Power side-channel leakage localization through adversarial training of deep neural networks

Jimmy Gammell*   Anand Raghunathan   Kaushik Roy
Elmore Family School of Electrical and Computer Engineering
Purdue University
West Lafayette, IN, United States
{jgammell, araghu, kaushik}@purdue.edu

Abstract

Supervised deep learning has emerged as an effective tool for carrying out power side-channel attacks on cryptographic implementations. While increasingly-powerful deep learning-based attacks are regularly published, comparatively-little work has gone into using deep learning to defend against these attacks. In this work we propose a technique for identifying which timesteps in a power trace are responsible for leaking a cryptographic key, through an adversarial game between a deep learning-based side-channel attacker which seeks to classify a sensitive variable from the power traces recorded during encryption, and a trainable noise generator which seeks to thwart this attack by introducing a *minimal* amount of noise into the power traces. We demonstrate on synthetic datasets that our method can outperform existing techniques in the presence of common countermeasures such as Boolean masking and trace desynchronization. Results on real datasets are weak because the technique is highly sensitive to hyperparameters and early-stop point, and we lack a holdout dataset with ground truth knowledge of leaking points for model selection. Nonetheless, we believe our work represents an important first step towards deep side-channel leakage localization without relying on strong assumptions about the implementation or the nature of its leakage. A PyTorch implementation of our algorithm and experiments can be found here.

1 Introduction

In recent years, supervised deep learning [LBH15] has emerged as a powerful strategy for carrying out side-channel attacks (SCAs) [MPP16]. Deep neural network (DNN)-based SCAs regularly achieve comparable or superior performance compared to traditional attacks, despite requiring significantly less manual data preprocessing based on *a priori* knowledge about the attacked cryptographic device [MPP16, CDP17, LZC+21]. These attacks pose a major threat to implementations of cryptographic algorithms such as the ubiquitous advanced encryption standard (AES) [DR98] and other symmetric-key algorithms whose secret keys can be determined by observing power consumption or electromagnetic emissions over time while plaintexts are encrypted.

Despite this threat, DNN-based SCAs have a unique potential to elucidate the vulnerabilities in cryptographic implementations and enable the design of targeted countermeasures to mitigate them. While traditional techniques such as template attacks [CRR03] and differential power analysis [KJJ99] can provide a lower bound on the extent to which a device is vulnerable to power SCAs and may identify a subset of the timesteps responsible for this vulnerability, they are limited by their reliance on manual preprocessing, such as extraction of a small number of 'points of interest' from raw power traces, knowledge of random Boolean mask values at profiling-time, or synchronization of traces. Because of this, these methods may not necessarily exploit sources of leakage which were not accounted for during the preprocessing stage. In contrast, supervised DNN-based SCAs are effective even when little or no preprocessing is done, enabling

---

*Corresponding author

the learned classifiers to utilize more sources of leakage. It is then possible to analyze the resulting learned classifiers to identify sources of leakage which were unknown prior to the attack.

In this work we propose a strategy to localize power trace side-channel leakage through an adversarial game played between a DNN-based classifier and a trainable noise generator, in a manner similar to generative adversarial networks [GPAM+14]. The classifier learns to predict the value of some sensitive variable using power traces, while the noise generator learns to introduce noise into the power traces to optimize a compromise between degrading classifier performance and adding as little noise as possible, in a sense we will subsequently precisely define. At the end of training, lots of noise will be added to the timesteps which have high utility for predicting the sensitive variable and little noise will be added to those with low utility, and on this basis the leaking timesteps can be identified. The classifier can take advantage of state-of-the-art DNN-based SCA techniques, allowing the efficacy of our leakage localization strategy to scale with the ever-increasing efficacy of attack strategies as they are published.

To our knowledge, most existing leakage localization strategies rely either on first-order statistics of the power traces, or on 'neural network attribution' techniques applied to DNNs which have been trained to carry out an SCA. In contrast to the former, our technique can identify a dependency between a sensitive variable and multiple timesteps of the input power trace, even when the sensitive variable is pairwise-independent of these timesteps and there is no *a priori* knowledge that the dependency exists (e.g. when Boolean masking is used to protect an attack point). It can also be more data-efficient when useful inductive biases can be imposed through choice of the classifier architecture and the objective function (e.g. convolutional architectures with pooling can bias the classifier towards temporal-translation-invariant predictors to mitigate the effects of trace desynchronization, and an L1-norm penalty on the output of the noise generator can bias it towards producing temporally-sparse noise). In contrast to the latter, our technique has a reduced tendency to falsely identify timesteps as non-leaking, because whereas there is no guarantee that a successful DNN-based attacker will use every dependency between the sensitive variable and power trace timesteps in its predictions, in our technique the classifier is trained simultaneously with an adversarial noise generator, and is forced to seek out new dependencies as those it presently relies on are be progressively attenuated by noise.

In summary, the contribution of this work is:

- We propose a technique for localizing the power trace timesteps responsible for leaking the cryptographic key to power side-channel attacks on AES implementations, through an adversarial game played between a deep neural network (DNN)-based side-channel attacker and a trainable noise generator seeking to thwart the attack. To our knowledge, this is the first proposed leakage localization technique based on an adversarial game played with a DNN.

- We demonstrate experimentally that in contrast to first-order statistics-based leakage localization techniques, our technique can localize leakage of Boolean-masked sensitive variables without profiling-time knowledge of the mask values. It also has improved data-efficiency when applied to desynchronized power traces.

- We demonstrate experimentally that compared to neural network attribution-based leakage localization techniques, our technique has a reduced tendency to fail to recognize sources of leakage when many are present.

We will subsequently present the methodology for our technique and for representative examples of leakage localization techniques based on first-order statistical techniques and neural network attribution. We will then experimentally show that our technique outperforms these baselines on synthetic datasets when common countermeasures are present. A major limitation of the present method is its high sensitivity to hyperparameters and early-stopping point, and it is thus challenging to use it on real datasets where we lack a holdout dataset with ground truth knowledge of leaking points for model selection. Nonetheless, we present results on the publicly-available DPAv4 and ASCADv1-fixed datasets with model selection based on visual inspection, and attain qualitatively-reasonable results.

2 Background and related work

2.1 Supervised deep learning for profiled power side-channel attacks

2.1.1 Profiled power side-channel attacks

Symmetric-key cryptographic algorithms such as AES [DR98] take as input a key $k' \in \mathbb{Z}_+$ and plaintext $p' \in \mathbb{Z}_+$ and return a ciphertext $c' \in \mathbb{Z}_+$, where p' can be uniquely determined from c' given k' but is statistically-independent of c' when nothing is known about k'. This allows parties with exclusive knowledge of k' to communicate over insecure channels without fear of eavesdroppers learning the communicated information, by encrypting the communications according to k', transmitting the resulting ciphertexts, and decrypting the ciphertexts upon receipt. In practice, the security of such communications hinges on k' remaining a secret value known only by the senders and intended recipients of encrypted messages.

While AES and other cryptographic algorithms are secure in the sense that there is no known computationally-feasible algorithm to determine k' by encrypting plaintexts and observing the resulting ciphertexts, *physical implementations* of cryptographic algorithms inevitably leak information through measurable physical signals which are statistically-dependent on the key, called *side-channels*. Side-channels include quantities such as instantaneous power consumption during encryption, electromagnetic radiation during encryption, and algorithm execution time. There is a large body of work on side-channel attacks [Koc96, KJJ99, CRR03, BCO04], which are algorithms to determine k' by observing side-channel emissions during encryptions. In this work we consider *profiled power side-channel attacks*, where the attacker has access to a clone of the target cryptographic implementation and repeatedly measures the instantaneous power consumption during encryption after specifying a plaintext and key, in order to model the probabilistic relationship between power consumption and the key.

When attacking AES implementations, instead of predicting k' directly it is common to learn a predictor of an 'attack point', which is an intermediate variable of the cryptographic algorithm which provides information about k' and strongly influences the power consumption due to being directly-manipulated. A common attack point choice is $z := \text{SBOX}(k \oplus p)$, where $k \in \{0, \ldots, 255\}$ and $p \in \{0, \ldots, 255\}$ denote individual bytes of k' and p' and SBOX is an invertible function which is publicly-available. Given knowledge of p, z uniquely determines the corresponding byte of k' through the identity $k = \text{SBOX}^{-1}(z) \oplus p$. The attack can be repeated for each pair of bytes (p, k) of p' and k' to recover the full key. In this work we will consider leakage of one such attack point z.

2.1.2 Profiling through supervised learning

Given a clone of the target device, an attacker can choose plaintexts and keys uniformly at random and measure the power traces while running the corresponding encryptions, to compile a dataset

$$\mathsf{D} := \{(x_i, z_i)\}_{i=1}^N \in \left(\mathbb{R}^d \times \{0, \ldots, 255\}\right)^N \tag{1}$$

where the values $x_i \in \mathbb{R}^d$ denote measured power traces and $z_i \in \{0, \ldots, 255\}$ denote the aforementioned attack points for one pair of key and plaintext bytes. The attacker can then assume the existence of some underlying data-generating distribution $p(x, z)$ on $\mathbb{R}^d \times \{0, \ldots, 255\}$, of which D consists of i.i.d. samples.

In such a scenario, there is a wide body of work on *supervised learning* techniques [BN06] for using D to find a predictor of attack points given their associated plaintexts, e.g. a function $h : \mathbb{R}^d \to \Delta\{0, \ldots, 255\}$ where $\Delta\{0, \ldots, 255\}$ denotes the set of probability mass functions on $\{0, \ldots, 255\}$, for which $h(x) \approx p(\cdot|x)$ with high probability for x generated by the data distribution p. These techniques entail defining a hypothesis space $\mathsf{H} \subset \mathbb{R}^d \to \Delta\{0, \ldots, 255\}$ of candidate functions, then searching H for a function which minimizes an objective function constructed to be minimal for 'high-quality' functions, according to some desired definition of 'quality'. A common choice is to minimize the *empirical risk minimization* objective,

$$h^* \in \arg\min_{h \in \mathsf{H}} \ \mathcal{L}(h) \quad \text{where} \quad \mathcal{L}(h) := \frac{1}{N} \sum_{i=1}^N \ell(h(x_i), z_i), \tag{2}$$

where $\ell : \Delta\{0, \ldots, 255\} \times \{0, \ldots, 255\} \to \mathbb{R}_+$ is a loss function which returns smaller values when $h(x_i)$ assigns higher mass to z_i.

2.1.3 Supervised deep learning

Over the past decade, deep learning-based techniques [LBH15] have underpinned major advances in diverse domains including natural language processing [OWJ$^+$22], photo-realistic image synthesis [BDS18, DN21], computer vision [KSH12, DBK$^+$20], playing video games [MKS$^+$15] and board games [SSS$^+$17], and protein structure prediction [JEP$^+$21], despite requiring little expert human knowledge or manual data preprocessing compared to prior approaches. Interest has consequently surged [PPM$^+$21] in deep learning-based side-channel analysis, with recent work exploring topics such as finding efficient and performant neural network architectures for this domain [MPP16, BPS$^+$20, ZBHV19, WAGP20], learning from protected AES implementations with little or no manual preprocessing to circumvent countermeasures [BPS$^+$20, CDP17, MS23], and learning from raw traces without point-of-interest selection [LZC$^+$21, BIK$^+$23].

A deep neural network (DNN) is a parametric function approximator $\Phi : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^{256}, \quad (x, \theta) \mapsto \Phi(x; \theta)$ composed by interleaving many parametric linear and nonlinear functions, which can approximate any continuous function from a compact subset of $\mathbb{R}^d$ to $\mathbb{R}^{256}$ to within arbitrarily-small error, provided it has an appropriate architecture. Details about common neural network architectures can be found in textbooks such as [Mur23].

When using DNNs for supervised learning tasks, we implicitly define the hypothesis space in terms of the neural network architecture, $\mathsf{H} = \{\Phi(\cdot; \theta) : \theta \in \mathbb{R}^p\}$, and redefine the objective, which is typically the empirical risk, in terms of θ:

$$\mathcal{L}(\theta) := \frac{1}{N} \sum_{i=1}^{N} \ell(\Phi(x_i; \theta), z_i). \tag{3}$$

We typically choose ℓ to be differentiable and minimize $\mathcal{L}$ using *minibatch stochastic gradient descent (SGD)*, where we randomly partition $\{1, \ldots, N\}$ into minibatch index sets $\mathsf{B}_1, \ldots, \mathsf{B}_T$ and then iteratively compute the parameters

$$\theta_{t+1} = \theta_t - \eta \tilde{\nabla}\mathcal{L}(\theta_t) \quad \text{where} \quad \tilde{\nabla}\mathcal{L}(\theta_t) = \frac{1}{|\mathsf{B}_t|} \sum_{i \in \mathsf{B}_t} [\nabla_\theta \ell(\Phi(x_i; \theta), z_i)]_{\theta=\theta_t} \tag{4}$$

and $\eta > 0$ is a scalar called the *learning rate*.

In practice this is usually repeated for multiple passes over D, called epochs. It is common to use variants of the minibatch SGD algorithm incorporating momentum and adaptive per-parameter learning rates (in this work we use the Adam optimizer [KB15]). Since there are generally a vast number of parameters $\theta \in \mathbb{R}^p$ which perfectly minimize the empirical risk but achieve varying levels of performance in expectation over data generated by p, it is common to modify $\mathcal{L}$ and the training algorithm, as well as the architecture Φ, to bias the iterates θ_t towards regions of $\mathbb{R}^p$ in which $\Phi(\cdot; \theta_t)$ is expected *a priori* to generalize well, or is observed to generalize well on an i.i.d. validation dataset which is not used in the computation of $\mathcal{L}$ or its gradient estimates.

2.2 Power side-channel leakage localization

Existing leakage localization approaches can broadly be categorized into those based on first-order statistical techniques, and those based on neural network attriburion techniques.

2.2.1 First-order statistical techniques

First-order statistical techniques are often used as an initial 'point-of-interest' detection step to reduce the dimensionality of raw power trace measurements prior to applying SCA techniques. Given a database of power traces in $\mathbb{R}^d$, the goal is to identify timesteps $\tau_1, \ldots, \tau_m \in \{1, \ldots, d\}$ at which power consumption has a strong dependence on the value of some sensitive variable, where $m \ll d$. The motivation is normally to reduce the computational cost of modeling the relationship between a power trace and the cryptographic key, given long power traces in which most measurements are nearly-independent of the key.

The general approach is to compute the average power measurement at each timestep given various distinct values of the sensitive variable (e.g. a byte, byte Hamming weight, or bit of the first SBOX operation of the AES algorithm), then assign a score to each timestep which is larger for timesteps at which the power

measurement varies more for different sensitive variable values. In this work we consider the signal-noise ratio (SNR) [MOP08], defined in Alg. 1, as a representative example of the category of first-order statistical leakage localization techniques, but there are many other examples, such as the sum of differences [GLP06], t-statistic [Wel47, MOBW13] and the χ^2 test [MRSS18].

Algorithm 1: Signal-to-noise ratio (SNR) computation

Input: $D = \{(x^{(i)}, z^{(i)})\}_{i=1}^N \in \left(\mathbb{R}^d \times \{0, \ldots, 255\}\right)^N$ // dataset of traces + attack point pairs
Output: $m \in \mathbb{R}^d$ // signal-noise ratio mask

1 **for** $z \in \{0, \ldots, 255\}$ **do**
2   Compute $\mu_z \leftarrow \text{Average}_{i \in \{1, \ldots, N\}: z^{(i)} = z} x^{(i)}$ // average power trace for fixed attack point value
3 Compute $\mu \leftarrow \text{Average}_{i \in \{1, \ldots, N\}} x^{(i)}$ // average power trace over full dataset
4 Compute $\sigma_{\text{signal}}^2 \leftarrow \text{Average}_{z \in \{0, \ldots, 255\}} (\mu_z - \mu)^2$ // signal variance
5 Compute $\sigma_{\text{noise}}^2 \leftarrow \text{Average}_{i \in \{1, \ldots, N\}} (x^{(i)} - \mu_{z^{(i)}})^2$ // noise variance
6 Compute $m \leftarrow \sigma_{\text{signal}}^2 / \sigma_{\text{noise}}^2$ // signal-noise ratio mask
7 **return** m

The main shortcoming of these techniques is that because they consider individual timesteps in isolation, they fail to detect dependencies between *multiple* timesteps and the cryptographic key. For example, *Boolean masking* is a common countermeasure against side-channel attacks, where instead of operating directly on a sensitive byte $z \in \{0, \ldots, 255\}$, the algorithm randomly draws a 'Boolean mask' byte $r \sim \mathcal{U}\{0, \ldots, 255\}$ and then operates on $z \oplus r$ where $\cdot \oplus \cdot$ denotes the bit-wise XOR operation. In this context, z is independent of $z \oplus r$ absent knowledge of r, but can be determined with certainty given r. Because r and $z \oplus r$ influence different timesteps in the power trace, techniques considering the timesteps in isolation will fail to detect this source of leakage. Many works sidestep this limitation by assuming knowledge of the Boolean mask values at profiling-time, but since these mask values are intermediate variables which are likely to be hidden from users of a cryptographic implementation, in a practical attack such knowledge may be unattainable or attainable only through great effort.

2.2.2 Neural network attribution techniques

Another approach for leakage localization is to train a DNN-based side-channel attacker using standard supervised learning techniques, then use one of a variety of 'neural network attribution' techniques to score the timesteps of the input power traces in terms of their influence on the classification decision. Given a database of power traces $x_1, \ldots, x_N \in \mathbb{R}^d$ and a trained classifier $\Phi^* : \mathbb{R}^d \to \mathbb{R}^{256}$, these techniques generate importance masks for each power trace, $m_1, \ldots, m_N \in \mathbb{R}^d$, where the score of a timestep increases with its amount of 'influence' on the predictions made by Φ^*, which can then be summarized by averaging over the dataset.

Many methods exist for generating these importance masks given a pretrained classifier Φ^*, including differentiating functions of the output of Φ^* with respect to its input [MDP18, HGG19], occluding sections of the input [HGG19], ablating sections of neural networks [WWJ+21], and singular vector canonical correlation analysis of internal representations [vdVPB21]. In this work, we consider the gradient visualization technique (GradVis) [MDP18], defined in Alg. 2, as a representative example.

While these techniques avoid the main shortcomings of first-order statistical techniques, they have the major shortcoming that there is no guarantee that Φ^* will rely on *all* relevant timesteps when classifying power traces, meaning that the resulting importance mask will assign high importance to only a subset of the timesteps which may be used to identify the cryptographic key. It has been documented in other domains that DNNs often ignore input-output relationships which may be useful for prediction; for example, CNN-based image classifiers are biased towards textures and often fail to capture relationships between object shapes and labels [GRM+19], DNN classifiers have been shown to ignore XOR relationships when

---
Algorithm 2: Gradient visualization (GradVis) computation

Input: $D = \{(x^{(i)}, z^{(i)})\}_{i=1}^{N} \in \left(\mathbb{R}^d \times \{0, \ldots, 255\}\right)^N$ // `dataset of traces + attack point`
    `values`

Input: $\Phi^* : \mathbb{R}^d \to \mathbb{R}^{256}$ // `deep neural network trained on` D `using standard supervised`
    `learning techniques`

Input: $\ell : \Delta\{0, \ldots, 255\} \times \{0, \ldots, 255\} \to \mathbb{R}_+$ // `loss function used to define the`
    `empirical risk when the DNN was trained`

Output: $m \in \mathbb{R}^d$ // `Gradient visualization mask`

1 for $i \in \{1, \ldots, N\}$ **do**

2   Compute $\tilde{m}^{(i)} \leftarrow \nabla_x \ell \left(\Phi^*(x), z^{(i)}\right)\big|_{x=x^{(i)}}$ // `sensitivity of loss to small perturbation`
        `of each input element`

3 Compute $m \leftarrow \text{Average}_{i \in \{1, \ldots, N\}} |\tilde{m}^{(i)}|$ // `average sensitivity over full dataset`

4 return m

---

linear relationships are present [HL20], and more-generally to ignore complex input-output relationships when simpler relationships (e.g. more linearly-decodable from the hidden DNN activations at initialization) are present [GJM$^+$20].

2.3 Other related work

Our approach of joint adversarial training of a DNN-based classifier and a noise generator was inspired by generative adversarial networks (GANs) [GPAM$^+$14]. While we are not aware of existing work applying GAN-like approaches to leakage localization, GANs have been used for data augmentation to facilitate power side-channel attacks [WCL$^+$20, MBPK22].

Leakage localization is similar to weakly-supervised semantic segmentation from labels [AK18], which is a well-studied problem in computer vision. In this setting, the goal is to learn a model which highlights pixels corresponding to a particular class in images, by training on image data with class-level labels but no pixel-level labels. Many techniques in this domain are based on attribution of DNN classifier predictions while adversarially-erasing regions of input images to encourage the classifier to use all discriminative regions of inputs [WFL$^+$17, ZWF$^+$18, KYK$^+$21]. These techniques are conceptually-similar to ours, the major difference being that they use non-persistent per-image adverarial masks whereas we use a single adversarial mask for the entire dataset which is incrementally trained.

GANs have been applied to 'data sanitization', where a 'generator' DNN seeks to minimally modify inputs while removing a dependence between the inputs and some sensitive variable. Similarly to our technique, this is achieved through an adversarial game played against a classifier DNN which tries to predict the sensitive variable from the inputs. This approach has been used to promote fairness by sanitizing datapoints or DNN activations of dependence on a protected attribute such as sex or race [ES15, HPK$^+$20, ZWF$^+$18, XDD$^+$17, MCPZ18], and to enable robustness to data distribution shifts by sanitizing inputs or DNN activations of dependence on properties of data which may change when the data distribution changes [LPWK18, LTG$^+$18, DDD$^+$20]. Our technique differs from these because instead of a DNN-based 'generator' we have a data-independent mask, and we seek a mask which is applicable to the full dataset rather than specialized for individual datapoints within it.

3 Methodology

3.1 Adversarial masking (AdvMask)

In Alg. 3 (illustrated schematically in Fig. 1) we present our AdvMask technique for side-channel leakage localization through simultaneous adversarial training of a DNN-based classifier and a noise generator. The classifier is trained in a supervised manner to classify noisy power traces based on the value of a sensitive variable, which in this work is the output of the first SBOX operation of the AES algorithm. Simultaneously,
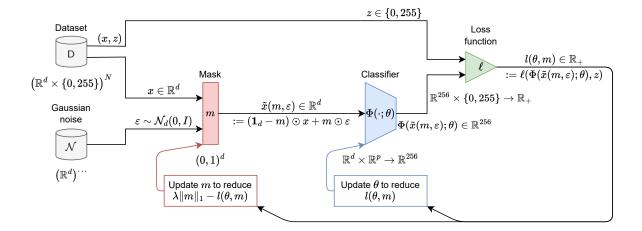
Figure 1: Schematic illustration of AdvMask, our adversarial masking technique for side-channel leakage localization.

a noise generator, represented by a data-independent vector $m \in (0,1)^d$ which dictates a convex combination between power trace timesteps $x \in \mathbb{R}^d$ and i.i.d. d-dimensional standard normal noise $\varepsilon \sim \mathcal{N}_d(0, I)$, is trained to optimize a composite objective function which rewards both increasing the loss of the classifier and penalizing the L1 norm of m $\|m\|_1 := \sum_{i=1}^d |m_i|$. We optimize these objectives using alternating minibatch stochastic gradient descent, which is similar to the training procedure for generative adversarial networks [GPAM+14] and can be easily implemented using the popular deep learning libraries such as PyTorch or Keras.

As mentioned previously, first-order statistical methods are unable to detect dependencies between sensitive variables and multiple power trace timesteps when the sensitive variable is pairwise-independent of each timestep, as is the case when Boolean masking is used to break a variable z into two statistically-independent shares r and $z \oplus r$. The reasons for this are that 1) such methods look for dependence between z and each individual timestep but not between z and sets of multiple timesteps, and 2) many such methods cannot detect nonlinear dependencies between z and timesteps of the power trace, such as the XOR function $z = r \oplus (z \oplus r)$ which must be computed to extract z from the shares. In contrast, DNN-based methods are capable of detecting these dependencies due to the ability of DNNs to learn arbitrary continuous functions on $\mathbb{R}^d$. Their ability to overcome Boolean masking without assuming knowledge of masks at profiling time has been demonstrated empirically [MPP16, BPS+20].

While neural network attribution methods don't suffer from the above limitation, they are limited insofar as there is no guarantee that the DNN they are applied to has absorbed every useful relationship between power traces and the target sensitive variables. Adversarial masking overcomes this issue by jointly training a classifier with an adversarial noise generator, in such a way that noise is gradually added to the timesteps which are most-used by the classifier. Over time, the classifier is thereby forced to learn new input-output relationships to retain its performance.

3.2 Synthetic power traces

In order to evaluate the efficacy of our method in a controlled setting and to observe the effect of various interventions on dataset properties, we have run a variety of experiments on synthetic power traces.

Our basic procedure for generating synthetic datasets is shown in Alg. 4, based on the Hamming weight power consumption model [MOP08]. We construct each power trace $x^{(i)}$ as a sum of 3 components:

$$x^{(i)} = \varepsilon_{\mathrm{lkg}}^{(i)} + \varepsilon_{\mathrm{rand}}^{(i)} + \varepsilon_{\mathrm{op}} \tag{5}$$

where

- $\varepsilon_{\mathrm{lkg}}^{(i)}$ denotes the attack point-dependent power consumption. We model this as an affine function of the

---

Algorithm 3: Our adversarial mask (AdvMask) computation

Input: $\mathsf{D} = \{(x^{(i)}, z^{(i)})\} \in \left(\mathbb{R}^d \times \{0, \ldots, 255\}\right)^N$ // dataset of traces + attack point values
Input: $\Phi : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^{256}, \;\; (x, \theta) \mapsto \Phi(x; \theta)$ // classifier neural network architecture
Input: $\ell : \mathbb{R}^{255} \times \{0, \ldots, 255\} \to \mathbb{R}_+$ // loss function used to define empirical risk of
     DNN classifier (default: categorical cross-entropy loss)
Input: $\mathrm{UpdateParams}^\theta : \mathbb{R}^p \to \mathbb{R}^p$ // parameter update function for the classifier
     (default: Adam optimizer with $\alpha = 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999$)
Input: $\mathrm{UpdateParams}^m : \mathbb{R}^d \to \mathbb{R}^d$ // parameter update function for the mask (default:
     Adam optimizer with $\alpha = 10^{-2}, \beta_1 = 0.5, \beta_2 = 0.0$)
Input: $\theta_0 \in \mathbb{R}^p$ // initial classifier parameters (default: $\theta_0 = \mathrm{KaimingUniformInit}(\Phi)$)
Input: $m_0 \in \mathbb{R}^d$ // initial pre-sigmoid mask values (default: $m_0 = -10 \cdot \mathbf{1}_d$)
Input: $\lambda \in \mathbb{R}_+$ // L1-norm penalty on mask
Output: $m \in \mathbb{R}^d$ // adversarial mask

---

1 **Procedure** ApplyMask($x \in \mathbb{R}^d$, $m \in \mathbb{R}^d$)
2   Draw random noise ε from $\mathcal{N}_d(0, I)$
3   Soft-threshold mask $\tilde{m} \leftarrow \mathrm{Sigmoid}(m)$
4   Compute masked example $\tilde{x} \leftarrow (\mathbf{1}_d - \tilde{m}) \odot x + \tilde{m} \odot \varepsilon$
5   **return** $\tilde{x}$

6 $t \leftarrow 0$
7 **while** not converged **do**
8   Draw datapoint (x_t, z_t) from D
9   Compute classifier gradient $g_t^\theta \leftarrow \nabla_\theta \left[\ell \left(\Phi(\texttt{ApplyMask}(x_t, m_t); \theta), z_t \right) \right]_{\theta = \theta_t}$
10   Update classifier parameters $\theta_{t+1} \leftarrow \mathrm{UpdateParams}^\theta(g_t^\theta)$
11   Compute mask gradient $g_t^m \leftarrow \nabla_m \left[\lambda \|m\|_1 - \ell \left(\Phi(\texttt{ApplyMask}(x_t, m); \theta_{t+1}), z_t \right) \right]_{m = m_t}$
12   Update mask parameters $m_{t+1} \leftarrow \mathrm{UpdateParams}^m(g_t^m)$
13 $t \leftarrow t + 1$
14 **return** $\mathrm{Sigmoid}(m_t)$

Hamming weight of the attack point at the leaking timesteps, and 0 elsewhere. We generally assume that leakage is *temporally-sparse*, i.e. most elements of this vector will be 0.

- $\varepsilon_{\mathrm{rand}}^{(i)}$ denotes the 'random' component of power consumption stemming from factors which are irrelevant to carrying out a side-channel attack (e.g. electrical noise, data-dependent power consumption for attack point-irrelevant data). We model this as Gaussian noise.

- $\varepsilon_{\mathrm{op}}$ denotes the operation-dependent power consumption which is the same for every encryption. We model this as Gaussian noise which is sampled once prior to dataset generation, then added to every generated power trace.

We additionally constrain the variance of the power trace be the same at every timestep (hence the expression in Ln. 5 of Alg. 4). The two critical assumptions are that 1) leakage is proportional to the Hamming weight of the attack point, and 2) leakage is temporally-sparse. Both are likely to approximately hold for unprotected devices which leak primarily due to power consumption when the attack point value is loaded onto a bus or a register [MOP08]. We will shortly modify Alg. 4 to simulate masking and hiding countermeasures which are common in protected cryptographic implementations.

Unless stated otherwise, subsequent experiments will use the default parameter values listed in Alg. 4.

3.3 Evaluation metrics

Given a mask $m := (m_i)_{i=1}^d \in [0, 1]^d$ and a dataset D generated by Alg. 4, we can define the index sets

$$\mathsf{I}_{\mathrm{lkg}} := \{\tau_1, \ldots, \tau_\ell\} \;\; \text{and} \;\; \mathsf{I}_{\neg\mathrm{lkg}} := \{1, \ldots, d\} \setminus \mathsf{I}_{\mathrm{lkg}}. \tag{6}$$

Algorithm 4: Basic synthetic dataset generation

Input: $N \in \mathbb{Z}_+$ // `number of datapoints to generate (default:` $N = 5 \times 10^4$`)`
Input: $d \in \mathbb{Z}_+$ // `timesteps per power trace (default:` $d = 128$`)`
Input: $\sigma_{\text{op}}^2, \sigma_{\text{rand}}^2 \in \mathbb{R}_+$ // `variance of resp. operation-dependent noise and random`
   `noise (default:` $\sigma_{\text{op}}^2 = 1, \sigma_{\text{rand}}^2 = 0.5$`)`
Input: $\tau_1, \ldots, \tau_\ell \in \{0, \ldots, d-1\}$ // `leaking timesteps (default: 1 leaking timestep with`
   $\tau_1 = \lfloor \frac{1}{2} d \rfloor$`)`
Input: $\alpha_1, \ldots, \alpha_\ell \in [0,1]$ // `proportion of variance due to attack point Hamming weight`
   `at leaking timesteps (default: each` $\alpha_i = \frac{1}{2}$`)`
Output: $\mathsf{D} = \{(x^{(i)}, z^{(i)})\}_{i=1}^N \in \left(\mathbb{R}^d \times \{0, \ldots, 255\} \right)^N$ // `Synthetic dataset of (power trace,`
   `attack point) pairs`

---

1 Sample ε_{op} from $\mathcal{N}_d(0, \sigma_{\text{op}}^2 I)$ // `operation-dependent noise added to every power trace`

2 for $i = 1, \ldots, N$ **do**

3   Sample $z^{(i)}$ from $\mathcal{U}\{0, \ldots, 255\}$ // `attack point value`

4   Sample $\varepsilon_{\text{rand}}^{(i)}$ from $\mathcal{N}_d(0, \sigma_{\text{rand}}^2 I)$ // `per-trace random noise`

5   Compute $\varepsilon_{\text{lkg}}^{(i,j)} \leftarrow \sqrt{\frac{\alpha_j}{2}} \left(\text{HammingWeight}(z^{(i)}) - 4 \right) + \sqrt{1 - \alpha_j}(\varepsilon_{\text{rand}}^{(i)})_j$ for $j = 1, \ldots, \ell$

       // `data-dependent power consumption at leaking points`

6   Compute $x^{(i)} \leftarrow \varepsilon_{\text{op}} + \left(\begin{array}{ll} \varepsilon_{\text{lkg}}^{(i,k)} & \text{if } j = \tau_k \text{ for some } k \in \{1, \ldots, \ell\} \\ (\varepsilon_{\text{rand}}^{(i)})_j & \text{else} \end{array} \right)_{j=1}^d$ // `power trace`

       `corresponding to` $z^{(i)}$

7 return $\mathsf{D} = \{(x^{(i)}, z^{(i)})\}_{i=1}^N$

We will summarize the quality of m with respect to D using the mean z-score

$$\text{Z-score}(m; \mathsf{I}_{\text{lkg}}, \mathsf{I}_{\neg\text{lkg}}) := \text{Average}_{i \in \mathsf{I}_{\text{lkg}}} \left[\frac{m_i - \text{Average}_{j \in \mathsf{I}_{\neg\text{lkg}}} m_j}{\text{StandardDeviation}_{j \in \mathsf{I}_{\neg\text{lkg}}} m_j} \right] \tag{7}$$

and the area under the precision-recall curve, where we define precision

$$\text{Precision}(m; z, \mathsf{I}_{\text{lkg}}, \mathsf{I}_{\neg\text{lkg}}) := \frac{|\{\tau = 1, \ldots, d : m_\tau \geq z, \tau \in \mathsf{I}_{\text{lkg}}\}|}{|\{\tau = 1, \ldots, d : m_\tau \geq z\}|} \tag{8}$$

and recall

$$\text{Recall}(m; z, \mathsf{I}_{\text{lkg}}, \mathsf{I}_{\neg\text{lkg}}) := \frac{|\{\tau = 1, \ldots, d : m_\tau \geq z, \tau \in \mathsf{I}_{\text{lkg}}\}|}{|\{\tau = 1, \ldots, d : m_\tau \geq z, \tau \in \mathsf{I}_{\text{lkg}}\} \cup \{\tau = 1, \ldots, d : m_\tau < z, \tau \in \mathsf{I}_{\text{lkg}}\}|}, \tag{9}$$

then numerically approximate the area under the curve with recall on the horizontal axis and precision on the vertical axis as we sweep the threshold z from 0 to 1.

     We will see that early stopping is necessary to select a high-quality mask when training for many epochs with both the GradVis and AdvMask algorithms. Given a set of masks $\mathsf{M} \subset [0,1]^d$ generated by e.g. recording the mask at the end of each epoch of training, we will select a 'best' mask $m^* \in \mathsf{M}$ using the policy

$$m^* \in \arg\max_{m \in \mathsf{M}'} \text{Z-score}(m; \mathsf{I}_{\text{lkg}}, \mathsf{I}_{\neg\text{lkg}}) \quad \text{where} \quad \mathsf{M}' := \arg\max_{m \in \mathsf{M}} \text{PR-AUC}(m; \mathsf{I}_{\text{lkg}}, \mathsf{I}_{\neg\text{lkg}}). \tag{10}$$

While this policy cannot be used in practice because it relies on 'oracle' knowledge of which timesteps leak the key, it provides a useful basis for comparing mask-generation algorithms on synthetic datasets for which the leaking timesteps are known.
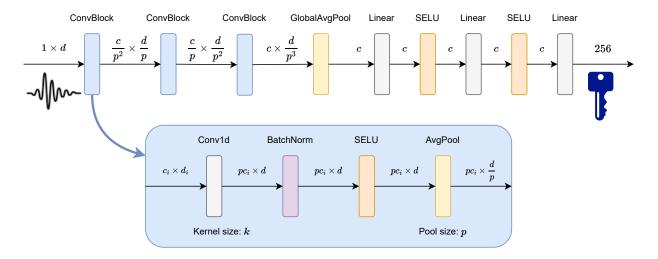
Figure 2: Convolutional neural network classifier architecture used in experiments.

4 Results

4.1 Implementation details and hyperparameter settings

We have implemented the following experiments in PyTorch [PGM$^+$19], with some evaluation metrics computed using scikit-learn [PVG$^+$11]. Our code can be found here.

Unless otherwise specified, in the following experiments we use the CNN architecture described in Fig. 2 for all classifiers. The architecture style based on convolution–activation–pooling blocks followed by linear–activation blocks was popular in early applications of DNNs to computer vision [LBBH98, KSH12, SZ15] and has proven successful for side-channel analysis of short power traces [BPS$^+$20, ZBHV19, WAGP20]. We use a global average pooling (GlobalAvgPool) layer [LCY13] after the convolutional blocks rather than the typical flattening layer, so that we can apply the architecture to arbitrary-length inputs without a quadratic increase in parameter count. As was done in [ZBHV19], we use batch normalization (BatchNorm) [IS15] and the scaled exponential linear unit (SELU) activation [KUMH17]. We train all models with the Adam [KB15] variant of minibatch stochastic gradient descent. For each dataset we compute the mean and standard deviation of all timesteps of all traces and use these to standardize the traces:

$$x^{(i)} \leftarrow \left(\frac{x_j^{(i)} - \mu}{\sigma} \right)_{j=1}^{d} \tag{11}$$

$$\text{where } \mu := \text{Average}_{\substack{i \in \{1,\dots,N\} \\ j \in \{1,\dots,d\}}} x_j^{(i)} \text{ and } \sigma := \text{StandardDeviation}_{\substack{i \in \{1,\dots,N\} \\ j \in \{1,\dots,d\}}} x_j^{(i)}.$$

We do not standardize the individual timesteps of each trace (i.e. compute per-timestep means and standard deviations $\mu, \sigma \in \mathbb{R}^d$) because the global average pooling layer removes information about the absolute position of timesteps, forcing the network to rely on relative position information which is encoded in the mean values of neighboring timesteps (i.e. the operation-dependent power consumption).

Unless otherwise specified, we use the hyperparameter values listed in Tbl. 1. These have been tuned in an *ad hoc* manner and seem to work in a variety of settings, but are not necessarily the optimal values for any particular setting. For each trial we use the same classifier architecture and classifier optimizer settings for both the GradVis and AdvMask methods. We compute the SNR using the full dataset, in contrast to the DNN-based methods where we partition the dataset into a training and validation subset and do not train on the latter.

Our goal with these experiments is to compare the trends and failure modes of each of the 3 considered methods in various settings. They do not conclusively demonstrate that AdvMask is 'better' than GradVis, and to do so it would be necessary to independently tune the hyperparameters for both methods under a

Hyperparameter	Default value
Classifier architecture	Fig. 2 with $c = 16$, $p = 2$, $k = 11$
Classifier optimizer	Adam($\alpha = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)
Mask optimizer	Adam($\alpha = 10^{-2}$, $\beta_1 = 0.5$, $\beta_2 = 0.0$, $\epsilon = 10^{-8}$)
Mask L1 norm penalty λ	1.0
Minibatch size	256
Validation split proportion	0.2
Training duration	10^4 minibatches

Table 1: Hyperparameter settings used in experiments, except where specified otherwise.

Figure 3: Sweep of the AdvMask L1 norm penalty, while varying the proportion of variance at leaking points due to the Hamming weight of the sensitive variable. Observe that for sufficiently-large variance, the optimal L1 decay value increases with variance. When the variance is very low, AdvMask fails to identify the leaking point and the optimal L1 norm penalty is volatile.

fixed computational budget for each trial, and to explore a wider variety of hyperparameters and neural network architectures to ensure we have not settled in a local minimum of the search space with respect to one method, which is sub-optimal for the other.

4.2 Experiments on synthetic data

Here we evaluate the performance of our AdvMask technique on synthetic power trace datasets generated in the manner described in section Sec. 3.2. We compare it to the SNR (Sec. 2.2.1) and GradVis (Sec. 2.2.2) techniques as representative examples of techniques based on resp. first-order statistical calculations and neural network attribution. We repeat L1 norm sweeps for 3 random seeds and dataset setting sweeps for 5 random seeds. In the subsequent plots, shaded areas cover the full range of values measured across these repeated trials, with dots denoting the median values and dotted lines interpolating the dots.

4.2.1 Impact of proportion of variance due to leakage

We first examine the impact of the proportion of variance due to leakage at the leaking points. We construct datasets according to Alg. 4, with all inputs set to their default values except that we sweep the proportion of variance due to the Hamming weight of the target variable, α_1, from 0 to 1.

In Fig. 3 we examine the impact of α_1 on the optimal mask L1 norm penalty λ. We see that for $\alpha_1 < 10^{-2}$ when there is very little leakage, our technique fails to detect the leaking point and the optimal value of λ is volatile. For $\alpha_1 \geq 10^{-2}$, our method starts to work and we see that the optimal λ increases with α_1.
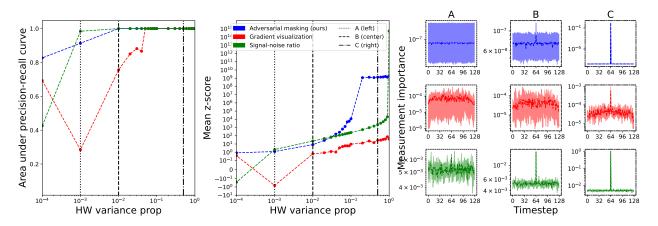
Figure 4: Sweep of the proportion of variance at leaking points due to the Hamming weight of the sensitive variable. Observe that AdvMask (ours) and SNR have similar minimal variance requirements, while GradVis requires significantly-more variance to identify the leaking timestep.
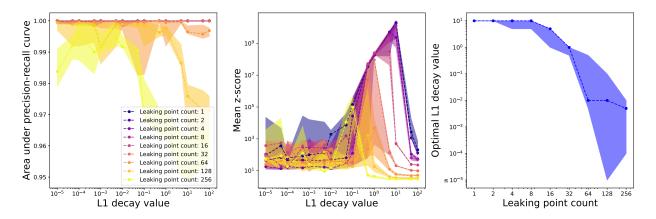

Figure 5: Sweep of the AdvMask L1 norm penalty, while varying the number of leaking points. Observe that the optimal norm penalty decreases as the number of leaking points increases.

In Fig. 4 we compare AdvMask to SNR and GradVis as the amount of dataset noise varies. We see that all 3 methods fail with α_1 is extremely low and succeed when it grows sufficiently large. AdvMask and SNR are comparably-robust to α_1, with the former outperforming the latter at moderate α_1, likely due to the inductive bias imposed by the L1 norm penalty in its training routine. GradVis performs the worst, failing to detect the leaking point until $\alpha_1 = 2 \times 10^{-1}$; we suspect it underperforms AdvMask because the noise added in the AdvMask training procedure provides a useful regularization effect which prevents overfitting.

4.2.2   Impact of number of distinct points which are leaking

Next, we vary the number of points per power trace which leak the key, setting $\tau_i = \left\lfloor \frac{i}{n+1} d \right\rfloor$ for $i = 1, \ldots, n$ while varying n. In the first experiment we also set the trace length to $d = 512$, while in the second experiment we leave all other inputs to Alg. 4 at their default values.

In Fig. 5 we examine the impact of n on the optimal L1 norm penalty λ. We see that as we increase n, the optimal value of λ decreases, which makes sense because λ determines the tradeoff the mask should make between increasing classifier loss and minimizing the amount of added noise, and as a greater proportion of points leaks, the amount of added noise should increase.

In Fig. 6 we compare AdvMask to SNR and GradVis as the number of leaking points varies. We see that the number of leaking points has no impact on the SNR, as it computes the statistics of each leaking point
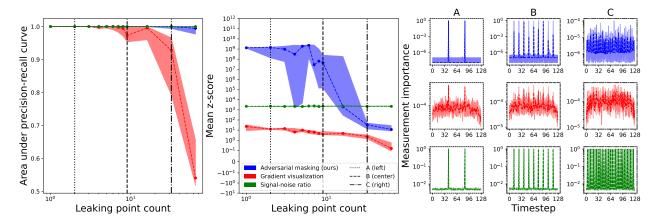
12

Figure 6: Sweep of the number of timesteps at which the key leaks. SNR is unaffected by the number of leaking points, whereas AdvMask (ours) sees a modest performance degradation and GradVis sees a very-substantial performance degradation.
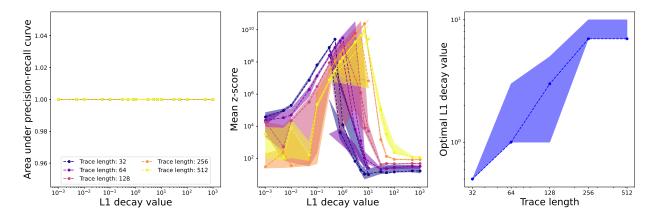

Figure 7: Varying-length power traces, sweeping the AdvMask norm penalty. Observe that the optimal penalty increases with trace length.

in isolation. Both AdvMask and GradVis seek reduced performance as n increases, but the performance degradation for GradVis is much stronger, as it begins to miss a substantial proportion of leaking points when $n \geq 64$. We believe AdvMask performs better in this setting because while in the GradVis setting the classifier can accurately predict the sensitive variable using only a small number of leaking points, the simultaneous adversarial training of the classifier and noise generator in the AdvMask setting forces the classifier to use as many points as possible as those it currently relies are progressively attenuated by the noise generator.

4.2.3   Impact of power trace length

Here we vary the number of timesteps per power trace d, while leaving other inputs to Alg. 4 at their default values.

In Fig. 7 we examine the impact of d on the optimal L1 norm penalty λ. We see that trace length does not substantially change results, with every λ successfully detecting the leaking point, though there is a small increase in the optimal λ as traces become longer.

In Fig. 8 we compare AdvMask to SNR and GradVis as the length of power traces increases. As expected, the length of the power trace has no impact on the performance of SNR, which considers timesteps in isolation. Surprisingly, both AdvMask and GradVis see some performance degradation when the lengths of
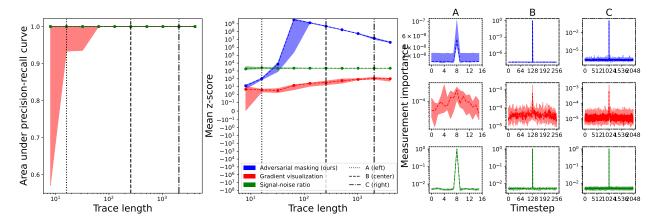
13

Figure 8: Sweep of the number of timesteps per power trace. SNR is unaffected by power trace length, whereas AdvMask (ours) and GradVis see some performance degradation for very-short power traces.
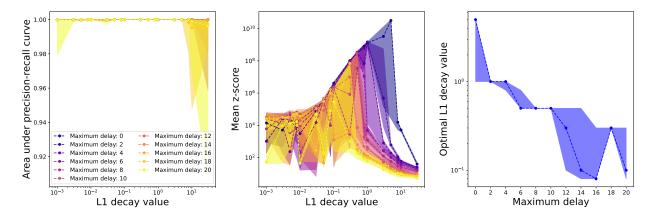

Figure 9: Datasets with randomly desynchronized traces, sweeping the AdvMask L1 norm penalty. Observe that the optimal value decreases as the amount of desynchronization increases.

power traces are very small. One possible explanation is that the CNN classifier architecture and optimizer hyperparameters used by both GradVis and AdvMask are inappropriate for small power traces. For example, convolutional layers offer a regularizing effect when the power trace length is substantially larger than the convolutional kernel, but become increasingly similar to linear layers as power trace length decreases.

4.2.4 Impact of trace desynchronization

Here we add random delays to the power traces to simulate trace desynchronization which may stem from e.g. random delays or clock jitter inserted into the AES implementation. To implement random delays of duration at most m timesteps, we execute Alg. 4 to generate traces of length $d + m$, then for each trace $x^{(i)}$ after Ln. 6 crop out a random d-length sub-trace:

$$x^{(i)} \leftarrow \left(x_j^{(i)} \right)_{j=m^{(i)}+1}^{d+m^{(i)}} \quad \text{for} \ \ m^{(i)} \sim \mathcal{U}\{0, \ldots, m\}. \tag{12}$$

In the subsequent experiments we vary m while leaving the other inputs to Alg. 4 at their default values.

In Fig. 9 we examine the impact of trace desynchronization on the optimal value of the mask L1 norm penalty λ. We see that the optimal value of λ decreases significantly as m increases. Trace desynchronization increases the number of leaking timesteps, so this effect makes sense given that increasing the number of timesteps which leak also decreases the optimal value of λ.
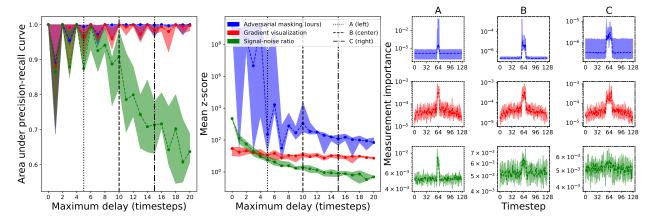
Figure 10: Dataset with randomly desynchronized traces, sweeping the maximum amount of desynchronization. Observe that the performance of SNR quickly drops off, whereas both AdvMask (ours) and GradVis are robust to desynchronization due to their use of a convolutional neural network for leakage localization.

In Fig. 10 we compare AdvMask to SNR and GradVis as the amount of desynchronization increases. Observe that while AdvMask and GradVis see a small performance degradation as the amount of desynchronization increases, they remain performant, detecting most of the leaking points. In contrast, SNR sees substantial performance degradation. This highlights one of the major advantages of deep learning leakage localization methods: it is possible to modify the DNN architecture or objective function to impose inductive biases towards functions which are expected *a priori* to generalize well. In this case, the convolution and pooling layers in the classifier architectures of AdvMask and GradVis bias them towards functions which are invariant to temporal shifts of the input, which makes them robust to trace desynchronization.

4.2.5 Impact of Boolean masking

Here we simulate Boolean masking, which is a common countermeasure to increase the difficulty of discerning sensitive variables by breaking them up into statistically-independent shares which are operated on at distant points in time. In Ln. 3 of Alg. 4 we sample both an attack point value $z^{(i)}$ and a Boolean mask value $r^{(i)}$ from $\mathcal{U}\{0, \ldots, 255\}$, then compute a 'masked' attack point $z^{(i)} \oplus r^{(i)}$ where $\cdot \oplus \cdot$ denotes the bitwise XOR operation. We then implement leakage for both $r^{(i)}$ and $z^{(i)} \oplus r^{(i)}$ at separate timesteps in the same manner as above.

Observe that $z^{(i)} \oplus r^{(i)}$ is independent of $z^{(i)}$ unless information about $r^{(i)}$ is known. This makes $z^{(i)}$ more-challenging to attack because to learn its value one must learn the values of both $r^{(i)}$ and $z^{(i)} \oplus r^{(i)}$ which are leaked at distant timesteps, then perform the nonlinear computation $r^{(i)} \oplus (z^{(i)} \oplus r^{(i)}) = z^{(i)}$. Because two values must be learned this is often called *second-order* leakage, while leakage in the manner described above is called *first-order* leakage.

In Fig. 11 we compare AdvMask to SNR and GradVis while varying the number of timesteps at which r and $r \oplus z$ leak, with $\alpha_i = 0.5$ for all i. In Fig. 12 we vary the proportion of variance due to r and $r \oplus z$ while leaving the number of leaking timesteps fixed at 4 (2 for each share). Second-order leakage is harder to classify than first-order leakage, and we make the following *ad hoc* changes to hyperparameter settings to increase the consistency of AdvMask and GradVis: we increase the dataset size to $N = 10^5$, we increase the classifier kernel size to $k = 51$, and we increase the mask L1 norm penalty to $\lambda = 10$. Otherwise, hyperparameters are left at their default settings.

We see that both increasing the α_i's and increasing the number of leaking points improves the ability of AdvMask and GradVis to identify leaking points, and when either of the settings is sufficiently-large, both methods work despite the presence of Boolean masking. In contrast, the SNR technique can never identify the leaking points regardless of the dataset configuration. This illustrates the other major advantage of deep learning-based leakage localization strategies over those based on first-order statistics: DNNs work in the presence of Boolean masking because they can approximate arbitrary continuous functions, whereas
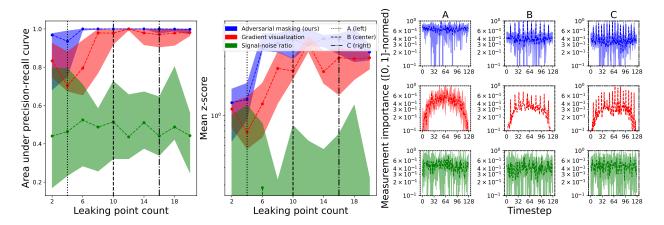
Figure 11: Boolean-masked dataset, sweeping the number of leaking points. Observe that SNR fails in all cases, and given a sufficient number of leaking points GradVis will generally identify some but not all leaking points. In contrast, AdvMask (ours) consistently identifies most or all leaking points when at least 6 are present.
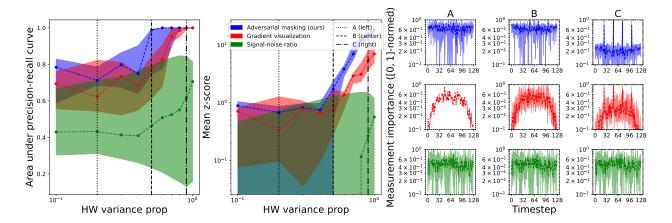

Figure 12: Boolean-masked dataset, sweeping the proportion of variance at leaking points due to the Hamming weight of the sensitive variable. Observe that SNR fails in all cases to identify second-order leaking points, whereas both AdvMask (ours) and GradVis succees when the proportion is sufficiently-large.

	DPAv4	ASVADv1-random
Classifier learning rate α	10^{-5}	10^{-4} until batch 2×10^5, then 10^{-5}
Mask learning rate α	10^{-4}	0 until batch 2×10^5, then 10^{-4}
Mask L1 norm penalty λ	10^{-2}	10^{-4}
Batch size	64	64
Number of batches	10^5	10^6
Classifier MLP width	64	64

Table 2: Hyperparameters for the experiments on the DPAv4 and ASCADv1-random datasets. Unless specified here, hyperparameters take on the same values as stated in Sec. 4.1.

first-order techniques are oblivious to this higher-order leakage because they consider single timesteps in isolation.

4.3 Experiments on real datasets

Here we present results from applying AdvMask to publicly-available AES power trace datasets. In Fig. 13 we apply it to the version of the DPAv4 context dataset [BBD$^+$14] which was released alongside [ZBHV19], which is an unprotected AES implementation (while the implementation is masked, this version of the dataset uses the masked attack points as targets, which amounts to assuming knowledge of the mask values at both profiling-time and attack time). In Fig. 14 we apply it to the synchronized ASCADv1-random dataset [BPS$^+$20], which has first-order Boolean masking.

In these experiments we cannot use the model selection strategy discussed in Sec. 3.3 because we do not have oracle knowledge of the leaking timesteps. Instead, we select models based on SNR measurements. Since the DPAv4 dataset is unprotected, SNR is a useful way to localize leakage and we compute the SNR with respect to the target variables. For the ASCADv1-random dataset, using the notation of [BPS$^+$20] we target the variable $\text{SBOX}(p[3] \oplus k[3])$, which leaks primarily through the variables $\text{SBOX}(p[3] \oplus k[3]) \oplus r_{\text{out}}$, r_{out}, $\text{SBOX}(p[3] \oplus k[3]) \oplus r[3]$, and $r[3]$, so we compute the SNR with respect to these 4 variables and take the average to get a vector $m \in \mathbb{R}^d$. For each of these datasets, we then construct 'oracle' leakage index sets using the following procedure:

1. Compute the SNR $\tilde{m} \in \mathbb{R}^d$ for a random dataset $\{(x_i, z_i)\}_{i=1}^N \in \left(\mathbb{R}^d \times \{0, \ldots, 255\}\right)^N$ where each $x_i \sim \mathcal{N}_d(0, I)$ and $z_i \sim \mathcal{U}\{0, \ldots, 255\}$ are drawn i.i.d.

2. Fit a Gaussian distribution to $\tilde{m}$: $\mu^*, \sigma^* \in \arg\max_{\mu, \sigma \in \mathbb{R}} \prod_{i=1}^d \mathcal{N}(\tilde{m}_i | \mu, \sigma)$.

3. Construct $\mathsf{I}_{\text{lkg}} := \{i = 1, \ldots, d : m_i \geq \mu^* + 3\sigma^*\}$ and $\mathsf{I}_{\neg\text{lkg}} := \{1, \ldots, d\} \setminus \mathsf{I}_{\text{lkg}}$.

Using these index sets we can then compute the mean z-score and the precision-recall AUC as described in Sec. 3.3. We take model checkpoints at the maximum values of both of these metrics, as well as after the completion of training. We make various *ad hoc* to the hyperparameter settings listed in Sec. 4.1; changed hyperparameters are listed in Tbl. 2.

The AdvMask results are mostly consistent with the SNR results on the DPAv4 dataset, as expected. On the ASCADv1 dataset the AdvMask method produces a large number of false-positive timesteps and a small number of false-negative ones, with respect to the SNR results. Early stopping seems to be critical in both cases, with AdvMask producing a large number of false-positive results if the classifier is trained for long enough to overfit to noise in the dataset. While we view these results as promising in the sense that AdvMask can produce qualitatively-similar results to SNR, the reliance on oracle knowledge of leaking points is a major limitation which will need to be addressed before AdvMask can be used for practical applications.
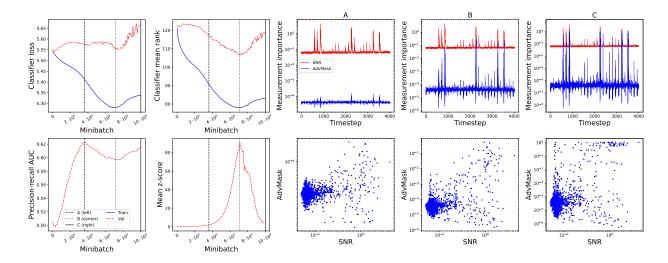
Figure 13: Results on the DPAv4 dataset. Performance on this dataset is fairly good, with AdvMask and SNR identifying similar leaking points throughout training.
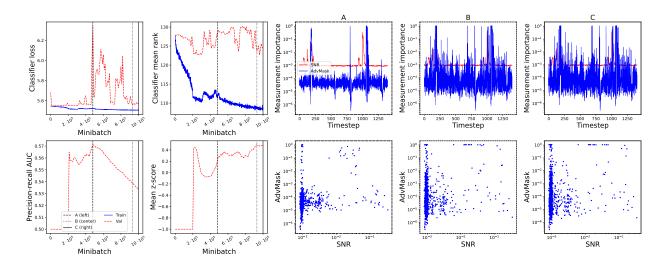

Figure 14: Results on the ASCADv1-random dataset. Performance on this dataset is highly-sensitive to classifier overfitting, and while the classifier identifies a subset of the peaks which are identified by SNR with ground truth knowledge of Boolean mask values, there are still a significant number of false-positive and false-negative leakage identifications.

5   Conclusion

We have proposed a technique for identifying the power trace timesteps which leave cryptographic implementations vulnerable to side-channel attacks, through an adversarial game played between a deep neural network (DNN)-based classifier which seeks to determine a sensitive variable from power traces, and a trainable noise generator which seeks to prevent the sensitive variable from leaking by adding as little noise as possible to the power traces. We have demonstrated on synthetic datasets that our technique has advantages over leakage localization algorithms based on first-order statistics when implementations have countermeasures such as trace desynchronization and Boolean masking. It outperforms neural network attribution algorithms when there are many sources of leakage, a strict subset of which are sufficient for accurately classifying the sensitive variable. We have also demonstrated that on the DPAv4 and ASCADv1-random datasets, our technique can produce qualitatively similar results to signal-noise ratio measurements which rely on profiling-time access to Boolean mask values.

A major barrier to practical usage of our technique is that early stopping is essential to prevent false-positive leakage identification due to overfitting by the classifier, yet there is no clear performance metric to monitor which does not rely on unrealistic oracle knowledge of which points are leaking. Therefore, future research should seek out non-oracle-based performance metrics which are correlated with the ground truth fidelity of the mask. One possible strategy would be to periodically run a template attack on the highest-leakage timesteps as indicated by the AdvMask output, and early-stop training when the template attack efficacy is maximized.

Nonetheless, this technique is a promising step towards using deep learning to localize side-channel leakage with little requirement for *a priori* knowledge about the nature of the leakage. Given the ever-shrinking need for human-specified implementation knowledge in side-channel attacks, it is critical to develop strategies for defending against side-channel attacks which make minimal assumptions about the specific timesteps or target variables which will be used to carry out attacks. Our technique satisfies this criterion because diverse DNN-based side-channel attacks can be adapted into classifiers and leveraged to identify which timesteps can be exploited to carry out a side-channel attack.

Acknowledgements

References

[AK18]   Jiwoon Ahn and Suha Kwak. Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4981–4990, 2018.

[BBD+14]   Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and improvements of the dpa contest v4 implementation. In *Security, Privacy, and Applied Cryptography Engineering: 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings 4*, pages 201–218. Springer, 2014.

[BCO04]   Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, Heidelberg, August 2004.

[BDS18]   Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[BIK+23]   Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean-Michel Picod, and Marina Zhang. Generic attacks against cryptographic hardware through long-range deep learning, 2023.

[BN06]   Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[BPS+20]   Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, June 2020.

[CDP17]   Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures – profiling attacks without pre-processing –. Cryptology ePrint Archive, Report 2017/740, 2017. https://eprint.iacr.org/2017/740.

[CRR03]   Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Heidelberg, August 2003.

[DBK+20]   Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[DDD+20]   Zhun Deng, Frances Ding, Cynthia Dwork, Rachel Hong, Giovanni Parmigiani, Prasad Patil, and Pragya Sur. Representation via representations: Domain generalization via adversarially learned invariant representations. *arXiv preprint arXiv:2006.11478*, 2020.

[DN21]   Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[DR98]   Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *International Conference on Smart Card Research and Advanced Applications*, pages 277–284. Springer, 1998.

[ES15]   Harrison Edwards and Amos Storkey. Censoring representations with an adversary. *arXiv preprint arXiv:1511.05897*, 2015.

[GJM+20]   Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

[GLP06]   Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. Templates vs. stochastic methods. In Louis Goubin and Mitsuru Matsui, editors, *CHES 2006*, volume 4249 of *LNCS*, pages 15–29. Springer, Heidelberg, October 2006.

[GPAM+14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[GRM+19]   Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019.

[HGG19]   Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 645–666. Springer, Heidelberg, August 2019.

[HL20]   Katherine Hermann and Andrew Lampinen. What shapes feature representations? exploring datasets, architectures, and training. In *Advances in Neural Information Processing Systems*, volume 33, pages 9995–10006, 2020.

[HPK+20]   Sunhee Hwang, Sungho Park, D. Kim, Mirae Do, and Hyeran Byun. Fairfacegan: Fairness-aware facial image-to-image translation. *ArXiv*, abs/2012.00282, 2020.

[IS15]   Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[JEP+21]   John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[KB15]   Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KJJ99]   Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

[Koc96]   Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

[KSH12]   Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[KUMH17]   Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

[KYK+21]   Hyeokjun Kweon, Sung-Hoon Yoon, Hyeonseong Kim, Daehee Park, and Kuk-Jin Yoon. Unlocking the potential of ordinary classifier: Class-specific adversarial erasing framework for weakly supervised semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6994–7003, 2021.

[LBBH98]   Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15]   Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LCY13]   Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[LPWK18]   Haoliang Li, Sinno Jialin Pan, Shiqi Wang, and Alex C. Kot. Domain generalization with adversarial feature learning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5400–5409, 2018.

[LTG+18]   Ya Li, Xinmei Tian, Mingming Gong, Yajing Liu, Tongliang Liu, Kun Zhang, and Dacheng Tao. Deep domain generalization via conditional invariant adversarial networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[LZC+21]   Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR TCHES*, 2021(3):235–274, 2021. https://tches.iacr.org/index.php/TCHES/article/view/8974.

[MBPK22]   Naila Mukhtar, Lejla Batina, Stjepan Picek, and Yinan Kong. Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In *Cryptographers' Track at the RSA Conference*, pages 297–321. Springer, 2022.

[MCPZ18]   David Madras, Elliot Creager, Toniann Pitassi, and Richard Zemel. Learning adversarially fair and transferable representations. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3384–3393. PMLR, 10–15 Jul 2018.

[MDP18] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. Cryptology ePrint Archive, Report 2018/1196, 2018. `https://eprint.iacr.org/2018/1196`.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[MOBW13] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? an a priori statistical power analysis of leakage detection tests. In *Advances in Cryptology-ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I 19*, pages 486–505. Springer, 2013.

[MOP08] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Advances in information security. Springer US, 2008.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. Cryptology ePrint Archive, Report 2016/921, 2016. `https://eprint.iacr.org/2016/921`.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the χ^2-test. *IACR TCHES*, 2018(1):209–237, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/838`.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against anssi's protected aes implementation on arm: end-to-end attacks with multi-task learning. *Journal of Cryptographic Engineering*, 13(2):129–147, 2023.

[Mur23] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[OWJ+22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[PPM+21] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. SoK: Deep learning-based physical side-channel analysis. Cryptology ePrint Archive, Report 2021/1092, 2021. `https://eprint.iacr.org/2021/1092`.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[SSS+17] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[SZ15] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

[vdVPB21] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In *Constructive Side-Channel Analysis and Secure Design: 11th International Workshop, COSADE 2020, Lugano, Switzerland, April 1–3, 2020, Revised Selected Papers 11*, pages 175–199. Springer, 2021.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR TCHES*, 2020(3):147–168, 2020. `https://tches.iacr.org/index.php/TCHES/article/view/8586`.

[WCL⁺20] Ping Wang, Ping Chen, Zhimin Luo, Gaofeng Dong, Mengce Zheng, Nenghai Yu, and Honggang Hu. Enhancing the performance of practical profiling side-channel attacks using conditional generative adversarial networks. *Cryptology ePrint Archive*, 2020.

[Wel47] Bernard L Welch. The generalization of 'student's'problem when several different population varlances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[WFL⁺17] Yunchao Wei, Jiashi Feng, Xiaodan Liang, Ming-Ming Cheng, Yao Zhao, and Shuicheng Yan. Object region mining with adversarial erasing: A simple classification to semantic segmentation approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1568–1576, 2017.

[WWJ⁺21] Lichao Wu, Yoo-Seung Won, Dirmanto Jap, Guilherme Perin, Shivam Bhasin, and Stjepan Picek. Explain some noise: Ablation analysis for deep learning-based physical side-channel analysis. Cryptology ePrint Archive, Report 2021/717, 2021. `https://eprint.iacr.org/2021/717`.

[XDD⁺17] Qizhe Xie, Zihang Dai, Yulun Du, Eduard Hovy, and Graham Neubig. Controllable invariance through adversarial feature learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 585–596, Red Hook, NY, USA, 2017. Curran Associates Inc.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. Cryptology ePrint Archive, Report 2019/803, 2019. `https://eprint.iacr.org/2019/803`.

[ZWF⁺18] Xiaolin Zhang, Yunchao Wei, Jiashi Feng, Yi Yang, and Thomas S Huang. Adversarial complementary learning for weakly supervised object localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1325–1334, 2018.