

Learning to Localize Leakage of Cryptographic Sensitive Variables

Jimmy Gammell¹ Anand Raghunathan¹ Abolfazl Hashemi¹ Kaushik Roy¹

Abstract

While cryptographic algorithms such as the ubiquitous Advanced Encryption Standard (AES) are secure, *physical implementations* of these algorithms in hardware inevitably ‘leak’ sensitive data such as cryptographic keys. A particularly insidious form of leakage arises from the fact that hardware consumes power and emits radiation in a manner that is statistically associated with the data it processes and the instructions it executes. Supervised deep learning has emerged as a state-of-the-art tool for carrying out *side-channel attacks*, which exploit this leakage by learning to map power/radiation measurements throughout encryption to the sensitive data operated on during that encryption. In this work we develop a principled deep learning framework for determining the relative leakage due to measurements recorded at different points in time, in order to inform *defense* against such attacks. This information is invaluable to cryptographic hardware designers for understanding *why* their hardware leaks and how they can mitigate it (e.g. by indicating the particular sections of code or electronic components which are responsible). Our framework is based on an adversarial game between a family of classifiers trained to estimate the conditional distributions of sensitive data given subsets of measurements, and a budget-constrained noise distribution which probabilistically erases individual measurements to maximize the loss of these classifiers. We demonstrate our method’s efficacy and ability to overcome limitations of prior work through extensive experimental comparison with 8 baseline methods using 3 evaluation metrics and 6 publicly-available power/EM trace datasets from AES, ECC and RSA implementations. We provide an open-source PyTorch implementation

of these experiments ([here](#)).

1. Introduction

The Advanced Encryption Standard (AES) ([Daemen & Rijmen, 1999; 2013](#)) is widely used and trusted for protecting sensitive data. For example, it is approved by the United States National Security Agency for protecting top secret information ([Committee on National Security Systems, 2003](#)), it is a major component of the Transport Layer Security (TLS) protocol ([Rescorla, 2000](#)) which underlies the security of HTTPS ([Rescorla, 2000](#)), and is used in payment card readers to secure card information before transmission to financial institutions ([Bluefin Payment Systems, 2023](#)).

AES aims to keep data secret when it is transmitted over insecure channels that are accessible to unknown and untrusted parties (e.g. via wireless transmissions which may be intercepted, or storage on hard drives which may be accessed by untrusted individuals). Prior to transmission, the data is first encoded and partitioned into a sequence of fixed-length bitstrings called *plaintexts*. Each plaintext is then *encrypted* into a *ciphertext* by applying an invertible function from a family of functions indexed by an integer called a *cryptographic key*. This family of functions is designed so that if the key is sampled uniformly at random, then the plaintext and ciphertext are marginally independent. The key is known to the sender and intended recipients of the transmission, and is kept secret from potential eavesdroppers. Thus, the intended recipients can use the key to *decrypt* the ciphertext back into the original plaintext, while eavesdroppers who possess the ciphertext but not the key learn nothing about the plaintext.

Clearly, such an algorithm is effective only if the cryptographic key remains outside of the hands of eavesdroppers. AES is believed to be ‘algorithmically secure’ in the sense that given an AES implementation with a fixed key, it is not feasible to determine the key by encrypting a chosen sequence of plaintexts and observing the resulting ciphertexts ([Mouha, 2021](#)). For reference, to our knowledge, the best known attack on the 128-bit version of AES under realistic conditions would require about 2^{125} such encryptions on average to successfully determine the key ([Tao & Wu, 2015](#)), compared to 2^{127} encryptions for a naive brute-force attack

¹Elmore School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA. Correspondence to: Jimmy Gammell <jgammell@purdue.edu>, Anand Raghunathan <araghu@purdue.edu>, Abolfazl Hashemi <abolfazl@purdue.edu>, Kaushik Roy <kaushik@purdue.edu>.

which randomly guesses and checks keys until success.

Despite the ‘algorithmic’ security of AES and other cryptographic algorithms, *physical implementations* of these algorithms in hardware inevitably ‘leak’ information about their cryptographic keys. This phenomenon, called *side-channel leakage*, occurs because hardware emits measurable physical signals that are statistically associated with the data it processes and the instructions it executes. In this work, we consider side-channel leakage due to statistical associations between sensitive data and its power consumption or electromagnetic (EM) radiation over time. Both are major security vulnerabilities for AES implementations (Kocher et al., 1999; Bronchain & Standaert, 2020; Quisquater & Samyde, 2001; Genkin et al., 2016) and tend to have similar properties due to EM radiation being dominated by the time derivative of power consumption. Note, however, that hardware emits many diverse physical signals which cause side-channel leakage, such as program/operation execution time (Kocher, 1996; Lipp et al., 2018; Kocher et al., 2019), temperature (Hutter & Schmidt, 2014), and sound due to vibration of electronic components (Genkin et al., 2014). Refer to appendix A for a layman-friendly overview with intuition-building examples.

Cryptographic implementations can be circumvented by *side-channel attacks*, which exploit side-channel leakage to learn sensitive data (e.g. cryptographic keys) of a target device. In this work, we consider *profiling* side-channel attacks, where the attacker is assumed to possess a clone of the target device and can repeatedly measure its power consumption over time while encrypting arbitrary plaintexts using arbitrary keys. Measurements of power consumption or EM radiation throughout encryption are recorded as a real vector called a *trace*, where each element encodes the measurement at a fixed point in time relative to the start of encryption. Attackers can use the clone device to model the conditional distribution of sensitive data given the trace, and can then collect traces from the target device and identify the data value which maximizes the likelihood of the data and traces according to their model.

Supervised deep learning has emerged as a state-of-the-art technique for this modeling task, achieving comparable or superior performance to prior approaches with far less data preprocessing and feature selection (Maghrebi et al., 2016; Benadjila et al., 2020; Zaid et al., 2020; Wouters et al., 2020; Bursztein et al., 2023). Older side-channel attacks were mostly based on parametric statistical tests and had major limitations such as restrictive assumptions about distributions (Chari et al., 2003; Schindler et al., 2005; Hospodar et al., 2011), requiring significant input dimensionality reduction due to poor scaling behavior (Chari et al., 2003; Archambeau et al., 2006), and limited ability to exploit statistical associations involving more than 2 random variables

(Messerges, 2000; Agrawal et al., 2005). In contrast, neural nets have proven capable of operating on raw power traces without feature selection (Lu et al., 2021; Bursztein et al., 2023) and overcoming ‘masking’ countermeasures by exploiting higher-order statistical associations (Benadjila et al., 2020; Zaid et al., 2020; Wouters et al., 2020). Consequently, deep learning is a major and growing threat to a wide assortment of security measures and evaluations that were designed with the limitations of older attacks in mind.

In this work, we seek to leverage deep learning to *defend* against side-channel attacks by identifying specific points in time at which power measurements are ‘useful’ for predicting sensitive data. Our intent is to enable the designers of implementations to understand *why* their implementations leak (e.g. by indicating the particular sections of code or electronic components which are responsible), as opposed to a mere indication of how vulnerable an implementation is to attacks. Our key contributions are:

- We propose a principled information theoretic quantity which measures the ‘leakiness’ of an individual power/EM radiation measurement, which is sensitive to arbitrarily high-order statistical associations between all available measurements and a chosen sensitive variable. Our ‘leakiness’ quantity is defined implicitly as the solution of a constrained optimization problem.
- We propose a novel deep learning algorithm called Adversarial Leakage Localization (ALL) which approximately solves this optimization problem. The algorithm is based on an adversarial game between a family of classifiers trained to estimate the conditional distributions of sensitive data given subsets of measurements, and a trainable budget-constrained noise distribution which probabilistically erases individual measurements to maximize the loss of the classifiers. Due to the budget, increasing the erasure probability of one measurement necessarily reduces that of other measurements. High erasure probabilities will be assigned to ‘high-leakage’ measurements, resulting in low probabilities being assigned to ‘low-leakage’ measurements. We can therefore view these probabilities as a quantification of the ‘leakiness’ of a measurement.
- We provide an open-source PyTorch implementation of our method as well as 8 baseline methods and 3 performance metrics, and compare them on 6 publicly-available power and EM radiation side-channel leakage datasets recorded from implementations of the AES, ECC, and RSA cryptographic standards. While there is a great deal of prior work on problems similar to ours, most of it is challenging to reproduce and benchmark due to issues such as private code and datasets, omission of important experimental details such as neural

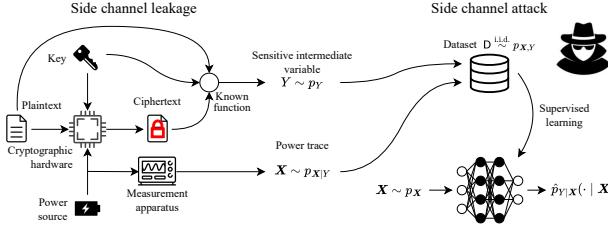


Figure 1. Diagram illustrating our probabilistic framing of side-channel leakage in the special case of power side-channel leakage from a symmetric-key (e.g. AES) cryptographic implementation. Cryptographic hardware encrypts a plaintext given a key, resulting in a ciphertext. The power consumption over time of the hardware is measured during encryption and encoded as a vector called a power trace. Consider a ‘sensitive’ intermediate variable in the cryptographic algorithm which is a known function of the key, plaintext, and ciphertext, and which gives information about the key given the plaintext and ciphertext. We view the power trace and sensitive variable as realizations of jointly-distributed random variables $\mathbf{X}, Y \sim p_{\mathbf{X}, Y}$ respectively, and side channel attacks can be carried out by using supervised learning to estimate $p_{Y|\mathbf{X}}$.

net architecture and hyperparameters, and a lack of standard quantitative performance metrics. We hope our code and procedure will facilitate reproducibility and benchmarking of future work in this area.

2. Background and setting

2.1. Probabilistic framing of power side-channel leakage

See Fig. 1 for a diagram illustrating our setting. We assume to have a cryptographic device that encrypts data in a manner dependent on some sensitive intermediate variable $y \in \mathcal{Y}$, where \mathcal{Y} is a finite set (e.g. consisting of bytestrings encoding all possible values of the variable). We assume to have some measurement apparatus that allows us to measure power/EM radiation traces during encryption, encoded as $x \in \mathbb{R}^T$ where $T \in \mathbb{Z}_{++}$ denotes the number of measurements per trace. We collect a dataset of traces and associated sensitive variable values, which we view as independent realizations of jointly-distributed random variables $\mathbf{X}, Y \sim p_{\mathbf{X}, Y}$, where p_Y is a simple known strictly-positive distribution (e.g. uniform) and $p_{\mathbf{X}|Y}$ is *a priori* unknown and dictated by factors such as the hardware, environment, and measurement setup. In this work we assume that conditional density functions $p_{\mathbf{X}|Y}(\cdot | y)$ exist and are strictly-positive for all $y \in \mathcal{Y}$, which is reasonable because power consumption usually has a ‘random’ component which is well-described by additive Gaussian noise (Mangard et al., 2007). It is natural and straightforward to use supervised learning in this setting, and most profiled power side channel attacks consist of collecting a dataset $D \stackrel{\text{i.i.d.}}{\sim} p_{\mathbf{X}, Y}$ and using supervised (deep or other-

wise) learning to model the conditional distribution $p_{Y|\mathbf{X}}$.

2.2. Quantifying leakage with mutual information

Given \mathbf{X}, Y where $\mathbf{X} = (X_1, \dots, X_T)$, we seek to assign to each X_t a scalar ‘leakiness’ quantity. Since leakage stems from the statistical association between \mathbf{X} and Y , it is natural to quantify the ‘leakiness’ of X_t as a function of the conditional mutual information (Shannon, 1948)

$$\mathbb{I}[Y; X_t | S] := \mathbb{E} \left[\log \left(\frac{p_{Y|X_t, S}(Y | X_t, S)}{p_{Y|S}(Y | S)} \right) \right] \quad (1)$$

for various sets $S \subset \{X_1, \dots, X_T\} \setminus \{X_t\}$. Intuitively, $\mathbb{I}[Y; X_t | S]$ tells us the extent to which our uncertainty about Y is reduced upon observing X_t , provided we have already observed the elements of S .

For each measurement X_t , there are 2^{T-1} possible conditioning sets S , and it is not obvious how to combine them to get a scalar ‘leakiness’ quantity γ_t^* . Clearly, we would like to have $\gamma_t^* > 0$ if $\mathbb{I}[Y; X_t] > 0$. More subtly, we desire $\gamma_t^* > 0$ if $\mathbb{I}[Y; X_t] = 0$ but $\mathbb{I}[Y; X_t | S] > 0$ for some S . This scenario, where X_t alone tells us nothing about Y but does tell us something useful *in combination* with some $X_{t'}$ where $t \neq t'$, is common in practice. For example, hardware designers often defend against side-channel attacks using masking countermeasures (Chari et al., 1999), which ensure that $\mathbb{I}[Y; X_t] \approx 0 \forall t$ but not that $\mathbb{I}[Y; \mathbf{X}] \approx 0$. Even without masking countermeasures, it is reasonable to expect high-order leakage in real hardware due to scenarios such as dependence of power consumption on Hamming distance between current and previous variable values, and low-pass filtering effects where predecessors to a leaking measurement contain information about the non-leaky component of that measurement and thus help to ‘isolate’ the leaking component. Finally, let us emphasize that we may have $\mathbb{I}[Y; X_t | S \cup S'] < \mathbb{I}[Y; X_t | S]$ if the information conveyed by $X_t | S$ is ‘redundant’ with that conveyed by S' (e.g. $\gamma_t^* = \mathbb{I}[Y; X_t | \{X_1, \dots, X_T\} \setminus \{X_t\}]$ would not be a good indicator of the ‘leakiness’ of X_t).

We will subsequently propose a technique which implies a natural definition of γ^* as a learned combination of $\mathbb{I}[Y; X_t | S]$ for $S \subset \{X_1, \dots, X_T\} \setminus \{X_t\}$. Before we do so, let us summarize existing work and its limitations through the lens of this framework.

3. Existing work and its limitations

Prior approaches to leakage localization can generally be categorized as either parametric statistics-based methods which check for pairwise associations between measurements X_t and Y , and neural net attribution methods based on first using supervised learning to model $p_{Y|\mathbf{X}}$ with a neural net, then using ‘attribution’ techniques to evaluate

the average impact of each input feature X_t on its outputs.

3.1. Parametric statistics-based methods

In the side-channel attack literature it is common to use first-order parametric statistical tests to check for pairwise associations between each measurement X_t and the sensitive variable Y . These are often used for ‘point of interest’ (feature) selection for classical side-channel attack algorithms which require low-dimensional inputs (Chari et al., 2003), but are also a simple and useful way to localize leakage. A prominent example is the signal to noise ratio (SNR) (Mangard et al., 2007):

$$\text{snr}(p_{\mathbf{X},Y}) := \frac{\text{Var}_{Y \sim p_Y} \mathbb{E}_{\mathbf{X} \sim p_{\mathbf{X}|Y}} [\mathbf{X}]}{\mathbb{E}_{Y \sim p_Y} \text{Var}_{\mathbf{X} \sim p_{\mathbf{X}|Y}} (\mathbf{X})}. \quad (2)$$

In this work we consider SNR as well as sum of squared differences (SOSD) (Chari et al., 2003) and correlation power analysis with a Hamming weight leakage model (CPA) (Brier et al., 2004) as representative examples of parametric statistics-based leakage localization, due to their popularity and proven efficacy for ‘point of interest’ selection (Fan et al., 2014).

The obvious limitation of these first-order methods is that they fail to assign leakage to measurements with high-order leakage, i.e. where $\mathbb{I}[Y; X_t] = 0$ but $\mathbb{I}[Y; X_t | S] > 0$ for some $S \subset \{X_1, \dots, X_T\} \setminus \{X_t\}$. Thus, as discussed in sec. 2.2, they are of limited use when masking (Chari et al., 1999) is present, and risk ignoring leaking points even without masking. Additionally, they make tacit assumptions about the form of $p_{\mathbf{X},Y}$. While prior work has proposed generalizations of these techniques which are sensitive to higher-order leakage, they typically either have exponential runtime in the maximum considered order of association, assume the existence of device flaws (e.g. a biased random number generator), or assume unrealistic knowledge of random intermediate variables or the points in time at which they directly influence power consumption (Messerges, 2000; Agrawal et al., 2005).

3.2. Neural net attribution

There is a great deal of prior work on localizing leakage by applying neural net interpretability techniques to trained deep neural net side-channel attackers (Masure et al., 2019; Hettwer et al., 2020; Jin et al., 2020; Zaid et al., 2020; Wouters et al., 2020; van der Valk et al., 2021; Wu & Johnson, 2021; Golder et al., 2022; Li et al., 2022; Perin et al., 2022; Schamberger et al., 2023; Yap et al., 2023; Li et al., 2024; Yap et al., 2025). Most of these techniques can be summarized as follows: 1) use supervised deep learning to model $\hat{p}_{Y|\mathbf{X}} \approx p_{Y|\mathbf{X}}$ with data, and 2) use neural net attribution techniques to estimate the average ‘influence’ of each input x_t on the conditional distribu-

tion $\hat{p}_{Y|\mathbf{X}}(\cdot | x_1, \dots, x_T)$. For example, the Gradient Visualization (GradVis) technique of Masure et al. (2019) uses the t -th element of the vector $\text{gradvis}(p_{Y|\mathbf{X}}) := \mathbb{E}_{\mathbf{X}, Y \sim p_{Y|\mathbf{X}}} [-\nabla_{\mathbf{x}} \log \hat{p}_{Y|\mathbf{X}}(Y | \mathbf{x})|_{\mathbf{x}=\mathbf{X}}]$ as an estimate of the ‘leakiness’ of X_t . The 1-occlusion technique of Zeiler & Fergus (2014), proposed as a leakage localization algorithm by Hettwer et al. (2020), uses the quantity $(\text{occl}(p_{Y|\mathbf{X}}))_t := \mathbb{E}_{\mathbf{X}, Y \sim p_{\mathbf{X},Y}} |\hat{p}_{Y|\mathbf{X}}(Y | \mathbf{X}) - \hat{p}_{Y|\mathbf{X}}(Y | \mathbf{X} \odot (1 - \delta_t))|$ where δ_t is the Kronecker delta. In this work we consider as neural net attribution baselines GradVis, 1-occlusion, saliency (Simonyan et al., 2014; Hettwer et al., 2020), layerwise relevance propagation (LRP) (Bach et al., 2015; Hettwer et al., 2020), and input * gradient (Shrikumar et al., 2017; Wouters et al., 2020). We implement GradVis using the description in (Masure et al., 2019) and the remainder using Captum (Kokhlikyan et al., 2020).

We find that these methods often detect only *some* of the leaking points, while ignoring others. We suspect the main reason for this is that methods such as GradVis and 1-occlusion, which perturb a single input to $\hat{p}_{Y|\mathbf{X}}$ while leaving the others fixed, are conceptually-similar to computing the conditional mutual information $\mathbb{I}[Y; X_t | \{X_1, \dots, X_T\} \setminus \{X_t\}]$. This quantity may be extremely small if X_t is ‘redundant’ with the other measurements, *regardless* of the ‘leakiness’ of X_t . For example, in appendix C.1 we show that if we sample $Y \sim \mathcal{U}\{-1, 1\}$ and $X_i | Y \sim \mathcal{N}(Y, \sigma^2)$ for $i = 1, 2, \dots$, the quantity $\mathbb{I}[Y; X_{n+1} | x_1, \dots, x_n]$ decays exponentially with n , at a rate which increases as σ^2 decreases. Additionally, supervised deep learning tends to exploit some but not all useful input-output associations (Geirhos et al., 2020; Hermann & Lampinen, 2020), so in practice the model $\hat{p}_{Y|\mathbf{X}}$ may not be informative about all leaking points, even if analyzed in a manner which somehow circumvents the above limitations.

The recent work by Yap et al. (2025) proposes an iterative greedy algorithm with the aim of finding a minimal-cardinality set of input features such that a trained neural net attains some minimal performance when all but this set of features is replaced by a constant. Schamberger et al. (2023) makes a similar observation to ourselves that 1-occlusion fails when there are redundant sources of leakage, and proposes a 2nd-order occlusion technique where the performance of a trained neural net is recorded while sliding a pair of windows across the trace and replacing the features in these windows with a constant, with all $\mathcal{O}(T^2)$ window positions enumerated and checked. Both methods may fail to identify leaking points if the trained neural net does not exploit all useful input-output associations, and the former has the additional limitation that there will generally be many minimal-cardinality sets which yield any given performance level, leading to failure to detect all leaking measurements. Furthermore, there are 2^T possible ‘occlusion masks’, and

due to computational constraints, iterative techniques such as these which simply enumerate and check the effect of different sets must necessarily be limited to checking only a tiny fraction of these.

We will additionally note that it is impossible in general to obtain the conditional distributions $p_{Y|S}$ for $S \subseteq \{X_1, \dots, X_T\}$ given only $p_{Y|X}$, so even ignoring the above limitations, attribution techniques which merely ‘interpret’ a trained neural net are not a mathematically-sound approach to mutual information-based leakage localization.

4. Our method: Adversarial Leakage Localization (ALL)

Given $X, Y \sim p_{X,Y}$ as defined in Section 2, where $X := (X_1, \dots, X_T)$, we seek to assign to each timestep t a scalar γ_t^* indicating the ‘amount of leakage’ about Y due to X_t . Clearly the quantities $\{\mathbb{I}[Y; X_t | S] : S \subset \{X_1, \dots, X_T\} \setminus \{X_t\}\}$ are relevant, but it is not obvious how these 2^{T-1} quantities should be combined into a single scalar measurement. Prominent prior approaches can be summarized as 1) parametric first-order statistical methods which are sensitive only to $\mathbb{I}[Y; X_t]$, and 2) neural net attribution methods which model only $p_{Y|X}$ and use this model to compute rough proxies for $\mathbb{I}[Y; X_t | \{X_1, \dots, X_T\} \setminus \{X_t\}]$ (or sometimes $\mathbb{I}[Y; X_t | S]$ for a small heuristically-chosen subset of possible S).

Here we propose a constrained optimization problem which implicitly defines an intuitively reasonable definition of γ_t^* which is sensitive to $\mathbb{I}[Y; X_t | S]$ for all S , and an adversarial deep learning algorithm which approximately solves it by modeling *all* conditional distributions $p_{Y|S}$ in an amortized manner which emphasizes those with a large impact on the objective. While our objective involves a sum over 2^T occlusion mask-like values, we can efficiently optimize it using stochastic gradient techniques with the REBAR gradient estimator (Tucker et al., 2017), which introduces a continuous relaxation-based control variate that allows us to leverage backpropagation through our neural net to get counterfactual information about each element of the mask (Maddison et al., 2017). Refer to appendix B for an extended version of this section with proofs and derivations.

4.1. Optimization problem

We define a vector $\gamma \in [0, 1]^T$ which we name the *erasure probabilities*. We use γ to parameterize a distribution over binary vectors in $\{0, 1\}^T$ as follows:

$$\mathcal{A}_\gamma \sim p_{\mathcal{A}_\gamma} \text{ where } \mathcal{A}_{\gamma,t} = \begin{cases} 1 & \text{with prob. } 1 - \gamma_t \\ 0 & \text{with prob. } \gamma_t. \end{cases} \quad (3)$$

For arbitrary vectors $x \in \mathbb{R}^T$, $\alpha \in \{0, 1\}^T$, let us denote $x_\alpha := (x_t : t = 1, \dots, T : \alpha_t = 1)$, i.e. the sub-vector

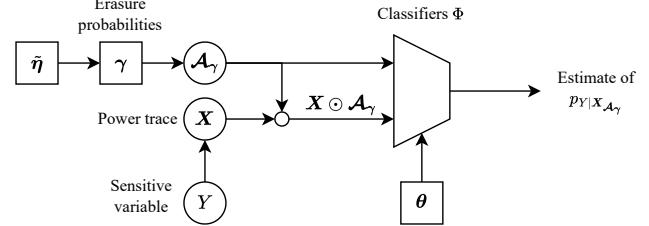


Figure 2. Our method entails estimating the conditional distributions $p_{Y|X_\alpha}$ for $\alpha \in \{0, 1\}^T$. We estimate all 2^T of these distributions using a single neural net Φ with weights θ , which takes two inputs: the binary random variable \mathcal{A}_γ , and a trace X with some of its elements randomly ‘masked’ out according to \mathcal{A}_γ .

of x containing its elements for which the corresponding element of α is 1. We can accordingly use \mathcal{A}_γ to obtain random sub-vectors $X_{\mathcal{A}_\gamma}$ of X . Note that γ_t denotes the probability that X_t will *not* be an element of $X_{\mathcal{A}_\gamma}$ (thus, ‘erasure probability’).

We assign to each element of γ a ‘cost’, defined as $c : [0, 1] \rightarrow \mathbb{R}_+ : x \mapsto \frac{x}{1-x}$. We seek to solve the constrained optimization problem

$$\min_{\gamma \in [0, 1]^T} \mathcal{L}(\gamma) := \mathbb{I}[Y; X_{\mathcal{A}_\gamma} | \mathcal{A}_\gamma] \text{ s.t. } \sum_{t=1}^T c(\gamma_t) = C \quad (4)$$

where $C > 0$ is a hyperparameter. Note that c is strictly increasing with $c(0) = 0$ and $\lim_{x \rightarrow 1^-} c(x) = \infty$. Additionally, for each t

$$\frac{\partial \mathcal{L}(\gamma)}{\partial \gamma_t} = - \sum_{\substack{\alpha \in \{0, 1\}^T \\ \alpha_t = 0}} p_{\mathcal{A}_{\gamma-t}}(\alpha_{-t}) \mathbb{I}[Y; X_t | X_\alpha]. \quad (5)$$

Informally, we see that each γ_t is ‘pushed’ towards 1 in proportion to a weighted average of $\mathbb{I}[Y; X_t | X_{\alpha_{-t}}]$ for $\alpha_{-t} \in \{0, 1\}^{T-1}$. Due to the budget constraint, increasing γ_t necessarily reduces other γ_τ , $\tau \neq t$. If γ^* is a solution to our optimization problem, we expect each γ_t^* to be closer to 1 if X_t is ‘leakier’ in the sense that it has greater mutual information with Y , conditioned on other X_τ , $\tau \neq t$. Thus, we propose using γ_t^* to measure the ‘leakiness’ of X_t .

4.2. Deep learning-based implementation

Let us re-frame this problem in a way that is amenable to standard deep learning techniques. We first remove the constraint by defining $\eta := \text{softmax}(\tilde{\eta})$ where $\tilde{\eta} \in \mathbb{R}^T$, which has the following one-to-one relationship with γ :

$$\begin{aligned} \forall t \quad c(\gamma_t) &= C\eta_t \\ \iff \gamma_t &= \text{sigmoid}(\log C + \log(\text{softmax}(\tilde{\eta})_t)). \end{aligned} \quad (6)$$

Note that the constraint is automatically satisfied for any $\tilde{\eta}$, and we can easily convert between γ and $\tilde{\eta}$ with

numerically-stable PyTorch operations.

Our optimization problem is now equivalent to

$$\min_{\tilde{\eta} \in \mathbb{R}^T} \sum_{\alpha \in \{0,1\}^T} p_{\mathcal{A}_{\gamma(\tilde{\eta})}}(\alpha) \mathbb{E} \log p_{Y|X_\alpha}(Y | X_\alpha). \quad (7)$$

We can use standard supervised deep learning techniques to approximate each conditional distribution $p_{Y|X_\alpha}$ with a neural net trained on data. Since it would be impractical to train 2^T neural nets independently, we take a similar approach to Lippe et al. (2022) (see Fig. 2) and approximate all distributions with a single net which takes as inputs our ‘erasure mask’ $\mathcal{A}_{\gamma(\tilde{\eta})}$, and a masked power trace $X \odot \mathcal{A}_{\gamma(\tilde{\eta})}$. We denote the network by $\Phi_\theta : Y \times \mathbb{R}^T \times \{0,1\}^T \rightarrow [0,1] : (y, x, \alpha) \mapsto \Phi_\theta(y | x \odot \alpha, \alpha)$ with weights $\theta \in \mathbb{R}^P$, where \odot denotes elementwise multiplication and each $\Phi_\theta(\cdot | x \odot \alpha, \alpha)$ is a probability mass function over Y . This leads to the following minimax optimization problem, which can be approximately solved with alternating minibatch stochastic gradient-style algorithms, similarly to GANs (Goodfellow et al., 2014):

$$\min_{\tilde{\eta} \in \mathbb{R}^T} \max_{\theta \in \mathbb{R}^P} \mathbb{E} \log \Phi_\theta(Y | X \odot \mathcal{A}_{\gamma(\tilde{\eta})}, \mathcal{A}_{\gamma(\tilde{\eta})}). \quad (8)$$

Informally, the inner maximization problem is a supervised learning problem where Φ_θ trains to predict labels Y given masked inputs $X \odot \mathcal{A}_\gamma$. In the outer minimization problem, the erasure probabilities γ train to maximize the loss of Φ_θ when classifying the masked inputs. Since increasing any γ_t increases the frequency at which Φ_θ is deprived of the information in X_t , doing so *in isolation* will monotonically decrease the performance of Φ_θ . However, due to the budget constraint, increasing γ_t necessarily means that some other γ_τ must be decreased. Thus, the optimal γ will have large γ_t where X_t is highly useful for Φ_θ , and small γ_t elsewhere.

Note that our objective function takes the form $\mathbb{E}_{\mathcal{A}_{\gamma(\tilde{\eta})}} f(\mathcal{A}_{\gamma(\tilde{\eta})})$ where the distribution of $\mathcal{A}_{\gamma(\tilde{\eta})}$ depends on $\tilde{\eta}$. Thus we cannot simply estimate its gradients with respect to $\tilde{\eta}$ by exchanging the order of differentiation and expectation. Instead, we use the REBAR estimator (Tucker et al., 2017), which yields unbiased low-variance gradient estimates by combining the REINFORCE estimator (Williams, 1992) with a control variate based on a continuous relaxation of $p_{\mathcal{A}_{\gamma(\tilde{\eta})}}$. This requires that we relax f to accept any $\alpha \in [0,1]^T$ rather than merely $\{0,1\}^T$. We use the relaxation

$$f(\alpha) = \mathbb{E} \log \Phi_\theta(Y | X \odot \alpha + \mathcal{E} \odot (1 - \alpha), \alpha) \\ \text{where } \mathcal{E} \sim \mathcal{N}(0,1)^T. \quad (9)$$

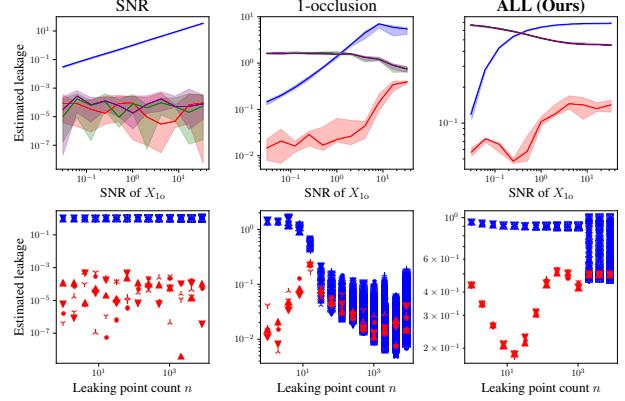


Figure 3. Experiments in simple settings where our technique succeeds but all considered baselines fail. Here for brevity we only show ALL (our method), SNR (the best-performing parametric statistical method), and 1-occlusion (the best-performing neural net attribution method). (**top row**) A dataset with the following features: X_{rand} (red) which does not leak, X_{1o} (blue) which has first-order leakage, and $X_{2o,1}$ (green) and $X_{2o,2}$ (purple), neither of which has first-order leakage, but the combination of which has *second-order* leakage. SNR (left) detects the first-order leakage but fails to detect the second-order leakage. 1-occlusion (center) and ALL (ours, right) successfully detect both the first- and second-order leakage. (**bottom row**) A dataset with a single non-leaky feature X_0 (red), and n first-order leaky features $X_i, i = 1, \dots, n$ (blue). SNR (left) successfully distinguishes between leaky and non-leaky points regardless of n , and ALL (ours, right) succeeds for n as large as 1024 before failing. However, 1-occlusion (center) fails with n as small as 32.

5. Experimental results

5.1. Simple settings where our technique succeeds but all considered baselines fail

Based on the discussions in sections 3 and 4, we expect the first-order parametric statistical algorithms to fail when high-order leakage is present, and neural net attribution methods to fail when there are many leaking features which have redundant information about the label, whereas we expect adversarial leakage localization (our method) to succeed in both settings. Here we experimentally demonstrate this using a pair of simple synthetic datasets. See Fig. 3 for abbreviated results and Appendix C.1 for full results and experimental details.

Our first dataset (top row of Fig. 3) is generated by sampling a label $Y \sim \mathcal{U}\{-1, 1\}$, an unobserved ‘Boolean mask’ $M \sim \mathcal{U}\{-1, 1\}$, and 4 input features: a nonleaky feature $X_{\text{rand}} \sim \mathcal{N}(0, 1)$, a first-order leaky feature $X_{1o} | Y \sim \mathcal{N}(Y, \sigma^2)$, and a pair of second-order leaky features $X_{2o,1} | M \sim \mathcal{N}(M, 1)$ and $X_{2o,2} | M, Y \sim \mathcal{N}(M \oplus Y, 1)$ where \oplus denotes the exclusive-or operation. We emphasize that $\mathbb{I}[Y; X_{1o}] > 0$, and while $\mathbb{I}[Y; X_{2o,1}] = \mathbb{I}[Y; X_{2o,2}] = 0$,

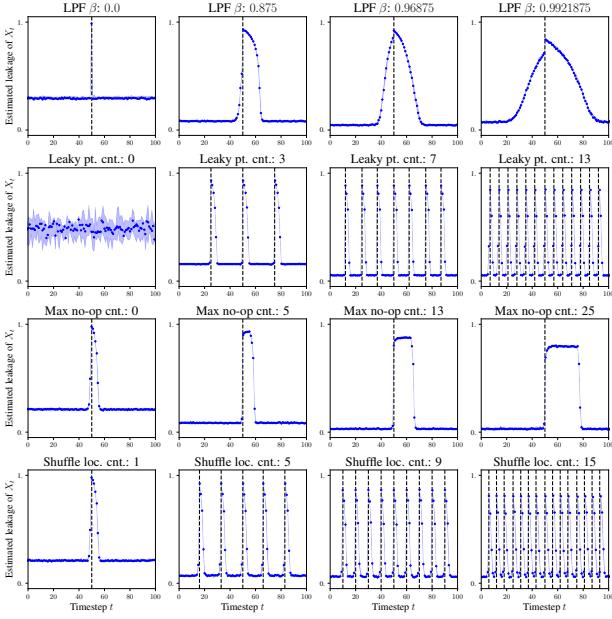


Figure 4. We validate that our adversarial leakage localization algorithm is consistent with ground truth leaky instruction timesteps on simulated AES power side-channel leakage datasets. Vertical dotted lines denote the ground-truth timestep of a leaky instruction. **(first row)** Power trace low-pass filtering strength, increasing from left to right. **(second row)** Number of leaky instructions, increasing from left to right. **(third row)** Maximum duration of random delay inserted before the leaky instruction, increasing from left to right. **(fourth row)** Leaky instruction happens randomly at one of n possible timesteps, with n increasing from left to right.

$\mathbb{I}[Y; X_{20,1}, X_{20,2}] > 0$. As expected, all first-order methods fail to detect the leakage of $X_{20,1}$ and $X_{20,2}$, whereas ALL (ours) and most neural net attribution methods successfully detect all leaky features.

Our second dataset (bottom row of Fig. 3) is generated by sampling a label $Y \sim \mathcal{U}\{-1, 1\}$, a single nonleaky feature $X_0 \sim \mathcal{N}(0, 1)$, and n leaky features $X_i \mid Y \sim \mathcal{N}(Y, 1)$, $i = 1, \dots, n$. We sweep n and observe the ability of the considered methods to distinguish X_0 from the remaining X_i 's. Consistent with our previous discussions, 1-occlusion fails for fairly small n , whereas ours works for at least $32 \times$ larger n , and SNR is unaffected by n .

5.2. Synthetic data experiments

We validate our method using simulated AES power side-channel leakage datasets we have implemented based on the Hamming weight leakage model of Mangard et al. (2007). This allows us to 1) validate our algorithm's output against known ground-truth leaky instruction timesteps, and 2) observe the effect of dataset parameters such as measurement low-pass filtering, leaky instruction count, and simulated

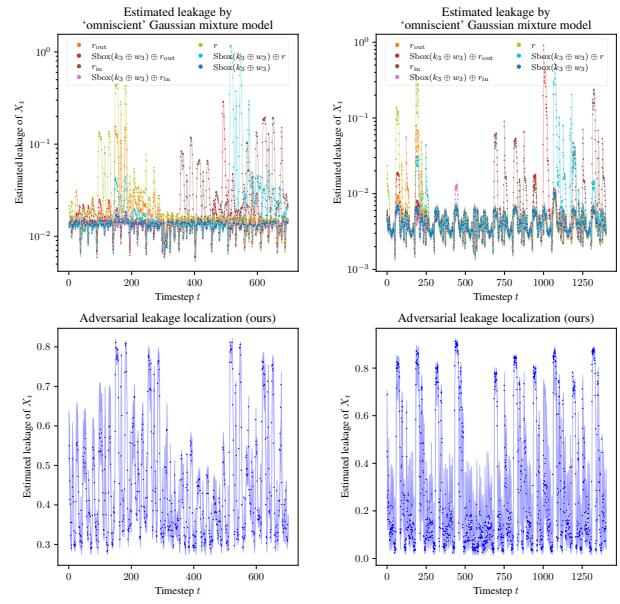


Figure 5. We successfully apply our technique to the ASCADv1 fixed- and variable-key datasets. **(top row)** Visualization of the ‘omniscient’ Gaussian mixture model (oGMM)-based leakage assessment on ASCADv1-fixed (left) and ASCADv1-variable (right). **(bottom row)** Visualization of the output of our adversarial leakage localization algorithm on ASCADv1-fixed (left) and ASCADv1-variable (right). Observe that our method largely has the same peaks as the oGMM model, despite lacking explicit knowledge of the internal Boolean masks and masked sensitive variables. In contrast, first-order parametric methods completely fail in these settings, and prior deep learning approaches detect fewer of the peaks.

countermeasures. We also run sweeps to observe the effect of our budget hyperparameter. See Fig. 4 for abbreviated results and appendix C.2 for full results and experiment-/dataset implementation details.

5.3. Real power and EM radiation leakage datasets

We evaluate our adversarial leakage localization algorithm on 6 publicly-available side-channel leakage datasets: ASCADv1 (fixed and variable key versions) (Benadjila et al., 2020), the version of DPAv4 (Zaid et al. (2020) version), AES-HD (Bhasin et al., 2020) (all of which are AES-128 implementations), OTiAiT (an EdDSA Curve2559 implementation) (Weissbart et al., 2019), and OTP (a 1024-bit RSA-CRT implementation) (Saito et al., 2022). These datasets span 3 microcontrollers and one FPGA, include both power and EM radiation measurements, and various countermeasures and targeted sensitive variables.

We compare our method to the eight neural net attribution and parametric statistical approaches listed in section 3. For fairness, all deep learning methods use a comparable neural

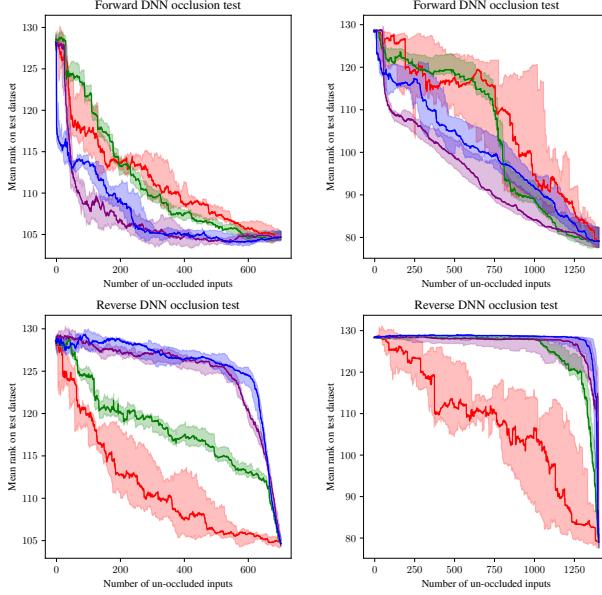


Figure 6. We evaluate considered leakage localization algorithms using the forward and reverse DNN occlusion tests on the ASCADv1-fixed (left column) and ASCADv1-variable (right column) datasets. **(top row)** Results are mixed according to the forward test, which is sensitive to false positives. Smaller area under curve is better. **(bottom row)** Our method performs well according to the reverse test, which is sensitive to false-negatives. Larger area under curve is better. **(blue)** Our adversarial leakage localization algorithm. **(purple)** The best-performing neural net attribution algorithm (input * grad for ASCADv1-fixed, GradVis for ASCADv1-variable). **(green)** The best-performing parametric statistical method (CPA for ASCADv1-fixed, SoSD for ASCADv1-variable). **(red)** Random guessing baseline.

net architecture with hyperparameters tuned via a 50-trial random search. Additionally, where available, we apply the neural net attribution methods to these pretrained Zaid et al. (2020); Wouters et al. (2020) architectures.

Our method outperforms the neural net attribution-based methods on all but one dataset and outperforms the parametric statistical methods on the datasets with mainly second-order leakage. Due to space constraints, here we present only selected results on ASCADv1 (fixed and variable key versions), with full results and experimental details deferred to appendix C.3. In particular, see table 3 for visualizations of our algorithm’s output on all considered datasets.

5.3.1. EVALUATING PERFORMANCE

We use 3 complementary performance evaluation strategies, inspired by Masure et al. (2019); Hettwer et al. (2020):

oGMM assessment We run an ‘omniscient’ leakage localization algorithm which has knowledge of the internal

randomly-generated Boolean mask variables used by the cryptographic implementations, where applicable. This is similar to the use of SNR with known Boolean masks by Masure et al. (2019). We run a sliding window over the available measurements, and for each of the possible window position we model the conditional distribution of each Boolean mask and masked sensitive variable using a Gaussian mixture model (GMM) (i.e. a Gaussian template attack (Chari et al., 2003) which uses only 1 ‘attack’ trace). We use these models to estimate the mutual information between each window of measurements and each mask + masked sensitive variable. We call the resulting leakage assessment the *oGMM assessment*. We consider a leakage localization algorithm ‘good’ if the average leakage it assigns to a window of points is positively associated with the corresponding oGMM element. See Fig. 5 for results on ASCADv1 (fixed/variable) and table 4 for a performance comparison of all algorithms according to this metric. Our method outperforms existing deep learning algorithms on 5 of the 6 considered datasets, and outperforms the parametric methods on the datasets with mainly second-order leakage.

Forward and reverse DNN occlusion tests Inspired by the KRPC and ZB-KGE techniques of Hettwer et al. (2020), here we evaluate by 1) training a deep neural net with supervised learning, 2) ranking each feature X_t according to its estimated ‘leakiness’, and 3) plotting the performance of the DNN on a test dataset when all but the top n ranked features have been replaced by a constant (‘occluded’), versus n . We carry out a *forward DNN occlusion test* where the top-ranked features are the leakiest, and a *reverse DNN occlusion test* where the top-ranked features are the least-leaky. Intuitively, the former is mainly sensitive to false positive leaky measurement detection, and the latter to false negatives. See Fig. 6 for results on ASCADv1 (fixed/variable), and tables 5 and 6, respectively, for performance comparisons of all algorithms according to these metrics. Results with the forward test are mixed, with many methods producing comparable results. We suspect it is not challenging to identify the leakiest measurements in a trace. According to the reverse test, our method outperforms all baselines on 5 of the 6 datasets while performing slightly worse than the best on the DPAv4 dataset.

6. Conclusion

We have proposed a principled and novel adversarial deep learning algorithm for localizing side-channel leakage from cryptographic implementations, based on an information theoretic definition of leakage which is sensitive to arbitrary input-output associations between side-channel emission measurements and the leaked data. Our algorithm outperforms prior deep learning-based leakage localization algorithms with respect to a diverse array of cryptographic

implementations and performance metrics, and outperforms popular parametric statistical methods on challenging implementations with primarily second-order leakage. In light of the ever-increasing efficacy of deep side-channel attacks and their ability to overcome security measures which were designed with the limitations of classical attacks in mind, our work marks a critical step towards understanding and mitigating the emerging vulnerabilities of cryptographic hardware.

Impact statement

The goal of our work is to enhance the security of cryptographic implementations against side-channel attacks by identifying the points in time at which they reveal sensitive information, thereby facilitating defenses and mitigation strategies. While any research that enhances understanding of side-channel leakage carries the risk of being repurposed for malicious purposes, we believe our work has little direct utility for carrying out attacks. We expect the net impact to be positive for cryptographic security because we introduce a new tool to defend against attacks, while relying solely on attack algorithms and datasets which are already publicly available.

Acknowledgements

We are thankful to Sakshi Choudhary, Zachary Ellis, Timur Ibrayev, Amogh Joshi, Amitangshu Mukherjee, Deepak Ravikumar, Arjun Roy, and Utkarsh Saxena for helpful discussions and feedback. The authors acknowledge the support from the Purdue Center for Secure Microelectronics Ecosystem – CSME#210205. This work was funded in part by CoCoSys JUMP 2.0 Center, supported by DARPA and SRC.

References

- Agrawal, D., Rao, J. R., Rohatgi, P., and Schramm, K. Templates as master keys. In *Cryptographic Hardware and Embedded Systems—CHES 2005: 7th International Workshop, Edinburgh, UK, August 29–September 1, 2005. Proceedings* 7, pp. 15–29. Springer, 2005.
- Archambeau, C., Peeters, E., Standaert, F. X., and Quisquater, J. J. Template attacks in principal subspaces. In *Cryptographic Hardware and Embedded Systems—CHES 2006: 8th International Workshop, Yokohama, Japan, October 10–13, 2006. Proceedings* 8, pp. 1–14. Springer, 2006.
- Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.
- Benadjila, R., Prouff, E., Strullu, R., Cagli, E., and Dumas, C. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.
- Bhasin, S., Jap, D., and Picek, S. AES HD dataset - 50 000 traces. AISyLab repository, 2020. https://github.com/AISyLab/AES_HD.
- Bluefin Payment Systems. Bluefin and ID TECH partner to deliver PCI validated Advanced Encryption Standard (AES) P2PE solution. Online, November 2023. URL <https://www.bluefin.com/news/bluefin-and-id-tech-partner-to-deliver-pci-validated-advanced-encryption-standard-aes-p2pe-solution/>.
- Brier, E., Clavier, C., and Olivier, F. Correlation power analysis with a leakage model. In Joye, M. and Quisquater, J.-J. (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2004*, pp. 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-28632-5.
- Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.
- Bronchain, O. and Standaert, F.-X. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–25, 2020.
- Bursztein, E., Invernizzi, L., Král, K., Moghimi, D., Picod, J.-M., and Zhang, M. Generic attacks against cryptographic hardware through long-range deep learning. *arXiv preprint arXiv:2306.07249*, 2023.
- Chari, S., Jutla, C. S., Rao, J. R., and Rohatgi, P. Towards sound approaches to counteract power-analysis attacks. In *Advances in Cryptology—CRYPTO’99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings* 19, pp. 398–412. Springer, 1999.
- Chari, S., Rao, J. R., and Rohatgi, P. Template attacks. In Kaliski, B. S., Koç, Ç. K., and Paar, C. (eds.), *Cryptographic Hardware and Embedded Systems - CHES 2002*, pp. 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.
- Committee on National Security Systems. Committee on National Security Systems Policy No. 15, Fact Sheet No. 1, June 2003. URL <https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/cnss15fs.pdf>.
- Daemen, J. and Rijmen, V. AES proposal: Rijndael document version 2. AES Algorithm Submission, September 1999. URL <https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf>.
- Daemen, J. and Rijmen, V. *The Design of Rijndael*. Springer Berlin, Heidelberg, March 2013.

- URL <https://link.springer.com/book/10.1007/978-3-662-04722-4>.
- Danial, J., Das, D., Golder, A., Ghosh, S., Raychowdhury, A., and Sen, S. Em-x-dl: Efficient cross-device deep learning side-channel attack with noisy em signatures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(1):1–17, 2021.
- Das, D., Golder, A., Danial, J., Ghosh, S., Raychowdhury, A., and Sen, S. X-deepsca: Cross-device deep learning side channel attack. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 1–6, 2019.
- Fan, G., Zhou, Y., Zhang, H., and Feng, D. How to choose interesting points for template attacks? *Cryptography ePrint Archive*, Paper 2014/332, 2014. URL <https://eprint.iacr.org/2014/332>.
- Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. A. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.
- Genkin, D., Shamir, A., and Tromer, E. RSA key extraction via low-bandwidth acoustic cryptanalysis. In Garay, J. A. and Gennaro, R. (eds.), *Advances in Cryptology — CRYPTO 2014*, pp. 444–461, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44371-2.
- Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., and Yarom, Y. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1626–1638, 2016.
- Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.
- Golder, A., Bhat, A., and Raychowdhury, A. Exploration into the explainability of neural network models for power side-channel analysis. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, GLSVLSI ’22, pp. 59–64, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393225. doi: 10.1145/3526241.3530346. URL <https://doi.org/10.1145/3526241.3530346>.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- Hermann, K. and Lampinen, A. What shapes feature representations? exploring datasets, architectures, and training. *Advances in Neural Information Processing Systems*, 33: 9995–10006, 2020.
- Hettwer, B., Gehrer, S., and Güneysu, T. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Paterson, K. G. and Stebila, D. (eds.), *Selected Areas in Cryptography — SAC 2019*, pp. 645–666, Cham, 2020. Springer International Publishing. ISBN 978-3-030-38471-5.
- Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., and Vandewalle, J. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293–302, 2011.
- Hutter, M. and Schmidt, J.-M. The temperature side channel and heating fault attacks. In *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27–29, 2013. Revised Selected Papers 12*, pp. 219–235. Springer, 2014.
- Jin, M., Zheng, M., Hu, H., and Yu, N. An enhanced convolutional neural network in side-channel attacks and its visualization. *arXiv preprint arXiv:2009.08898*, 2020.
- Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In Bengio, Y. and LeCun, Y. (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings*, 2014. URL <http://arxiv.org/abs/1312.6114>.
- Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In Wiener, M. (ed.), *Advances in Cryptology — CRYPTO’ 99*, pp. 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9.
- Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P’19)*, 2019.
- Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Koblitz, N. (ed.), *Advances in Cryptology — CRYPTO ’96*, pp. 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68697-2.
- Kokhlikyan, N., Miglani, V., Martin, M., Wang, E., Al-sallakh, B., Reynolds, J., Melnikov, A., Kliushkina, N., Araya, C., Yan, S., et al. Captum: A unified and generic model interpretability library for pytorch. *arXiv preprint arXiv:2009.07896*, 2020.

- Li, Y., Huang, Y., Jia, F., Zhao, Q., Tang, M., and Ren, S. A gradient deconvolutional network for side-channel attacks. *Computers & Electrical Engineering*, 98:107686, 2022.
- Li, Y., Zhu, J., Liu, Z., Tang, M., and Ren, S. Deep learning gradient visualization-based pre-silicon side-channel leakage location. *IEEE Transactions on Information Forensics and Security*, 2024.
- Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- Lippe, P., Cohen, T., and Gavves, E. Efficient neural causal discovery without acyclicity constraints. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=eYciPrLuUhG>.
- Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018.
- Lu, X., Zhang, C., Cao, P., Gu, D., and Lu, H. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 235–274, 2021.
- Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2017. URL <https://openreview.net/forum?id=S1jE5L5gl>.
- Magharebi, H., Portigliatti, T., and Prouff, E. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering: 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings* 6, pp. 3–26. Springer, 2016.
- Mangard, S., Oswald, E., and Popp, T. *Power analysis attacks*. Springer New York, NY, 1st edition, March 2007. doi: 10.1007/978-0-387-38162-6. URL <https://link.springer.com/book/10.1007/978-0-387-38162-6>.
- Masure, L., Dumas, C., and Prouff, E. Gradient visualization for general characterization in profiling attacks. In *Constructive Side-Channel Analysis and Secure Design: 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3–5, 2019, Proceedings* 10, pp. 145–167. Springer, 2019.
- Messerges, T. S. Using second-order power analysis to attack dpa resistant software. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 238–251. Springer, 2000.
- Mouha, N. Review of the Advanced Encryption Standard. NIST Interagency/Internal Report (NISTIR) 8319, National Institute of Standards and Technology, July 2021. URL <https://csrc.nist.gov/pubs/ir/8319/final>.
- Perin, G., Wu, L., and Picek, S. I know what your layers did: Layer-wise explainability of deep learning side-channel analysis. *Cryptology ePrint Archive*, Paper 2022/1087, 2022. URL <https://eprint.iacr.org/2022/1087>.
- Picek, S., Perin, G., Mariot, L., Wu, L., and Batina, L. SoK: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023.
- Quisquater, J.-J. and Samyde, D. ElectroMagnetic Analysis (EMA): Measures and counter-measures for smart cards. In Attali, I. and Jensen, T. (eds.), *Smart Card Programming and Security*, pp. 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45418-2.
- Rechberger, C. and Oswald, E. Practical template attacks. In Lim, C. H. and Yung, M. (eds.), *Information Security Applications*, pp. 440–456, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31815-6.
- Rescorla, E. HTTP over TLS. RFC 2818, May 2000. URL <https://www.rfc-editor.org/info/rfc2818>.
- Saito, K., Ito, A., Ueno, R., and Homma, N. One truth prevails: A deep-learning based single-trace power analysis on rsa–crt with windowed exponentiation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 490–526, 2022.
- Schamberger, T., Egger, M., and Tebelmann, L. Hide and seek: Using occlusion techniques for side-channel leakage attribution in cnns. In Zhou, J., Batina, L., Li, Z., Lin, J., Losiuk, E., Majumdar, S., Mashima, D., Meng, W., Picek, S., Rahman, M. A., Shao, J., Shimaoka, M., Soremekun, E., Su, C., Teh, J. S., Udovenko, A., Wang, C., Zhang, L., and Zhauniarovich, Y. (eds.), *Applied Cryptography and Network Security Workshops*, pp. 139–158, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-41181-6.
- Schindler, W., Lemke, K., and Paar, C. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems—CHES 2005*:

- 7th International Workshop, Edinburgh, UK, August 29–September 1, 2005. Proceedings 7*, pp. 30–46. Springer, 2005.
- Shannon, C. E. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- Shrikumar, A., Greenside, P., and Kundaje, A. Learning important features through propagating activation differences. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 3145–3153. PMLR, 06–11 Aug 2017. URL <https://proceedings.mlr.press/v70/shrikumar17a.html>.
- Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ICLR, 2014.
- Tao, B. and Wu, H. Improving the biclique cryptanalysis of AES. In Foo, E. and Stebila, D. (eds.), *Information Security and Privacy*, pp. 39–56, Cham, 2015. Springer International Publishing. ISBN 978-3-319-19962-7.
- Tucker, G., Mnih, A., Maddison, C. J., Lawson, J., and Sohl-Dickstein, J. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30, 2017.
- van der Valk, D., Picek, S., and Bhasin, S. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. In Bertoni, G. M. and Regazzoni, F. (eds.), *Constructive Side-Channel Analysis and Secure Design*, pp. 175–199, Cham, 2021. Springer International Publishing. ISBN 978-3-030-68773-1.
- Wang, Z., Yan, W., and Oates, T. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pp. 1578–1585. IEEE, 2017.
- Weissbart, L., Picek, S., and Batina, L. One trace is all it takes: Machine learning-based side-channel attack on eddsa. In *Security, Privacy, and Applied Cryptography Engineering: 9th International Conference, SPACE 2019, Gandhinagar, India, December 3–7, 2019, Proceedings* 9, pp. 86–105. Springer, 2019.
- Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.
- Wouters, L., Arribas, V., Gierlich, B., and Preneel, B. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 147–168, 2020.
- Wu, Y. and Johnson, J. Rethinking "batch" in batchnorm. *arXiv preprint arXiv:2105.07576*, 2021.
- Yap, T., Benamira, A., Bhasin, S., and Peyrin, T. Peek into the black-box: Interpretable neural network using sat equations in side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 24–53, 2023.
- Yap, T., Picek, S., and Bhasin, S. Occpois: Points of interest based on neural network's key recovery in side-channel analysis through occlusion. In Mukhopadhyay, S. and Stănică, P. (eds.), *Progress in Cryptology – INDOCRYPT 2024*, pp. 3–28, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-80311-6.
- Zaid, G., Bossuet, L., Habrard, A., and Venelli, A. Methodology for efficient CNN architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–36, 2020.
- Zeiler, M. D. and Fergus, R. Visualizing and understanding convolutional networks. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T. (eds.), *Computer Vision – ECCV 2014*, pp. 818–833, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10590-1.

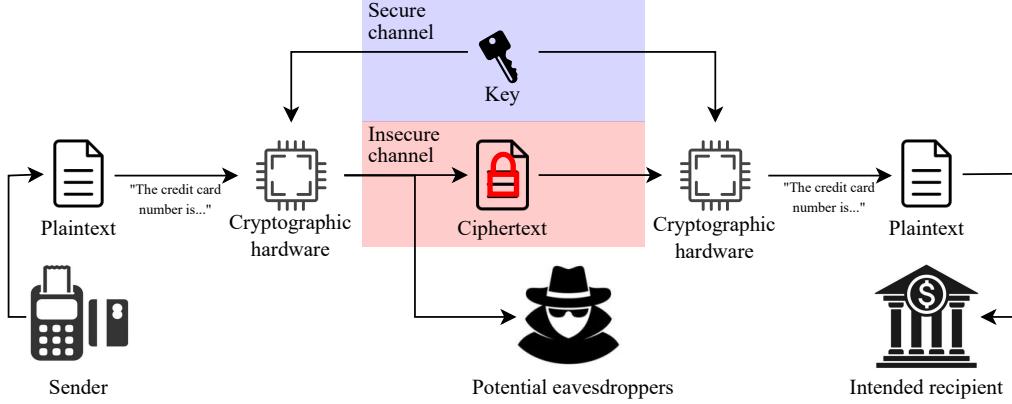


Figure 7. Diagram illustrating the main components of symmetric-key cryptographic algorithms, which enable secure transmission of data over insecure channels where it may be intercepted by eavesdroppers. The data is first partitioned and encoded as a sequence of plaintexts. Each plaintext is transformed into a ciphertext by an invertible function indexed by a cryptographic key. The key is transmitted over a secure channel to intended recipients of the data, allowing them to invert the function and recover the original plaintext. The set of functions is designed so that absent this key, the ciphertext gives no information about the plaintext. Thus, the data remains secure even if eavesdroppers have access to the ciphertext.

A. Extended background

Here we provide a high-level overview of the AES algorithm and power side-channel attacks aimed at a machine learning audience. Since our algorithm views the cryptographic algorithm and hardware as a black box to be characterized with data, a deep understanding is not necessary to understand and appreciate our work. Thus, we omit many details and aim to impart an intuitive understanding of these topics. Interested readers may refer to Daemen & Rijmen (2013) for a detailed introduction to the AES algorithm, to Mangard et al. (2007) for a detailed introduction to power side-channel attacks, and to Picek et al. (2023) for a survey of supervised deep learning-based power side-channel attacks on AES implementations.

A.1. Cryptographic algorithms

Data is often transmitted over insecure channels which leave it accessible not only to intended recipients, but also to unknown and untrusted parties. For example, when a signal is wirelessly transmitted from one antenna to another, an eavesdropper could set up a third antenna between the two and intercept the signal. Alternately, data stored on a hard drive by one user of a computer may be accessed by a different user. Cryptographic algorithms aim to preserve the privacy of data under such circumstances by transforming it so that it is meaningful only in combination with additional data which is known to its intended recipients but not to the untrusted parties.

In this work we mostly focus on the advanced encryption standard (AES), which is a symmetric-key cryptographic algorithm. See Fig. 7 for a diagram illustrating the important components of such algorithms. The unencrypted data to be transmitted is encoded and partitioned into a sequence of fixed-length bitstrings called *plaintexts*. The cryptographic algorithm encrypts each plaintext into a *ciphertext* by applying an invertible function from a set of functions indexed by an integer called the *cryptographic key*. This set of functions is designed so that if one were to sample a key and plaintext uniformly at random from the sets of all possible keys and plaintexts, then the plaintext and ciphertext would be marginally independent. Thus, such an algorithm may be used to securely transmit data by ensuring that the sender and recipient of the data know a shared key,¹ and that the key is kept secret from all potential eavesdroppers on the data.

¹The key is typically shared using an asymmetric-key cryptographic algorithm such as RSA or ECC. Asymmetric-key cryptography is slow and resource-intensive, so when a sufficiently-large amount of data must be transmitted, it is more-efficient to share the key with an asymmetric-key algorithm and then transmit data using a symmetric-key algorithm than to simply transmit the data with an asymmetric-key algorithm.

A.2. Side-channel attacks

Many symmetric-key cryptographic algorithms are believed to be secure in the sense that it is not feasible to determine their cryptographic key by encrypting known plaintexts and observing the resulting ciphertexts. Any such algorithm with a finite number of possible keys is vulnerable to ‘brute-force’ attacks based on arbitrarily guessing and checking keys until success, but doing so requires checking half of all possible keys in the average case, which is unrealistic for algorithms such as AES which has either 2^{128} , 2^{192} , or 2^{256} possible keys. To our knowledge the best known such attack against AES reduces the required number of guesses by less than a factor of 8 compared to a naive brute force attack (Mouha, 2021; Tao & Wu, 2015).

However, while algorithms may be secure when considering only their intended inputs and outputs, *hardware executing these algorithms* will inevitably emit measurable physical signals which are statistically associated with their intermediate variables and operations. Examples of such signals include a device’s power consumption over time (Kocher et al., 1999), the amount of time it takes to execute a program or instruction (Kocher, 1996; Lipp et al., 2018; Kocher et al., 2019), electromagnetic radiation it emits (Quisquater & Samyde, 2001; Genkin et al., 2016), and sound due to vibrations of its electronic components (Genkin et al., 2014). This phenomenon is called *side-channel leakage*, and can be exploited to determine sensitive data such as a cryptographic key through *side-channel attacks*.

As a simple example of side-channel leakage, consider the following Python function which checks whether a password is correct:

```
def is_correct(provided_password: str, correct_password: str) -> bool:
    if len(provided_password) != len(correct_password):
        return False
    for i in range(len(provided_password)):
        if provided_password[i] != correct_password[i]:
            return False
    return True
```

Suppose the password consists of n characters, each with c possible values. Consider an attacker seeking to determine the correct password by feeding various guessed passwords until the function returns `True`. Naively, the attacker could simply guess and check all possible m -length passwords for $m = 1, \dots, n$. This would require $\mathcal{O}(c^n)$ calls to the function, which would be extremely costly for realistically-large c and n . However, an attacker with knowledge of the function’s implementation could dramatically reduce this cost by observing that the function’s *execution time* depends on `correct_password`. Because the function exits immediately if `len(provided_password) != len(correct_password)`, the attacker can determine the length of `correct_password` in $\mathcal{O}(n)$ time by feeding increasing-length guesses to `is_correct` until its execution time increases. Next, because `is_correct` exits the first time it detects an incorrect character, the attacker can sequentially determine each of the characters of `correct_password` by checking all c possible values of each character and noting that the correct value leads to an increase in execution time. Thus, although `is_correct` is secure against attackers which use only its intended inputs and outputs, it provides *essentially no security* against attackers which measure its execution time.

In this work we focus on side-channel leakage due to the power consumption over time of a device. A device’s power consumption is inevitably statistically-associated with the operations it executes and the data it operates on, because these dictate which components are active and the order and manner in which they operate. There are many types of components with different functionality, and components with the same intended functionality are not identical due to imperfect manufacturing processes. These differences impact power consumption. While in general the association between power consumption and data is multifactorial and difficult to describe, in Fig. 8 we illustrate a simple relationship which accounts for a significant portion of the leakage in a device characterized by Mangard et al. (2007).

A.3. Power side-channel attacks on AES implementations

Side-channel attacks are techniques which exploit side-channel leakage to learn sensitive information such as cryptographic keys. There are many categories of attacks, but in this work we focus on a category called *profiled* side-channel attacks on symmetric-key cryptographic algorithms. These attacks assume that the ‘attacker’ has access to a clone of the actual cryptographic device to be attacked, and the ability to encrypt arbitrary plaintexts with arbitrary cryptographic keys, observe the resulting ciphertexts, and measure the side-channel leakage during encryption. In practice, these assumptions almost certainly overestimate the capabilities of attackers – for example, while in some cases an attacker could plausibly identify

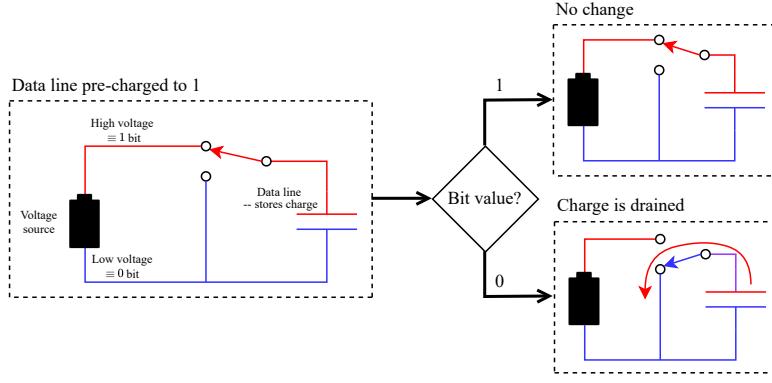


Figure 8. Diagram illustrating one reason there is power side-channel leakage in the device characterized by Mangard et al. (2007, ch. 4). Data is transmitted over a bus consisting of multiple wires, with one wire representing each bit. Each wire represents a 0 bit as some prescribed ‘low’ voltage and a 1 bit as a ‘high’ voltage. Energy is consumed when the voltage of a wire changes from low to high because positive and negative charges, which are attracted to one-another, must be separated to create a high concentration of positive charge on the wire. When ‘writing’ data to the bus, this particular device first ‘pre-charges’ all wires to 1, then drains charge from the wires which should represent 0. Thus, because the 0’s must be changed to 1’s before the next write, energy is consumed in proportion to the number of 0’s, thereby creating a statistical association between the device’s power consumption and the data it operates on.

the hardware and source code of a cryptographic implementation, purchase copies of this hardware, program them with the source code, and characterize these devices, the nature of the side-channel leakage of these purchased copies would differ from those of the actual device due to PVT (pressure, voltage, temperature) variations (e.g. due to imperfect manufacturing processes, environment, and measurement setup). It has been demonstrated that profiled side-channel attacks can be effective despite this, especially when numerous copies of the target hardware are used for profiling (Das et al., 2019; Danial et al., 2021). Regardless, this type of attack provides an upper bound on the vulnerability of a device to side-channel attacks, which is a useful metric for hardware designers.

While there are diverse types of profiled side-channel attacks, at a high level the following steps encompass the important elements of these attacks:

1. Select some ‘sensitive’ intermediate variable of the cryptographic algorithm which reveals the cryptographic key (or part of it).
2. Compile a dataset of (side-channel leakage, intermediate variable) pairs by repeatedly randomly selecting a key and plaintext, encrypting the plaintext using the key and recording the resulting ciphertext and side-channel leakage during encryption, and computing the intermediate variable based on knowledge of the cryptographic algorithm.
3. Use supervised learning to train a parametric function approximator to predict intermediate variables from recordings of side-channel leakage during encryption.
4. Measure side-channel leakage during encryptions by the actual target device. Use the trained predictor to predict sensitive variables from side-channel leakage. Potentially, these predictions can be combined to get a better estimate of the key.

In the case of power side-channel attacks on AES, it is generally infeasible to directly target the cryptographic key because care is taken by hardware designers to prevent it from directly influencing power consumption. Instead, it is common to target an intermediate variable called the `SubBytes` output, which is computed as

$$y := \text{Sbox}(k \oplus w) \quad (10)$$

where $k \in \{0, 1\}^{n_{\text{bits}}}$ is the key, $w \in \{0, 1\}^{n_{\text{bits}}}$ is the plaintext, $n_{\text{bits}} \in \mathbb{Z}_{++}$ is the number of bits of the key and plaintext, \oplus is the bitwise exclusive-or operation, and $\text{Sbox} : \{0, 1\}^{n_{\text{bits}}} \rightarrow \{0, 1\}^{n_{\text{bits}}}$ is an invertible function which is widely known and the same for all AES implementations. Note that if the plaintext is known, the key can be computed as

$$k = \text{Sbox}^{-1}(y) \oplus w. \quad (11)$$

Additionally, it is common to independently target subsets of the bits of the cryptographic key (e.g. the individual bytes). This is reasonable because many devices can operate on only a subset of the bytes in a single machine instruction, in which case one gains little by attacking more than this number of bytes simultaneously. Even in devices for which this is not the case, subsets of bits will still be statistically associated with power consumption.

A.3.1. TEMPLATE ATTACK: EXAMPLE OF A CLASSICAL PROFILED SIDE-CHANNEL ATTACK

In order to underscore the advantage of deep learning over previous side-channel attack algorithms, we will here describe the template attack algorithm of Chari et al. (2003), variations of which are the state-of-the-art non-deep learning based attacks. The attack is based on modeling the joint distribution of power consumption and intermediate variable as a Gaussian mixture model, as described in algorithm 1.

Algorithm 1: The Gaussian template attack algorithm of Chari et al. (2003)

```

Input: Profiling (training) dataset  $D := \{(\mathbf{x}^{(n)}, y^{(n)} : n \in [1..N]\} \subset \mathbb{R}^T \times \{0, 1\}^{n_{\text{bits}}}$ , attack (testing) dataset
 $D_{\text{attack}} := \{(\mathbf{x}_{\text{a}}^{(n)}, w_{\text{a}}^{(n)}) : n \in [1..N_{\text{a}}]\} \subset \mathbb{R}^T \times \{0, 1\}^{n_{\text{bits}}}$ , ‘points of interest’
 $T_{\text{poi}} := \{t_m : m = 1, \dots, \tilde{T}\} \subset [1..T]$ 
Output: Predicted key  $k^*$ 

1 Function  $\text{get\_y}(k, w)$ 
2   return  $\text{Sbox}(k \oplus w)$  // calculate intermediate variable for given key
3 for  $n \in [1..N]$  do
4    $\tilde{\mathbf{x}}^{(n)} \leftarrow (\tilde{x}_{t_m}^{(n)} : m = 1, \dots, \tilde{T})$  // prune power traces to ‘points of interest’
5 for  $y \in \{0, 1\}^{n_{\text{bits}}}$  do
6   // fit a multivariate Gaussian mixture model to the training dataset
7    $D_y \leftarrow \{\tilde{\mathbf{x}}^{(n)} : n \in [1..N], y^{(n)} = y\}$ 
8    $N_y \leftarrow |D_y|$ 
9    $\boldsymbol{\mu}_y \leftarrow \frac{1}{N_y} \sum_{\tilde{\mathbf{x}} \in D_y} \tilde{\mathbf{x}}$ 
10   $\boldsymbol{\Sigma}_y \leftarrow \frac{1}{N_y - 1} \sum_{\tilde{\mathbf{x}} \in D_y} (\tilde{\mathbf{x}} - \boldsymbol{\mu}_y)(\tilde{\mathbf{x}} - \boldsymbol{\mu}_y)^{\top}$ 
11 for  $n \in [1..N_{\text{a}}]$  do
12    $\tilde{\mathbf{x}}_{\text{a}}^{(n)} \leftarrow (\tilde{x}_{\text{a}, t_m}^{(n)} : m = 1, \dots, \tilde{T})$  // prune power traces of attack dataset
13   // predict key value which maximizes log-likelihood of attack dataset
14    $k^* \leftarrow \arg \max_{k \in \{0, 1\}^{n_{\text{bits}}}} \sum_{n=1}^{N_{\text{a}}} \left[ \log \mathcal{N} \left( \tilde{\mathbf{x}}^{(n)} ; \boldsymbol{\mu}_{\text{get\_y}(k, w_{\text{a}}^{(n)})}, \boldsymbol{\Sigma}_{\text{get\_y}(k, w_{\text{a}}^{(n)})} \right) + \log N_{\text{get\_y}(k, w_{\text{a}}^{(n)})} \right]$ 
15 return  $k^*$ 

```

Note that this algorithm assumes that the joint distribution is well-described by a Gaussian mixture model, which may not hold in practice. Additionally, due to the near-cubic runtime of the matrix inversion of each $\boldsymbol{\Sigma}_y$ required to compute the Gaussian density functions, this algorithm requires pruning power traces down to a small number of ‘high-leakage’ timesteps. Follow-up work (Rechberger & Oswald, 2005) proposed performing principle component analysis on the traces and modeling the coefficients of the top principle components rather than individual timesteps. Nonetheless, these constraints mean that the efficacy of this attack is contingent on simplifying assumptions and judgement of which points are ‘leaky’ using simple statistical techniques and implementation knowledge, limiting its usefulness as a way for hardware designers to evaluate the amount of side-channel leakage from their device.

A.3.2. PRACTICAL PROFILED DEEP LEARNING SIDE-CHANNEL ATTACKS ON AES IMPLEMENTATIONS

Here we will give a common and concrete setting and method for performing profiled power side-channel attacks on AES implementations, which is used for all of our experiments.

Consider an AES-128 implementation, which has a 128-bit cryptographic key and plaintext. Typically, attackers target each of the 16 bytes of the key independently rather than attacking the full key at once. This practice tacitly assumes that the bytes of the sensitive variable are statistically-independent given the power trace, which is reasonable because many AES

operations (including those which are commonly targeted) are performed independently on the individual bytes. Thus, it is a convenient way to simplify the attack with only a small performance degradation.

Additionally, it is difficult and uncommon to try to directly map power traces to associated cryptographic keys, because great care is taken by hardware designers to ensure that the key does not directly impact power consumption. Instead, attackers generally target ‘sensitive’ intermediate variables which unavoidably directly impact power consumption and can be combined with the plaintext and ciphertext to learn the key. We consider one such intermediate variable which is referred to as the first `SubBytes` output, and is equal to

$$y := \text{Sbox}(k \oplus w), \quad (12)$$

where $k \in \{0, 1\}^8$ is one byte of the cryptographic key, $w \in \{0, 1\}^8$ is the corresponding byte of the plaintext, \oplus denotes the bitwise exclusive-or operation, and $\text{Sbox} : \{0, 1\}^8 \rightarrow \{0, 1\}^8$ is an invertible function which is publicly-available and the same for all AES implementations. Note that if w is known, as is assumed in the profiled side-channel attack setting, then k can be recovered as

$$k = w \oplus \text{Sbox}^{-1}(y). \quad (13)$$

In the context of profiled power side-channel analysis, one assumes to have a ‘profiling’ dataset (i.e. a training dataset) and an ‘attack’ dataset (i.e. a test dataset). Suppose we target n_{bytes} bytes of the sensitive variable. In our setting, the profiling dataset consists of ordered pairs of power traces and their associated sensitive intermediate variables:

$$\mathcal{D} := \left\{ (\mathbf{x}^{(n)}, y^{(n)}) : n \in [1 \dots N] \right\} \subset \mathbb{R}^T \times \{0, 1\}^{n_{\text{bytes}} \times 8} \quad (14)$$

and the attack dataset consists of ordered pairs of power traces and their associated plaintexts:

$$\mathcal{D}_a := \left\{ (\mathbf{x}_a^{(n)}, w_a^{(n)}) : n \in [1 \dots N_a] \right\} \subset \mathbb{R}^T \times \{0, 1\}^{n_{\text{bytes}} \times 8}. \quad (15)$$

Many works prove the concept of their approaches by targeting only a single byte of the sensitive variable. When multiple bytes are targeted, it is common to either train a separate neural network for each byte of the sensitive variable, or to amortize the cost of targeting these bytes by training a single neural network with a shared backbone and a separate head for each byte. In this work we exclusively target single bytes, though it would be straightforward to extend our approach to the multitask learning setting.

Consider a neural network architecture $\Phi : \mathbb{Y} \times \mathbb{R}^T \times \mathbb{R}^P \rightarrow \mathbb{R}_+ : (y, \mathbf{x}, \boldsymbol{\theta}) \mapsto \Phi(y \mid \mathbf{x}; \boldsymbol{\theta})$, where each $\Phi(\cdot \mid \mathbf{x}; \boldsymbol{\theta})$ is a probability mass function over \mathbb{Y} . In the case of a multi-headed network with each head independently predicting a single byte, we compute this probability mass of $y \in \mathbb{Y}$ as the product of the mass assigned to each of its bytes. We train the network by approximately solving the optimization problem

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}(\boldsymbol{\theta}) := \frac{1}{N} \sum_{n=1}^N \log \Phi(y^{(n)} \mid \mathbf{x}^{(n)}; \boldsymbol{\theta}). \quad (16)$$

Given $\hat{\boldsymbol{\theta}} \in \arg \max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}(\boldsymbol{\theta})$, we then identify the key which maximizes our estimated likelihood of our attack dataset and key as follows:

$$\hat{k} \in \arg \max_{k \in \{0, 1\}^{n_{\text{bytes}} \times 8}} \sum_{n=1}^{N_a} \log \Phi \left(\left(\text{Sbox}(k_i \oplus w_{a,i}^{(n)}) : i = 1, \dots, n_{\text{bytes}} \right) \mid \mathbf{x}_a^{(n)}; \hat{\boldsymbol{\theta}} \right) \quad (17)$$

where we denote by k_i and $w_i^{(n)}$ the individual bytes of k and $w^{(n)}$.

B. Extended method with derivations

Given $\mathbf{X}, Y \sim p_{\mathbf{X}, Y}$ as defined in Section 2, where $\mathbf{X} = (X_1, \dots, X_T)$, we seek to assign to each timestep t a scalar γ_t^* indicating the ‘amount of leakage’ of information about Y due to power measurement X_t . Clearly the quantities $\{\mathbb{I}[Y; X_t \mid S] : S \subset \{X_1, \dots, X_T\} \setminus \{X_t\}\}$ are relevant, but it is not obvious how they should be weighted into a single scalar measurement. Here we propose an optimization problem which implicitly defines such a weighting scheme, as well as a deep learning-based algorithm to approximately solve it.

B.1. Optimization problem

We define a vector $\gamma \in [0, 1]^T$ which we name the *erasure probabilities*. We use γ to parameterize a distribution over binary vectors in $\{0, 1\}^T$ as follows:

$$\mathbf{A}_\gamma \sim p_{\mathbf{A}_\gamma} \quad \text{where} \quad A_{\gamma,t} = \begin{cases} 1 & \text{with probability } 1 - \gamma_t \\ 0 & \text{with probability } \gamma_t, \end{cases} \quad (18)$$

i.e. its elements are independent Bernoulli random variables where the t -th element has parameter $p = 1 - \gamma_t$. For arbitrary vectors $\mathbf{x} \in \mathbb{R}^T$, $\alpha \in \{0, 1\}^T$, let us denote $\mathbf{x}_\alpha := (x_t : t = 1, \dots, T : \alpha_t = 1)$, i.e. the sub-vector of \mathbf{x} containing its elements for which the corresponding element of α is 1. We can accordingly use \mathbf{A}_γ to obtain random sub-vectors $\mathbf{X}_{\mathbf{A}_\gamma}$ of \mathbf{X} . Note that γ_t denotes the probability that X_t will *not* be an element of $\mathbf{X}_{\mathbf{A}_\gamma}$ (thus, ‘erasure probability’).

We assign to each element of γ a ‘cost,’ defined as

$$c : [0, 1] \rightarrow \mathbb{R}_+ : x \mapsto \frac{x}{1-x}. \quad (19)$$

We seek to solve the constrained optimization problem

$$\min_{\gamma \in [0, 1]^T} \quad \mathcal{L}_{\text{ideal}}(\gamma) := \mathbb{I}[Y; \mathbf{X}_{\mathbf{A}_\gamma} | \mathbf{A}_\gamma] \quad \text{such that} \quad \sum_{t=1}^T c(\gamma_t) = C \quad (20)$$

for hyperparameter $C > 0$. Note that c is strictly-increasing with $c(0) = 0$ and $\lim_{x \rightarrow 1^-} c(x) = \infty$ so that for finite C any optimal γ will be in $[0, 1]^T$, and increasing some γ_t necessarily reduces some of the other γ_τ , $\tau \neq t$. Additionally, for each t we can re-write our objective as

$$\mathcal{L}_{\text{ideal}}(\gamma) = \sum_{\alpha \in \{0, 1\}^T} p_{\mathbf{A}_\gamma}(\alpha) \mathbb{I}[Y; \mathbf{X}_\alpha] \quad (21)$$

$$= \sum_{\substack{\alpha \in \{0, 1\}^T \\ \alpha_t = 0}} p_{\mathbf{A}_{\gamma,-t}}(\alpha_{-t}) [(1 - \gamma_t) \mathbb{I}[Y; X_t, \mathbf{X}_\alpha] + \gamma_t \mathbb{I}[Y; \mathbf{X}_\alpha]] \quad (22)$$

$$= \sum_{\substack{\alpha \in \{0, 1\}^T \\ \alpha_t = 0}} p_{\mathbf{A}_{\gamma,-t}}(\alpha_{-t}) [\mathbb{I}[Y; X_t, \mathbf{X}_\alpha] - \gamma_t \mathbb{I}[Y; X_t | \mathbf{X}_\alpha]], \quad (23)$$

which implies

$$\frac{\partial \mathcal{L}_{\text{ideal}}(\gamma)}{\partial \gamma_t} = - \sum_{\substack{\alpha \in \{0, 1\}^T \\ \alpha_t = 0}} p_{\mathbf{A}_{\gamma,-t}}(\alpha_{-t}) \mathbb{I}[Y; X_t | \mathbf{X}_\alpha]. \quad (24)$$

B.2. Estimating mutual information with deep neural nets

We cannot solve equation 20 directly because we lack an expression for $p_{\mathbf{X}, Y}$. Here we derive an equivalent optimization problem which uses deep learning to characterize $p_{\mathbf{X}, Y}$ using data.

Consider the family $\{\Phi_\alpha\}_{\alpha \in \{0, 1\}^T}$ with each element a deep neural net

$$\Phi_\alpha : \mathbb{Y} \times \mathbb{R}^{\sum_{t=1}^T \alpha_t} \times \mathbb{R}^P \rightarrow [0, 1] : (y, \mathbf{x}_\alpha, \theta) \mapsto \Phi_\alpha(y | \mathbf{x}_\alpha; \theta). \quad (25)$$

We assume each $\Phi_\alpha(\cdot | \mathbf{x}; \theta)$ is a probability mass function over \mathbb{Y} (e.g. the neural net has a softmax output activation). We define the optimization problem

$$\min_{\gamma \in [0, 1]^T} \max_{\theta \in \mathbb{R}^P} \quad \mathcal{L}_{\text{adv}}(\gamma, \theta) := \mathbb{E} \log \Phi_{\mathbf{A}_\gamma}(Y | \mathbf{X}_{\mathbf{A}_\gamma}; \theta) \quad \text{such that} \quad \sum_{t=1}^T c(\gamma_t) = C. \quad (26)$$

Proposition B.1. Consider the objective function \mathcal{L}_{adv} of equation 26. Suppose there exists some $\boldsymbol{\theta}^* \in \mathbb{R}^P$ such that $\Phi_{\alpha}(y | \mathbf{x}_{\alpha}; \boldsymbol{\theta}^*) = p_{Y|\mathbf{X}_{\alpha}}(y | \mathbf{x}_{\alpha})$ for all $\alpha \in \{0, 1\}^T$, $\mathbf{x} \in \mathbb{R}^T$, $y \in \mathbb{Y}$. Then

$$\boldsymbol{\theta}^* \in \arg \max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}) \quad \forall \boldsymbol{\gamma} \in [0, 1]^T. \quad (27)$$

Furthermore, for all $y \in \mathbb{Y}$ and for all $\boldsymbol{\gamma} \in [0, 1]^T$, $\alpha \in \{0, 1\}^T$ such that $p_{\mathcal{A}_{\gamma}}(\alpha) > 0$,

$$\Phi_{\alpha}(y | \mathbf{x}_{\alpha}; \hat{\boldsymbol{\theta}}) = p_{Y|\mathbf{X}_{\alpha}}(y | \mathbf{x}_{\alpha}) \quad p_{\mathbf{X}}\text{-almost surely} \quad \forall \hat{\boldsymbol{\theta}} \in \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}). \quad (28)$$

Proof. Note that since each $\Phi_{\alpha}(\cdot | \mathbf{x}, \boldsymbol{\theta})$ is a probability mass function over \mathbb{Y} , by Gibbs' inequality we have

$$\mathbb{E} \log \Phi_{\alpha}(Y | \mathbf{X}_{\alpha}; \boldsymbol{\theta}) \leq \mathbb{E} \log p_{Y|\mathbf{X}_{\alpha}}(Y | \mathbf{X}_{\alpha}) \quad \forall \alpha \in \{0, 1\}^T, \boldsymbol{\theta} \in \mathbb{R}^P. \quad (29)$$

Thus,

$$\mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}^*) \geq \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}) \quad \forall \boldsymbol{\theta} \in \mathbb{R}^P, \boldsymbol{\gamma} \in [0, 1]^T, \quad (30)$$

which implies the first claim.

Next, consider some fixed $\boldsymbol{\gamma} \in [0, 1]^T$ and $\hat{\boldsymbol{\theta}} \in \arg \min_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta})$. We must have $\mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \hat{\boldsymbol{\theta}}) = \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}^*)$. Thus,

$$0 = \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}^*) - \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \hat{\boldsymbol{\theta}}) \quad (31)$$

$$= \mathbb{E} \left[\log p_{Y|\mathbf{X}_{\alpha}}(Y | \mathbf{X}_{\alpha}) - \log \Phi_{\alpha}(Y | \mathbf{X}_{\alpha}; \hat{\boldsymbol{\theta}}) \right] \quad (32)$$

$$= \sum_{\alpha \in \{0, 1\}^T} p_{\mathcal{A}_{\gamma}}(\alpha) \mathbb{E} \left[\log p_{Y|\mathbf{X}_{\alpha}}(Y | \mathbf{X}_{\alpha}) - \log \Phi_{\alpha}(Y | \mathbf{X}_{\alpha}; \hat{\boldsymbol{\theta}}) \right]. \quad (33)$$

By Gibbs' inequality, each of the expectations in the summation is nonnegative, which implies that whenever $p_{\mathcal{A}_{\gamma}}(\alpha) > 0$ we must have

$$0 = \mathbb{E} \left[\log p_{Y|\mathbf{X}_{\alpha}}(Y | \mathbf{X}_{\alpha}) - \log \Phi_{\alpha}(Y | \mathbf{X}_{\alpha}; \hat{\boldsymbol{\theta}}) \right] \quad (34)$$

$$= \int_{\mathbb{R}^{\sum_{t=1}^T \alpha_t}} p_{\mathbf{X}_{\alpha}}(\mathbf{x}_{\alpha}) \mathbb{KL} \left[p_{Y|\mathbf{X}_{\alpha}}(\cdot | \mathbf{x}_{\alpha}) \| \Phi_{\alpha}(\cdot | \mathbf{x}_{\alpha}; \hat{\boldsymbol{\theta}}) \right] d\mathbf{x}_{\alpha}. \quad (35)$$

Since $\mathbb{KL} \left[p_{Y|\mathbf{X}_{\alpha}}(\cdot | \mathbf{x}_{\alpha}) \| \Phi_{\alpha}(\cdot | \mathbf{x}_{\alpha}; \hat{\boldsymbol{\theta}}) \right] \geq 0$ with equality if and only if $p_{Y|\mathbf{X}_{\alpha}}(y | \mathbf{x}_{\alpha}) = \Phi_{\alpha}(y | \mathbf{x}_{\alpha}; \hat{\boldsymbol{\theta}}) \forall y \in \mathbb{Y}$, this must be the case except possibly for $\mathbf{x} \in \mathbb{R}^T$ where

$$\int_{\{\mathbf{x}_{\alpha}: \mathbf{x} \in \mathbb{R}^T\}} p_{\mathbf{X}_{\alpha}}(\mathbf{x}_{\alpha}) d\mathbf{x}_{\alpha} = 0 \implies \int_{\mathbb{R}^T} p_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = 0. \quad (36)$$

This implies the second claim. \square

Corollary B.2. Under the assumptions of Proposition B.1, equations 20 and 26 are equivalent.

Proof. Observe that for any $\boldsymbol{\gamma} \in [0, 1]^T$,

$$\max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\boldsymbol{\gamma}, \boldsymbol{\theta}) = \max_{\boldsymbol{\theta} \in \mathbb{R}^P} \mathbb{E} \log \Phi_{\mathcal{A}_{\gamma}}(Y | \mathbf{X}_{\mathcal{A}_{\gamma}}(Y | \mathbf{X}_{\mathcal{A}_{\gamma}}; \boldsymbol{\theta})) \quad (37)$$

$$= \sum_{\alpha \in \{0, 1\}^T} p_{\mathcal{A}_{\gamma}}(\alpha) \mathbb{E} \log p_{Y|\mathbf{X}_{\alpha}}(Y | \mathbf{X}_{\alpha}) \quad \text{by Prop. B.1} \quad (38)$$

$$= - \sum_{\alpha \in \{0, 1\}^T} p_{\mathcal{A}_{\gamma}}(\alpha) \mathbb{H}[Y | \mathbf{X}_{\alpha}] \quad (39)$$

$$\equiv \sum_{\alpha \in \{0, 1\}^T} p_{\mathcal{A}_{\gamma}}(\alpha) [\mathbb{H}[Y] - \mathbb{H}[Y | \mathbf{X}_{\alpha}]] \quad \text{because } \mathbb{H}[Y] \text{ is not a function of } \boldsymbol{\gamma} \quad (40)$$

$$= \sum_{\alpha \in \{0, 1\}^T} p_{\mathcal{A}_{\gamma}}(\alpha) \mathbb{I}[Y; \mathbf{X}_{\alpha}] \quad (41)$$

$$= \mathbb{I}[Y; \mathbf{X}_{\mathcal{A}_{\gamma}} | \mathcal{A}_{\gamma}] \quad (42)$$

$$= \mathcal{L}_{\text{ideal}}(\boldsymbol{\gamma}). \quad (43)$$

This implies the result. \square

Corollary B.3. Suppose the assumptions of Proposition B.1 are satisfied, and let $\hat{\theta} \in \arg \min_{\theta \in \mathbb{R}^P} \mathcal{L}_{\text{adv}}(\gamma, \theta)$ for some $\gamma \in [0, 1]^T$. Consider $\alpha := \alpha' + \alpha''$ where $\alpha', \alpha'' \in \{0, 1\}^T$ such that $\alpha'_t = 1 \implies \alpha''_t = 0$ and $\alpha''_t = 1 \implies \alpha'_t = 0$, and $p_{\mathcal{A}_\gamma}(\alpha) > 0$. For all $y \in \mathbb{Y}$, it follows immediately from Proposition B.1 that $p_{\mathbf{X}}$ -almost everywhere we can use our classifiers to compute the pointwise mutual information quantities

$$\text{pmi}(y; \mathbf{x}_{\alpha'} | \mathbf{x}_{\alpha''}) := \log p_{Y|\mathbf{X}_\alpha}(y | \mathbf{x}_\alpha) - \log p_{Y|\mathbf{X}_{\alpha''}}(y | \mathbf{x}_{\alpha''}) \quad (44)$$

$$= \log \Phi_\alpha(y | \mathbf{x}_\alpha; \hat{\theta}) - \log \Phi_{\alpha''}(y | \mathbf{x}_{\alpha''}; \hat{\theta}). \quad (45)$$

This is useful because it allows us to assess leakage from *single* power traces, as opposed to merely summarizing distributions of power traces. There are scenarios where a power measurement might leak for some traces but not for others. For example, a common countermeasure is to randomly delay leaky instructions or swap their order with another instruction so that they do not occur at a deterministic time relative to the start of encryption. One could use pmi computations to determine the timetep at which the leaky instruction has been run in a single trace.

Since it would be impractical to train 2^T deep neural networks independently, we implement the family of classifiers by a single neural net with input dropout and with the dropout mask fed to the neural net as an auxiliary input:

$$\Phi : \mathbb{Y} \times \mathbb{R}^T \times \{0, 1\}^T \times \mathbb{R}^P \rightarrow [0, 1] : (y, \mathbf{x}, \alpha, \theta) \mapsto \Phi(y | \mathbf{x} \odot \alpha, \alpha; \theta) \quad (46)$$

where $\Phi_\alpha(y | \mathbf{x}_\alpha; \theta) := \Phi(y | \mathbf{x} \odot \alpha, \alpha; \theta)$. This approach was inspired by Lippe et al. (2022).

B.3. Re-parametrization into an unconstrained optimization problem

We would like to approximately solve equation 26 using an alternating stochastic gradient descent-style approach, similarly to GANs (Goodfellow et al., 2014). Thus, it is convenient to express it as an unconstrained optimization problem. We first define a new vector $\eta \in \Delta^{T-1}$ where $\Delta^{T-1} := \{(\delta_1, \dots, \delta_T) \in \mathbb{R}_+^T : \sum_{t=1}^T \delta_t = 1\}$ denotes the T -simplex. We then define γ to be the vector satisfying the equality

$$c(\gamma_t) = C\eta_t \quad (47)$$

$$\implies \frac{\gamma_t}{1 - \gamma_t} = C\eta_t \quad (48)$$

$$\implies \log \gamma_t - \log(1 - \gamma_t) = \log C + \log \eta_t \quad (49)$$

$$\implies \gamma_t = \text{sigmoid}(\log C + \log \eta_t). \quad (50)$$

If we define $\tilde{\eta} := \text{softmax}(\tilde{\eta})$ for $\tilde{\eta} \in \mathbb{R}^T$, then we can express

$$\gamma_t = \text{sigmoid}(\log C + \log \tilde{\eta}_t - \text{logsumexp}(\tilde{\eta})), \quad (51)$$

which allows us to map the unconstrained vector $\tilde{\eta}$ to γ or $\log \gamma$ using numerically-stable PyTorch operations. Our constrained optimization problem 26 is thus equivalent to the following unconstrained problem:

$$\min_{\tilde{\eta} \in \mathbb{R}^T} \max_{\theta \in \mathbb{R}^P} \mathcal{L}(\tilde{\eta}, \theta) := \mathbb{E} \log \Phi(Y | \mathbf{X} \odot \mathcal{A}_{\gamma(\tilde{\eta})}, \mathcal{A}_{\gamma(\tilde{\eta})}; \theta). \quad (52)$$

B.4. Estimating gradients with respect to $\tilde{\eta}$

It is infeasible to exactly compute the expectation with respect to $\mathcal{A}_{\gamma(\tilde{\eta})}$ because doing so would require summing over 2^T terms. Thus, we approximate it with Monte Carlo integration. Note that our objective takes the form

$$\mathcal{L}(\tilde{\eta}, \theta) = \mathbb{E} f(\mathcal{A}_{\gamma(\tilde{\eta})}), \quad (53)$$

where $f(\alpha) = \mathbb{E}_{\mathbf{X}, Y} \log \Phi(Y | \mathbf{X} \odot \alpha, \alpha, \theta)$ and the distribution of $\mathcal{A}_{\gamma(\tilde{\eta})}$ depends on $\tilde{\eta}$. In this case it is not straightforward to estimate the gradient $\nabla_{\tilde{\eta}} \mathcal{L}(\tilde{\eta}, \theta)$ because $\nabla_{\tilde{\eta}} \mathbb{E} f(\mathcal{A}_{\gamma(\tilde{\eta})}) \neq \mathbb{E} \nabla_{\tilde{\eta}} f(\mathcal{A}_{\gamma(\tilde{\eta})})$. A straightforward solution would be

to use the REINFORCE gradient estimator (Williams, 1992):

$$\nabla_{\tilde{\eta}} \mathbb{E} f(\mathcal{A}_{\gamma(\tilde{\eta})}) = \nabla_{\tilde{\eta}} \sum_{\alpha \in \{0,1\}^T} p_{\mathcal{A}_{\gamma(\tilde{\eta})}}(\alpha) f(\alpha) \quad (54)$$

$$= \sum_{\alpha \in \{0,1\}^T} \nabla_{\tilde{\eta}} p_{\mathcal{A}_{\gamma(\tilde{\eta})}}(\alpha) f(\alpha) \quad (55)$$

$$= \sum_{\alpha \in \{0,1\}^T} p_{\mathcal{A}_{\gamma(\tilde{\eta})}}(\alpha) \nabla_{\tilde{\eta}} p_{\mathcal{A}_{\gamma(\tilde{\eta})}}(\alpha) f(\alpha) \quad (56)$$

$$= \mathbb{E} \nabla_{\tilde{\eta}} \log p_{\mathcal{A}_{\gamma(\tilde{\eta})}}(\alpha) f(\alpha). \quad (57)$$

While this estimator is unbiased, it has high variance and requires many samples to converge to a good estimate, and for our application we find that it yields poor results. We instead use the REBAR gradient estimator (Tucker et al., 2017), which gives unbiased low-variance gradient estimates by relaxing f to accept ‘soft’ inputs $\alpha \in [0, 1]^T$ rather than ‘hard’ inputs $\alpha \in \{0, 1\}^T$ and using the reparameterization trick (Kingma & Welling, 2014) to estimate gradients of the resulting relaxed objective, then using these biased but low-variance estimates as a control variate to reduce the variance of the REINFORCE estimator. We use the following relaxation:

$$f(\alpha) = \mathbb{E}_{\mathbf{X}, Y \sim p_{\mathbf{X}, Y}, \boldsymbol{\varepsilon} \sim \mathcal{N}(0, 1)^T} \log \Phi(Y \mid \mathbf{X} \odot \alpha + \boldsymbol{\varepsilon} \odot (\mathbf{1} - \alpha), \alpha; \theta). \quad (58)$$

This is equivalent in the limit to the unrelaxed version of f because since $\boldsymbol{\varepsilon}$ is independent of Y , the neural net should learn to ignore the elements of its input which have been replaced by noise. Given this relaxation, we can now use the REBAR estimator. The expression for the estimator is lengthy, so we refer interested readers to (Tucker et al., 2017) and to our code: ([here](#)). Performance appears highly sensitive to subtle implementation details, and to our knowledge there is currently no generic bug-free implementation of REBAR written in PyTorch. Thus, we advise readers seeking to reproduce our work to follow our implementation closely.

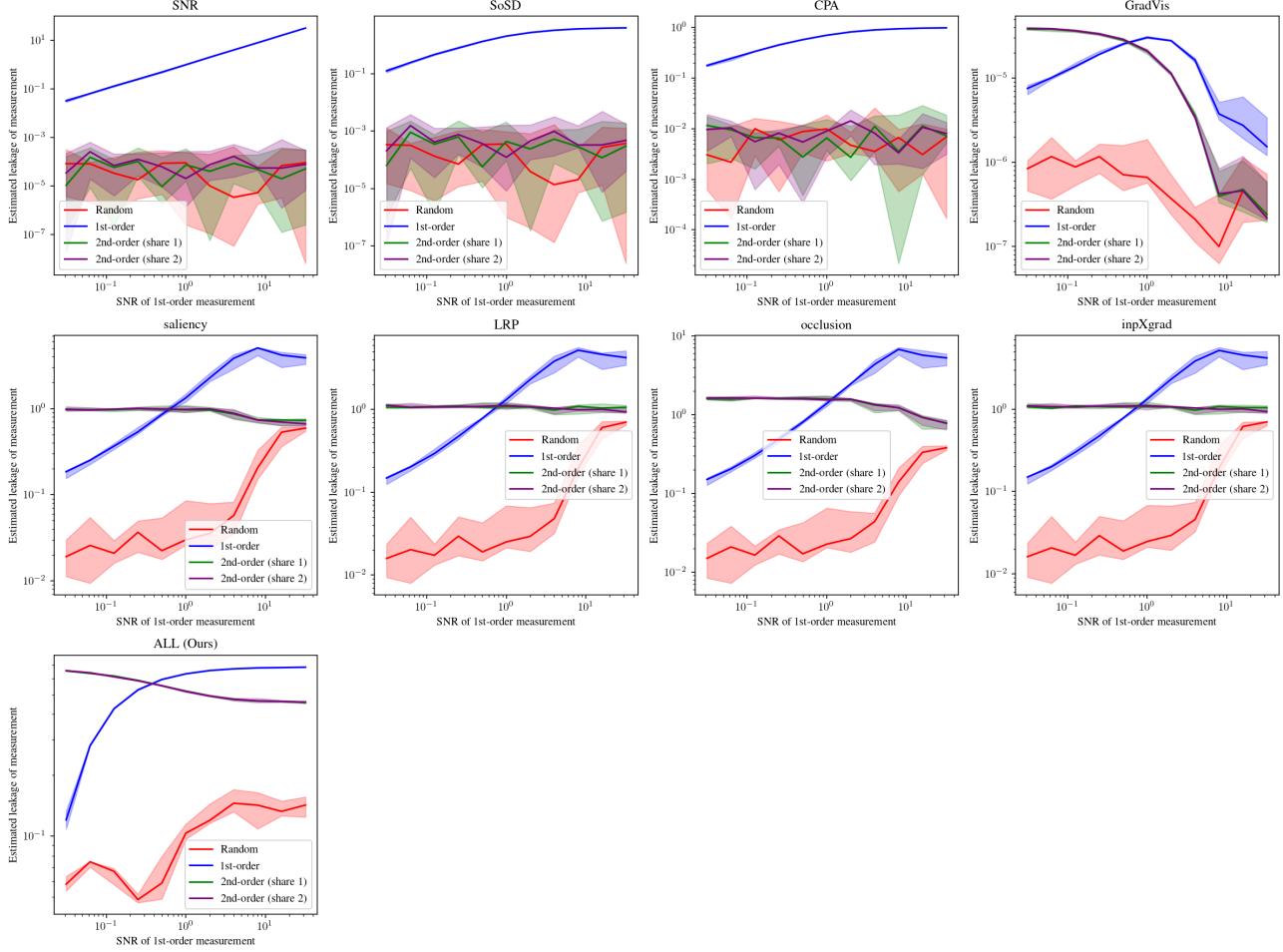


Figure 9. Plots of the leakage assigned to X_{rand} (red), X_{1o} (blue), $X_{2o,1}$ (purple), and $X_{2o,2}$ (green) as defined in Section C.1.1. As expected, the first-order statistical methods correctly assign leakage to the first-order leaking feature, but fail to distinguish the second-order leaking features from noise. Our adversarial leakage localization algorithm and all considered neural net attribution algorithms correctly identify that these features are leaking. It appears that for the neural net attribution methods, the gap closes when σ^2 is extremely small. This is likely because the conditional mutual information of the fixed-variance leaking points is extremely small.

C. Extended experiments

C.1. Toy settings where our technique works and baselines don't

C.1.1. FIRST-ORDER STATISTICAL METHODS CAN ONLY DETECT FIRST-ORDER LEAKAGE

First-order statistical methods are only sensitive to associations between single power measurements and the label, and fail to identify leakage when a sensitive variable is dependent on a set of measurements but pairwise-independent of each of them. To illustrate this, we generate instances of the following random variables:

$$Y \sim \mathcal{U}\{-1, 1\}, \quad M \sim \mathcal{U}\{-1, 1\}, \quad \mathbf{X} = (X_{\text{rand}}, X_{1o}, X_{2o,1}, X_{2o,2})$$

$$\text{where } X_{\text{rand}} \sim \mathcal{N}(0, 1), \quad X_{1o} \mid Y \sim \mathcal{N}(Y, \sigma^2), \quad X_{2o,1} \mid \{Y, M\} \sim \mathcal{N}(Y \oplus M, 1), \quad X_{2o,2} \mid M \sim \mathcal{N}(M, 1) \quad (59)$$

where \oplus denotes the exclusive-or operation: $y \oplus m = \begin{cases} 1 & y \neq m \\ -1 & y = m \end{cases}$. Note that Y is independent of both M and $Y \oplus M$,

but the set $\{M, Y \oplus M\}$ fully determines the value of Y : $Y = M \oplus (Y \oplus M)$. This is similar to Boolean masking which is used in some protected AES implementations, such as those used to generate ASCADv1-fixed and -variable (Benadjila et al., 2020).

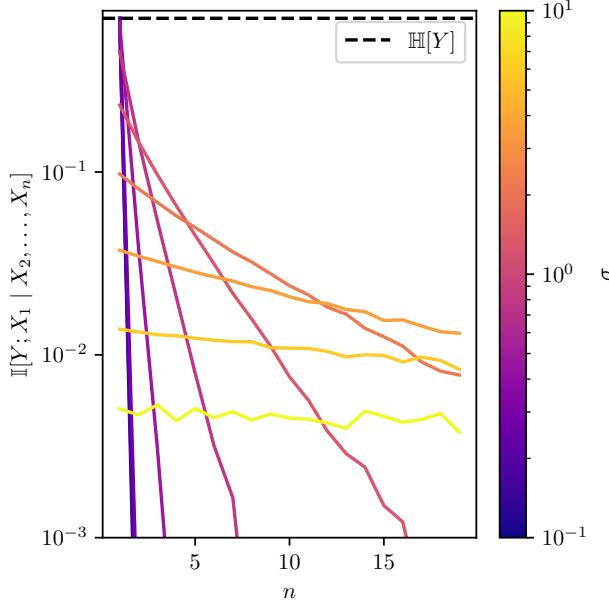


Figure 10. A numerical experiment evaluating the scaling behavior of $\mathbb{I}[Y; X_1 | X_2, \dots, X_n]$ vs n for various values of σ , where $Y \sim \mathcal{U}\{-1, 1\}$ and each $X_i | Y \sim \mathcal{N}(Y, \sigma^2)$. Observe that for small σ each X_i is approximately a point mass on Y , and we have $\mathbb{I}[Y; X_1] \approx \mathbb{H}[Y]$ and $\mathbb{I}[Y; X_1 | X_2, \dots, X_n] \approx 0$ for each n . For large σ , each X_i gives us little information about Y and we have $\mathbb{I}[Y; X_1 | X_2, \dots, X_n] \ll \mathbb{H}[Y]$ approximately constant with n .

We generate an infinitely-large dataset consisting of i.i.d. instances of the random variable $(X_{\text{rand}}, X_{10}, X_{20,1}, X_{20,2}, Y)$. All models are trained for 1e4 steps with minibatch size 1e3. Supervised and ALL classifiers are multilayer perceptrons with a single 500-neuron hidden layer and ReLU hidden activation. The classifiers and the $\tilde{\eta}$ ALL parameter are trained with the AdamW optimizer with weight decay disabled and the other settings left at their default values. For ALL we use $\bar{\gamma} = 0.5$. Results are shown in Fig. 9.

C.1.2. NEURAL NET ATTRIBUTION METHODS FAIL TO IDENTIFY SOME LEAKING POINTS WHEN MANY ARE PRESENT

Neural nets trained with supervised learning do not necessarily exploit all exploitable input-output associations (Geirhos et al., 2020; Hermann & Lampinen, 2020), which leads neural net attribution methods to fail to distinguish leaking and non-leaking points. Hermann & Lampinen (2020) find that neural nets tend to ignore XOR associations when first-order associations are present. In line with this finding, we have seen in Section C.1.1 that saliency and GradVis assign the same leakage to the second-order leaking points as to the non-leaking point when the first-order point is sufficiently-discriminative, and while LRP, 1-occlusion, and inpxgrad assign higher leakage to the former, the margin between the two becomes quite small.

Additionally, neural nets may fail to distinguish between leaky and nonleaky points when there are many leaky points with ‘redundant’ information about the label. Consider the following random variables:

$$Y \sim \mathcal{U}\{-1, 1\}, \quad \mathbf{X} = (X_1, \dots, X_n) \quad \text{where} \quad X_i | Y \sim \mathcal{N}(Y, \sigma^2) \quad \text{for } i = 1, \dots, n. \quad (60)$$

In Fig. 10 we plot the quantity $\mathbb{I}[Y; X_1 | X_2, \dots, X_n]$ as we sweep n , for various values of σ^2 . Empirically we see that

$$\mathbb{I}[Y; X_1 | X_2, \dots, X_n] \approx Ik^n \quad \text{for some } I \in \mathbb{R}_+, k \in (0, 1), \quad (61)$$

where k increases with σ^2 . Informally, this makes sense for the following reason: we have $0 \leq \mathbb{I}[Y; X_1 | X_2, \dots, X_n] \leq \mathbb{H}[Y]$. Because the quantities X_i are independent and each is statistically-associated with Y but do not fully determine it, $\mathbb{I}[Y; X_1 | X_2, \dots, X_n]$ strictly decreases as n increases, and by the monotone convergence theorem we must have $\lim_{n \rightarrow \infty} \mathbb{I}[Y; X_1 | X_2, \dots, X_n] = 0$.

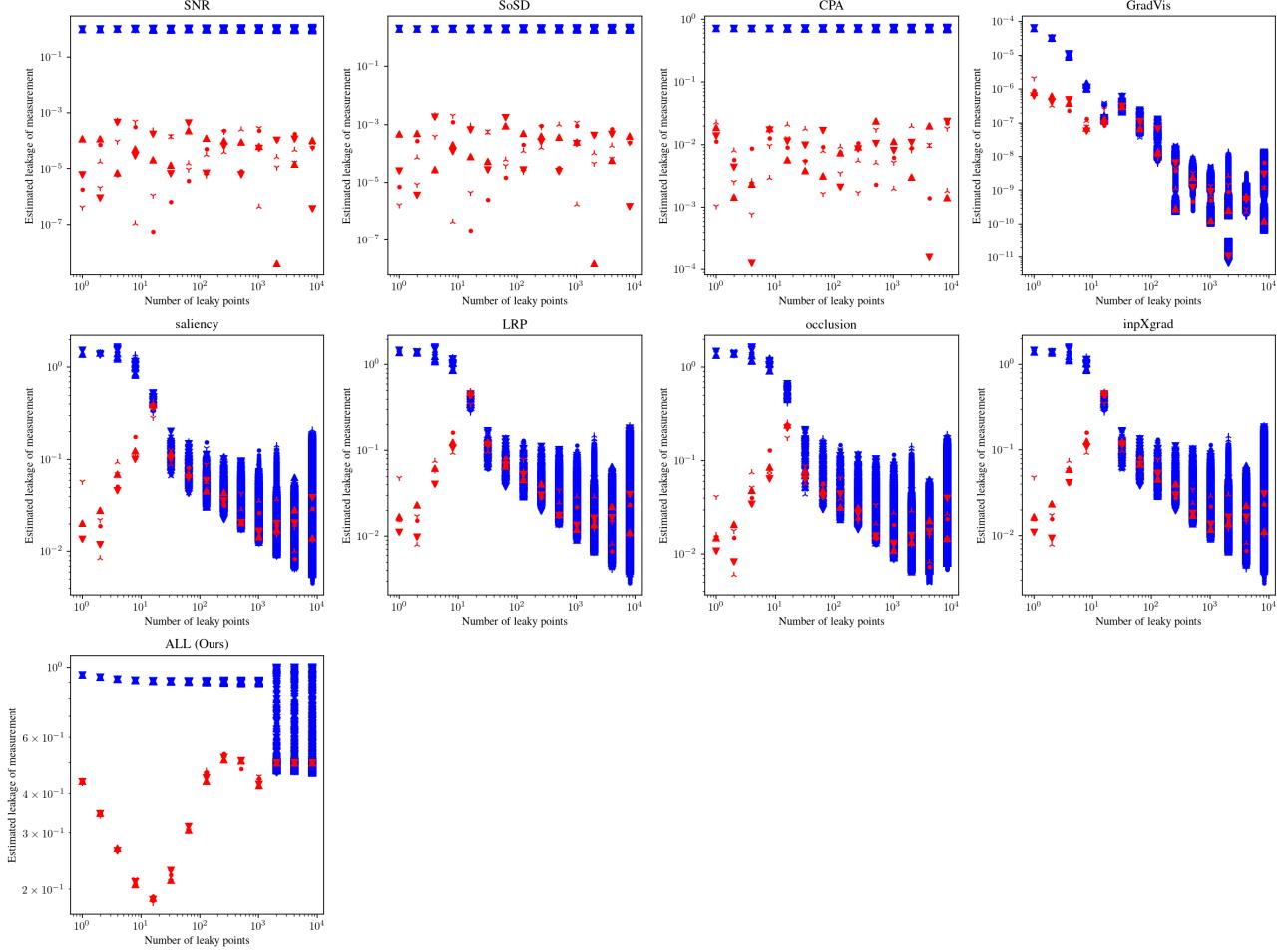


Figure 11. Plots of the leakage assigned to the nonleaky feature X_0 (red) and the leaky features $X_i, i \geq 1$ (blue) by our considered leakage localization algorithms. In line with our expectations, our adversarial leakage localization algorithm successfully distinguishes between them for n as large as 1024, whereas all considered neural net attribution baselines fail for n as small as 32. The parametric statistical methods are not sensitive to n because they only consider pairwise associations between the features and label.

In other words, as n increases, Y becomes nearly conditionally-independent of any element of \mathbf{X} given the other elements, despite being pairwise dependent on all of them. The neural net attribution methods can be viewed as estimating the conditional distribution $p_{Y|\mathbf{X}}(y | x_1, \dots, x_n)$ and estimating through various means the ‘influence’ of each input x_i on the output. Consider the case where $p_{Y|\mathbf{X}}(y | x_1, \dots, x_n) \approx p_{Y|X_2, \dots, X_n}(y | x_2, \dots, x_n)$. Then we have

$$\frac{\partial}{\partial x_1} p_{Y|\mathbf{X}}(y | x_1, \dots, x_n) \approx \frac{\partial}{\partial x_1} p_{Y|X_2, \dots, X_n}(y | x_2, \dots, x_n) = 0 \quad (62)$$

$$\implies (\text{gradvis}(p_{Y|\mathbf{X}}))_1 = -\frac{\partial}{\partial x_1} \log p_{Y|\mathbf{X}}(y | x_1, \dots, x_n) = -\frac{\frac{\partial}{\partial x_1} p_{Y|\mathbf{X}}(y | x_1, \dots, x_n)}{p_{Y|\mathbf{X}}(y | x_1, \dots, x_n)} \approx 0, \quad (63)$$

$$(\text{saliency}(p_{Y|\mathbf{X}}))_1 = \left| \frac{\partial}{\partial x_1} p_{Y|\mathbf{X}}(y | x_1, \dots, x_n) \right| \approx 0, \quad (64)$$

$$\text{and } (\text{inpxgrad}(p_{Y|\mathbf{X}}))_1 = x_1 \frac{\partial}{\partial x_1} p_{Y|\mathbf{X}}(y | x_1, \dots, x_n) \approx 0. \quad (65)$$

Additionally,

$$(\text{occl}(p_{Y|\mathbf{X}}))_1 = p_{Y,\mathbf{X}}(y | x_1, \dots, x_n) - p_{Y|\mathbf{X}}(y | 0, x_2, \dots, x_n) \quad (66)$$

$$\approx p_{Y|X_2, \dots, X_n}(y | x_2, \dots, x_n) - p_{Y|X_2, \dots, X_n}(y | x_2, \dots, x_n) = 0. \quad (67)$$

For sufficiently-large n , the estimated ‘leakiness’ of leaky features thus decays to zero and the neural net attribution methods do not distinguish between leaky and non-leaky timesteps.

In Fig. 11 we generate infinitely-large datasets according to equation 60, with an additional nonleaky feature $X_0 \sim \mathcal{N}(0, 1)$. We compare the leakage assigned to X_0 vs. X_1, X_2, \dots by our adversarial leakage localization algorithm and considered baselines. The experimental setup is the same as in Section C.1.1 except that we use $\bar{\gamma} = 0.9$ for ALL, as we find this works better.

C.2. Synthetic datasets

Here we present experiments done on synthetic AES power trace datasets. These are useful because unlike with real datasets, we have ground-truth knowledge about which points are leaking and can compare this against our method’s output. Additionally, we can observe the behavior of our method while sweeping parameters such as low-pass filtering strength and leaking instruction count.

C.2.1. DATA GENERATION PROCEDURE

We base our synthetic power trace datasets on the Hamming weight leakage model of Mangard et al. (2007, ch. 4). This model² assumes that we have a device which executes a cryptographic algorithm as a sequence of operations on data. As above, let $\mathbf{X} := (X_t : t = 1, \dots, T)$ be a random vector with range \mathbb{R}^T which encodes power consumption. Let $\mathbf{D} := (D_t : t = 1, \dots, T)$ and $\mathbf{O} := (O_t : t = 1, \dots, T)$ be random vectors denoting the data and operations, respectively, where each D_t has range $\{0, 1\}^{n_{\text{bits}}}$ (i.e. a sequence of n_{bits} bits) and each O_t has range $[1 \dots n_{\text{ops}}]$ for some $n_{\text{bits}}, n_{\text{ops}} \in \mathbb{Z}_{++}$. For each $t \in [1 \dots T]$, we can decompose

$$X_t = X_{\text{data},t} + X_{\text{op},t} + X_{\text{resid},t} \quad (68)$$

with dependency structure illustrated in the causal diagram of Fig. 12. Note that $X_{\text{data},t}$ is directly associated with the data D_t , $X_{\text{op},t}$ is directly associated with the operation O_t , and $X_{\text{resid},t}$ captures the randomness in power consumption we would see if we were to repeatedly measure power consumption with a fixed operation and data (e.g. due to other processes on the device independently of the encryption process, or noise due to the thermal motion of electrons in wires).

The authors of Mangard et al. (2007) experimentally characterize the power consumption of a cryptographic device and find that it is reasonable to approximate $X_{\text{data},t}$ as Gaussian noise with D_t -dependent mean, $X_{\text{op},t}$ as Gaussian noise with

²Mangard et al. (2007) uses the notation $P_{\text{total}} = P_{\text{op}} + P_{\text{data}} + P_{\text{el. noise}} + P_{\text{const}}$. For clarity and consistency, we alter the notation, consolidate $P_{\text{el. noise}}$ and P_{data} into a single variable, and more-explicitly define the probabilistic nature of the variables and the associations between them.

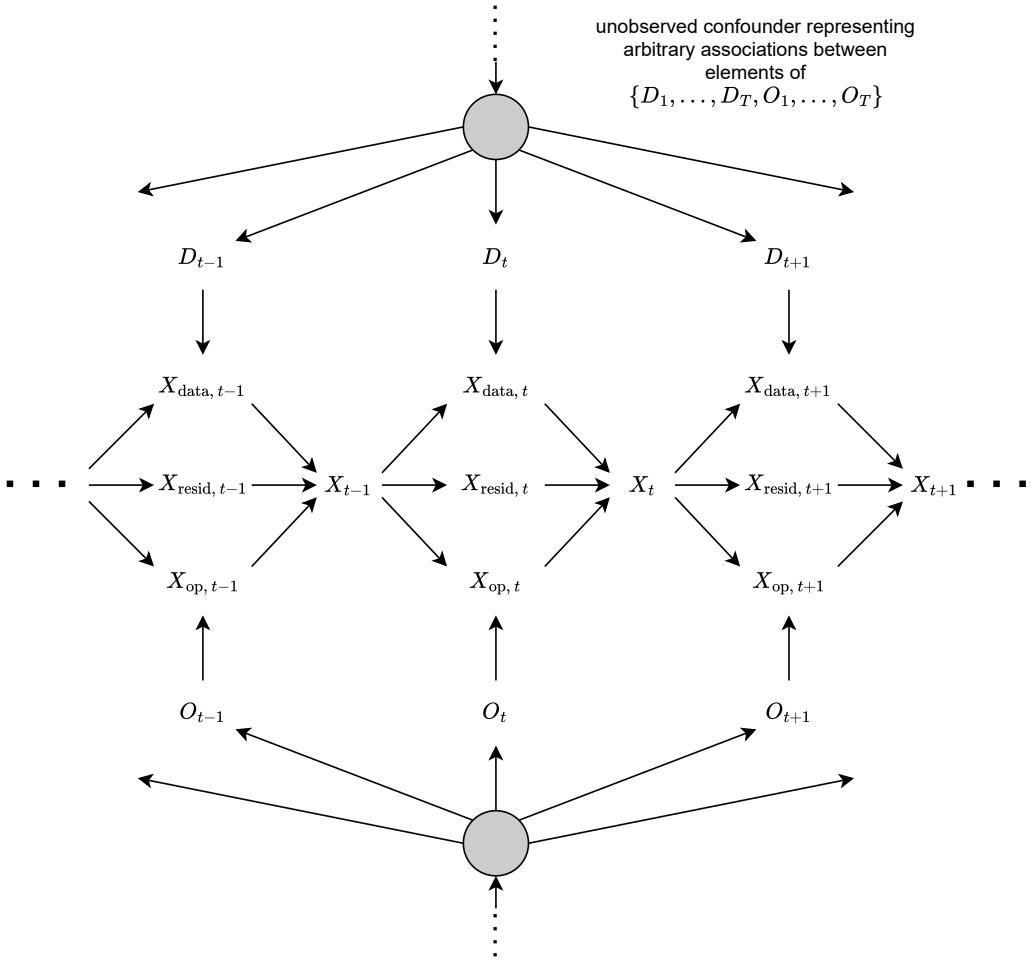


Figure 12. Causal diagram which shows the independence conditions between different components of power consumption which we have assumed in our synthetic power trace dataset. Power consumption at a time t is decomposed as $X_t = X_{\text{data}, t} + X_{\text{op}, t} + X_{\text{resid}, t}$ where $X_{\text{data}, t}$ is directly associated with the present data D_t , $X_{\text{op}, t}$ is directly associated with the present operation O_t , and $X_{\text{resid}, t}$ is directly associated with neither, and accounts for all sources of randomness in power consumption not directly associated with the data or operation. We assume that arbitrary associations may exist between the data and operations at different points in time. We assume that power consumption at time $t + 1$ is associated with that at time t due to low-pass filtering effects in real circuits and measurement equipment.

O_t -dependent mean, and $X_{\text{resid},t}$ as Gaussian noise with a constant mean (which we will assume to be 0). For their device, the mean of $X_{\text{data},t}$ is proportional to

$$n_{\text{bits}} - \text{HammingWeight}(D_t) := \sum_{k=1}^{n_{\text{bits}}} 1 - D_{t,k}, \quad (69)$$

i.e. the number of bits of D_t which are equal to 0. Additionally, the per- O_t means of $X_{\text{op},t}$ are approximately Gaussian-distributed.

We adopt these approximations for our experiments, though we emphasize that they are not universally-applicable to cryptographic devices. For example, the Hamming weight dependence of the mean of $X_{\text{data},t}$ on D_t is due to the fact that their device ‘pre-charges’ all of its data bus lines to 1, then drains the charge from the lines which should represent 0, thereby consuming power proportional to the number of lines which represent 0. Many devices operate differently. Additionally, cryptographic hardware is often explicitly designed to obfuscate the association between power consumption and data/operations as a defense mechanism against side-channel attacks.

Algorithm 2: Simplified procedure for generating synthetic power trace datasets based on the Hamming weight leakage model of [Mangard et al. \(2007\)](#).

Input: Dataset size $N \in \mathbb{Z}_{++}$,
 Timesteps per power trace $T \in \mathbb{Z}_{++}$,
 Bit count $n_{\text{bits}} \in \mathbb{Z}_{++}$,
 Operation count $n_{\text{ops}} \in \mathbb{Z}_{++}$,
 1st-order leaking timestep count $n_{\text{lkg}} \in \mathbb{Z}_+$,
 Data-dependent noise variance $\sigma_{\text{data}}^2 \in \mathbb{R}_+$,
 Operation-dependent noise variance $\sigma_{\text{op}}^2 \in \mathbb{R}_+$,
 Residual noise variance $\sigma_{\text{resid}}^2 \in \mathbb{R}_+$

Output: Synthetic dataset $D \subset \mathbb{R}^T \times [1 .. 2^{n_{\text{bits}}}]$

```

1  $\{k^{(n)} : n \in [1 .. N]\} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}(\{0, 1\}^{n_{\text{bits}}})$  // cryptographic keys
2  $\{w^{(n)} : n \in [1 .. N]\} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}(\{0, 1\}^{n_{\text{bits}}})$  // plaintexts
3  $\{o_t : t \in [1 .. T]\} \stackrel{\text{i.i.d.}}{\sim} \mathcal{U}([1 .. n_{\text{ops}}])$  // operations
4  $\{\tilde{x}_{\text{op},o} : o \in [1 .. n_{\text{ops}}]\} \stackrel{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_{\text{op}}^2)$  // per-operation power consumption
5  $x_{\text{op}} \leftarrow (\tilde{x}_{\text{op},o_t} : t = 1, \dots, T)$  // operation-dependent power consumption
6  $T_{\text{lkg}} \sim \mathcal{U}\left(\frac{[1 .. T]}{n_{\text{lkg}}}\right)$  // leaking timesteps, sampled w/o replacement
7 for  $n \in [1 .. N]$  do
8    $y^{(n)} \leftarrow \text{Sbox}(k^{(n)} \oplus w^{(n)})$  // sensitive variable
9    $x_{\text{resid}}^{(n)} \sim \mathcal{N}_T(\mathbf{0}, \sigma_{\text{resid}}^2 \mathbf{I})$  // residual power consumption
10  for  $t \in T_{\text{lkg}}$  do
11     $d_t^{(n)} \leftarrow y^{(n)}$  // timesteps at which the sensitive variable leaks
12  for  $t \in [1 .. T] \setminus T_{\text{lkg}}$  do
13     $d_t^{(n)} \sim \mathcal{U}(\{0, 1\}^{n_{\text{bits}}})$  // other data which we treat as random
14  for  $t \in [1 .. T]$  do
15     $x_{\text{data},t}^{(n)} \leftarrow \sigma_{\text{data}}(4 - \text{HammingWeight}(d_t^{(n)})) / \sqrt{2}$  // data-dependent power consumption
16   $x^{(n)} \leftarrow x_{\text{data}}^{(n)} + x_{\text{op}} + x_{\text{resid}}^{(n)}$  // total power consumption
17 return  $\{(x^{(n)}, y^{(n)}) : n \in [1 .. N]\}$ 

```

A simplified version of our data generation procedure is shown in algorithm 2. In our experiments we have also simulated desynchronization, Boolean masking and shuffling countermeasures. We have also explored low-pass filtering (exponentially-weighted moving averaging) the traces as a rough approximation to the ‘inertia’ of power consumption in real devices. We omit these details for brevity in the above algorithm, but they can be found in our code ([here](#)).

C.2.2. SYNTHETIC DATA EXPERIMENTS

We run numerous experiments while sweeping our algorithm’s hyperparameter $\bar{\gamma}$ as well as various properties of the dataset. For the following experiments we use the following setup: Φ is a multilayer perceptron with 3 500-neuron hidden layers and ReLU hidden activation, trained with the AdamW optimizer with learning rate $1e-3$ and weight decay $1e-2$ (applied only to weights, not biases), and otherwise default PyTorch settings. We train for $1e4$ steps with minibatch size $1e3$. $\tilde{\eta}$ is also trained with the AdamW optimizer with learning rate $1e-3$ and weight decay 0.0 , and otherwise default settings. No learning rate schedules are used. We track convergence by plotting each element of γ vs. training steps, and verifying that they are all roughly flat at the end of training.

We run several experiments while sweeping our $\bar{\gamma}$ parameter over the values $\{0.05, 0.1, 0.5, 0.9, 0.95\}$. Unless otherwise specified, we use the following dataset settings: $N = \infty$, $T = 101$, $n_{\text{bits}} = 8$, $n_{\text{ops}} = 32$, $n_{\text{lkg}} = 1$, $\sigma_{\text{data}}^2 = 1.0$, $\sigma_{\text{resid}}^2 = 1.0$, $\sigma_{\text{op}}^2 = 1.0$. Additionally, we apply a discrete time-approximated low-pass filter with strength β , implemented as an exponentially-decaying weighted moving average: $x \mapsto (\beta x_{t-1} + (1 - \beta)x_t : t = -\infty, \dots, \infty)$. We use a default value of $\beta = 0.5$. By default we do not use random no-ops or shuffling.

Our experiments are as follows: in Fig. 13, we sweep β over the values $\{1 - 2^{-n} : n = 0, \dots, 7\}$. In Fig. 14 we sweep n_{lkg} over the values $\{0, 1, 3, 5, \dots, 13\}$. In Fig. 15 we insert random no-ops, with the maximum number of no-ops swept over $\{0, 1, 5, 9, \dots, 25\}$. In Fig. 16 we randomly shuffle the leaky instruction over multiple possible positions, with possible position count swept over $\{1, 3, 5, \dots, 15\}$. For each experiment we repeat each configuration 5 times with different random seeds, reporting median values as solid lines and min – max as a shaded interval.

In all our experiments we see reasonable results for at least some values of $\bar{\gamma}$. We find that $\bar{\gamma}$ works well for all experiments, and we use it for the plots in the main paper. It appears that performance often becomes inconsistent for large $\bar{\gamma}$, while for small $\bar{\gamma}$ we get successful detection of the leakiest points, but less sensitivity to the less-leaky points. It appears that $\bar{\gamma}$ is an important parameter which should be tuned.

C.3. Comparison to baselines on real datasets

C.3.1. EVALUATING PERFORMANCE WITHOUT GROUND TRUTH KNOWLEDGE OF WHICH POINTS ARE LEAKING

It is not obvious how to evaluate the fidelity of a leakage assessment on real datasets. In general, prior work does so by evaluating some notion of the ‘usefulness’ of subsets of points to a classifier trained to predict the sensitive variable, and checking the extent to which this increases monotonically with the estimated leakage. We use 3 complementary performance evaluation strategies which are in line with this intuition.

‘Omniscient’ Gaussian Mixture Model assessment We consider a sliding window of measurements, and for each window we train a Gaussian mixture model to predict sensitive variables. Such an approach will not work in the presence of Boolean masking because of the fact that doing so requires second-order associations between temporally-distant timesteps, as well as the limited expressiveness of Gaussian mixture models. To overcome this limitation, we train a separate Gaussian mixture model to predict the value of each random share of the sensitive variable (e.g. a Boolean mask, and the masked SubBytes variable). Thus, while we cannot exploit arbitrary associations, we exploit temporally-local associations such as what we expect to occur when low-pass filtering is present, and the assumed-important nonlocal associations involving the random shares. We refer to such models as ‘omniscient’ because the leakage localization algorithms we evaluate do not have labels for the random shares.

We fit Gaussian mixture models by first fitting the target-conditional means, then using the AdamW optimizer to train the target-conditional precision matrices to maximize the log-likelihood of a profiling dataset. We use the default hyperparameters with weight decay value $1e-2$, and train for $1e3$ steps with batch size $1e6/T$. We shuffle the data using the same random seed for each window. Our code contains a PyTorch implementation of this procedure which allows many GMMs to be fit in parallel using a GPU. We use this approach rather than explicitly computing the precision matrices because it requires less computation and avoids numerical issues due to near-singular covariance matrices on certain datasets.

To evaluate leakage localization outputs, we do the following:

1. For each window position and each share of the sensitive variable, we fit a Gaussian mixture model of the conditional distribution of the measurements given the share.

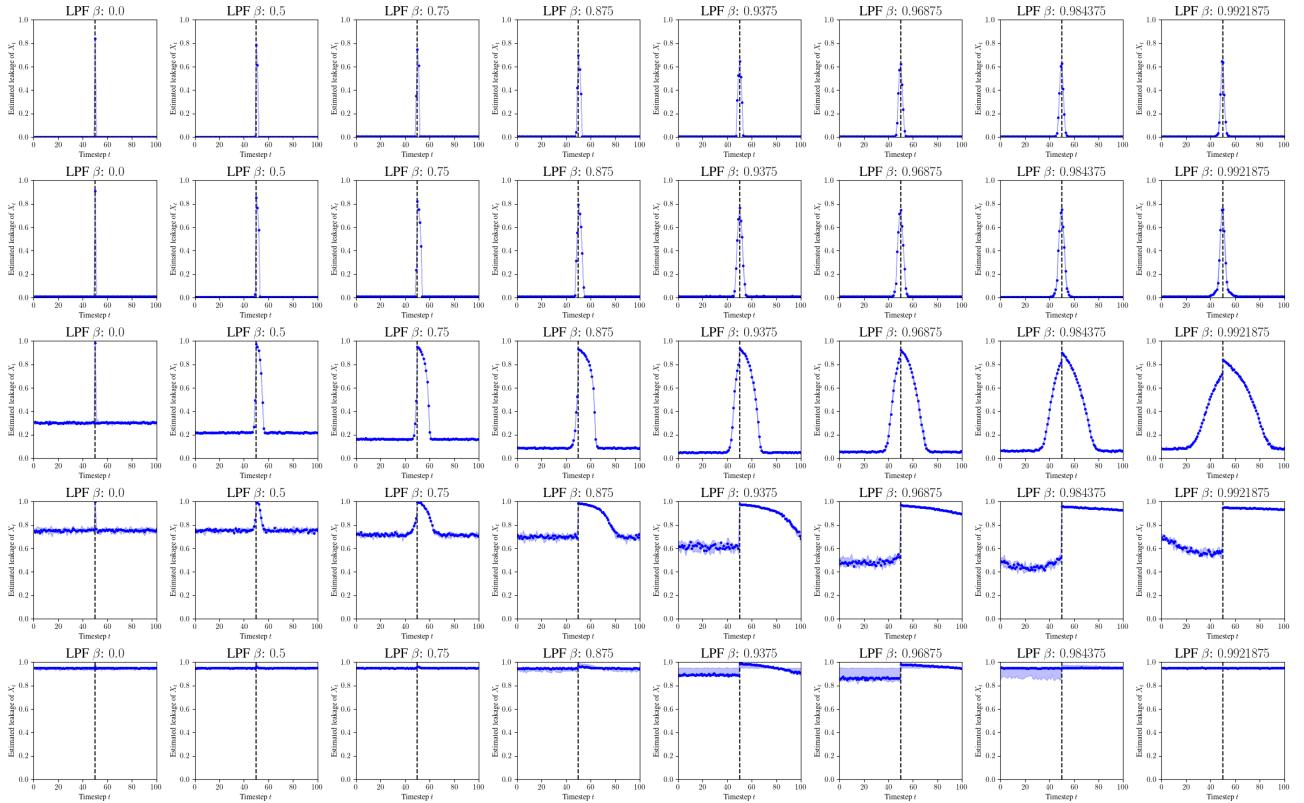


Figure 13. Sweep of low-pass filtering strength β and our algorithm’s hyperparameter $\bar{\gamma}$. From top to bottom we have $\bar{\gamma} = 0.05, 0.1, 0.5, 0.9, 0.95$. From left to right we have $\beta = (1 - 2^{-n} : n = 0, \dots, 7)$. Results appear to be reasonable for $\bar{\gamma} \leq 0.5$, with some strange and incorrect behavior for larger values.

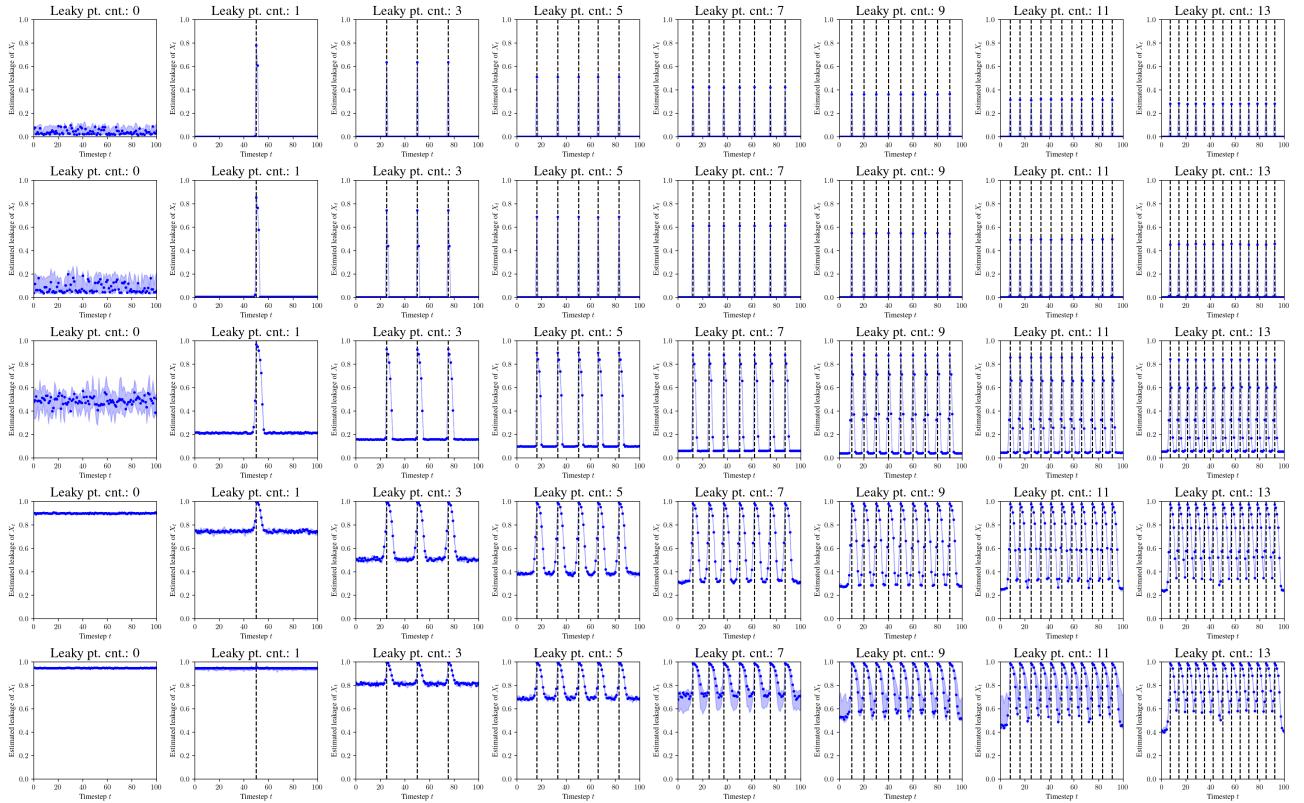


Figure 14. Sweep of leaky instruction count and our algorithm’s hyperparameter $\bar{\gamma}$. From top to bottom we have $\bar{\gamma} = 0.05, 0.1, 0.5, 0.9, 0.95$. From left to right we have leaky instruction counts of 0, 1, 3, 5, 7, 9, 11, 13. Results appear to be reasonable regardless of the value of $\bar{\gamma}$ we use here.

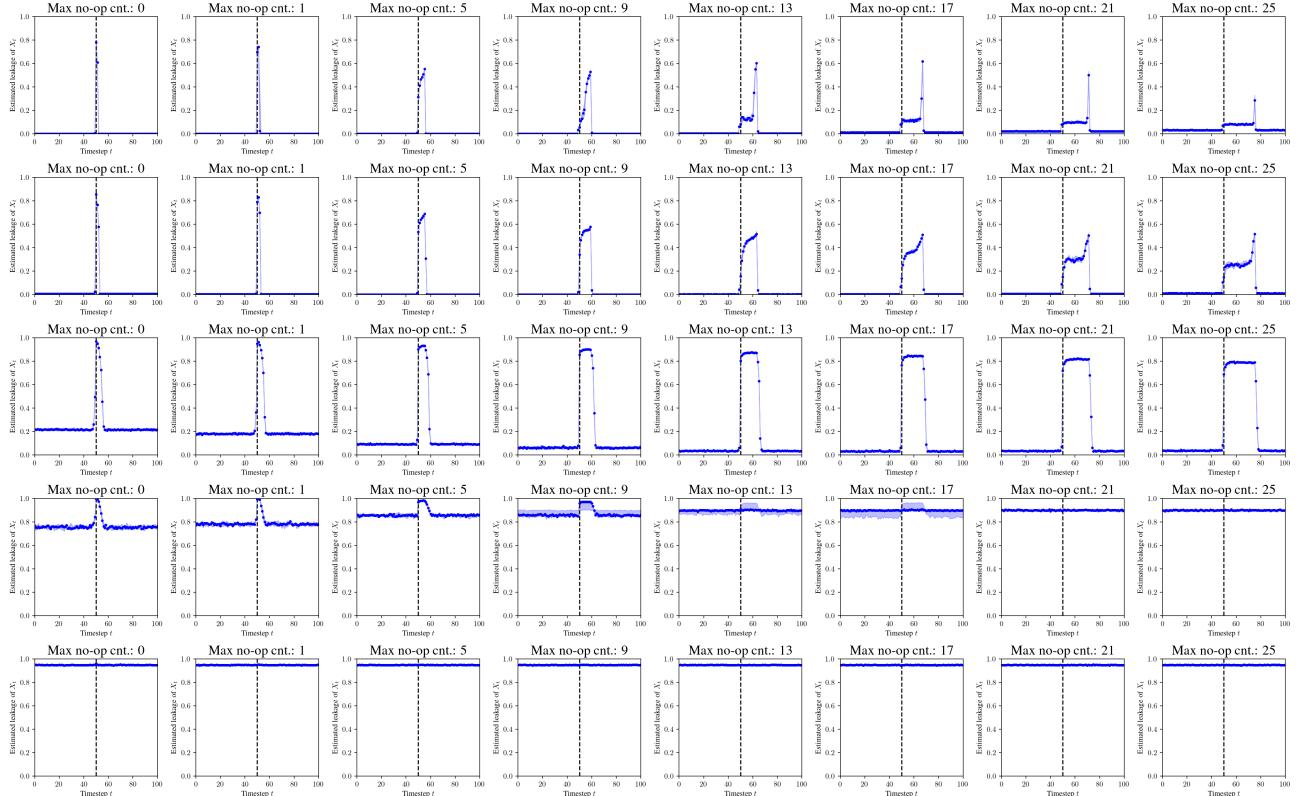


Figure 15. We insert a random number of no-ops before the leaky instruction, and sweep the maximum number which may be added, while also sweeping our algorithm’s hyperparameter $\bar{\gamma}$. From top to bottom we have $\bar{\gamma} = 0.05, 0.1, 0.5, 0.9, 0.95$. From left to right we have maximum no-op count of 0, 1, 5, 9, 13, 17, 21, 25. Results seem to be reasonable for $\bar{\gamma} \leq 0.5$, with inconsistent or unsuccessful behavior above that.

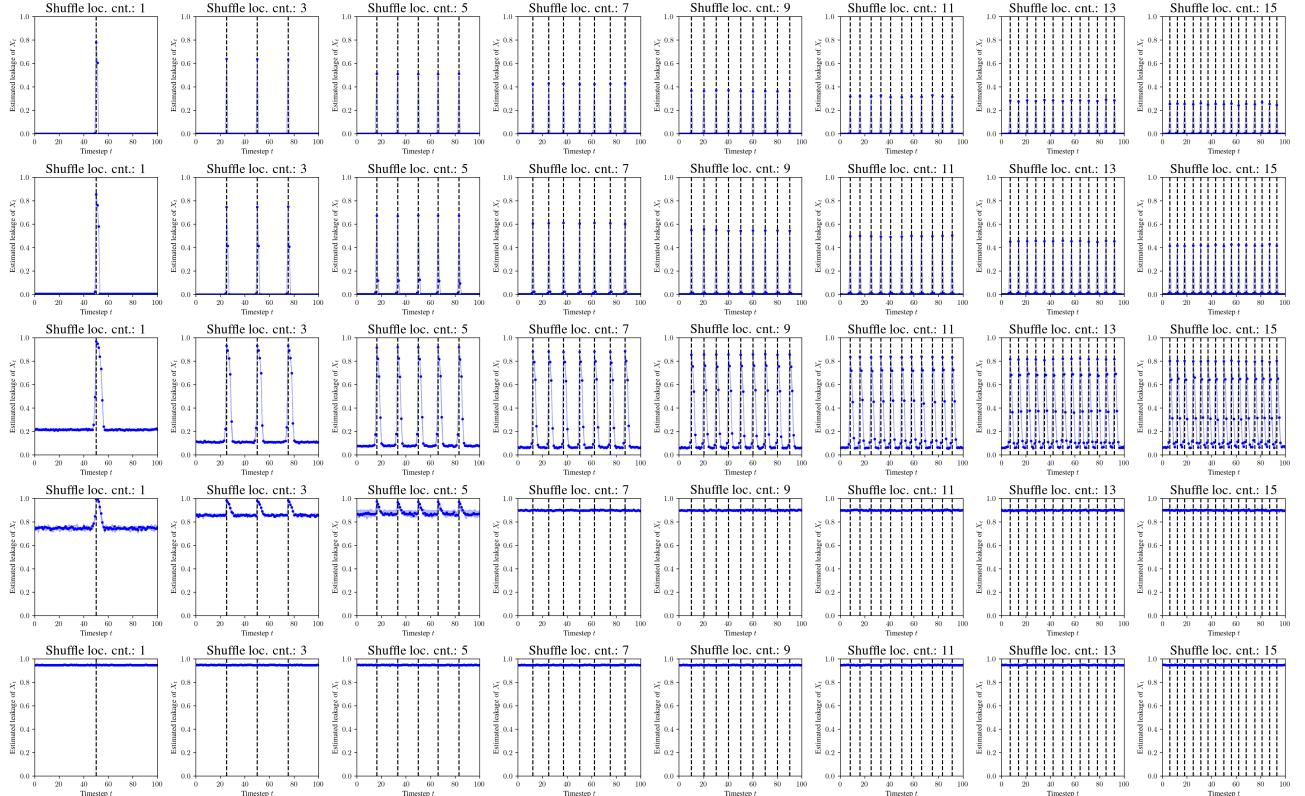


Figure 16. Instead of having a leaky instruction at a deterministic location, we randomly insert the leaky instruction at one of n locations for each trace. Here we sweep n as well as our algorithm's hyperparameter $\bar{\gamma}$. From top to bottom we have $\bar{\gamma} = 0.05, 0.1, 0.5, 0.9, 0.95$. From left to right we have $n = 1, 3, 5, 7, 9, 11, 13, 15$. Results appear to be reasonable for $\bar{\gamma} \leq 0.5$, with inconsistent or unsuccessful behavior above that.

2. Using these models, we estimate the mutual information between the share and the window of measurements. Mutual information is estimated as $\mathbb{I}[Y; \mathbf{X}] \approx -\frac{1}{D} \sum_{y \in Y} \sum_{(\mathbf{x} \in D)} \hat{p}(\mathbf{x}, y) \log \hat{p}(y | \mathbf{x})$. We assume that $\hat{p}(y) = \frac{1}{|Y|}$, which allows us to estimate $\hat{p}(\mathbf{x}, y)$ and $\hat{p}(y | \mathbf{x})$ from $\hat{p}(\mathbf{x} | y)$.
3. We compute the average mutual information over all random shares and compare with the leakage localization output, averaged over the corresponding windows.
4. Optionally, we can summarize this as a scalar using the Spearman rank correlation coefficient. This correlation coefficient is the Pearson correlation between the *ranks* of the pair of vectors, and is sensitive to the extent to which one is a monotone function of the other. We use this rather than the Pearson coefficient or similar because different localization algorithms have different ‘shapes’, and we do not want this to influence comparisons.

Forward and reverse DNN occlusion tests The idea behind these metrics is to successively ‘occlude’ inputs to a deep neural net and see how its holdout performance changes, as a proxy for some notion of the ‘importance’ of these inputs. These neural nets use the hyperparameters found according to the procedure of appendix C.3.2, although we make sure to use different random seeds for the occluded DNN as for the DNN which generated any neural net attribution assessments which are being evaluated. For the forward test, we do the following:

1. Initialize a mask $\alpha = \mathbf{0} \in \mathbb{R}^T$.
2. Repeatedly compute the mean rank of the DNN on a test dataset when all inputs are multiplied by α . Then set to 1 the leakiest element of α which is currently equal to 0.

The reverse test is similar:

1. Initialize a mask $\alpha = \mathbf{0} \in \mathbb{R}^T$.
2. Repeatedly compute the mean rank of the DNN on a test dataset when all inputs are multiplied by α . Then set to 1 the *least-leaky* element of α which is currently equal to 0.

We can then plot these mean rank measurements over time, and optionally summarize them as scalars by computing their averages (‘area’ under the curve when we consider the horizontal axis to span the interval $[0, 1]$.) For the forward test, a lower area under the curve is better because it indicates that the estimated ‘leakiest’ elements were useful to the DNN, causing its rank to decrease quickly. Conversely, for the reverse test, a higher area under the curve is better because it indicates that the estimated ‘least-leaky’ elements were not useful to the DNN. We expect the former metric to be sensitive to false-positive leakage detection and the latter to false-negative detection.

C.3.2. PROCEDURE FOR TRAINING SUPERVISED MODELS

Classifier architecture The architectures of Zaid et al. (2020); Wouters et al. (2020) are popular for doing side-channel attacks on the ASCADv1, AES-HD, and DPAv4 datasets. These are effectively an average pooling layer with kernel size 2, followed by a narrow multilayer perceptron. These architectures have been optimized with the goal of minimizing parameter count while retaining strong classification performance, and are manually designed for each dataset they consider. In contrast, we want a highly-expressive architecture which can leverage nearly-arbitrary input-output association, we do not care about parameter-efficiency, and we do not want to fine-tune our architecture for different datasets. Thus, for most of our experiments we use a simple multilayer perceptron architecture with 3 500-neuron hidden layers and ReLU hidden activations. The exception is AES-HD, where we find this architecture completely fails to generalize. Here we instead use a multilayer perceptron with 1 2-neuron hidden layer, which is the architecture proposed by Wouters et al. (2020) except without input downsampling and with ReLU used in place of SELU as the hidden activation.

We also considered using VGG-style CNN architectures similar to those of Benadjila et al. (2020) and some variants of Zaid et al. (2020); Wouters et al. (2020), as well as the ResNet-style architecture of Wang et al. (2017), but found in preliminary experiments that these architectures were worse-performing in terms of classification accuracy and leakage localization efficacy, in addition to being slower to train. We suspect that the inductive biases of convolutional layers are a poor fit for the datasets we are using.

Leakage localization through power consumption

| Hyperparameter (default) | Search space | Selected value | | | | | |
|---|--|----------------|------------------|------------------|-------------------|-------------------|---------------------|
| | | DPAv4 (Zaid) | ASCADv1-fixed | ASCADv1-var | AES-HD | OTiAiT | OTP |
| Learning rate (1e-3) | {1e-6, ..., 9e-6, 1e-5, 2e-5, ..., 9e-4} | 7e-4 | 3e-4 | 7e-4 | 9e-4 | 8e-4 | 6e-4 |
| β_1 (0.9) | {0.0, 0.5, 0.9, 0.99} | 0.5 | 0.5 | 0.0 | 0.5 | 0.9 | 0.99 |
| Weight decay (1e-2) | {0.0, 1e-4, 1e-2} | 0.0 | 1e-2 | 1e-2 | 0.0 | 0.0 | 0.0 |
| Learning rate schedule | {constant, cosine annealing} | constant | cosine annealing | cosine annealing | cosine annealing | cosine annealing | constant |
| Training steps | n/a | 1e4 | 1e4 | 1e4 | 1e4 | 1e3 | 1e3 |
| Minibatch size | n/a | 256 | 256 | 256 | 256 | 256 | 256 |
| Optimizer | n/a | AdamW | AdamW | AdamW | AdamW | AdamW | AdamW |
| Performance metric (random guessing value, lowest value) | | | | | | | |
| Rank ($\frac{1}{2}(Y + 1), 1$) | | 3.7 ± 0.3 | 106.3 ± 0.3 | 81.7 ± 0.5 | 125 ± 2 | 1.001 ± 0.001 | 1.0012 ± 0.0002 |
| Cross-entropy loss ($\log Y , 0$) | | 3.9 ± 0.3 | 5.64 ± 0.04 | 5.30 ± 0.01 | 5.562 ± 0.004 | 0.003 ± 0.002 | 0.011 ± 0.005 |

Table 1. Results of random hyperparameter searches for our supervised learning baseline models on our considered datasets. For each dataset we run 50 trials with a random hyperparameter configuration sampled uniformly from the search space. We then select the best-performing configuration, train 5 models with different random seeds, and report the mean \pm standard deviation of its performance metrics.

As a sanity check that our training procedure produces comparable results to prior work, we also evaluate the pretrained Zaid et al. (2020); Wouters et al. (2020)-style models provided [here](#) on DPAv4, ASCADv1-fixed, and AES-HD.

We implemented GradVis based on the description in Masure et al. (2019). We used Captum (Kokhlikyan et al., 2020) for the remaining neural net attribution algorithms.

Training and hyperparameter tuning We select hyperparameters by doing a random search for 50 trials and selecting the ‘best-performing’ configuration on a validation dataset consisting of 20% of the training datapoints. Here, as is common in side-channel analysis, most of our models attain near-random accuracy on their validation datasets, so we instead quantify performance with the *rank* of the correct label (i.e. the number of labels to which the model assigns at least as much mass as the correct label; lower is better) on average over the validation dataset. We follow the following procedure to select the ‘best-performing’ model:

1. Note the best rank achieved over every epoch of every trial.
2. Discard all trials for which the minimum rank over all epochs exceeds $1.01 \times$ the best rank.
3. Select the model with the best minimum validation loss over all epochs.

This procedure is intended to break ties when many configurations attain similarly-small ranks, as is the case for DPAv4, OTiAiT, and OTP. The search spaces and chosen hyperparameter configurations are shown in table 1.

For all models we use the AdamW optimizer (Loshchilov & Hutter, 2018) and cross-entropy loss function. Weight decay is applied only to weights and not to biases or BatchNorm statistics. Datapoints are standardized feature-wise based on the sample statistics of the profiling (training + validation) dataset. All model weights are initialized with `torch.nn.init.xavier_uniform_` (Glorot & Bengio, 2010). We find that initializing the output weights this way is essential, and many architecture/dataset combinations will completely fail to generalize with the default PyTorch initialization. After training, we select the model checkpoint at which the validation rank was minimal. Unless otherwise specified, hyperparameters are left at their default PyTorch values.

After selecting the hyperparameter configurations, we train 5 models with different random seeds. The resulting training curves are shown in Fig. 17. For the AES datasets, we plot in Fig. 18 the evolution of the rank over time while accumulating the predictions as is often done in the context of side-channel attacks.

C.3.3. PROCEDURE FOR ADVERSARIAL LEAKAGE LOCALIZATION

Architecture of the classifiers Φ For fairness, we use a very similar architecture for Φ as for the supervised classifiers: a multilayer perceptron with 3 500-neuron hidden layers and ReLU hidden activations. Φ takes 2 inputs: the ‘masked’ trace $\mathcal{A}_\gamma \odot \mathbf{X}$, and the mask \mathcal{A}_γ . We simply concatenate both inputs along the channel dimension and feed them to the multilayer perceptron. Similarly to for the supervised learning methods, we originally tried more-sophisticated VGG and ResNet-style architectures, but found them to work poorly compared to the multilayer perceptron.

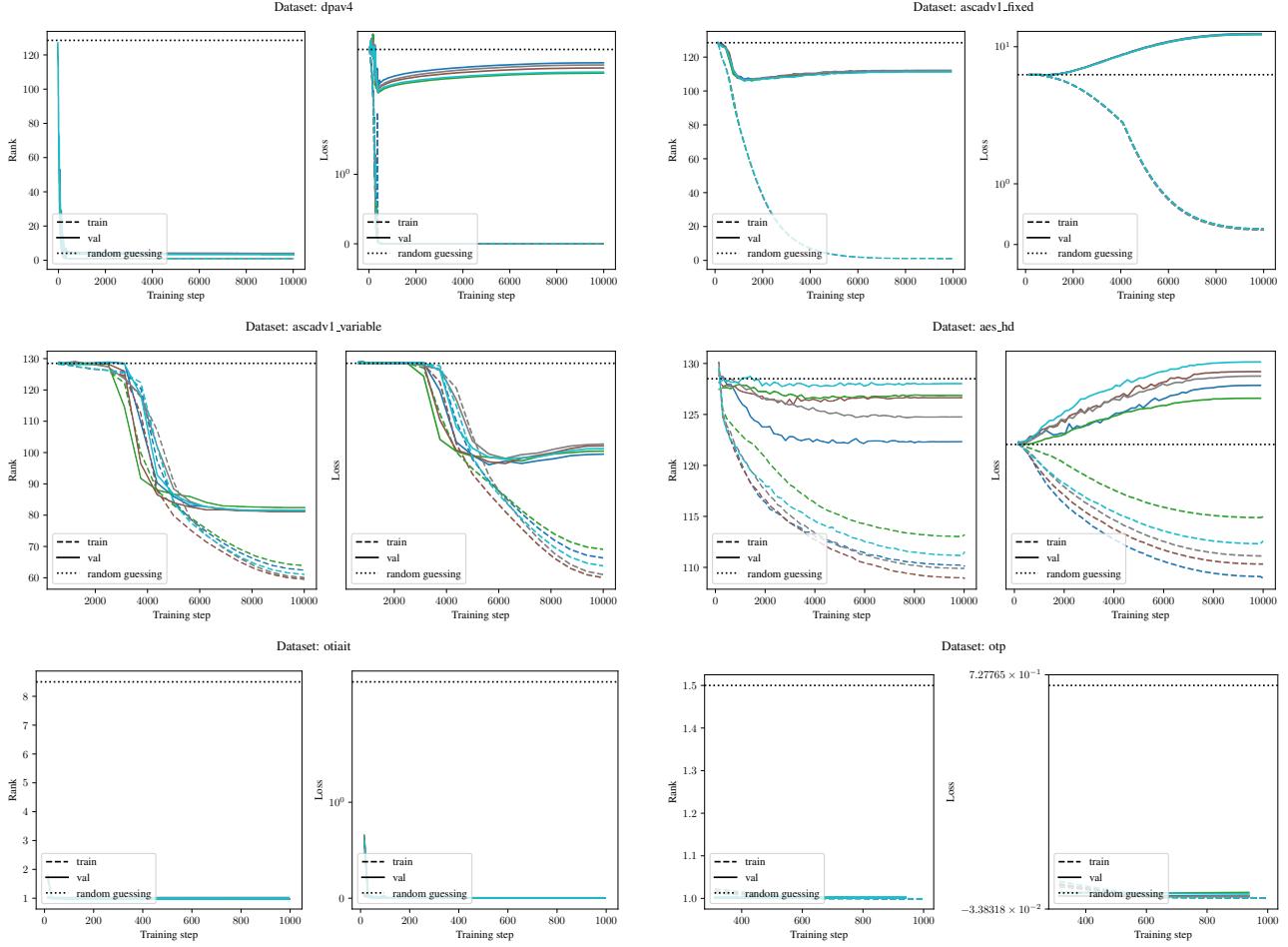


Figure 17. Training curves of the classifiers used to compute the neural network attribution-based leakage assessments. Colors correspond to distinct random seeds. Dashed lines denote metrics on the training set, solid lines on the validation dataset, and dotted the metric values we would expect if we were to randomly guess the label.

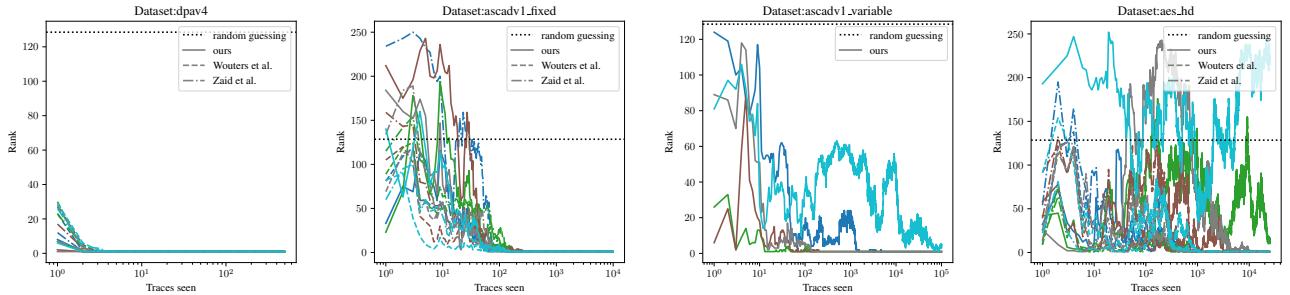


Figure 18. Ranks over time while accumulating predictions for traces in the attack dataset, all of which have the same key. Distinct colors correspond to distinct keys. We also plot ranks over time using the pretrained Zaid et al. (2020); Wouters et al. (2020)-style models from [here](#) when available.

Leakage localization through power consumption

| Hyperparameter (default) | Search space | Selected value | | |
|--|----------------------------------|----------------|--------|-------|
| | | DPAv4 (Zaid) | OTiAiT | OTP |
| Learning rate of $\tilde{\eta}$ (1e-3) | {1e-5, 1e-4, 1e-3, 1e-2} | 1e-2 | 1e-3 | 1e-3 |
| Learning rate of θ (1e-3) | { 1e-8, 1e-7, 1e-6, 1e-5, 1e-4 } | 1e-4 | 1e-6 | 1e-4 |
| $\bar{\gamma}$ (0.5) | { 0.1, 0.5, 0.9 } | 0.5 | 0.1 | 0.5 |
| Classifiers weight decay (0.0) | { 1e-2, 1, 1e2 } | 1 | 1e2 | 1e2 |
| $\tilde{\eta}$ steps per θ step (1) | { 1, 2, 4 } | 1 | 2 | 2 |
| Training steps | n/a | 1e4 | 1e4 | 1e4 |
| Minibatch size | n/a | 256 | 256 | 256 |
| Optimizer | n/a | AdamW | AdamW | AdamW |

| Hyperparameter (default) | Search space | Selected value | | |
|--|--|-----------------|--------------------|--------|
| | | ASCADv1 (fixed) | ASCADv1 (variable) | AES-HD |
| Learning rate for θ (pretrain) (1e-3) | { 1e-5, ..., 9e-5, 1e-4, 2e-4, ..., 9e-4 } | 4e-4 | 3e-4 | 5e-4 |
| β_1 for θ (pretrain) (0.9) | { 0.0, 0.5, 0.9, 0.99 } | 0.5 | 0.0 | 0.5 |
| Weight decay for θ (pretrain) (0.0) | { 0.0, 1e-4, 1e-2 } | 1e-2 | 1e-4 | 0.0 |
| Learning rate for θ (1e-3) | { 1e-2, 1e-1, 1 } | 4e-6 | 3e-5 | 5e-5 |
| Learning rate for $\tilde{\eta}$ (1e-3) | { 1e-5, 1e-4, 1e-3, 1e-2 } | 1e-4 | 1e-3 | 1e-2 |
| $\bar{\gamma}$ (0.5) | { 0.1, 0.5, 0.9 } | 0.5 | 0.5 | 0.9 |
| Classifiers weight decay (0.0) | { 1e-2, 1, 1e2 } | 1 | 0.0 | 0.0 |
| $\tilde{\eta}$ steps per θ step (1) | { 1, 2, 4 } | 2 | 1 | 2 |
| θ pretrain steps | n/a | 1e4 | 2e4 | 1e4 |
| Training steps | n/a | 2e4 | 2e4 | 1e4 |
| Minibatch size | n/a | 256 | 256 | 256 |
| Optimizer | n/a | AdamW | AdamW | AdamW |

Table 2. Results of random hyperparameter searches for our adversarial leakage localization (ALL) algorithm on considered datasets. For each dataset we run 50 trials with a random hyperparameter configuration sampled uniformly from the search space. We then select the best-performing configuration and train 5 models with different random seeds. We find that performance on ASCADv1 (fixed), ASCADv1 (variable), and AES-HD is highly sensitive to the hyperparameters of the classifiers, so instead of doing a 50-trial random search we first do a 25-trial random search for classifiers hyperparameters which minimize mean validation rank while leaving γ fixed at $0.5 \cdot \mathbf{1}$, then do a 25-trial random search for ALL hyperparameters using the optimal pretrained classifiers as a starting point.

Training and hyperparameter tuning See table 2 for our hyperparameter tuning search spaces and chosen settings. Our method requires tuning the ‘budget’ hyperparameter C , as well as optimizer settings for $\tilde{\eta}$. We find that the optimal C is sensitive to the input dimensionality T , but sensitivity is greatly reduced by instead tuning $\bar{\gamma} \in (0, 1)$ with $C = \sum_{t=1}^T c(\bar{\gamma}) = T \frac{\bar{\gamma}}{1-\bar{\gamma}}$. $\bar{\gamma} = 0.5$ appears to be a good default value. We optimize $\tilde{\eta}$ with the AdamW optimizer and find that while the learning rate must be tuned, default values are fine for the remaining optimizer settings. We tune hyperparameters to maximize the area under the curve of the reverse DNN occlusion test on a validation dataset, and do not early-stop our runs, as described in appendix C.3.3. The optimal settings generally have the learning rate of θ about $100 \times$ smaller than that of $\tilde{\eta}$, with 2 $\tilde{\eta}$ training steps taken per θ training step. Unlike GANs, which are notoriously sensitive to hyperparameters, random seed and early stopping (Brock et al., 2018), our technique does not appear excessively sensitive to these. We conjecture this is because in the ideal setting, the optimal classifier weights θ are ‘almost’ the same regardless of $\tilde{\eta}$ in the sense described in Proposition B.1.

Since the ASCADv1 (fixed/variable) and AES-HD datasets are quite sensitive to the classifier optimizer settings, we find that a single end-to-end hyperparameter tuning run works poorly because the runs will fail to converge if the classifiers don’t learn anything. Thus, we first do a 25-trial random search while optimizing for classification performance with γ fixed to $0.5 \cdot \mathbf{1}$. We then do another 25-trial random search using this pretrained classifier as a starting point while tuning $\tilde{\eta}$ as well.

C.4. Full leakage localization results

Here we list our full results. We compare the following leakage localization algorithms: our adversarial leakage localization (ALL) technique, signal to noise ratio (SNR) (Mangard et al., 2007), sum of squared difference (SoSD) (Chari et al., 2003), correlation power analysis with a Hamming weight leakage model (CPA) (Brier et al., 2004), gradient visualization (GradVis) (Masure et al., 2019), saliency (Simonyan et al., 2014; Hettwer et al., 2020), 1-occlusion (Zeiler & Fergus, 2014; Hettwer et al., 2020), layerwise relevance propagation (LRP) (Bach et al., 2015; Hettwer et al., 2020), input * gradient (Shrikumar et al., 2017; Wouters et al., 2020), as well as a random guessing baseline based on using i.i.d. Gaussian noise for each of the ‘leakiness’ measurements. Where there is a pair of citations, we have first listed the paper which introduced a technique, followed by the paper which first applied it to side-channel leakage localization. Each of these techniques has been applied to the following datasets: ASCADv1 (fixed/random) (Benadjila et al., 2020), DPAv4 (Zaid version) (Zaid et al., 2020), AES-HD (Bhasin et al., 2020), One Trace is All it Takes (Weissbart et al., 2019), and One Truth Prevails (Saito et al., 2022). In addition to models trained as described in sections ?? and C.3.3, we have applied the neural net attribution methods to the pretrained (Zaid et al., 2020; Wouters et al., 2020)-style architectures provided [here](#), where available. We have omitted LRP from these pretrained model evaluations because its Captum implementation does not work out-of-the-box on these architectures.

In table 3 we provide images of our ‘best’ ALL outputs, resulting from the hyperparameter sweep described in Section C.3.3 and corresponding to all following performance metrics and plots. We also provide images of the oGMM assessment for each random share, and information about the datasets corresponding to each image.

In table 4 we list the Spearman rank correlation coefficient between the leakage assessments and the oGMM assessment. In table 5 we list the areas under the forward DNN occlusion curves. In table 6 we list the areas under the reverse DNN occlusion curves.

We provide visualizations of the leakage assessments and plot of leakage assessments vs the oGMM assessment for all considered algorithms. Results on ASCADv1-fixed are shown in Fig. 19. Results on ASCADv1-variable are shown in Fig. 20. Results on DPAv4 (Zaid version) are shown in Fig. 21. Results on AES-HD are shown in figure 22. Results on One Trace is All it Takes (OTiAiT) are shown in Fig. 23. Results on One Truth Prevails (OTP) are shown in Fig. 24.

We provide forward and reverse DNN occlusion test visualizations for all considered algorithms. Results on ASCADv1-fixed are shown in Fig. 25. Results on ASCADv1-variable are shown in Fig. 26. Results on DPAv4 (Zaid version) are shown in Fig. 27. Results on AES-HD are shown in Fig. 28. Results on One Trace is All it Takes (OTiAiT) are shown in Fig. 29. Results on One Truth Prevails (OTP) are shown in Fig. 30.

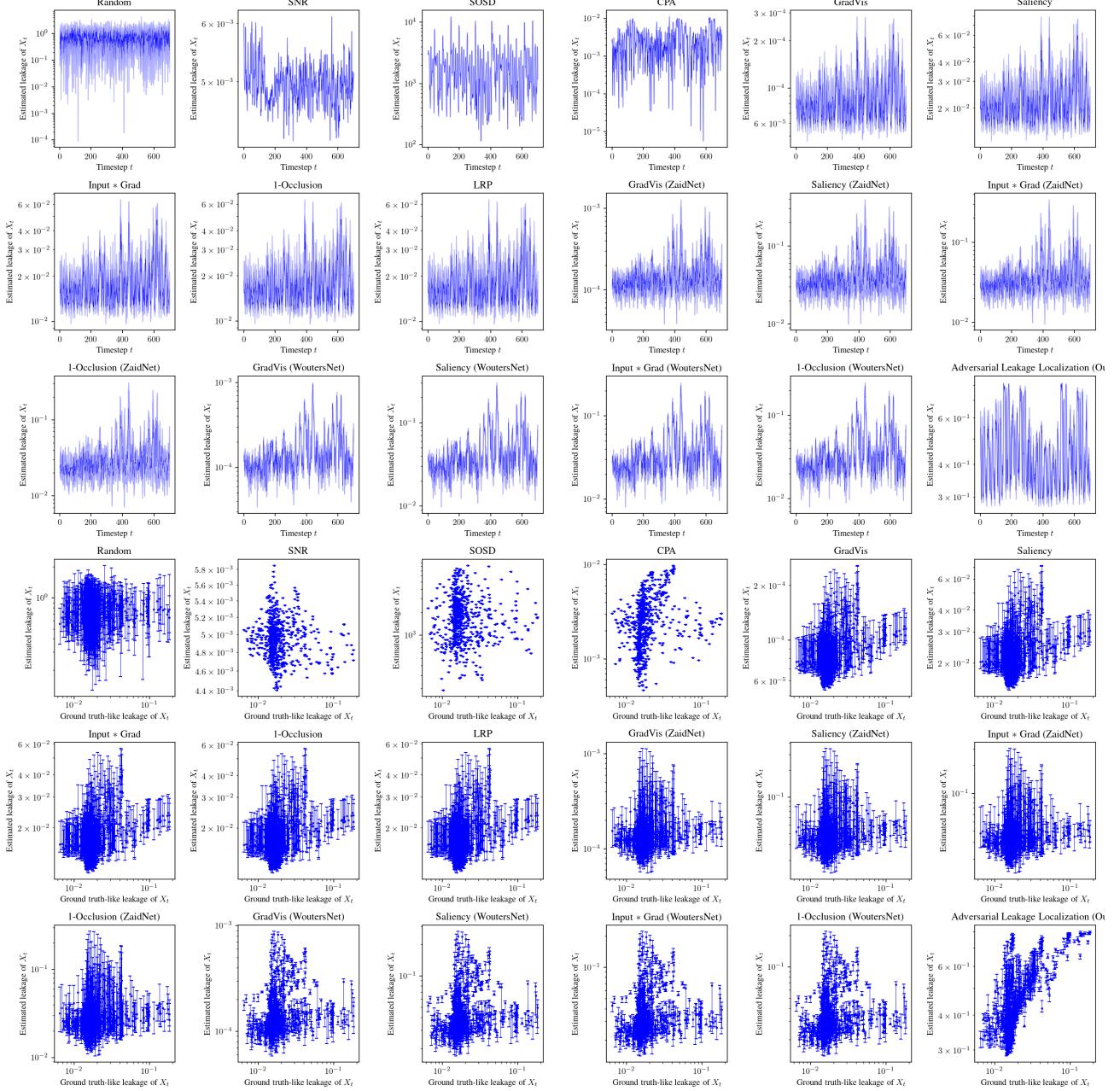


Figure 19. Leakage assessment visualizations (top half of rows) and plot of the leakage assessment vs the oGMM assessment (bottom half of rows) on the ASCADv1-fixed dataset.

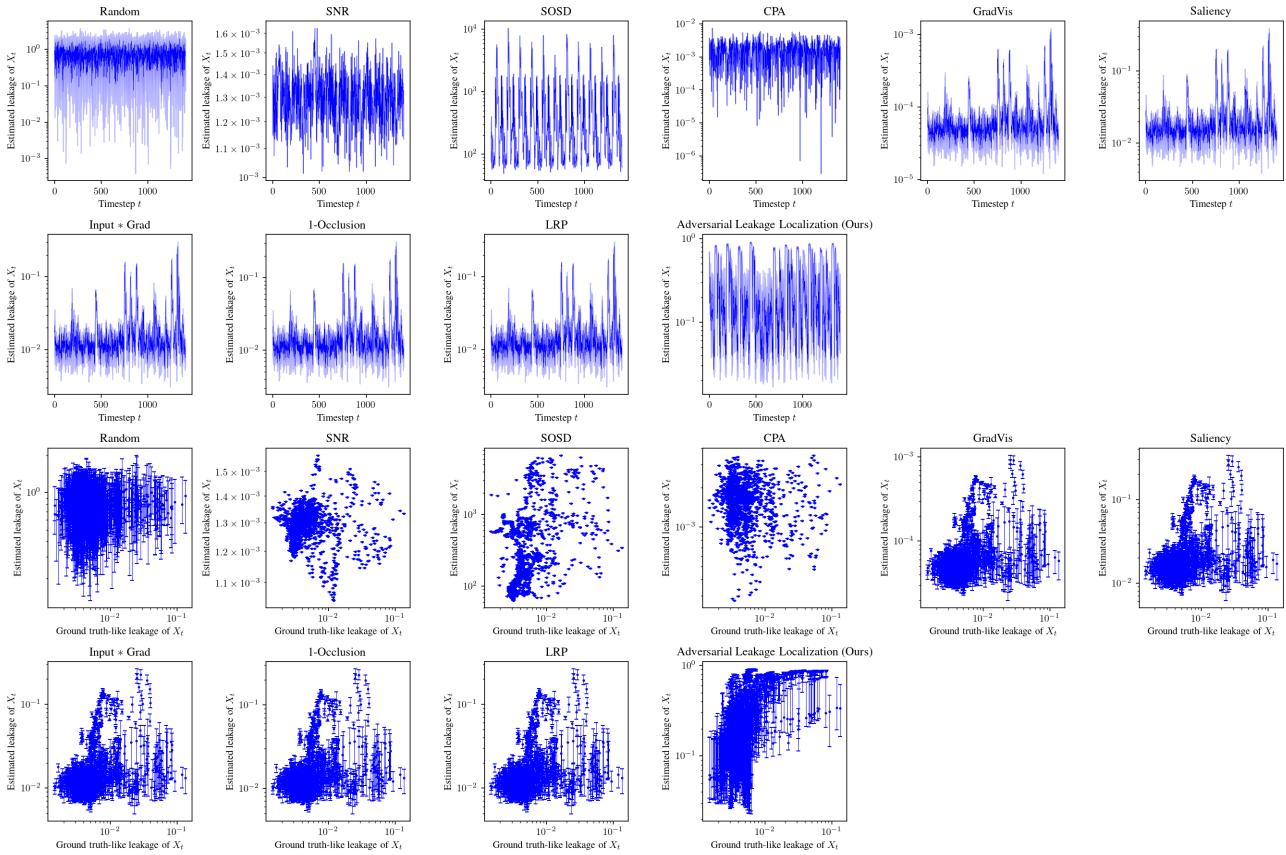


Figure 20. Leakage assessment visualizations (top half of rows) and plot of the leakage assessment vs the oGMM assessment (bottom half of rows) on the ASCADv1-variable dataset.

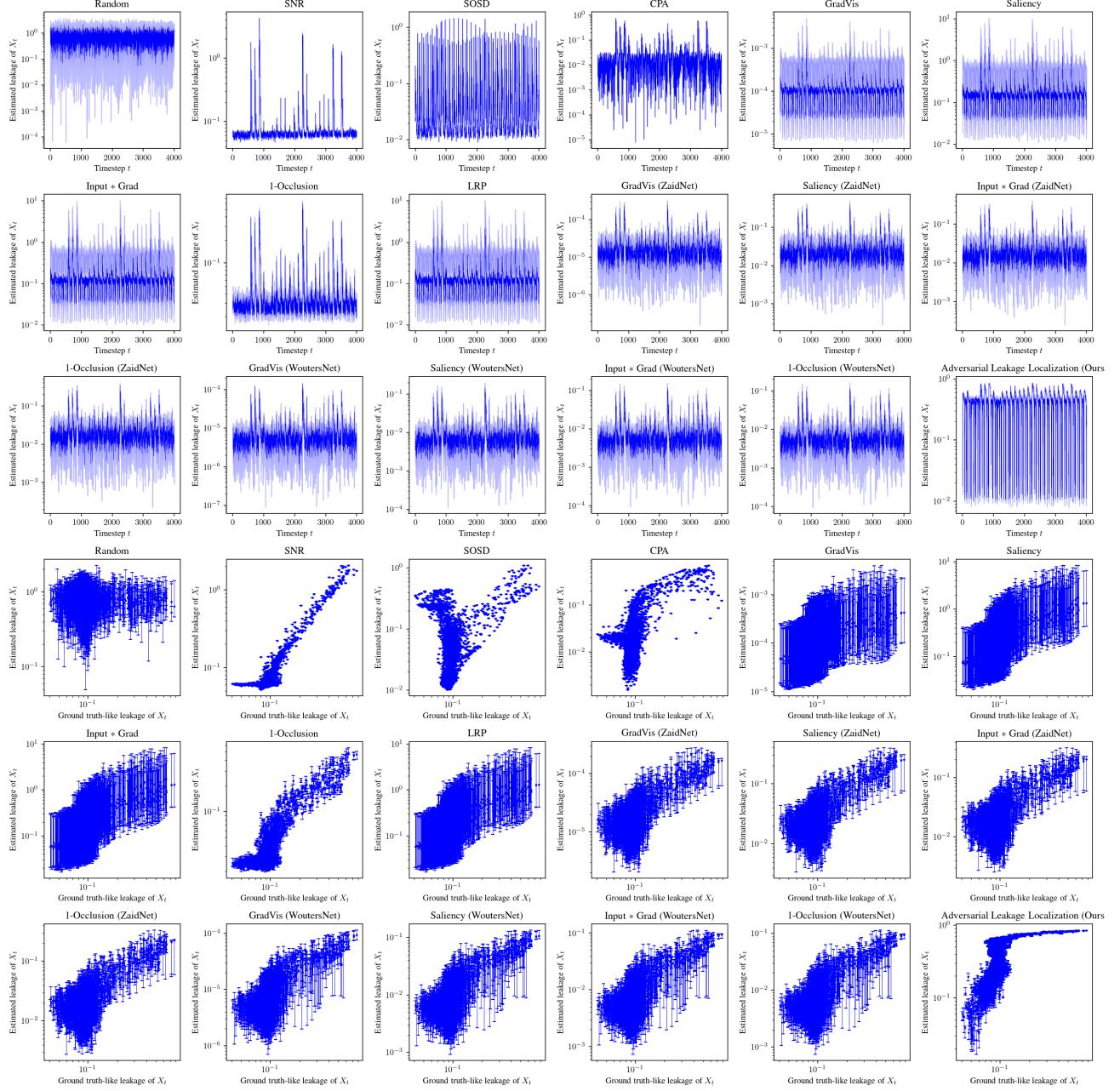


Figure 21. Leakage assessment visualizations (top half of rows) and plot of the leakage assessment vs the oGMM assessment (bottom half of rows) on the DPAv4 (Zaid version) dataset.

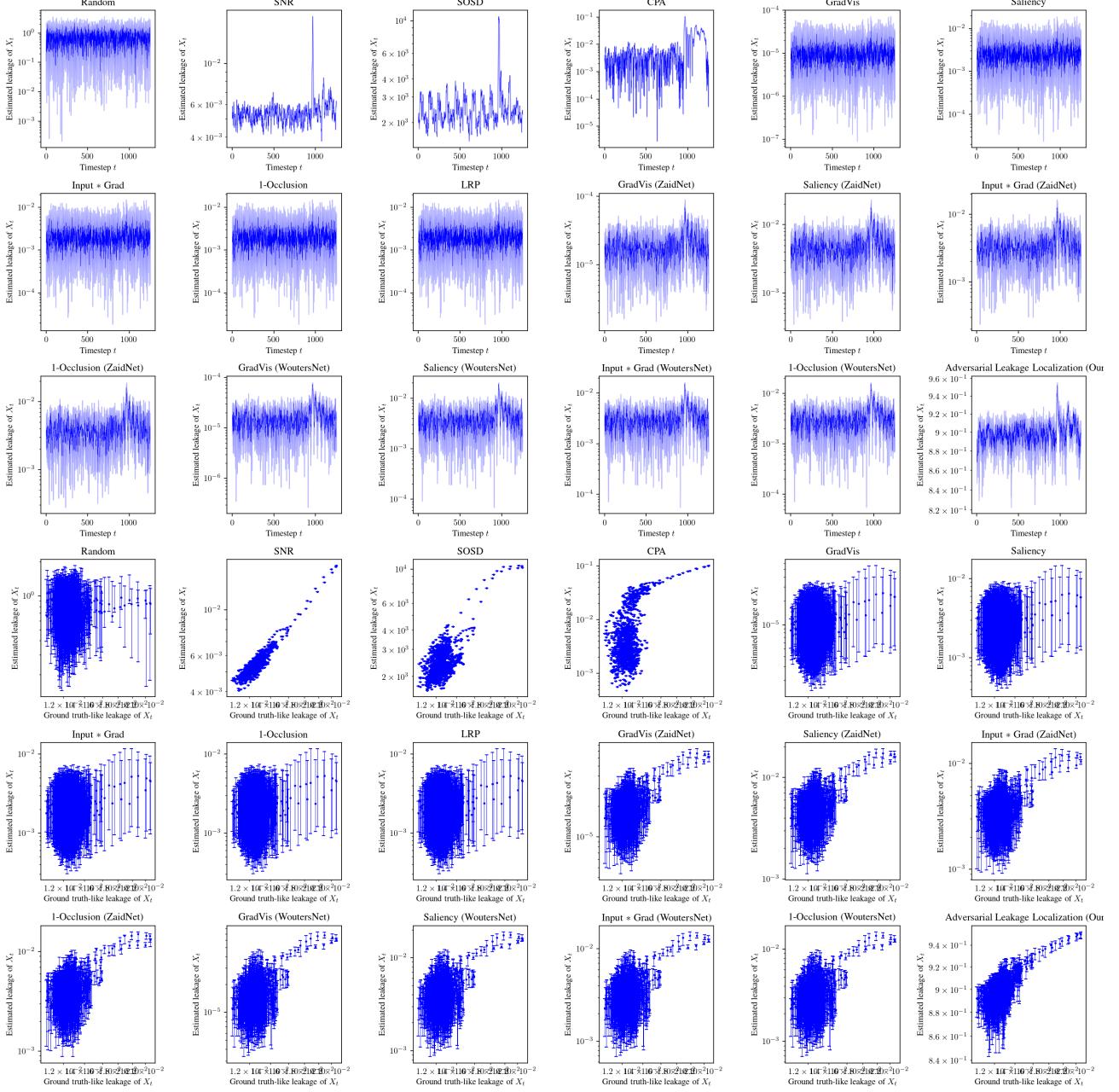


Figure 22. Leakage assessment visualizations (top half of rows) and plot of the leakage assessment vs the oGMM assessment (bottom half of rows) on the AES-HD dataset.

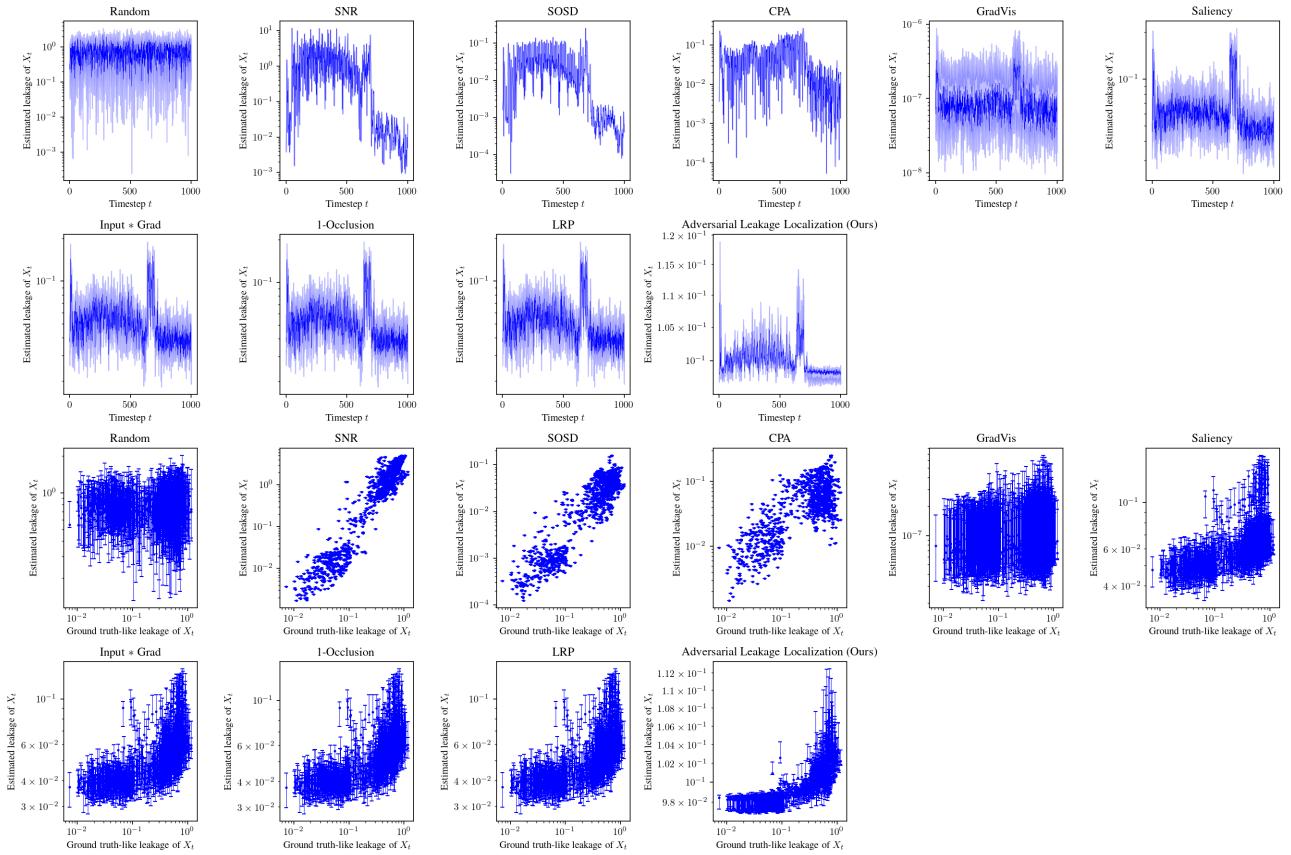


Figure 23. Leakage assessment visualizations (top half of rows) and plot of the leakage assessment vs the oGMM assessment (bottom half of rows) on the One Trace is All it Takes (OTiTAT) dataset.

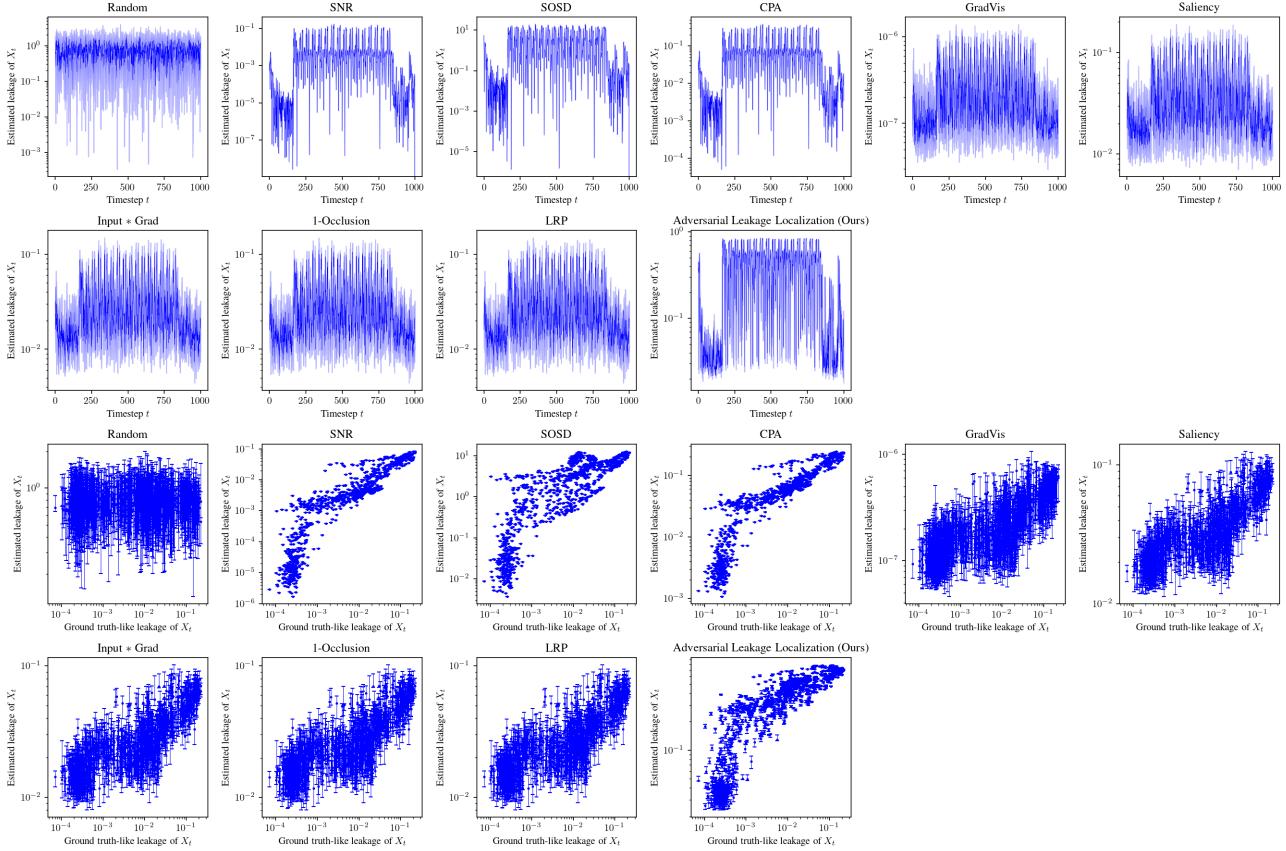


Figure 24. Leakage assessment visualizations (top half of rows) and plot of the leakage assessment vs the oGMM assessment (bottom half of rows) on the One Truth Prevails (OTP) dataset.

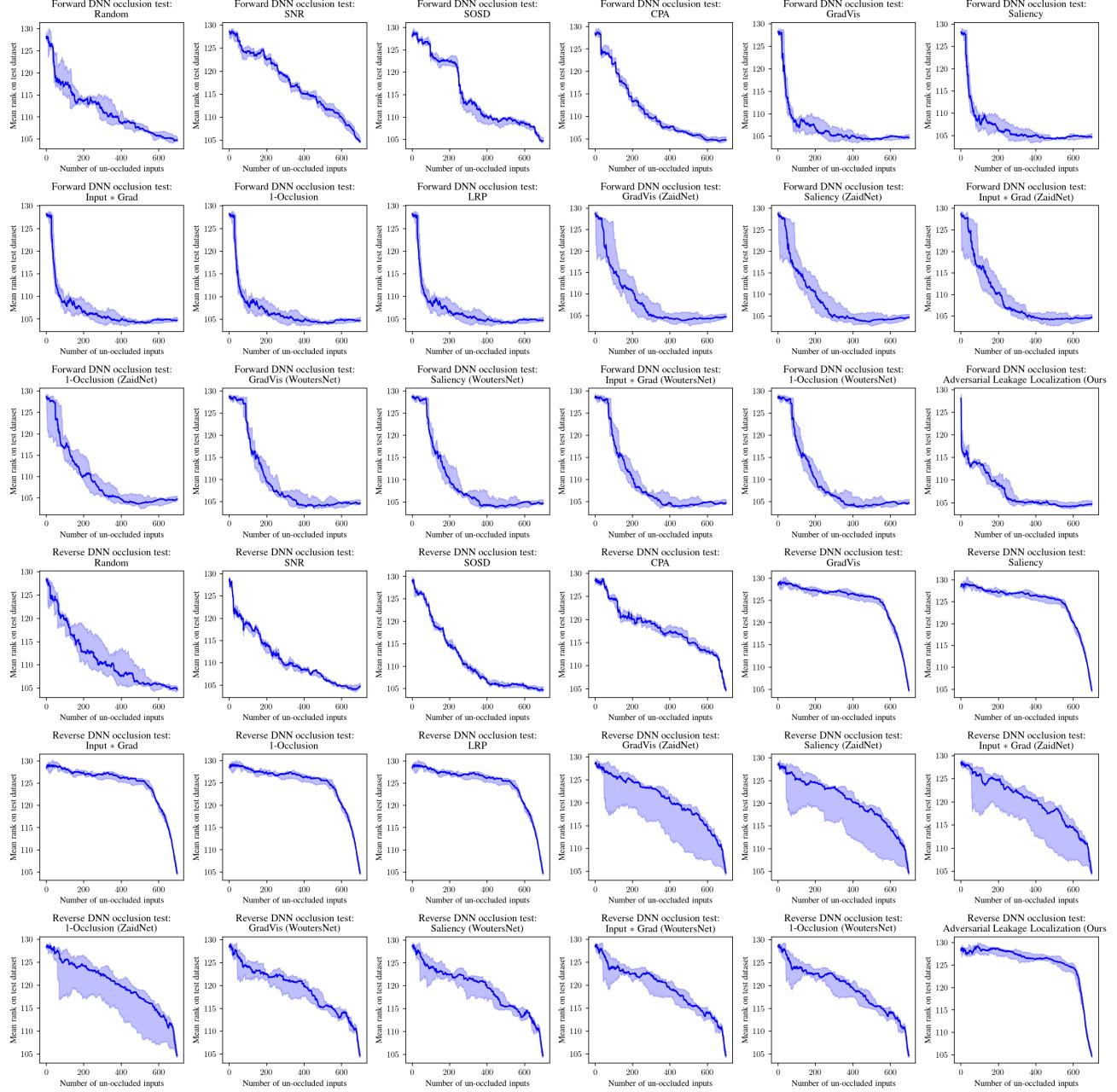


Figure 25. Forward and reverse DNN occlusion test results on the ASCADv1-fixed dataset.

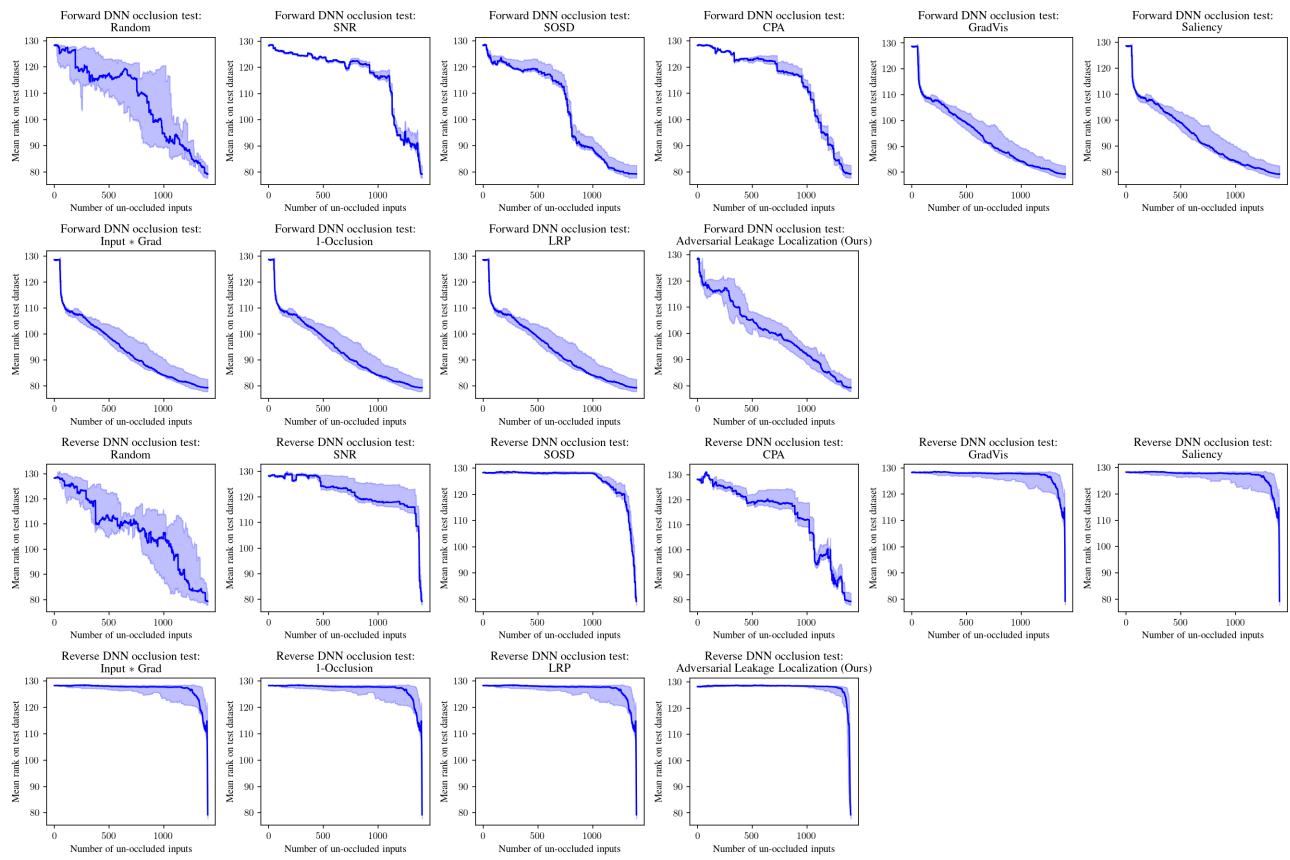


Figure 26. Forward and reverse DNN occlusion test results on the ASCADv1-variable dataset.

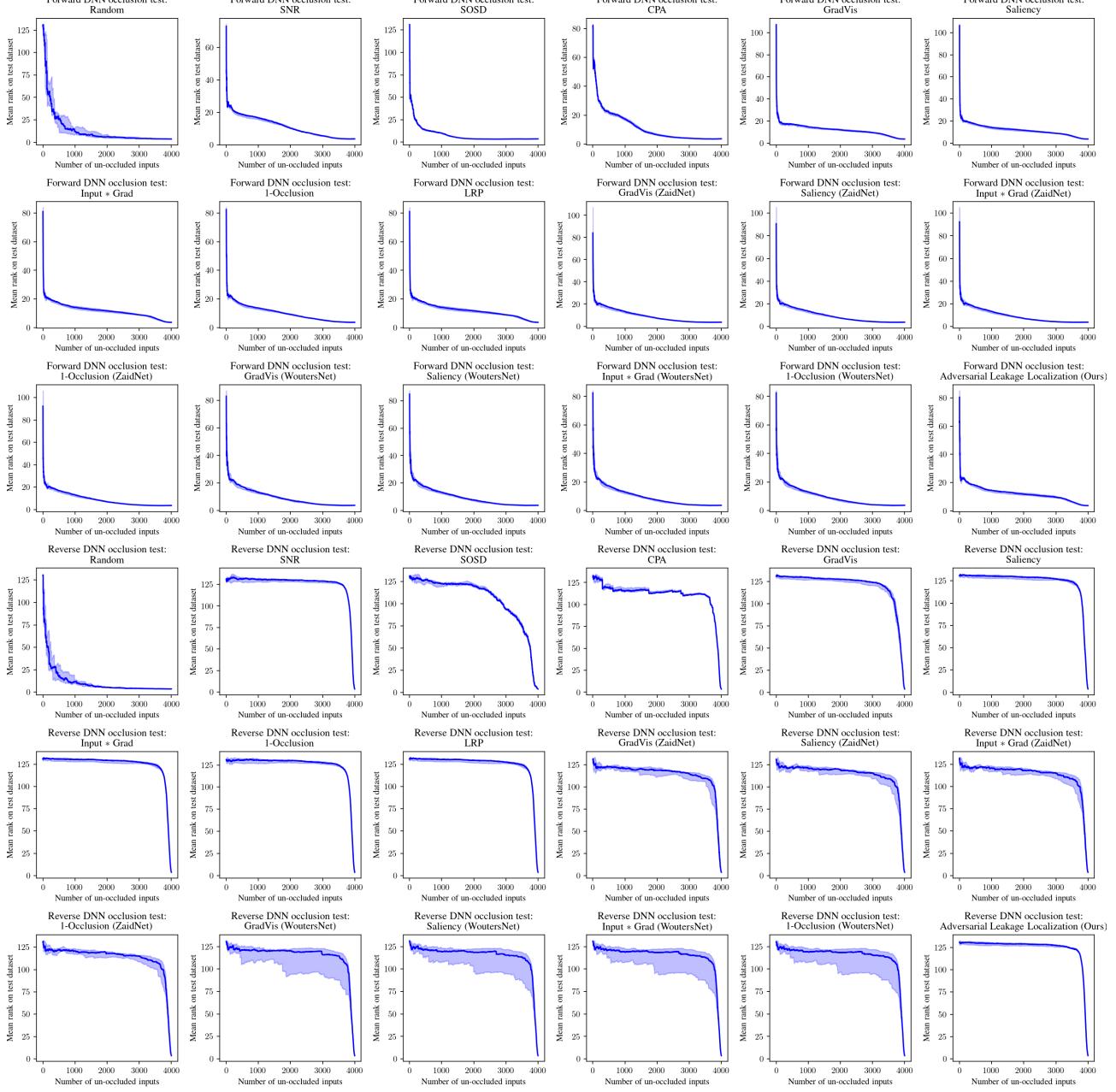


Figure 27. Forward and reverse DNN occlusion test results on the DPav4 (Zaid version) dataset.

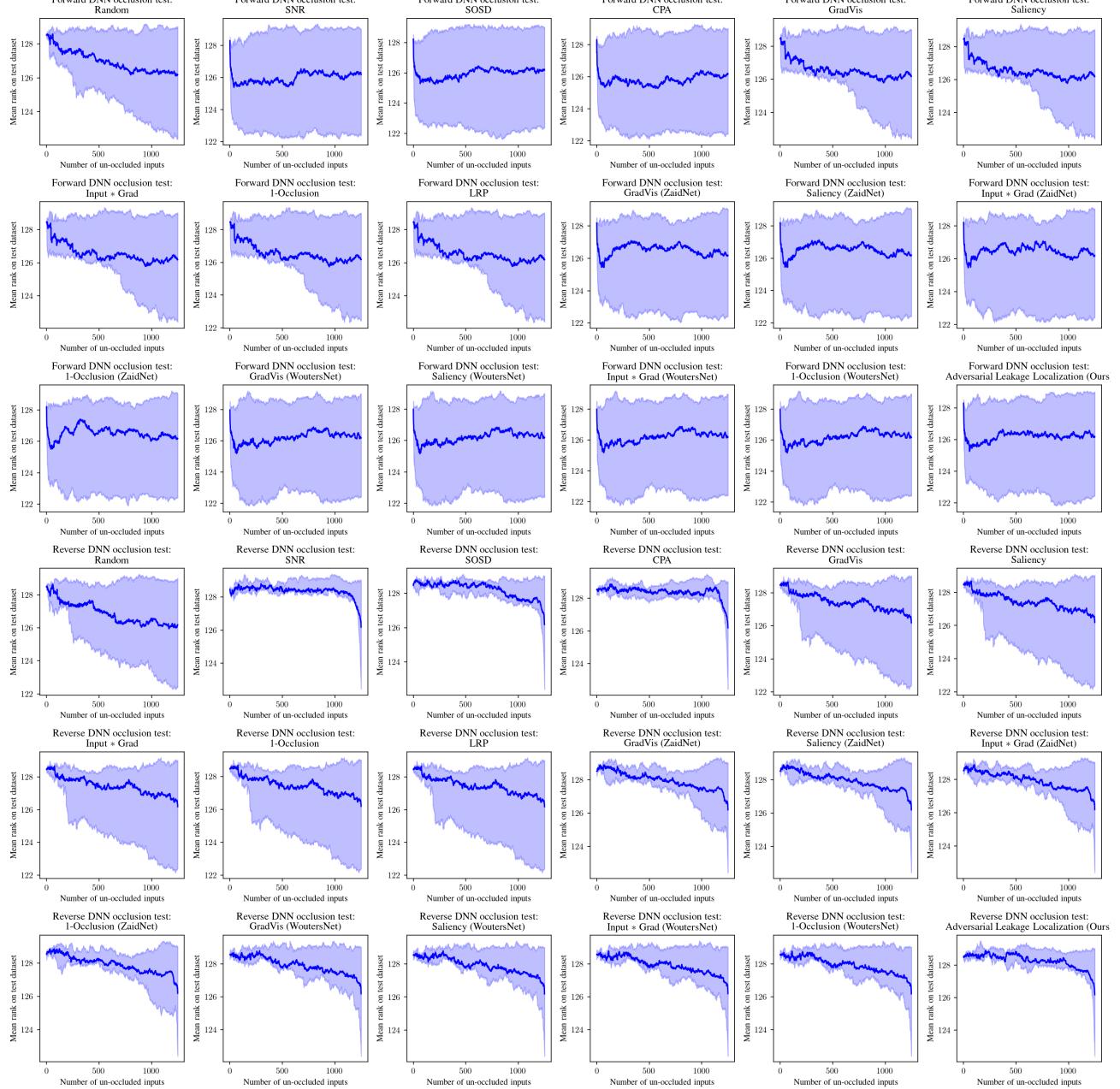


Figure 28. Forward and reverse DNN occlusion test results on the AES-HD dataset.

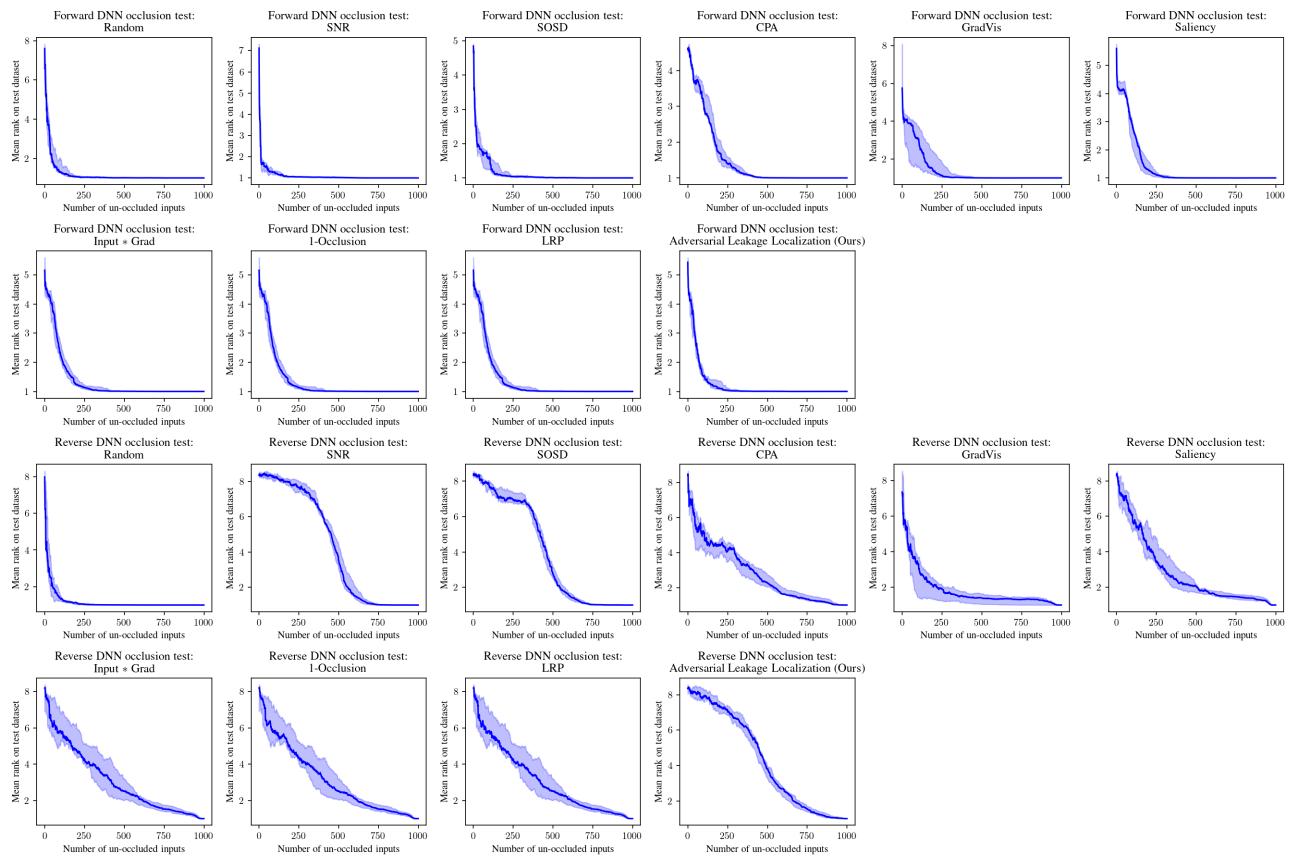


Figure 29. Forward and reverse DNN occlusion test results on the One Trace is All it Takes (OTiAiT) dataset.

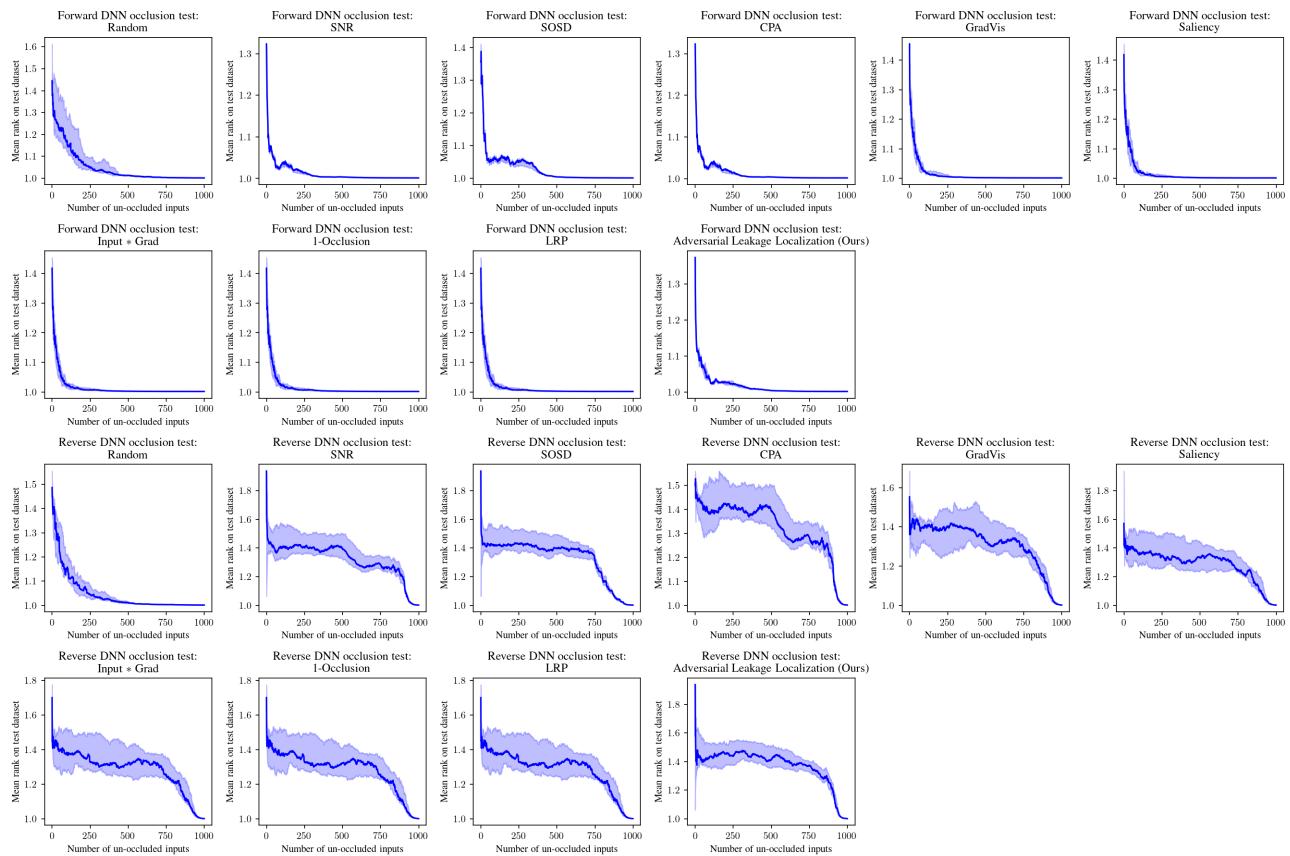


Figure 30. Forward and reverse DNN occlusion test results on the One Truth Prevails (OTP) dataset.

Table 3. Non-cherry picked outputs of our technique on various cryptographic implementations, chosen by a 50-trial random hyperparameter search. We denote by k_n , w_n , m_n , k_n^* , and c_n the n -th byte (counting from 0) of the AES key, plaintext, mask, last AES round key, and ciphertext, respectively. Dots denote median and shading denotes min – max over 5 random seeds.

| Dataset | ASCADv1 (fixed key) | ASCADv1 (variable key) | DPAv4 (Zaid version) |
|---------------------------|---|---|---|
| Reference/link | (Benadjila et al., 2020) (link) | (Benadjila et al., 2020) (link) | (Zaid et al., 2020) (link) |
| Algorithm | AES-128 | AES-128 | AES-128 |
| Hardware | ATMega8515 (MCU) | ATMega8515 (MCU) | ATMega 163 (MCU) |
| Quantity measured | Power | Power | Power |
| Countermeasures | Boolean masking | Boolean masking | Rotating SBox mask (known) |
| Sensitive variable | $\text{Sbox}(k_3 \oplus w_3)$ | $\text{Sbox}(k_3 \oplus w_3)$ | $\text{Sbox}(k_0 \oplus w_0) \oplus m_0$ |
| | | | |
| Dataset | AES-HD | OTiAiT | OTP |
| Reference/link | (Bhasin et al., 2020) (link) | (Weissbart et al., 2019) (link) | (Saito et al., 2022) (link) |
| Algorithm | AES-128 | EdDSA w/ Curve2559 | 1024-bit RSA-CRT |
| Hardware | Xilinx Virtex-5 (FPGA) | STM32F4 (MCU) | STM32F4 (MCU) |
| Quantity measured | EM radiation | Power | EM radiation |
| Countermeasures | None | None | Dummy load |
| Sensitive variable | $\text{Sbox}^{-1}(k_{11}^* \oplus c_{11}) \oplus c_7$ | Ephemeral key nibble | Dummy load? |

Leakage localization through power consumption

| Method | Dataset | | | | | |
|--------------------------------|---------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|---------------------------------------|
| | ASCADv1 (fixed) | ASCADv1 (random) | DPAv4 (Zaid version) [†] | AES-HD [†] | OTiAiT [†] | OTP [†] |
| Random | -0.0318 ± 0.0146 | -0.0126 ± 0.0463 | -0.0265 ± 0.0236 | -0.053 ± 0.046 | 0.0132 ± 0.0470 | 0.00238 ± 0.03361 |
| First-order parametric methods | SNR | -0.0862 | 0.0534 | <u>0.650</u> | <u>0.823</u> | <u>0.892</u> |
| | SOSD | -0.0455 | 0.448 | 0.142 | 0.418 | 0.813 |
| | CPA | 0.374 | -0.123 | 0.405 | 0.303 | 0.625 |
| Neural net attribution | GradVis | 0.369 ± 0.023 | 0.367 ± 0.069 | 0.416 ± 0.008 | 0.0803 ± 0.0571 | 0.340 ± 0.090 |
| | Saliency | 0.367 ± 0.023 | 0.364 ± 0.065 | <u>0.443 ± 0.005</u> | 0.0803 ± 0.0570 | 0.692 ± 0.037 |
| | Occlusion | 0.375 ± 0.023 | 0.357 ± 0.064 | 0.428 ± 0.016 | 0.0783 ± 0.0575 | 0.757 ± 0.020 |
| | Input * Grad | 0.374 ± 0.023 | 0.356 ± 0.065 | <u>0.444 ± 0.005</u> | 0.05 ± 0.04 | 0.752 ± 0.021 |
| | LRP | 0.374 ± 0.023 | 0.356 ± 0.065 | <u>0.444 ± 0.005</u> | 0.0787 ± 0.0577 | 0.752 ± 0.021 |
| | GradVis (ZaidNet) | 0.284 ± 0.047 | | 0.269 ± 0.023 | 0.154 ± 0.032 | |
| | Saliency (ZaidNet) | 0.285 ± 0.041 | | 0.265 ± 0.024 | 0.154 ± 0.033 | |
| | Input * Grad (ZaidNet) | 0.269 ± 0.042 | | 0.256 ± 0.024 | 0.149 ± 0.030 | |
| | Occlusion (ZaidNet) | 0.269 ± 0.045 | | 0.258 ± 0.023 | 0.150 ± 0.033 | |
| | GradVis (WoutersNet) | 0.330 ± 0.054 | | 0.278 ± 0.033 | 0.149 ± 0.047 | |
| | Saliency (WoutersNet) | 0.331 ± 0.052 | | 0.276 ± 0.032 | 0.149 ± 0.046 | |
| | Input * Grad (WoutersNet) | 0.332 ± 0.050 | | 0.268 ± 0.034 | 0.146 ± 0.046 | |
| | Occlusion (WoutersNet) | 0.331 ± 0.050 | | 0.268 ± 0.034 | 0.146 ± 0.047 | |
| | ALL (ours) | <u>0.622 ± 0.024</u> | <u>0.535 ± 0.089</u> | 0.425 ± 0.005 | <u>0.458 ± 0.018</u> | <u>0.845 ± 0.003</u> |
| | | | | | | <u>0.9182 ± 0.0007</u> |

Table 4. Spearman correlation coefficient with the ‘omniscient’ Gaussian mixture model (oGMM) leakage assessment (higher is better). Our Adversarial Leakage Localization (ALL) technique outperforms all previously-proposed deep learning-based leakage localization techniques on 5 of the 6 considered datasets, and outperforms first-order statistical methods on datasets with Boolean masking. Results are reported as mean \pm standard deviation over 5 random seeds, using the best hyperparameters from a 50-trial random search. The boxed result indicates the best-performing method, and the underlined result denotes the best-performing deep learning-based method. We also run the neural net attribution techniques using [these](#) pretrained models with the architectures of [Zaid et al. \(2020\)](#) (denoted ZaidNet) and [Wouters et al. \(2020\)](#) (denoted WoutersNet) where available. [†]Since these datasets have primarily first-order leakage, it is unsurprising that first-order statistical methods outperform deep learning methods. We include the results to demonstrate that ALL outperforms prior deep learning methods and that it works on diverse cryptographic implementations.

| Method | Dataset | | | | | |
|--------------------------------|---------------------------|-----------------------------------|------------------------------|---------------------------------|-------------------------------|-----------------------------------|
| | ASCADv1 (fixed) | ASCADv1 (random) | DPAv4 (Zaid version) | AES-HD | OTiAiT | OTP |
| Random | 111.6 ± 0.3 | 108 ± 5 | 13 ± 2 | <u>127 ± 1</u> | 1.21 ± 0.04 | 1.05 ± 0.02 |
| First-order parametric methods | SNR | 117.2 ± 0.6 | 116.7 ± 0.7 | 11.4 ± 0.2 | <u>126 ± 2</u> | <u>1.10 ± 0.02</u> |
| | SOSD | 114.9 ± 0.5 | 105 ± 2 | <u>8.0 ± 0.8</u> | <u>126 ± 2</u> | 1.14 ± 0.03 |
| | CPA | 111.5 ± 0.4 | 114 ± 1 | 11.5 ± 0.3 | <u>126 ± 2</u> | <u>1.49 ± 0.04</u> |
| Neural net attribution | GradVis | 107.0 ± 0.5 | <u>95 ± 2</u> | 12.1 ± 0.3 | <u>127 ± 1</u> | 1.4 ± 0.2 |
| | Saliency | 107.1 ± 0.5 | <u>95 ± 2</u> | 11.8 ± 0.3 | <u>127 ± 1</u> | 1.39 ± 0.04 |
| | Occlusion | 107.1 ± 0.5 | <u>95 ± 2</u> | 10.1 ± 0.2 | <u>127 ± 1</u> | 1.36 ± 0.04 |
| | Input * Grad | <u>107.2 ± 0.5</u> | <u>95 ± 2</u> | 11.8 ± 0.4 | <u>127 ± 1</u> | 1.36 ± 0.04 |
| | LRP | <u>107.2 ± 0.5</u> | <u>95 ± 2</u> | 11.8 ± 0.4 | <u>127 ± 1</u> | 1.36 ± 0.04 |
| | GradVis (ZaidNet) | 108.8 ± 0.8 | | <u>9.3 ± 0.2</u> | <u>126 ± 2</u> | |
| | Saliency (ZaidNet) | 108.8 ± 0.8 | | <u>9.3 ± 0.2</u> | <u>126 ± 2</u> | |
| | Input * Grad (ZaidNet) | 109.0 ± 0.6 | | <u>9.2 ± 0.2</u> | <u>126 ± 2</u> | |
| | Occlusion (ZaidNet) | 109.3 ± 0.6 | | <u>9.2 ± 0.2</u> | <u>126 ± 2</u> | |
| | GradVis (WoutersNet) | 109.9 ± 0.5 | | <u>9.6 ± 0.3</u> | <u>126 ± 2</u> | |
| | Saliency (WoutersNet) | 109.8 ± 0.5 | | <u>9.6 ± 0.3</u> | <u>126 ± 2</u> | |
| | Input * Grad (WoutersNet) | 109.7 ± 0.5 | | <u>9.4 ± 0.3</u> | <u>126 ± 2</u> | |
| | Occlusion (WoutersNet) | 109.7 ± 0.4 | | <u>9.4 ± 0.3</u> | <u>126 ± 2</u> | |
| | ALL (ours) | <u>107.5 ± 0.3</u> | 101 ± 2 | 12.2 ± 0.4 | <u>126 ± 2</u> | <u>1.23 ± 0.03</u> |
| | | | | | | 1.0161 ± 0.0009 |

Table 5. Area under the curve of the forward DNN occlusion test (lower is better). Our Adversarial Leakage Localization (ALL) technique outperforms all previously-proposed deep learning-based leakage localization techniques on 5 of the 6 considered datasets, and outperforms first-order statistical methods on datasets with Boolean masking. Results are reported as mean \pm standard deviation over 5 random seeds, using the best hyperparameters from a 50-trial random search. The boxed result indicates the best-performing method, and the underlined result denotes the best-performing deep learning-based method. We also run the neural net attribution techniques using [these](#) pretrained models with the architectures of [Zaid et al. \(2020\)](#) (denoted ZaidNet) and [Wouters et al. \(2020\)](#) (denoted WoutersNet) where available.

| | Method | Dataset | | | | | |
|--------------------------------------|---------------------------|--------------------|--------------------|----------------------|--------------------|--------------------|--------------------|
| | | ASCADv1 (fixed) | ASCADv1 (random) | DPAv4 (Zaid version) | AES-HD | OTiAiT | OTP |
| | Random | 112 ± 1 | 108 ± 4 | 11.5 ± 0.4 | 127 ± 1 | 1.20 ± 0.05 | 1.048 ± 0.008 |
| First-order parametric methods | SNR | 111.0 ± 0.2 | 123 ± 2 | <u>126 ± 1</u> | <u>128.5 ± 0.3</u> | <u>4.26 ± 0.07</u> | 1.33 ± 0.04 |
| | SOSD | 111.6 ± 0.2 | 125.6 ± 0.6 | <u>105.7 ± 0.9</u> | <u>128.3 ± 0.3</u> | <u>3.94 ± 0.08</u> | 1.34 ± 0.04 |
| | CPA | 118.2 ± 0.4 | 114 ± 2 | 111.5 ± 0.9 | <u>128.4 ± 0.3</u> | <u>2.7 ± 0.2</u> | 1.32 ± 0.04 |
| Neural net attribution | GradVis | 124.9 ± 0.3 | 127 ± 1 | 121 ± 1 | 127 ± 1 | 1.8 ± 0.2 | 1.31 ± 0.05 |
| | Saliency | 124.8 ± 0.3 | 127 ± 1 | <u>124 ± 1</u> | 127 ± 1 | 2.8 ± 0.2 | 1.29 ± 0.05 |
| | Occlusion | 124.8 ± 0.3 | 127 ± 1 | <u>124 ± 1</u> | 127 ± 1 | 3.2 ± 0.2 | 1.29 ± 0.05 |
| | Input * Grad | 124.8 ± 0.3 | 127 ± 1 | <u>124 ± 1</u> | 127 ± 1 | 3.1 ± 0.2 | 1.29 ± 0.05 |
| | LRP | 124.8 ± 0.3 | 127 ± 1 | <u>124 ± 1</u> | 127 ± 1 | 3.1 ± 0.2 | 1.29 ± 0.05 |
| | GradVis (ZaidNet) | 119 ± 3 | | 113 ± 2 | 128.0 ± 0.5 | | |
| | Saliency (ZaidNet) | 119 ± 3 | | 113 ± 2 | 128.0 ± 0.5 | | |
| | Input * Grad (ZaidNet) | 119 ± 2 | | 113 ± 2 | 128.0 ± 0.5 | | |
| | Occlusion (ZaidNet) | 119 ± 2 | | 113 ± 2 | 128.0 ± 0.5 | | |
| | GradVis (WoutersNet) | 119.2 ± 0.9 | | 112 ± 7 | <u>128.1 ± 0.6</u> | | |
| | Saliency (WoutersNet) | 119.3 ± 0.9 | | 112 ± 7 | <u>128.1 ± 0.5</u> | | |
| | Input * Grad (WoutersNet) | 119.3 ± 0.9 | | 112 ± 7 | <u>128.1 ± 0.6</u> | | |
| | Occlusion (WoutersNet) | 119.3 ± 0.9 | | 112 ± 7 | <u>128.1 ± 0.6</u> | | |
| ALL (ours) | | <u>125.5 ± 0.4</u> | <u>127.6 ± 0.3</u> | <u>124.5 ± 0.8</u> | <u>128.4 ± 0.3</u> | <u>4.3 ± 0.1</u> | <u>1.38 ± 0.04</u> |

Table 6. Area under the curve of the reverse DNN occlusion test (higher is better). Our Adversarial Leakage Localization (ALL) technique outperforms all previously-proposed deep learning-based leakage localization techniques on 5 of the 6 considered datasets, and outperforms first-order statistical methods on datasets with Boolean masking. Results are reported as mean ± standard deviation over 5 random seeds, using the best hyperparameters from a 50-trial random search. The boxed result indicates the best-performing method, and the underlined result denotes the best-performing deep learning-based method. We also run the neural net attribution techniques using these pretrained models with the architectures of Zaid et al. (2020) (denoted ZaidNet) and Wouters et al. (2020) (denoted WoutersNet) where available.