Gradient Visualization for General Characterization in Profiling Attacks

Loïc Masure[1, 2] Cécile Dumas[1] Emmanuel Prouff[2, 3]

[1]Univ. Grenoble Alpes, CEA, LETI, DSYS, CESTI, F-38000 Grenoble
loic.masure@cea.fr

[2]Sorbonne Universités, UPMC Univ. Paris 06, CNRS, INRIA, Laboratoire d'Informatique de Paris 6 (LIP6), Équipe PolSys, 4 place Jussieu, 75252 Paris Cedex 05, France

[3]ANSSI, France

Friday, April 5th 2019, COSADE, Darmstadt

FROM RESEARCH TO INDUSTRY

cea tech

leti

Outline

1. Context
2. The Neural Networks paradigm
3. Characterization with gradient visualization
4. Experimental results

Context

About me: PhD student, working on Statistical Learning applied to Side Channel Analysis

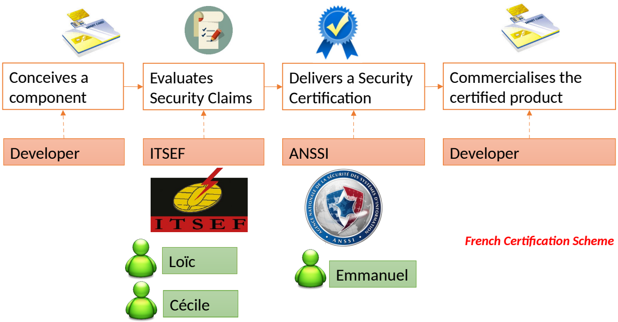


Figure: French certification scheme

Evaluating Side-Channel Vulnerabilities

Evaluating worst-case scenarios from a developer point of view.

Open samples

- *Open samples* are admitted for evaluation
- They are used to previously characterize the behaviour of the device $\Rightarrow$ *Profiling Attacks*

Profiling: two steps

1. Characterization with statistical tools (SNR, T-Test, χ^2, ...)
2. Profiling with Generative models: Template Attacks

Evaluating Side-Channel Vulnerabilities

Evaluating worst-case scenarios from a developer point of view.

Open samples

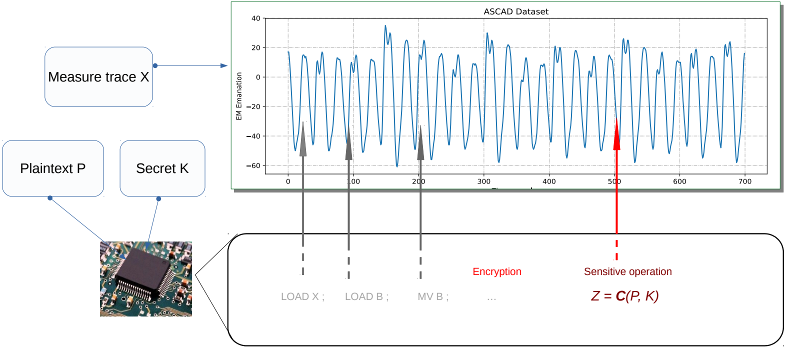
- *Open samples* are admitted for evaluation
- They are used to previously characterize the behaviour of the device $\Rightarrow$ *Profiling Attacks*

Profiling **with Deep Learning**: two steps

1. Characterization with statistical tools (SNR, T-Test, χ^2, ...)
2. Profiling with **Discriminative** models: **Convolutional Neural Networks**

Outline

Notations in Side-Channel Analysis



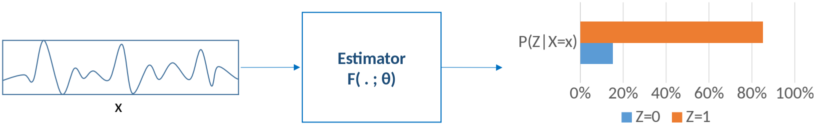
Profiling Attacks

Profiling step

Follows Maximum Likelihood principles

Requires to know the probability distribution $F^* \triangleq \Pr[Z|\mathbf{X}]$

Reality: unknown for the evaluator/attacker. Estimation with parametric models $F(., \theta^*)$!



Why Deep learning?

SCA suits the DL framework

Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model

Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model
- DL does not require too much prior knowledge (e.g. leakage model)

Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model
- DL does not require too much prior knowledge (e.g. leakage model)
- DL shown to be robust against some counter-measures

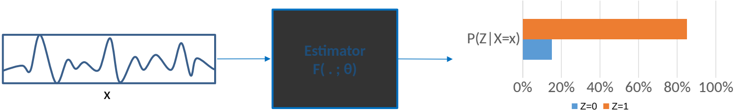
Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model
- DL does not require too much prior knowledge (e.g. leakage model)
- DL shown to be robust against some counter-measures

New problematics

Deep Learning provides black-box models:



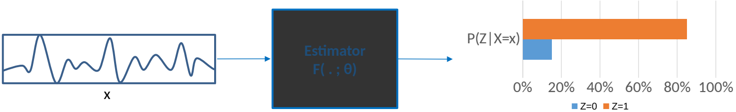
Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model
- DL does not require too much prior knowledge (e.g. leakage model)
- DL shown to be robust against some counter-measures

New problematics

Deep Learning provides black-box models:



Lack of posterior knowledge about the learned leakage model: how did the model learn?

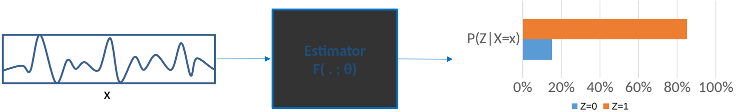
Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model
- DL does not require too much prior knowledge (e.g. leakage model)
- DL shown to be robust against some counter-measures

New problematics

Deep Learning provides black-box models:



Lack of posterior knowledge about the learned leakage model: how did the model learn?
Lack of trust on the Deep Learning tools: where did the model get the information?

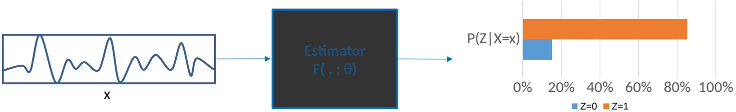
Why Deep learning?

SCA suits the DL framework

- Profiling a target device $\sim$ training a DL model
- DL does not require too much prior knowledge (e.g. leakage model)
- DL shown to be robust against some counter-measures

New problematics

Deep Learning provides black-box models:



Lack of posterior knowledge about the learned leakage model: how did the model learn?
Lack of trust on the Deep Learning tools: where did the model get the information? **Issue addressed in this talk!**

FROM RESEARCH TO INDUSTRY

Outline

Our Contribution: the Gradient Visualization

- We propose a characterization technique based on a trained CNN

Our Contribution: the Gradient Visualization

- We propose a characterization technique based on a trained CNN
- Able to detect Points of Interest (PoIs) as long as the model has learned something

Our Contribution: the Gradient Visualization

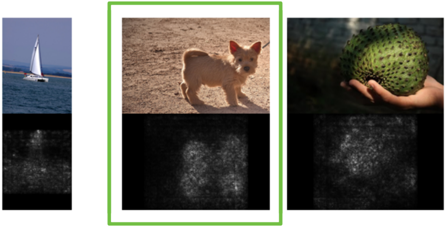
- We propose a characterization technique based on a trained CNN
- Able to detect Points of Interest (PoIs) as long as the model has learned something
- Already proposed in Image Recognition [SVZ13; Spr+14]

Our Contribution: the Gradient Visualization

- We propose a characterization technique based on a trained CNN
- Able to detect Points of Interest (PoIs) as long as the model has learned something
- Already proposed in Image Recognition [SVZ13; Spr+14]
- Starts to be used in SCA [Tim19; HGG19]

Our Contribution: the Gradient Visualization

- We propose a characterization technique based on a trained CNN
- Able to detect Points of Interest (PoIs) as long as the model has learned something
- Already proposed in Image Recognition [SVZ13; Spr+14]
- Starts to be used in SCA [Tim19; HGG19]



Our Contribution: the Gradient Visualization

- We propose a characterization technique based on a trained CNN
- Able to detect Points of Interest (PoIs) as long as the model has learned something
- Already proposed in Image Recognition [SVZ13; Spr+14]
- Starts to be used in SCA [Tim19; HGG19]

Not at the state of the art in Image Recognition. So why such a choice for Side Channel Analysis?

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \to \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example

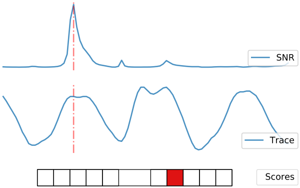
An explanation

- Assume the informative leakage is very localized (few PoIs)

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \rightarrow \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example



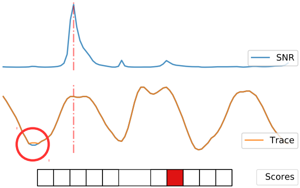
An explanation

- Assume the informative leakage is very localized (few PoIs)
- Consider a new trace and its label $\mathbf{x}, z$

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \to \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example



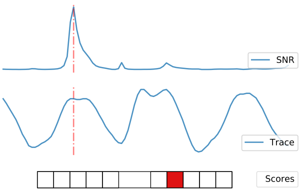
An explanation

- Assume the informative leakage is very localized (few PoIs)
- t_0 non informative: $\mathbf{x}[t_0] \mapsto \mathbf{x}[t_0] + \epsilon$ not sensitive
- In other words, t_0 non informative $\to \frac{\partial}{\partial \mathbf{x}[t_0]} F^*(\mathbf{x})[z] \approx 0$

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \to \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example



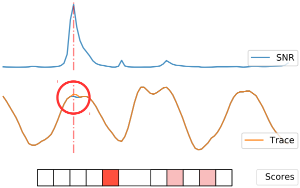
An explanation

- Assume the informative leakage is very localized (few PoIs)
- t_0 non informative: $\mathbf{x}[t_0] \mapsto \mathbf{x}[t_0] + \epsilon$ not sensitive
- In other words, t_0 non informative $\to \frac{\partial}{\partial \mathbf{x}[t_0]} F^*(\mathbf{x})[z] \approx 0$

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \to \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example



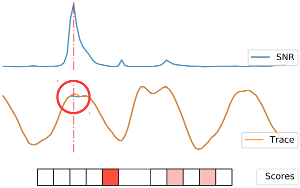
An explanation

- Assume the informative leakage is very localized (few PoIs)
- t_1 informative: $\mathbf{x}[t_1] \mapsto \mathbf{x}[t_1] + \epsilon$ is likely to affect the optimal model's decision
- t_1 informative
  $\to \left| \frac{\partial}{\partial \mathbf{x}[t_1]} F^*(\mathbf{x})[z] \right| > 0$

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \to \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example



An explanation

- Assume the informative leakage is very localized (few PoIs)
- t_1 informative: $\mathbf{x}[t_1] \mapsto \mathbf{x}[t_1] + \epsilon$ is likely to affect the optimal model's decision
- t_1 informative
  $\to \left| \frac{\partial}{\partial \mathbf{x}[t_1]} F^*(\mathbf{x})[z] \right| > 0$

Consequences

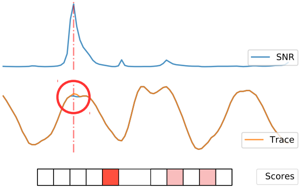
If t is a PoI, then it should be seen in the gradients $\nabla_{\mathbf{x}} F^*(\mathbf{x})[z]$

Q: Why such a choice for Side Channel Analysis?

Let us start with an ideal case

Ideal case: we know $F^* = \Pr[Z|\mathbf{X}]$ (*i.e.* $F^* : \mathbb{R}^D \to \mathcal{P}(\mathcal{Z}) \subset [0,1]^{|\mathcal{Z}|}$)

An example



An explanation

- Assume the informative leakage is very localized (few PoIs)
- t_1 informative: $\mathbf{x}[t_1] \mapsto \mathbf{x}[t_1] + \epsilon$ is likely to affect the optimal model's decision
- t_1 informative $\to \left| \frac{\partial}{\partial \mathbf{x}[t_1]} F^*(\mathbf{x})[z] \right| > 0$

Consequences

If t is a PoI, then it should be seen in the gradients $\nabla_{\mathbf{x}} F^*(\mathbf{x})[z]$

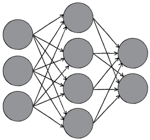
Q: Why such a choice for Side Channel Analysis?

What is the link with Deep learning?

We do not know F^*, but we can replace it with a Deep Neural Net

Deep Neural Networks

Composition of simple operations (a.k.a layers), alternating between linear (λ) and non-linear (σ) layers. Linear layers are parametrized by real values gathered into a vector θ



Theorem (Universal Approximation [HSW90], informal)

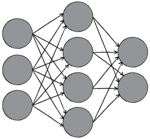
Can we approximate F^* with $F(., \theta^*)$ with an arbitrary uniform precision?

What is the link with Deep learning?

We do not know F^*, but we can replace it with a Deep Neural Net

Deep Neural Networks

Composition of simple operations (a.k.a layers), alternating between linear (λ) and non-linear (σ) layers. Linear layers are parametrized by real values gathered into a vector θ



Theorem (Universal Approximation [HSW90], informal)

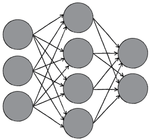
Can we approximate F^* with $F(., \theta^*)$ with an arbitrary uniform precision?
Yes

What is the link with Deep learning?

We do not know F^*, but we can replace it with a Deep Neural Net

Deep Neural Networks

Composition of simple operations (a.k.a layers), alternating between linear (λ) and non-linear (σ) layers. Linear layers are parametrized by real values gathered into a vector θ



Theorem (Universal Approximation [HSW90], informal)

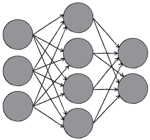
Can we approximate F^* with $F(., \theta^*)$ with an arbitrary uniform precision?
Yes
And what about the derivatives of F^*?

What is the link with Deep learning?

We do not know F^*, but we can replace it with a Deep Neural Net

Deep Neural Networks

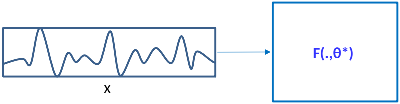
Composition of simple operations (a.k.a layers), alternating between linear (λ) and non-linear (σ) layers. Linear layers are parametrized by real values gathered into a vector θ



Theorem (Universal Approximation [HSW90], informal)

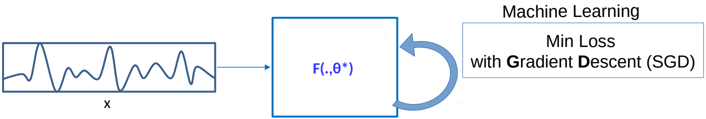
Can we approximate F^* with $F(., \theta^*)$ with an arbitrary uniform precision?
Yes
And what about the derivatives of F^*?
As well!

How to find such an approximator?



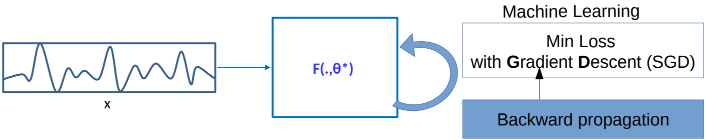
Learning θ...

How to find such an approximator?



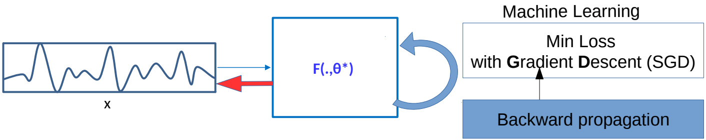
Learning θ...consist in minimizing a loss $\ell(F(\mathbf{x}, \theta^*), z)$ by applying a Gradient Descent.

How to find such an approximator?



Learning θ...consist in minimizing a loss $\ell(F(\mathbf{x}, \theta^*), z)$ by applying a Gradient Descent.
$\nabla_\theta \ell(F(\mathbf{x}, \theta^*), z)$ computed with the backprop algorithm.

How to find such an approximator?

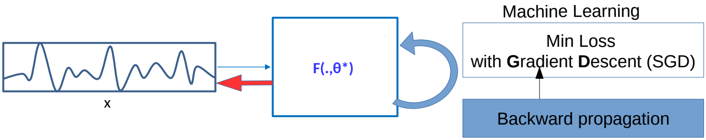


Learning θ...consist in minimizing a loss $\ell(F(\mathbf{x}, \theta^*), z)$ by applying a Gradient Descent.

$\nabla_\theta \ell(F(\mathbf{x}, \theta^*), z)$ computed with the backprop algorithm.

Side effect: $\nabla_\mathbf{x} \ell(F(\mathbf{x}, \theta^*), z)$ is also computed for free !

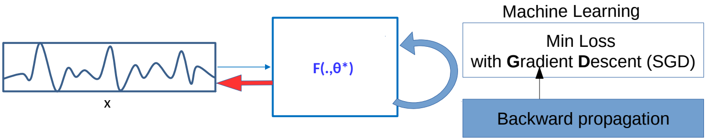
How to find such an approximator?



Learning θ...consist in minimizing a loss $\ell(F(\mathbf{x}, \theta^*), z)$ by applying a Gradient Descent.
$\nabla_\theta \ell(F(\mathbf{x}, \theta^*), z)$ computed with the backprop algorithm.
Side effect: $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta^*), z)$ is also computed for free !

Q: Wait a minute: is that really what we want?
We got $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta^*), z)$, we wanted $\nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[z]$.

How to find such an approximator?



Learning θ...consist in minimizing a loss $\ell(F(\mathbf{x}, \theta^*), z)$ by applying a Gradient Descent.
$\nabla_\theta \ell(F(\mathbf{x}, \theta^*), z)$ computed with the backprop algorithm.
Side effect: $\nabla_\mathbf{x} \ell(F(\mathbf{x}, \theta^*), z)$ is also computed for free !

Q: Wait a minute: is that really what we want?
We got $\nabla_\mathbf{x} \ell(F(\mathbf{x}, \theta^*), z)$, we wanted $\nabla_\mathbf{x} F(\mathbf{x}, \theta^*)[z]$.
A: Yes !
Both are equivalent.

Concretely, how to implement this method?

Very straightforward in Pytorch [Noa]:

```
def gradient_viz(model, x, z):
  """
  Generates the gradient visualization from a trace or
  a batch of traces.
  """
  # Enables x to store its gradient during the backprop
  x.requires_grad_()

  # Forward pass
  probas = model(x)
  loss = lossFunc(probas, z)

  # Gradient initialization
  model.zero_grad()

  # Backward pass
  loss.backward()

  # Post-processing of the gradient
  gradients = x.grad.abs()
  return gradients
```

With Tensorflow: tf.abs(tf.gradients(probas[:,Z], X)).

Outline

Application on experimental data

Description

ASCAD dataset [Pro+18]: $50,000$ traces, each of 700 points
Corresponds to the first AES round
Three cases studied:

1. **No countermeasure**: synchronized traces, no masking
2. **Artificial random shift**
3. **Synchronized traces, boolean masking (unknown masks)**

Trained model

CNN with a VGG-like architecture
Grid search of hyperparameters
Best model: minimal trace number when the guessing entropy reaches 2

First experiment: no countermeasure

Average number of traces to recover the secret key: 3

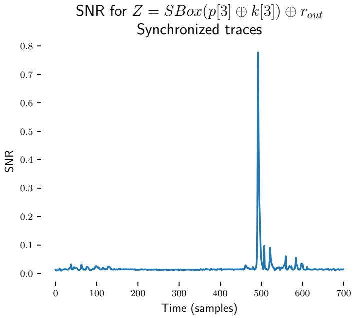


Figure: SNR

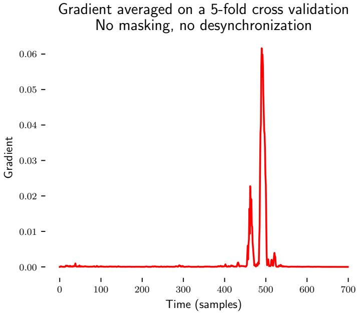


Figure: Gradient Visualization

Second experiment: with desynchronization

Average number of traces to recover the secret key: 3.6

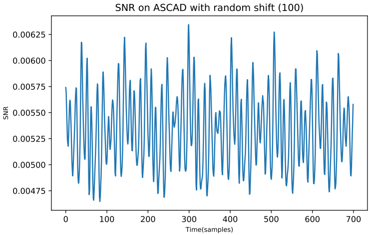


Figure: No PoI emphasized ☹

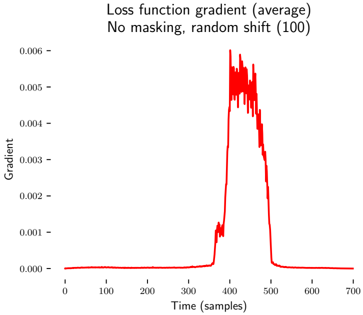


Figure: Band of peaks ☺

Second experiment: with desynchronization

Average number of traces to recover the secret key: 3.6

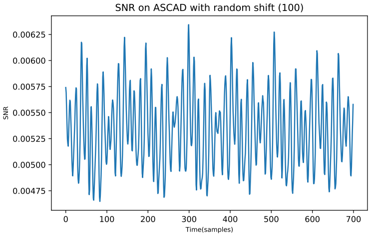


Figure: No PoI emphasized ☹

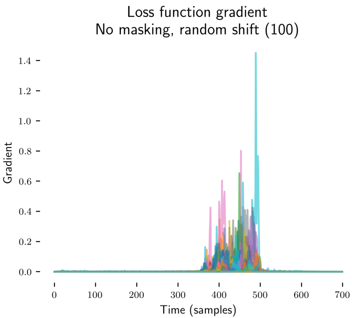


Figure: Characterization for each trace ☺

Third experiment: with masking

Average number of traces to recover the secret key: ≈ 100

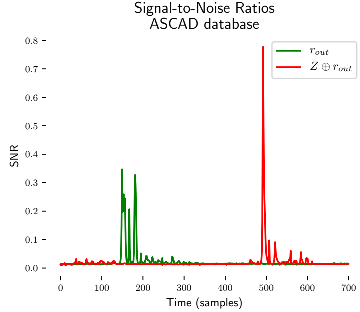


Figure: Requires knowledge of the masks ☹

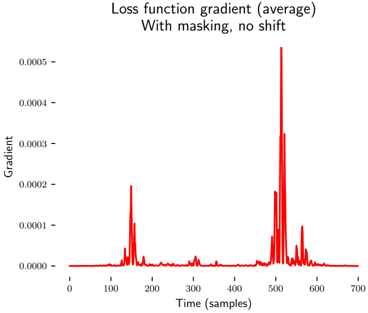


Figure: No kowledge required ☺

Be careful not to overfit !

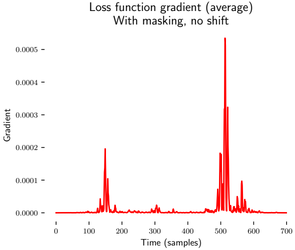


Figure: GV without overfitting 🙂

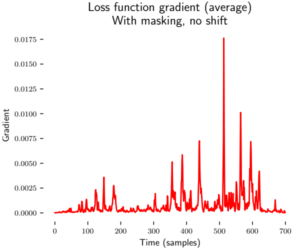


Figure: GV with overfitting 🙁

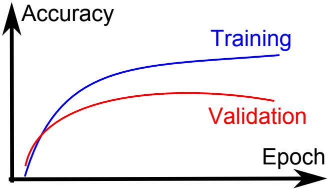


Figure: Solution: early-stopping

Conclusions

- We have proposed a new characterization method, simple but promising

Conclusions

- We have proposed a new characterization method, simple but promising
- Current research topic in characterization

Conclusions

- We have proposed a new characterization method, simple but promising
- Current research topic in characterization
- Should lead to better understanding the vulnerabilities $\Longrightarrow$ help developers to improve their products

Conclusions

- We have proposed a new characterization method, simple but promising
- Current research topic in characterization
- Should lead to better understanding the vulnerabilities $\Longrightarrow$ help developers to improve their products

Thank You!

Questions?

References I

[HGG19] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. *Deep Neural Network Attribution Methods for Leakage Analysis and Symmetric Key Recovery*. 143. 2019. URL: https://eprint.iacr.org/2019/143 (visited on 02/21/2019).

[HSW90] K. Hornik, M. Stinchcombe, and H. White. "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks". In: *Neural Networks* 3.5 (1990), pp. 551–560. ISSN: 0893-6080. DOI: 10.1016/0893-6080(90)90005-6.

[Pro+18] Emmanuel Prouff et al. *Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database*. 053. 2018. URL: http://eprint.iacr.org/2018/053 (visited on 01/19/2018).

[Noa] *PyTorch*. URL: https://www.pytorch.org (visited on 11/14/2018).

References II

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: *arXiv:1312.6034 [cs]* (Dec. 20, 2013). arXiv: 1312.6034. URL: http://arxiv.org/abs/1312.6034 (visited on 09/07/2018).

[Spr+14] Jost Tobias Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: *arXiv:1412.6806 [cs]* (Dec. 21, 2014). arXiv: 1412.6806. URL: http://arxiv.org/abs/1412.6806 (visited on 09/07/2018).

[Tim19] Benjamin Timon. "Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (Feb. 28, 2019), pp. 107–131. ISSN: 2569-2925. DOI: 10.13154/tches.v2019.i2.107-131. URL: https://tches.iacr.org/index.php/TCHES/article/view/7387 (visited on 03/25/2019).

Analysis of overfitting

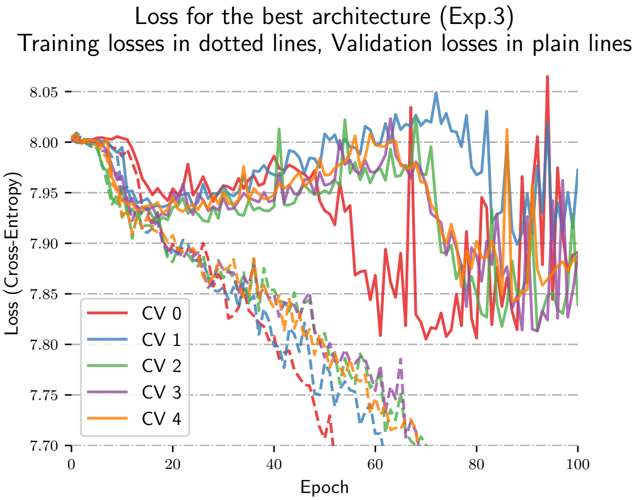


Figure: The loss during training.

Illustration on simulated data

Description

Simulation on $n = 4$ bits.
One or several shares that leak in a Hamming weights model with white Gaussian noise, mixed with fool points (same marginal pdf).
Training with a *small* Multi-Layer Perceptron with *exhaustive* data to guess the xor of the shares.

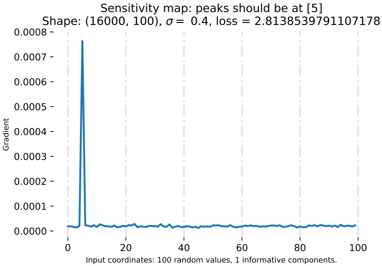


Figure: Average Gradient of the loss function w.r.t. the simulated traces.

Illustration on simulated data

Description

Simulation on $n = 4$ bits.
One or several shares that leak in a Hamming weights model with white Gaussian noise, mixed with fool points (same marginal pdf).
Training with a *small* Multi-Layer Perceptron with *exhaustive* data to guess the xor of the shares.

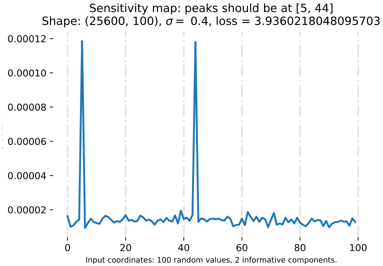


Figure: Average Gradient of the loss function w.r.t. the simulated traces.

Illustration on simulated data

Description

Simulation on $n = 4$ bits.
One or several shares that leak in a Hamming weights model with white Gaussian noise, mixed with fool points (same marginal pdf).
Training with a *small* Multi-Layer Perceptron with *exhaustive* data to guess the xor of the shares.

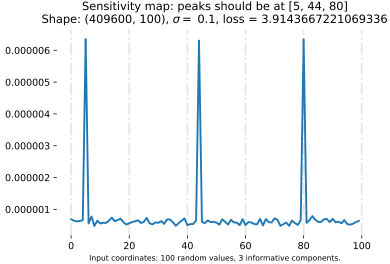


Figure: Average Gradient of the loss function w.r.t. the simulated traces.

Wait, this is not exactly what we were looking for !

We wanted $\nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[z]$ but we got $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta^*), z)$.

1. **What is the link between the two terms?**

Wait, this is not exactly what we were looking for !

We wanted $\nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[z]$ but we got $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta^*), z)$.

1. **What is the link between the two terms?**
   The loss gradient can be computed from the Jacobian matrix with the chain rule for derivatives:

$$\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta), z) = J_F(\mathbf{x}, \theta)^T \nabla_{\mathbf{y}} \ell(F(\mathbf{x}, \theta), z). \tag{1}$$

2. **Why not giving the Jacobian matrix directly?**

Wait, this is not exactly what we were looking for !

We wanted $\nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[z]$ but we got $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta^*), z)$.

1. **What is the link between the two terms?**
   The loss gradient can be computed from the Jacobian matrix with the chain rule for derivatives:

$$\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta), z) = J_F(\mathbf{x}, \theta)^T \nabla_{\mathbf{y}} \ell(F(\mathbf{x}, \theta), z). \quad (1)$$

2. **Why not giving the Jacobian matrix directly?**
   Surprisingly, the Deep Learning frameworks compute the loss gradient more efficiently. The Jacobian is not even explicitly computed !
3. **Should we be concerned about that?**

Wait, this is not exactly what we were looking for !

We wanted $\nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[z]$ but we got $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta^*), z)$.

1. **What is the link between the two terms?**
   The loss gradient can be computed from the Jacobian matrix with the chain rule for derivatives:
   $$\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta), z) = J_F(\mathbf{x}, \theta)^T \nabla_{\mathbf{y}} \ell(F(\mathbf{x}, \theta), z). \quad (1)$$
2. **Why not giving the Jacobian matrix directly?**
   Surprisingly, the Deep Learning frameworks compute the loss gradient more efficiently. The Jacobian is not even explicitly computed !
3. **Should we be concerned about that?**
   No. Remind that $J_F(\mathbf{x}, \theta)$ is made with the $\nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[s], s \in \mathcal{Z}$. Furthermore, $\nabla_{\mathbf{y}} \ell(F(\mathbf{x}, \theta), z)$ is actually proportional to the one-hot vector encoding z. It follows that $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta), z) \propto \nabla_{\mathbf{x}} F(\mathbf{x}, \theta^*)[z]$.

Remark: It is still possible to get the Jacobian matrix.

The Jacobian matrix in practise

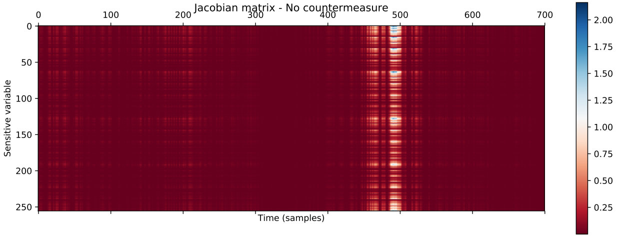


Figure: The Jacobian matrix in Experiment 1