Avenger Ensemble: Genetic Algorithm-Driven Ensemble Selection for Deep Learning-based Side-Channel Analysis

Zhao Minghui[1[0009−0002−2928−0299]] and Trevor Yap[1,2[0000−0001−8651−574X]]

[1] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore

[2] Temasek Laboratories, Nanyang Technological University, Singapore

minghui002@e.ntu.edu.sg, trevor.yap@ntu.edu.sg

Abstract. Side-Channel Analysis (SCA) exploits physical vulnerabilities in systems to reveal secret keys. With the rise of Internet-of-Things, evaluating SCA attacks has become crucial. Profiling attacks, enhanced by Deep Learning-based Side-Channel Analysis (DLSCA), have shown significant improvements over classical techniques. Recent works demonstrate that ensemble methods outperform single neural networks. However, almost every existing ensemble selection method in SCA only picks the top few best-performing neural networks for the ensemble, which we coined as Greedily-Selected Method (GSM), which may not be optimal. This work proposes Evolutionary Avenger Initiative (EAI), a genetic algorithm-driven ensemble selection algorithm, to create effective ensembles for DLSCA. We investigate two fitness functions and evaluate EAI across four datasets, including AES and Ascon implementations. We show that EAI outperforms GSM, recovering secrets with the least number of traces. Notably, EAI successfully recovers secret keys for Ascon datasets where GSM fails, demonstrating its effectiveness.

Keywords: Side-channel analysis · Deep learning · Ensemble learning · Genetic algorithm

1 Introduction

Side-channel analysis (SCA) exploits the physical vulnerabilities of a system like power consumption [8] and electromagnetic emanation [2] to reveal the secret key. Evaluating such attacks has become crucial, especially with the rise in the usage of Internet-of-Things (IoTs) in recent years [12]. One such SCA is known as the profiling attack. In the profiling attack model, it is assumed that the attacker has access to a clone device that closely resembles the target device. Deep Learning-based Side-Channel Analysis (DLSCA) has been intensively explored in recent years [13, 14, 1, 22]. It was demonstrated that with Deep Neural Networks (DNNs) it can significantly outperform classical SCA techniques like template attacks [13]. Furthermore, DLSCA has garnered significant interest because of its capability to recover the secret key of the protected implementations without needing to resynchronize traces.

Recent works have shown that using an ensemble of DNNs shown to have significant performance improvement compared to using a single best neural network [17, 18, 23]. DNNs of varying hyperparameters can extract different information from identical traces, yielding improved performance when combined together as an ensemble. Most of the works in DLSCA create an ensemble from greedily selecting the top few best-performing DNNs [17–19], herein referred to as the Greedily-Selected Method (GSM). However, the GSM may not be optimal as other combinations of DNNs could attain better results. Therefore, a natural question is:

Can we develop a methodology to automatically select neural networks for efficient ensemble formation?

To address this, we focus on ensemble selection, which is defined as the process of selecting pre-trained DNNs to create an effective ensemble.

Our Contributions. Our contributions are stated as follows:

1. In this work, we proposed a new genetic algorithm-driven ensemble selection algorithm called Evolutionary Avenger Initiative (EAI) to generate a best-performing ensemble. To the best of our knowledge, this is the first work to investigate ensemble selection within the context of SCA. Furthermore, we investigate two fitness functions for EAI: ge_{+ntge} and validation loss.
2. We assess the efficiency of our methodology across four datasets, comprising three AES datasets (software and hardware implementations) of the widely recognized symmetric-key standard [5], and one Ascon dataset, NIST's lightweight cryptography standard winner, implemented in software [6]
3. By integrating EAI, we manage to recover the secret with the least number of traces across all datasets tested, outperforming traditional ensemble selection techniques, i.e., GSM. Furthermore, EAI with ge_{+ntge} has yielded significant performance gains across all tested settings and datasets. Notably, for the Ascon datasets (i.e., Ascon2 and Ascon4), EAI successfully recovered the secret key within 1000 attack traces, whereas the conventional GSM method failed to do so. This highlights the effectiveness of EAI.

We validate our approach on first-order masking traces, leaving higher-order masking for future work. The source code can be accessed at the following web link. [3]

Paper Organization: The paper is organized as follows. Section 3 provides the background necessary for profiled SCA using an ensemble of DNN and Genetic Algorithm. Section 4 presented our proposed genetic algorithm methodology to generate the best-performing ensemble for profiling attacks. Section 5 presents and analyzes our results. Finally, Section 6 concludes with some discussion and provides some future works.

[3] anonymous for reviews

2 Related Work

Ensemble within DLSCA The introduction of ensemble methods in DLSCA is attributed to [17]. This pioneering work presented a methodology for computing the maximum log-likelihood of an ensemble for bagging and showcased its efficacy on AES -based datasets. However, they use GSM to build their ensemble, which may be suboptimal. Subsequently, [18] successfully apply GSM to Ascon implementations. Instead of considering GSM, the authors of [23] consider the "diversity" between the optimal DNNs and the non-optimal DNNs. Therefore, they proposed a loss function called Ensemble loss. The Ensemble loss helps to train a diverse ensemble to recover the secret. Nonetheless, the Ensemble loss has a drawback: it cannot scale for more than three classes. Consequently, its application is restricted, and it cannot be applied to block ciphers like AES (with 256 classes). The authors only tested it on an ECC dataset. Unlike [23], we do not take into account the training process but consider the same scenarios as [17]. Perin et al. [17] assume that a set of DNNs with randomly generated hyperparameters have already been trained. We highlight that the above methodologies considered are bagging [4]. In recent years, methods for stacking DNNs [11] and boosting DNNs [20] have been explored in SCA. For this work, we will focus on bagging ensembles. Furthermore, we exploit the capabilities of the genetic algorithm to find the optimal combination to produce the best-performing ensemble.

Ensemble Construction/Selection The idea of constructing an ensemble is not new, as various works have considered [9, 24]. Prior works use Bayesian optimization to tune the hyperparameters of one neural network while building the ensemble [9] or using Bayesian optimization to form an ensemble through trained DNNs [15]. The authors of [24] model the ensemble construction as an optimization problem. They propose a method called Neural Ensemble Search to solve this optimization problem in order to attain a well-performing ensemble. For our work, we consider ensemble selection methodology. There have been advancements in ensemble selection that leveraged genetic algorithms to optimize ensemble composition. Notably, [25] proposed GASEN, which utilizes genetic algorithms to optimize weight coefficients that capture inter-DNN correlations. Specifically, GASEN selects DNNs for the ensemble based on their weight coefficients, including only those exceeding a predefined threshold. Ortiz et al. propose EARN, a multi-objective evolutionary approach that generates efficient DNN ensembles [16] where they consider bagging, boosting, and stacking into the ensemble. Inspired by these, we develop a genetic algorithm for ensemble selection within SCA context.

3 Background

3.1 Profiling attack using Ensemble

One of the most common side-channel settings is known as the profiling attack. It assumes the worst-case scenario where the adversary has access to a clone device similar to the target device. The profiling attack is executed in two phases: profiling and attack phase.

In the profiling phase, the adversary either knows or can manipulate the key of the clone device. Then, distinguishers can be built from the profiling traces of a known set of random public variables (plaintext or ciphertext). During this phase, the adversary collects a set of traces $\boldsymbol{t}$ corresponding to known public variables to train distinguishers.

In the attack phase, the adversary performs the attack by collecting several attack traces from another set of known public variables of the target device. Typically, the traces are given to a single trained distinguisher for key recovery.

In [17] consider multiple distinguishers for key recovery. Formally, the traces are given to the trained distinguishers to obtain their output probability scores for each hypothetical sensitive value. These probabilities are combined together:

$$score(k) = \sum_{j=1}^{N_{model}} \sum_{i=1}^{N_a} \log(Pr_j(Z = z_{i,k}|\mathbf{t}_i)) \tag{1}$$

where N_a represents the number of attack traces used, N_{model} is the number of models in the ensemble, $Pr_j(Z = z_{i,k}|\mathbf{t}i)$ denotes the probability output by the j^{th} distinguisher in the ensemble, and $z_{i,k}$ is the hypothetical sensitive value which depends on the key candidate k and the trace $\mathbf{t}_i$.

The $score(k)$ is computed for each key $k \in \mathcal{K}$, where $\mathcal{K}$ is the set of all possible key values. An attacker can sort the scores in descending order to create a guess vector $[G_0, \ldots, G_{|\mathcal{K}|}]$ where G_0 corresponds to the score for the most likely key candidate while the $G_{|\mathcal{K}|}$ represents the score for the least likely key candidate. Let the index of the guess vector be the rank of the key. Then, we define the Guessing Entropy (GE) as the rank of the correct key averaging over multiple experiments. In our analysis, we calculate this average over 100 separate experiments. When GE reaches zero, it indicates a completely successful attack - the correct key was consistently ranked first. To quantify attack performance, we denote the least number of traces for GE to reach zero as $NTGE$, which measures the minimum number of traces needed to achieve $GE = 0$.

3.2 Genetic algorithm framework

The genetic algorithm is a population-based optimization method inspired by natural evolution. Let $\mathcal{X}$ be the search space, and $f : \mathcal{X} \to \mathbb{R}$ be the fitness function that evaluates potential solutions. The population at generation t is denoted as

$$P^t = \{x_1^t, x_2^t, \ldots, x_n^t\},$$

where each individual chromosome $x_i^t \in \mathcal{X}$ represents a candidate solution, also referred to as a chromosome.

By evolving the population over successive generations, the genetic algorithm seeks to improve the overall fitness of the population, moving closer to the optimal solution. Each individual chromosome x_i^t is evaluated using the fitness function f, which assigns a numerical score to indicate its suitability or quality as a solution. The fitness scores guide the selection process, helping to determine which chromosomes will contribute genetic material to the next generation. The evolution process consists of the following phases:

1. **Initial Population:** The initial population P^0 is typically generated randomly across the search space $\mathcal{X}$.
2. **Elitism:** It preserves the best chromosomes from one generation to the next. The basic idea is to ensure that the fittest chromosomes are included in the next generation without any changes.
3. **Selection Method:** In genetic algorithms, several distinct selection methods exist for choosing chromosomes to generate subsequent generations. We shall recall the commonly used **Tournament Selection**. The Tournament Selection process in a genetic algorithm involves sampling multiple tournaments of size ℓ, where ℓ chromosomes are randomly chosen from the population. The fittest chromosome in each tournament is selected to generate new chromosomes for the next generation.
4. **Generic Operator:** Various genetic operators are proposed to generate new chromosomes; here, we will recall two such operators: Crossover and Mutation.
   - **Crossover:** The crossover operator is inspired by biological reproduction, where genetic material from two or more parent chromosomes is combined to produce one or more offspring. This process creates new chromosomes by exchanging segments between parents, thus propagating beneficial traits across generations. Crossover is typically applied with a high probability, denoted by Pr_{cross}, to enhance the convergence rate towards optimal solutions by exploiting existing genetic diversity.
   - **Mutation:** Mutation introduces variability and maintains diversity within a population. It involves making small, random changes to the values (or "genes") within an individual chromosome. These random tweaks help the algorithm explore new potential solutions, preventing premature convergence to suboptimal solutions. Mutation is typically applied with a low probability, denoted by $Pr_{\text{mut}} = 1 - Pr_{\text{cross}}$, to avoid drastic changes while enabling steady exploration of the solution space.

4 Evolutionary Avengers Initiative

In this section, we present our algorithm, called the Evolutionary Avengers Initiative (EAI). Each DNN can be viewed as a "superhero" endowed with distinct strengths. Drawing inspiration from the Avengers of the Marvel Cinematic Universe, EAI aims to assemble a diverse group of "superheroes" called the Avenger Ensemble to recover the secret key.

The overview of the EAI Framework is described in Algorithm 1. EAI first initializes the population using a set of pre-trained DNNs from set $\mathcal{M}$. The population consists of N_{ens} ensembles, with each ensemble consisting of N_{model}

DNNs. Subsequently, the steps described below are repeated over N_{gen} generations. Firstly, each ensemble within the population is evaluated using the fitness function (Line 4 of Algorithm 1). Next, the best-performing ensemble is then preserved for the next generation, a process known as elitism (Line 4 of Algorithm 1). Finally, generic operations such as crossover or mutation are applied to generate new ensembles (Line 5 of Algorithm 1). We shall describe each step in detail.

Algorithm 1 Evolutionary Avengers Initiative (EAI)

```
    Input: $\mathcal{M}$: Set of all trained models, $N_{ens}$: Population size, $N_{model}$: Number of
models per ensemble, $Pr_{\text{cross}}$: Crossover rate, $Pr_{\text{mut}}$: Mutation rate, ge_fitness_fn:
Fitness function, $N_{gen}$: Number of generations
    Output: $E_{best}$: Best-performing ensemble, $s_{best}$: Fitness value of $E_{best}$
1: Initialize Population of First Generation: $P = initialize\_pop(\mathcal{M}, N_{ens}, N_{model})$
2: $E_{best} = \emptyset, s_{best} = \infty$
3: for $t \leftarrow 1$ to $N_{gen}$ do
4:     Evaluate population and apply Elitism:
           $P_{new}, S, E_{best}, s_{best} = Eval\_\&\_Elite(\texttt{ge\_fitness\_fn}, P, P_{new}, E_{best}, s_{best})$
5:     Apply Crossover or Mutation:
           $P_{new} = GeneticOp(\mathcal{M}, P, S, Pr_{\text{cross}}, Pr_{\text{mut}})$
6:     $P = P_{new}$
7: end for
8: return $E_{best}, s_{best}$
```

Initialize Population. Prior to the start of EAI, we train a set of DNNs, denoted as $\mathcal{M}$. We label each DNN with M_i for $0 \leq i \leq |\mathcal{M}|$. EAI will first initialize the population (Line 1 in Algorithm 1). Algorithm 2 describes how to initialize the population. The initial population P^0 is generated by picking N_{model} distinct models from $\mathcal{M}$ randomly to create a new ensemble E (Line 5 in Algorithm 2). Then, we check if ensemble E is a duplicate in the initial population P^0 (Line 6 in Algorithm 2). If it is, E is discarded; if not, the ensemble E is added into the initial population of ensembles (Line 6 to 8 in Algorithm 2). This process is repeated until there are N_{ens} ensembles within the population P^0.

Evaluate Population and Apply Elitism. In this step, all the ensembles' performances are evaluated using a fitness function. Furthermore, the top-performing ensemble is preserved for the next generation. Algorithm 3 provides the overall methodology. Moreover, when evaluating the ensembles' performance, the best-

Algorithm 2 Initialize Population

```
1: procedure $initialize\_pop(\mathcal{M}, N_{ens}, N_{model})$
2:     $P^0 \leftarrow \emptyset$                                   ▷ Population of unique ensembles
3:     $i = 0$
4:     while $i < N_{ens}$ do
5:         $E \leftarrow$ Randomly select $N_{model}$ distinct models from $\mathcal{M}$.
6:         if $E$ is not a duplicate in $P^0$ then
7:             $P^0 \leftarrow P^0 \cup \{E\}$                       ▷ Add ensemble to population
8:             $i = i + 1$
9:         end if
10:    end while
11:    return $P^0$
12: end procedure
```

performing ensemble throughout all the generations is recorded (line 6 to 8 of Algorithm 3).

- **Elitism:** Next, when applying elitism to preserve the ensemble(s) for the next generation. EAI only considers the top-performing ensemble within this population to be preserved. It is possible to preserve more than one top-performing ensemble, but it requires sorting the performance of the ensembles, which is time-consuming. Therefore, we only select the top ensemble in the population. Furthermore, this enhances the search capability, allowing for more exploration of a diverse range of ensembles to identify the optimal combination.
- **Fitness Functions:** We investigate two fitness functions for EAI in the context of SCA. Namely, we explore ge_{+ntge} and val_loss as the fitness function for ensemble selection.
  1. $\boldsymbol{ge_{+ntge}}$ **:** The fitness function $\boldsymbol{ge_{+ntge}}$ is first proposed by [7]. The authors demonstrate that the metric yields consistently better results when combined with multifidelity hyperparameter tuning, called Bayesian Optimization HyperBand, to identify a single best-performing DNN. This composite metric combines GE with $NTGE$ to provide a comprehensive evaluation of side-channel analysis performance. The ge_{+ntge} function is defined as:

$$ge_{+ntge}(\theta) = \begin{cases} NTGE & \text{if } GE = 0, \\ GE + N_a + c & \text{otherwise} \end{cases}$$

     where θ represents the model configuration/hyperparameters, N_a is the fixed number of attack traces for evaluation, and c is a small positive constant (set to 100 in our experiments). This metric considers both the ability to recover the key (GE) and the efficiency of the attack ($NTGE$). It also penalizes configurations that fail to recover the key within the given number of traces. This provides a single, comprehensive metric for optimizing SCA models.

2. **val_loss:** Research has shown that minimizing categorical cross-entropy loss effectively maximizes the mutual information between the leakage model and the trace data, a concept referred to as perceived information in SCA. Here, we investigate whether using the validation loss of attack traces as a fitness function, denoted as val_loss, can aid in ensemble selection.

Algorithm 3 Evaluate Population and Apply Elitism

```
Require:
 1: procedure $Eval\_\&\_Elite(\texttt{ge\_fitness\_fn}, P, P_{new}, E_{best}, s_{best})$
 2:     $S \leftarrow []$                                   ▷ Array of scores for current generation
 3:     for each ensemble $E \in P$ do
 4:         $s = \texttt{ge\_fitness\_fn}(E)$
 5:         $S \leftarrow S + [s]$                          ▷ Array Concatenation
 6:         if $s < s_{best}$ then
 7:             Set $s_{best} = s$ and $E_{best} = E$.
 8:         end if
 9:     end for
10:     $top\_pop\_ensemble \leftarrow \arg\min_{E \in P} S[E]$
11:     $P_{new} \leftarrow \{top\_pop\_ensemble\}$        ▷ Elitism: retain the best ensemble
12:     return $P_{new}, S, E_{best}, s_{best}$
13: end procedure
```

Genetic Operators. The remaining ensembles in the population are generated using tournament selection, crossover, and mutation, which replace the non-elite ensembles. This configuration supports diversity within the population, which is critical for the effectiveness of subsequent generations.

Algorithm 4 Genetic Operators with Adaptive Rates

```
1: procedure GeneticOp($\mathcal{M}, P, S, Pr_{\text{cross}}, Pr_{\text{mut}}$)
2:     $\delta \leftarrow |set(tuple(ind)$ for $ind$ in $P)|/|P|$    ▷ Calculate population diversity
3:     $Pr_{\text{cross}}^{\text{adaptive}} \leftarrow Pr_{\text{cross}} \times \delta$    ▷ Adaptive crossover rate
4:     $Pr_{\text{mut}}^{\text{adaptive}} \leftarrow Pr_{\text{mut}} \times (1 + (1 - \delta))$    ▷ Adaptive mutation rate
5:     while $|P_{new}| < N_{ens}$ do
6:         if random() $< r_{cross}^{adaptive}$ then    ▷ Adaptive crossover operation
7:             $\texttt{parent}_1 \leftarrow$ TournamentSelection($P, S$)
8:             $\texttt{parent}_2 \leftarrow$ TournamentSelection($P, S$)
9:             $\texttt{offspring}_1, \texttt{offspring}_2 \leftarrow$ Crossover($\texttt{parent}_1, \texttt{parent}_2, N_{model}$)
10:            $P_{new} \leftarrow P_{new} \cup \{\texttt{offspring}_1, \texttt{offspring}_2\}$
11:        else    ▷ Adaptive mutation operation
12:            $\texttt{parent} \leftarrow$ TournamentSelection($P, S$)
13:            $P_{new} \leftarrow P_{new} \cup \{$Mutation($\texttt{parent}, \mathcal{M}, r_{mut}^{adaptive}$)$\}$
14:        end if
15:    end while
16:    return $P_{new}$
17: end procedure
```

Algorithm 4 outlines the methodology used to generate new offspring ensembles. To ensure diversity and effectiveness in the evolutionary process, we employ adaptive genetic operators [10], where the rates dynamically adjust based on population diversity (i.e., the proportion of unique ensembles in the population). This adaptive approach helps mitigate premature convergence and redundancy, which can arise from fixed rates.

Let δ represent population diversity, calculated as the ratio of unique ensembles to the total population size. The adaptive mutation rate is defined as:

$$Pr_{\text{mut}}^{\text{adaptive}} = Pr_{\text{mut}} \times (1 + (1 - \delta)) .$$

When δ is low, this formula increases the mutation rate since $(1 - \delta)$ is larger. A higher mutation rate potentially introduces new models from the set of all the pre-trained DNN, $\mathcal{M}$, into the population, encouraging exploration and generating novel ensembles. Conversely, when diversity is high, the mutation rate decreases to avoid disrupting the existing diversity.

Similarly, the adaptive crossover rate is defined as:

$$Pr_{\text{cross}}^{\text{adaptive}} = Pr_{\text{cross}} \times \delta.$$

This rate increases proportionally with δ, allowing for more frequent recombination within the population when the population contains a wide variety of ensembles. A high δ indicates ample genetic material, making crossover more effective in creating new and potentially superior ensembles. On the other hand, when δ is low, the reduced crossover rate prevents the overexploitation of similar ensembles. At the same time, increasing the probability of mutation allows the population to have more unique ensembles.

To implement these adaptive mechanisms, we set the base mutation probability to $Pr_{\text{mut}} = 0.1$ and the base crossover probability to $Pr_{\text{cross}} = 0.9$, following standard practice in genetic algorithms. By maintaining the diversity of ensembles within the population through adaptive mutation and crossover rates, this approach promotes the generation of unique and high-performing ensembles throughout the evolutionary process.

1. **Tournament Selection:** The selection method that EAI uses would be the Tournaments Selection. For each tournament, ℓ ensembles are randomly selected from the population. Using the fitness value computed before, pick the best-performing ensemble out of the ℓ ensemble. This ensemble will be called a parent ensemble. Figure 1 provides an illustration of a Tournament Selection when $\ell = 3$.

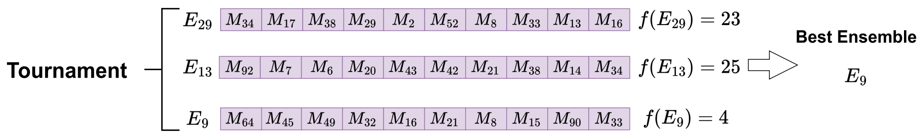

Tournament
E_{29} M_{34} M_{17} M_{38} M_{29} M_{2} M_{52} M_{8} M_{33} M_{13} M_{16} $f(E_{29}) = 23$
E_{13} M_{92} M_{7} M_{6} M_{20} M_{43} M_{42} M_{21} M_{38} M_{14} M_{34} $f(E_{13}) = 25$
E_{9} M_{64} M_{45} M_{49} M_{32} M_{16} M_{21} M_{8} M_{15} M_{90} M_{33} $f(E_{9}) = 4$
Best Ensemble
E_9

Fig. 1: Illustration of a Tournament Selection within EAI when $\ell = 3$.

2. **Crossover:** For Crossover, it uses two tournaments for creating two ensembles (aka offspring) as shown in Figure **??**. Assume that the ensembles $parent_1$ and $parent_2$ have the best fitness scores for two different tournaments. Then Crossover randomly generates two indexes c_1 and c_2. The two offspring ensembles are produced via:

$$\texttt{offspring}_1[j] = \begin{cases} \texttt{parent}_1[j] & \text{if } j < c_1 \text{ or } j > c_2 \\ \texttt{parent}_2[j] & \text{if } c_1 \leq j \leq c_2 \end{cases} \tag{2}$$

$$\texttt{offspring}_2[j] = \begin{cases} \texttt{parent}_2[j] & \text{if } j < c_1 \text{ or } j > c_2 \\ \texttt{parent}_1[j] & \text{if } c_1 \leq j \leq c_2 \end{cases} \tag{3}$$

   Figure 2 provides an illustration of Crossover when $c_1 = 3$ and $c_2 = 6$ are chosen. This indicates that the segments from position 3 to 6 (inclusive) are swapped between the parents We highlight that if crossover results in duplicate individuals within the population, the process is repeated to ensure uniqueness.
3. **Mutation:** As for Mutation, it only uses one tournament. During the Mutation process, one DNN from the selected ensemble is randomly replaced with a different pre-trained DNN from the set $\mathcal{M}$. Figure 3 depicts the Mutation process.

If the algorithm chooses Crossover when selecting the last ensemble for the next generation, we pick the first offspring produced.

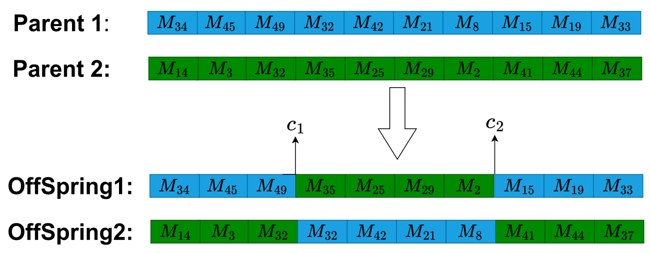

Parent 1:
M_{34} M_{45} M_{49} M_{32} M_{42} M_{21} M_8 M_{15} M_{19} M_{33}
Parent 2:
M_{14} M_3 M_{32} M_{35} M_{25} M_{29} M_2 M_{41} M_{44} M_{37}
c_1
c_2
OffSpring1:
M_{34} M_{45} M_{49} M_{35} M_{25} M_{29} M_2 M_{15} M_{19} M_{33}
OffSpring2:
M_{14} M_3 M_{32} M_{32} M_{42} M_{21} M_8 M_{41} M_{44} M_{37}

Fig. 2: Illustration of Crossover within EAI when $c_1 = 3$ and $c_2 = 6$.

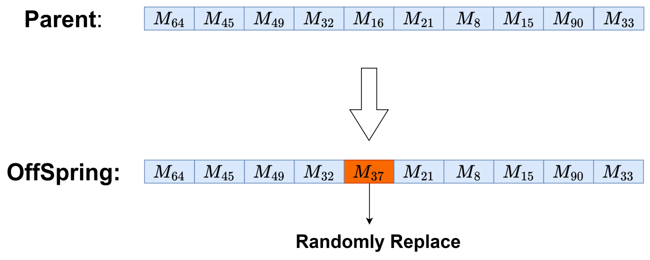

Parent:
M_{64} M_{45} M_{49} M_{32} M_{16} M_{21} M_8 M_{15} M_{90} M_{33}
OffSpring:
M_{64} M_{45} M_{49} M_{32} M_{37} M_{21} M_8 M_{15} M_{90} M_{33}
Randomly Replace

Fig. 3: Illustration of Mutation within EAI when taking a new model M_{37} from $\mathcal{M}$.

Time Complexity Analysis: Recall that $|\mathcal{M}|$ is the number of models, N_{gen} is the number of generations, N_{ens} is the population size, and N_{model} is the number of models per ensemble. The time complexity of *initialize_pop* is $O(N_{ens} \times N_{model})$. For each generation, the time complexity is $O(N_{model})$ for *Eval_&_Elite* while $O(N_{ens} \times N_{model})$ for *GenericOp*. Since the time complexity mainly comes from *GenericOp*, the overall time complexity of the EAI algorithm is given by $O(N_{gen} \times N_{ens} \times N_{model})$.

5 Experiment Result and Setting

5.1 Datasets and Leakage Models

ASCADf & ASCADr: Both ASCADf and ASCADr are part of the commonly used ASCADv1 dataset [3]. It comprises traces from first-order masked AES implementation on an 8-bit AVR microcontroller. Specifically, we target the third Sbox of the first round. ASCADf contains traces of the same fixed key for both profiling and attack. On the other hand, ASCADr considers the case where the profiling traces are generated from a random key setting while a fixed key is used to generate the attack traces. Both datasets contain 50000 profiling traces and 10000 attack traces. The traces comprise 700 sample points for ASCADf and 1400 sample points for ASCADr. We target the third byte of the first round sbox output, specifically $Sbox_{AES}(pt_3 \oplus k_3^*)$ where pt_3 represents the third plaintext byte and k_3^* denotes the third byte of the first round key.

Ascon*:* We use the publicly available datasets by [21] for Ascon. We will only investigate the first-order protected implementation of Ascon-128. The Ascon implementation is running on a ChipWhisperer Lite board on top of a STM32F4. The traces are collected using an 8-bit oscilloscope. There are 50000 profiling traces from a random key setting and 10000 attack traces collected from a fixed key setting. We target the first round of permutation proposed by [21, 18], where the sensitive variable is of the form:

$$y = k_1 \wedge (255 \oplus IV \oplus \mathfrak{n}_0) \oplus \mathfrak{n}_0 \oplus \mathfrak{n}_1,$$

where IV is the constant from the initialization value, while $\mathfrak{n}_0, \mathfrak{n}_1$ are 8-bits nonces values. Lastly, k_1 is the 8 bits key we are trying to recover. There are a total of 8 different bytes. We shall focus particularly on bytes 2 and 4, which have proven challenging for analysis as shown in [18] even with GSM ensemble. We denote Ascon2 and Ascon4 for byte 2 and byte 4 respectively.

AES_HD: The AES_HD dataset represents power leakage measurements from an unprotected AES hardware implementation running on an FPGA with a round-based architecture. Our analysis targets the side-channel leakage during the last round, specifically focusing on the Hamming Distance leakage model, i.e. $Sbox^{-1}_{AES}(ct_{15} \oplus k_{15}) \oplus ct_{11}$, where ct_i refers to the i^{th} ciphertext byte and k_{15} corresponds to the 15^{th} byte of the last round key. The dataset consists of 45000 profiling traces used for training and 3000 attack traces for evaluation.

Throughout this work, we use 1000 attack traces for each dataset.

Leakage Models: In this work, we consider three different leakage models.

- **Identity (ID):** The Identity leakage model assumes that the sensitive intermediate values, such as the AES S-box output $Sbox_{AES}(pt \oplus k)$, leak directly through side-channel emissions.
- **Hamming Weight (HW):** The HW leakage model corresponds to the Hamming weight of the sensitive variable. For example, for AES S-box output, we have $HW(Sbox_{AES}(pt \oplus k))$ as the leakage model. This model assumes that the amount of leakage is proportional to the number of 1-bits in the binary representation of the sensitive value.
- **Hamming Distance (HD):** The HD leakage model considers the side channel traces leaks the XOR of two sensitive variables.

For ASCADr and ASCADf datasets, we will investigate with ID and HW leakage models. As for Ascon datasets, we will present results for ID leakage model. On the other hand, we only use HD leakage model for AES_HD.

5.2 Hyperparameters used for DNN and EAI

DNN's Hyperparameter Search Space In this study, we consider the commonly used Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN) in our study. Specifically, we consider 50 random architectures for each: solely MLPs, solely CNNs, and a Diverse DNN consisting of 25 MLPs and 25 CNNs, with hyperparameter search spaces defined in Tables 1.

Table 1: Hyperparameter Search Space

MLP	
Parameter	**Values**
Layers	1 to 7 (in step of 1)
Neurons	$10, 20, 50, 100, 200, 300, 400, or 500$
Activation Functions	ReLU, SELU, ELU, or Tanh
Batch Size	$100 - 1000$ (in steps of 100)
Learning Rate	$1e-3, 5e-4, 1e-4, 5e-5, or 1e-5$
Optimizer	RMSprop or Adam
Weight Initialization	Random uniform, Xavier uniform, or He uniform
CNN	
Parameter	**Values**
Convolutional Layers	$1 - 4$ (in step of 1)
Initial Filters	$4, 8, 12, 16$
Initial Kernel Size	$26 - 52$ (in step of 2)
Pooling Type	Max pooling, Average pooling
Pooling Size	$2, 4, 6, 8, 10$
Padding	$0, 4, 8, 12, 16$
Fully Connected Layers Neurons	$1 - 7$ (in step of 1) $10, 20, 50, 100, 200, 300, 400, 500$
Activation Functions	ReLU, SELU, ELU, Tanh
Batch Size	$100 - 1000$ (in steps of 100)
Learning Rate	$1e-3, 5e-4, 1e-4, 5e-5, 1e-5$
Optimizer	RMSprop, Adam
Weight Initialization	Random uniform, Xavier uniform, He uniform

Hyperparameters for EAI The hyperparameters for EAI are shown in Table 2. All the hyperparameters listed are consistent across all datasets, except for the number of generations. This is because for both Ascon2 and Ascon4 it require more time to run, hence we decrease the number of generations.

Table 2: Hyperparameters for EAI across all Datasets

Configuration for Each Dataset and Hyperparameters of EAI					
	ASCADf	**ASCADr**	Ascon**2**	Ascon**4**	**AES_HD**
Number of Generations, N_{gen}	50	50	5	5	50
Crossover Probability $Pr_{\mathbf{cross}}$			0.9		
Mutation Probability, $Pr_{\mathbf{mut}}$			0.1		
Models per Ensemble, N_{model}			10		
Number of Ensembles per Generation, N_{ens}			30		
Models per Tournament, ℓ			3		

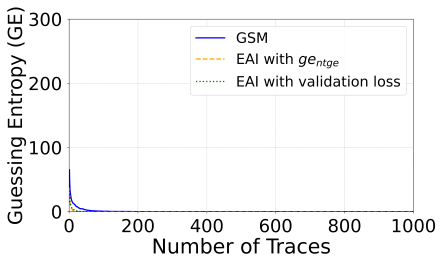
5.3 Experimental Results on Publicly Available Datasets

ASCADf. As shown in Table 3 and Figure 4, we analyze the performance of different algorithms on the ASCADf dataset. The results demonstrate that our EAI with the ge_{+ntge} variant significantly outperforms both EAI with val_loss and GSM in terms of number of traces required to recover the key. Across different models, the ID leakage model consistently performs more effectively than the HW leakage model. Notably, the MLP model with ID leakage model achieves the best performance ($NTGE = 29$). We observed significant differences in performance between the use of fitness functions when applying EAI. Our results show that EAI using val_loss is ineffective in recovering the secret key in certain cases. Conversely, EAI with ge_{+ntge} consistently recovers the secret key across every scenario, surpassing GSM's performance. However, it requires a longer

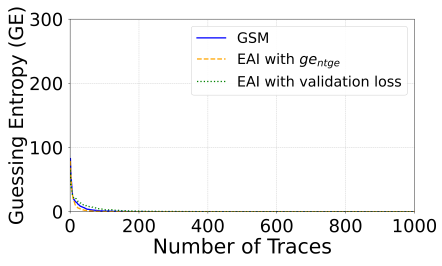
Table 3: $NTGE$ for different algorithms and leakage models in ASCADf Dataset

	MLP		CNN		Diverse DNN	
	HW	ID	HW	ID	HW	ID
GSM	785	365	825	385	817	329
EAI with ge_{+ntge}	615	29	727	140	590	58
EAI with val_loss	($GE = 19$)	98	($GE = 3$)	615	($GE = 66$)	113

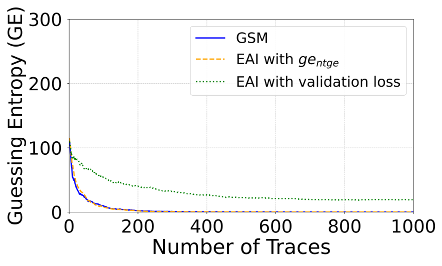
time when executing EAI compared to GSM, as shown in Figure 5. This is as



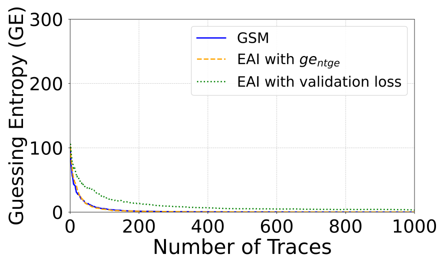
(a) MLP (ID).



(b) CNN (ID).



(c) MLP (HW)



(d) CNN (HW)

Fig. 4: Guessing entropy for ASCADf .

expected since GSM only needs to select the top few models and execute the attack phase, but EAI will need to compute the fitness function for different ensembles over many different generations.

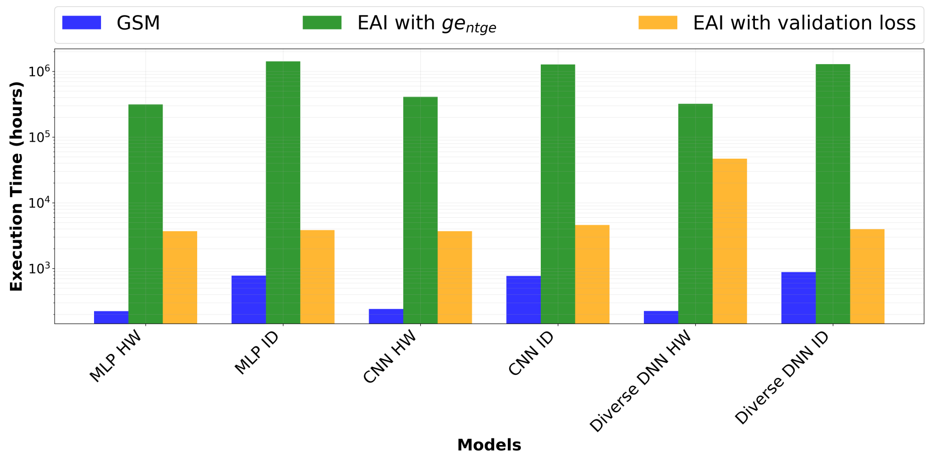
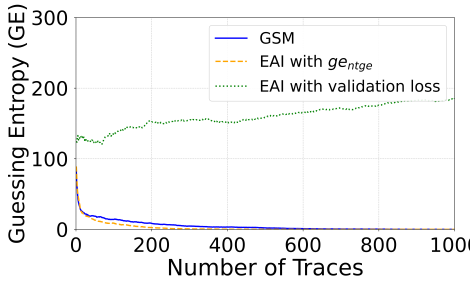


Fig. 5: Execution time comparisons for different algorithms and models in the ASCADf Dataset.

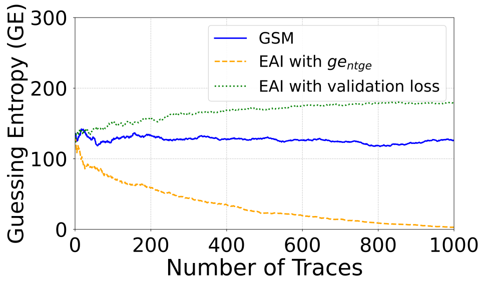
ASCADr. Similar to ASCADf, Table 4 and Figure 6 demonstrates that EAI with ge_{+ntge} maintains superior performance on the ASCADr dataset, outperforming GSM for all scenarios. In contrast to the ASCADf dataset, the HW leakage model outperforms the ID leakage model. The EAI with val_loss shows particularly poor performance, failing to recover the encryption key across all model and leakage model combinations.

Table 4: $NTGE$ for different algorithms and leakage models in ASCADr Dataset

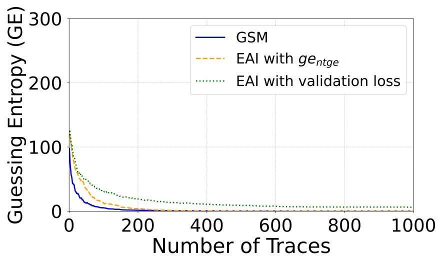
	MLP		CNN		Diverse DNN	
	HW	ID	HW	ID	HW	ID
GSM	548	976	810	$(GE = 126)$	445	$(GE = 14)$
EAI with ge_{+ntge}	296	523	619	$(GE = 3)$	257	923
EAI with val_loss	$(GE = 1)$	$(GE = 185)$	$(GE = 6)$	$(GE = 178)$	$(GE = 7)$	$(GE = 178)$



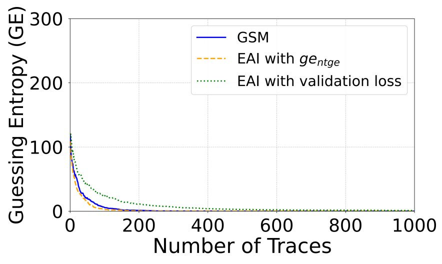
(a) MLP (ID).



(b) CNN (ID).



(c) CNN (HW)



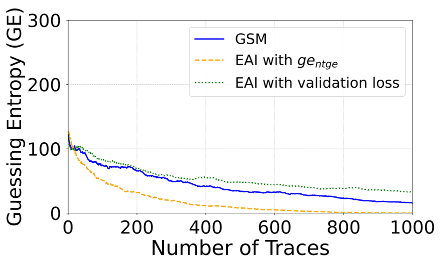
(d) MLP (HW)

Fig. 6: Guessing entropy for ASCADr.

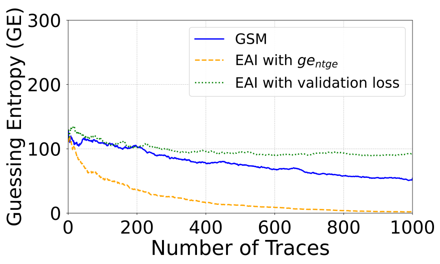
Ascon2 Our research investigated both ID and HW leakage models. Due to the unsuccessful recovery of the secret key using the HW leakage model across all tested methods, we will only present findings from the ID leakage model just like in [18]. We see that with GSM and EAI with val_loss, we are unable to successfully recover the secret key with $GE \geq 15$. However, using EAI with ge_{+ntge}, we manage to obtain the secret key with $NTGE = 977$ when building an ensemble of MLPs. Furthermore, we attain $GE \leq 3$ for having CNN ensembles and under the Diverse DNN setting. This overall shows the effectiveness of EAI with ge_{+ntge}.

Table 5: $NTGE$ for different algorithms and leakage models in the Ascon2 Dataset

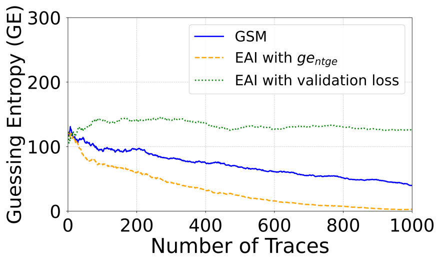
Algorithm	MLP	CNN	Diverse DNN
GSM	$(GE = 15)$	$(GE = 54)$	$(GE = 40)$
EAI with ge_{+ntge}	977	$(GE = 1)$	$(GE = 3)$
EAI with val_loss	$(GE = 34)$	$(GE = 93)$	$(GE = 126)$



(a) MLP model



(b) CNN model



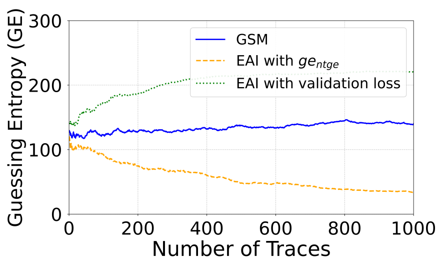
(c) Diverse DNN model

Fig. 7: Guessing entropy evaluation for different model architectures in Ascon2 Dataset.

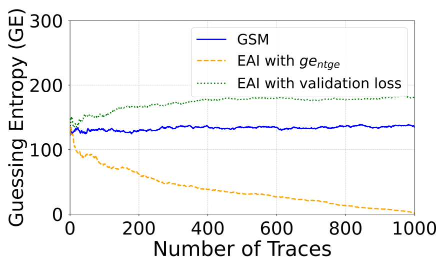
Ascon4 Our findings show that Ascon4 yields similar results to Ascon2. Both GSM and EAI with val_loss attain high GE values, suggesting that these methods are unable to recover the secret key. However, under the Diverse DNN setting, EAI with ge_{+ntge} recovers the secret key with 914 attack traces. Furthermore, when building a CNN-only ensemble, EAI with ge_{+ntge} attain $GE = 1$. These instances suggest the effectiveness of EAI with ge_{+ntge} in obtaining well-performing ensembles for successful key recovery in SCA.

Table 6: $NTGE$ for different algorithms and leakage models in the Ascon4 Dataset.

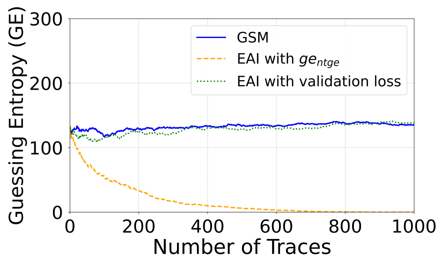
Algorithm	MLP	CNN	Diverse DNN
GSM	$(GE = 140)$	$(GE = 135)$	$(GE = 136)$
EAI with ge_{+ntge}	$(GE = 34)$	$(GE = 1)$	914
EAI with val_loss	$(GE = 220)$	$(GE = 179)$	$(GE = 138)$



(a) MLP model



(b) CNN model



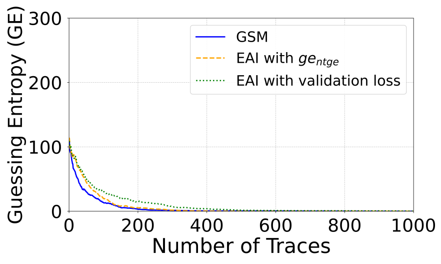
(c) Diverse DNN model

Fig. 8: Guessing entropy evaluation for different model architectures in Ascon4 Dataset.

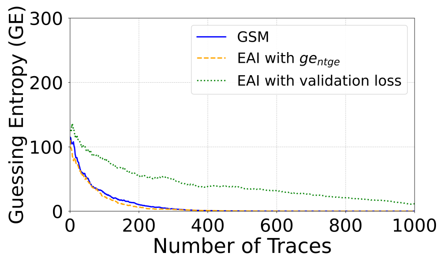
AES_HD Lastly, we also investigate the effectiveness of our methodology on hardware traces. Here, we only consider the HD leakage model as stated in Section 5.1. EAI with ge_{+ntge} we manage to recover the secret for all the scenarios tested and outperform both GSM and EAI with val_loss in terms of the $NTGE$ needed for successful key recovery. This highlights once again the using ge_{+ntge} as the fitness function is crucial in finding well-performing neural networks.

Table 7: Number of traces for different algorithms and leakage models in the AES_HD Dataset

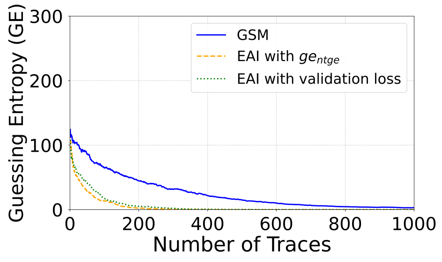
Algorithm	MLP	CNN	Diverse DNN
GSM	925	840	$(GE = 3)$
EAI with ge_{+ntge}	569	700	456
EAI with val_loss	998	$(GE = 12)$	662



(a) MLP model



(b) CNN model



(c) Diverse DNN model

Fig. 9: Guessing entropy evaluation for different model architectures in AES_HD Dataset.

6 Discussion and Future Work

In this work, we propose a novel methodology called EAI for identifying well-performing ensembles from pre-trained neural networks. EAI significantly reduces the number of traces required for successful key recovery compared to traditional GSM when paired with an effective fitness function. We investigated two fitness functions, ge_{+ntge} and val_loss. We observe that with ge_{+ntge} as the fitness function, EAI consistently outperforms other methods across five different datasets. Furthermore, it managed to recover the secret key across all the datasets tested, showing the efficiency of the EAI with ge_{+ntge} as an ensemble selection algorithm. However, with val_loss, it can be observed that it did not find a well-performing ensemble, and in many instances, GSM has much better performances. This suggests that choosing the right fitness function is very important, therefore, we recommend to use ge_{+ntge} as the primary fitness function when employing EAI. However, EAI demands more computational resources, as evidenced by the longer execution times. Therefore, if resources are limited, GSM remains a preferable alternative.

We hope that this work opens up a new dimension of research into finding the optimal ensemble among pre-trained DNNs. Several directions offer potential for exploration. Firstly, as mentioned above, EAI requires lengthy execution times. One possible direction is to enhance this approach by leveraging multi-fidelity optimization methods [7]. Furthermore, ge_{+ntge} fitness function necessitates knowledge of the secret key for deployment. This limitation suggests that its application is restricted to white-box settings during evaluation, specifically for identifying worst-case scenarios. Developing a novel fitness function for EAI that does not require secret key knowledge and achieves comparable performance would be an intriguing area of study.

References

1. Acharya, R.Y., Ganji, F., Forte, D.: Information Theory-based Evolution of Neural Networks for Side-channel Analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(1), 401–437 (2023). https://doi.org/10.46586/TCHES.V2023.I1.401-437, https://doi.org/10.46586/tches.v2023.i1.401-437
2. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side-Channel(s). In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 29–45. Springer (2002). https://doi.org/10.1007/3-540-36400-5"˙4, https://doi.org/10.1007/3-540-36400-5_4
3. 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). https://doi.org/10.1007/s13389-019-00220-8, https://doi.org/10.1007/s13389-019-00220-8
4. Bishop, C.M.: Pattern recognition and machine learning, 5th Edition. Information science and statistics, Springer (2007), https://www.worldcat.org/oclc/71008143

5. Daemen, J., Rijmen, V.: The Design of Rijndael - The Advanced Encryption Standard (AES), Second Edition. Information Security and Cryptography, Springer (2020). https://doi.org/10.1007/978-3-662-60769-5, https://doi.org/10.1007/978-3-662-60769-5
6. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight Authenticated Encryption and Hashing. J. Cryptol. **34**(3), 33 (2021). https://doi.org/10.1007/S00145-021-09398-9, https://doi.org/10.1007/s00145-021-09398-9
7. Eng, T.Y.H., Bhasin, S., Weissbart, L.: Train Wisely: Multifidelity Bayesian Optimization Hyperparameter Tuningin Side-Channel Analysis. IACR Cryptol. ePrint Arch. p. 170 (2024), https://eprint.iacr.org/2024/170
8. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1“˙25, https://doi.org/10.1007/3-540-48405-1_25
9. Levesque, J., Gagné, C., Sabourin, R.: Bayesian Hyperparameter Optimization for Ensemble Learning. In: Ihler, A., Janzing, D. (eds.) Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence, UAI 2016, June 25-29, 2016, New York City, NY, USA. AUAI Press (2016), http://auai.org/uai2016/proceedings/papers/73.pdf
10. Lin, W., Lee, W., Hong, T.: Adapting crossover and mutation rates in genetic algorithms. J. Inf. Sci. Eng. **19**(5), 889–903 (2003), http://www.iis.sinica.edu.tw/page/jise/2003/200309_10.html
11. Llavata, D., Cagli, E., Eyraud, R., Grosso, V., Bossuet, L.: Deep Stacking Ensemble Learning Applied to Profiling Side-Channel Attacks. In: Bhasin, S., Roche, T. (eds.) Smart Card Research and Advanced Applications - 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14-16, 2023, Revised Selected Papers. Lecture Notes in Computer Science, vol. 14530, pp. 235–255. Springer (2023). https://doi.org/10.1007/978-3-031-54409-5“˙12, https://doi.org/10.1007/978-3-031-54409-5_12
12. Lueth, K.L.: State of the IoT 2020: 12 billion IoT connections, surpassing non-IoT for the first time (2021)
13. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10076, pp. 3–26. Springer (2016). https://doi.org/10.1007/978-3-319-49445-6“˙1, https://doi.org/10.1007/978-3-319-49445-6_1
14. Masure, L., Dumas, C., Prouff, E.: A Comprehensive Study of Deep Learning for Side-Channel Analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 348–375 (2020). https://doi.org/10.13154/TCHES.V2020.I1.348-375, https://doi.org/10.13154/tches.v2020.i1.348-375
15. Mendoza, H., Klein, A., Feurer, M., Springenberg, J.T., Hutter, F.: Towards Automatically-Tuned Neural Networks. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) Proceedings of the 2016 Workshop on Automatic Machine Learning, AutoML 2016, co-located with 33rd International Conference on Machine Learning (ICML 2016), New York City, NY, USA, June 24, 2016. JMLR

Workshop and Conference Proceedings, vol. 64, pp. 58–65. JMLR.org (2016), http://proceedings.mlr.press/v64/mendoza_towards_2016.html

16. Ortiz, M., Scheidegger, F., Casas, M., Malossi, A.C.I., Ayguadé, E.: Generating Efficient DNN-Ensembles with Evolutionary Computation. CoRR **abs/2009.08698** (2020), https://arxiv.org/abs/2009.08698
17. Perin, G., Chmielewski, L., Picek, S.: Strength in Numbers: Improving Generalization with Ensembles in Machine Learning-based Profiled Side-channel Analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(4), 337–364 (2020). https://doi.org/10.13154/TCHES.V2020.I4.337-364, https://doi.org/10.13154/tches.v2020.i4.337-364
18. Rezaeezade, A., Basurto-Becerra, A., Weissbart, L., Perin, G.: One for All, All for Ascon: Ensemble-Based Deep Learning Side-Channel Analysis. In: Andreoni, M. (ed.) Applied Cryptography and Network Security Workshops - ACNS 2024 Satellite Workshops, AIBlock, AIHWS, AIoTS, SCI, AAC, SiMLA, LLE, and CIMSS, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14586, pp. 139–157. Springer (2024). https://doi.org/10.1007/978-3-031-61486-6“˙9, https://doi.org/10.1007/978-3-031-61486-6_9
19. Savu, I., Krcek, M., Perin, G., Wu, L., Picek, S.: The Need for MORE: Unsupervised Side-Channel Analysis with Single Network Training and Multi-output Regression. In: Wacquez, R., Homma, N. (eds.) Constructive Side-Channel Analysis and Secure Design - 15th International Workshop, COSADE 2024, Gardanne, France, April 9-10, 2024, Proceedings. Lecture Notes in Computer Science, vol. 14595, pp. 113–132. Springer (2024). https://doi.org/10.1007/978-3-031-57543-3“˙7, https://doi.org/10.1007/978-3-031-57543-3_7
20. Wang, H., Dubrova, E.: Tandem deep learning side-channel attack on FPGA implementation of AES. SN Comput. Sci. **2**(5), 373 (2021). https://doi.org/10.1007/S42979-021-00755-W, https://doi.org/10.1007/s42979-021-00755-w
21. Weissbart, L., Picek, S.: Lightweight but Not Easy: Side-channel Analysis of the Ascon Authenticated Cipher on a 32-bit Microcontroller. IACR Cryptol. ePrint Arch. p. 1598 (2023), https://eprint.iacr.org/2023/1598
22. Wu, L., Rezaeezade, A., Alipour, A., Perin, G., Picek, S.: Leakage Model-flexible Deep Learning-based Side-channel Analysis. IACR Commun. Cryptol. **1**(3), 41 (2024). https://doi.org/10.62056/AY4C3TXOL7, https://doi.org/10.62056/ay4c3txol7
23. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Efficiency through Diversity in Ensemble Models applied to Side-Channel Attacks - A Case Study on Public-Key Algorithms -. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 60–96 (2021). https://doi.org/10.46586/TCHES.V2021.I3.60-96, https://doi.org/10.46586/tches.v2021.i3.60-96
24. Zaidi, S., Zela, A., Elsken, T., Holmes, C.C., Hutter, F., Teh, Y.W.: Neural Ensemble Search for Uncertainty Estimation and Dataset Shift. In: Ranzato, M., Beygelzimer, A., Dauphin, Y.N., Liang, P., Vaughan, J.W. (eds.) Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. pp. 7898–7911 (2021), https://proceedings.neurips.cc/paper/2021/hash/41a6fd31aa2e75c3c6d427db3d17ea80-Abstract.html

25. Zhou, Z., Wu, J., Jiang, Y., Chen, S.: Genetic Algorithm based Selective Neural Network Ensemble. In: Nebel, B. (ed.) Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001. pp. 797–802. Morgan Kaufmann (2001)