SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

January 2023, Vol. 66 112302:1–112302:19
https://doi.org/10.1007/s11432-021-3451-1

A nonprofiled side-channel analysis based on variational lower bound related to mutual information

Chi ZHANG[1], Xiangjun LU[1], Pei CAO[1], Dawu GU[1*], Zheng GUO[2] & Sen XU[3]

[1]*School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai* 200240*, China;*
[2]*ZhiXun Crypto Testing and Evaluation Technology Co., Ltd., Shanghai* 201601*, China;*
[3]*Viewsource Information Science and Technology Co., Ltd., Shanghai* 200241*, China*

Received 8 August 2021/Revised 23 December 2021/Accepted 8 March 2022/Published online 23 December 2022

Abstract In this paper, we attempt to improve the practical performance of the nonprofiled side-channel analysis (NonSCA) with the help of neural networks. We first derive a variational lower bound related to mutual information (VLBRMI) optimized for the context of NonSCA, which possesses a set of adjustable parameters and whose maximum value linearly depends on the mutual information. Then, we propose a new NonSCA method called neural mutual information analysis (NMIA) that exploits the maximum VLBRMI as the distinguisher. We present an estimator of the maximum VLBRMI, which uses neural networks to instantiate the VLBRMI and trains the neural networks to approximate the maximum VLBRMI so that we can implement the NMIA efficiently. Finally, we evaluate the NMIA on several datasets. The experimental results show that NMIA outperforms the correlation power analysis, the mutual information analysis (MIA) based on histograms, the MIA based on kernel density estimation, and the state-of-the-art NonSCA method based on neural networks.

Keywords side-channel analysis, nonprofiled method, variational lower bound, mutual information, neural networks

Citation Zhang C, Lu X J, Cao P, et al. A nonprofiled side-channel analysis based on variational lower bound related to mutual information. Sci China Inf Sci, 2023, 66(1): 112302, https://doi.org/10.1007/s11432-021-3451-1

1 Introduction

Side-channel analysis (SCA) is a set of powerful cryptanalysis methods against cryptographic implementations. When a cryptographic device manipulates an operand or an operation related to the secret stored in the device, it unintentionally produces the side-channel traces, e.g., power consumption [1] and electromagnetic radiation [2], that contains the information about the secret; an SCA method can restore the secret based on the corresponding side-channel traces.

Nonprofiled SCA (NonSCA) is a primary type of SCA, which can directly analyze the target device without the profiling phase. A NonSCA method employs a statistical measurement known as a distinguisher to measure the degree of similarity between the actual side-channel traces and the hypothetical leakages for each possible value of the secret. Theoretically, only the correct value of the secret gives rise to the highest degree of similarity. Therefore, we can sort all possible values of the secret based on their degrees of similarity and regard the possible value with the highest degree of similarity as the most likely value of the correct secret.

The distinguisher is an important component of NonSCA [3,4]. As the first proposed NonSCA method, differential power analysis (DPA) [1] uses the difference-of-means test as the distinguisher. Despite its limited efficiency, DPA demonstrated the feasibility of NonSCA and promoted subsequent research. Correlation power analysis (CPA) [5] is a milestone in the evolution of NonSCA. Pearson's correlation coefficient is the distinguisher of CPA, and its implementation is straightforward and fast. Moreover,

[*] Corresponding author (email: dwgu@sjtu.edu.cn)

CPA is optimal if the side-channel traces linearly depend on the hypothetical leakages [6]. Therefore, CPA is still a common and efficient choice.

However, CPA fails when the hypothetical leakages drift away from reality [7, 8]. To break this restriction, the researchers proposed the concept of generic distinguisher [9–11], whose goal is leading to a successful NonSCA without requiring assumptions about the hypothetical leakage model, the type of dependency, and other device-dependent factors. Mutual information analysis (MIA) [12] is one of the most popular generic NonSCA methods; it uses mutual information (MI) [13] as its distinguisher. Benefiting from the theoretical properties of MI, MIA can handle any dependency (both linear and nonlinear) between the side-channel traces and the hypothetical leakages [7, 10, 14–18]. These features make MIA superior to other generic methods in terms of adaptability to a wide range of contexts [7, 14, 18, 19]. Unfortunately, MIA faces a practical issue, i.e., estimating MI, which is a well-known challenge [20] in many fields because it is hard to precisely obtain the underlying probability density function (PDF) with finite samples. Many parametric PDF estimators, e.g., Gaussian mixture models, and nonparametric PDF estimators, e.g., histograms and kernel density estimation (KDE), have been investigated and evaluated for MIA [7, 12, 14, 17, 19, 21]. There is no simple answer regarding which estimator is the best because they behave diversely and have different costs in various scenarios [17]. Therefore, MIA is a theoretically optimal NonMIA method in the general case but has relatively poor performance in practice.

Recently, Timon [22] has proposed a NonSCA method that casts the task of measuring the degree of similarity involved in NonSCA into a multiclass classification task. This method uses neural networks to solve the multiclass classification problem, so we call it neural-network-based NonSCA (NNSCA). The central idea of NNSCA is that only the samples related to the correct value of the secret make the neural networks converge and result in high classification accuracy. NNSCA performs well in simple and complicated scenarios and has become the state-of-the-art NonSCA method based on neural networks.

In this work, we attempt to propose a new neural-network-based NonSCA method that maintains the theoretical advantages of MIA but performs better than CPA, MIA, and NNSCA in practice.

In [23], the authors introduced a new way of estimating MI based on neural networks for general purpose, called mutual information neural estimator (MINE). From the perspective of product testing, Cristiani et al. [24] have used MINE to assess the degree of side-channel leakage for a device. However, the MINE was proved to be neither a lower bound nor a higher bound on MI [25]. Hence, the MINE has no theoretical guarantee of convergence to MI and cannot be a stable distinguisher for NonSCA.

Inspired by [23–25], we turn to derive a new metric called a variational lower bound related to MI (VLBRMI), which is parameterized by a set of adjustable parameters and whose maximum value linearly depends on the MI. Then, we use the maximum VLBRMI as the distinguisher to form a new NonSCA method called neural mutual information analysis (NMIA), which is theoretically equivalent to MIA but is more efficient than MIA in terms of implementation.

The contributions of this work are summarized below:

• Starting from the definition of MI, we formally derive a tractable VLBRMI, which is optimized according to the NonSCA context.

• We propose NMIA that uses the maximum VLBRMI as the distinguisher to perform NonSCA and prove that NMIA is theoretically equivalent to MIA.

• We present an estimator of the maximum VLBRMI based on neural networks, which can be used to implement NMIA efficiently.

• We conduct several experiments to evaluate NMIA and compare NMIA with the three popular NonSCA methods, i.e., CPA, MIA, and NNSCA, in various situations. Moreover, we investigate the influence of the hyperparameters of neural networks on the performance of NMIA.

2 Preliminaries

2.1 Notations and terminologies

A calligraphic letter $\mathcal{X}$ denotes a set and $|\mathcal{X}|$ denotes the cardinality of $\mathcal{X}$. For a positive integer n, we use $\{x_i\}_{i=1}^n$ to abbreviate the set $\{x_1, x_2, \ldots, x_n\}$.

A real-valued random variable (vector) is represented by a nonbold (bold) capital letter, e.g., X ($\boldsymbol{X}$). The corresponding lowercase letters (e.g., x and $\boldsymbol{x}$) denote the realizations. By default, all vectors are column vectors unless otherwise specified. A bold capital sans-serif letter $\mathsf{\mathbf{X}} \in \mathbb{R}^{n\times m}$ denotes a real-valued matrix with n rows and m columns. The i-th row and j-th column of the matrix are represented by $\mathbf{X}[i,:]$ and $\mathbf{X}[:,j]$, respectively.

For a d-dimensional random vector $\boldsymbol{X} = [X_1, \ldots, X_d]^{\mathrm{T}}$, its realization $\boldsymbol{x} = [x_1, \ldots, x_d]^{\mathrm{T}} \in \mathbb{R}^{d\times 1}$ is a possible value taken by $\boldsymbol{X}$. All possible values compose the space of $\boldsymbol{X}$, denoted by $\mathcal{X}$. $\boldsymbol{X}$ induces a probability distribution $P_{\boldsymbol{X}}$ with a probability function $p(\boldsymbol{x})$, known as a probability mass function (PMF) if $\boldsymbol{X}$ is discrete or a PDF if $\boldsymbol{X}$ is continuous. The expectation of a function $f(\boldsymbol{x})$ under the distribution $P_{\boldsymbol{X}}$ is written as $\mathbb{E}_{P_{\boldsymbol{X}}}[f(\boldsymbol{x})]$.

The Kullback-Leibler divergence (KLD) measures the dissimilarity between two probability distributions [13]. Consider two probability distributions $P_{\boldsymbol{X}}$ and $Q_{\boldsymbol{X}}$ with probability functions $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$, respectively. The KLD between $P_{\boldsymbol{X}}$ and $Q_{\boldsymbol{X}}$ is defined as follows:

$$\mathrm{KL}[p(\boldsymbol{x})||q(\boldsymbol{x})] = \mathbb{E}_{P_{\boldsymbol{X}}}\left[\log\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]. \tag{1}$$

The KLD is nonnegative, and the equality holds when $p(\boldsymbol{x}) = q(\boldsymbol{x})$.

2.1.1 *Mixed random vectors*

Let us further consider one d_x-dimensional continuous vector $\boldsymbol{X}$ and one discrete d_y-dimensional vector $\boldsymbol{Y}$. They can form a $(d_x + d_y)$-dimensional mixed random vector $[\boldsymbol{X}^{\mathrm{T}}, \boldsymbol{Y}^{\mathrm{T}}]^{\mathrm{T}}$ in the joint space $\mathcal{X} \times \mathcal{Y}$ with the joint distribution $P_{\boldsymbol{XY}}$ and the joint probability function $p(\boldsymbol{x}, \boldsymbol{y})$. In addition, $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are the marginal PDF of $\boldsymbol{X}$ and the marginal PMF of $\boldsymbol{Y}$, respectively. Furthermore, we have the conditional distribution $P_{\boldsymbol{X}|\boldsymbol{Y}}$ with the conditional PDF is $p(\boldsymbol{x}|\boldsymbol{y})$ and the conditional distribution $P_{\boldsymbol{Y}|\boldsymbol{X}}$ with the conditional PMF $p(\boldsymbol{y}|\boldsymbol{x})$. We can regard $p(\boldsymbol{y}|\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{y})$ as functions of $\boldsymbol{x}$ and $\boldsymbol{y}$.

The conditional expectation of a function $f(\boldsymbol{x}, \boldsymbol{y})$ given $\boldsymbol{X}$ is defined as

$$\mathbb{E}_{P_{\boldsymbol{Y}|\boldsymbol{X}}}[f(\boldsymbol{x}, \boldsymbol{y})] = \sum_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{x}, \boldsymbol{y})p(\boldsymbol{y}|\boldsymbol{x}),$$

which is still a random variable because of the randomness of $\boldsymbol{X}$. We can take the expectation of the conditional expectation with respect to $\boldsymbol{X}$:

$$\mathbb{E}_{P_{\boldsymbol{X}}}[\mathbb{E}_{P_{\boldsymbol{Y}|\boldsymbol{X}}}[f(\boldsymbol{x}, \boldsymbol{y})]] = \int_{\boldsymbol{x}\in\mathcal{X}} p(\boldsymbol{x}) \sum_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{x}, \boldsymbol{y})p(\boldsymbol{y}|\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}. \tag{2}$$

Similarly, for the expectations conditioned on $\boldsymbol{Y}$, we have

$$\mathbb{E}_{P_{\boldsymbol{X}|\boldsymbol{Y}}}[f(\boldsymbol{x}, \boldsymbol{y})] = \int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}, \boldsymbol{y})p(\boldsymbol{x}|\boldsymbol{y})\,\mathrm{d}\boldsymbol{x},$$

and

$$\mathbb{E}_{P_{\boldsymbol{Y}}}[\mathbb{E}_{P_{\boldsymbol{X}|\boldsymbol{Y}}}[f(\boldsymbol{x}, \boldsymbol{y})]] = \sum_{\boldsymbol{y}\in\mathcal{Y}} p(\boldsymbol{y}) \int_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}, \boldsymbol{y})p(\boldsymbol{x}|\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}. \tag{3}$$

Finally, the expectation of $f(\boldsymbol{x}, \boldsymbol{y})$ under the joint distribution $P_{\boldsymbol{XY}}$ is defined as follows:

$$\mathbb{E}_{P_{\boldsymbol{XY}}}[f(\boldsymbol{x}, \boldsymbol{y})] = \sum_{\boldsymbol{y}\in\mathcal{Y}} \int_{\boldsymbol{x}\in\mathcal{X}} p(\boldsymbol{x}, \boldsymbol{y})f(\boldsymbol{x}, \boldsymbol{y})\,\mathrm{d}\boldsymbol{x}, \tag{4}$$

and we have that

$$\mathbb{E}_{P_{\boldsymbol{XY}}}[f(\boldsymbol{x}, \boldsymbol{y})] = \mathbb{E}_{P_{\boldsymbol{X}}}[\mathbb{E}_{P_{\boldsymbol{Y}|\boldsymbol{X}}}[f(\boldsymbol{x}, \boldsymbol{y})]] = \mathbb{E}_{P_{\boldsymbol{Y}}}[\mathbb{E}_{P_{\boldsymbol{X}|\boldsymbol{Y}}}[f(\boldsymbol{x}, \boldsymbol{y})]]. \tag{5}$$

2.2 Nonprofiled SCA

In the context of NonSCA, we assume that a target device executes a cryptographic algorithm with random plaintext and a secret key. The cryptographic algorithm has a sensitive variable that depends on a part of the plaintext called a public datum and a part of the secret key called a subkey.

Let R, S, and Z denote the public datum, the subkey, and the sensitive variable, respectively. The subkey space $\mathcal{S}$ is small so that we can enumerate all possible values of the subkey. Let n_{s} be the size of subkey space, i.e., $n_\mathrm{s} = |\mathcal{S}|$, and the set $\mathcal{S} = \{s_k\}_{k=1}^{n_\mathrm{s}}$, where each possible value s_k is known as a subkey candidate. In addition, let $g(\cdot)$ denote the intermediate function of R and S such that $Z = g(R, S)$.

Let $s_{k^*} \in \mathcal{S}$ ($k^* \in \{1, 2, \dots, n_\mathrm{s}\}$) denote the correct subkey stored in the device. The device calculates the sensitive variable $Z_{k^*} = g(R, s_{k^*})$ with the public datum R. During the manipulation of Z_{k^*}, the device produces corresponding side-channel traces. A side-channel trace is a group of time-series data containing a set of real numbers called points. Let d_t ($d_\mathrm{t} \in \mathbb{N}$) denote the number of points in a side-channel trace, and we can use a d_t-dimensional continuous random vector $\boldsymbol{T}$ to represent the side-channel trace.

2.2.1 *Analysis procedure*

Although we do not know s_{k^*}, we can enumerate all subkey candidates, one of which must equal s_{k^*}. For each candidate $s_k \in \mathcal{S}$, we can calculate $Z_k = g(R, s_k)$ with the known R. Next, we empirically choose a hypothetical leakage model $M(\cdot)$ that maps Z_k to a hypothetical leakage $H_k = M(Z_k)$. The leakage model $M(\cdot)$ mimics the actual relationship between $\boldsymbol{T}$ and Z_{k^*} to some extent. In this work, we consider two leakage models: identity (ID) and hamming weight (HW) [11].

If the leakage model is reasonable, the H_{k^*} calculated by s_{k^*} most resembles $\boldsymbol{T}$. Therefore, to determine which s_k is the correct s_{k^*}, we measure the degree of similarity between $\boldsymbol{T}$ and H_k for each s_k through a distinguisher $D(\cdot)$. Finally, we can rank all subkey candidates by sorting $D(\boldsymbol{T}, H_k)$ and obtain the most likely subkey candidate:

$$\hat{s} = \underset{s_k \in \mathcal{S}}{\arg\max}\, D(\boldsymbol{T}, H_k).$$

However, the above procedure is abstract, and the practical procedure is as follows:

(1) We make the device execute the cryptographic algorithm $n_\mathrm{t} \in \mathbb{N}$ times with n_t random public data $\{r_i\}_{i=1}^{n_\mathrm{t}}$ and acquire n_t actual side-channel traces $\{\boldsymbol{t}_i\}_{i=1}^{n_\mathrm{t}}$.

(2) We enumerate all subkey candidates $\{s_k\}_{k=1}^{n_\mathrm{s}}$ and compute the hypothetical leakage $h_{i,k}$ for each public datum r_i and each subkey candidate s_k, where $h_{i,k} = M \circ g(r_i, s_k)$. As a consequence, we obtain n_s datasets $\{\mathcal{D}_k\}_{k=1}^{n_\mathrm{s}}$, where $\mathcal{D}_k$ corresponds to s_k and equals $\{(\boldsymbol{t}_i, h_{i,k})\}_{i=1}^{n_\mathrm{t}}$.

(3) We use $\mathcal{D}_k$ to estimate $D(\boldsymbol{T}, H_k)$. Let $D^{(n_\mathrm{t})}(\boldsymbol{T}, H_k)$ denote the estimation of $D(\boldsymbol{T}, H_k)$ with n_t samples, and we can obtain the most likely candidate

$$\hat{s} = \underset{s_k \in \mathcal{S}}{\arg\max}\, D^{(n_\mathrm{t})}(\boldsymbol{T}, H_k).$$

The analysis is successful if $\hat{s} = s_{k^*}$.

2.2.2 *Mutual information analysis*

MIA [12] employs MI as the distinguisher to perform a NonSCA. The MI between $\boldsymbol{T}$ and each H_k is defined as follows:

$$I(\boldsymbol{T}, H_k) = \mathbb{E}_{P_{\boldsymbol{T}H_k}}\left[\log \frac{p(\boldsymbol{t}, h_k)}{p(\boldsymbol{t})p(h_k)}\right],$$

where $I(\boldsymbol{T}, H_k) \geqslant 0$ and the equality holds when $\boldsymbol{T}$ and H_k are independent [13]. The continuous random vector $\boldsymbol{T}$ and the discrete random variable H_k compose a mixed random vector $[\boldsymbol{T}^\mathrm{T}, H_k]^\mathrm{T}$ with a joint distribution $P_{\boldsymbol{T}H_k}$.

Since the forms of the probability functions are not limited, $I(\boldsymbol{T}, H_k)$ can measure any relation between $\boldsymbol{T}$ and H_k. In practice, however, we cannot directly calculate accurate MI and must use finite samples to estimate $I(\boldsymbol{T}, H_k)$. Histograms and KDE are two widely studied estimators in the SCA literature [7, 12, 14, 17, 19, 21].

2.2.3 *Encoding of hypothetical leakages*

In the previous subsections, we treated the hypothetical leakage H_k as a random variable. However, we will further encode H_k into another form in the rest of this paper. Let $\mathrm{En}(\cdot)$ be an encoding function that converts H_k to a d_h-dimensional random vector $\boldsymbol{H}_k$, i.e., $\boldsymbol{H}_k = \mathrm{En}(H_k)$. We consider the following three encoding functions.

• Vanilla. Vanilla encoding keeps H_k unchanged, i.e., $\boldsymbol{H}_k = H_k$. We list it here for consistence and convenience.

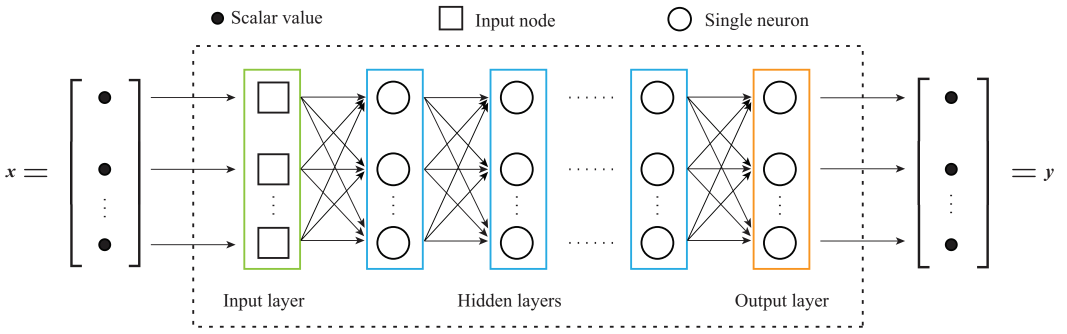

Figure 1 (Color online) An example of the MLP with the input vector $\boldsymbol{x}$ and the output vector $\boldsymbol{y}$. A solid circle represents an element (a scalar) of the vector, a small square represents an input node, and a hollow circle represents a single neuron. Multiple input nodes or neurons compose a layer represented by a rectangle.

• One-hot. One-hot encoding is a sparse way of representing data in a binary vector in which each bit represents one possible value and only a single bit can be 1 while all others are 0. For example, $\boldsymbol{h}_k = [0,0,0,1,0,0,0,0,0]^{\mathrm{T}}$ when $h_k = 3$ for the HW of an 8-bit sensitive variable.

• Binary. Binary encoding converts H_k into its binary form. For example, $\boldsymbol{h}_k = [0,0,0,0,0,0,1,1]^{\mathrm{T}}$ when $h_k = 3$ for the ID of an 8-bit sensitive variable.

Note that $\boldsymbol{H}_k$ is another form of H_k, and the probability of $\boldsymbol{H}_k$ taking the value $\boldsymbol{h}_k$ equals the probability of H_k taking the value h_k as long as $\boldsymbol{h}_k = \mathrm{En}(h_k)$. For notational simplicity, we use $\mathcal{H}_k$ to denote the space both for $\boldsymbol{H}_k$ and H_k.

2.3 Multilayer perceptron

The multilayer perceptron (MLP) is a primary type of neural network. It defines a mapping $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ with adjustable parameters $\boldsymbol{\theta}$ [26]. We employ an MLP as a universal function approximator [27, 28] to solve the optimization problem involved in this work. Hence, we next briefly introduce the concept of an MLP and the procedure for training an MLP.

2.3.1 *The architecture of an MLP*

As Figure 1 shows, an MLP is composed of many nodes arranged in stacked layers, including one input layer, several hidden layers, and one output layer.

By convention, an MLP with $l-1$ hidden layers is called an l-layer MLP. Let L_j denote the j-th layer of the MLP, where $j \in \{0,1,\ldots,l\}$, and let n_j denote the number of nodes in L_j. The L_0 and L_l are the input layer and output layer, respectively.

For any $j \in \{1,2,\ldots,l\}$, the layer L_j has two parameters: one weight matrix $\mathbf{W}_j \in \mathbb{R}^{n_{j-1}\times n_j}$ and one bias row vector $\boldsymbol{b}_j \in \mathbb{R}^{1\times n_j}$. Moreover, $\sigma_j(\cdot)$ denotes a nonlinear mapping of L_j called an activation function. Building an MLP means configuring the values of l, $(n_0,\ldots,n_l)$, and $(\sigma_1,\ldots,\sigma_l)$, which are collectively called hyperparameters.

The MLP often takes a batch of samples as the input. Let $\mathbf{X}_{\mathrm{B}} \in \mathbb{R}^{n_{\mathrm{b}}\times n_0}$ denote the batch input of the MLP that contains n_{b} samples $\{\boldsymbol{x}_i\}_{i=1}^{n_{\mathrm{b}}}$, where each sample $\boldsymbol{x}_i \in \mathbb{R}^{1\times n_0}$ is an n_0-dimensional row vector, i.e., $\mathbf{X}_{\mathrm{B}}[i,:] = \boldsymbol{x}_i$. Here, n_{b} is known as the batch size. Consequently, we can obtain the batch output of the MLP $\mathbf{Y}_{\mathrm{B}} \in \mathbb{R}^{n_{\mathrm{b}}\times n_l}$ by computing the output of each layer $\mathbf{A}_j \in \mathbb{R}^{n_{\mathrm{b}}\times n_j}$ in order

$$\mathbf{A}_j = \sigma_j(\boldsymbol{A}_{j-1}\mathbf{W}_j + \boldsymbol{b}_j), \tag{6}$$

where $\mathbf{A}_0 = \mathbf{X}_{\mathrm{B}}$ and $\mathbf{Y}_{\mathrm{B}} = \mathbf{A}_l$.

Let $\boldsymbol{\theta}$ represent all MLP parameters, i.e., all weight matrixes and all bias vectors. The MLP can be regarded as a parametric function $f(\cdot;\boldsymbol{\theta}) : \mathbb{R}^{n_{\mathrm{b}}\times n_0} \to \mathbb{R}^{n_{\mathrm{b}}\times n_l}$; for the input $\mathbf{X}_{\mathrm{B}}$, we have $\mathbf{Y}_{\mathrm{B}} = f(\mathbf{X}_{\mathrm{B}};\boldsymbol{\theta})$. According to the universal approximation theorem [27, 28], an MLP can represent a wide variety of functions when given appropriate parameters. In other words, we can build an MLP and adjust the parameters to mimic the function of interest.

2.3.2 *Training the MLP*

Consider an MLP $f(\cdot;\boldsymbol{\theta})$ and a dataset $\mathcal{D}$ containing $n \in \mathbb{N}$ samples, i.e., $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^{n}$[1)]. We can use a matrix $\mathbf{X} \in \mathbb{R}^{n\times n_0}$ to represent $\mathcal{D}$, where $\mathbf{X}[i,:] = \boldsymbol{x}_i$. A loss function $J(\mathbf{X},\boldsymbol{\theta})$ maps the parameters and all samples to a real value called loss, and it is also written as $J(\mathbf{Y})$, where $\mathbf{Y} = f(\mathbf{X};\boldsymbol{\theta})$.

We can use a mini-batch stochastic gradient descent (SGD) algorithm [26] as the optimizer to train the MLP, i.e., learning the parameters of the MLP to minimize the associated loss function. The optimizer iteratively uses a batch of samples ($\mathbf{X}_{\rm B}$) to calculate the MLP output ($\mathbf{Y}_{\rm B}$), where $\mathbf{Y}_{\rm B} = f(\mathbf{X}_{\rm B};\boldsymbol{\theta})$, and the gradients of the loss function with respect to the MLP parameters ($\nabla_{\boldsymbol{\theta}} J(\boldsymbol{Y}_{\rm B})$); then, it uses the gradients to update the parameters via the specific updating strategy ($\mathrm{opt}(\cdot)$) of the optimizer, namely, $\boldsymbol{\theta} = \mathrm{opt}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} J(\boldsymbol{Y}_{\rm B}))$. The training process finishes when the MLP converges or other conditions are met. A skeleton training procedure is shown in Algorithm 1. The ceil operator $\lceil x \rceil$ stands for rounding x to obtain the lowest integer greater than or equal to x.

Algorithm 1 A skeleton procedure for training an MLP based on a minibatch SGD optimizer

```
Input: $\mathcal{D}$: the dataset; $n$: the sample size; $f(\cdot;\boldsymbol{\theta})$: the MLP with parameters $\boldsymbol{\theta}$; $J(\cdot)$: the loss function; $n_{\rm b}$: the batch size; $n_{\rm e}$:
    the number of epochs; opt$(\cdot)$: the specific updating strategy of the minibatch SGD optimizer.
Output: $f(\cdot;\boldsymbol{\theta})$: the trained MLP.
 1: $N_{\rm B} = \lceil n/n_{\rm b} \rceil$;
 2: for $i = 0$ to $n_{\rm e} - 1$ do
 3:     Shuffle $\mathcal{D}$ randomly and divide the shuffled $\mathcal{D}$ into $N_{\rm B}$ batches;
 4:     for $j = 0$ to $N_{\rm B} - 1$ do
 5:         $\mathbf{X}_{\rm B} \leftarrow$ draw the $j$-th batch;
 6:         $\mathbf{Y}_{\rm B} = f(\mathbf{X}_{\rm B};\boldsymbol{\theta})$;
 7:         $\mathbf{G}_{\rm B} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{Y}_{\rm B})$;
 8:         $\boldsymbol{\theta} = \mathrm{opt}(\boldsymbol{\theta}, \mathbf{G}_{\rm B})$;
 9:     end for
10: end for
```

In the field of neural networks, a forward pass refers to the procedure of calculating $J(\boldsymbol{Y}_{\rm B})$ from $\boldsymbol{X}_{\rm B}$, and a backward pass refers to the procedure of calculating the gradients $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{Y}_{\rm B})$ from $J(\boldsymbol{Y}_{\rm B})$. Training an MLP requires that the computations involved in the forward and backward passes are feasible. In other words, the loss function and its gradients with respect to the parameters should be tractable for the batch samples.

3 Methodology

3.1 Variational lower bounds related to MI

Let us consider the MI between $\boldsymbol{T}$ and $\boldsymbol{H}_k$ for the subkey s_k

$$I(\boldsymbol{T},\boldsymbol{H}_k) = \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}\left[\log \frac{p(\boldsymbol{t},\boldsymbol{h}_k)}{p(\boldsymbol{t})p(\boldsymbol{h}_k)}\right]. \tag{7}$$

Given the conditional PMF $p(\boldsymbol{h}_k|\boldsymbol{t})$, because $p(\boldsymbol{t},\boldsymbol{h}_k) = p(\boldsymbol{h}_k|\boldsymbol{t})p(\boldsymbol{t})$, we can rewrite (7) as follows:

$$I(\boldsymbol{T},\boldsymbol{H}_k) = \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}\left[\log \frac{p(\boldsymbol{h}_k|\boldsymbol{t})}{p(\boldsymbol{h}_k)}\right]. \tag{8}$$

However, the conditional PMF $p(\boldsymbol{h}_k|\boldsymbol{t})$ is unknown in practice, and we cannot directly calculate $I(\boldsymbol{T},\boldsymbol{H}_k)$.

Inspired by [25], we import a parametric function $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ with a set of parameters $\boldsymbol{\theta}_k$ as an auxiliary term. Given a fixed $\boldsymbol{\theta}_k$, there is a mapping $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) : \mathcal{T}\times\mathcal{H}_k \rightarrow \mathbb{R}$. We assume that $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) \in (0,1]$ for all $\boldsymbol{t} \in \mathcal{T}$ and all $\boldsymbol{h}_k \in \mathcal{H}_k$, and the summation $\sum_{\boldsymbol{h}_k\in\mathcal{H}_k} q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ equals 1 for any fixed $\boldsymbol{t} \in \mathcal{T}$. As a consequence, $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ acts as a valid conditional PMF corresponding to a conditional distribution $Q_{\boldsymbol{H}_k|\boldsymbol{T}}$ for a specific $\boldsymbol{t}$. We will discuss the details of $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ in Subsection 3.2; here, we only use it as a black box. We insert $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ into (8) to derive the following equations:

$$I(\boldsymbol{T},\boldsymbol{H}_k) = \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}\left[\log \frac{p(\boldsymbol{h}_k|\boldsymbol{t})q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)}{p(\boldsymbol{h}_k)q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)}\right] \tag{9a}$$

1) This work focuses on unsupervised situations, so $\mathcal{D}$ does not contain the expected MLP outputs.

$$= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)] - \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)] + \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}\left[\log \frac{p(\boldsymbol{h}_k|\boldsymbol{t})}{q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)}\right] \tag{9b}$$

$$= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)] - \mathbb{E}_{P_{\boldsymbol{H}_k}}[\mathbb{E}_{P_{\boldsymbol{T}|\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]] + \mathbb{E}_{P_{\boldsymbol{T}}}\left[\mathbb{E}_{P_{\boldsymbol{H}_k|\boldsymbol{T}}}\left[\log \frac{p(\boldsymbol{h}_k|\boldsymbol{t})}{q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)}\right]\right], \tag{9c}$$

where we split the logarithm into three terms and expand the expectations according to (5). The second term of (9c) can be expanded according to (3) as follows:

$$\begin{aligned}\mathbb{E}_{P_{\boldsymbol{H}_k}}[\mathbb{E}_{P_{\boldsymbol{T}|\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]] &= \sum_{\boldsymbol{h}_k\in\mathcal{H}_k} p(\boldsymbol{h}_k)\int_{\boldsymbol{t}\in\mathcal{T}} p(\boldsymbol{t}|\boldsymbol{h}_k)\,\log p(\boldsymbol{h}_k)\,\mathrm{d}\boldsymbol{t} \\ &= \sum_{\boldsymbol{h}_k\in\mathcal{H}_k} p(\boldsymbol{h}_k)\,\log p(\boldsymbol{h}_k)\underbrace{\int_{\boldsymbol{t}\in\mathcal{T}} p(\boldsymbol{t}|\boldsymbol{h}_k)\,\mathrm{d}\boldsymbol{t}}_{\text{equals 1}} \\ &= \mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)],\end{aligned} \tag{10}$$

For the third term of (9c), according to the definition of the KLD (1), we have

$$\mathrm{KL}[p(\boldsymbol{h}_k|\boldsymbol{t})||q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)] = \mathbb{E}_{P_{\boldsymbol{H}_k|\boldsymbol{T}}}\left[\log \frac{p(\boldsymbol{h}_k|\boldsymbol{t})}{q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)}\right].$$

Since the KLD is nonnegative for all $\boldsymbol{t}\in\mathcal{T}$, we have

$$\mathbb{E}_{P_{\boldsymbol{T}}}\left[\mathbb{E}_{P_{\boldsymbol{H}_k|\boldsymbol{T}}}\left[\log \frac{p(\boldsymbol{h}_k|\boldsymbol{t})}{q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)}\right]\right] = \mathbb{E}_{P_{\boldsymbol{T}}}[\mathrm{KL}[p(\boldsymbol{h}_k|\boldsymbol{t})||q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)]] \geqslant 0, \tag{11}$$

where the equality holds when $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) = p(\boldsymbol{h}_k|\boldsymbol{t})$ for all $\boldsymbol{t}\in\mathcal{T}$.

Substituting (10) into (9c) and subtracting (11) from (9c) obtain a lower bound:

$$I_{\mathrm{LB0}}(\boldsymbol{T},\boldsymbol{H}_k) = \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)] - \mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)] \leqslant I(\boldsymbol{T},\boldsymbol{H}_k). \tag{12}$$

The lower bound I_{LB0} reaches its maximum value $I(\boldsymbol{T},\boldsymbol{H}_k)$ when $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) = p(\boldsymbol{h}_k|\boldsymbol{t})$. Since I_{LB0} varies with $\boldsymbol{\theta}_k$ and its maximum value equals the MI, it is called a variational lower bound on MI.

Intuitively, if the two expectations in (12) are computable, we can maximize I_{LB0} via an optimization algorithm, namely, learning the appropriate values of the parameters $\boldsymbol{\theta}_k$. If the optimization problem can be solved well, the maximized I_{LB0} is an estimation of the MI. In this case, we can use the maximized I_{LB0} as the distinguisher to perform a NonSCA, which is theoretically equivalent to MIA.

Let us further consider the second term of (12), i.e., $\mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]$. Since the space of $\boldsymbol{H}_k$ and the PMF $p(\boldsymbol{h}_k)$ are known in advance, we can calculate $\mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]$ precisely. For example, for an n-bit sensitive variable Z_k, if we choose ID model, we have $p(h_k) = 1/2^n$ and $\mathcal{H}_k = \{0,\ldots,2^n-1\}$; if we choose the HW model, we have $p(h_k) = \binom{h_k}{n}$ and $\mathcal{H}_k = \{0,\ldots,n\}$. Moreover, we have that $p(\boldsymbol{h}_k) = p(h_k)$ where $\boldsymbol{h}_k = \mathrm{ED}(h_k)$, so $\mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]$ is computable.

From another view, $\mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]$ can be regarded as a constant because the space $\mathcal{H}_k$ and the PMF $p(\boldsymbol{h}_k)$ remains the same for all subkey candidates. Therefore, omitting $\mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]$ from all I_{LB0} would not affect the final rank of the subkey candidates, and we can compact $I_{\mathrm{LB0}}(\boldsymbol{T},\boldsymbol{H}_k)$ to obtain another lower bound:

$$I_{\mathrm{LB1}}(\boldsymbol{T},\boldsymbol{H}_k) = \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)] \leqslant I(\boldsymbol{T},\boldsymbol{H}_k) + \mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]. \tag{13}$$

Similarly, $I_{\mathrm{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ is tight when $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) = p(\boldsymbol{h}_k|\boldsymbol{t})$. Because the maximum value of I_{LB1} equals the sum of the MI and a constant, we call I_{LB1} a VLBRMI. The maximized $I_{\mathrm{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ is an estimation of $I(\boldsymbol{T},\boldsymbol{H}_k) + \mathbb{E}_{P_{\boldsymbol{H}_k}}[\log p(\boldsymbol{h}_k)]$.

From the perspective of NonSCA, the maximized $I_{\mathrm{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ is a valid distinguisher and is equivalent to the maximized $I_{\mathrm{LB0}}(\boldsymbol{T},\boldsymbol{H}_k)$ because the omitted term has no contribution to the rank of the subkey candidates. Therefore, the NonSCA based on the maximized $I_{\mathrm{LB1}}(\boldsymbol{T},H_k)$ is also equivalent to MIA as long as the maximization problem can be solved. Consequently, the VLBRMI I_{LB1} provides us with a bridge that transforms the task of measuring the degree of similarity between $\boldsymbol{T}$ and $\boldsymbol{H}_k$ into the task of solving the following optimization problem:

$$\boldsymbol{\theta}_k^* = \mathop{\arg\max}_{\boldsymbol{\theta}_k\in\boldsymbol{\Theta}_k} I_{\mathrm{LB1}}(\boldsymbol{T},\boldsymbol{H}_k),$$

where $\boldsymbol{\Theta}_k$ is the parameter space. Next, we show the concrete form of $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ to make $I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ computable.

3.2 Concrete form of I_{LB1}

Recall that $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ is a legal conditional PMF; this PMF must be nonnegative for any $\boldsymbol{h}_k \in \mathcal{H}_k$ conditioned on a specific $\boldsymbol{t}$, and the summation of $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ over the space $\mathcal{H}_k$ must equal one. Let us define $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ based on an energy-type function to meet the above requirements

$$q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) = \frac{1}{C(\boldsymbol{t})}\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)), \quad \text{where } C(\boldsymbol{t}) = \sum_{\boldsymbol{h}'\in\mathcal{H}_k}\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}')). \tag{14}$$

The parametric function $c_{\boldsymbol{\theta}_k}(\cdot)$ maps the sample $(\boldsymbol{t},\boldsymbol{h}_k)$ to a real number. The exponential function $\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k))$ ensures that $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k) > 0$. The term $C(\boldsymbol{t})$, called a partition function, ensures that $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ is correctly normalized. Note that $C(\boldsymbol{t})$ only depends on $\boldsymbol{t}$ and is not related to the $\boldsymbol{h}_k$, so we use $\boldsymbol{h}'$ instead of $\boldsymbol{h}_k$ in the definition of $C(\boldsymbol{t})$ in (14) for distinction. Calculating the partition function needs to exhaust all elements of $\mathcal{H}_k$, which is feasible because $\mathcal{H}_k$ is finite and $|\mathcal{H}_k|$ is small. Therefore, the partition function is computable if $c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}')$ can be calculated.

We employ an MLP to instantiate $c_{\boldsymbol{\theta}_k}$, and the MLP parameters become the $\boldsymbol{\theta}_k$. The sample $(\boldsymbol{t},\boldsymbol{h}_k)$ composes a $(d_\text{t}+d_\text{h})$-dimensional row vector $\boldsymbol{x} \in \mathbb{R}^{1\times(d_\text{t}+d_\text{h})}$, where $\boldsymbol{x} = [\boldsymbol{t}^\text{T}, \boldsymbol{h}_k^\text{T}]$. The MLP maps $\boldsymbol{x}$ to a scalar $y \in \mathbb{R}$, where $y = c_{\boldsymbol{\theta}_k}(\boldsymbol{x})$. We can build the MLP with the following hyperparameters:

- $n_0 = d_\text{t} + d_\text{h}$. The number of input nodes equals the dimensionality of $\boldsymbol{x}$.
- $n_l = 1$. The output layer has one neuron because we need a scalar output.
- $\sigma_l(\cdot)$ is the identity function. The last layer does not need the activation function because the MLP output y can be any real number.
- The other MLP hyperparameters are left to be configured according to the practical scenario under study.

The MLP output y can be used to calculate $q(\boldsymbol{h}_k|\boldsymbol{t};\boldsymbol{\theta}_k)$ according to (14). In the rest of this paper, $c_{\boldsymbol{\theta}_k}(\boldsymbol{x})$ is equivalent to $c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)$ and is a computable function.

Substituting (14) into (13) yields the concrete $I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$:

$$\begin{aligned}
I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k) &= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}\left[\log\frac{\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k))}{C(\boldsymbol{t})}\right] && \text{(15a)}\\
&= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log(\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)))] - \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\log C(\boldsymbol{t})] && \text{(15b)}\\
&= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)] - \mathbb{E}_{P_{\boldsymbol{T}}}[\mathbb{E}_{P_{\boldsymbol{H}_k|\boldsymbol{T}}}[\log C(\boldsymbol{t})]] && \text{(15c)}\\
&= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)] - \int_{\boldsymbol{t}\in\mathcal{T}} p(\boldsymbol{t})\sum_{\boldsymbol{h}_k\in\mathcal{H}_k}\log C(\boldsymbol{t})p(\boldsymbol{h}_k|\boldsymbol{t})\,\mathrm{d}\boldsymbol{t} && \text{(15d)}\\
&= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)] - \int_{\boldsymbol{t}\in\mathcal{T}} p(\boldsymbol{t})\log C(\boldsymbol{t})\underbrace{\sum_{\boldsymbol{h}_k\in\mathcal{H}_k} p(\boldsymbol{h}_k|\boldsymbol{t})}_{\text{equals 1}}\,\mathrm{d}\boldsymbol{t} && \text{(15e)}\\
&= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)] - \mathbb{E}_{P_{\boldsymbol{T}}}[\log C(\boldsymbol{t})], && \text{(15f)}
\end{aligned}$$

where

$$\mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)] = \int_{\boldsymbol{t}\in\mathcal{T}}\sum_{\boldsymbol{h}_k\in\mathcal{H}_k} p(\boldsymbol{t},\boldsymbol{h}_k)c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)\,\mathrm{d}\boldsymbol{t}, \tag{16}$$

and

$$\mathbb{E}_{P_{\boldsymbol{T}}}[\log C(\boldsymbol{t})] = \int_{\boldsymbol{t}\in\mathcal{T}} p(\boldsymbol{t})\log C(\boldsymbol{t})\,\mathrm{d}\boldsymbol{t}. \tag{17}$$

Intuitively, we can set $-I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ as the loss function and train the MLP $c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)$ to minimize $-I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$, which is equivalent to maximizing $I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$. However, calculating $-I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ is impossible because the two expectations in (15f) contain integrals (see (16) and (17)) that cannot be directly calculated. In other words, $-I_{\text{LB1}}(\boldsymbol{T},\boldsymbol{H}_k)$ is intractable, and it cannot be a valid loss function. We further need to derive a tractable loss function to train $c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)$.

3.2.1 *Monte Carlo estimation*

For a general expectation involving an integral, i.e.,

$$\mathbb{E}_{P_X}[f(x)] = \int_{x\in\mathcal{X}} p(x)f(x)\mathrm{d}x, \tag{18}$$

we can estimate it with finite samples via the Monte Carlo (MC) method [29]. Given n independent and identically distributed (i.i.d.) samples $\{x_i\}_{i=1}^{n}$ drawn from P_X, the MC estimation of (18) is defined as follows:

$$\mathbb{E}^n_{P_X}[f(x)] = \frac{1}{n}\sum_{i=1}^{n} f(x_i). \tag{19}$$

The $\mathbb{E}^n_{P_X}[f(x)]$ is also a random variable whose value depends on the given set of n samples. By the strong law of large numbers [29], $\mathbb{E}^n_{P_X}[f(x)]$ coverges almost surely to $\mathbb{E}_{P_X}[f(x)]$ as we increase n. In addition, the MC estimation is unbiased, namely, the expectation of $\mathbb{E}^n_{P_X}[f(x)]$ equals $\mathbb{E}_{P_X}[f(x)]$.

In the context of a practical NonSCA, we have a dataset $\mathcal{D}_k = \{(\boldsymbol{t}_i, h_{i,k})\}_{i=1}^{n_\mathrm{t}}$ for each s_k, which can be seen as a dataset containing n_t i.i.d. samples drawn from the joint distribution $P_{\boldsymbol{TH}_k}$. Hence, the MC estimation of (16) can be obtained with $\mathcal{D}_k$

$$\mathbb{E}^{n_\mathrm{t}}_{P_{\boldsymbol{TH}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t},\boldsymbol{h}_k)] = \frac{1}{n_\mathrm{t}}\sum_{i=1}^{n_\mathrm{t}} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}_{i,k}). \tag{20}$$

For the term $\mathbb{E}_{P_{\boldsymbol{T}}}[\log C(\boldsymbol{t})]$, because it is the expectation with respect to the $P_{\boldsymbol{T}}$ other than the $P_{\boldsymbol{TH}_k}$, we can regard the trace set $\{\boldsymbol{t}_i\}_{i=1}^{n_\mathrm{t}}$ as n_t i.i.d. samples from the marginal distribution $P_{\boldsymbol{T}}$ and use the trace set to calculate the MC estimation of (17) as follows:

$$\mathbb{E}^{n_\mathrm{t}}_{P_{\boldsymbol{T}}}[\log C(\boldsymbol{t})] = \frac{1}{n_\mathrm{t}}\sum_{i=1}^{n_\mathrm{t}} \log C(\boldsymbol{t}_i). \tag{21}$$

Since $\log C(\boldsymbol{t}_i)$ is computable, combining (20) and (21) yields a computable MC estimation of $-I_{\mathrm{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$ with the dataset $\mathcal{D}_k$

$$-I^{n_\mathrm{t}}_{\mathrm{LB1}}(\mathcal{D}_k) = -\frac{1}{n_\mathrm{t}}\sum_{i=1}^{n_\mathrm{t}} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}_{i,k}) + \frac{1}{n_\mathrm{t}}\sum_{i=1}^{n_\mathrm{t}} \log C(\boldsymbol{t}_i). \tag{22}$$

3.2.2 *A tractable loss function*

Let a matrix $\mathbf{X}_k \in \mathbb{R}^{n_\mathrm{t}\times(d_\mathrm{t}+d_\mathrm{h})}$ represent the dataset $\mathcal{D}_k$, where $\mathbf{X}_k[i,:] = [\boldsymbol{t}_i^\mathrm{T}, \boldsymbol{h}_{i,k}^\mathrm{T}]$, and we can define a loss function

$$J_k(\mathbf{X}_k,\boldsymbol{\theta}_k) = -I^{n_\mathrm{t}}_{\mathrm{LB1}}(\mathcal{D}_k). \tag{23}$$

The gradients of $J_k(\mathbf{X}_k,\boldsymbol{\theta}_k)$ with respect to $\boldsymbol{\theta}_k$ are derived as follows:

$$\nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_k,\boldsymbol{\theta}_k) = -\frac{1}{n_\mathrm{t}}\sum_{i=1}^{n_\mathrm{t}} \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}_{k,i}) + \frac{1}{n_\mathrm{t}}\sum_{i=1}^{n_\mathrm{t}} \nabla_{\boldsymbol{\theta}_k}\log C(\boldsymbol{t}_i), \tag{24}$$

where

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_k}\log C(\boldsymbol{t}_i) &= \frac{1}{C(\boldsymbol{t}_i)}\nabla_{\boldsymbol{\theta}_k} C(\boldsymbol{t}_i)\\ &= \frac{1}{C(\boldsymbol{t}_i)}\nabla_{\boldsymbol{\theta}_k}\sum_{\boldsymbol{h}'\in\mathcal{H}_k}\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}'))\\ &= \frac{1}{C(\boldsymbol{t}_i)}\sum_{\boldsymbol{h}'\in\mathcal{H}_k}\nabla_{\boldsymbol{\theta}_k}\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}'))\\ &= \frac{1}{C(\boldsymbol{t}_i)}\sum_{\boldsymbol{h}'\in\mathcal{H}_k}\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}'))\nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i,\boldsymbol{h}').\end{aligned} \tag{25}$$

The term $\nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)$ denotes the gradients of the MLP output with respect to the MLP parameters, whose computations are feasible and efficient owing to the back propagation algorithm [30]. Since $C(\boldsymbol{t}_i)$, $\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}'))$, and $\nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}')$ are all computable, we are able to calculate $\nabla_{\boldsymbol{\theta}_k} \log C(\boldsymbol{t}_i)$. Therefore, calculating the gradients $\nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$ is feasible.

Let us pay more attention to $\nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$. This term denotes the gradients of $J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$, which is the MC estimation of $-I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$, but it is not the gradients of $-I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$. Therefore, it indeed gives us the best direction to update the parameters for decreasing $J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$, but it may not be the best direction for decreasing $-I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$. We need to prove that $\nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$ is unbiased with respect to the gradients of $-I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$ to ensure that the optimization procedure converges correctly.

According to (15), the gradients of $-I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$ are as follows:

$$-\nabla_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k) = -\nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)] + \nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{P_{\boldsymbol{T}}}[\log C(\boldsymbol{t})]. \tag{26}$$

For the first term, we have

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)] &= \nabla_{\boldsymbol{\theta}_k} \sum_{\boldsymbol{h}_k \in \mathcal{H}_k} \int_{\boldsymbol{t} \in \mathcal{T}} p(\boldsymbol{t}, \boldsymbol{h}_k) c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)\, \mathrm{d}\boldsymbol{t} \\ &= \sum_{\boldsymbol{h}_k \in \mathcal{H}_k} \int_{\boldsymbol{t} \in \mathcal{T}} p(\boldsymbol{t}, \boldsymbol{h}_k) \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)\, \mathrm{d}\boldsymbol{t} \\ &= \mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)],\end{aligned}$$

where the integral and the differentiation are interchangeable [31]. The expectation $\mathbb{E}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)]$ can be estimated by the MC method, namely,

$$\mathbb{E}^{n_{\text{t}}}_{P_{\boldsymbol{T}\boldsymbol{H}_k}}[\nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)] = \frac{1}{n_{\text{t}}} \sum_{i=1}^{n_{\text{t}}} \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}_{k,i}). \tag{27}$$

For the second term of (26), we have $\nabla_{\boldsymbol{\theta}_k} \mathbb{E}_{P_{\boldsymbol{T}}}[\log C(\boldsymbol{t})] = \mathbb{E}_{P_{\boldsymbol{T}}}[\nabla_{\boldsymbol{\theta}_k} \log C(\boldsymbol{t})]$, and its MC estimation is as follows:

$$\mathbb{E}^{n_{\text{t}}}_{P_{\boldsymbol{T}}}[\nabla_{\boldsymbol{\theta}_k} \log C(\boldsymbol{t})] = \frac{1}{n_{\text{t}}} \sum_{i=1}^{n_{\text{t}}} \nabla_{\boldsymbol{\theta}_k} \log C(\boldsymbol{t}_i). \tag{28}$$

Combining (27) and (28) yields the MC estimation of $-\nabla_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$:

$$-\nabla^{n_{\text{t}}}_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k) = -\frac{1}{n_{\text{t}}} \sum_{i=1}^{n_{\text{t}}} \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, h_{k,i}) + \frac{1}{n_{\text{t}}} \sum_{i=1}^{n_{\text{t}}} \nabla_{\boldsymbol{\theta}_k} \log C(\boldsymbol{t}_i).$$

According to (24), we have $-\nabla^{n_{\text{t}}}_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k) = \nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$. Because $-\nabla^{n_{\text{t}}}_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$ is the MC estimation of $-\nabla_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$, it is unbiased with respect to $-\nabla_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$. Therefore, $\nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$ is also unbiased with respect to $-\nabla_{\boldsymbol{\theta}_k} I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$, which ensures that training the MLP to minimize $J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$ implicitly minimizes $-I_{\text{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$.

The forward and backward passes. According to the training procedure shown in Algorithm 1, we use one batch of samples instead of the entire dataset to update the parameters in each iteration. Let $\mathbf{X}_{\text{B},k} \in \mathbb{R}^{n_{\text{b}} \times (d_{\text{t}} + d_{\text{h}})}$ be the one batch of samples obtained from $\mathbf{X}_k$, i.e., n_{b} arbitrary rows of $\mathbf{X}_k$. In the forward pass, the batch loss with respect to $\mathbf{X}_{\text{B},k}$ is calculated as follows:

$$J_k(\mathbf{X}_{\text{B},k}, \boldsymbol{\theta}_k) = -\frac{1}{n_{\text{b}}} \sum_{i=1}^{n_{\text{b}}} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}_{i,k}) + \frac{1}{n_{\text{b}}} \sum_{i=1}^{n_{\text{b}}} \log C(\boldsymbol{t}_i). \tag{29}$$

In the backward pass, the batch gradients are calculated as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_k} J_k(\mathbf{X}_{\text{B},k}, \boldsymbol{\theta}_k) &= -\frac{1}{n_{\text{b}}} \sum_{i=1}^{n_{\text{b}}} \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}_{k,i}) + \frac{1}{n_{\text{b}}} \sum_{i=1}^{n_{\text{b}}} \nabla_{\boldsymbol{\theta}_k} \log C(\boldsymbol{t}_i) \\ &= -\frac{1}{n_{\text{b}}} \sum_{i=1}^{n_{\text{b}}} \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}_{k,i}) + \frac{1}{n_{\text{b}}} \sum_{i=1}^{n_{\text{b}}} \sum_{\boldsymbol{h}' \in \mathcal{H}_k} \frac{\exp(c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}'))}{C(\boldsymbol{t}_i)} \nabla_{\boldsymbol{\theta}_k} c_{\boldsymbol{\theta}_k}(\boldsymbol{t}_i, \boldsymbol{h}').\end{aligned} \tag{30}$$

Therefore, $J_k(\mathbf{X}_k, \boldsymbol{\theta}_k)$ is tractable with respect to $\mathbf{X}_{\text{B},k}$ based on (29) and (30), and it can act as a valid loss function to train $c_{\boldsymbol{\theta}_k}(\boldsymbol{t}, \boldsymbol{h}_k)$ via the minibatch SGD optimizer.

3.2.3 *Comparison of VLBRMI and other neural-network-based MI estimators*

MINE [23] and the methods proposed in [25] are common-used and widely-studied neural-network-based MI estimators in the machine learning area. As discussed in [25], MINE is neither a lower bound nor a higher bound on MI, so it has no theoretical guarantee of convergence to MI and cannot be a stable MI estimator. Furthermore, the MI estimators proposed in [25] are all variational lower bounds on MI that are derived from the I_{LB0}. Because their goal is to approximate the accurate value of MI, they must take the second expectation of I_{LB0} into consideration. However, the second expectation of I_{LB0} is intractable; in other words, the second expectation cannot be directly calculated or represented by neural networks. Therefore, these MI estimators further derive a lower bound on the second expectation, and consequently, they obtain a lower bound on I_{LB0}. The lower bound on I_{LB0} is also a variational lower bound on MI because its maximum equals the MI. Then, the maximum of the lower bound on I_{LB0} can be approximated by training the corresponding neural networks. In this way, they can indirectly obtain the estimation of MI. As the paper [25] mentioned, these methods might suffer from the unstable training process for the neural network because of the existence of the second expectation.

On the contrary, we proved that the second expectation of I_{LB0} is a fixed value for all subkey candidates, so cancelling the second expectation out would not affect the analysis result in NonSCA context. Based on this fact, we propose the lower bound I_{LB1}, whose maximum equals the sum of the mutual information and a fixed value, so we call it a variational lower bound related to MI (not “on MI”). We can train neural networks to estimate the maximum I_{LB1} for each subkey candidate and use the estimated maximum I_{LB1} to sort all subkey candidates. In this case, the training process is stable because we do not need to deal with the second expectation of I_{LB0}. From another view, our proposed method is more efficient than the neural-network-based MI estimators because of the lack of the second expectation.

3.3 Neural mutual information analysis

This subsection combines all concepts mentioned above to propose a new NonSCA method. We call this method NMIA because it uses neural networks and is theoretically equivalent to MIA. The procedure of NMIA is summarized as follows.

(1) Constructing the datasets. We acquire n_{t} pairs of the side-channel trace and the public datum $\{(\boldsymbol{t}_i, r_i)\}_{i=1}^{n_{\mathrm{t}}}$ and enumerate n_{s} subkey candidates $\{s_k\}_{k=1}^{n_{\mathrm{s}}}$. Then, we calculate the encoded hypothetical leakages $\boldsymbol{h}_{i,k} = \mathrm{En} \circ M \circ g(r_i, s_k)$, where the intermediate function $g(\cdot)$, the leakage model $M(\cdot)$, and the encoding function $\mathrm{En}(\cdot)$ are chosen in advance. The d_{t}-dimensional $\boldsymbol{t}_i$ and the d_{h}-dimensional $\boldsymbol{h}_{i,k}$ form a $(d_{\mathrm{t}} + d_{\mathrm{h}})$-dimensional row vector $\boldsymbol{x}_{k,i} = [\boldsymbol{t}_i^{\mathrm{T}}, \boldsymbol{h}_{i,k}^{\mathrm{T}}]$. Consequently, we construct n_{s} datasets $\{\mathcal{D}_k\}_{k=1}^{n_{\mathrm{s}}}$, each of which contains n_{t} i.i.d. samples drawn with the joint distribution $P_{\boldsymbol{T}\boldsymbol{H}_k}$, i.e., $\mathcal{D}_k = \{\boldsymbol{x}_{k,i}\}_{i=1}^{n_{\mathrm{t}}}$. A matrix $\mathbf{X}_k \in \mathbb{R}^{n_{\mathrm{t}} \times (d_{\mathrm{t}}+d_{\mathrm{h}})}$ represents the $\mathcal{D}_k$, where $\mathbf{X}_k[i,:] = \boldsymbol{x}_{k,i}$.

(2) Building the MLPs. For each subkey candidate $s_k \in \mathcal{S}$, we build an MLP with parameters $\boldsymbol{\theta}_k$ to instantiate the function $c_{\boldsymbol{\theta}_k}$, where the parameters $\boldsymbol{\theta}_k$ contain all weight matrixes and bias vectors of the k-th MLP. For a fair comparison among all subkey candidates, we set the same hyperparameters for all MLPs. As a consequence, we obtain n_{s} MLPs with the same architecture. As Subsection 3.2 mentioned, we have $n_0 = d_{\mathrm{t}} + d_{\mathrm{h}}$ and $n_l = 1$, and the activation function $\sigma_l(\cdot)$ is an identity function. The other hyperparameters need to be configured empirically or by trial and error. In addition, the k-th MLP is associated with a loss function $J_k(\mathbf{X}_k, \boldsymbol{\theta}_k) = -I_{\mathrm{LB1}}^{n_{\mathrm{t}}}(\mathcal{D}_k)$.

(3) Training the MLPs. We configure the same n_{b}, n_{e} and $\mathrm{opt}(\cdot)$ for all MLPs to synchronize the training procedures of all MLPs as much as possible. Next, we train the n_{s} MLPs with the corresponding datasets $\{\mathbf{X}_k\}_{k=1}^{n_{\mathrm{s}}}$ via the training procedure shown in Algorithm 1.

(4) Ranking the subkey candidates. After training, we obtain optimized parameters $\boldsymbol{\theta}_k$ for each subkey candidate. Then, we use $\mathbf{X}_k$ to calculate the value of $I_{\mathrm{LB1}}^{n_{\mathrm{t}}}(\mathcal{D}_k)$, which acts as an estimation of the maximum $I_{\mathrm{LB1}}(\boldsymbol{T}, \boldsymbol{H}_k)$. Finally, we rank all subkey candidates based on the calculated $I_{\mathrm{LB1}}^{n_{\mathrm{t}}}(\mathcal{D}_k)$ and select the most likely subkey

$$\hat{s} = \mathop{\arg\max}_{s_k \in \mathcal{S}} I_{\mathrm{LB1}}^{n_{\mathrm{t}}}(\mathcal{D}_k).$$

Table 1 Choices of leakage models and encoding functions for the NonSCA methods

NonSCA method	Leakage model $M(\cdot)$	Encoding function $\mathrm{En}(\cdot)$
CPA	HW, ID	Vanilla
H-MIA, K-MIA	HW, ID	Vanilla
NNSCA	HW, ID	One-hot
NMIA	HW, ID	Vanilla, one-hot for HW; vanilla, one-hot, binary for ID

4 Experiments

4.1 Experimental settings

4.1.1 *Target cryptographic algorithm*

We choose AES as the target cryptographic algorithm to analyze and attempt to restore the first byte of the AES master key with the first byte of the plaintext and the side-channel traces. Consequently, the public datum R and the subkey S are both 8-bit random variables, namely, $n_{\mathrm{s}} = 256$ and $\mathcal{S} = \{0, 1, \dots, 255\}$. We use both simulated and public side-channel traces to evaluate NMIA.

4.1.2 *Baselines*

We compare NMIA with CPA, MIA, and NNSCA. MIA is further divided into histogram-based MIA (H-MIA) and KDE-based MIA (K-MIA), which use histograms and KDE to estimate the MI, respectively.

The analysis procedure of NNSCA is similar to that of NMIA. It creates n_{s} neural networks to solve the n_{s} multiclass classification problems. For a fair comparison, NNSCA is also implemented based on MLPs in our experiments. The fixed hyperparameters of the MLPs for NNSCA are as follows:

• $n_0 = d_{\mathrm{t}}$. NNSCA treats $\boldsymbol{t}$ as input, so the number of nodes in the input layer equals d_{t}.

• $n_l = |\mathcal{H}_k|$. NNSCA uses the one-hot encoding of h_k as the MLP output, so the number of neurons in the output layer equals the size of the space $\mathcal{H}_k$.

• $\sigma_l(\cdot) = \mathrm{softmax}(\cdot)$. The activation function of the output layer is the softmax function [30], which is widely used in classification problems.

In addition, NNSCA uses the cross entropy as the loss function (see [22] for more details), and the remaining hyperparameters of NNSCA are left to be tuned like those of NMIA.

4.1.3 *Implementations*

We use Python to implement all NonSCA methods; the MLPs involved in NMIA and NNSCA are implemented based on the third-party library, TensorFlow 2.4.1 [32]. The experiments are run at a workstation with graphics processing units (GPUs), namely, NVIDIA GEFORCE RTX 2080 Ti.

For simplicity, we assume that all hidden layers of the MLP have the same number of neurons, denoted by n_{mlp}, and the same type of activation function, denoted by σ_{mlp}. Hence, we have that $n_1 = n_2 = \cdots = n_{l-1} = n_{\mathrm{mlp}}$ and $\sigma_1 = \sigma_2 = \cdots = \sigma_{l-1} = \sigma_{\mathrm{mlp}}$. In addition, we choose Adam [33] as the specific minibatch SGD optimizer. The learning rate of Adam, denoted by lr, needs to be set in advance.

As mentioned above, we need to configure many hyperparameters to implement the NonSCA methods. For clarity, we describe them as follows.

• Hypothetical leakage model and encoding function. As Table 1 lists, we have different choices of leakage models and encoding functions for different NonSCA methods. In terms of leakage model, all methods choose the HW model or the ID model to calculate h_k. In terms of encoding function, CPA, H-MIA, and K-MIA use the vanilla encoding of h_k; NNSCA uses one-hot encoding of h_k (see Subsection 4.1.2); NMIA uses vanilla or one-hot for the HW model and uses vanilla, one-hot, or binary for the ID model.

• H-MIA and K-MIA. H-MIA has one hyperparameter, the bins or equivalently the bin width; K-MIA has two hyperparameters: the type of kernel and the bandwidth. The recommended values of these hyperparameters were introduced in [18], and we use them in our case.

• NNSCA and NMIA. Table 2 lists the necessary hyperparameters of the MLP for NNSCA and NMIA. Because we attempt to use as few traces as possible to perform NMIA and NNSCA, the architecture of the MLP cannot be too complicated; otherwise, the training process has difficulty converging. We consider

Table 2 Hyperparameters of the MLP for NNSCA and NMIA

Hyperparameters	Available choices	Description
l	$2, 3, 4$	The number of hidden layers ($l-1$)
n_{mlp}	$1, 2, 4, 8, 16$	The number of neurons in each hidden layer
σ_{mlp}	ReLU, LReLU, ELU	The activation function of the hidden layer
α	$0.1, 0.3, 0.5, 0.7, 0.9$	The extra parameter for LReLU and ELU
n_{b}	$32, n_{\mathrm{t}}$	The batch size
n_{e}	2000	The number of epochs
lr	0.001	The learning rate of Adam

n_{mlp} in the range $\{1, 2, 4, 8, 16\}$ and l in the range $\{2, 3, 4\}$. As for σ_{mlp}, we consider three types: rectified linear unit (ReLU) [34], leaky ReLU (LReLU) [35], and exponential linear unit (ELU) [36]. LReLU and ELU have another one hyperparameter α, and we limit α to the range $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. For the training process, let $n_{\mathrm{b}} = 32$ when $n_{\mathrm{t}} > 32$ and $n_{\mathrm{b}} = n_{\mathrm{t}}$ when $n_{\mathrm{t}} \leqslant 32$. Furthermore, we have that $n_{\mathrm{e}} = 2000$ and $\mathrm{lr} = 0.001$.

We use the grid search strategy to exhaust all possible combinations of the hyperparameters for each NonSCA method and use the guessing entropy (GE) [37] as the evaluation metric of these methods. For any NonSCA method, its GE with respect to a specific number of samples is calculated by repeating the method 100 times with the random samples.

4.2 Simulated traces

To generate n_{t} simulated traces, we first produce n_{t} random bytes as the public data $\{r_i\}_{i=1}^{n_{\mathrm{t}}}$ and one random byte as the secret subkey s_{k^*}. Then, we calculate $z_{k^*,i}$ for each r_i, where $z_{k^*,i} = \mathbf{SBox}[r_i \oplus s_{k^*}]$, and map $z_{k^*,i}$ to a simulated side-channel trace t_i via the strategies discussed below.

4.2.1 *Linear case*

A usual way to simulate side-channel traces is adding Gaussian noise to the HW of $z_{k^*,i}$:

$$t_i = \frac{\mathrm{HW}(z_{k^*,i})}{8} + \xi_i,$$

where the HW of $z_{k^*,i}$ is normalized, and the noise ξ_i is drawn from a Gaussian distribution with a mean of zero and a standard deviation γ. We choose γ in the range $\{0.01, 0.5\}$ to mimic high and low signal-to-noise ratio (SNR) environments, respectively. In this case, the simulated trace linearly depends on the HW of $z_{k^*,i}$.

High-SNR situation. We mount the NonSCA methods on $n_{\mathrm{t}} \in \{2, 3, \ldots, 10\}$ high-SNR traces. Table 3 lists the hyperparameters based on which each method obtains the best GE. Note that the fixed hyperparameters are not listed. All methods use the HW model to calculate the hypothetical leakages. The bin width of H-MIA is determined by the rule that sets the number of bins as the number of distinct hypothetical leakage values (such as 9 for HW model) [18], which we call number-of-label rule. For NNSCA and NMIA, the GEs corresponding to all possible hyperparameters of the MLP are presented in Appendix A. We find that LReLU with $\alpha = 0.9$ works better than ReLU and ELU on average.

Figure 2 shows the GEs of the NonSCA methods with respect to n_{t}. The GE of NMIA converges to 0 when $n_{\mathrm{t}} = 6$, while the GEs of CPA, K-MIA, and NNSCA converge to 0 when $n_{\mathrm{t}} = 8$, and that of H-MIA fails to become zero. In addition, NMIA also outperforms the other methods when $n_{\mathrm{t}} \leqslant 6$.

Here, we should note that if the goal of training the MLP is to estimate the maximum VLBRMI as accurately as possible, only six traces is not enough to train the MLP, whose $l = 3$ and $n_{\mathrm{mlp}} = 16$, well. However, our primary goal is to sort the subkey candidates based on the estimated maximum VLBRMIs, so we pay more attention to the ranking of the estimated maximum VLBRMIs rather than the accurate value of the maximum VLBRMIs. Even if a small number of traces may not be adequate to train all MLPs well, there still is one MLP that behaves better than the others. The six high-SNR traces can make the MLP with respect to the correct subkey converge more quickly than the others, making the corresponding estimated maximum VLBRMI the largest one.

Table 3 The hyperparameters of the NonSCA methods leading to the best GEs on the high-SNR dataset

Method	Leakage model	Other hyperparameters
CPA	HW	–
H-MIA	HW	Bin width: number-of-label rule [18]
K-MIA	HW	Kernel: Gaussian [18]
NNSCA	HW	$l = 4$, $n_{\mathrm{mlp}} = 2$, $\sigma_{\mathrm{mlp}} = \mathrm{LReLU}$, $\alpha = 0.9$
NMIA	HW	En$(\cdot)$: vanilla, $l = 3$, $n_{\mathrm{mlp}} = 16$, $\sigma_{\mathrm{mlp}} = \mathrm{LReLU}$, $\alpha = 0.9$

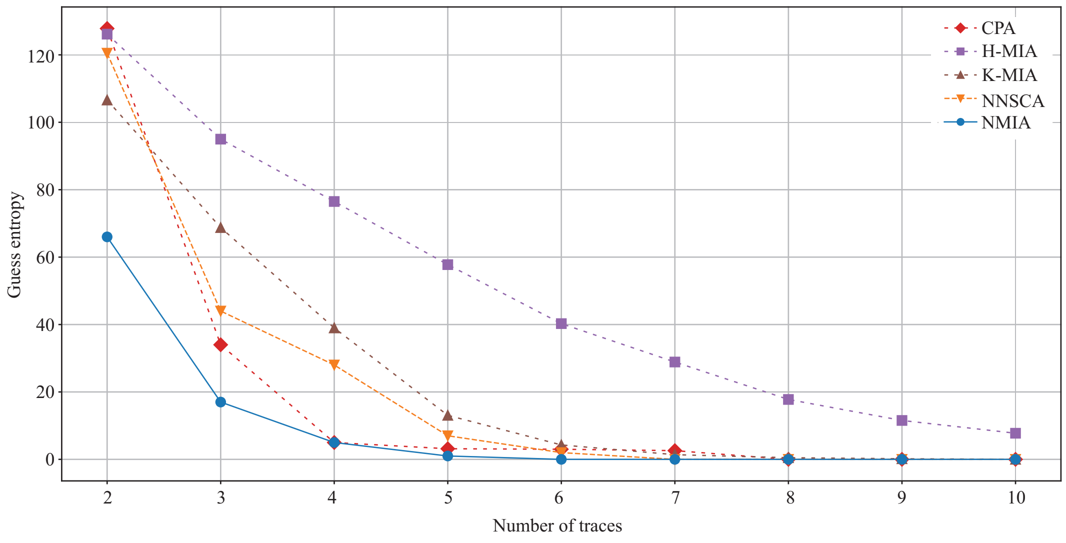

Figure 2 (Color online) The best GEs obtained with the high-SNR dataset.

Low-SNR situation. Figure 3 illustrates the best GEs with respect to $n_{\mathrm{t}} \in \{32, 64, \ldots, 320\}$ low-SNR traces, and Table 4 shows the corresponding hyperparameters. The best GE of NMIA converges to 0 when $n_{\mathrm{t}} = 128$ while that of CPA converges to 0 when $n_{\mathrm{t}} = 192$. Unfortunately, the GEs of the two MIA methods and that of NNSCA fail to become 0.

4.2.2 *Nonlinear case*

Now, we consider nonlinear side-channel traces. According to [11], let $z_j \in \{0, 1\}$ $(0 \leqslant j \leqslant 7)$ be the j-th bit of $z_{k^*,i}$, and let $\boldsymbol{u} \in \{0, 1\}^8$ be an 8-dimensional binary vector, where $\boldsymbol{u} = [u_0, u_1, \ldots, u_7]$ and $u_j \in \{0, 1\}$. We can generate the nonlinear side-channel traces as follows:

$$t_i = \sum_{\boldsymbol{u} \in \{0,1\}^8} \left(\alpha_{\boldsymbol{u}} \prod_{j=0}^{7} z_j^{u_j} \right) + \xi_i,$$

where $\alpha_{\boldsymbol{u}}$ denotes the coefficient drawn from the range $[-1, 1]$ uniformly, and ξ_i is the Gaussian noise with a mean of zero and a standard deviation $\gamma = 0.01$. This kind of simulated traces no longer linearly depends on the HW of $z_{k^*,i}$.

Figure 4 shows the best GEs with respect to $n_{\mathrm{t}} \in \{32, 48, \ldots, 128\}$ nonlinear traces, and Table 5 lists the corresponding hyperparameters. To reach the best GEs, K-MIA, and NMIA use the ID model instead of the HW model, and NMIA uses binary encoding instead of vanilla encoding.

As Figure 4 shows, the GE of NMIA converges to 0 when $n_{\mathrm{t}} = 48$; however, the GE of CPA does not converge to 0 as we expected. K-MIA, H-MIA, and NNSCA outperform CPA, but their GEs are still greater than 0 when $n_{\mathrm{t}} = 128$.

In this experiment, we generated completely nonlinear traces, and almost no linear dependency exists between the side-channel traces and the hypothetical leakage. CPA behaves poorly because the distinguisher of CPA, i.e., Pearson's correlation coefficient, can only measure the linear dependency and cannot

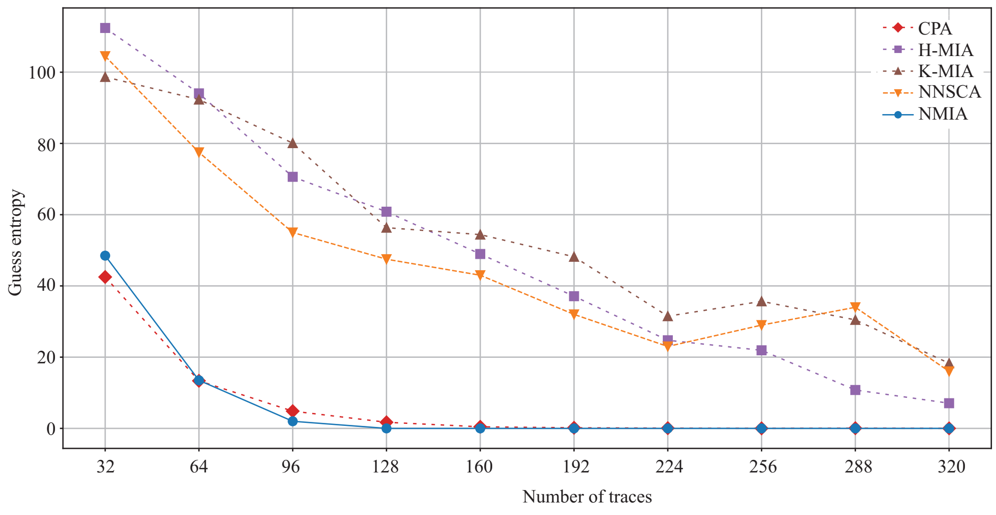

Figure 3 (Color online) The best GEs obtained with the low-SNR dataset.

Table 4 The hyperparameters of the NonSCA methods leading to the best GEs on the low-SNR dataset

Method	Leakage model	Other hyperparameters
CPA	HW	–
H-MIA	HW	Bin width: number-of-label rule [18]
K-MIA	HW	Kernel: tophat [18]
NNSCA	HW	$l = 4$, $n_{\text{mlp}} = 16$, $\sigma_{\text{mlp}} = \text{LReLU}$, $\alpha = 0.9$
NMIA	HW	En$(\cdot)$: vanilla, $l = 3$, $n_{\text{mlp}} = 16$, $\sigma_{\text{mlp}} = \text{LReLU}$, $\alpha = 0.9$

quantify the nonlinear dependency [7,8]. Therefore, for this nonlinear case, the Pearson's correlation coefficient with respect to the correct subkey candidate is low and indistinguishable from the Pearson's correlation coefficients with respect to the wrong subkey candidates. The guessing entropy of CPA with respect to the correct subkey is approximately equal to the result of random guessing. On the contrary, NMIA, MIA, and NNSCA can deal with the nonlinear traces in theory, and their GEs indeed show a downward trend.

4.3 DPA Contest V4.1

We also evaluate NMIA on the public side-channel traces provided by DPA Contest V4.1[2]. This dataset contains the side-channel traces corresponding to an implementation of AES-256 with a masking scheme called rotating SBox masking. The dataset has 100000 traces, and each trace has 435002 points.

This implementation predefines a vector **Mask** containing 16 fixed masks. Let R_i and S_i denote the i-th byte of the plaintext and the first-round key, respectively, where $i \in \{0, 1, \ldots, 15\}$. The masked output of the SBox in the first round is calculated as follows:

(1) Draw a random index from the range $\{0, 1, \ldots, 15\}$ as the offset.

(2) $X_i = R_i \oplus \textbf{Mask}[(\text{offset} + i)\%16]$.

(3) $\text{Sin}_i = X_i \oplus S_i$.

(4) $\text{MaskedSout}_i = \textbf{SBox}[\text{Sin}_i \oplus \textbf{Mask}[(\text{offset} + i)\%16]] \oplus \textbf{Mask}[(\textbf{offset} + i + 1)\%16]$.

4.3.1 *First-order analysis*

When the offset is known, S_i is the only unknown variable, in which case we can treat MaskedSout_0 as the sensitive variable to recover S_0. This strategy is called first-order NonSCA since we only target one sensitive variable. We extract the 101589-th point of the trace, which is the position most correlated with $\text{HW}(\text{MaskedSout}_0)$, to form the target trace.

2) TELECOM ParisTech SEN research group. DPA contest (4th edition), http://www.dpacontest.org/v4/.

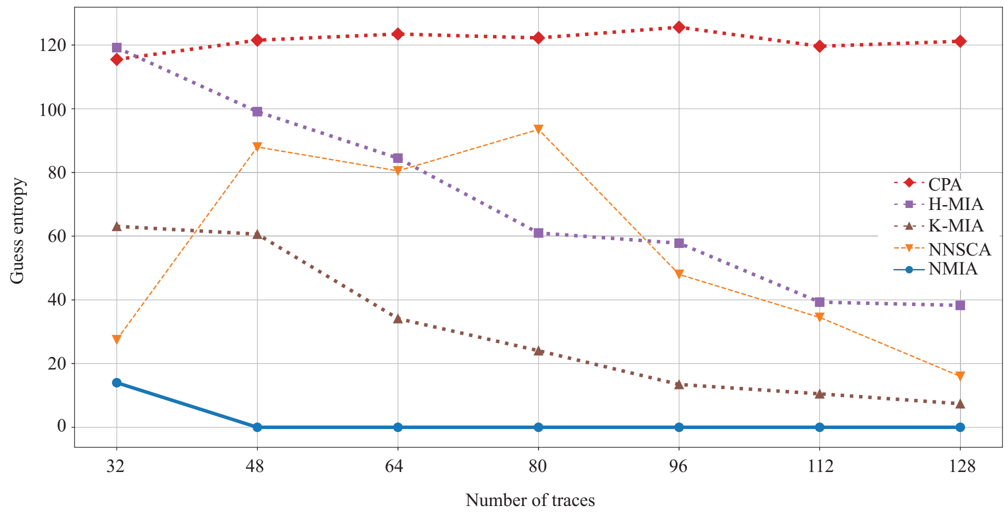

Figure 4 (Color online) The best GEs obtained with the nonlinear dataset.

Table 5 The hyperparameters of the NonSCA methods leading to the best GEs on the nonlinear dataset

Method	Leakage model	Other hyperparameters
CPA	HW	–
H-MIA	HW	Bin width: number-of-label rule [18]
K-MIA	ID	Kernel: Gaussian [18]
NNSCA	HW	$l = 3$, $n_{\text{mlp}} = 16$, $\sigma_{\text{mlp}} = \text{LReLU}$, $\alpha = 0.9$
NMIA	ID	$\text{En}(\cdot)$: binary, $l = 3$, $n_{\text{mlp}} = 16$, $\sigma_{\text{mlp}} = \text{LReLU}$, $\alpha = 0.9$

The best GEs with respect to $n_{\text{t}} \in \{2, 4, \ldots, 32\}$ are shown in Figure 5, and the hyperparameters are shown in Table 6. This result is similar to that of the high-SNR dataset. The GE of NMIA converges to 0 when $n_{\text{t}} = 12$ and is smaller than that of the other methods when $n_{\text{t}} \leqslant 12$. The GEs of CPA and NNSCA converge to 0 when $n_{\text{t}} = 18$, while those of K-MIA and H-MIA cannot converge to 0 when $n_{\text{t}} = 32$.

4.3.2 *Second-order analysis*

Finally, we consider the offset to be unknown. Let us expand MaskedSout_0 as follows:

$$\begin{aligned}\text{MaskedSout}_0 &= \textbf{SBox}[\text{Sin}_0 \oplus \textbf{Mask}[\text{offset}\%16]] \oplus \textbf{Mask}[(\text{offset}+1)\%16]\\ &= \textbf{SBox}[R_0 \oplus \textbf{Mask}[\text{offset}\%16] \oplus \textbf{Mask}[\text{offset}\%16] \oplus S_0] \oplus \textbf{Mask}[(\text{offset}+1)\%16]\\ &= \textbf{SBox}[R_0 \oplus S_0] \oplus \textbf{Mask}[(\text{offset}+1)\%16].\end{aligned}$$

Since $X_1 = R_1 \oplus \textbf{Mask}[(\text{offset}+1)\%16]$, we have that $\text{MaskedSout}_0 \oplus X_1 = \textbf{SBox}[R_0 \oplus S_0] \oplus R_1$. Because R_0 and R_1 are known, we can treat $\text{MaskedSout}_0 \oplus X_1$ as the sensitive variable to be analyzed. A NonSCA targeting the combination of two variables is known as second-order analysis [38, 39]. We locate the indices of the points most correlated with MaskedSout_0 and X_1, which are 101589 and 9564, respectively. For each trace, we combine the 101589-th and the 9564-th points, denoted by a_1 and a_2, into one point via the product rule [39]: $(a_1 - \bar{a}_1)(a_2 - \bar{a}_2)$, where $\bar{a}_1$ and $\bar{a}_2$ are the sample mean of the 101589-th and the 9564-th points over the selected n_{t} traces, respectively.

We treat the combined point as the target trace, on which we mount the NonSCA methods. The best GEs with respect to $n_{\text{t}} \in \{128, 160, \ldots, 512\}$ are shown in Figure 6, and the corresponding hyperparameters are shown in Table 7. The GE of NMIA converges to 0 when $n_{\text{t}} = 352$ while that of CPA converges to 0 when $n_{\text{t}} = 416$. The GEs of the other methods fail to converge to 0.

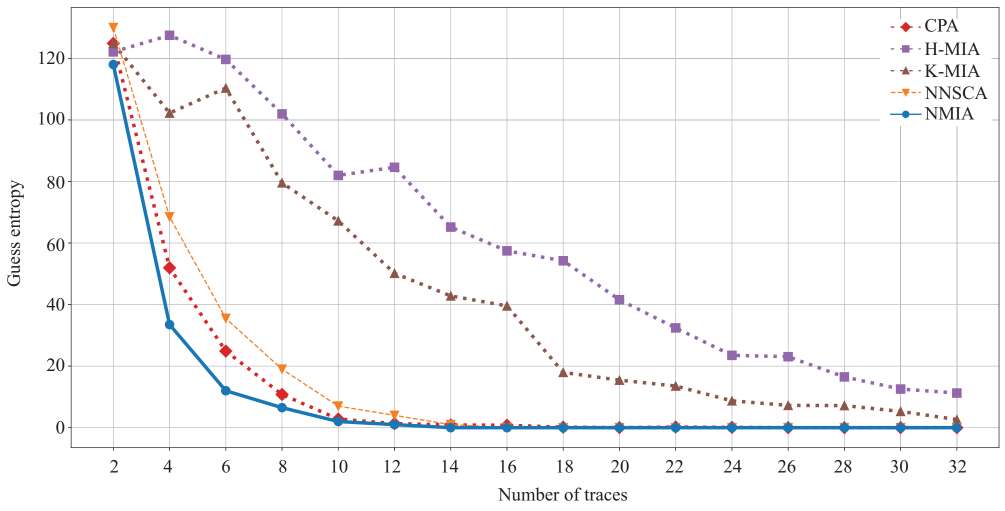

Figure 5 (Color online) The best GEs obtained with the DPA Contest V4.1 dataset in terms of the first-order analysis.

Table 6 The hyperparameters of the NonSCA methods leading to the best GEs on the DPA Contest V4.1 dataset in terms of the first-order analysis

Method	Leakage model	Other hyperparameters
CPA	HW	–
H-MIA	HW	Bin width: number-of-label rule [18]
K-MIA	HW	Kernel: Gaussian [18]
NNSCA	HW	$l = 2$, $n_{\text{mlp}} = 2$, $\sigma_{\text{mlp}} = \text{LReLU}$, $\alpha = 0.9$
NMIA	HW	En($\cdot$): vanilla, $l = 3$, $n_{\text{mlp}} = 4$, $\sigma_{\text{mlp}} = \text{LReLU}$, $\alpha = 0.9$

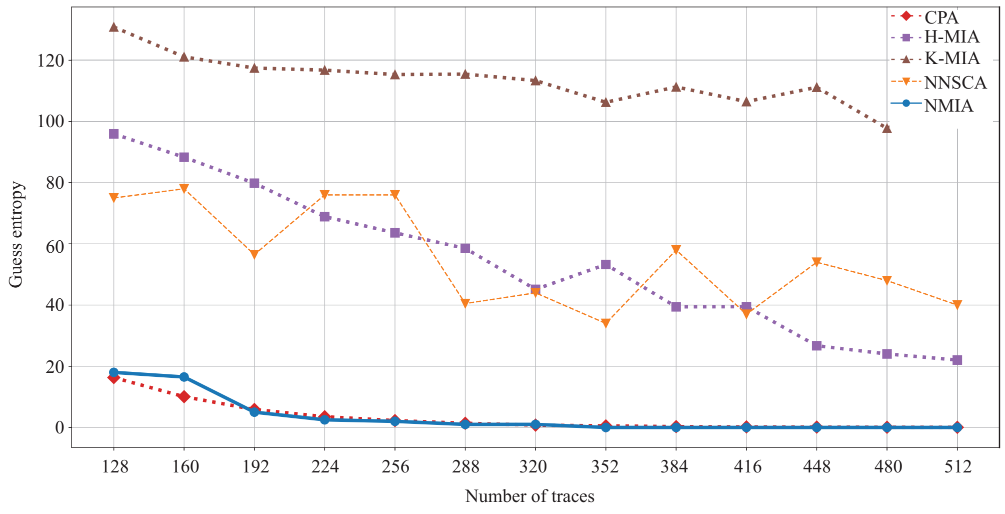

Figure 6 (Color online) The best GEs obtained with the DPA Contest V4.1 dataset in terms of the second-order analysis.

5 Conclusion

According to the experimental results, NMIA obtains its best result with the HW model and the vanilla encoding when the actual traces align with the HW model; otherwise, it obtains the best result with the ID model and the binary encoding. NMIA performs slightly better than CPA for the linear traces and much better than CPA for the nonlinear traces. In addition, NMIA outperforms H-MIA, K-MIA, and NNSCA in all experiments. Therefore, we can conclude that NMIA not only maintains the theoretical advantages of MIA but also performs well in practice like CPA.

Although NMIA and MIA are theoretically equivalent, their specific implementations make their prac-

Table 7 The hyperparameters of the NonSCA methods leading to the best GEs on the DPA Contest V4.1 dataset in terms of the second-order analysis

Method	Leakage model	Others hyperparameters
CPA	HW	–
H-MIA	HW	Bin width: number-of-label rule [18]
K-MIA	HW	Kernel: tophat [18]
NNSCA	HW	$l = 3$, $n_{\text{mlp}} = 8$, $\sigma_{\text{mlp}} =$ LReLU, $\alpha = 0.9$
NMIA	ID	En($\cdot$): binary, $l = 3$, $n_{\text{mlp}} = 8$, $\sigma_{\text{mlp}} =$ LReLU, $\alpha = 0.9$

tical performance different. Specifically, since neither the maximum VLBRMI involved in NMIA nor the MI involved in MIA can be directly calculated in practice, the practical implementation of NMIA and the practical implementation of MIA cannot reach their theoretical performance. The practical NMIA needs to employ neural networks to approximate the maximum VLBRMI; the practical MIA needs to employ histogram, kernel density estimator, or other probability density function estimators to approximate the MI. A neural network can act as a universal function approximator, and it needs fewer side-channel traces to fit the target function than the histogram (or the kernel density estimator) needs to fit the MI. Consequently, the practical NMIA performs better than the practical MIA and behaves similarly to CPA.

Unfortunately, the training time of MLP for NMIA is longer than that for NNSCA because NMIA has more calculations in one forward and backward pass (see (29) and (30)). In addition, we only considered finite hyperparameters and simple MLP architectures in this paper; however, there might be other neural networks with advanced architectures that can improve the performance of NMIA. We will investigate these issues in our following research. Moreover, we can also use the proposed VLBRMI for other SCA applications, such as using VLBRMI to assess the degree of side-channel leakage or as a penalty term to train large-scale neural networks for the profiled SCA, which are also left to be studied in the future.

Acknowledgements This work was supported by National Natural Science Foundation of China (Grant No. 62072307).

Supporting information Appendix A. The supporting information is available online at info.scichina.com and link.springer.com. The supporting materials are published as submitted, without typesetting or editing. The responsibility for scientific accuracy and content remains entirely with the authors.

References

1 Kocher P, Jaffe J, Jun B. Differential power analysis. In: Proceedings of Annual International Cryptology Conference, Santa Barbara, 1999. 388–397

2 Gandolfi K, Mourtel C, Olivier F. Electromagnetic analysis: concrete results. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Paris, 2001. 251–261

3 Standaert F X, Gierlichs B, Verbauwhede I. Partition vs. comparison side-channel distinguishers: an empirical evaluation of statistical tests for univariate side-channel attacks against two unprotected CMOS devices. In: Proceedings of International Conference on Information Security and Cryptology, Seoul, 2008. 253–267

4 Mangard S, Oswald E, Standaert F X. One for all-all for one: unifying standard differential power analysis attacks. IET Inf Secur, 2011, 5: 100–110

5 Brier E, Clavier C, Olivier F. Correlation power analysis with a leakage model. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Cambridge, 2004. 16–29

6 Heuser A, Rioul O, Guilley S. Good is not good enough. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Busan, 2014. 55–74

7 Veyrat-Charvillon N, Standaert F X. Mutual information analysis: how, when and why? In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Lausanne, 2009. 429–443

8 Renauld M, Standaert F X, Veyrat-Charvillon N, et al. A formal study of power variability issues and side-channel attacks for nanoscale devices. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, 2011. 109–128

9 Veyrat-Charvillon N, Standaert F X. Generic side-channel distinguishers: improvements and limitations. In: Proceedings of Annual International Cryptology Conference, Santa Barbara, 2011. 354–372

10 Reparaz O, Gierlichs B, Verbauwhede I. Generic DPA attacks: curse or blessing? In: Proceedings of International Workshop on Constructive Side-Channel Analysis and Secure Design, Paris, 2014. 98–111

11 Whitnall C, Oswald E, Standaert F X. The myth of generic DPA . . . and the magic of learning. In: Proceedings of Cryptographers' Track at the RSA Conference, San Francisco, 2014. 183–205

12 Gierlichs B, Batina L, Tuyls P, et al. Mutual information analysis. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Washington DC, 2008. 426–442

13 Thomas M C, Joy A T. Relative entropy and mutual information. In: Elements of Information Theory, 2nd ed. Hoboken: Wiley, 2006. 19–20

14 Prouff E, Rivain M. Theoretical and practical aspects of mutual information based side channel analysis. In: Proceedings of International Conference on Applied Cryptography and Network Security, Paris, 2009. 499–518

15 Standaert F X, Veyrat-Charvillon N, Oswald E, et al. The world is not enough: another look on second-order DPA. In: Proceedings of International Conference on the Theory and Application of Cryptology and Information Security, Sin-

gapore, 2010. 112–129

16 Gierlichs B, Batina L, Preneel B, et al. Revisiting higher-order DPA attacks. In: Proceedings of Cryptographers' Track at the RSA Conference, San Francisco, 2010. 221–234

17 Whitnall C, Oswald E. A comprehensive evaluation of mutual information analysis using a fair evaluation framework. In: Proceedings of Annual Cryptology Conference, Santa Barbara. 2011. 316–334

18 Batina L, Gierlichs B, Prouff E, et al. Mutual information analysis: a comprehensive study. J Cryptol, 2011, 24: 269–291

19 de Chérisey, Guilley S, Heuser A, et al. On the optimality and practicability of mutual information analysis in some scenarios. Cryptogr Commun, 2018, 10: 101–121

20 Paninski L. Estimation of entropy and mutual information. Neural Comput, 2003, 15: 1191–1253

21 Aumonier S. Generalized correlation power analysis. In: Proceedings of the Ecrypt Workshop Tools For Cryptanalysis, Krakow, 2007

22 Timon B. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans Cryptogr Hardware Embed Syst, 2019, 2: 107–131

23 Belghazi M I, Baratin A, Rajeshwar S, et al. Mutual information neural estimation. In: Proceedings of International Conference on Machine Learning, Stockholmsmassan, 2018. 531–540

24 Cristiani V, Lecomte M, Maurine P. Leakage assessment through neural estimation of the mutual information. In: Proceedings of International Conference on Applied Cryptography and Network Security, Rome, 2020. 144–162

25 Poole B, Ozair S, van den Oord A, et al. On variational bounds of mutual information. In: Proceedings of International Conference on Machine Learning, Long Beach, 2019. 5171–5180

26 Goodfellow I, Bengio Y, Courville A. Deep feedforward networks. In: Deep Learning. Cambridge: MIT Press, 2016. 168–224

27 Leshno M, Lin V Y, Pinkus A, et al. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Netw, 1993, 6: 861–867

28 Lu Z, Pu H, Wang F, et al. The expressive power of neural networks: a view from the width. In: Advances in Neural Information Processing Systems. San Francisco: Curran Associates, 2017

29 Weinzierl S. Introduction to Monte Carlo methods. 2000. ArXiv:hep-ph/0006269

30 LeCun Y, Bengio Y, Hinton G. Deep learning. Nature, 2015, 521: 436–444

31 Mohamed S, Rosca M, Figurnov M, et al. Monte Carlo gradient estimation in machine learning. J Mach Learn Res, 2020, 21: 1–62

32 Abadi M, Agarwal A, Barham P, et al. Tensorflow: large-scale machine learning on heterogeneous distributed systems. 2016. ArXiv:1603.04467

33 Kingma D P, Ba J. Adam: a method for stochastic optimization. 2014. ArXiv:1412.6980

34 Nair V, Hinton G E. Rectified linear units improve restricted boltzmann machines. In: Proceedings of International Conference on International Conference on Machine Learning, Haifa, 2010. 807–814

35 Maas A L, Hannun A Y, Ng A Y. Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of International Conference on Machine Learning, Atlanta, 2013

36 Clevert D A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (ELUs). 2016. ArXiv:1511.07289

37 Standaert F X, Malkin T G, Yung M. A unified framework for the analysis of side-channel key recovery attacks. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, 2009. 443–461

38 Joye M, Paillier P, Schoenmakers B. On second-order differential power analysis. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, Edinburgh, 2005. 293–308

39 Prouff E, Rivain M, Bevan R. Statistical analysis of second order differential power analysis. IEEE Trans Comput, 2009, 58: 799–811