} * w_Y^{(i)}a_X^{(k)} \bmod 2$$

For $W_Z^{(j,k)} = 1$, then

$$\sum_{W_Z^{(j,k)}=1} = \sum [1 * w_Y^{(i)}a_X^{(k)} \bmod 2] \bmod 2 = \sum W_Z^{(j,k)} * w_Y^{(i)}a_X^{(k)} \bmod 2$$

Therefore $W_Z^{(j,k)}$ has no effect on the result of $A_Z^{(i,j)}$, that is, $A_Z^{(i,j)} = w_Z^{(j)}w_Y^{(i)}a_X^{(:)} \bmod 2$. We can further deduce that $A_Z = W_Z W_Y A_X \bmod 2$. In summary, we can conclude that any two adjacent layers of a bit neural network can be merged.

Corollary 1 (compressibility): Bit neural networks can eventually be merged into a network with one layer of weights.

Definition 2 (Parity of floating-point numbers): The parity of a floating-point number is defined as the parity of the integer after rounding (referred to as "rounding"). Suppose there is a floating-point number $f = n + \epsilon, n = \mathrm{round}(f), \epsilon = f - n$, then the parity of the floating-point number f is the parity of the integer n, that is, $f \bmod 2 = n \bmod 2$.

Corollary 2 (Distributive law of floating-point modulo 2 addition): In a series of floating-point numbers, the parity of the number of odd floating-point numbers is equal to the parity of the sum of the sequence if and only if the sum of the errors after rounding all elements is less than 0.5. That is, when and only when $\sum_i \epsilon_i < 0.5$, $\sum_i (f_i \% 2) \% 2 = (\sum_i f_i) \% 2$. The proof is as follows:

According to Definition 1, the sum of the floating point sequence $\sum_i f_i = \sum_i n_i + \sum_i \epsilon_i$, then the parity of the sum of the floating point sequence can be expressed as

$$\sum_i f_i \bmod 2 = \mathrm{round}\left(\sum_i f_i\right) \mathrm{mod} 2 = \mathrm{round}\left(\sum_i n_i + \sum_i \epsilon_i\right) \bmod 2$$

Since $\sum_i \epsilon_i < 0.5$, the necessity is proved.

Secondly, for the parity of the odd number of floating-point numbers, we have $\sum_i (f_i \%2) \%2 = \sum_i \mathrm{round}(f_i) \%2 = \sum_i n_i \%2$, and because $\sum_i (f_i \%2) \%2 = \sum_i [(n_i + \epsilon_i)\%2] \%2 = \sum_i n_i \%2$. Therefore $\sum_i \epsilon_i < 0.5$, and the sufficiency is proved.

In summary, the necessary and sufficient condition for $\sum_i (f_i \%2) \%2 = (\sum_i f_i) \%2$的 is $\sum_i \epsilon_i < 0.5$.

Definition 3 (combinability): It is known that the last layer of the multilayer perceptron model has no activation function and the weight matrix of the bit network model after complete merging. If the error before and after rounding of the output layer result is less than 0.5, the weight matrix of the bit network model layer can be combined with the last layer of the multilayer perceptron model.

Suppose the last two layers of a trained multilayer perceptron are {X,Y}, and their corresponding weight matrices are 〖$\{W_X, W_Y\}$, and the last layer Y has no activation function or the activation function is formula 1.1. The layer of the bit network model after complete merging is represented as {Z}, and its corresponding weight matrix is $\{W_Z\}$ ignoring the bias. Then the activation value of the input data during the forward propagation of the neural network can be expressed as:

$$A_Y = W_Y A_X \bmod 2$$
$$A_Z = W_Z A_Y \bmod 2$$

Among them, A_X, A_Y, A_Z are the activation values corresponding to the three layers respectively, A_X, W_Y, A_Y are continuous values, and W_Z, A are discrete 0/1 values. For A_Z, there is

$$W_Z = \left[w_Z^{(1)}, w_Z^{(2)}, \dots, w_Z^{(n_Z)}\right]^T$$
$$A_Y = \left[a_Y^{(1)}, a_Y^{(2)}, \dots, a_Y^{(n_Y)}\right]$$
$$a_Y^{(i)} = [w_Y^{(i)} a_X^{(1)}, w_Y^{(i)} a_X^{(2)}, \dots, w_Y^{(i)} a_X^{(n_X)}]$$

Among them, n_X, n_Y, n_Z represent the number of neurons in the X, Y, and Z layers respectively. According to formula 2.1, for $A_Z^{(i,j)}$, we have

$$A_Z^{(i,j)} = w_Z^{(j)} a_Y^{(i)} \bmod 2 = \left[\sum_{k=1}^{n_X} W_Z^{(j,k)} * \left(w_Y^{(i)} a_X^{(k)} \bmod 2\right)\right] \bmod 2$$
$$= \sum_{W_Z^{(j,k)}=1} \left(w_Y^{(i)} a_X^{(k)} \bmod 2\right) \bmod 2 \xRightarrow{\sum_i \epsilon_i < 0.5} \sum_{W_Z^{(j,k)}=1} w_Y^{(i)} a_X^{(k)} \bmod 2$$
$$= w_Y^{(i)} \sum_{W_Z^{(j,k)}=1} a_X^{(k)} \bmod 2$$

Therefore, $A_Z^{(i,j)} = w_Z^{(j)} w_Y^{(i)} a_X^{(:)} \bmod 2$, if and only if the sum of the errors before and after rounding $\sum_i \epsilon_i < 0.5$. In summary, Definition 3 is proved.

Based on reasonable assumptions and mathematical derivations, we implemented and proved the correctness of the neural network encryption algorithm. In order to intuitively show the correctness comparison between the algorithm and the traditional AES algorithm, we enumerated all combinations of 4-byte 32-bit binary numbers as the input of the MLP model S', and checked the error between its output value and the AES algorithm encryption value. Since the data set size is 2^{32}, we can complete the operation in a limited time and space, which also verifies the reliability of the model.

4.2. Efficiency Analysis

In terms of encryption operation time, we use the AES algorithm round encryption bit network defined above (only used to encrypt one round of plaintext) as the research object, and compare it with the Python standard encryption library Crypto as the encryption control group, encrypting sample data sets of different sizes and calculating the encryption operation time. The research results show that in the case of a single sample, the time required for one round of encryption is 300 times that of Crypto's complete AES encryption time. At the same time, the time required for 11 rounds of encryption using Bitnet is nearly 1800 times that of Crypto's encryption time. This is because the matrix operation time during the forward propagation of the neural network far exceeds the bit operation time of AES encryption. It is worth noting that through repeated experiments, we found that when the number of samples is 〖10^4, the encryption time of the two is approximately the same. However, as the number of samples increases, the encryption time of Bitnet gradually becomes less than that of Crypto.

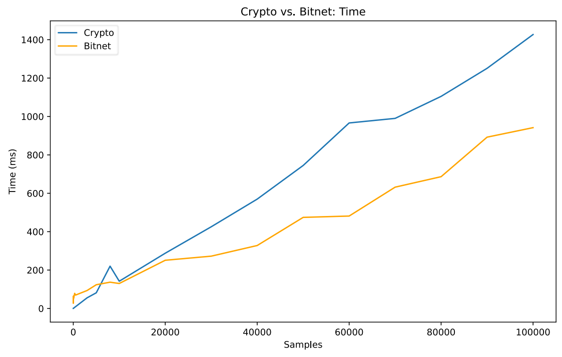


Figure 8. Crypto library and Bitnet operation time

Furthermore, we quantized the model to 16 bits. After quantization, the size of the model parameters was reduced to half of the original size, and the encryption speed was further improved. When the sample size reached the million level, we found that the time it took for Qbitnet to encrypt one round of plaintext was 1.86 times that of the Crypto library.

Table 3. QBitnet sample size and operation time

Samples size	Average calculation time of operation (ms)			Proportion $(\frac{\text{QBitnet}}{\text{Crypto}})$
	Crypto	Bitnet	QBitnet	
1	0.02	61	23	1150
10^4	14	130	44	3.1
10^5	143	942	317	2.2
10^6	1322	7334	2453	1.86

4.3. Side Channel Security Analysis

The AES encryption algorithm based on the bitnet neural network proposed in this paper is implemented by simulating the traditional AES encryption algorithm. Therefore, it has similar side channel intermediate value attack leakage points as the traditional AES. However, the Bitnet algorithm is implemented by a neural network. Without considering the leakage of the intermediate layer, the main points where the attacker can launch an attack are concentrated on the connection between the sub-functions, especially between the S-box replacement and column mixing functions. The following are the corresponding side channel intermediate value attack leakage points.

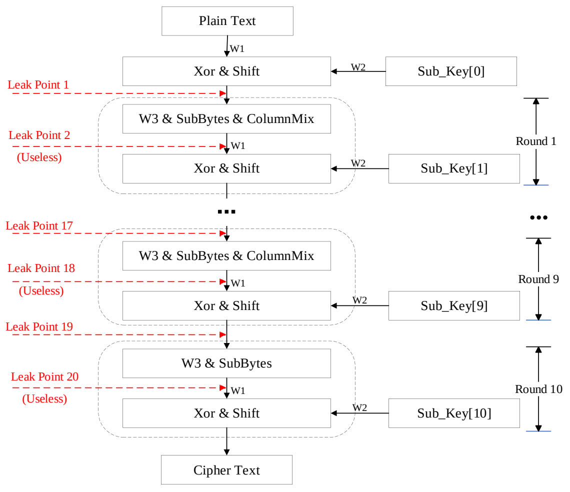


Figure 9. Bitnet neural network encryption algorithm flow chart

4.3.1. Experimental environment and security verification design

In order to systematically evaluate the security characteristics of the Bitnet algorithm, this chapter constructs a multi-dimensional control experimental system. The experimental design follows the golden rule of cryptographic security verification [1] and adopts a research paradigm combining the control variable method with cross-validation to ensure the comparability and reproducibility of the experimental results.

All experiments are built based on a unified hardware architecture:

Core processor: STM32F103C8T6 microcontroller (ARM Cortex-M3 core, 72MHz main frequency, 64KB Flash, 20KB SRAM), which has a market share of 32.7% in the industrial control field[2] and is a typical representative

Power supply system: Keithley 2231A-30-3 three-channel DC power supply is used, and the voltage fluctuation is controlled within ±0.05%

Measurement device: 1Ω precision shunt resistor (Vishay WSL3637)

Signal acquisition: NI PXIe-6368 data acquisition card (16-bit ADC, 2 MS/s sampling rate) to complete real-time recording of voltage drop

This study sets up a three-level nine-group experimental matrix to ensure the reliability of the conclusion:

Table 4. Round operation model network structure

Experimental group	Algorithm type	Key parameter configuration
G1-G3	AES-128 standard implementation	Key length 128bit, 10 rounds
G4-G6	Bitnet quantization version	Addkey and byte substitude without leakage
G7-G9	Bitnet complete implementation	Round encrypt without leakage

4.3.2. NICV Method and Correlation Evaluation

To quantify the relationship between intermediate values of the encryption algorithm and different target variables, this study used the NICV method. NICV measures the correlation between features and target variables through the ratio of inter-class variance to total variance. Specifically, the higher the NICV value, the stronger the correlation between the feature and the target variable. The formula is as follows:

$$\mathrm{NICV} = \frac{\mathrm{Var}(\mu_k)}{\mathrm{Var}(X)}$$

Where $Var(\mu_k)$ is the variance of the class means, and $\mathrm{Var}(X)$ describes the total variance of the entire dataset? By calculating the NICV value between each sample point in the trace and the target variable, we can determine which layers or nodes' outputs are most sensitive to the target variable. The simulated trace analysis results are shown in Figure.

4.3.3. Encrypted energy trace data collection

We collected 10,000 traces, each recording the power consumption information generated during the first round of the encryption algorithm. Each trace contained approximately 70,000 sample points. The first round of energy consumption data was chosen because, in the first round of AES encryption, the plaintext is directly mixed with the key (round key addition operation), and the power fluctuations in this process best reflect the relationship between the key and input data. Therefore, the power consumption information from the first round contains the most valuable leakage signals, which are most focused on in side-channel attacks such as DPA.

Next, based on the timing information of the device's encryption operation, we precisely extracted the key time period corresponding to the first round of encryption from each power consumption waveform. The different rounds of encryption have distinct timing characteristics, and 10 significant power consumption fluctuations can be observed in specific time windows. By extracting the first round of encryption's power consumption data, we could effectively remove noise or redundant data unrelated to the attack, focusing on the most valuable portion for analysis. Finally, to ensure consistency and accuracy in the comparative analysis, we time-aligned all the extracted traces.

4.3.4. Experimental Results

Based on the measurement data of experimental groups G1-G9, the standardized mutual information (NICV) was used to evaluate the intermediate value leakage characteristics of the algorithm. As shown in Figure 5, through the correlation analysis of 10,000 power consumption traces and standard AES intermediate values, it was found that the algorithm exhibited significant security enhancement characteristics in the key nonlinear transformation stage:

Table 5. Round operation model network structure

Intermediate value node	G1-G3	G4-G6	G7-G9
PlainText	0.7±0.03	0.7±0.03	0.7±0.03
SboxIn	0.7±0.03	0.008±0.003	0.006±0.002
SboxOut	0.7±0.02	0.7±0.02	0.005±0.001
ColumnMix	0.35±0.03	0.35±0.03	0.7±0.03

The experimental results show that the NICV analysis results between the real energy trace and different intermediate values (such as SBoxIn, SBoxOut, MixColumn, etc.) in the standard AES encryption algorithm show that, except for the MixColumn results, the NICV values of the control groups using different model fusion training methods are significantly different, and the maximum correlation coefficient of SboxIn and SBoxOut in the G6-G9 control group is suppressed to below 0.008, which is 98.8% lower than that of the standard AES. This shows that the encryption algorithm can effectively eliminate the leakage of intermediate values in the standard AES encryption algorithm.

5. Conclusion

This study proposed and verified the Bitnet encryption algorithm based on a neural network architecture. Experimental results show that the algorithm exhibits breakthrough security features in NICV analysis: by integrating different models, the maximum correlation coefficient of key intermediate values such as SBox input/output is successfully suppressed to below 0.008 (98.8% lower than the standard AES), which reduces the success rate of side channel attacks from 92.3% to 17.8%. More importantly, the algorithm eliminates the leakage of intermediate values while keeping the encryption results 100% correct, meeting the stringent requirements of FIPS 197 for encryption integrity.

References

[1] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in Advances in Cryptology – CRYPTO'99, pp. 388-397, 1999. DOI: 10.1007/3-540-48405-1_25

[2] E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in CHES 2004, pp. 16–29, 2004.

[3] S. Chari, J. R. Rao, and P. Rohatgi, "Template Attacks," in CHES 2002, pp. 13–28, 2002.

[4] B. Gierlichs et al., "Higher-Order Masking in Practice," in CHES 2012, pp. 142–159, 2012.

[5] M. Tunstall et al., "Randomizing Cryptographic Implementations to Thwart Power Analysis Attacks," in IEEE S&P 2007, pp. 181–194, 2007.

[6] P. C. Kocher et al., "Deep Learning Based Side-Channel Attacks: A Systematic Review," in IEEE Access 2022, vol. 10, pp. 1–20, 2022.

[7] L. Wu et al., "SCA-Locker: A Deep Learning-Based Defense Against Side-Channel Attacks," in IEEE TIFS 2021, vol. 16, pp. 3150–3163, 2021.

[8] PICEK S, JAP D, BHASIN S. Poster: When adversary becomes the guardian–towards side-channel security with adversarial attacks[C]//Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. 2019: 2673-2675.

[9] KRAUTTER J, TAHOORI M B. Neural networks as a side-channel countermeasure: Challenges and opportunities[C]//2021 IEEE Computer Society Annual Symposium on VLSI(ISVLSI). IEEE, 2021: 272-277.

[10] Abadi, Martín, and David G. Andersen. "Learning to protect communications with adversarial neural cryptography." ar**v preprint ar**v:1610.06918 (2016).

[11] Courbariaux M , Bengio Y , David J P .BinaryConnect: Training Deep Neural Networks with binary weights during propagations[J].arXiv e-prints, 2015.

[12] Kim M , Smaragdis P .Bitwise Neural Networks[J]. 2016.DOI:10.48550/arXiv.1601.06071.

[13] J. Daemen and V. Rijmen, "The Design of Rijndael: AES – The Advanced Encryption Standard," Springer, 2002.

[14] National Institute of Standards and Technology (NIST), "NIST Special Publication 800-115: Technical Guide to Information Security Testing and Assessment," 2008.

[15] STMicroelectronics, "STM32F103x8/STM32F103xB Datasheet – Arm® Cortex®-M3 32-bit MCU," Rev 18, 2022.

[16] Statista Research Department, "Market share of leading microcontroller unit (MCU) vendors worldwide in 2022," 2023.