Zhong Y, Feng JH, Cui XX *et al.* Machine learning aided key-guessing attack paradigm against logic block encryption. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 36(5): 1102–1117 Sept. 2021. DOI 10.1007/s11390-021-0846-6

Machine Learning Aided Key-Guessing Attack Paradigm Against Logic Block Encryption

Yi Zhong[1], Jian-Hua Feng[1], *Senior Member, CCF*, Xiao-Xin Cui[1,*], *Member, CCF, IEEE*, and Xiao-Le Cui[2], *Member, CCF*

[1] *Institute of Microelectronics, Peking University, Beijing 100871, China*
[2] *Key Laboratory of Integrated Microsystems, Peking University Shenzhen Graduate School, Shenzhen 518055, China*

E-mail: {zhongy, fengjh, cuixx}@pku.edu.cn; cuixl@pkusz.edu.cn

Received July 29, 2020; accepted August 25, 2021.

Abstract Hardware security remains as a major concern in the circuit design flow. Logic block based encryption has been widely adopted as a simple but effective protection method. In this paper, the potential threat arising from the rapidly developing field, i.e., machine learning, is researched. To illustrate the challenge, this work presents a standard attack paradigm, in which a three-layer neural network and a naive Bayes classifier are utilized to exemplify the key-guessing attack on logic encryption. Backed with validation results obtained from both combinational and sequential benchmarks, the presented attack scheme can specifically accelerate the decryption process of partial keys, which may serve as a new perspective to reveal the potential vulnerability for current anti-attack designs.

Keywords hardware security, logic encryption, machine learning, neural network, naive Bayes classifier

1 Introduction

In the course of achieving trusted and reliable application specific integrated circuit (ASIC) designs, hardware security remains as a paramount concern to be confronted over the past few decades. Emerging from its vulnerabilities to the attacks at different design levels, hardware security issue involves a wide range of countermeasures to enhance protection, where secret key based static authentication is one of the common techniques [1]. In this context, the original logic block is often equipped with a tamper-proof encryption module to authorize the user. The encryption methods may vary from simple logic gates to huge memory units. Logic block encryption offers several merits, including low design overhead and functional camouflage, thus mitigating the risks of malicious counterfeiting, piracy and hardware trojans [2].

However, despite the virtues above, logic block encryption may be greatly challenged by the novel attacks with the aid of machine learning (ML) [3–6]. To retrieve the keys in the encryption module, people used to believe that an adversary needs to verify every key combination; therefore the traversal complexity is exponential to the length of keys. Nevertheless, the stereotype should be abandoned as numerous algorithms, like feature extraction in a supervised learning task, can support the adversary in effectively mining the data derived from the logic block and interpreting the encryption trace. As a result, the key-guessing process is largely accelerated, while the protection techniques become invalid.

Most of the existing researches have concentrated on the application of machine learning in enhancing side-channel analysis (SCA) attacks. SCA attacks usually leverage physical information, such as processing time, power consumption and electromagnetic emanation, to reveal the cryptographic keys. However, the main is-

Regular Paper
Special Section of 2020 CCF Integrated Circuit Design and Automation Conference
This work was supported by the 111 Project under Grant No. B18001, the National Key Research and Development Program of China under Grant No. 2018YFB2202605, the Guangdong Science and Technology Project of China under Grant No. 2019B010155002, and the National Natural Science Foundation of China under Grant No. 61672054.
*Corresponding Author

sue remains as the feature extraction and classification problem for the physical trace data. In the literature, Hospodar *et al.*[3] exemplified firstly the usage of least squares support vector machines as the learning algorithm. Gilmore *et al.*[4] proposed principal component analysis for dimensional reduction and fed the data into a neural network to realize decryption. Besides, Maghrebi *et al.*[5] articulated the attack schemes that utilize convolutional neural network (CNN), autoencoder, long and short term memory (LSTM) and multilayer perceptron (MP). Moreover, Das *et al.*[6] constructed a fully-connected deep neural network to execute the SCA attack on multiple devices.

Whereas, it is notable to address the efficiency of ML-aided SCA attack which may be dramatically hindered by the fluctuation of power pin noise and power camouflage tricks[7, 8], which directly deteriorates the data quality and thus limits the learning accuracy. The ML-aided SCA attack also brings forward a demand of sampling precision and denoising ability for test facilities as well as data post processing, such as the alignment for power traces, which may restrict its feasibility and practicality.

Hence, for the sake of avoiding the flaws, we introduce the ML-aided attack techniques to another attack category, namely logic cryptanalysis, to implement decryption operation. In the context, the ML-aided attack shall mine the correlation trace of the exact logic values rather than physical parameters to deduce the cryptographic keys. The data source used for decryption can be easily and reliably obtained from the input/output pins by applying a certain volume of vector patterns.

In brief, the key contributions of this work can be concluded as follows.

- This work introduces the machine learning based attack to logic cryptanalysis rather than previous SCA-based counterparts, but also steps further to transform the decryption process to an optimization problem that can be handled by existing mathematical tools.
- This work establishes a standard logic key-guessing attack paradigm, and then elaborates how an adversary can use reasonable computing resources to implement a successful key-guessing attack against logic encryption.
- This work helps accelerate the decryption process by employing light-weighted ML methods to firstly deal with the "easy keys". As the keys remaining to be deciphered have been reduced to a small number after several epochs, the total decryption time consumed will be compressed.
- This work analyzes the potential logic correlation during the encryption phase, reveals the vulnerability of current logic block encryption, and finally helps enhance hardware security by exploring some possible countermeasures.

As for the organization, Section 2 presents the machine learning aided key-guessing attack paradigm and lists the related knowledge and methods. Section 3 implements the attack paradigm on ISCAS'85/89 and ITC'99 benchmarks to exemplify its feasibility and effectiveness. Section 4 discusses the underlying mathematical mechanism that supports our machine learning based logic attack and explores the potential anti-attack countermeasures, and Section 5 concludes the paper.

2 Preliminaries

2.1 Machine Learning Aided Attack Paradigm

As the major focus of this paper, we present a standard key-guessing attack paradigm. The paradigm rests upon the assumption of the existence of an encrypted netlist as well as an additional activated integrated circuit (IC), as depicted in Fig.1.

Definition 1 (Encrypted Netlist). *An encrypted netlist consists of the gate-level information of the encrypted IC, which can be obtained from reverse engineering.*

Definition 2 (Activated IC). *An activated IC is a functional circuit that produces resultant outputs when applying arbitrary input patterns, which can be purchased from the open market.*

The activated IC has been authorized, formally equivalent to the encrypted netlist when right keys are applied. Mathematically, the encrypted netlist and the activated IC can be expressed as the following logic functions, respectively:

$$\boldsymbol{O}_{\text{en}} = f(\boldsymbol{I}_{\text{en}}, \boldsymbol{K}_{\text{en}}), \text{ inverse function } \boldsymbol{K}_{\text{en}} = g(\boldsymbol{O}_{\text{en}}), \quad (1)$$

$$\boldsymbol{O}_{\text{right}} = f(\boldsymbol{I}_{\text{en}}, \boldsymbol{K}_{\text{right}}), \text{ inv. func. } \boldsymbol{K}_{\text{right}} = g(\boldsymbol{O}_{\text{right}}), \quad (2)$$

where the encrypted netlist $f(\cdot,\cdot)$ produces output $\boldsymbol{O}_{\text{en}}$ when applying input $\boldsymbol{I}_{\text{en}}$ and key $\boldsymbol{K}_{\text{en}}$, and functions the same as the activated IC when right key $\boldsymbol{K}_{\text{right}}$ is applied. In view of the independence between $\boldsymbol{I}$ and $\boldsymbol{K}$ as they are both input variables, the inverse function $\boldsymbol{K} = g(\boldsymbol{I}, \boldsymbol{O})$ can be further simplified as $\boldsymbol{K} = g(\boldsymbol{O})$.

Generally, only when the inverse function $g(\cdot)$ is explicitly expressed can we reveal the keys $\boldsymbol{K}_{\text{right}}$. How-

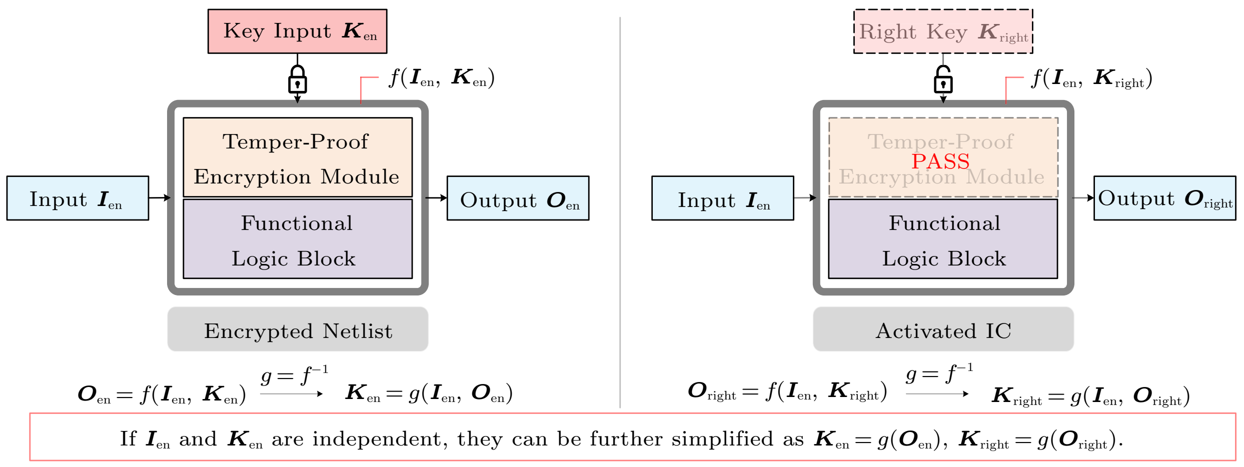

Key Input $\boldsymbol{K}_{\text{en}}$
$f(\boldsymbol{I}_{\text{en}}, \boldsymbol{K}_{\text{en}})$
Temper-Proof Encryption Module
Functional Logic Block
Input $\boldsymbol{I}_{\text{en}}$
Output $\boldsymbol{O}_{\text{en}}$
Encrypted Netlist
$\boldsymbol{O}_{\text{en}} = f(\boldsymbol{I}_{\text{en}}, \boldsymbol{K}_{\text{en}}) \xrightarrow{g=f^{-1}} \boldsymbol{K}_{\text{en}} = g(\boldsymbol{I}_{\text{en}}, \boldsymbol{O}_{\text{en}})$
Right Key $\boldsymbol{K}_{\text{right}}$
$f(\boldsymbol{I}_{\text{en}}, \boldsymbol{K}_{\text{right}})$
Temper-Proof
PASS
Encryption Module
Functional Logic Block
Input $\boldsymbol{I}_{\text{en}}$
Output $\boldsymbol{O}_{\text{right}}$
Activated IC
$\boldsymbol{O}_{\text{right}} = f(\boldsymbol{I}_{\text{en}}, \boldsymbol{K}_{\text{right}}) \xrightarrow{g=f^{-1}} \boldsymbol{K}_{\text{right}} = g(\boldsymbol{I}_{\text{en}}, \boldsymbol{O}_{\text{right}})$
If $\boldsymbol{I}_{\text{en}}$ and $\boldsymbol{K}_{\text{en}}$ are independent, they can be further simplified as $\boldsymbol{K}_{\text{en}} = g(\boldsymbol{O}_{\text{en}})$, $\boldsymbol{K}_{\text{right}} = g(\boldsymbol{O}_{\text{right}})$.

Fig.1. Standard attack paradigm against logic block encryption. Note that the inverse function can be further simplified for the independent variables $\boldsymbol{I}$ and $\boldsymbol{K}$.

ever, it proves solving the inverse function $g(\cdot)$ is even harder than NP-complete problems [9].

Despite the complexity of figuring out a precise solution, machine learning can deduce the right keys in a totally different way. Basically, when provided with sufficient logic trace data, one can use supervised learning to construct a fitting function $r(\cdot)$ for the target function $g(\cdot)$. The data pairs $(\boldsymbol{O}_{\text{en}}, \boldsymbol{K}_{\text{en}})$ collected from the encrypted netlist, serving as labelled training datasets, are learned in the training phase, while the data $\boldsymbol{O}_{\text{right}}$ derived from the activated IC, serving as the test datasets, will be substituted to the trained fitting function $r(\cdot)$ so as to approximately estimate the key values $\boldsymbol{K}_{\text{right}}$ as $r(\boldsymbol{O}_{\text{right}})$ in the inference phase.

2.2 Logic Encryption

In general, logic encryption refers to a built-in locking mechanism by inserting additional key gates to accomplish encryption within a logic block, where encryption nodes are carefully chosen and adequately mixed with the original circuit [2]. Fig.2 depicts a simple implementation for logic encryption. By mixing XOR/XNOR gates or MUXs to the original circuit, the encryption units will serve as buffers or inverters (or fault nodes) according to the key values. Only when the correct keys $K_1 = 1$, $K_2 = 0$, $K_3 = 1$ and $K_4 = 0$ are applied will the modified circuit work properly.

Logic encryption outperforms other encryption

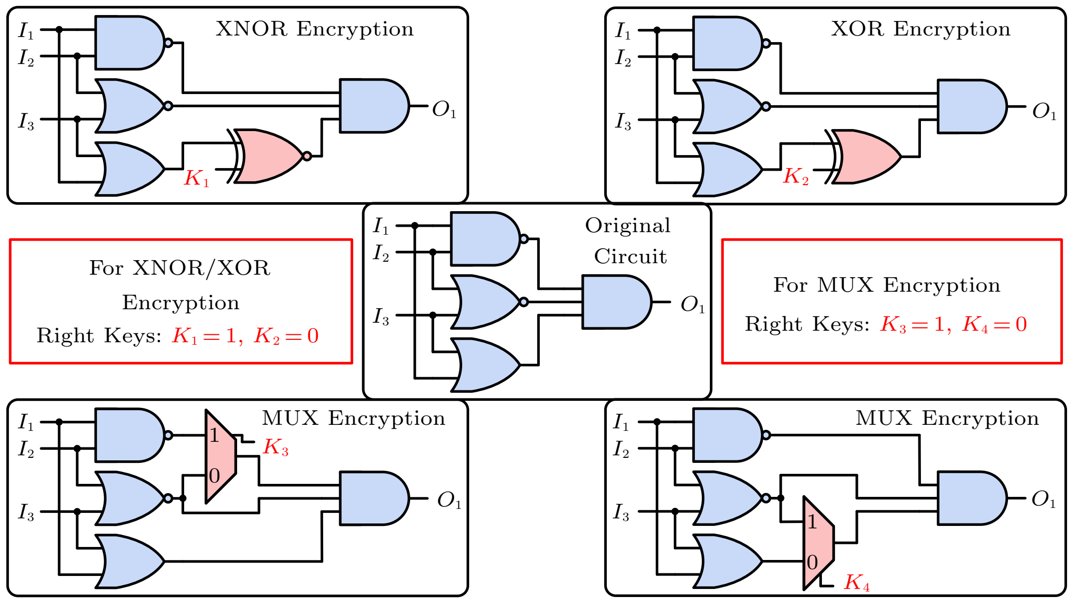


Fig.2. Logic encryption, where K_1 and K_2 are encrypted with XNOR/XOR gates, while K_3 and K_4 with MUXs.

methods for its convenience and low overhead to implement as well as obfuscation with the original circuits[9, 10]. It has already been implemented to both combinational and sequential circuits, e.g., full adder and subtractor[11], pseudorandom number generator (PRNG)[12], flip-flop[13], and scalable multi-module designs[14].

On account of the selection of the cryptographic nodes, EPIC[9], the first to present the concept of logic encryption, suggested inserting additional key gates randomly to enhance circuit security. Later work, namely the fault analysis based logic encryption in [10], chose the highest fault impact (FI) nodes to maintain 50% Hamming distance. [15] modifies the strategy by selecting a majority of keys based on FI while the others for preventing path sensitization. Other innovations on decreasing the overhead of logic encryption were previously discussed in reusing key-based logic gates[16] and low-overhead implementation[17].

Previous researches have demonstrated effective attacks on logic encryption. Except for SCA attacks like differential power analysis (DPA) in [18], they can be roughly divided into two categories: topology-indispensable or not. The former may refer to the sensitization attack[19] (which propagates key ports to be observed on outputs) and the logic cone analysis attack[20] (which distinguishes logic cones, then "divide-and-rule"). They must resort to specific logical structures to sensitize key information. Hence their attack efficiency highly hinges on the topology of the target logic block. In contrast, the Hill-climbing[21] or the Boolean satisfiability (SAT) attack[22] performs only on the external input, key and output ports regardless of the detail of the netlist. Hence mitigating this type of attacks has become the so-called hot topics in recent years[23–28]. It is notable that authors in [21] and [22] already adopted similar principles widely used in machine learning, such as the gradient descent algorithm. In this work, we step further to present the attack paradigm with general machine learning methods.

2.3 Neural Network

A fully-connected (FC) neural network (NN) consists of an input layer, several hidden layers and an output layer (in Fig.3). Between adjacent layers, it organizes as every neuron is connected to all the neurons in its front and back layers. The training method we use here is the feed-forward backprop (BP) algorithm, which is the most common in supervised learning and is well supported by current mathematical tools.

In the forward phase, the network propagates input vectors forward and produces resultant output vectors. Each neuron sums up all the weighted inputs and modulates the sum through a nonlinear activation function, tanh(·) in (3). As the output vector $\boldsymbol{o}$ undergoes a comparison process with the labelled vector $\boldsymbol{t}$, loss function E is derived in (4).

$$y_j = a\left(\sum_{i=0}^{n} w_{ji} \times x_i\right), \quad \text{where } a(\cdot) = \tanh(\cdot). \tag{3}$$

$$\text{global loss function: } E = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2. \tag{4}$$

In the backward phase, the network propagates the loss function E back and adjusts the weights based on the gradient descent algorithm, where η is the learning

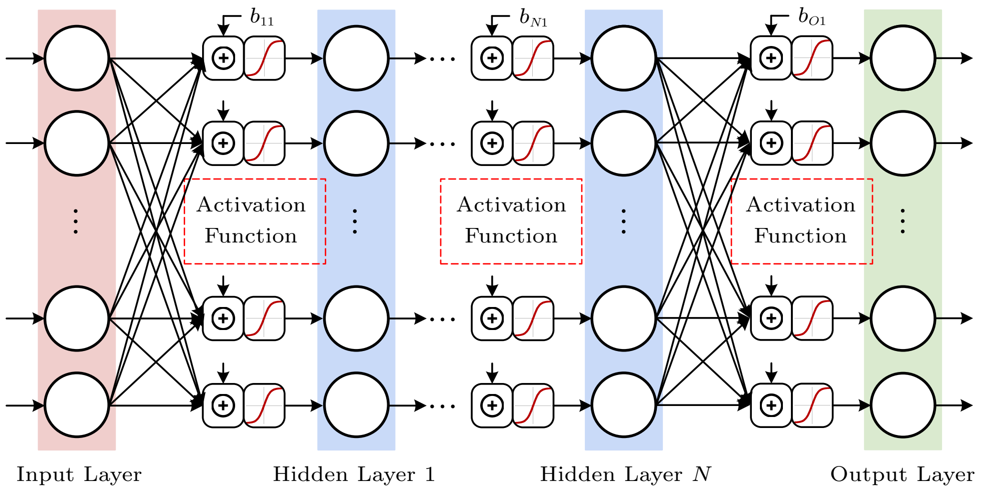


Fig.3. FC neural network trained with the BP algorithm.

rate in (5):

$$\text{weight updating rule: } \Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}. \quad (5)$$

2.4 Naive Bayes Classifier

Naive Bayes classifier is a widely adopted tool for binary classification. Given the fitting function $r(\cdot)$ derived from the feature variables $\boldsymbol{x}$, Bayes classification rule h^* is decided by:

$$y^* = h^*(\boldsymbol{x}) = \begin{cases} 1, & \text{if } r(\boldsymbol{x}) > 1/2, \\ 0, & \text{if } r(\boldsymbol{x}) < 1/2. \end{cases} \quad (6)$$

Naive Bayes classifier is known as the optimal classifier when assuming strong independence between the variables. Once the fitted values $r(\boldsymbol{x})$ for secret keys are obtained, the naive Bayes classifier will judge the keys as logic 1 if their values are above 1/2 and vice versa, as a typical binary classification problem.

For a more comprehensive look at the preliminaries mentioned above, hereinafter we will address more about their roles. Firstly, on the basis of the assumption of our attack paradigm, the kernel problem has been transformed to solving the inverse function $g(\cdot)$ as the secret keys can be derived from $\boldsymbol{K}_{\text{right}} = g(\boldsymbol{O}_{\text{right}})$, where the circuits are encrypted by logic encryption. Secondly, in order to deduce a possible fitting function for $g(\cdot)$ in reasonable time, we introduce the neural network trained and inferred by using the datasets collected from the encrypted netlist and the activated IC, respectively. The reason why we use the neural network lies on the issue that the network architecture can be easily manipulated, which helps extract some high dimensional features if not too overfitted. Thirdly, after training, the fitted value $r(\boldsymbol{x})$ for right keys $\boldsymbol{K}_{\text{right}}$ can be derived by substituting $\boldsymbol{O}_{\text{right}}$ into the trained neural network, but generally it produces an output within the range of [0, 1]. In order to classify them as exact logic 1 or 0, naive Bayes classifier is finally adopted to binarize $r(\boldsymbol{x})$.

To facilitate reading, we list the most frequently used symbols in Table 1.

3 Implementation

3.1 Procedure of the Attack Paradigm

In terms of the implementation procedure, Fig.4 exemplifies how an adversary conducts a machine learning aided attack against logic block based encryption. 1) The target circuits are encrypted by a specific node-selecting strategy beforehand, i.e., random inserting or the highest fault impact (FI) encryption with XORs or MUXs. 2) By applying random input vectors $\boldsymbol{I}_{\text{en}}$ and random key vectors $\boldsymbol{K}_{\text{en}}$, resultant training datasets ($\boldsymbol{O}_{\text{en}}$, $\boldsymbol{K}_{\text{en}}$) as well as test dataset $\boldsymbol{O}_{\text{right}}$ are collected. 3) A certain machine learning algorithm is adopted to calculate the fitting function for inverse function $g(\cdot)$, and here the neural network is employed, where the number of the neurons in the input and the output layer is consistent with the number of output nodes and key nodes, respectively. 4) Finally, the keys values inferred by substituting $\boldsymbol{O}_{\text{right}}$ into $r(\cdot)$ of the trained network are fed into the naive Bayes classifier to accomplish classification.

Table 1. Quick Reference to Frequently-Used Symbols

Symbol	Meaning
$f(\cdot,\cdot)$	Logic function of encrypted netlist, independent variables: $\boldsymbol{I}$ and $\boldsymbol{K}$, dependent variable: $\boldsymbol{O}$
$g(\cdot)$	Inverse function of $f(\cdot,\cdot)$, independent variable: $\boldsymbol{O}$, dependent variable: $\boldsymbol{K}$
$r(\cdot)$	Fitting function of $g(\cdot)$, derived from neural network
$\boldsymbol{I}_{\text{en}}$	Random vectors applied on the encrypted input nodes
$\boldsymbol{K}_{\text{en}}$	Random vectors applied on the encrypted key nodes
$\boldsymbol{K}_{\text{right}}$	Right vectors applied on the encrypted key nodes
$\boldsymbol{O}_{\text{en}}$	Resultant output response when applying random keys
$\boldsymbol{O}_{\text{right}}$	Resultant output response when applying right keys

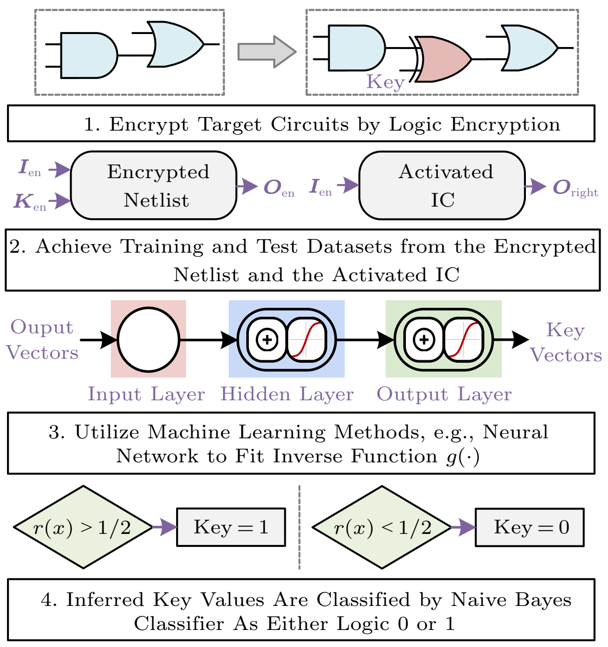

Key
1. Encrypt Target Circuits by Logic Encryption
$\boldsymbol{I}_{\text{en}}$
$\boldsymbol{K}_{\text{en}}$
Encrypted Netlist
$\boldsymbol{O}_{\text{en}}$
$\boldsymbol{I}_{\text{en}}$
Activated IC
$\boldsymbol{O}_{\text{right}}$
2. Achieve Training and Test Datasets from the Encrypted Netlist and the Activated IC
Ouput Vectors
Input Layer
Hidden Layer
Output Layer
Key Vectors
3. Utilize Machine Learning Methods, e.g., Neural Network to Fit Inverse Function $g(\cdot)$
$r(x) > 1/2$
Key = 1
$r(x) < 1/2$
Key = 0
4. Inferred Key Values Are Classified by Naive Bayes Classifier As Either Logic 0 or 1

Fig.4. Procedure of the attack paradigm.

The encryption algorithm is embodied as Algorithm 1. A designer may choose to insert either XOR/XNOR or MUXs on random or the highest FI location as key gates arbitrarily. For the latter, the highest FI for an internal node i is formulated as (7) in [14]:

$$FI_i = (NoP_0 \times NoO_0 + NoP_1 \times NoO_1), \qquad (7)$$

where NoP_0 denotes the number of patterns that detect stuck-at-0 fault at the output and NoO_0 denotes the total number of the output bits that get affected by the fault, while NoP_1 and NoO_1 represent the stuck-at-1 fault and the total number of the affected output bits correspondingly.

Algorithm 1. Logic Block Encryption Algorithm [9, 14]

```
Input: all the internal nodes of a logic block;
Output: encrypted nodes and key values;
    // For random insertion
1: for ($i := 1$, $i \leqslant$ the number of the keys, $i++$)
2:     Randomly select an internal node and insert the key gate here;
3: end for
4: Finish encryption when reaching the target number of keys;
    // For the highest fault impact (FI) insertion
1: for ($i := 1$, $i \leqslant$ the number of the internal nodes, $i++$)
2:     Detect the s-a-0 fault ($NoP_0$) and the affected bit ratio ($NoO_0$);
3:     Detect the s-a-1 fault ($NoP_1$) and the affected bit ratio ($NoO_1$);
4:     Compute $FI_i := (NoP_0 \times NoO_0 + NoP_1 \times NoO_1)$;
5: end for
6: Select the highest FI nodes to encrypt using XOR/XNOR gates;
```

A node with the highest FI can affect most of the outputs for most of the input patterns if a fault occurs. FI is used to enhance the Hamming distance in [14]. Mathematically, (7) should be equivalent to the total affected bits when one reverses the logical value of a certain node. Therefore, one can reverse each internal node and apply random vectors on the inputs to figure out how many outputs are affected by the reversed node, and then find the nodes with the highest impact. Here Table 2 displays the internal nodes with the highest FI values for the C432 benchmark for instance, which has been regularized to [0, 1]. The selected nodes are encrypted by either logic 1 or 0 arbitrarily.

Table 2. Highest FI Nodes & Applied Key Values on C432

Key	Node	FI	Value
K_1	N_{199}	0.491	1
K_2	N_{296}	0.407	0
K_3	N_{357}	0.303	1
K_4	N_{203}	0.285	1
K_5	N_{381}	0.278	0
K_6	N_{386}	0.238	1
K_7	N_{393}	0.200	1
K_8	N_{356}	0.186	0
K_9	N_{348}	0.186	0
K_{10}	N_{355}	0.185	1
K_{11}	N_{354}	0.185	1
K_{12}	N_{353}	0.185	1
K_{13}	N_{352}	0.183	1
K_{14}	N_{351}	0.182	0
K_{15}	N_{350}	0.181	1
K_{16}	N_{349}	0.181	0
K_{17}	N_{422}	0.177	1
K_{18}	N_{425}	0.176	0
K_{19}	N_{260}	0.164	0
K_{20}	N_{285}	0.163	1

Afterwards, the decryption process is executed as the flow described in Algorithm 2. Firstly, the encrypted netlist and the activated IC are fed with random input vectors ($\boldsymbol{I}_{\text{en}}$, $\boldsymbol{K}_{\text{en}}$) and $\boldsymbol{I}_{\text{en}}$ to produce the resultant outputs $\boldsymbol{O}_{\text{en}}$ and $\boldsymbol{O}_{\text{right}}$, respectively, in which the volume of the two datasets equals the number of random vectors. For training the neural network, the training datasets ($\boldsymbol{O}_{\text{en}}$, $\boldsymbol{K}_{\text{en}}$) need to be observed and collected on the encrypted netlist. In the meanwhile, the test dataset $\boldsymbol{O}_{\text{right}}$ should be obtained on the activated IC. Next, one utilizes the BP algorithm to minimize the loss function of the network by feeding the training pairs ($\boldsymbol{O}_{\text{en}}$, $\boldsymbol{K}_{\text{en}}$). Once training gets finished, test data $\boldsymbol{O}_{\text{right}}$ will be substituted to the input layer in a typical inference phase. As the network has been trained to a fitting function $r(\cdot)$ for $g(\cdot)$, the output layer shall produce the inferred values $r(\boldsymbol{O}_{\text{right}})$ for $\boldsymbol{K}_{\text{right}}$. The values $r(\boldsymbol{O}_{\text{right}})$, being decimals within the range of [0, 1] in general, shall be further binarized as exact logic 1 or 0, where the quantization problem is tackled by employing the naive Bayes classifier with a decision boundary of 0.5. Ultimately the decryption process can be accomplished, and the binary values for those cryptographic keys can be determined.

Algorithm 2. Logic Block Decryption Algorithm

```
Input: $\boldsymbol{O}_{\rm en}$, $\boldsymbol{K}_{\rm en}$ for the encrypted circuit and $\boldsymbol{O}_{\rm right}$ for the activated circuit;
Output: retrieved key values $\boldsymbol{K}_{\rm right}$;
1: for ($k := 1, k \leqslant$ the number of the input/output pair vectors, $k++$)
2:     Train the neuron network by using the training dataset ($\boldsymbol{O}_{\rm en}, \boldsymbol{K}_{\rm en}$);
3: end for
4: for ($k := 1, k \leqslant$ the number of the input/output pair vectors, $k++$)
5:     Infer the key values $\boldsymbol{K}_{\rm right}$ by substituting test dataset $\boldsymbol{O}_{\rm right}$;
6: end for
7: Judge the keys $\boldsymbol{K}_{\rm right}$ as logic 0 or 1 by the Naive Bayes classifier;
```

3.2 Attack Demonstration on C432 Benchmark

For a more comprehensive illustration of the decryption procedure, hereinafter, we will employ the C432 benchmark as an example to expound the attack details based on Fig.5.

The target C432 benchmark contains 36 inputs and 7 outputs as well as 20 key nodes whose values are encrypted as Table 2 (logic 0: encrypted with XOR; logic 1: encrypted with XNOR).

Above all, as Fig.5(a) draws, a random vector generator generates a set of (50%, 50%) vectors $\boldsymbol{I}_{\rm en}$, $\boldsymbol{K}_{\rm en}$ (50% probability of being logic 0 and 50% probability of being logic 1) for the encrypted netlist, and the same $\boldsymbol{I}_{\rm en}$ for the activated IC, where the number of inputs, keys and outputs are 36, 20, and 7, respectively. As this is the first epoch, no keys have been retrieved before, hence they are all fed with random vectors and expressed as "?".

Upon reaching a certain volume of random vectors, the training pair ($\boldsymbol{O}_{\rm en}$, $\boldsymbol{K}_{\rm en}$) is then fed into the neural network to get trained, serving as its input layer and target output label, respectively. The number of input and output neurons is natural to decide, which should be equal to the number of C432 outputs and keys. The main issue lies on the hidden layers. In order to determine a reasonable network structure, here in Fig.5(b) we have tested a so-called typical network (structured as 7-100-20), a fat network (7-10000-20) and a 10-hidden-layer deep network (7-100-...-100-20) to fit inverse function $g(\cdot)$. Results show that the loss function value does not decline more as the width enlarges or the depth deepens. They are all around 0.213 as depicted in Fig.5(c). A possible reason for this is overfitting in these large and deep networks. Therefore, in the context of efficiency and performance, a typical 3-layer network is chosen as a reasonable network in our implementation hereinafter.

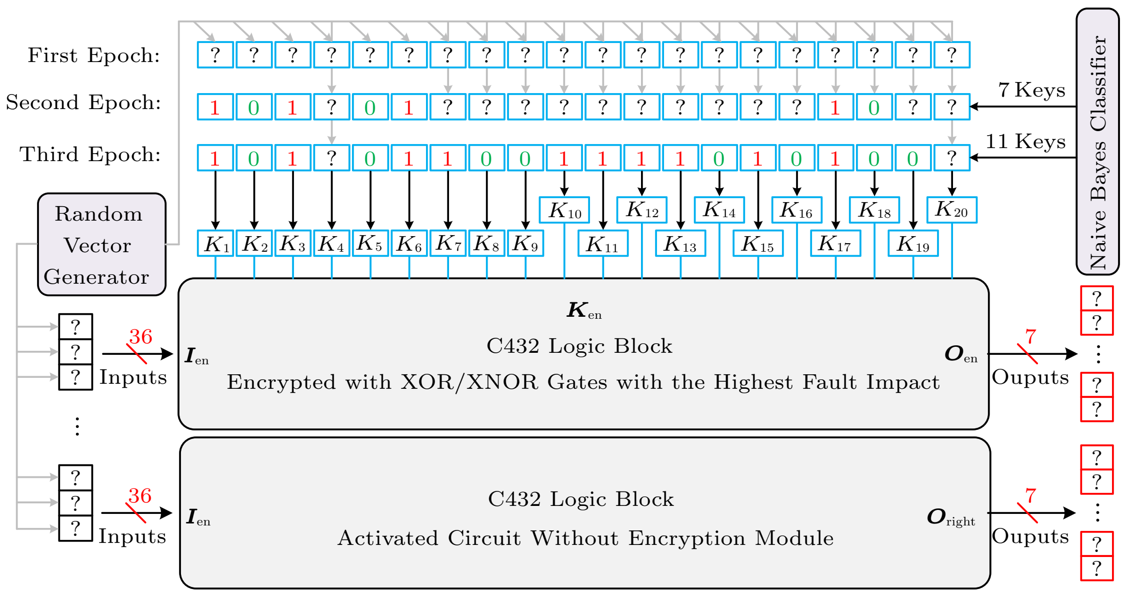
Once the training gets finished, the test data $\boldsymbol{O}_{\rm right}$ is substituted into the input layer, and one can collect the expected values for $\boldsymbol{K}_{\rm right}$ from the output layer of the trained network. As the network output, $r(\boldsymbol{O}_{\rm right})$ is generally a value within [0, 1]; thus it will undergo a classification process in terms of its exact value, in which the decimal will be binarized as logic 0 or 1 by naive Bayes classifier. The full attack flow above is summarized in Fig.5(d).

Note that in each epoch, there will be part of keys being retrieved. During the next epoch, those keys should be fixed as their inferred values while the rest of keys will still be applied random vectors. As an iterative process, decryption is conducted successively. For better comprehension, Figs.6(a)–6(c) illustrate the decryption implementation conducted on the C432 benchmark.

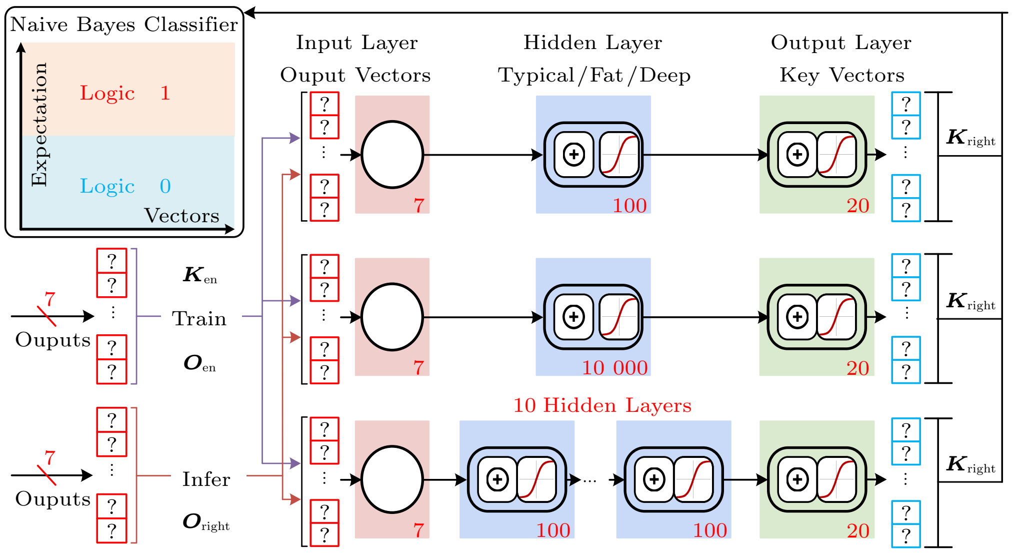
In the first epoch, seven keys are explicitly revealed in comparison with the decision boundary of 0.5, in which K_1, K_6, K_{17} and K_3 are logic 1 while K_2, K_{18} and K_5 are logic 0. Then the seven key values are fixed while the rest keys are still applied by the random vectors (expressed as "?" in Fig.5(a)). The reason why we fix the retrieved ones is that they conceal the correlation between the benchmark outputs with the rest key nodes. As the epoch goes on, more information about the secret keys will be exposed. Hence in the second epoch, 11 keys can be retrieved and fixed; while in the third epoch, as 18 keys have been fixed, the last two keys are revoked, ultimately resolving all the 20 cryptographic keys.

The evolution curves delineated in Fig.6 are achieved by increasing the training pair vectors gradually, ranging from 5 to 100 000. It is notable that those curves can converge within 1 000–5 000 vectors, where the time used for training is much less than 1s. With regard to the key values listed in Table 2, the proposed neural network succeeds in retrieving all the keys correctly.

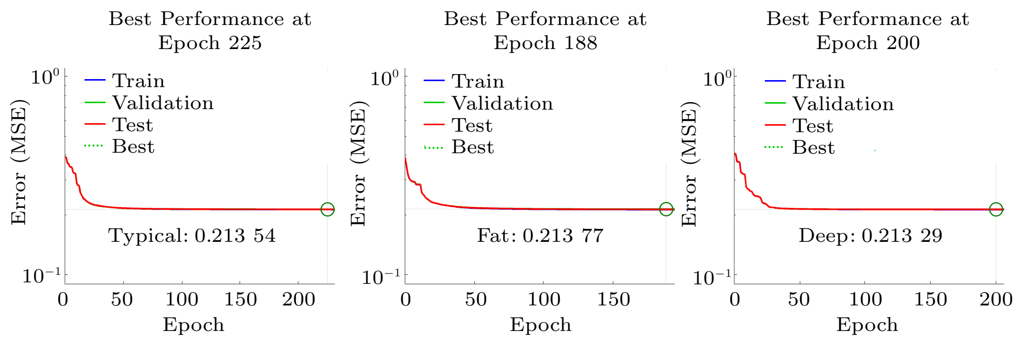
To further testify the efficiency of the proposed attack paradigm on the C432 benchmark encrypted with other techniques, including different node selection methods like random inserting or the highest fault impact inserting and encryption gates like XOR/XNOR and MUXs, we conduct the attack implementation as summarized in Table 3.


First Epoch:
Second Epoch:
Third Epoch:
7 Keys
11 Keys
Naive Bayes Classifier
Random Vector Generator
K_1 … K_{20}
$\boldsymbol{K}_{\rm en}$
C432 Logic Block
Encrypted with XOR/XNOR Gates with the Highest Fault Impact
C432 Logic Block
Activated Circuit Without Encryption Module
36 Inputs
$\boldsymbol{I}_{\rm en}$
$\boldsymbol{O}_{\rm en}$
$\boldsymbol{O}_{\rm right}$
7 Ouputs

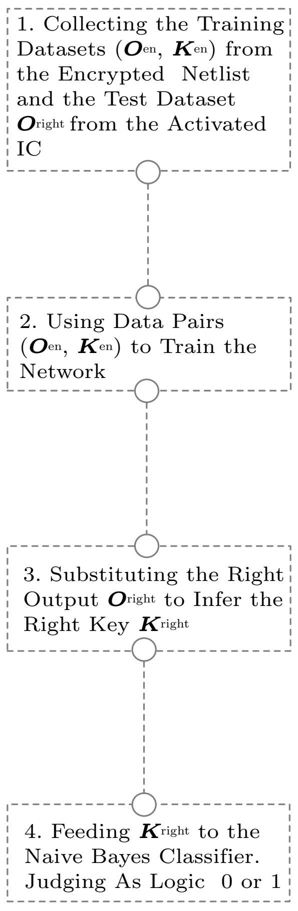
(a)



(b)

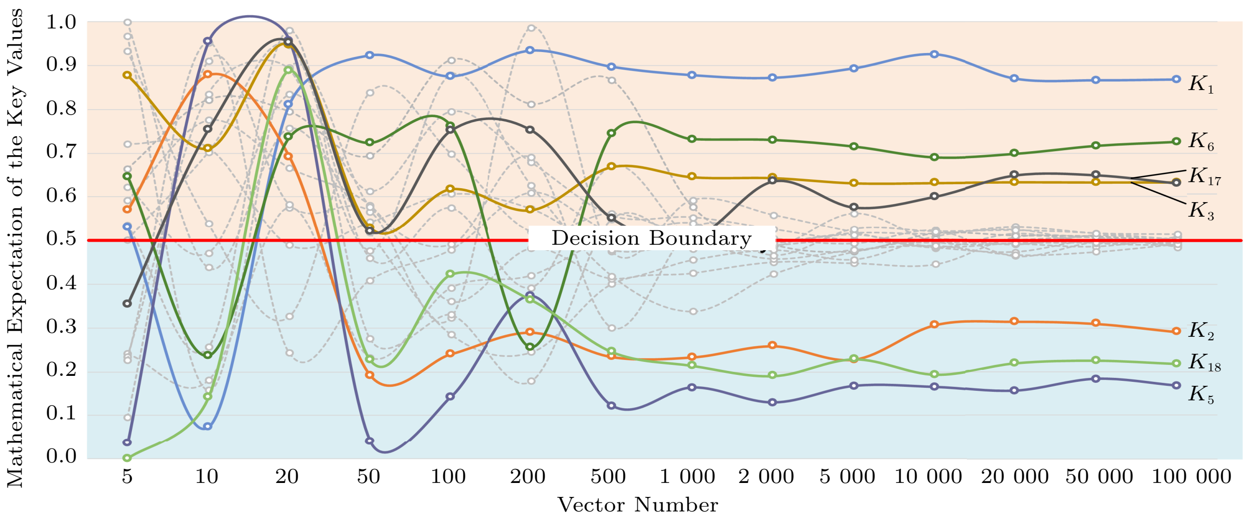


(c)

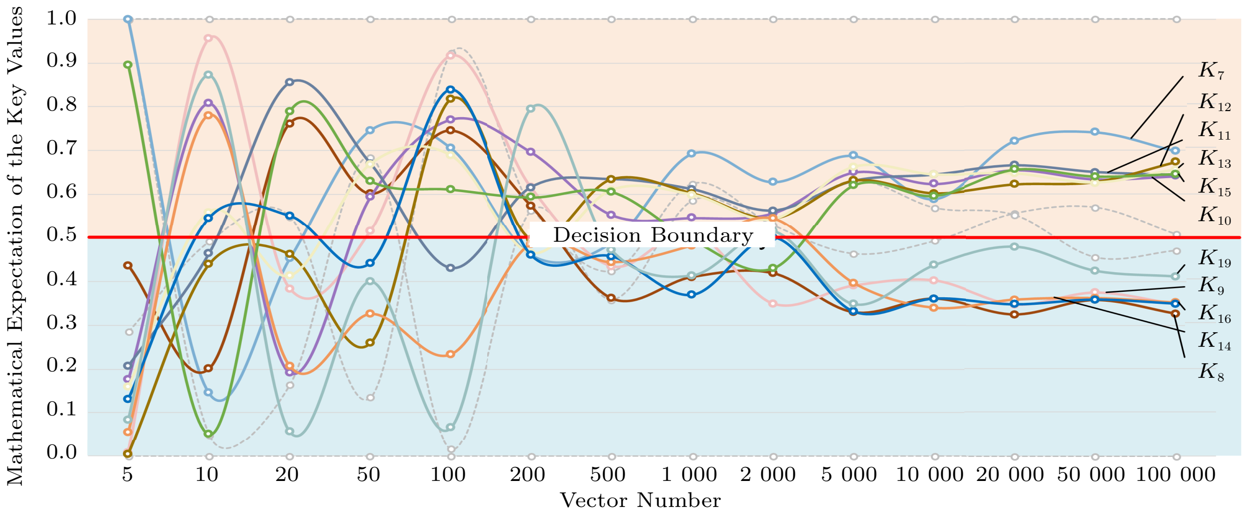

1. Collecting the Training Datasets ($\boldsymbol{O}_{\rm en}$, $\boldsymbol{K}_{\rm en}$) from the Encrypted Netlist and the Test Dataset $\boldsymbol{O}_{\rm right}$ from the Activated IC
2. Using Data Pairs ($\boldsymbol{O}_{\rm en}$, $\boldsymbol{K}_{\rm en}$) to Train the Network
3. Substituting the Right Output $\boldsymbol{O}_{\rm right}$ to Infer the Right Key $\boldsymbol{K}_{\rm right}$
4. Feeding $\boldsymbol{K}_{\rm right}$ to the Naive Bayes Classifier. Judging As Logic 0 or 1

(d)

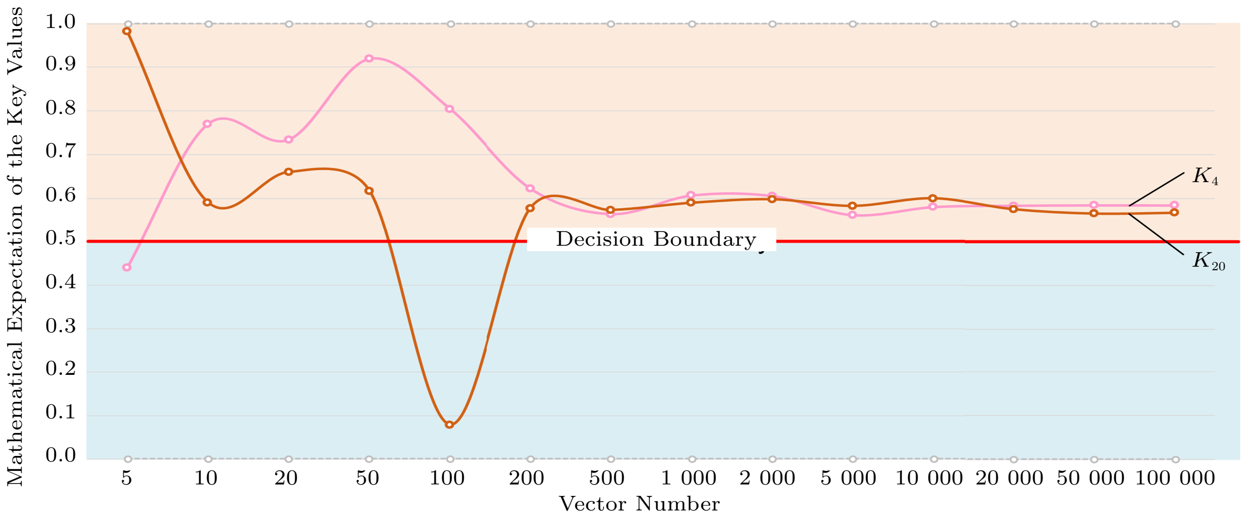
Fig.5. ML-based attack on the C432 benchmark. (a) Collected datasets and retrieved keys. (b) Training and inference with typical, fat or deep neural networks. (c) Loss function curve for the three networks. (d) Attack implementation flow.



(a)



(b)



(c)

Fig.6. Decrypting C432 benchmark within three epochs. (a) Retrieved keys in the 1st epoch: K_1–K_3, K_5, K_6, K_{17} and K_{18}. (b) Retrieved keys in the 2nd epoch: K_7–K_{16} and K_{19}. (c) Retrieved keys in the 3rd epoch: K_4 and K_{20}.

Table 3. Attack Implementation Summary on C432

Encryption Method	1st Iteration	2nd Iteration	3rd Iteration
C432_EPIC_XOR	9/40	13/40	17/40
C432_FI_XOR	9/40	27/40	28/40
C432_EPIC_MUX	10/40	16/40	20/40
C432_FI_MUX	16/40	35/40	40/40

Note: C432_EPIC_XOR means that the benchmark is C432, the encryption method is EPIC (random inserting) and the inserted logic gate is XOR/XNOR. x/y: number of retrieved keys/number of total encrypted keys.

For the four cases listed above, the proposed attack scheme performs better on FI inserting than on random inserting, and on MUXs than on XOR/XNOR encryption. When encrypted by FI inserting and MUXs, one can reveal all the keys within three epochs even when the key number increases to 40. A possible reason for better performance on FI inserting can refer to the augment on Hamming distance, as the encryption method unconsciously amplifies the correlation between the key nodes with the outputs. For MUXs, compared with XOR/XNOR gates, there is an additional fault node to be selected, which may also introduce unconscious statistical correlation between two logic cones.

3.3 Extending to Other Benchmarks

To further demonstrate the efficiency of the proposed attack paradigm, the same attack flow is implemented on the ISCAS'85, ISCAS'89 and ITC'99 benchmarks. The target logic blocks contain both combinational and sequential circuits with various topographies and have been successively used as a basis for comparing results in the area of test generation or circuit validation. For strictly validating the proposed attack scheme, encryption is conducted by referring to EPIC [9] to randomly insert XOR and XNOR gates at internal nodes (because decryption may be harder in this case according to Table 3). The number of the secret keys varies with the size of the benchmarks, ranging from 4 for the smaller circuit C17 to 100 for larger circuit C2670 and B04.

In practical terms, the proposed attack paradigm exhibits a different attack efficiency on each benchmark, which is reasonable in view of their unique topological structures. Some representative key curves in their first epoch are depicted in Fig.7. For each benchmark, the input, output and key port layouts are displayed in Fig.7(a). And the evolution curves for secret keys are portrayed in Fig.7(b). As the vector number (horizontal axis in log scale) increases from 5 to 100 000, the evolution curves may firstly oscillate around the classification boundary 0.5 due to random fluctuation. Then when the vector number gets larger, conservatively like 20 000 vectors (in vertical dash line), it will be feasible to obtain a converged evolution curve in which the expectation for each key takes on a stable value, probably above or below the boundary.

Fig.8 describes the time consumption to implement the attack flow on the above benchmarks for one epoch. Each data box describes the time distribution for 10 different attempts. The plots are divided into blue, green and purple groups, representing 20 000, 50 000 and 100 000 training pairs, respectively. As 20 000 vectors are conservative enough to obtain a converged evolution curve, the time consumed is quite acceptable.

Table 4 summarizes the implementation results on the benchmarks. The same attack is conducted for three epochs with each testing up to 100 000 vectors. The attack efficiency may hinge on the topography of the benchmarks, as the attack scheme is most suitable for retrieving the keys that expose most information to the outputs, namely "easy keys". The ratio of the keys retrieved is relatively low, which is reasonable when considering the strict condition, i.e., randomly inserting XOR/XNOR gates. There are cases in which a wrong key has no effect on the outputs, and the correct key combination is more than one. Moreover, more keys may be successively retrieved in the next epochs. But in view of their relatively low confidence level, we ignore that.

Table 4. Attack Implementation Summary

Circuit	Input	Output	Gate	DFF	Retrieved Keys in 3 Epochs
C17	5	2	6	0	4/4
C432	36	7	160	0	17/40
C499	41	32	202	0	12/50
C880	60	26	383	0	30/50
C1908	33	25	880	0	9/50
C2670	233	140	1 193	0	29/100
S27	4	1	10	3	4/5
S344	9	11	160	15	10/40
S386	7	7	149	6	19/50
S510	19	7	211	6	29/50
S832	18	19	287	5	19/50
B01	2	2	39	5	4/10
B04	11	8	632	66	16/100
B10	11	6	174	17	11/50

Note: x/y: number of retrieved keys/number of total encrypted keys.

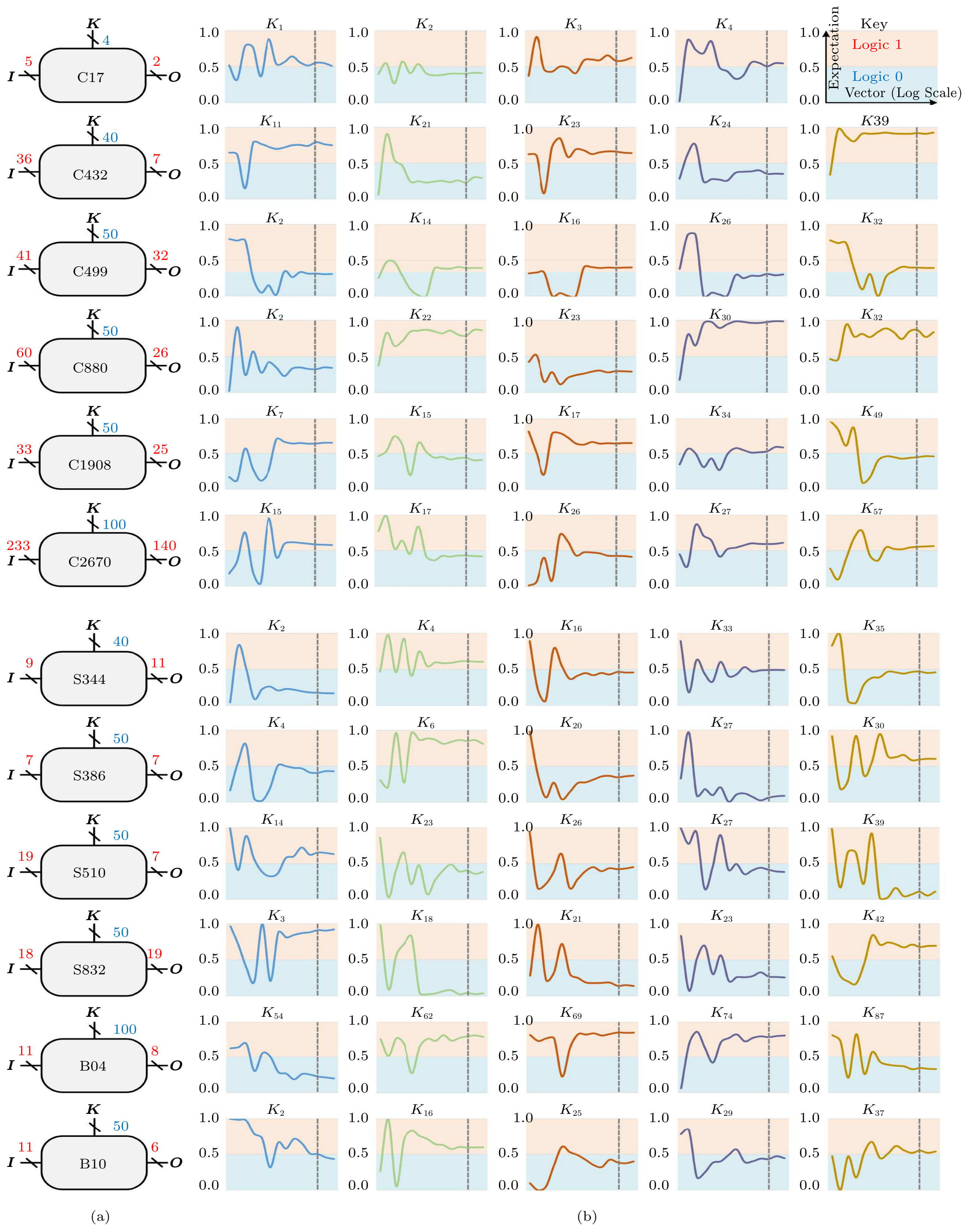


Fig.7. Implementation results on ISCAS'85/89 and ITC'99 benchmarks. (a) Input, output and key port layouts. (b) Some representative evolution curves for secret keys.

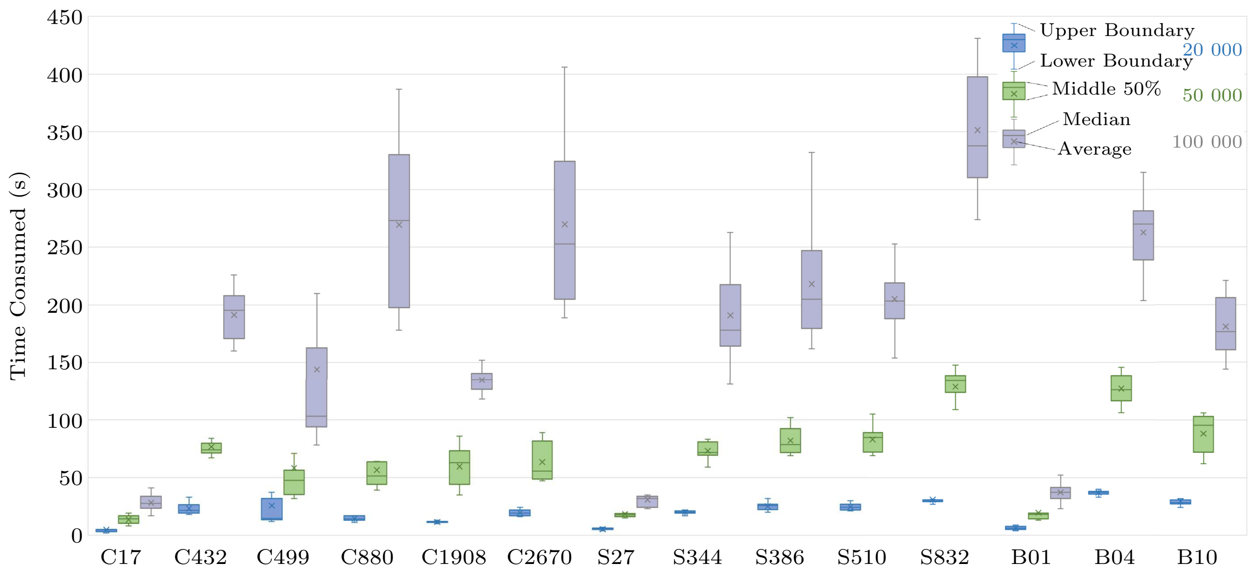


Fig.8. Time consumption for one epoch on an Intel i7-6700HQ CPU.

As a comparison with other attack schemes in logic cryptanalysis, the ML-aided attack paradigm does not ask for detailed netlist to sensitize special paths between key nodes and outputs like [19] and [20]. And it can retrieve the "easiest keys" in each epoch successively, which may also be useful to compress the decryption time if combined with tricks in [21] and [22], as they are hindered a lot if the number of keys is too large. One can utilize the limited computation resources to decipher the "easy keys" as priority, and resort to traditional methods to tackle the rest, which may be suitable for many more scenarios.

4 Discussion

It has been demonstrated in Section 3 that typical logic encryption is vulnerable to the proposed ML-aided attack paradigm. Here, we will try to concentrate on the underlying mathematical basis that supports the attack and move on to explore some potential anti-attack countermeasures.

4.1 Underlying Mathematical Comprehension

The proposed attack paradigm essentially hinges on the mathematical correlation between output $\boldsymbol{O}_{\rm en}$ and key $\boldsymbol{K}_{\rm en}$ to build a fitting function $r(\cdot)$ to deduce the proper keys $\boldsymbol{K}_{\rm right}$. Its foundation even lies on the basic logic gates. Specifically, the relevance between two logic nodes can be evaluated by the Pearson correlation coefficient as (8):

$$\rho(A,O)=\frac{cov(A,O)}{\sqrt{var(A)\times var(O)}},\rho(A,O)\in[-1,1],\quad(8)$$

where $cov(\cdot,\cdot)$ is covariance, $var(\cdot)$ represents variance and $\rho(\cdot,\cdot)$ describes the correlation extend.

Considering an AND logic gate in Fig.9, when applying random vectors $(1-p_1,\ p_1)$ on input A (the probability of being logic 1 is p_1) and $(1-p_2,\ p_2)$ on B, $\rho(A,O)$ and $\rho(B,O)$ can be quantified. Fig.9(a) illustrates when p_2 is closer to 1, input A and output O are much more positively relevant. Similarly, for OR gate in Fig.9(b), when p_2 is closer to 0, output O will be dependent on input A.

Correlation analysis helps us understand the logic dependency on digital systems. Fig.10 shows the Pearson correlation coefficient on the C17 benchmark. As anticipated, N_7 has no relationship with N_{10}, N_{11}, N_{16} and N_{22} that belong to another logic cone, while sensitizing its influence on N_{19} and N_{23}. The underlying mathematical principle for keys in logic encryption remains the same. Any modification of the original circuit will be propagated to certain nodes as correlation somehow. By tracking the correlation traces, the machine learning aided attack finally retrieves the keys and breaks the encryption.

As a supplementary experiment, a similar attack of linear regression is included in Fig.11. The experimental results confirm that any mathematical methods, not restricted to machine learning, can also be utilized to model the fitting function $g(\cdot)$, but they may not be ef-

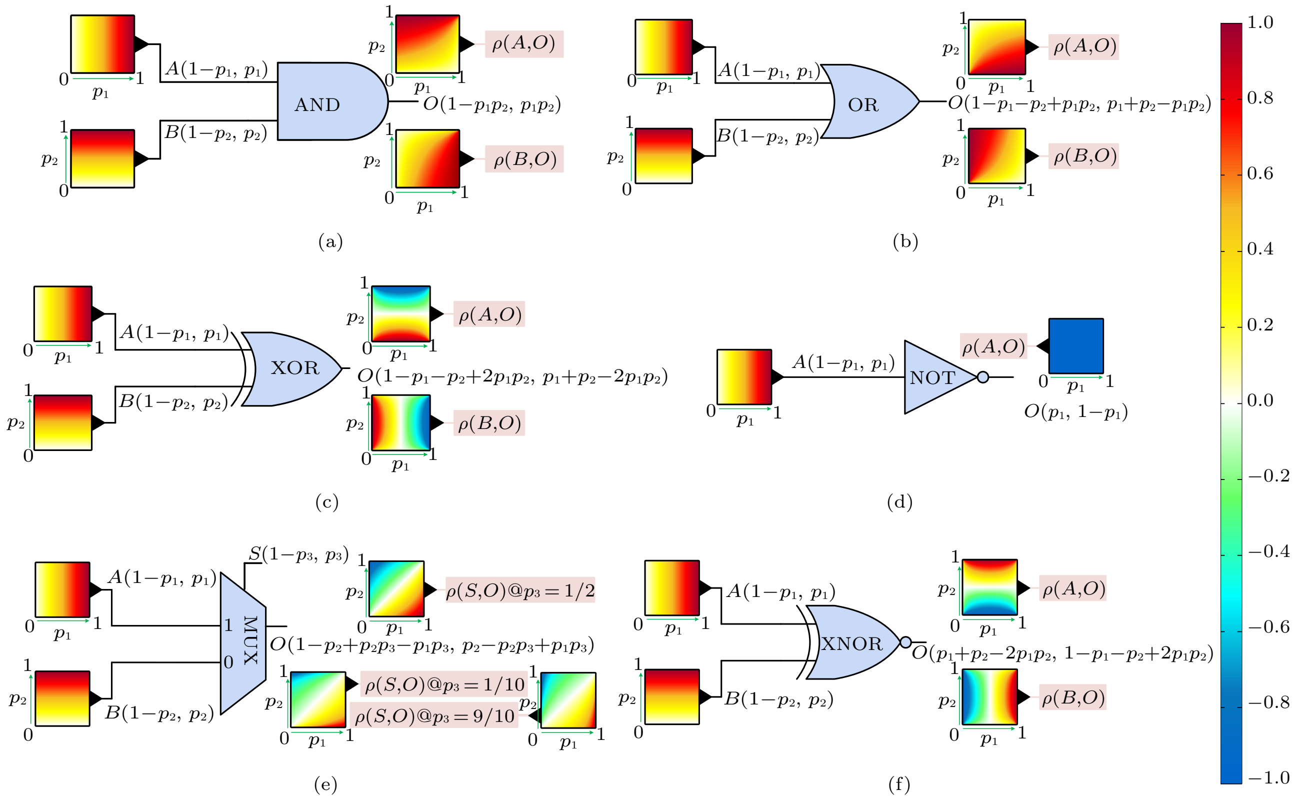


Fig.9. Pearson correlation coefficient between the output and inputs for basic logic gates. (a) AND gate. (b) OR gate. (c) XOR gate. (d) NOT gate. (e) MUX. (f) XNOR gate.

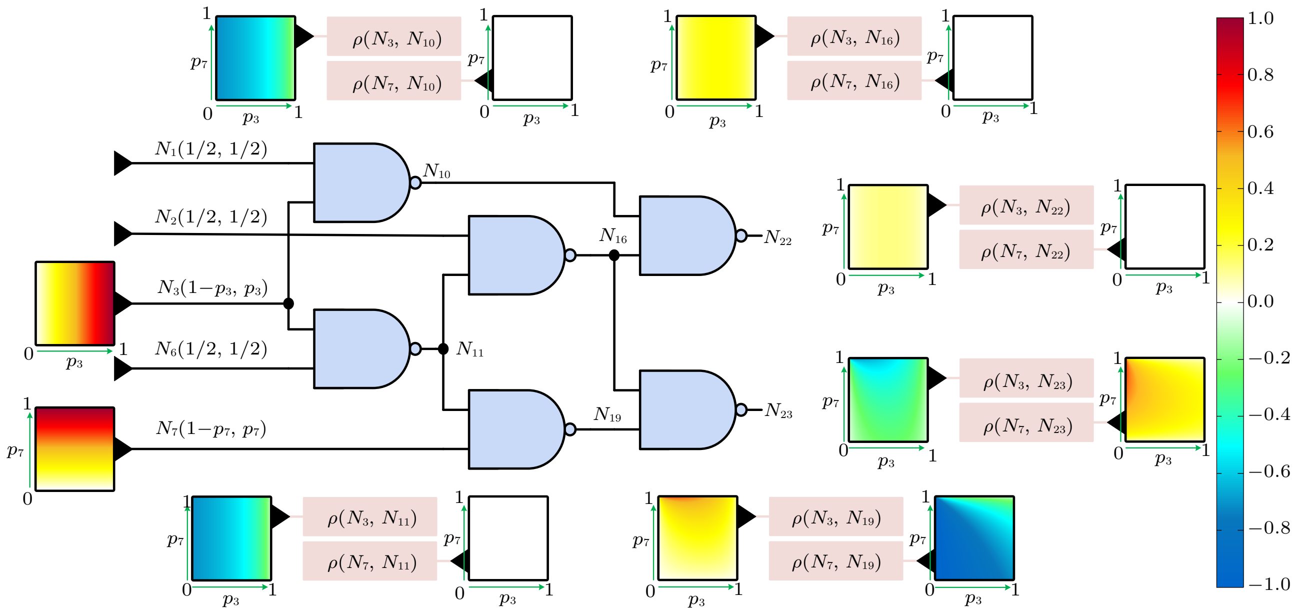


Fig.10. Pearson correlation coefficient between nodes N_3, N_7 and other internal nodes on C17 benchmark.

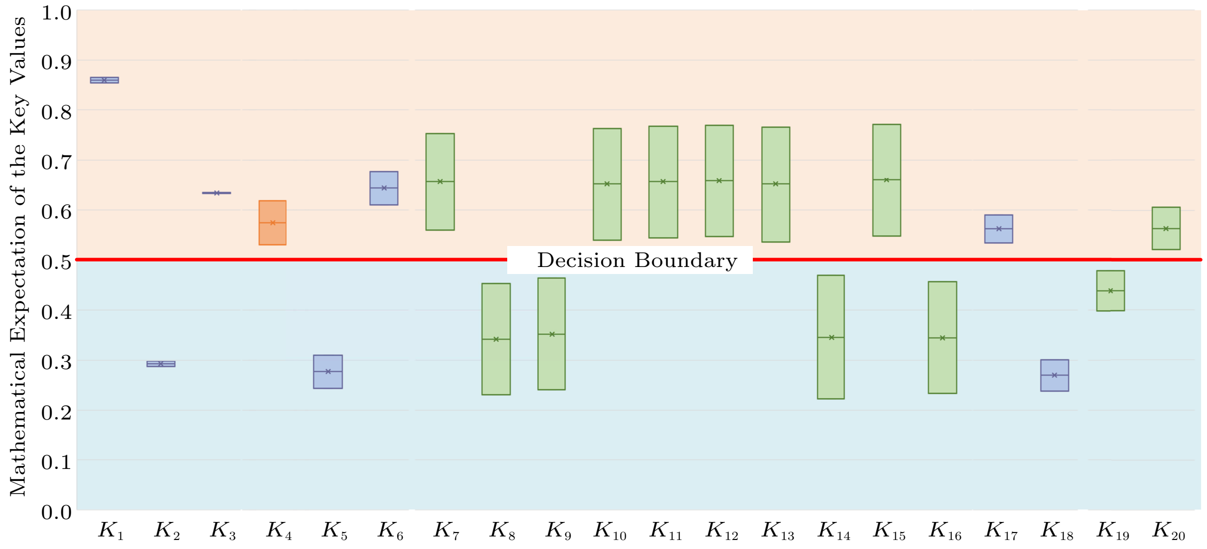


Fig.11. Linear regression fitting: boxes show the 95% confidence interval of the keys (blue: first epoch; green: second epoch; orange: third epoch).

ficient enough to extract the high-dimensional features, compared with the neural network.

4.2 Enhancing Logic Encryption

In fact, typical logic encryption does not conceal the encryption track but even exposes it to an experienced attacker. Some previous camouflage tricks such as the current domain signature attenuation [7] and the machine learning assisted countermeasure [8] in SCA-resistant designs may not be appropriate for logic attack-resistant design as the logic outputs cannot be reasonably manipulated as the power trace. This is because a good encryption also needs to ensure its obscureness, i.e., a wrong key should result in a wrong output for all input patterns [29]. However, when deliberately eliminating the correlation between keys and outputs, an applied wrong key may affect only a few output bits, which shall deteriorate the encryption efficacy.

Therefore, as our consideration, in order to mitigate the potential risks, a promising encryption algorithm can refer to the attempts of, for instance, adding a key generation unit [30], physical unclonable function (PUF) [31] to separate the outputs with the intermediate keys rather than the real key ports, and trying to conceal the internal details. Nevertheless, logically concealing the huge block also remains as a complicated issue and is therefore beyond the scope of this work.

5 Conclusions

In this paper, a machine learning aided attack scheme against logic block encryption was presented. Above all, a standard attack paradigm was defined, in which we adopted random and the highest FI inserting as the encryption mechanisms and selected a three-layer neural network as the decryption tool to exemplify how a successful attack can be executed. With the aid of the simple neural network and naive Bayes classifier, the proposed attack scheme has been implemented on the ISCAS'85/89 and ITC'99 benchmarks, succeeding in retrieving partial keys in three epochs. We also shed light on the vulnerability of the logic block based encryption and explored some preliminary countermeasures. It is notable that the underlying mathematical principle for the attack still hinges on the correlation between keys and outputs, and a possible anti-attack scheme could explore the possibility of maintaining logic obscureness and correlation confusion at the same time.

References

[1] Bhunia S, Tehranipoor M. Introduction to hardware security. In *Hardware Security: A Hands-on Learning Approach* (1st edition), Bhunia S, Tehranipoor M (eds.), Morgan Kaufmann, 2019, pp.1-20. DOI: 10.1016/B978-0-12-812477-2.00006-X.

[2] Rajendran J, Sinanoglu O, Karri R. Regaining trust in VLSI design: Design-for-trust techniques. *Proceedings of the IEEE*, 2014, 102(8): 1266-1282. DOI: 10.1109/JPROC.2014.2332154.

[3] Hospodar G, Gierlichs B, Mulder E D, Verbauwhede I, Vandewalle J. Machine learning in side-channel analysis: A first study. *Journal of Cryptographic Engineering*, 2011, 1(4): Article No. 293. DOI: 10.1007/s13389-011-0023-x.

[4] Gilmore R, Hanley N, O'Neill M. Neural network based attack on a masked implementation of AES. In *Proc. the 2015 IEEE International Symposium on Hardware Oriented Security and Trust*, May 2015, pp.106-111. DOI: 10.1109/HST.2015.7140247.

[5] Maghrebi H, Portigliatti T, Prouff E. Breaking cryptographic implementations using deep learning techniques. In *Proc. the 2016 International Conference on Security, Privacy, and Applied Cryptography Engineering*, December 2016, pp.3-26. DOI: 10.1007/978-3-319-49445-6_1.

[6] Das D, Golder A, Danial J, Ghosh S, Raychowdhury A, Sen S. X-DeepSCA: Cross-device deep learning side channel attack. In *Proc. the 56th ACM/IEEE Design Automation Conference*, June 2019, Article No. 134. DOI: 10.1145/3316781.3317934.

[7] Das D, Danial J, Golder A, Ghosh S, Wdhury A R, Sen S. Deep learning side-channel attack resilient AES-256 using current domain signature attenuation in 65nm CMOS. In *Proc. the 2020 IEEE Custom Integrated Circuits Conference*, March 2020. DOI: 10.1109/CICC48029.2020.9075889.

[8] Shan W W, Zhang S, Xu J M, Lu M Y, Shi L X, Yang J. Machine learning assisted side-channel-attack countermeasure and its application on a 28-nm AES circuit. *IEEE Journal of Solid-State Circuits*, 2020, 55(3): 794-804. DOI: 10.1109/JSSC.2019.2953855.

[9] Roy J A, Koushanfar F, Markov I L. EPIC: Ending piracy of integrated circuits. In *Proc. the 2008 Design, Automation and Test in Europe*, March 2008, pp.1069-1074. DOI: 10.1109/DATE.2008.4484823.

[10] Rajendran J, Zhang H, Zhang C, Rose G S, Pino Y, Sinanoglu O, Karri R. Fault analysis-based logic encryption. *IEEE Transactions on Computers*, 2015, 64(2): 410-424. DOI: 10.1109/TC.2013.193.

[11] Pritika K, Vinodhini M. Logic encryption of combinational circuits. In *Proc. the 3rd International Conference on Electronics, Materials Engineering & Nano-Technology*, August 2019. DOI: 10.1109/IEMENTech48150.2019.8981198.

[12] Kiryakina M A, Kuzmicheva S A, Ivanov M A. Encrypted PRNG by logic encryption. In *Proc. the 2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering*, January 2020, pp.356-358. DOI: 10.1109/EIConRus49466.2020.9038921.

[13] Karmakar R, Chatopadhyay S, Kapur R. Encrypt flip-flop: A novel logic encryption technique for sequential circuits. arXiv:1801.04961, 2018. https://arxiv.org/abs/1801.04961, January 2021.

[14] Šišejković D, Merchant F, Leupers R, Ascheid G, Kegreiss S. Inter-Lock: Logic encryption for processor cores beyond module boundaries. In *Proc. the 2019 IEEE European Test Symposium*, May 2019. DOI: 10.1109/ETS.2019.8791528.

[15] Karmakar R, Prasad N, Chattopadhyay S, Kapur R, Sengupta I. A new logic encryption strategy ensuring key interdependency. In *Proc. the 30th International Conference on VLSI Design and the 16th International Conference on Embedded Systems*, January 2017, pp.429-434. DOI: 10.1109/VLSID.2017.29.

[16] Juretus K, Savidis I. Reduced overhead gate level logic encryption. In *Proc. the 2016 International Great Lakes Symposium on VLSI*, May 2016, pp.15-20. DOI: 10.1145/2902961.2902972.

[17] Chen X M, Liu Q Y, Wang Y, Xu Q, Yang H Z. Low-overhead implementation of logic encryption using gate replacement techniques. In *Proc. the 18th International Symposium on Quality Electronic Design*, March 2017, pp.257-263. DOI: 10.1109/ISQED.2017.7918325.

[18] Yasin M, Mazumdar B, Ali S S, Sinanoglu O. Security analysis of logic encryption against the most effective side-channel attack: DPA. In *Proc. the 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, October 2015, pp.97-102. DOI: 10.1109/DFT.2015.7315143.

[19] Yasin M, Rajendran J, Sinanoglu O, Karri R. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016, 35(9): 1411-1424. DOI: 10.1109/TCAD.2015.2511144.

[20] Lee Y W, Touba N A. Improving logic obfuscation via logic cone analysis. In *Proc. the 16th Latin-American Test Symposium*, March 2015. DOI: 10.1109/LATW.2015.7102410.

[21] Plaza S M, Markov I L. Solving the third-shift problem in IC piracy with test-aware logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2015, 34(6): 961-971. DOI: 10.1109/TCAD.2015.2404876.

[22] Subramanyan P, Ray S, Malik S. Evaluating the security of logic encryption algorithms. In *Proc. the 2015 IEEE International Symposium on Hardware Oriented Security and Trust*, May 2015, pp.137-143. DOI: 10.1109/HST.2015.7140252.

[23] Yasin M, Mazumdar B, Rajendran J, Sinanoglu O. SAR-Lock: SAT attack resistant logic locking. In *Proc. the 2016 IEEE International Symposium on Hardware Oriented Security and Trust*, May 2016, pp.236-241. DOI: 10.1109/HST.2016.7495588.

[24] Xie Y, Srivastava A. Anti-SAT: Mitigating SAT attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019, 38(2): 199-207. DOI: 10.1109/TCAD.2018.2801220.

[25] Chen Y C. Tree-based logic encryption for resisting SAT attack. In *Proc. the 26th IEEE Asian Test Symposium*, November 2017, pp.46-51. DOI: 10.1109/ATS.2017.21.

[26] Shen Y Q, Rezaei A, Zhou H. SAT-based bit-flipping attack on logic encryptions. In *Proc. the 2018 Design, Automation & Test in Europe Conference & Exhibition*, March 2018, pp.629-632. DOI: 10.23919/DATE.2018.8342086.

[27] Shen Y Q, Li Y, Kong S Y, Rezaei A, Zhou H. SigAttack: New high-level SAT-based attack on logic encryptions. In *Proc. the 2019 Design, Automation & Test in Europe Conference & Exhibition*, March 2019, pp.940-943. DOI: 10.23919/DATE.2019.8714924.

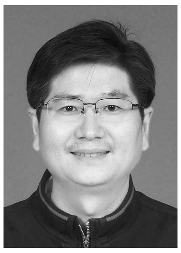
[28] Kasarabada Y, Chen S Y, Vemuri R. On SAT-based attacks on encrypted sequential logic circuits. In *Proc. the 20th International Symposium on Quality Electronic Design*, March 2019, pp.204-211. DOI: 10.1109/ISQED.2019.8697421.

[29] Rajendran J, Pino Y, Sinanoglu O, Karri R. Logic encryption: A fault analysis perspective. In *Proc. the 2012 Design, Automation & Test in Europe Conference & Exhibition*, March 2012, pp.953-958. DOI: 10.1109/DATE.2012.6176634.

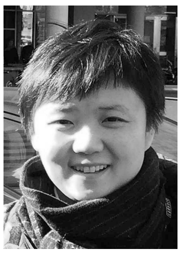
[30] Karmakar R, Chattopadhyay S, Kapur R. Enhancing security of logic encryption using embedded key generation unit. In *Proc. the 2017 International Test Conference in Asia (ITC-Asia)*, September 2017, pp.131-136. DOI: 10.1109/ITC-ASIA.2017.8097127.

[31] Mobaraki S, Amirkhani A, Atani R E. A novel PUF based logic encryption technique to prevent SAT attacks and trojan insertion. In *Proc. the 9th International Symposium on Telecommunications*, December 2018, pp.507-513. DOI: 10.1109/ISTEL.2018.8661086.

Yi Zhong received his B.S. degree in microelectronics from Peking University, Beijing, in 2018. He is currently a Ph.D. candidate in microelectronic and solid-state electronics at the Laboratory of SoC, Institute of Microelectronics, Peking University, Beijing. His current research interests include IC design on neuromorphic system and hardware security.



Jian-Hua Feng received his B.S. degree in semiconductor physics and devices from Harbin Institute of Technology, Harbin, in 1986, his M.S. degree in microelectronics from Xidian University, Xi'an, in 1995, and his Ph.D. degree in computer system architecture from Xi'an Microelectronic Technology Institute, Xi'an, in 2000. He is currently a research professor with Peking University, Beijing. His current research interests include IC design and test, hardware security and reliability prediction.



Xiao-Xin Cui received her B.S. degree in cybernation from Beihang University, Beijing, in 2002, and her Ph.D. degree in microelectronics from Peking University, Beijing, in 2007. She is currently an associate professor with Peking University, Beijing. Her current research interests include neuromorphic circuits and hardware security.

Xiao-Le Cui received his B.S. degree in communication engineering from Xidian University, Xi'an, in 1997, his M.S. degree in computer engineering from Xi'an Institute of Microelectronics, Xi'an, in 2000, and his Ph.D. degree in system engineering from Beihang University, Beijing, in 2004. He is currently a professor with the Peking University Shenzhen Graduate School, Shenzhen. His current research interests include VLSI testing, reliability and safety of electronic systems, and dependable computing.