One for All, All for Ascon: Ensemble-Based Deep Learning Side-Channel Analysis

Azade Rezaeezade[1](✉), Abraham Basurto-Becerra[2], Léo Weissbart[2], and Guilherme Perin[3]

[1] Delft University of Technology, Delft, The Netherlands
a.rezaeezade-1@tudelft.nl
[2] Radboud University, Nijmegen, The Netherlands
[3] Leiden University, Leiden, The Netherlands

Abstract. In recent years, deep learning-based side-channel analysis (DLSCA) has become an active research topic within the side-channel analysis community. The well-known challenge of hyperparameter tuning in DLSCA encouraged the community to use methods that reduce the effort required to identify an optimal model. One of the successful methods is ensemble learning. While ensemble methods have demonstrated their effectiveness in DLSCA, particularly with AES-based datasets, their efficacy in analyzing symmetric-key cryptographic primitives with different operational mechanics remains unexplored.

Ascon was recently announced as the winner of the NIST lightweight cryptography competition. This will lead to broader use of Ascon and a crucial requirement for thorough side-channel analysis of its implementations. With these two considerations in view, we utilize an ensemble of deep neural networks to attack two implementations of Ascon. Using an ensemble of five multilayer perceptrons or convolutional neural networks, we could find the secret key for the Ascon-protected implementation with less than 3 000 traces. To the best of our knowledge, this is the best currently known result. We can also identify the correct key with less than 100 traces for the unprotected implementation of Ascon, which is on par with the state-of-the-art results.

Keywords: Side-channel Analysis · Deep Learning · Ensemble · Ascon

1 Introduction

Introducing the Ascon family as a new standard for authenticated encryption [NIS23] has raised interest in the available implementations that could be used in embedded devices and their security. Then, evaluating the physical security of cryptographic implementations against side-channel analysis (SCA) is a crucial step in developing secure embedded devices.

SCA is an implementation attack that exploits measurements of unintended physical leakages of sensitive information from a device through side channels

M. Andreoni (Ed.): ACNS 2024 Workshops, LNCS 14586, pp. 139–157, 2024.
https://doi.org/10.1007/978-3-031-61486-6_9

such as power consumption, electromagnetic emission, or timing. The analysis methods for SCA have evolved since the first works on the subject, introducing simple power analysis (SPA) and differential power analysis (DPA) [KJJ99]. Today, SCA methods are numerous and are classified into two main categories: profiled and non-profiled attacks.

Non-profiled attacks, including Differential Power Analysis (DPA) [KJJ99], Correlation Power Analysis (CPA) [BCO04], and Mutual Information Analysis (MIA) [GBTP08], are techniques where an attacker uses a large set of measurements and statistical tools to exploit the leakage of secret information.

Profiled attacks include techniques like template attack [CRR02], stochastic attacks [SLP05], and machine learning-based attacks [MPP16], where the attacker mounts the attack in two phases. In the first phase, known as profiling/template building, the attacker needs access to a clone device to build profiles. In the second phase, which is known as the attack/template matching phase, the attacker matches the built profiles to recover the secret data.

Deep learning-based side-channel analysis (DLSCA) has become a research hot spot from 2016 [MPP16]. The studies have reported many advantages for this approach, including the robustness to different masking and hiding countermeasures [MBC+20,MPP16,CDP17] and removing the need for pre-processing [CDP17,KPH+19]. Despite all these advantages, it is frequently emphasized that the principal challenge in DLSCA is the selection of a neural network model that is tailored to the specific nuances of the problem at hand [PPM+23].

To overcome that challenge, different works have used various approaches, including hyperparameter tuning [WPP22,RWPP21,AGF21], regularization techniques [RB22], or designing a methodology for model selection [ZBHV20]. An interesting and effective strategy proposed to circumvent (or, at least alleviate) the challenge of finding an optimal model is the utilization of ensemble techniques [PCP20], where multiple sub-optimal neural network models combine to enhance the overall performance of DLSCA. While the results presented in that work demonstrate the utility of the ensemble method in enhancing attack performance, a gap in generalization across various cryptographic primitive implementations is evident. Until recently, the publicly available datasets for symmetric-key cryptography were centered around the AES primitive, as discussed in [PPM+23].

Consequently, the effectiveness of many proposals, including the ensemble, has been validated using only AES-based datasets. This raises a question about the efficiency of diverse proposals in DLSCA for AES when considering other cryptographic primitives. Ascon, the NIST lightweight cryptography competition winner, currently being standardized for broad public use, is an ideal subject for such investigation. The known vulnerability of the Ascon encryption mode of operation to side-channel analysis and the availability of its datasets make it an ideal candidate for this research. This shift toward considering Ascon as a benchmark in DLSCA research not only aligns with its growing use but also provides a broader perspective on the adaptability and efficiency of DLSCA across different cryptographic primitives.

In this research, we attack two software implementations (one unprotected and one protected) of the Ascon primitive using the ensemble method. The key contributions of our work are:

- **Extension beyond AES-based experiments:** Previous research demonstrated the effectiveness of the ensemble method in DLSCA, only focusing on the AES primitive. Our study extends this, showing that the ensemble method is also effective for other cryptographic primitives. We particularly highlight improved attack performance on protected Ascon implementations, where the challenge is more significant, and the attacks have not been very successful so far. The successful results with ensembles give hope that other DLSCA proposals aiming at AES may generalize for other cryptographic primitives.
- **Exploring Ascon in the context of side-channel analysis:** With Ascon being standardized by NIST and its usage expected to increase, there is a pressing need for comprehensive side-channel analysis of its implementations. In this research, we successfully attack both protected and unprotected Ascon implementations using the ensemble method. Our attack using the ensemble method outperforms the state-of-the-art results, emphasizing the necessity of designing and implementing adequate countermeasures for vulnerable operations in Ascon's implementation.

2 Preliminaries

2.1 Ascon Primitive

Ascon is a family of cryptographic algorithms designed to provide secure encryption and authentication in resource-constrained environments. This family of cryptography primitives is based on sponge construction [BDPVA12] and was selected by NIST in February 2023 to be standardized [NIS23] for lightweight applications. Ascon is an authenticated encryption algorithm that includes associated data, meaning it not only encrypts a message to maintain its confidentiality but also attaches a tag to the encrypted message and associated data to ensure integrity. The algorithm can take four inputs: plaintext P, associated data A, nonce M, and a key k. It outputs the authenticated ciphertext C and an authentication tag T. The algorithm includes four operation phases: *initialization, associated data process, plaintext process (ciphertext process in decryption), and finalization.* Figure 1 shows these four phases of Ascon.

In Ascon-128, the input of the initialization phase is a 320-bit initial state ($IV||k||M$ in Fig. 1 consisting of the 64-bit constant IV, the 128-bit key k, and the 128-bit fresh nonce M) in the form of five 64-bit words x_0 to x_4. This five-word state updates through the algorithm phases and is used as the secret state (or the sponge state) for encryption (decryption) and tag generation. The initialization phase includes twelve (same) permutation functions (shown as p^a in Fig. 1) that update the initial state. The permutation function consists of three parts: 1) the addition of the round constants, 2) a non-linear five-bit S-box (substitution layer), and 3) a linear diffusion layer.

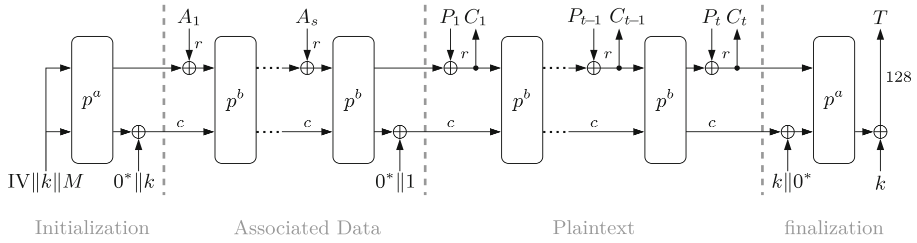


Fig. 1. ASCON's mode of operation and S-box.

During a data exchange, data like headers and metadata must remain in plaintext, but maintaining integrity is crucial for this data. The optional associated data processing phase maintains its integrity. In this phase, when an associated data block (A_i) is received, its first $r = 64$ bits is XORed to the first $r = 64$ bits of the sponge state, then the whole sponge state is permuted $b = 6$ times (p^b in Fig. 1). The associated data processing phase updates the sponge state using the associated data blocks. Then, the updated sponge state attains the plaintext process phase. The plaintext process phase XORs the 64-bit plaintext block P_i with the first $r = 64$ bits of the sponge state to produce ciphertext block C_i. Then, the whole state is transformed by the permutations $p^b, b = 6$ to update the sponge state for the next plaintext block. The finalization phase XORes the key with the sponge state and transforms the results with $p^a, a = 12$, to provide the 128-bit authentication tag T. For more details about different parts of the Ascon primitive, refer to [DEMS16].

2.2 Ensembles

Ensemble techniques combine multiple predictors (machine learning models or deep neural networks) to reduce generalization error [GBC16]. The predictor can be a simple machine learning method like a decision tree or an advanced one like a deep neural network. Ensemble techniques work because different predictors may capture various aspects of the data, and by combining them, one can often achieve better performance than every single model contributing to the ensemble. There are different techniques for ensemble predictors, including voting, bagging, boosting, and stacking. These techniques are different in how they create and combine the models. For example, bagging involves training multiple models independently and averaging their predictions. This method is useful for reducing variance and overfitting. Boosting, on the other hand, trains models sequentially, with each new model focusing on the data points that previous models miss-classified, aiming to improve the predictive performance iteratively. In deep learning, ensemble methods typically involve different architectures or configurations of neural networks, such as varying numbers of layers, nodes, or activation functions [PCP20].

The ensemble method has also been used in the domain of SCA (see Sect. 3). Our ensembling approach is aligned with the one Perin et al. used in [PCP20]. Their technique is specialized bagging (bootstrap aggregating), where the models in the ensemble are selected through a random search, and each model is trained on the entire dataset (the "bag" used for training every single model is equal to the whole training dataset). The models are trained independently. We diverge from common practices like majority voting or averaging to integrate the models' output. That is because, in the context of DLSCA, models often provide uncertain predictions about the class of a trace[1]. The accuracy of models in an attack phase is marginally better (or sometimes even worse) than a random guess [PCP20]. Consequently, techniques like majority voting or averaging are ineffective in enhancing attack performance. As an alternative, we exploit the small bias of the model outputs toward the correct class by utilizing the guessing entropy metric. Section 2.3 provides more information about this side-channel metric.

2.3 Deep Learning-Based Side-Channel Analysis

As mentioned in Sect. 1, profiling SCA has a phase of template building where the attacker gathers many traces from a clone device and builds the templates using those traces and a phase of template matching where the attacker matches the traces from the device under attack with the templates to find out to which template the traces belong. The output of the template matching phase is a matrix of probabilities showing with what probability each trace belongs to each template. Looking into the procedure of profiling SCA, deep learning (machine learning) classification is a natural choice for profiling attacks. Template building is equivalent to learning the classes[2] using many examples during the training phase and template matching is equivalent to classification on not previously seen data in the test phase. Using softmax as the last layer of a deep neural network, we can obtain the probability matrix as before. Using that matrix, we can calculate the common metrics like guessing entropy and the required number of attack traces to measure the performance of profiling SCA.

Guessing Entropy. In the attack phase of a profiled attack, guessing entropy [Mas94, KB07] is the average number of guesses that must be made before finding the correct key. The output of a profiling side-channel attack with Q attack traces is a probability vector of key hypotheses $\mathbf{h} = [h_1, h_2, \ldots, h_{|\mathbb{K}|}]$ in decreasing order (i.e., h_1 has the highest probability and $h_{|\mathbb{K}|}$ the lowest probability of being the correct key), where $|\mathbb{K}|$ is the key space. The information about each key candidate is calculated using $h_i = \Sigma_{j=1}^{Q} \log p(x_j, y)$, where $p(x_j, y)$ is the probability that the trace x_i belongs to class y. Thus, guessing entropy is

[1] Trace is the whole or part of the measurement given as input to the neural network.
[2] Usually, classes are all the possible intermediate values specified by the selected leakage model, something we know like the plaintext or public nonce, and the secret key.

the average position of the correct key, k^*, in $\mathbf{h}$. Equation (1) shows the formal definition of guessing entropy:

$$GE = E(rank_{k^*}(\mathbf{h})), \tag{1}$$

where $rank_{k^*}(\mathbf{h})$ denotes the position of the correct key k^* in the probability vector $\mathbf{h}$, and E is the expectation operator. An attack is considered a successful attack if $GE = 1$.

For the ensemble of models, we accumulate all individual models' output probabilities class-wise for each trace, and then the rest of the procedure is the same as above.

Required Number of Attack Traces. The required number of attack traces is the minimum number of traces needed to always place the correct key in the first position of $\mathbf{h}$.

3 Related Work

Two research streams are closely related to the work presented in this paper. The first stream employs ensemble methods to improve SCA. The second stream targets implementations of Ascon. The following section briefly summarizes what has been done so far.

Ensemble methods were used in the SCA domain as soon as the community started to use machine learning. For example, Picek et al. in [PHJ+17] used Random Forest (which is an ensemble of decision trees), Rotation Forest, and MultiBoosting, all methods that use ensembling to improve the accuracy of predictions. In [LMBM13] and [MPP16], researchers again used Random Forest, which is one of the most popular options for machine learning-based SCA (next to Support Vector Machines). In a recent work, which we consider as the first and the most relevant from the DLSCA perspective, Perin et al. used bagging of multiple deep neural networks for attacking different AES implementation [PCP20]. One can find more details about [PCP20] in Sect. 2.2.

Several researchers have analyzed Ascon's side-channel resistance since its submission to the NIST lightweight competition. In [RAD20], a method named SCARL is used to recover the secret key of an Ascon Artix-7 FPGA implementation. SCARL uses LSTM autoencoders for dimensionality reduction of S-box operations power measurements and reinforcement learning for clustering key candidates. In [SS23], transfer learning is used from gate-level power simulation traces for an Ascon software implementation running on a custom-made RISC-V SoC to improve the performance of DLSCA using raw power traces as input (measured from a chip prototype of the same design). In [WP23], multi-task learning is used to evaluate the side-channel resistance of protected and unprotected Ascon datasets.

4 Experimental Setup

4.1 Attack Point and Leakage Model

During the initialization and finalization phases of the Ascon encryption mode of operation as described in Fig. 1, the secret key is directly involved with the nonce, a user input data. Since this phase processes something we know (the nonce) and the secret information we aim to obtain (the key), it can be the target of side-channel analysis. As is the usual case for symmetric cryptography, the best point to attack is the non-linear S-box output of first-round permutation. The S-box operation in Ascon takes 5-bit inputs from the sponge state and gives 5-bit outputs (Fig. 2). The S-box operates column-wise, i.e., the input includes only one bit from each word x_0 to x_4 at a time.

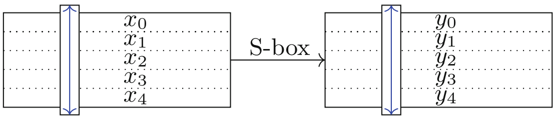


Fig. 2. Ascon column-wise S-box. S-box operation takes 5-bit input that includes only one bit from each word x_i and gives 5-bit output that contains one bit from each word y_i.

The Ascon S-box layer is a column-wise operation on the sponge state, applied to an individual column of the sponge state with 64 columns. One significant benefit of the Ascon S-box operation is its ability to be executed through XOR and AND operations on x_is [DEMS16]. Taking x_is as the inputs of the S-box layer and y_is as the outputs of this layer, the outputs of the non-linear S-box can be calculated as:

$$\begin{aligned}
y_0 &= x_0 + x_1 + x_2 + x_3 + x_1x_2 + x_0x_1 + x_1x_4 \\
y_1 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_1x_2 + x_1x_3 + x_2x_3 \\
y_2 &= x_1 + x_2 + x_4 + x_3x_4 + 1 \\
y_3 &= x_0 + x_1 + x_2 + x_3 + x_4 + x_0x_3 + x_0x_4 \\
y_4 &= x_1 + x_3 + x_4 + x_0x_1 + x_1x_4 = x_1(1 + x_0 + x_4) + x_3 + x_4.
\end{aligned} \tag{2}$$

In our approach to target the first round of permutation, we substitute the x_i values with their original values, which consist of the public constant, the key's high and low parts, and the nonce's high and low parts for x_0 to x_4, respectively. By examining Eq. (2), it is evident that in y_4, all components are public except for x_1 (the key's high part). This characteristic makes y_4 a practical intermediate value for side-channel attacks. Furthermore, it is feasible to deduce

x_1 using a divide-and-conquer[3] strategy and retrieve x_1 with 8-bit chunks. We use the following leakage model to recover the whole x_1 in eight attacks.

$$Y = k_i^{(1)} \& (255 \oplus IV_i \oplus M_i^{(1)}) \oplus M_i^{(1)} \oplus M_i^{(2)} \quad i = 0, ..., 7 \tag{3}$$

To obtain the remaining key bits (x_2), we use y_0 or y_1 (since they have non-linear terms including x_2) as the intermediate value. The other half of the key, x_1, can be taken as a known, and the recovered value can be replaced in the selected intermediate value.

Comparing with Attacking AES: As mentioned in Sect. 1, most of the published research in the DLSCA domain focused on AES-based datasets. Here, we mention some similarities and differences between the side-channel analysis of AES and Ascons to make our attack easier to understand.

The most highlighted difference between attacking Ascon and AES stems from these algorithms' structure differences. AES-128 has ten rounds of S-box, shift rows, mix columns (except for the last round), and add round key, while Ascon has a sponge-based construction with initialization and finalization phases. In the AES, the side-channel attacks target the first or last rounds because only these two rounds operate on a known value (plaintext in the first round and ciphertext in the last round) and the key. The structure of the Ascon algorithm is entirely different from AES, and only the initialization and finalization phases of this algorithm seem to be vulnerable against SCA. While the known values in the case of AES are plaintext or ciphertext, they are nonce and the tag in the case of Ascon. Consequently, there are fundamental differences in the S-box implementation of Ascon and AES. For instance, in the case of AES, S-box input is a combination of key and plaintext, while it is a combination of two bits of the key, two bits of the nonce, and one bit of the initial value in the case of the Ascon. However, in both cases, the S-box output seems the most effective point to attack as this point offers non-linearity. Also, divide-and-conquer is helpful in both cases.

4.2 Datasets

Considering the primary goal of this research is evaluating the effectiveness of the ensemble method for attacking Ascon primitive implementation, we use two publicly available datasets[4] introduced in [WP23]. The first dataset, referred to as Ascon-Unprotected from this point, is provided using the 32-bit optimized implementation of Ascon-128 v1.2. The other dataset, referred to as Ascon-Protected, is provided using a first-order protected implementation of Ascon. This implementation uses bit-interleaved and a specific masking countermeasure designed

[3] Divide-and-conquer strategy is a strategy to recover a long key by retrieving its smaller parts separately.

[4] https://zenodo.org/records/10229484.

to be efficient with the Ascon S-box[5]. The C implementations by the Ascon team are available in their GitHub repository [SDG+20]. Traces are collected using a ChipWhisperer Lite board and an 8-bit precision oscilloscope, coupled with the STM32F4 target running at a frequency of 7.37MHz. Both datasets contain traces of Ascon's first-round permutation during the initialization phase. We use 60 000 traces from the datasets; the first 50 000 traces are collected with random keys, used for training, and 10 000 traces are collected using a fixed key, used for the attack phase. Traces from the Ascon-Unprotected dataset have 772 samples, and from the Ascon-Protected dataset have 1 408 samples.

4.3 Neural Network Architectures

Multilayer Perceptron (MLP) is a class of feedforward artificial neural networks (ANNs) that consist of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each layer has one or more neurons connected to the neurons in the following layer through weighted edges. These neurons typically include a non-linear activation function, which allows the network to learn non-linear relationships between input and output data. The MLP learns to map input data to the correct output through an iterative optimization algorithm that adjusts the weights of the connections by minimizing a loss function. To minimize the loss function, ANNs mostly use gradient descent and back-propagation. MLPs are usually used for tasks like classification and regression.

Convolutional Neural Network (CNN) is another class of feedforward neural networks. CNNs have one or more convolutional layers followed by one or more fully connected layers. The convolutional layers apply a set of learnable filters (sometimes known as kernels) to the input. Filters are simply vectors (in the one-dimensional convolution) or matrices (in the two-dimensional convolution) of coefficients that update during a process similar to that of MLPs. An activation function is used after each convolution layer to add non-linearity. Then, there can be a max/average pooling layer that simply reduces the spatial size of the representation expanded by the filters. The network extracts the most important features using a combination of filters and pooling layers. CNNs are commonly used for classification, but their architecture can be adapted for regression by altering the final layer and the loss function.

4.4 Methodology

This section provides a set of experiments to inspect the ensemble method's effectiveness for DLSCA. To evaluate the efficacy of ensembles, we compare

[5] The masking technique used in the implementation is called Domain-Oriented Masking (DOM). It is a specialized technique used in cryptographic hardware, but it can also be applied in software implementation. It involves dividing a circuit into separate domains, each handling only one part of the data. This separation ensures that each domain only accesses a specific portion of the data, reducing the risk of data leakage through side-channel attacks. For more reading, one can look into the implementation of Ascon and [GIB18].

Table 1. Random search range for MLP and CNN hyperparameters.

Hyperparameter	Range
MLP's Architecture Hyperparameters	
Number of neurons	[30, 40, 50, 60, 70, 80, 90, 100, 120, 150]
Number of layers	[2, 8], step = 1
CNN's Architecture Hyperparameters	
Number of convolution layers	[2, 4], step = 1
Number of filters	[4, 20], step = 2
First layer's filter size	[4, 24], step = 2
$i^(th)$ layer filter size	$((i-1)filter\_size)^2$
Stride	[2, 10], step = 2
Pooling	"Average", "Max"
Pooling size	[4, 10], step = 2
Pooling stride	[4, 10], step = 2
Number of dense layers	[2, 4], step = 1
Number of neurons in dense layers	[50, 100, 150, 200, 300, 400, 500]
Common Learning Hyperparameters in MLP and CNN	
Learning rate	random.uniform(0.0001, 0.001)
Activation function	"relu", "tanh", "selu", "elu"
Optimizer	"Adam"
Weight initialization	"he_uniform"
Batch size	128
Epochs	10

the performance of the ensemble (a group of the neural network models) with the performance of *the best model*. To show that the results can be generalized, the experiments are conducted on two different datasets introduced in Sect. 4.2(Ascon-Unprotected and Ascon-Protected). To assure that the results are valid for various neural network topologies, we employed MLP and CNN models (Sect. 4.3) combined with the leakage model introduced in Sect. 4.1. Using two datasets, two neural network topologies, and a single leakage model gives us four combinations. In each combination, we aim to retrieve x_1, which is eight bytes of the sixteen-byte secret key. We use a divide-and-conquer strategy, i.e., we repeat the following steps eight times for each combination, and each time, we retrieve a sub-key of size one byte.

– **Acquiring best predictor:** In [WPP22], Wu et al. showed that random search can reach neural network models with top performance when one attacks relatively easy datasets. Considering this, we generate fifty different models using random search. The range of hyperparameters for the random search is given in Table 1). Then, we use guessing entropy to compare the performance of these fifty models and take the model with the best performance as *the best model*. It is worth mentioning that the selected model is not the best possible model. Other, more advanced hyperparameter tuning techniques (like reinforcement learning [RWPP21] or Bayesian optimization [WPP22]) or searching with a wider range of hyperparameters with more randomly generated models can lead to models with better performance. Hence, our experiments aim not to find an optimal model, and we only want to investigate whether the ensemble performs better than the single best model. We report the best model's guessing entropy (GE-Best) and its required number of attack traces (NT-Best) as the performance of the best model.

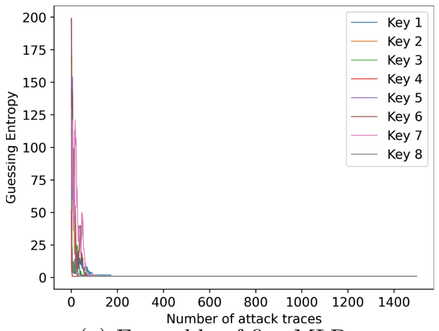
- **Acquiring ensemble:** To benefit from the ensemble method in general, a group of neural networks that individually can learn the problem and give predictions better than random guesses is needed. Since accuracy is not a good metric to judge the performance of a model in the side-channel analysis domain, we use guessing entropy to select models that perform the best among the randomly generated models. We take five[6] models with the smallest guessing entropy from the pool of randomly generated models to be used in the ensemble.
  The selected models do not necessarily need to find the key (reach $GE = 1$); they only need to reduce the GE to small values. In Sect. 2.3, we have seen that guessing entropy can be calculated by accumulating the probability that the neural network gives for each key hypothesis over the attack traces. We sum up the probabilities for each key hypothesis from all individual models in the ensemble and accumulate that over the attack traces. We report the final guessing entropy as the ensemble guessing entropy and refer to it as GE-Ensemble. The required number of attack traces can be calculated using the GE-Ensemble, which we call NT-Ensemble.
- **Comparing the best model and the ensemble performance:** The final step is comparing the performance of the best-acquired models (GE-Best and NT-Best) and the performance of the ensemble model (GE-Ensemble and NT-Ensemble). The selected group of predictors for the ensemble always includes the best predictor, and improved performance means that the ensemble method was effective.

5 Experimental Results

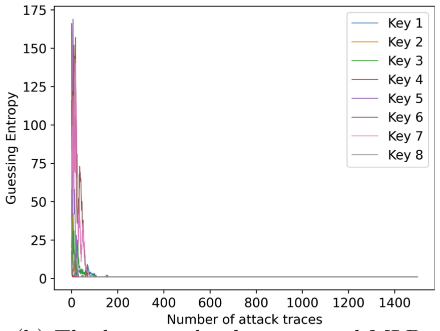
The objective of this section is to demonstrate the effectiveness of using the ensemble method for side-channel analysis of the Ascon primitive implementation. As noted in Sect. 1, the ensemble method has only been utilized to attack AES primitive implementations. Thus, its effectiveness for other primitives is unclear. We demonstrate that the ensemble technique enables successful attacks on both unprotected and protected implementations of the Ascon primitive. Our

[6] This number can vary depending on the problem, the complexity of individual models, and the desired balance between performance and complexity. In our experiments, we observed that five models could already offer good performance improvement.

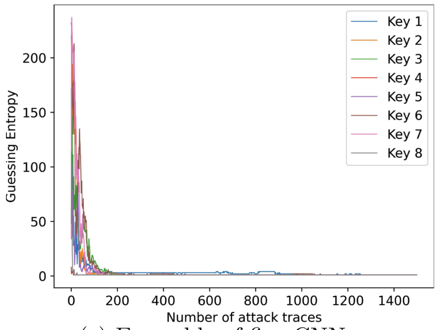
results confirm that the ensemble method is the most efficient technique to attack Ascon's protected implementation so far. Moreover, the success of the ensemble method attacking Ascon's unprotected implementation matches the success of a model selected through Bayesian optimization [WP23], again confirming that the ensemble of weaker learners can match the performance of a single model selected through an advanced hyperparameter tuning process.



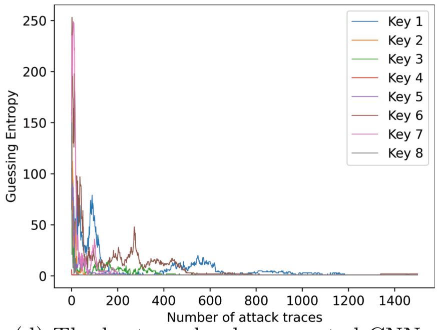
(a) Ensemble of five MLPs.



(b) The best randomly generated MLP.



(c) Ensemble of five CNNs.

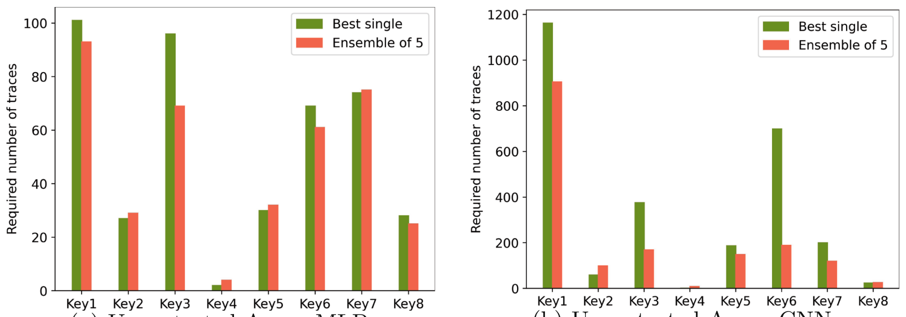


(d) The best randomly generated CNN.

Fig. 3. Guessing entropy for Ascon-Unprotected. On top, each color shows the evolution of guessing entropy for ensemble of five MLPs (a) and the best-found MLP (b) selected from a pool with fifty randomly generated MLP for each sub-key. On the bottom, each color shows the evolution of guessing entropy for ensemble of five CNNs (c) and the best-found CNN (d) among fifty randomly generated ones for each sub-key. (Color figure online)

5.1 Ascon-Unprotected

This section presents experimental results when attacking Ascon-Unprotected, an unprotected software implementation of Ascon, using the ensemble method, and compares the results with the performance of the best-found models with random search. Figure 3a shows the evolution of guessing entropy using the ensemble method for eight sub-keys. For each sub-key, the ensemble combines



(a) Unprotected-Ascon-MLP. (b) Unprotected-Ascon-CNN.

Fig. 4. The required number of attack traces with and without ensemble method in the Ascon-Unprotected dataset. On the left side, the required number of attack traces using the best MLP (green) and the ensemble of five MLPs (orange) is compared. On the right side, the required number of attack traces using the best CNN (green) and the ensemble of five CNNs (orange) is compared. (Color figure online)

the five best MLP neural networks selected among fifty randomly generated ones. In contrast, Fig. 3b depicts the guessing entropy evolution for the same sub-keys but employing the best-found MLP model. Observe that the reduction in guessing entropy is generally fast, though slightly slower for certain sub-keys. Figure 4a offers a clearer view of the impact of using the ensemble method. The effectiveness is most evident for key 3, where the required number of attack traces drops from 100 to 70. However, in half of the cases, using the ensemble method slightly increased the required number of attack traces. This observation is not unusual in scenarios where the problem tackled by the deep neural networks is relatively straightforward. For instance, a closer look into the performance of all randomly generated MLPs for sub-key 4 (key 4 in Fig. 3a) shows that more than 80% of the models could reveal the key with fewer than 10 traces, indicating that the attack is relatively easy for all the generated models. Consequently, finding an optimal model for this sub-key does not need much effort, and using the ensemble method does not offer additional performance benefits.

Turning to CNNs, Fig. 3c and Fig. 3d show the guessing entropy evolution for the same eight sub-keys, using an ensemble of the five best CNNs and the best-found CNN among fifty randomly generated ones for each sub-key. The ensemble's overall performance generally surpasses that of the best CNNs. However, when comparing MLP and CNN performances, it is apparent that either the best MLP or the ensemble of MLPs is typically more effective in key recovery. This observation has been mentioned in [RB22] as the "general ability of MLP models to find the key" and the "potential ability of CNN models to find the key". As discussed in [RB22], a limited number of MLP neural networks are more successful in reducing guessing entropy on average than the same number of CNN neural networks. Yet, with a more detailed architecture search, usually the best-found CNN outperforms the best-found MLP in key recovery.

A comparison of Fig. 4a and Fig. 4b reveals that the best CNN requires at least five times more traces than the best MLP to recover a key. This observation suggests that our search within the hyperparameter space detailed in Table 1 was not detailed enough, with no CNN model coming close to the optimal solution among the randomly generated models. However, the ensemble of CNN models could improve the attack performance compared to the best-found CNN, indicating that the ensemble is more helpful when dealing with a group of weak models rather than a group of powerful models. Comparing our results to the multi-task model on the Unprotected-Ascon dataset from previous work [WP23], we can see that the ensemble method with CNNs is on par with the multi-task model, recovering the key with around 1 000 traces. However, the ensemble method with MLPs can recover the key with about 100 traces, which is significantly better than the multi-task model where for some sub-keys more than 1 000 traces were needed.

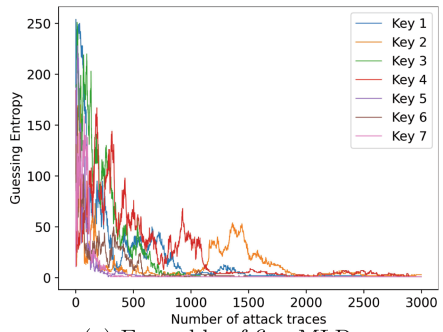
5.2 Ascon-Protected

Next, we outline experimental results when attacking Ascon-Protected, a first-order protected software implementation of Ascon. The experiments in this section present a more challenging test for the efficacy of the ensemble method, particularly because the considered dataset is not easy to break [WP23]. Figure 5a illustrates the evolution of guessing entropy using an ensemble of five MLP neural networks. Figure 5b shows the same attack using the best-found MLP for each sub-key. Comparing these two figures shows that the reduction in guessing entropy using the ensemble method is much faster than the best-found MLP. The superior performance of the ensemble method is highlighted when analyzing the required number of attack traces. Figure 6a compares the required number of attack traces for both the ensembles and the best MLP. Clearly, the best MLP could only reveal sub-key 3 (key 3 in Fig. 6a) and sub-key 8 (key 8 in Fig. 6a), whereas the ensemble of MLPs successfully recovered all the sub-keys except sub-key 2 (key 2 in Fig. 6a)[7].

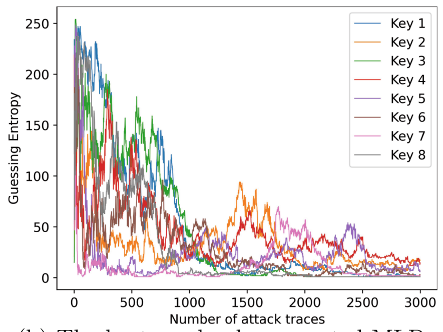
Similar observations apply to the CNN models, as shown in Fig. 5c and Fig. 5d. The ensemble method allows for the reduction of all sub-keys guessing entropy to $GE = 1$, except for sub-key 4 (key 4 in Fig. 5c), while none of the best-found CNNs in the pools of randomly generated CNN models could reduce GE to one. The stark contrast is further evident in Fig. 6b.

Considering the results from the ensemble learning on the Ascon-Protected and Ascon-Unprotected datasets, we can conclude that the ensemble method is significantly more effective for challenging datasets, where finding optimal models is more difficult. This conclusion can be supported by the similar performance

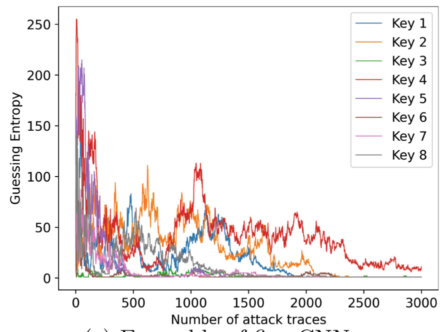
[7] This observation again emphasizes that extracting some sub-keys is more challenging than others. This difficulty stems from the difference in the amount of leakage for each sub-key. This difference in leakage can come from the architecture of the target (related to the hardware) or the implementation of the algorithm (related to the software). However, this is a common phenomenon in SCA, and to justify it, we need to get deeper into hardware and software implementations.



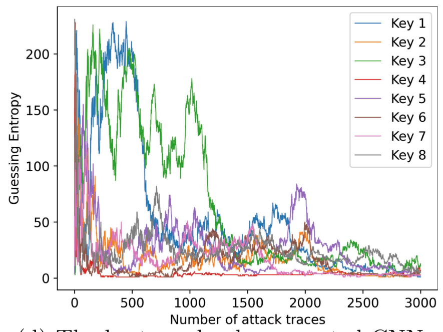
(a) Ensemble of five MLPs.



(b) The best randomly generated MLP.



(c) Ensemble of five CNNs.

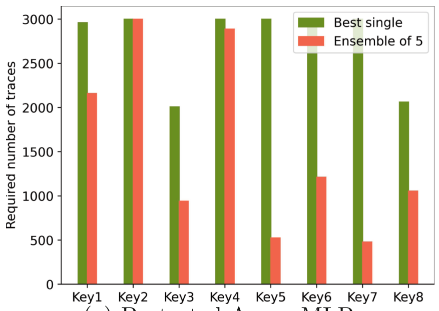


(d) The best randomly generated CNN.

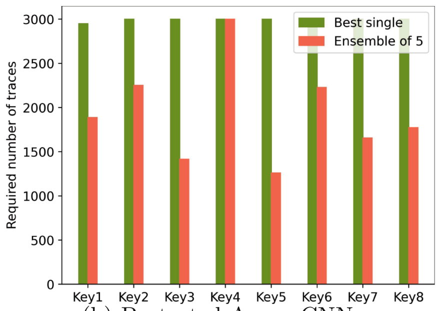
Fig. 5. Guessing entropy for Ascon-Protected. On top, each color shows the evolution of guessing entropy for ensemble of five MLPs (a) and the best-found MLP (b) selected from a pool with fifty randomly generated MLP for each sub-key. On the bottom, each color shows the evolution of guessing entropy for ensemble of five CNNs (c) and the best-found CNN (d) among fifty randomly generated ones for each sub-key.

of both the ensembles and the best-found MLP model in the Ascon-Unprotected dataset and the considerably improved results using the ensemble method in the Ascon-Protected dataset. The difference in the effectiveness of using the ensemble method in these two datasets stems from the difficulty of finding optimal and sub-optimal neural network models. Since it is relatively easy to find powerful models for the Ascon-Unprotected dataset, the ensemble method does not offer much improvement. In contrast, in the case of Ascon-Protected, almost all the best-found models performed poorly. However, combining those weak models through the ensemble method could still significantly improve the attack performance.

It is worth mentioning that using an ensemble of good models is more effective compared to an ensemble of poor models (as expected). While the ensemble method can offer better performance even using poor models, combining good models provides more performance benefits [MK23]. One should consider that with a good model, we do not mean an optimal model but a sub-optimal one



(a) Protected-Ascon-MLP.



(b) Protected-Ascon-CNN.

Fig. 6. The required number of attack traces with and without ensemble method in the Ascon-Protected dataset. On the left side, the required number of attack traces using the best MLP (green) and the ensemble of five MLPs (orange) is compared. On the right side, the required number of attack traces using the best CNN (green) and the ensemble of five CNNs (orange) is compared. (Color figure online)

that can still find the key or reduce the guessing entropy to small values. In the case of the Ascon-Protected dataset, most of the best-found models in our experiments were not good enough to break the target. To find individual models with better performance, we could extend the range of the hyperparameters outlined in Table 1 and increase the number of models in the random models' pool to increase the chance of finding better models.

The result from our ensemble method on the Ascon-Protected dataset significantly improved over the previous work [WP23], where the authors could not recover all the bits of the key with their multi-task model. We can recover the key with less than 3 000 traces using the ensemble method.

6 Conclusions and Future Work

This research investigated the effectiveness of applying an ensemble method to attack both protected and unprotected implementations of the Ascon primitive. While the ensemble method was considered before in DLSCA, its effectiveness for symmetric-key primitives was only validated using AES-based datasets, leading to questions about its applicability to primitives with different operational logic. Our research demonstrated the successful application of ensemble methods to Ascon implementations. Besides, using the ensemble of neural network models, we improved state-of-the-art attacks on Ascon's protected implementation, underscoring that future implementations should consider the current vulnerabilities and that stronger countermeasures are needed to prevent DLSCA. Our experimental results show that with an ensemble of (only) five neural network models, it is possible to extract the secret key with less than 3 000 traces from the protected implementation and, at most, with 100 traces from the unprotected implementation. One possible future work in this direction is using better (and more) models for the ensemble, where we stipulate it can improve the final performance even further.

As the next step, we intend to investigate whether an ensemble of neural networks of different types (ensemble of different topologies like MLP and CNN) trained using different leakage models can improve the attack performance. Our intuition is that a model with a particular topology trained with the same leakage model tends to generate less diverse predictions than models with a different topology trained with different leakage models. Indeed, when we use a dataset and a specific combination of neural network topologies and leakage models, the acquired models are less diverse and mostly focus on similar leakage (points of interest). By integrating diverse neural network types and leakage models into our ensemble, we aim to extract a richer spectrum of information from individual traces, potentially leading to more potent and efficient DLSCA.

References

[AGF21] Acharya, R.Y., Ganji, F., Forte, D.: InfoNEAT: information theory-based neuroevolution of augmenting topologies for side-channel analysis. CoRR, abs/2105.00117 (2021)

[BCO04] Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J.-J. (eds.) CHES 2004. LNCS, vol. 3156, pp. 16–29. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28632-5_2

[BDPVA12] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: Miri, A., Vaudenay, S. (eds.) SAC 2011. LNCS, vol. 7118, pp. 320–337. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28496-0_19

[CDP17] Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) CHES 2017. LNCS, vol. 10529, pp. 45–68. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66787-4_3

[CRR02] Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Kaliski, B.S., Koç, K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36400-5_3

[DEMS16] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2. Submission to the CAESAR Competition **5**(6), 7 (2016)

[GBC16] Goodfellow, I.J., Bengio, Y., Courville, A.C.: Deep Learning. Adaptive Computation and Machine Learning. MIT Press, Cambridge (2016)

[GBTP08] Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 426–442. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_27

[GIB18] Groß, H., Iusupov, R., Bloem, R.: Generic low-latency masking in hardware. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 1–21 (2018)

[KB07] Köpf, B., Basin, D.A.: An information-theoretic model for adaptive side-channel attacks. In: Ning, P., De Capitani di Vimercati, S., Syverson, P.F. (eds.) Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, 28–31 October 2007, pp. 286–296. ACM (2007)

[KJJ99] Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_25

[KPH+19] Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. Unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3), 148–179 (2019)

[LMBM13] Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES. In: Francillon, A., Rohatgi, P. (eds.) CARDIS 2013. LNCS, vol. 8419, pp. 61–75. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08302-5_5

[Mas94] Massey, J.L.: Guessing and entropy. In: Proceedings of 1994 IEEE International Symposium on Information Theory, p. 204 (1994)

[MBC+20] Masure, L., et al.: Deep learning side-channel analysis on large-scale traces. In: Chen, L., Li, N., Liang, K., Schneider, S. (eds.) ESORICS 2020. LNCS, vol. 12308, pp. 440–460. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58951-6_22

[MK23] Mohammed, A., Kora, R.: A comprehensive review on ensemble deep learning: opportunities and challenges. J. King Saud. Univ. Comput. Inf. Sci. **35**(2), 757–774 (2023)

[MPP16] Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) SPACE 2016. LNCS, vol. 10076, pp. 3–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49445-6_1

[NIS23] NIST Information Technology Laboratory. NIST lightweight cryptography standardization process. The National Institute of Standards and Technology (2023). https://csrc.nist.gov/News/2023/lightweight-cryptography-nist-selects-ascon

[PCP20] 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)

[PHJ+17] Picek, S., et al.: Side-channel analysis and machine learning: a practical perspective. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, 14–19 May 2017, pp. 4095–4102. IEEE (2017)

[PPM+23] Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: SoK: deep learning-based physical side-channel analysis. ACM Comput. Surv. **55**(11), 227:1–227:35 (2023)

[RAD20] Ramezanpour, K., Ampadu, P., Diehl, W.: SCARL: side-channel analysis with reinforcement learning on the Ascon authenticated cipher. arXiv preprint arXiv:2006.03995 (2020)

[RB22] Rezaeezade, A., Batina, L.: Regularizers to the rescue: fighting overfitting in deep learning-based side-channel analysis. IACR Cryptology ePrint Archive, p. 1737 (2022)

[RWPP21] Rijsdijk, J., Lichao, W., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 677–707 (2021)

[SDG+20] Schläffer, M., Dobraunig, C., Großschädl, J., dos Santos, L.C., Bachmann, F., Eichlseder, M.: ASCON-C Implementation. Github repository (2020). https://github.com/ascon/ascon-c

[SLP05] Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 30–46. Springer, Heidelberg (2005). https://doi.org/10.1007/11545262_3

[SS23] Shanmugam, D., Schaumont, P.: Improving side-channel leakage assessment using pre-silicon leakage models. In: Kavun, E.B., Pehl, M. (eds.) COSADE 2023. LNCS, vol. 13979, pp. 105–124. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-29497-6_6

[WP23] Weissbart, L., Picek, S.: Lightweight but not easy: side-channel analysis of the Ascon authenticated cipher on a 32-bit microcontroller. IACR Cryptology ePrint Archive, p. 1598 (2023)

[WPP22] Wu, L., Perin, G., Picek, S.: I choose you: automated hyperparameter tuning for deep learning-based side-channel analysis. IEEE Trans. Emerg. Top. Comput. 1–12 (2022)

[ZBHV20] Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient CNN architectures in profiling attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(1), 1–36 (2020)