Power Analysis Attack Based on GA-Based Ensemble Learning

Xiaoyi Duan, Ye Huang, Yuting Wang, Yu Gu, Jianmin Tong, Zunyang Wang, and Ronglei Hu(✉)

Beijing Electronic Science and Technology Institute, Fengtai District, Beijing, China
huronglei@sina.com

Abstract. Perin et al. proposed the random ensemble leaning method. This method generates multiple neural network models, randomly sets the parameters of the models, then the models are integrated to perform power analysis attacks. Compared with single model, this method is more efficient, which performs very well in enhancing the performance of side-channel attacks. However, Perin's solution does not solve the problem of combinatorial optimization during ensemble and relatively requires more neural networks to be integrated. This paper proposes a GA-based Ensemble Learning method, which generates multiple neural network models, then obtains the optimal parameters of the models through the genetic algorithm to solve the optimal combination problem of the integrated neural network, and finally use the network with optimal parameters for power analysis attacks. Compared with Perin's method, the proposed method needs less neural network ensemble to achieve better results. Compared with Perin's random ensemble learning method on three data sets (ASCAD_f, ASCAD_r, CHES CTF) respectively, in order to achieve the same attack effect, GA-based Ensemble method reduces 10 models in ensemble scale than Perin's Random Ensemble learning method. The GA-based ensemble learning method can further improve the attack performance of the ensemble and reduce the scale of the ensemble, thereby saving the training cost.

Keywords: Index Terms—Power Analysis Attack · Machine Learning · Hyperparameter Search · Ensemble Learning · Genetic Algorithm

1 Introduction

The growing embedded computing market, especially the IoT market, requires large amounts of confidential data to be processed on electronic devices. Cryptographic algorithms are often integrated into these devices to encrypt confidential data and, if not properly protected, are vulnerable to side-channel attacks (SCA). Depending on the level of access and control of the target device, side-channel attacks can be classified as analytical attacks (e.g. Template Attacks [3], Linear Regression [4], Machine Learning [5]) or non-analytical attacks (e.g. DPA [1]), CPA [6], MIA [7]).

This paper is supported by "the Fundamental Research Funds for the Central Universities" (Grant Number: 328202207, 328202247).

S. Goel and P. R. Nunes de Souza (Eds.): ICDF2C 2023, LNICST 570, pp. 308–322, 2024.
https://doi.org/10.1007/978-3-031-56580-9_19

In the past few years, deep learning has been widely used in side channel attacks. The selection of super parameters is a key factor affecting the generalization effect of depth neural networks, which can be inferred by observing the influence of under fitting or over fitting. One way to solve this problem is to obtain the best model that is theoretically most suitable for attacking the specified data set through powerful super parameter adjustment. However, when applied to energy analysis attacks, the cost may be very expensive. A real attack scenario may be present with thousands of different super parameter combinations, which need to be tested within a reasonable time. This is particularly difficult when attacking a protected AES implementation because hundreds of thousands of recorded tracks are required in the training set.

Therefore, this paper considers integrating multiple models with approximate average class probability output, rather than simply selecting the best model from a large number of super parameter combinations. In this paper, firstly, the genetic algorithm in the optimization selection method is adopted to select small model parameters with good performance from multiple parameter combinations, and then a group of optimal models are obtained from these small models using the integration method. The results of this paper emphasize that the integration process ensures significantly higher efficiency and better attack effect compared with the single best model obtained by using super parameter search.

1.1 Related Work

The side-channel attack was first proposed by the American cryptographer Kocher in 1999 [1], which provides another attack idea besides the mathematical analysis of the cryptographic algorithm itself. It uses the physical information generated in the process of hardware data encryption, such as power consumption, electromagnetic radiation, time, etc. to crack the encryption algorithm [2].

In 2011, Hospodar et al. [8] first applied machine learning to side-channel attacks. Based on the Hamming weight leakage model, Hospodar et al. successfully attacked some software implementations of the Advanced Encryption Standard (AES) through Support Vector Machine (SVM). Hospodar et al. [9] classified the median value in the template attack with the least squares support vector machine. In 2012, Heusar et al. [10] attacked multi-bit values with multi-class SVM (Hamming weight model). In 2013, Martinasek et al. [11] proposed a neural network-based AES side-channel attack method and classified AES keys. At the CHES conference in 2015, Whitnall C [12] et al. used an unsupervised machine learning method to propose a classification recognizer which can tolerate a certain difference between analysis traces and attack traces. At the CHES conference in 2017, EleonoraCagli et al. [13] proposed an attack strategy based on convolutional neural networks (CNNs). The experimental results show that this strategy greatly simplifies the attack method because it does not require previous trace realignment and any precise POI selection. In 2018, Prouff et al. [14] proposed a method to verify the optimal hyperparameters for a specific set of traces which provides valuable information for side-channel attacks, but still remains challenging to use the same conclusions for datasets acquired and measured by different devices. In 2019, Jaehun Kim [15] et al. applied various techniques to improve the generalization ability of deep neural networks to side-channel information, and added noise to the input traces

as a regularization technique. In 2019, Benjamin Hettwer et al. [16] proposed to use cryptographic information (plaintext and ciphertext) as an additional input to the first fully connected layer in a CNN to improve key enumeration. In 2019, the work presented by Bejamin et al. [17] considered a DPA-like model for deep learning attacks, where in the AES implementation, the model was trained for each key byte candidate. The training and validation metrics were then considered as references to distinguish between correct and incorrect candidate key bytes. In 2020, Wouters et al. [18] proposed a method to select hyperparameters related to the size of layers (the number of learnable parameters, i.e. weights and biases) in a CNN, including the number of filters, kernel size, stride, and the number of neurons in the fully connected layer. It also shows how to achieve similar attack performance using a smaller neural network structure.

In the past few years, deep learning has been widely used in side-channel attacks. The choice of hyperparameters is a key factor affecting the generalization effect of deep neural networks, which can be inferred by observing the effects of underfitting or overfitting. One way to address this problem is through powerful hyperparameter tuning to obtain a theoretically best model, aiming at attacking a given dataset. However, when applied to power analysis attacks, it can be very costly. A realistic attack scenario may preset thousands of different hyperparameter combinations, which need to be tested in a reasonable amount of time. This is especially difficult when attacking protected AES implementations because hundreds of thousands of recorded traces need to be used in the training set.

In 2020, GuilhermePerin et.al [19] proposed an integrated learning-based power analysis attack method based on traditional machine learning parameter search technologies. This method will generate a certain number of models in advance and randomly set the parameters of these models within a certain parameter range, then train these models for ensembles, and conduct power analysis attacks through the ensemble models. The results show that compared with a single model, this method performs very well in enhancing the performance of side-channel attacks and making the output class probabilities more stable. However, Perin's solution does not consider the combinatorial optimization problem during ensemble. The neural network parameters of each ensemble are randomly generated, and the parameters of the ensemble model are not optimally selected. Therefore, more neural networks need to be integrated to achieve better results.

While random ensemble improves generalization through ensemble, it performs very well in enhancing the performance of side-channel attacks and making the output class probabilities more stable than using a single model. However, random ensemble did not optimize the parameters of the ensemble multiple neural network models, resulting in insufficient generalization ability, and the need of more models to achieve a better ensemble effect. To this end, this paper proposes a GA (Genetic Algorithm, GA)-based ensemble scheme, which generates multiple neural network models, then obtains the optimal parameters of these neural network models through genetic algorithm, and finally uses the neural network of these optimal parameters. Compared with Perin's method, even a smaller scale of neural network ensemble can achieve better results.

1.2 Our Contribution

(1) In this paper, the parameters of the integrated multiple neural network models are optimally arranged, grouped, sequenced and screened to obtain the optimal combination, rather than simply randomly setting the parameters of these integrated multiple neural network models. The parameters of the integrated multiple neural network models are optimized by the genetic algorithm, making the combinatorial optimization problem solved and the optimal integrated model obtained. The results of this paper emphasize that compared with Perin's Random Ensemble method, the GA-based Ensemble learning method can further improve the attack performance of the ensemble and reduce the scale of the ensemble, thus saving the training cost.
(2) In this paper, the proposed method is analysed using three widely used datasets, ASCAD_f, ASCAD_r, and CHES CTF, and the experimental results are validated by the guessing entropy. The results show that the same attack effect is achieved, GA-based Ensemble method reduces 10 models in ensemble scale than Perin's Random Ensemble method. It can be seen that, compared with Perin's Random Ensemble, the GA-based Ensemble method can further improve the attack performance of the ensemble and reduce the scale of the ensemble, thereby saving the training cost.

1.3 Structure of This Article

The structure of this paper is as follows. Section 2 discusses the datasets studied in this paper and the rationale for the ensemble. Section 3 discusses how to choose appropriate metrics to help find the optimal model, and how to use genetic algorithms to choose and optimize a specific combination of hyperparameters. Section 4 compares experiments with Perin' Random Ensemble method on three datasets. Finally, Sect. 5 discusses possible future research directions and concludes the paper.

2 Background

2.1 Ensembles of Machine Learning

In specific application scenarios, machine learning methods may sometimes fail to achieve good enough performance. However, it is possible to try to improve performance by combining multiple learners (machine learning models), which is called ensemble learning. More precisely, the ensemble is combining the decisions of complementary learners (or classifiers) to improve the generalization performance of a single learner (and thus reduce generalization error). The ensemble combines multiple hypotheses (classifiers) to form better hypotheses. In general, if the errors of the individual classifiers are (to some extent) uncorrelated, and the error rate of the classifiers is lower than random guessing, then the ensemble can be more successful than a single classifier. The cost of building an ensemble may not be significantly higher than building a single learner, because when using a single learner, model selection and hyperparameter tuning often produce multiple versions of the model, which is similar to the cost of building multiple single learners in an ensemble learning. At the same time, since the combination strategy is generally simpler, combining multiple learners often only requires little computational cost. It is not difficult to see from the figures that the overall model complexity of the small models ensemble method is smaller and easier to train.

Common ensemble strategies include:

(1) Boosting: It is considered as a sequential ensemble learning method. It constructs and combines weak classifiers (classifiers that are only slightly correlated with the true class) to achieve the same performance as strong classifiers (classifiers that are strongly correlated with the true class). Weak classifiers only need to consider a small portion of the training set, thus making the training process faster. The boosting process works in iterations, where each iteration represents a weak classifier training. The training data comes from the original distribution. The error is evaluated in each iteration. Before a new iteration, the distribution of the new training data is adjusted according to certain criteria. Finally, based on the selection of training data, a classifier with the best metric will be obtained. The Boosting strategy was used in the analysis SCA of S. Picek et al. [20].
(2) Bootstrap aggregating (Bagging): This method considers the mean or linear combination (or weighted sum) of all individual classifier predictions. Bagging uses bootstrap sampling to generate different base classifiers, creating multiple subsamples of the dataset by substitution (which means that the two sample values are independent, i.e. their covariance is equal to 0), training a classifier for each subsample, and calculating the average prediction for each classifier. Bagging is not uncommon in SCA[3], for example, Random Forest (Randon Forest) uses Bagging.
(3) Stacking: In this special case, one first uses the original training dataset to train the first-level classifiers. Then the output of the first-level classifiers is used as input features, and the corresponding original tags are utilized as new tags to form a new dataset to train a secondary classifier. The individual classifiers in the training process can be generated by different algorithms.

In this paper, the Bagging strategy is used to integrate the models. In 2007, Friedman et al. [21] proposed that Bagging can reduce the variance of high-order components, which makes bagging more suitable for highly nonlinear learners. Highly nonlinear learners are generally unstable, that is, their generalization ability changes with the distribution of collected data samples. Bagging can effectively improve the ensemble performance of unstable base learners.

Similar to other ensemble strategies, the performance of Bagging converges with the size of the ensemble. Given a training set, Bagging uses self-sampling to generate a random sample set for training the base learners. Given a test sample, the predicted output of the base learners can be represented as a probability distribution of random variables.

Suppose Y as a binary classification problem, ie. $Y \in \{ -1, 1\}$. Bagging often uses a voting strategy to combine the prediction results of the base learner. It may be assumed that the average voting is used, then the final result of using T base classifiers is:

$$\overline{Y_T} = \frac{1}{T} \sum_{i=1}^{T} Y_i \tag{1}$$

where Y_i denotes the result of the i_{th} base classifier. Let $E[Y]$ represent the mathematical expectation of $\overline{Y_T}$, which is known by the law of large numbers:

$$\lim_{T \to \infty} P(|\overline{Y_T} - E[Y]| < \varepsilon) = 1 \tag{2}$$

Unless $E[Y] = 0$, otherwise there are:

$$\lim_{T\to\infty} P(sgn(\overline{Y_T}) = sgn(E[Y])) = 1 \tag{3}$$

Therefore, unless the performance of all base learners is poor enough that the results are equivalent to random guessing (which is almost impossible), the error rate of bagging will gradually converge to a stable level as the number of ensemble learners increases.

2.2 Datasets

The experiments mainly focus on three datasets, all of which use the AES encryption algorithm, and the data comes from the power measurement process in the power analysis attack. Table 1 provides the details of the different datasets.

Table 1. The datasets

Dataset	Training Traces	Test Traces	Features	Countermeasure
ASCAD_f	50000	10000	2000	Masking
ASCAD_r	200000	100000	1400	Masking
CHES CTF 2018	45000	5000	2000	Masking

The first target dataset is the ASCAD dataset [22], which was captured on an 8-bit AVR microcontroller running a masked AES-128 implementation. There are two versions of the ASCAD dataset: The first version has a fixed key, with 50,000 traces for analysis/training, and 10,000 traces for testing. This dataset is denoted as ASCAD_f. The second version has random keys and the dataset consists of 200,000 traces for analysis and 100,000 for testing. This dataset is denoted as ASCAD_r. For both versions, the third byte of the key is attacked because it is the first byte containing the mask.

The second dataset, CHES CTF 2018, is the CHES Capture the flag (CTF) AES-128 trace set published in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). These traces consist of shielded AES-128 encryption running on a 32-bit STM microcontroller. The training set in the experiments of this paper consists of 45,000 traces with a fixed key. The test set includes 5000 traces. The key used in the test set is different from the key used in the training set. Each trace consists of 2000 feature points. For this dataset, the 1st byte is chosen to be attacked.

3 Model Optimization and Ensemble

3.1 The Way to Find the Optimal Model

In the process of using deep learning for SCA, one of the most important goals is to adjust the parameters of the deep neural network to obtain the corresponding model, so that it can achieve the best attack performance. The tuning, i.e. the choice of hyperparameters, is usually a two-layer optimization problem. The first goal is to learn the neural

network parameters (such as weights and biases); the second goal is the performance hyperparameters about the chosen network structure. In deep learning-based SCA, a leakage model may also be selected as an optimization target. In theory, it is possible to traverse all deep neural network configurations over a significant amount of time and monitor output metrics to continually tune hyperparameters. Such a hyperparameter search process generates many analytical models, which are then cross-validated to obtain the model with the smallest generalization error.

Frank Hutter et al. [23] proposed general criteria for evaluating models: Suppose h is a neural network model with a set of hyperparameters to be selected $\wedge = (\lambda_1, \lambda_2, ..., \lambda_n)$, after the training set T_{train} training, the test set After the T_{test} testing, several training models $H = (h_{\lambda 1}, h_{\lambda 2}, ..., h_{\lambda n})$ are obtained, and the optimal model is selected according to formula (4):

$$h_{best} = \arg\min_{\lambda \in \wedge} L_{val}(h_\lambda, T_{train}, T_{test}) \tag{4}$$

where $L_{val}(h_\lambda, T_{train}, T_{test})$ represents the test loss of the model h_λ.

The hyperparameters of the model h are all chosen within a predefined range of hyperparameter values, and all the trained models constitute the set H. Considering the training complexity of each individual model, the size of the set H must be limited in order to get results in a reasonable time. For example, training a deep neural network on a dataset containing millions of data traces to crack a single key byte can take several minutes (using parallel GPUs). Whereas, if considering the AES implementation of the 16-bit key byte analysis attack, the hyperparameter optimization process needs to evaluate hundreds of different sets of hyperparameters, which would take weeks or even months.

Therefore, from the perspective of attack efficiency, this paper proposes to integrate multiple models with less training complexity instead of obtaining a single optimal model through large-scale hyperparameter search. During the tuning phase a large number of machine learning models can be obtained, so the ensemble does not add additional computational complexity. The time complexity increases only when all models are combined in the attack phase, and grows linearly with the number of models used.

3.2 Ensemble Models

If the models have the same configuration, the neural network may learn from the training set and provide similar classification results. Therefore, the hyperparameters of each model should be different, thus learning different features from the same training set. The main goal of integrating the models is to improve the performance of SCA.

When deep learning-based SCA succeeds, the main reason is that for the correct candidate key, the summed probability obtained by formula (5) is greater than any other candidate key. Therefore, a successful ensemble should increase the summed probability of the correct candidate key while reducing the impact of incorrect candidate keys.

Formula (5) shows that the best model can be selected from hyperparameter optimization through a loss function. Itis simple and effective in traditional machine learning, since the lowest test loss represents the best generalization. However, according to the analysis in Sect. 2.2, the location of the correct key indicates the average number of

times to recover the correct key, which is a more concerned indicator for attackers in SCA. Therefore, referring to the research of GuilhermePerin [14] et al., this paper uses the guessing entropy to select the best model, according to formula (5):

$$h_{best} = \arg\min_{\lambda \in \wedge} GE(h_\lambda, T_{train}, T_{test}) \tag{5}$$

where $GE(h_\lambda, T_{train}, T_{test})$ represents the guessing entropy of the model h_λ.

In the process of ensemble, rather than simply selecting a particular best model, the integrated output class probabilities are calculated by aggregating multiple training models and summing up their output class probabilities. The probability of each candidate key is calculated by aggregating the probabilities of all individual models by using the bagging method to build the ensemble (see Sect. 2.1 for details). Then, the summation probability $S_e(k)$ of ensemble learning is calculated for each key byte candidate k, as shown in formula (6):

$$S_e(k) = \sum_{h=1}^{N} \sum_{i=1}^{Q} \log(p_{h,i,k}) \tag{6}$$

where N represents the number of machine learning models. $p_{h,j,k}$ refers to the output class probability of model h and trace i, according to the label j, the leakage model l, and the input pk_i. The models used in the ensemble process are all derived from the single available model generated during the training process, and they have been ranked according to the results computed on the test set according to formula (5).

3.3 Optimal Ensemble Parameter Selection Based on Genetic Algorithm

Genetic Algorithm (GA) is an optimization algorithm inspired by natural selection. It is a population-based search algorithm, which uses the concept of survival of the fittest in evolutionary theory to generate new populations by repeatedly using genetic operators on the individuals existing in the population. In GA, chromosome representation, selection, crossover, mutation and fitness function calculation are several key elements [24].

The implementation steps of GA to achieve model selection optimization are as follows: A population of m sets of hyperparameter combinations are randomly initialized, called H. Calculate the accuracy of the model formed by each hyperparameter combination in H. Based on the accuracy values, two models are selected from population H, namely C1 and C2. A single-point crossover operator with crossover chance (Cc) is applied on C1 and C2 to produce offspring, i.e. O. Then, the uniform mutation operator is applied on offspring (O) with mutation chance (Mc) to produce O'. All new offspring O' constitute a new population H'. The selection, crossover and mutation operations are repeated in the next generation population until the pre-set final generation is reached. GA dynamically changes the search process through the probability of crossover and mutation, and achieves the optimal solution. GA can modify encoded genes, can evaluate multiple individuals and generate multiple optimal solutions. Therefore, the genetic algorithm has a better global search ability. Since using GA to optimize the model is only the preparation before the ensemble learning, the accuracy is still used as the index

to evaluate the model, which ensures the computational efficiency of the whole process. The algorithm flow for pre-selection optimization of the ensemble model using GA is as follows:

Algorithm 1:***Model Optimization using Genetic Algorithm (GA)***

Input:

Population size: m

Maximum number of generations: MAX

Number of selected models: n

Output:

Global Best Models: H_1, H_2, … ,H_n

begin

Generate initial population of m models: $H_i(i=1,2,...,m)$

Set iteration count t= 0

Train models and compute the accuracy of each model

while*(t<MAX)*

Select a pair of chromosomes from the initial population

Crossover on selected pair with crossover chance

Mutation on the offspring with mutate chance

Replace the old generation with new generation

t = t+1

end while

return the top n models: H_1, H_2, … ,H_n

end

4 Experimental Results

In this experiment, all training processes use the Keras package in Python, and the GPU used is NVIDIA Tesla V100 GPU. an improved ensemble learning model based on genetic algorithms is employed to attack the power traces of the three datasets. For the data of each dataset, Perin's Random Ensemble and GA-based Ensemble are used to train the MLP model respectively, and calculate the guessing entropy and success rate of a single byte for comparison. So as to obtain a better ensemble effect.

4.1 ASCAD_f

For the ASCAD_f dataset, Fig. 1 and Fig. 2 show the guessing entropy and success rate of training the MLP model using the Hamming weight leakage model, respectively. First, the performance of both ensemble strategies improves with the increase of ensemble scale, and finally converges. When the improved ensemble learning model based on genetic algorithm is used, better convergence performance can be achieved when the ensemble scale reaches 30, while 40 models are required for random ensemble. In the case of the same ensemble scale, the ensemble method optimized by genetic algorithm needs fewer traces to converge, so it has better attack performance. Overall, in order to

achieve the same attack effect, the ensemble method optimized by the genetic algorithm uses 10 fewer models than the random ensemble. Therefore, the introduction of the genetic algorithm reduces the scale of the ensemble learning, thereby saving the training cost.

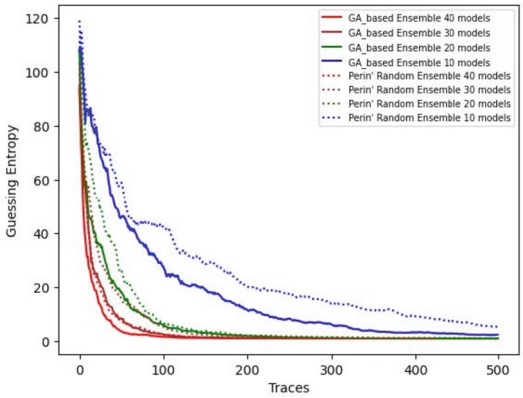

Fig. 1. Guessing Entropy of ASCAD_f dataset using Hamming weight model to train MLP model

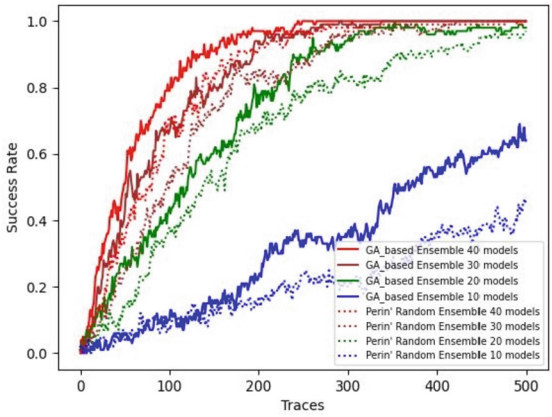

Fig. 2. Success Rate of ASCAD_f dataset using Hamming weight model to train MLP model

Figures 3 and 4 show the guessing entropy and success rate of training the MLP model using the byte leakage model. It can be seen that for the fixed key data in the ASCAD_f dataset, the byte model can attack the key more effectively. In this case, both strategies only need to integrate a small number of models (about 20) to achieve more ideal attack effect. When 20 models are integrated, the number of traces required for GE and SR convergence can be reduced by 24 and 64 using the genetic algorithm, respectively.

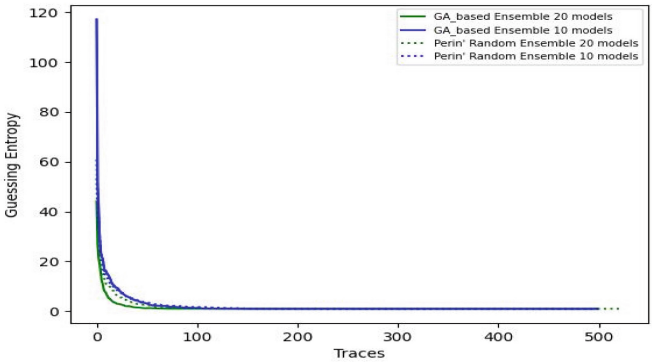

Fig. 3. Guessing Entropy of ASCAD_f dataset using Byte model to train MLP model

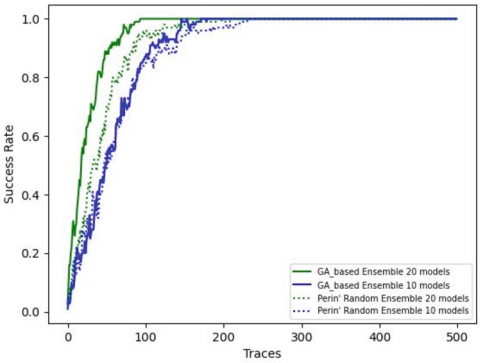

Fig. 4. Success Rate of ASCAD_f dataset using Byte model to train MLP model

4.2 ASCAD_r

Figures 5–8 show the results of training the MLP model using the Hamming weight and the byte leakage model, respectively, for the ASCAD_r dataset.

Figures 5 and 6 show the guessing entropy and success rate of training the MLP model using the Hamming weight leakage model. Similar to the results of ASCAD_f, the GA-based ensemble method requires fewer traces to converge. When 30 models are integrated, the GA-based ensemble reduces the number of traces required for GE convergence by 40 compared to the random ensemble, the number of traces required for SR convergence is reduced by 31. The overall scale of ensemble to achieve the same attack effect is also smaller.

Figures 7 and 8 show the guessing entropy and success rate of training the MLP model using the byte leakage model. Different from the ASCAD_f dataset, the overall attack effect using the byte leakage model is worse than that of the Hamming weight model. This is probably due to the fact that the keys in ASCAD_r are random, making it more difficult to use a byte-compromising model with more classes (256 classes). It is worth noting that in the random ensemble, GE did not reach convergence in the end,

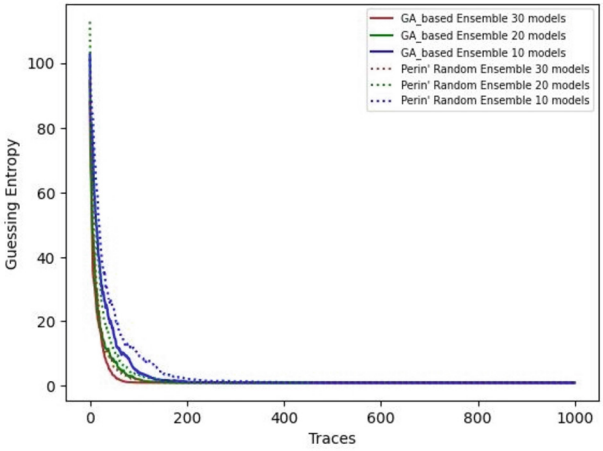

Fig. 5. Guessing Entropy of ASCAD_r dataset using Hamming weight model to train MLP model

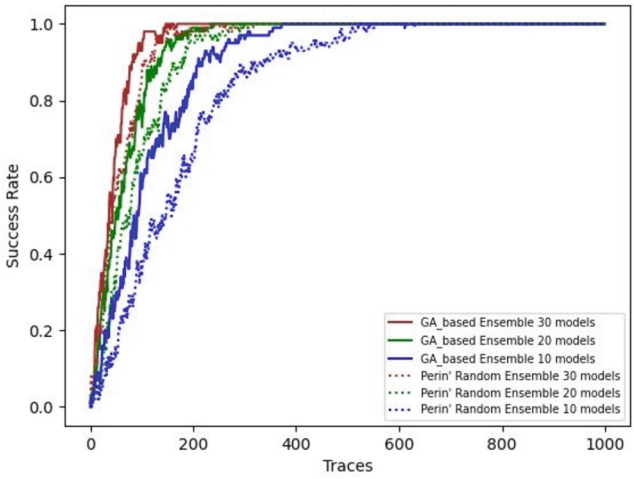

Fig. 6. Success Rate of ASCAD_r dataset using Hamming weight model to train MLP model

and the highest SR was only about 0.4. After using the genetic algorithm, not only GE could eventually converge, but SR could also reach 1. Therefore, when using the byte leakage model, using the genetic algorithm can greatly improve the attack effect.

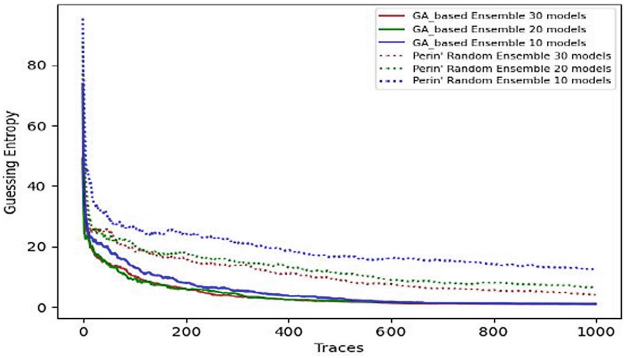

Fig. 7. Guessing Entropy of ASCAD_r dataset using byte model to train MLP model

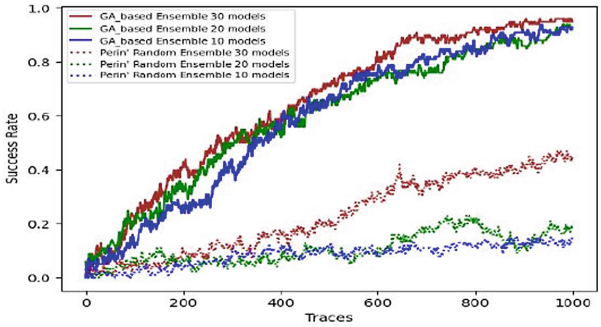

Fig. 8. Success Rate of ASCAD_r dataset using byte model to train MLP model

4.3 Comparison of Two Ensemble Methods Under Three Datasets.

Table 2 shows the comparison of three data sets in two integration schemes. It can be seen from the table that under the three data sets, the guess entropy of GA-based ensemble is better than that of Perin's random ensemble.

Table 2. Comparison of two ensemble methods under three datasets(MLP)

GE = 1 number of traces (Hamming weight model)										
Dataset	ASCAD_f					ASCAD_r				
Number of ensemble models	10	20	30	40	50	10	20	30	40	50
GA-based ensemble	395	275	120	97	99	174	96	68	62	60
Perin's random ensemble	850	340	244	130	102	236	177	108	69	68

5 Summary

This paper proposes a method to optimize the selection of the parameters of the ensemble multiple neurons through the genetic algorithm, which can significantly improve the effect of the ensemble. And on the three datasets, Perin's Random Ensemble and GA-based Ensemble are used to train the MLP model, and the guessing entropy and success rate of a single byte are calculated for comparison. The experiments in this paper show that, compared with Perin's Random Ensemble, the GA-based ensemble requires fewer traces to converge the guessing entropy and achieve a higher success rate when the ensemble size is the same. From the ensemble training process, in order to achieve the same attack performance, the introduction of the genetic algorithm can reduce the scale of ensembles by about 10, thereby saving the training cost.

Applying the analysis of ensemble ideas to a wider range of datasets and investigating the feasibility of introducing ensemble in larger neural network structures such as GANs are future research directions.

References

1. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25
2. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_9
3. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, çK., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3
4. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_3
5. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES. J. Cryptographic Eng. **5**(2), 123–139 (2015)
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2
7. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_27
8. Hospodar, G., et al.: Least squares support vector machines for side-channel analysis. Center Adv. Secur. Res. Darmstadt, 99–104(2011)
9. Hospodar, G., et al.: Machine learning in side-channel analysis: a first study. J. Cryptographic Eng. **1**(4), 293–302 (2011)
10. Heuser, A., Zohner, M.: Intelligent machine homicide. In: Schindler, W., Huss, S.A. (eds.) COSADE 2012. LNCS, vol. 7275, pp. 249–264. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29912-4_18
11. Martinasek, Z., Zeman, V.: Innovative method of the power analysis. Radioengineering **22**(2), 586–594 (2013)

12. Whitnall, C., Oswald, E.: Robust profiling for DPA-style attacks. In: Güneysu, T., Handschuh, H. (eds.) CHES 2015. LNCS, vol. 9293, pp. 3–21. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48324-4_1
13. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_3
14. Emmanuel, P., et al.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. CoRR 1–45 (2018)
15. Kim, J., et al.: Make some noise. Unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Trans. Cryptographic Hardware Embed. Syst. 148–179 (2019)
16. Hettwer, B., Gehrer, S., Güneysu, T.: Profiled power analysis attacks using convolutional neural networks with domain knowledge. In: Cid, C., Jacobson Jr., M. (eds. Selected Areas in Cryptography – SAC 2018. SAC 2018. Lecture Notes in Computer Science, vol. 10529, pp. 45–68. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-10970-7_22
17. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptographic Hardw. Embed. Syst. 107–131 (2019)
18. Wouters, L., et al.: Revisiting a methodology for efficient CNN architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems, 147–168(2020)
19. Perin, G., Chmielewski, Ł., Picek, S.: Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Trans. Cryptographic Hardw. Embed. Syst. 337–364 (2020)
20. Picek, S., et al.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Trans. Cryptographic Hardw. Embed. Syst. **2019**(1), 1–29 (2020)
21. Picek, S., et al.: Side-channel analysis and machine learning: a practical perspective. In: 2017 International Joint Conference on Neural Networks (IJCNN). IEEE (2017)
22. Friedman, J.H., Hall, P.: On bagging and nonlinear estimation. J. Stat. Planning Infer. **137**(3), 669–683 (2007)
23. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning: Methods, Systems, Challenges. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5
24. Katoch, S., Chauhan, S.S., Kumar, V.: A review on genetic algorithm: past, present, and future. Multimedia Tools Appl. **80**(5), 8091–8126 (2021)