AutoPOI

automated points of interest selection for side-channel analysis

Remmerswaal, Mick G.D.; Wu, Lichao; Tiran, Sébastien; Mentens, Nele

DOI
10.1007/s13389-023-00328-y

Publication date
2023

Document Version
Final published version

Published in
Journal of Cryptographic Engineering

Journal of Cryptographic Engineering
https://doi.org/10.1007/s13389-023-00328-y

REGULAR PAPER

AutoPOI: automated points of interest selection for side-channel analysis

Mick G. D. Remmerswaal[1] · Lichao Wu[2] · Sébastien Tiran[3] · Nele Mentens[1,4]

Received: 23 December 2022 / Accepted: 19 June 2023

Abstract
Template attacks (TAs) are one of the most powerful side-channel analysis (SCA) attacks. The success of such attacks relies on the effectiveness of the profiling model in modeling the leakage information. A crucial step for TA is to select relevant features from the measured traces, often called points of interest (POIs), to extract the leakage information. Previous research indicates that properly selecting the input leaking features could significantly increase the attack performance. However, due to the presence of SCA countermeasures and advancements in technology nodes, such features become increasingly difficult to extract with conventional approaches such as principle component analysis (PCA) and the Sum Of Squared pairwise T-difference-based method (SOST). This work proposes a framework, AutoPOI, based on proximal policy optimization to automatically find, select and scale down features. The input raw features are first grouped into small regions. The best candidates selected by the framework are further scaled down with an online-optimized dimensionality reduction neural network. Finally, the framework rewards the performance of these features with the results of TA. Based on the experimental results, the proposed framework can extract features automatically that lead to comparable state-of-the-art performance on several commonly used datasets.

Keywords Side-channel analysis · Points of interest selection · Deep reinforcement learning · Proximal policy optimization

1 Introduction

Since the pioneering work of Paul Kocher with differential power analysis (DPA) [14], many improvements have been made in side-channel analysis (SCA).

Among the newly developed attacks, the template attack (TA) is considered one of the most potent candidates [4]. TA contains two phases: a profiling phase and an attack phase. In the profiling phase, the attacker creates templates of the leakage information based on a similar or identical device under the attacker's control. Then, an attacker uses these templates to retrieve the hidden assets from leakages acquired from the device under attack. The most classical approach to building templates is forming a multivariate normal distribution for each cluster with a mean vector and a covariance matrix [19]. More advanced techniques, such as machine learning (ML) and deep learning (DL), have been recently applied in profiling SCA [2, 10, 11, 18, 22], which proves their competitiveness/superiority in breaking various devices compared with the conventional statistic-based approaches. One of the main advantages of DL-based approaches is their limited (or no) requirement for leakage preprocessing. However, such methods are criticized due to the complexity of the model and the lack of interpretability.

On the other hand, Template Attacks could be more favorable since they are based on a statistical model with limited tunable hyperparameters. Unfortunately, the effectiveness of the TA heavily relies on the preprocessing of the leakage measurements [22], more specifically, points of interest (POI) selection, which tries to capture the most relevant

✉ Lichao Wu
lichao.wu9@gmail.com

Mick G. D. Remmerswaal
mickremmerswaal@gmail.com

Sébastien Tiran
sebastien.tiran@gmail.com

Nele Mentens
nele.mentens@kuleuven.be

1 Leiden University, Leiden, The Netherlands

2 Delft University of Technology, Delft, The Netherlands

3 Delft, The Netherlands

4 KU Leuven, Leuven, Belgium

Published online: 20 July 2023

features from within the measurements and uses these to mount an attack. Indeed, POI selection is an essential step in the SCA life cycle and can dictate the performance of, arguably, one of the strongest attacks [16, 38]. However, a proper POI selection can be challenging due to environmental noise and countermeasures. Even worse, most SCA research is benchmarked on preprocessed datasets with predefined (and unrealistic) narrow time windows, which may lead to a reduced drive to research proper POI selection methods. From an attacker's perspective, *"how to find the optimal strategy for POI selection for a given dataset?"* is still an unanswered question [19].

Fortunately, finding an optimal strategy to reach a goal is one of the strengths of reinforcement learning (RL). RL has already been employed in the field of SCA and produced state-of-the-art performance when optimizing network architectures for deep learning attacks [27]. This work introduces a deep reinforcement learning-driven framework called AutoPOI for auto-matic points of interest selection. The framework generally provides a fire-and-forget method, devised as an alternative to manually selecting POIs. AutoPOI automatically selects several points of interest (POIs) and subsequently scales down the dimensions further by delivering an optimized dimensionality reduction network based on the triplet network [38]. Since this framework automatically combines the conventional POI selection method and the DL-based method, it reduces the amount of work and domain knowledge needed for the proper points of interest selection.

The contributions of this work are the following:

- This work is the first to propose the use of Deep Reinfo/032/nt Learning for POI selection through the AutoPOI framework.
- The results show that state-of-the-art attack performance can be achieved with AutoPOI, alleviating the need for predefined narrow time windows.
- With the extracted features from the AutoPOI framework, the Template Attack reaches outstanding attack performance compared to the state-of-art.

This paper is divided into several sections. Section 2 gives background information into policy-based reinforcement learning and Proximal Policy Optimization. Then, Sect. 3 provides insight into the related work in SCA, emphasizing points of interest selection. Section 4 introduces the proposed framework and explains how Proximal Policy Optimization is used for points of interest selection. Section 5 describes the experimental setup and the datasets used for benchmarking. Section 7 gives the results and discussion for each dataset. Finally, a conclusion and future work are outlined in Sect. 8.

2 Background

2.1 Notation

For the mathematical equations in this paper, numerical vectors are denoted with a bar; matrices are denoted in bold capitals, and sets are denoted with calligraphic letters. For SCA, a set of leakage traces $\mathcal{T}$ consists of traces t_i. Each trace is associated with either a plaintext d_i or a ciphertext c_i. The key space is defined as the set of all keys, $\mathcal{K}$ consisting of individual keys k_i and the correct key k^*. For reinforcement learning, we denote the learnable parameters associated with a neural network, at a certain timestep t, as θ_t.

2.2 Profiling side-channel analysis

Profiling side-channel analysis assumes an attacker has a clone device identical (or at least similar) to the device to be attacked. During the profiling phase, an attacker first measures leakage traces from the cloned device, then creates profiles based on these leakages. Finally, these profiles are applied to the device under attack; the secret information is predicted based on the profiles' output.

Using Template Attack as an example, given a key k_j and a trace $\bar{t_i}$, the conditional probability $p(k_j|\bar{t_i})$ an be calculated using Bayes Theorem, as shown in Eq. (1). An extension to multiple traces is shown in Eq. (2).

$$p(k_j|\bar{t_i}) = \frac{p(t_i|k_j)p(k_j)}{\sum_{l=1}^{K}(p(t_i|k_l)p(k_l))} \tag{1}$$

$$p(k_j|\mathbf{T}) = \frac{\left(\prod_{i=1}^{T} p(t_i|k_j)\right)p(k_j)}{\sum_{l=1}^{K}\left(\left(\prod_{i=1}^{T} p(t_i|k_l)\right)p(k_l)\right)} \tag{2}$$

Often, the intermediate data, instead of the key, is used to build the templates. An attacker controls the parameters used for the template, namely the plaintext d_i or ciphertext c_i and the key k_i. The template of each intermediate data h_{d_i,k_i} is defined according to a multivariate normal distribution with a mean vector and a covariance matrix $(\bar{m}, \mathbf{C})$ [19], such that $h_{d_i,k_i} = (\bar{m}, \mathbf{C})_{d_i,k_i}$. Therefore, the probability $p(t_i|k_l)$ can be transformed to $p(t_i|h_{d_l,k_l})$. Furthermore, the probability is then calculated using a maximum likelihood equation as depicted in Eq. (3).

$$p(t|(\bar{m}, \mathbf{C})_{d_i,k_i}) = \frac{\exp\left(-\frac{1}{2}(t-\bar{m})^T\mathbf{C}^{-1}(t-\bar{m})\right)}{\sqrt{(2\pi)^T\det(\mathbf{C})}} \tag{3}$$

The maximum likelihood for each template is calculated for each trace, which is then mapped to key guesses based on their relationship with the targeted intermediate data. The key guess with the highest maximum likelihood is k^*.

2.2.1 Points of interest selection

Points of interest (POI) selection is the method of distinguishing between relevant (to the secret information) and irrelevant or redundant features within the traces [22]. In general, there are three approaches for POI selection:

- Feature selection methods.
- Dimensionality reduction methods.
- Deep learning-based methods.

Feature selection methods create a subset of the input features and use these as the attack features. One of the most used Feature Selection methods is signal-to-noise ratio (SNR) [29] [19] and is a measurement to compare the amount of the desired signal against the unwanted amount of noise. Another technique is the Sum Of Squared T-Differences (SOST) introduced in work by Gierlichs et al. [9]. Both methods select POIs based on a top-n approach.

Dimensionality reduction methods transform the original features, using statistical analysis or mathematical operations, to a new subspace of features and use the subspace for the attack. Two methods for Dimensionality Reduction used for POI selection are principal component analysis (PCA) [12] and linear discriminant analysis (LDA) [8]. PCA and LDA find a linear combination of the variables to separate the data according to the variance. The main difference is that LDA considers the class label, whereas PCA ignores these.

Deep learning-based methods transform raw features into a new set of features. With the recent shift from statistical analysis for POI selection to Machine Learning techniques, Wu et al. [38] introduced the triplet network for feature extraction. The triplet network uses similarity learning to distinguish greater similarities between leakages of the same label while simultaneously increasing the distance of leakages with differing labels.

This work introduces a new approach to POI selection based on Deep Reinforcement Learning, the AutoPOI framework. The framework is based on the Proximal Policy Optimization algorithm and provides an automated method of finding and combining relevant POIs tailored to a dataset.

2.2.2 Hypothetical leakage models

Side-channel analysis usually consists of adopting a divide-and-conquer approach and attacking a key in chunks to recover it fully. When targeting the AES, a typical choice of length for these chunks is a byte, which corresponds to the amount of data that goes through the AES S-Boxes.

Different leakage models can be adopted in practice; their results may vary depending on the target device. The Hamming Weight (HW) leakage model classifies a byte according to its HW, while the Identify (ID) model classifies a byte according to each of its 256 possible values. A typical approach for AES is to target the S-box output of the first round or the S-box input of the last round when considering the HW or ID models. Another type of leakage model is the result of the XOR between two values. Often the Hamming Weight of this XOR is calculated and is referred to as the Hamming Distance. A typical approach for AES is to compute the XOR (or HW of the XOR, i.e., HD) between the final output and the S-box input of the last round. In this paper, all leakage models are considered benchmarks for each dataset.

2.2.3 Metrics

Guessing entropy (GE) [34] is commonly used to evaluate the effectiveness of SCA. The Guessing Entropy is based on a guessing vector $\mathbf{g} = [g_1, \ldots, g_{|K|}]$. Here, $|K|$ denotes the search space of the key, in the case of AES $|K| = 256$ for a byte. $\mathbf{g}$ contains the key candidates in decreasing order of probability: g_1 is the most likely, and $g_{|K|}$ is the least likely key candidate.

GE is the average ranking of the correct key k^* among the other key guesses, where the averaging is done over multiple attacks. The GE is calculated for each new test trace processed, resulting in a vector describing the evolution of the GE with the number of test traces processed. This is called the ranking vector.

An attack is successful if it achieves a GE of 0 (the correct key is assigned with the highest rank among all key candidates). If the target of the attack is not the full key but only one byte, it is commonly referred to as Partial Guessing Entropy. This work uses these terms interchangeably.

2.3 Reinforcement learning

Reinforcement learning (RL) [35] is the act of learning through taking actions from observations made within an environment while being given an increasing reward for correct actions taken. A graphical representation can be found in Fig. 1. An agent makes observations from the environment called states. In time step t, the agent receives state S_t from the environment and acts by following a specific policy π or transition probability T by taking action A_t. The environment takes action into account, gives a reward R_t based on a reward function $f(S_t, A_t)$, and returns a new state S_{t+1}. When the agent reaches a predetermined terminal state, the environment sends a done signal to the agent. From there on out, a new sequence of states, actions, and rewards begin.

2.4 Deep reinforcement learning

Deep reinforcement learning (DRL) is the class of RL algorithms that use Artificial Neural Networks.

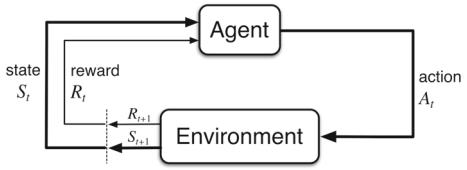


Fig. 1 A graphical representation of a generic RL environment [35]

2.4.1 Q-Networks

The work by Mnih et al. [20] describes the development of the Deep Q-Network (DQN) algorithm in their efforts to create a single algorithm capable of solving a wide range of challenging tasks. Instead of a Q-table that stores the values that map states to actions, the value is predicted by a neural network using states as inputs. Then, according to the ϵ-greedy strategy, as shown in Eq. (4), DQN samples an action.

$$a_s = \begin{cases} \underset{a \in A}{\text{argmax}}\ Q(s,a) & 1-\epsilon \\ \text{random}\, a & \epsilon \end{cases} \tag{4}$$

The method defines ϵ as the probability of taking a random action from the action space. $\epsilon = 1.0$ is akin to pure arbitrary action sampling, and $\epsilon = 0.0$ is akin to deterministically taking action with the highest Q-value.

To stabilize the network's learning and emulate the learning of past experiences in humans, Experience Replay [17] is used. The idea behind Experience Replay is for an agent to build an action model of executable actions and their consequences. This way, the agent can learn from the model what actions produce favorable outcomes without actually executing them. It is implemented by initializing a dataset D of past experiences. Then, at each timestep t, the algorithm adds experience containing the state s_t, the action a_t, the reward r_t, and the followup state s_{t+1}, to the experience dataset D. At each learning iteration, the algorithm gathers a random batch from D and uses it to update the network's weights.

Each timestep t, the network is trained by minimizing the loss function L_t of the neural network concerning the weights θ, with the following equation:

$$L_t(\theta_t) = \mathbb{E}_{s,a\sim p(\cdot)}[(y_t - Q_{\theta_t}(s,a))^2] \tag{5}$$

with

$$y_t = \begin{cases} r_t & \text{terminal } s_{t+1} \\ r_t + \gamma \max_{a'} Q_{\theta_{t-1}}(s_{t+1}, a') & \text{non-terminal } s_{t+1} \end{cases} \tag{6}$$

Here, the expectation to be minimized is the squared difference in future discounted rewards y_t, which are calculated with the previous parameters θ_{t-1}, and the current rewards $Q_{\theta_t}(s,a)$. Note that the expectation is calculated given a known state s and action a according to a probability over all actions.

2.4.2 Actor-critic architecture

In contrast with value-based RL algorithms, such as the previously mentioned DQN algorithm, policy-based methods directly approximate the optimal policy π^*. Commonly, this is achieved by using stochastic gradient ascent algorithms. Unfortunately, two issues arise when calculating policy gradients: noisy and high variance [36]. To solve these issues, Williams [37] introduced a baseline $b_t(s_t)$ to be subtracted from the policy gradient.

$$\nabla_\theta J(\theta) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - b_t(s_t)). \tag{7}$$

One common method is choosing the estimate of the value function $V(s_t)$ as the baseline. Since the baseline only depends on the state, it will not impact the gradient of the policy. The idea behind this is that the algorithm constantly checks if a specific action a_t is better or worse than the average action, given the state s_t. This is more commonly known as the advantage function:

$$A(a_t, s_t) = Q(a_t, s_t) - V(s_t). \tag{8}$$

This approach forms the basis of the actor-critic architecture, as depicted in Fig. 2, where the policy π is seen as the actor and the value function as the baseline b_t is seen as the critic [35]. After each action made by the actor, the critic evaluates the new state and concludes if the new state is better or worse. If the critic concludes a positive change, the loss will enforce that this action is taken more commonly. In contrast, if the critic concludes a negative change the loss will enforce the action to be taken less often. An advantage of actor-critic methods is that they require less computation to calculate action values. An example of this are continuous-valued actions. Any other method learning just the action values must learn an infinite set of values, one for each action. Using actor-critic methods, where the policy is explicitly stored, these computations are not needed [35].

2.4.3 Trust region policy optimization

Trust region policy optimization (TRPO) by Schulman et al. [31] introduces an algorithm for smoother policy learning. It does so by applying a Kullback–Leibler (KL) Divergence [15] on parameter updates in the policy.

Instead of directly applying the policy gradient, TRPO uses a surrogate loss function to update its parameters. The

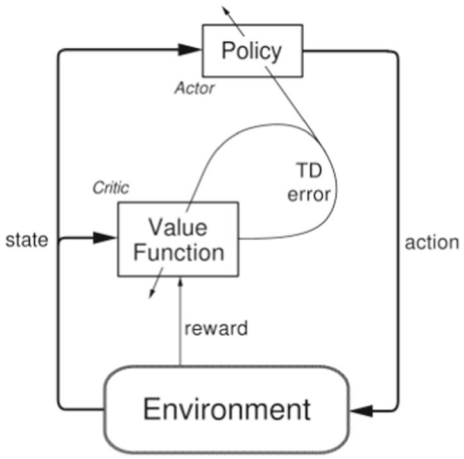


Fig. 2 An overview of the Actor-Critic Architecture [35]

surrogate loss has its roots in Importance Sampling [33], which is used to estimate the expected value of a function $f(x)$, where x follows a distribution $p(x)$. Then, instead of sampling x from p, it is sampled from another distribution q that is used to approximate p:

$$\mathbb{E}_p[f(x)] = \mathbb{E}_q\left[\frac{f(x)p(x)}{q(x)}\right]. \tag{9}$$

If $q(x)$ is sufficiently close to $p(x)$, then the estimation is sufficiently accurate.

The idea behind this is to make sure that the policy does not drift to far from its previous parameters. Constraining the KL Divergence helps the policy stay within a certain *trust region*. This helps the algorithm to emulate a smoother learning curve, reducing the chance of learning collapse. TRPO uses the loss function found in Eq. (10)

$$\max_{\pi} L(\pi) = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{\pi(a|s)}{\pi_{\text{old}}(a|s)} A^{\pi_{\text{old}}}(s,a)\right] \tag{10}$$

subject to

$$\mathbb{E}[KL(\pi, \pi_{\text{old}}) \leq \epsilon]. \tag{11}$$

Here, ϵ is a hyperparameter to be set, also note that $A^{\pi_{\text{old}}}(s,a)$ is the advantage function as depicted in Eq. (8).

2.4.4 Proximal policy optimization

Schulman et al. [32] introduced proximal policy optimization (PPO), which builds upon their earlier work in TRPO and results in an algorithm that is simpler to implement, more general, and with better computational complexity. Instead of using a constraint on the KL Divergence between the new and old policy, a clipping of the ratio was proposed:

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{12}$$

where ϵ denotes a hyperparameter to be set. The loss function is used to clip the probability ratio when it improves the objective while unrestricting it when it worsens the objective. In other words, Proximal Policy Optimization restricts policy updates that are too large, leading to smoother learning and a more negligible probability of policy collapse.

3 Related work

After the introduction of side-channel analysis (SCA) by Kocher et al. [14], the seminal work by Chari et al. [4] introduced template attacks (TAs), which would drive the research in the SCA community for many years. Although being the most potent attack from an information-theory standpoint, its assumptions can be somewhat daunting and sometimes impossible (unlimited traces). Years later, more advanced methods were devised, such as the Stochastic Models presented in the work of Schindler et al. [30], which aims to reduce the amount of traces needed for profiling significantly. Further work was done by Choudhary and Kuhn [6], where the authors introduced pooling the covariance matrices used in the profiling phase and attaining a significant speed-up of the attack. These methods remain one of the most popular methods in both academic and industry, mainly due to the strength of performance and the fact that no hyperparameter tuning is needed.

The performance of profiled SCA (more specifically, TA) heavily relies on the points of interest (POI) selection. In 2015, Lerman et al. [16] even concluded that, with proper POI selection, TA outperforms Machine Learning attacks. Over the years, several techniques have been researched to reduce the complexity of TA. One of the first works was in 2006, where Archambeau et al. [1] introduced Principal Component Analysis to create a principal subspace. The principal subspace reduced the dimension of the traces by 99.99% and resulted in being able to classify 93.3% of the traces correctly.

Picek et al. [22] explored many different POI selection methods used frequently in Side-Channel Analysis. The authors concluded that feature selection is a very important step in attacks where the data are noisy and contains various countermeasures. Next, in Perin et al.'s work [21], the authors explored different setups of POIs for the preprocessing of DL attacks. The authors concluded that a proper POI selection method could boost the attack performance dramatically.

More recently, Wu et al. [38] used a Machine Learning technique called Similarity Learning to show that with proper feature engineering, Template Attacks remain feasible and are even able to outperform current state-of-the-art Deep Learning techniques. The main drawback of the triplet network is that for each dataset, the hyperparameters have to be tuned. Rioja et al. [28] introduced an automated DL tuner based on the Estimation of Distribution Algorithms

(EDAs), which could automatically choose good-performing POIs and therefore reduce the need for human intervention.

The first instance of reinforcement learning applied in Side-Channel Analysis concerning POI selection, to the best of our knowledge, is Side-channel Analysis with Reinforcement Learning (SCARL). In this paper, Ramezanpour et al. [26] introduce an algorithm that preprocesses the data with an autoencoder and, with the help of a self-supervised Actor-Critic model, can cluster features based on the inter-cluster difference on the mean. However, their method is only tested on one specific cipher, the Ascon [7] cipher, and needs 24,000 traces to find the correct partial key.

4 AutoPOI framework

The AutoPOI Framework is a framework that automatically finds, selects and scales down Points Of Interests. It does so by selecting regions from input traces, which fed into a Neural Network to extract the most promising embeddings. The embeddings represent the selected Points Of Interest and are used to perform Template Attacks. A graphical overview of the framework is shown in Fig. 3. The framework operates in three phases,

(1) the region selection phase,
(2) the network architecture selection phase, and
(3) the embeddings extraction phase.

In phase 1, a Proximal Policy Optimization network selects promising regions from the input traces. Phase 2 consists of another Proximal Policy Optimization network that selects an optimized Neural Network architecture. This optimized network is used in phase 3 to extract embeddings from the selected regions. An algorithmic overview is shown in Algorithm 1.

Algorithm 1 AutoPOI framework.

```
region_ppo ← build_PPO_net()
dim_red_ppo ← build_PPO_net()
for ep ∈ episodes do
    selected_regions ← select_regions(region_ppo)
    dim_red_net ← create_network(dim_red_ppo)
    POIs ← extract_POIs(traces, selected_regions)
    features ← extract_features(dim_red_net, POIs)
    ranks ← perform_attack(features)
    reward ← calculate_reward(ranks)
    train_PPO_nets(region_ppo, dim_red_ppo)
end for
```

Knowing that the leakages could span multiple raw features (e.g., masked data), the framework aggregates the features to ensure that leakages spanning various points are selected in one go, thereby spanning a greater range of possible leakage points. Specifically, the features are aggregated in regions of length n. This value n is determined by the trace length and the number of regions available as $n = length/regions$. The number of regions available is a hyperparameter that needs to be set beforehand. At each episode, the environment reduces the maximum number of regions r_{cur} with Exponential Decay to explicitly induce an exploration-vs-exploitation dichotomy. The algorithm is set up first such that it has enough room to explore various options. Eventually selecting a smaller number of the best-performing regions. Furthermore, since the search space of features can be rather large, reducing the number of the to-be-selected regions provides a speed up of the framework.

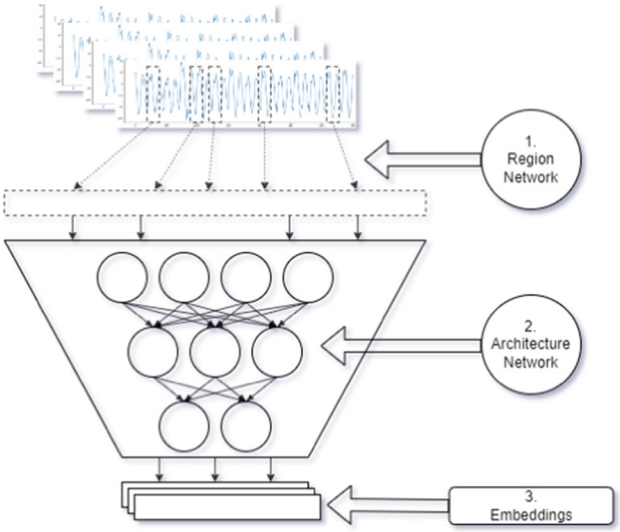


Fig. 3 Graphical overview of AutoPOI framework

To kick-start the learning process of distinguishing between well-performing (sensitive data-related) and bad-performing (others) regions, the maximum and the minimum number of to-be-selected regions $r_{\max}$ and $r_{\min}$ are defined based on a percentage of the total number of regions. Equation (13) gives the Exponential Decay function,

$$r_{\text{cur}} = \max(\lfloor r_0 e^{-\lambda ep} \rfloor, r_{\min}), \tag{13}$$

where λ denotes the decay factor and ep denotes the current episode of the framework. An example of the decay function with $\lambda = 0.002$, $r_{\max} = 1000$ and $r_{\min} = 100$ is shown in Fig. 4.

4.1 Phase 1: feature selection

This phase of the framework is responsible for the selection of multiple regions from the traces. It is based on the Proximal Policy Optimization algorithm, explained in Sect. 2.4.4. The architecture of the PPO network is a neural network with five layers. The architecture of the network is found in Table 1. The average pooling layer is added to reduce the number of

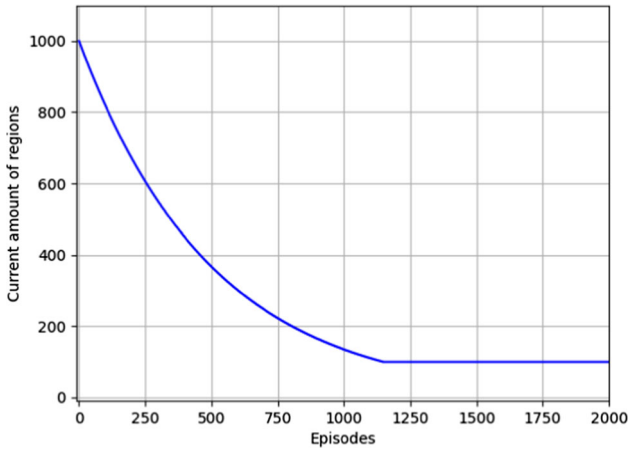


Fig. 4 An example of the exponential decay function in Eq. (13). $\lambda = 0.002$, $r_{\max} = 1000$ and $r_{\min} = 100$

Table 1 Network architecture for the region selection PPO network

Region PPO network	
Avg Pool layer	Kernel = 3, Stride = 2
FC layer	Neurons = 256
Activation layer	ReLU
FC layer	Neurons = 256
Activation layer	ReLU
FC layer	Neurons = 128
Activation layer	ReLU
FC layer	Neurons = 64
Activation layer	ReLU
FC layer	Neurons = action space size

inputs the network has to take into account, thereby speeding up the process. This architecture was chosen based on its relative simplicity. Nevertheless, using other forms of Neural Networks, such as Convolutional Neural Networks, could provide interesting results as well. It may however take more fine-tuning.

The algorithm's layout is shown in Algorithm 2. At each iteration, the environment provides the network with a subset of the profiling traces. The subset size is dictated by the batch size provided in the environment.

In each iteration, one region of all possible regions is selected until it has reached the number of regions to select.

Each trace in the network is run through the network and outputs raw network outputs, often called logits. Since the network is set up to take only one action for multiple inputs, the logits are summed. Then, the summed logits are used to create a categorical distribution from where one action is sampled. This action represents the selected region for that iteration.

The PPO algorithm has the Actor-Critic architecture, as explained in Sect. 2.4.2. This means that a critic value is calculated with a similar network, but outputs only one value.

Algorithm 2 Region Selection Algorithm

Require: $env, region\_model$
  $phase \leftarrow$ **region**
  $obs \leftarrow$ `reset`$(env, phase)$
  while not $done$ **do**
    $logits, val \leftarrow$ `region_model`(obs)
    $logits \leftarrow \sum logits$
    $val \leftarrow mean(val)$
    $m \leftarrow \bar{0}$ ▷ Length of m is determined by logits
    $regions \leftarrow$ `get_selected_regions`(env)
    for each $r \in regions$ **do**
      $m[r] \leftarrow 1$
    end for
    $logits \leftarrow$ `apply_mask`$(logits, m)$
    $dist \leftarrow$ `create_categorical_ditribution`$(logits)$
    $a \leftarrow$ `sample`$(dist)$
    $log\_prob \leftarrow$ `get_log_prob`$(dist, a)$
    $next\_obs, done \leftarrow$ `step`(env, a)
    $train\_data \leftarrow$ `setup_train_data`$(obs, a, val, log\_prob, m)$
    $obs \leftarrow next\_obs$
    if $done$ **then**
      `break_while`
    end if
  end while

This value can be interpreted as a score for the performance of the network. Since the network takes in a batch of traces, there is also a batch of critic values. In this phase, the critic value is averaged, representing the average state of the network. To ensure that regions are not duplicated, invalid action masking (IAM) [13] is applied. IAM is the method of replacing certain logits with a large negative number, such that it defaults to a probability of practically 0 when creating a categorical distribution.

4.2 Phase 2: dimensionality reduction

In the second phase of the framework, the algorithm iteratively builds up a Triplet Tetwork [38]. The Triplet Network is named after the inputs it is provided with, *triplets*. A triplet consists of an anchor a, a positive p, and a negative n. The anchor and the positive share the same label, while the negative has another label. All three are run through the same network, and the loss is calculated as shown in Eq. (14).

$$loss = \mathbf{max}(dist(a, p) - dist(a, n) + margin, 0), \tag{14}$$

where *dist* denotes the Euclidean distance.

As with the previous phase, this phase uses a proximal policy optimization algorithm to find the best triplet network for selected regions. The architecture of the PPO network can be found in Table 2.

The architecture of the PPO network is chosen to reflect the significant difference in state size from the Region Network shown in Table 1. The states built by the environment constitute the current number of layers present in the net-

Table 2 Network architecture for the dimensionality reduction PPO network

Dimensionality reduction PPO network	
FC layer	Neurons = 32
Activation layer	ReLU
FC layer	Neurons = 32
Activation layer	ReLU
FC layer	Neurons = 32
Activation layer	ReLU
FC layer	Neurons = action space size

work, the type of layer selected in the previous step, the output shape of the layer selected in the previous step, and if the algorithm has achieved a terminal state. A general layout of the algorithm is shown in Algorithm 3. At each iteration of the algorithm, a state is processed by the network, returning logits and a value. The action space of Algorithm 3 has also been masked with IAM. In addition, several restrictions have been implemented to guide the algorithm in building valid networks. An overview of these restrictions is found in Table 3; a graphical overview of the state transitions is shown in Fig. 5. Several hyperparameters are made available to be chosen by the algorithm. An overview of each layer and the respective hyperparameters are shown in Table 4.

It is important to note that not only are state transitions restricted, but the hyperparameters that belong to those states as well. Since the network's purpose is to reduce the dimensions, the outputs of the following state cannot exceed the inputs to that state. For instance, if the output of the current state is of dimension 64, every action that leads to a new layer with an output dimension larger than 64 is determined invalid and is masked as such.

Algorithm 3 Dimensionality Reduction Algorithm

```
Require: env, network_model
  phase ← network
  obs ← reset(env, phase)
  while not done do
    logits, val ← network_model(obs)
    m ← determine_mask(obs)
    logits ← apply_mask(logits, m)
    dist ← create_categorical_ditribution(logits)
    act ← sample(dist)
    log_prob ← get_log_probability(dist, action)
    next_obs, done ← step(env, action)
    train_data ← setup_train_data(obs, act, val, log_prob, m)
    obs ← next_obs
    if done then
      break_while
    end if
  end while
```

After the selection of the optimizer, the network is built with the selected layers and hyperparameters. Training is done for 1 epoch with a batch size of 512 and a margin of 0.4 following [38].

4.3 Reward function

For the framework to learn, a reward function is needed. This reward function is an adaptation of the function found in [27]. Two adaptions were made. The first was to remove the notion of the accuracy metric. In [27], the goal was to classify key guesses with a CNN correctly. However, no classification metric is available since no labels are associated with Points Of Interest. Second, since this work focuses solely on generating high-quality POIs, the reward for the size of the networks is removed. The reward function used in this work is shown in Eq. (15).

$$r = \frac{t' + GE'_{10} + 0.5GE'_{50}}{2.5} \tag{15}$$

$$t' = \frac{t_{\max} - \min(t_{\max}, GE_{k^*})}{t_{\max}} \tag{16}$$

$$GE'_{10} = \frac{128 - \min(GE_{10}, 128)}{128} \tag{17}$$

$$GE'_{50} = \frac{128 - \min(GE_{50}, 128)}{128}. \tag{18}$$

Here, r denotes the final reward calculated with three separate reward functions. The first reward function, depicted in Eq. (16), calculates t', which uses the first time the GE of the correct key k^* reaches < 1 and calculates a score between 0 and 1. $t_{\max}$ denotes the number of attack traces used for the attack. Equation (17) calculates a score between 0 and 1 using GE_{10}, which resembles the GE when 10% of the maximum number of traces are used. Finally, Eq. (18) calculates a score between 0 and 1 using GE_{50}, which resembles the GE when 50% of the traces are used.

The last two metrics are added to ensure that, although a complete GE convergence was not achieved given a certain number of attack traces, the actions taken are not disregarded as entirely wrong. These metrics were chosen to incentivize the learning to focus on reducing the number of traces needed to converge to the correct key guess, ultimately leading to better features.

5 Datasets

5.1 ASCAD

The ASCAD dataset [2] is created by acquiring EM traces from an ATMega8515 controller running an AES-128 implementation. The chip card itself has no hardware security implementation. The authors implemented masking to counter first-order side-channel attacks [24].

Table 3 On overview of each state and the transition restrictions

State	Restrictions
Null state	Can only transition to pooling, convolutional or terminal layers
Pooling layer	Cannot transition to other pooling layers
Convolutional layer	No restrictions
Activation layer	Cannot transition to other activation layers
Fully connected layer	No restrictions
Terminal layer	Can only transition to optimizers

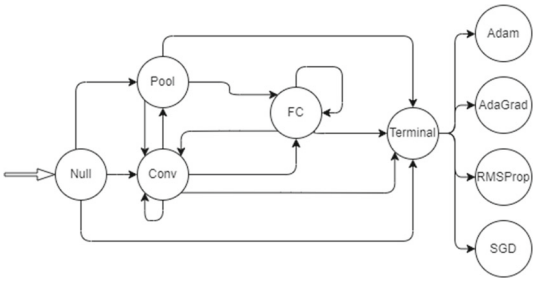


Fig. 5 Graphical overview of the state transitions. Note that for ease of viewing, activation layers are removed

ASCAD_F: This dataset version has a *fixed key* and consists of 50,000 profiling traces for profiling and 10,000 attack traces. Note that traces with 700 features (requires knowledge of r mask share) are commonly used in related works. To make our work closer to realistic settings, we select a time window with 5000 features, corresponding to the Sbox output when using key byte 3, the first masked key byte. A total of 45,000 traces are used as the profiling set, this set is used to train the proposed framework. For the calculation of the rewards, a separate set of attack traces is used consisting of 5000 traces. For testing purposes, another 5000 traces are used.

ASCAD_R: This dataset version has *random keys*, with 200,000 traces for profiling and 100,000 for the attack. The keys are randomized for 33% of the attack traces. Similarly, we extend the pre-selected window to 5000 features corresponding to the processing of the third masked key byte based on SNR of the Sbox output. As with the fixed key dataset, a total of 45,000 traces are used as the profiling set. Again, this set is used to train the proposed framework. For the calculation of the rewards, a separate set of attack traces is used consisting of 5000 traces. For testing purposes another 5000 traces are used. Note that for reward and testing purposes, the keys are fixed and not randomized.

Table 5 Hyperparameters for the training of the PPO networks

Hyperparameter	Value
Policy learning rate	0.0003
Value learning rate	0.0001
Training epochs	20

For both ASCAD_F and ASCAD_R, the hamming weight (HW) and Identity (ID) leakage models are used to benchmark the proposed framework.

5.2 AES_HD dataset

The AES_HD dataset [3] is a dataset created by measuring EM emission from an unprotected Xilinx Virtex-5 FPGA. This dataset has a *fixed key*. This work uses the input and output of the last round SBox ($Sbox^{-1}(c_7 \oplus k_7) \oplus c_{11}$) as explained by Picek et al. [23]. As with previous datasets and to create a more equal experimental environment, again 45,000 traces are selected for the training of the proposed framework. Both the reward and final testing sets contain 5000 traces. The traces selected contain a total of 1250 features. For the AES_HD dataset, the HD leakage model is used to benchmark the proposed framework.

5.3 CHES_CTF dataset

The CHES_CTF dataset is a *fixed key* data set created for the annual Capture-The-Flag event organized by the Conference on Cryptographic Hardware and Embedded Systems (CHES). The traces are taken from a 32-bit STM Controller running a masked AES-128 encryption algorithm.

Table 4 Hyperparameter overview of the possible combinations of layers

Layer	Hyperparameters	
Fully Connected layer	Neurons: [256, 128, 64, 32, 16, 8]	
Convolutional layer	Kernel: [256, 128, 64, 32, 16, 8]	Stride: [16, 8, 4, 2, 1]
Pooling Layer	Kernel: [256, 128, 64, 32, 16, 8]	Stride: [16, 8, 4, 2, 1]
Activation layer	Type: ReLU, Tanh, SeLU	
Embeddings layer	Neurons: [64, 32, 16, 8]	
Optimizer	Type: Adam, AdaGrad, RMSProp, SGD	LR: [1e−4, 3e−4, 1e−3, 3e−3, 1e−2, 3e−2]

This dataset originates from the year 2018 and is publicly available [5]. Similarly as with previous datasets, a total of 45,000 traces are used to compose the training set. For the reward and test set, 5000 traces are used. The traces have a dimension of 2200 features. For the CHES_CTF dataset, the HW leakage model is used to benchmark the proposed framework.

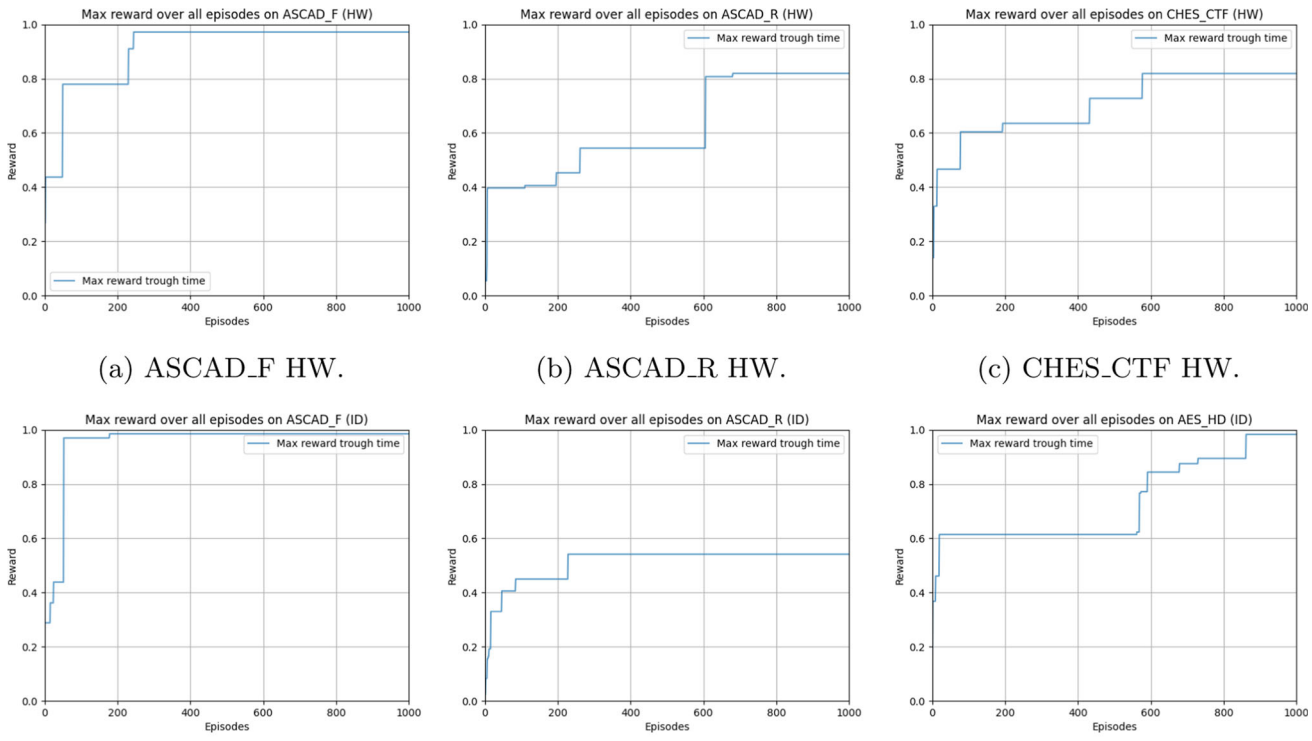
6 Experimental environment

The proposed framework is trained for a total of 1000 episodes. During training, the framework chooses a set of regions based on a batch size of 512 traces. Subsequently, the framework chooses a network architecture. As described in Algorithm 3, the selected network is trained for 1 epoch with a batch size of 512 and a margin of 0.4. Training of the PPO algorithms of the proposed framework is done with the hyperparameters given in Table 5. The hyperparameters are chosen based on previous implementations [25] and the algorithm's authors recommendations [32].

7 Results and discussion

The proposed framework was run on each dataset for a total of 1000 episodes, in which, during training, the network found, selected, and scaled-down various numbers of Points Of Interest. The best-performing set of POIs and the best-performing NN were then used for the guessing entropy calculation (averaged over 100 attacks). Training on the AES_HD dataset took 8 h, and training on CHES_CTF took 9 h. For both ASCAD datasets, training took 12 h.

To gain insight into the learning of the proposed framework, Fig. 6 shows the max reward through time (episode). One can observe that the reward constantly increases with more episodes, meaning that the framework is learning from the environment and gradually producing better attack results. Specifically, the results show that the proposed framework found a well-performing set of POIs and a network architecture within 244 episodes for the ASCAD_F dataset with HW and 178 episodes for the Identity model. For the ASCAD_R datasets, the proposed framework reaches the highest reward within 681 episodes for the Hamming Weight leakage model and 228 episodes for the Identity leakage model. For the AES_HD datasets, the highest reward is reached within 862 episodes, and for the CHES_CTF dataset, the good POIs and network architecture were found within 577 episodes.



(a) ASCAD_F HW. (b) ASCAD_R HW. (c) CHES_CTF HW.

(d) ASCAD_F ID. (e) ASCAD_R ID. (f) AES_HD HD.

Fig. 6 Rewards during training on different datasets and leakage models

Table 6 A summary of the results of each method on each of the four datasets (HW/HD)

Dataset	SOST	SNR	PCA	LDA	Triplet	AutoPOI
AES_HD	> 5000	1094	2513	1104	1664	**990**
CHES_CTF	4510	> 5000	> 5000	> 5000	> 5000	**1830**
ASCAD_F	4522	1184	203	> 5000	194	**193**
ASCAD_R	> 5000	> 5000	452	> 5000	**164**	1499

Bold indicates best performance

Table 7 A summary of the results of each method on each of the two datasets (ID)

Dataset	SOST	SNR	PCA	LDA	Triplet	AutoPOI
ASCAD_F	> 5000	> 5000	436	> 5000	**158**	180
ASCAD_R	> 5000	> 5000	> 5000	> 5000	> 5000	> 5000

Bold indicates best performance

Among all tested settings, except ASCAD_R with the ID leakage model, all tested settings reach a reward above 0.8, indicating the framework manages to find both promising input regions and triplet network architectures via interactions. The results show the proposed framework's effectiveness in finding good POIs and network architectures.

Next, we benchmark our framework with different POI selecting methods and Template Attack. Specifically, both conventional methods and Deep Learning methods are taken into consideration. As can be seen from Tables 6 and 7, which provide GE to reach < 1 for each dataset, the proposed framework is the only method able to provide consistent results. Especially when using the Hamming Weight leakage model, the proposed framework is the only method that can break all four datasets, as observed in Table 8. Not only is the proposed framework consistent with finding POIs, but it can also find optimized POIs such that it attains state-of-the-art performance for three out of four datasets for the Hamming Weight. Although our framework fails to break ASCAD_R with the ID leakage model in the current setting, increasing the number of episodes could be a possible solution. On the other hand, the conventional feature selection methods and triplet networks are only functional with specific settings. Therefore, it can be considered that our approach is more general in terms of point of interest selection.

Table 8 An overview of the percentage of finding the correct partial key within the maximum amount of traces

SOST	SNR	PCA	LDA	Triplet	AutoPOI
0.33	0.33	0.66	0.16	0.66	0.83

8 Conclusions and future work

This paper introduces a novel reinforcement learning-driven framework, AutoPOI, based on Proximal Policy Optimization, which can find, select, and scale down POIs. The framework analyzes leakage traces and designates regions of features. The proposed framework selects several of these regions as POIs. After that, the framework constructs a neural network to provide a scaled-down version of the selected regions. Template attacks are mounted with these scaled-down features, and rewards are given based on the attack performance obtained using a specific reward trace set. The framework automatically adapts to the rewards given, thereby finding the best-performing regions and networks tailored to each dataset.

The attack performance, represented by guessing entropy, is extensively tested for each dataset. The results show that the framework can break almost all datasets where the current state-of-the-art methods cannot. Furthermore, the proposed framework is efficient in finding promising POIs and network architectures, achieving state-of-the-art performance for most attack settings. Not only is the running time of the algorithm short compared to other currently used methods, but the results also show that early on during training, the proposed framework can find well-performing POIs and network architectures. For future work, it would be interesting to test the framework on more datasets and several common countermeasures, such as desynchronization and noisy (Gaussian noise) data. Furthermore, implementing an early-stopping mechanism would be helpful in reducing the time consumption of the framework.

Declarations

Conflicts of interest The authors declare no competing interests.

References

1. Archambeau, C., Peeters, E., Standaert, F.X., Quisquater, J.J.: Template attacks in principal subspaces. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 1–14. Springer (2006)
2. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptogr. Eng. **10**(2), 163–188 (2020)
3. Bhasin, S., Jap, D., Picek, S.: AES HD dataset—50,000 traces. AISyLab repository (2020). https://github.com/AISyLab/AES_HD
4. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 13–28. Springer (2002)
5. Ches ctf - dataset (2022). https://chesctf.riscure.com/2018/content?show=training
6. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: International Conference on Smart Card Research and Advanced Applications, pp. 253–270. Springer (2013)
7. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon. Submission to the CAESAR competition: http://ascon.iaik.tugraz.at (2014)
8. Fisher, R.A.: The use of multiple measurements in taxonomic problems. Ann. Eugen. **7**(2), 179–188 (1936)
9. Gierlichs, B., Lemke-Rust, K., Paar, C.: Templates vs. stochastic methods. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 15–29. Springer (2006)
10. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 106–111. IEEE (2015)
11. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptogr. Eng. **1**(4), 293–302 (2011)
12. Hotelling, H.: Analysis of a complex of statistical variables into principal components. J. Educ. Psychol. **24**(6), 417 (1933)
13. Huang, S., Ontañón, S.: A closer look at invalid action masking in policy gradient algorithms. arXiv preprint arXiv:2006.14171 (2020)
14. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual International Cryptology Conference, pp. 388–397. Springer (1999)
15. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951)
16. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: International Workshop on Constructive Side-Channel Analysis and Secure Design, pp. 20–33. Springer (2015)
17. Lin, L.J.: Reinforcement learning for robots using neural networks. Carnegie Mellon University (1992)
18. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering, pp. 3–26. Springer (2016)
19. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards, vol. 31. Springer, Berlin (2008)
20. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
21. Perin, G., Wu, L., Picek, S.: Exploring feature selection scenarios for deep learning-based side-channel analysis. Cryptology ePrint Archive (2021)
22. Picek, S., Heuser, A., Jovic, A., Batina, L.: A systematic evaluation of profiling through focused feature selection. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. **27**(12), 2802–2815 (2019)
23. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 1–29 (2019)
24. Prouff, E., Rivain, M.: A generic method for secure sbox implementation. In: International Workshop on Information Security Applications, pp. 227–244. Springer (2007)
25. Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N.: Stable-baselines3: reliable reinforcement learning implementations. J. Mach. Learn. Res. **22**(1), 12348–12355 (2021)
26. Ramezanpour, K., Ampadu, P., Diehl, W.: SCARL: side-channel analysis with reinforcement learning on the ascon authenticated cipher. arXiv preprint arXiv:2006.03995 (2020)
27. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. In: IACR Transactions on Cryptographic Hardware and Embedded Systems, pp. 677–707 (2021)
28. Rioja, U., Batina, L., Flores, J.L., Armendariz, I.: Auto-tune POIs: estimation of distribution algorithms for efficient side-channel analysis. Comput. Netw. **198**, 108405 (2021)
29. Roy, D.B., Bhasin, S., Guilley, S., Heuser, A., Patranabis, S., Mukhopadhyay, D.: CC meets FIPS: a hybrid test methodology for first order side channel analysis. IEEE Trans. Comput. **68**(3), 347–361 (2018)
30. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: International Workshop on Cryptographic Hardware and Embedded Systems, pp. 30–46. Springer (2005)
31. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897. PMLR (2015)
32. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
33. Shelton, C.R.: Importance sampling for reinforcement learning with multiple objectives (2001)
34. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, pp. 443–461. Springer (2009)
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
36. Sutton, R.S., McAllester, D., Singh, S., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems, vol. 12 (1999)
37. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Mach. Learn. **8**(3), 229–256 (1992)
38. Wu, L., Perin, G., Picek, S.: The best of two worlds: deep learning-assisted template attack. Cryptology ePrint Archive (2021)