FROM RESEARCH TO INDUSTRY

Feature Extraction for Side-Channel Attacks

Eleonora Cagli

05/12/2018, Paris

CESTI - Centre d'Évaluation de la Sécurité des Technologies de l'Information - CEA Grenoble

LIP6 - Laboratoire d'Informatique de Paris 6

PhD Supervisor : Emmanuel Prouff (ANSSI)
CEA Supervisor : Cécile Dumas (CESTI - CEA Grenoble)

Contents

Secure Component and Embedded Cryptography

Secure Component and Embedded Cryptography

- Sensitive applications
- Pervasive aspect
- Hostile environment

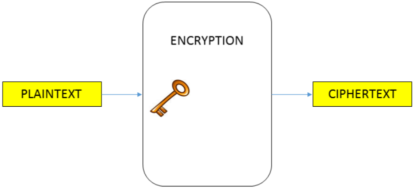
$\Rightarrow$ Requires protection against very high-level attacker

Secure Component and Embedded Cryptography

- Sensitive applications
- Pervasive aspect
- Hostile environment

$\Rightarrow$ Requires protection against very high-level attacker

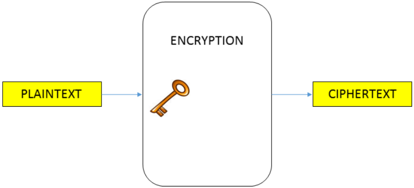
Side-Channel Vulnerability of Embedded Cryptography



Attack $\Longrightarrow$ a secret

	Classical Attacks	Side-Channel Attacks
	Mathematical vulnerability	
	Black Box	

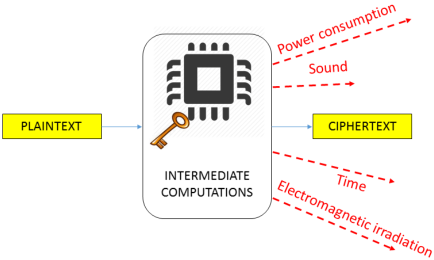
Side-Channel Vulnerability of Embedded Cryptography



Attack $\Longrightarrow$ 

Classical Attacks	Side-Channel Attacks
Mathematical vulnerability	
Black Box	

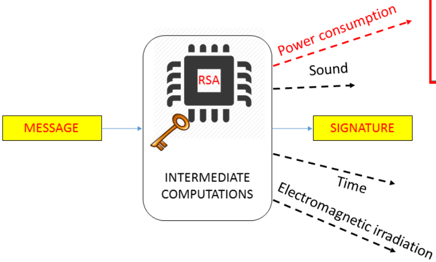
Side-Channel Vulnerability of Embedded Cryptography

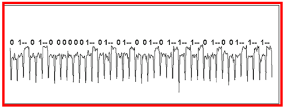


Attack $\Longrightarrow$ 

Classical Attacks	Side-Channel Attacks
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

Side-Channel Vulnerability of Embedded Cryptography



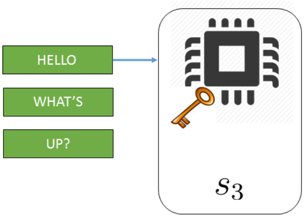

Source - Paul Kocher et al. "Introduction to differential power analysis".

Simple attack:
a single input

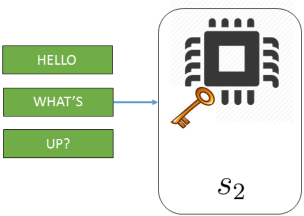
Attack $\Longrightarrow$ 

Classical Attacks	Side-Channel Attacks
Mathematical vulnerability	Physical vulnerability
Black Box	Grey Box / Divide-and-conquer

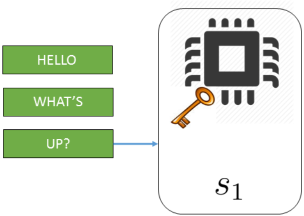
Advanced Side-Channel Attacks



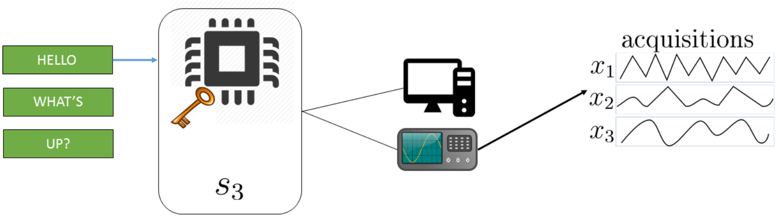
Advanced Side-Channel Attacks



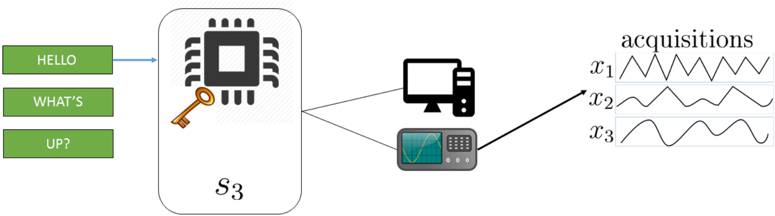
Advanced Side-Channel Attacks

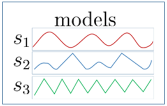


Advanced Side-Channel Attacks

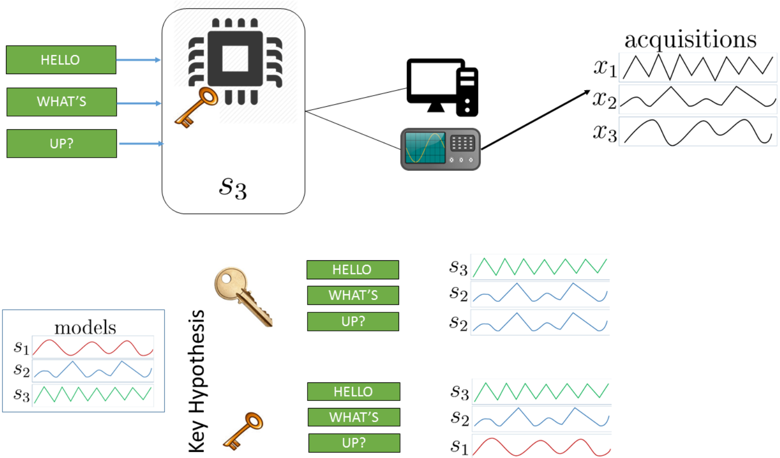


Advanced Side-Channel Attacks

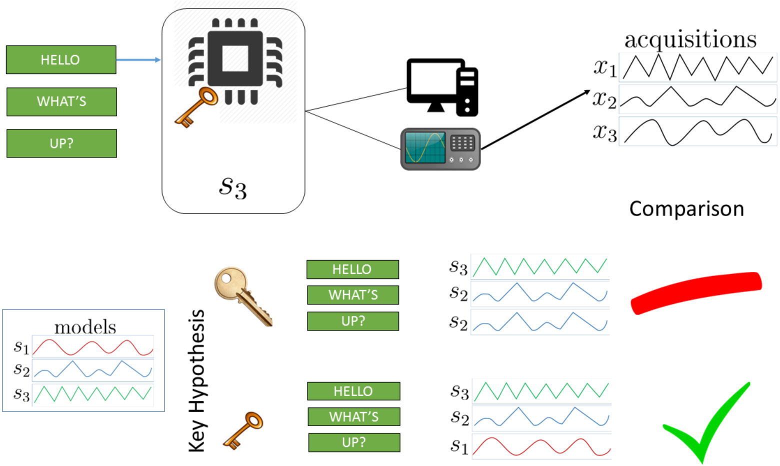




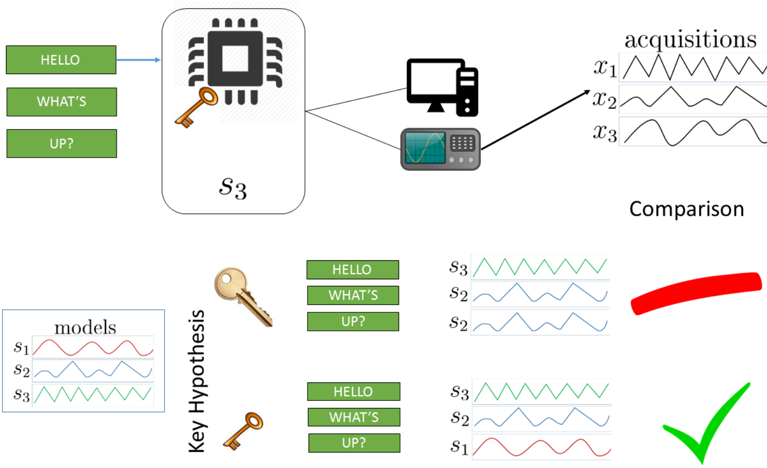
Advanced Side-Channel Attacks



Advanced Side-Channel Attacks

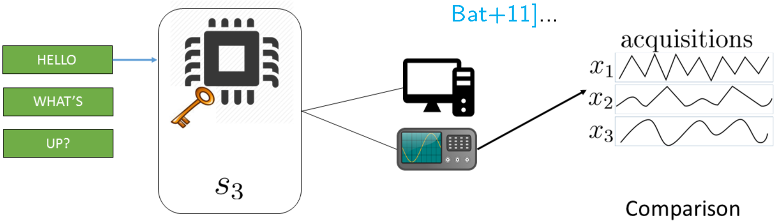


Advanced Side-Channel Attacks

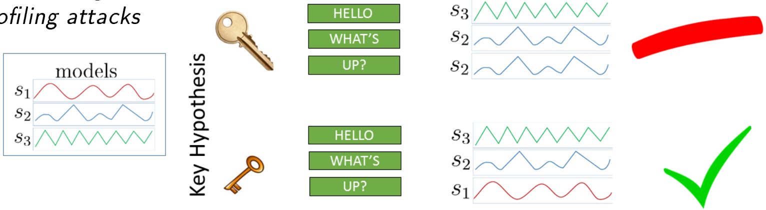


Advanced Side-Channel Attacks

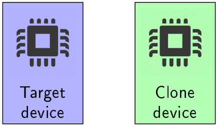
Differential Power Analysis [KJJ99]
Correlation Power Analysis [BCO04]
Mutual Information Analysis [Gie+08; Bat+11]...



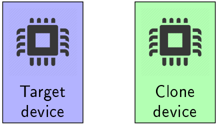
Non-profiling attacks
Profiling attacks



Profiling Attacks...Supervised Learning



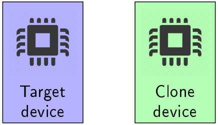
Profiling Attacks...Supervised Learning



Machine Learning

Supervised Learning

Profiling Attacks...Supervised Learning

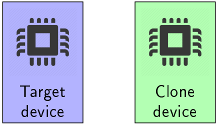


Machine Learning

Learn from data via statistic models
Task - Performance - Experience [TM97]

Supervised Learning

Profiling Attacks...Supervised Learning



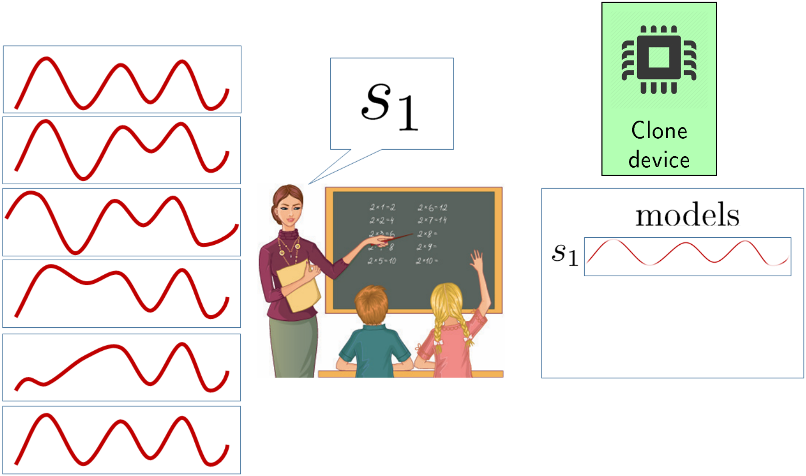
Machine Learning

Learn from data via statistic models
Task - Performance - Experience [TM97]

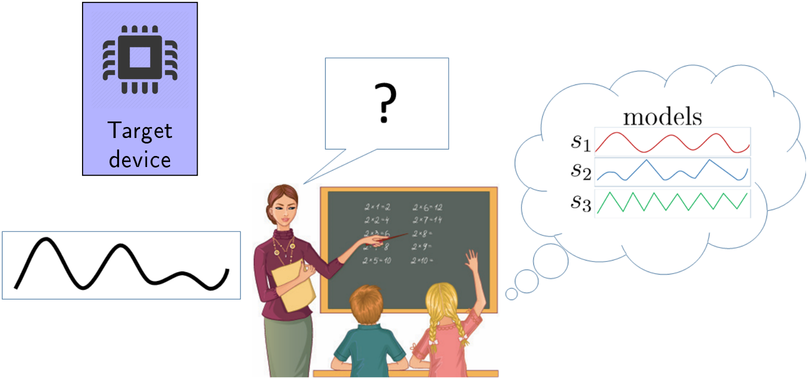
Supervised Learning

The *supervised* learning algorithms access to a dataset of examples, each associated in general to a *target* or *label*.

Classroom Side-Channel Attacks



Classroom Side-Channel Attacks



Classification

Classification problem

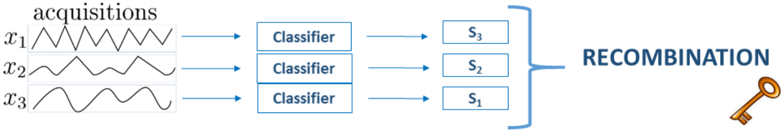
Assign to a datum $\vec{X}$ a label Z among a set of possible labels $\mathcal{Z} = \{s_1, s_2, s_3\}$, or probabilities.

Classification

Classification problem

Assign to a datum $\vec{X}$ a label Z among a set of possible labels $\mathcal{Z} = \{s_1, s_2, s_3\}$, or probabilities.

Advanced Attack as Multiple Classification Problems



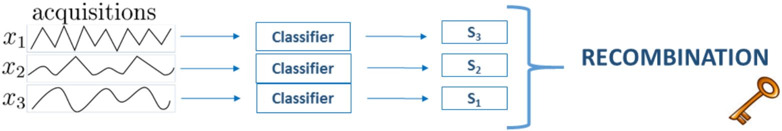
Classification

Machine Learning classifiers in Side-Channel literature: SVM ([Hos+11; HZ12]), RF ([LBM14; LBM15])

Classification problem

Assign to a datum $\vec{X}$ a label Z among a set of possible labels $\mathcal{Z} = \{s_1, s_2, s_3\}$, or probabilities.

Advanced Attack as Multiple Classification Problems



Contents

Notations

Notations and generalities

- Side-channel traces: realizations of a random vector $\vec{X} \in \mathbb{R}^D$
- D is the number of time samples (or features)
- Target: a *sensitive* variable $Z = f(\mathrm{e}, \mathrm{k})$ in $\mathcal{Z} = \{s_1, \dots, s_{|\mathcal{Z}|}\}$

Profiling attack scenario

- labelled traces $\mathcal{D}_{\text{train}} = (\vec{x}_i, e_i, k_i)_{i=1}^{N}$, acquired under known secrets
- attack traces $\mathcal{D}_{\text{attack}} = (\vec{x}_i, e_i)_{i=1}^{N_a}$ acquired under unknown secrets

Profiling Attack

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\vec{X} \mid Z}\left((\vec{x}_i)_{i=1,\dots,N_a}, (f(e_i, k))_{i=1,\dots,N_a} \right)$$

Profiling Attack

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$ $p_{\vec{X}}$ p_Z (generative model)
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\vec{X} \mid Z}\left((\vec{x}_i)_{i=1,\dots,N_a}, (f(e_i,k))_{i=1,\dots,N_a} \right)$$

- A-posteriori probability score for each key hypothesis k

$$d_k = p_{Z \mid \vec{X}}\left(f(e_i,k)_{i=1,\dots,N_a}, (\vec{x}_i)_{i=1,\dots,N_a} \right) \quad ,$$

Profiling Attack

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$ $p_{\vec{X}}$ p_Z (generative model)
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\vec{X} \mid Z}\left((\vec{x}_i)_{i=1,\dots,N_a}, (f(e_i,k))_{i=1,\dots,N_a} \right)$$

- A-posteriori probability score for each key hypothesis k

$$d_k = p_{Z \mid \vec{X}}\left(f(e_i,k)_{i=1,\dots,N_a}, (\vec{x}_i)_{i=1,\dots,N_a} \right) \quad ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$ $p_{\vec{X}}$ p_Z (generative model)
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\vec{X} \mid Z}\left((\vec{x}_i)_{i=1,\dots,N_a}, (f(e_i,k))_{i=1,\dots,N_a} \right)$$

- A-posteriori probability score for each key hypothesis k

$$d_k = p_{Z \mid \vec{X}}\left(f\,(e_i,k)_{i=1,\dots,N_a}, (\vec{x}_i)_{i=1,\dots,N_a} \right) \quad ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- estimate
  - $p_{\vec{X} \mid Z=z}$ $p_{\vec{X}}$ p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $p_{Z \mid \vec{X}=\vec{x}}$ (discriminative model)

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\vec{X} \mid Z}\left((\vec{x}_i)_{i=1,\dots,N_a}, (f(e_i,k))_{i=1,\dots,N_a} \right)$$

- A-posteriori probability score for each key hypothesis k

$$d_k = p_{Z \mid \vec{X}}\left(f(e_i,k)_{i=1,\dots,N_a}, (\vec{x}_i)_{i=1,\dots,N_a} \right) ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- mandatory dimensionality reduction $[\mathcal{D}_{\mathbf{train}} \longrightarrow \epsilon \colon \mathbb{R}^D \rightarrow \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$ $p_{\epsilon(\vec{X})}$ p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $p_{Z \mid \epsilon(\vec{X})=\epsilon(\vec{x})}$ (discriminative model)

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\epsilon(\vec{X}) \mid Z}\left((\epsilon(\vec{x}_i))_{i=1,\dots,N_a}, (f(e_i,k))_{i=1,\dots,N_a} \right)$$

- A-posteriori probability score for each key hypothesis k

$$d_k = p_{Z \mid \epsilon(\vec{X})}\left(f\,(e_i,k)_{i=1,\dots,N_a}, (\epsilon(\vec{x}_i))_{i=1,\dots,N_a} \right) \quad ,$$

Profiling Attack

$\vec{X} \in \mathbb{R}^D$

Curse of dimensionality!

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho \colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon \colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\rho(\vec{X}))\mid Z=z}\ p_{\epsilon(\rho(\vec{X}))}\ p_Z$ (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $p_{Z\mid \rho(\epsilon(\vec{X}))=\epsilon(\rho(\vec{x}))}$ (discriminative model)

Attack phase

- Likelihood score for each key hypothesis k

$$d_k = p_{\epsilon(\rho(\vec{X}))\mid Z}\left((\epsilon(\rho(\vec{x}_i)))_{i=1,\dots,N_a}, (f(e_i,k))_{i=1,\dots,N_a} \right)$$

- A-posteriori probability score for each key hypothesis k

$$d_k = p_{Z\mid \epsilon(\rho(\vec{X}))}\left(f\,(e_i,k)_{i=1,\dots,N_a}, (\epsilon(\rho(\vec{x}_i)))_{i=1,\dots,N_a} \right) ,$$

Objectives

Profiling phase

- manage de-synchronization problem [$\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D$]
- mandatory dimensionality reduction [$\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C$]
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$ $p_{\epsilon(\vec{X})}$ p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

Objectives

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$ $p_{\epsilon(\vec{X})}$ p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

- Dimensionality reduction techniques for Template Attack

Objectives

Profiling phase

- manage de-synchronization problem [$\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D$]
- mandatory dimensionality reduction [$\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C$]
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}\; p_{\epsilon(\vec{X})}\; p_Z$ (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $Z \mid \epsilon(\vec{X}) = \epsilon(\vec{x})$ (discriminative model)

Objectives

- Dimensionality reduction techniques for Template Attack
- Consider countermeasures (masking, hiding)

Objectives

Profiling phase

- manage de-synchronization problem [$\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D$]
- mandatory dimensionality reduction [$\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C$]
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}\ p_{\epsilon(\vec{X})}\ p_Z$ (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $Z \mid \vec{X} = \vec{x}$ (discriminative model)

Objectives

- Dimensionality reduction techniques for Template Attack
- Consider countermeasures (masking, hiding)
- More generally, ameliorate the profiling attack strategy

Objectives

Profiling phase

- manage de-synchronization problem **DEEP LEARNING** $[\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\vec{X}) \mid Z=z}$, $p_{\epsilon(\vec{X})}$ p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**) [CRR03]
  - $Z \mid \vec{X} = \vec{x}$ (discriminative model)

Objectives

- Dimensionality reduction techniques for Template Attack
- Consider countermeasures (masking, hiding)
- More generally, ameliorate the profiling attack strategy

Dimensionality Reduction: State of the Art

Dimensionality Reduction

$$\epsilon \colon \mathbb{R}^D \to \mathbb{R}^C$$
$$\vec{x} \mapsto \epsilon(\vec{x})$$

- Feature selection (Points of Interest selection)
- Feature extraction

Dimensionality Reduction: State of the Art

Dimensionality Reduction

$$\epsilon : \mathbb{R}^D \rightarrow \mathbb{R}^C$$
$$\vec{x} \mapsto \epsilon(\vec{x})$$

- Feature selection (Points of Interest selection)
- Feature extraction

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

Dimensionality Reduction: State of the Art

Dimensionality Reduction

$$\epsilon \colon \mathbb{R}^D \to \mathbb{R}^C$$
$$\vec{x} \mapsto \epsilon(\vec{x})$$

- Feature selection (Points of Interest selection)
- Feature extraction

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

Dimensionality Reduction: State of the Art

Dimensionality Reduction

$$\epsilon\colon \mathbb{R}^D \to \mathbb{R}^C$$
$$\vec{x} \mapsto \epsilon(\vec{x})$$

- Feature selection (Points of Interest selection)
- Feature extraction

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

Contributions

- **Linear Dimensionality Reduction** ([CARDIS 2015]):
  - PCA, choise of components ELV
  - LDA in case of undersampling
- **Kernel Discriminant Analysis** ([CARDIS 2016]): application of an appropriate kernel trick to LDA, in order to manage masking countermeasure
- **Convolutional Neural Networks** ([CHES 2017]):
  - discriminative model by means of neural network classifiers
  - convolutional layers to manage desynchronisation (a form of hiding)
  - Data Augmentation techniques to reduce overfitting
- **ASCAD public dataset** (submitted paper):
  - deep learning open comparison platform (implementation sources, side-channel traces, attack scripts)
  - hyper-parametrization methodology proposal and test

Contributions

- **Linear Dimensionality Reduction** ([CARDIS 2015]):
  - PCA, choise of components ELV
  - LDA in case of undersampling
- **Kernel Discriminant Analysis** ([CARDIS 2016]): application of an appropriate kernel trick to LDA, in order to manage masking countermeasure
- **Convolutional Neural Networks** ([CHES 2017]):
  - discriminative model by means of neural network classifiers
  - convolutional layers to manage desyncrhonisation (a form of hiding)
  - Data Augmentation techniques to reduce overfitting
- **ASCAD public dataset** (submitted paper):
  - deep learning open comparison platform (implementation sources, side-channel traces, attack scripts)
  - hyper-parametrization methodology proposal and test

Contents

Dimensionality reduction in presence of masking

$(d-1)$th-order Sharing (or Masking)

Split each sensitive Z into shares M_1, ..., M_d

- Random *masks*: $M_1, \ldots, M_{d-1}$
- *Masked* variable: $M_d = Z \oplus M_1^{-1} \oplus \cdots \oplus M_{d-1}$

Shares are handled at time samples

$$t_1, \ldots, t_d \qquad \text{(in general different if software countermeasure)}$$

Indistinguishability of $p_{\vec{X} \mid Z=z}$ up to order $d-1$

Dimensionality reduction in presence of masking

$(d-1)$th-order Sharing (or Masking)

Split each sensitive Z into shares $M_1, \ldots, M_d$

- Random *masks*: $M_1, \ldots, M_{d-1}$
- *Masked* variable: $M_d = Z \oplus M_1^{-1} \oplus \cdots \oplus M_{d-1}$

Shares are handled at time samples

$t_1, \ldots, t_d$ (in general different if software countermeasure)

Indistinguishability of $p_{\vec{X} \mid Z=z}$ up to order $d-1$

$f(z) = \mathbb{E}\left[\vec{X}[t_1]\vec{X}[t_2]\ldots\vec{X}[t_d] | Z = z\right]$ non-constant $\Rightarrow$ dth-order attack

Dimensionality reduction in presence of masking

$(d-1)$th-order Sharing (or Masking)

Split each sensitive Z into shares $M_1, \dots, M_d$

- Random *masks*: $M_1, \dots, M_{d-1}$
- *Masked* variable: $M_d = Z \oplus M_1^{-1} \oplus \dots \oplus M_{d-1}$

Shares are handled at time samples

$$t_1, \dots, t_d \qquad \text{(in general different if software countermeasure)}$$

Indistinguishability of $p_{\vec{X} \mid Z=z}$ up to order $d-1$

$f(z) = \mathbb{E}\left[\vec{X}[t_1]\vec{X}[t_2]\dots\vec{X}[t_d] | Z = z\right]$ non-constant $\Rightarrow$ dth-order attack

$\Rightarrow$ extract features containing $\vec{X}[t_1]\vec{X}[t_2]\dots\vec{X}[t_d]$ **(Necessary condition)**

How to detect the d-tuple $t_1, \ldots, t_d$?

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

- Point-wise statistics
- Exploit $\mathbb{E}[\vec{X}|Z = z]$

- Combine all time samples ✓
- Linear combinations $\mathbb{E}[A\vec{X}|Z = z]$

How to detect the d-tuple $t_1, \dots, t_d$?

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

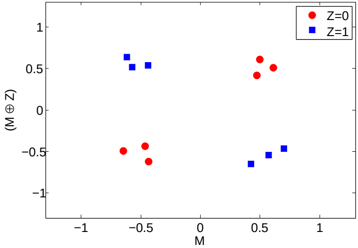
Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

- Point-wise statistics
- Exploit $\mathbb{E}[\vec{X}|Z = z]$

- Combine all time samples
- Linear combinations $\mathbb{E}[A\vec{X}|Z = z]$



Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

How to detect the d-tuple $t_1, \ldots, t_d$?

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

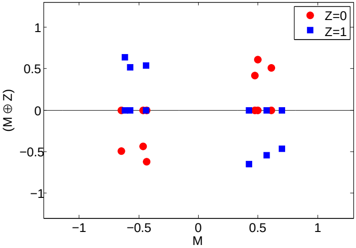
Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

- Point-wise statistics
- Exploit $\mathbb{E}[\vec{X}|Z = z]$

- Combine all time samples
- Linear combinations $\mathbb{E}[A\vec{X}|Z = z]$



Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

How to detect the d-tuple $t_1, \ldots, t_d$?

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

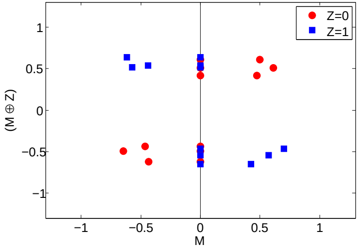
Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

- Point-wise statistics
- Exploit $\mathbb{E}[\vec{X}|Z = z]$

- Combine all time samples
- Linear combinations $\mathbb{E}[A\vec{X}|Z = z]$



Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

How to detect the d-tuple $t_1, \dots, t_d$?

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

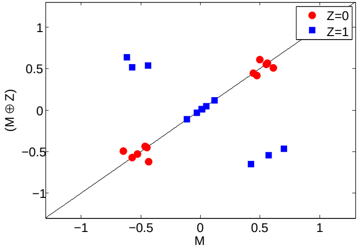
Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

- Point-wise statistics −
- Exploit $\mathbb{E}[\vec{X}|Z = z]$ −

- Combine all time samples ✓
- Linear combinations $\mathbb{E}[A\vec{X}|Z = z]$ −



Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

How to detect the d-tuple $t_1, \ldots, t_d$?

Feature selection

ϵ performs a sub-sampling

- SOD [CRR03]
- SOST [BDP10]
- SNR [MOP08]/ NICV [Bha+14]
- t-test, F-test,... [GLRP06; CK14]

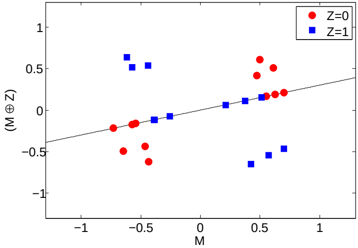
Linear feature extraction

ϵ performs linear combinations
$\epsilon(\vec{x}) = A\vec{x}$ with $A \in M_{\mathbb{R}}(C, D)$

- Principal Component Analysis (PCA) [Arc+06; BHW12]
- Linear Discriminant Analysis (LDA) [SA08; Bru+15]
- Projection Pursuits (PP) [Dur+15]

- Point-wise statistics
- Exploit $\mathbb{E}[\vec{X}|Z = z]$

- Combine all time samples
- Linear combinations $\mathbb{E}[A\vec{X}|Z = z]$



Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

PoIs Research

A lacking literature

- many HO attacks papers assume the knowledge of $t_1, \dots, t_d$
- PoI research exploiting the masks knowledge in profiling phase
- Hand selection via educated guess [Osw+06]
- Feature Selection for Higher-Order Attacks $\longrightarrow$ Projection Pursuits [Dur+15]

Kernel Discriminant Analysis starting point

Naive strategy: infer over all possible d-tuples

KDA: the purpose

Problem

All dth-degree monomials in the trace coordinates lie in:

$$\mathcal{F} = \mathbb{R}^{\binom{D+d-1}{d}} \qquad \textit{feature space}$$

⚠ Dimension increasing combinatorially with d and D

$$\begin{tikzcd}
\mathbb{R}^D \arrow[r, "\Phi"] \arrow[rr, bend left, "\epsilon^{\mathrm{KPCA}}"] \arrow[rr, bend right, "\epsilon^{\mathrm{KDA}}"'] & \mathcal{F} \arrow[r, shift left, "\epsilon^{\mathrm{PCA}}"] \arrow[r, shift right, "\epsilon^{\mathrm{LDA}}"'] & \mathbb{R}^C
\end{tikzcd}$$

Φ non-linear function

KDA

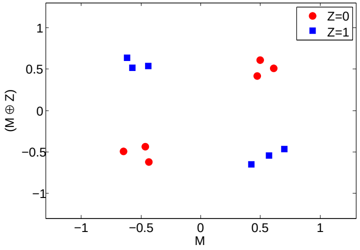
KDA allows performing LDA in $\mathcal{F}$, remaining in $\mathbb{R}^D$.

KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)

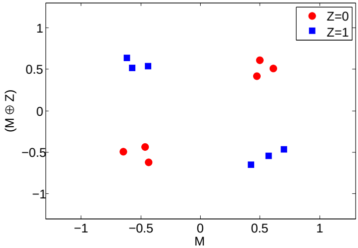


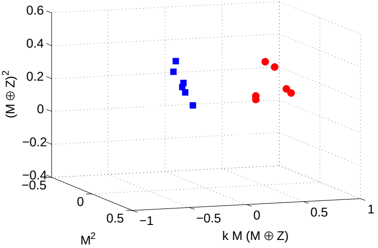
KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)





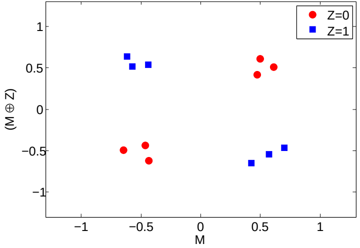
$$\Phi : \mathbb{R}^D \to \mathbb{R}^{\binom{D+d-1}{d}}$$
$$\Phi(t_1, t2) = (t_1^2, t_2^2, k t_1 t_2)$$

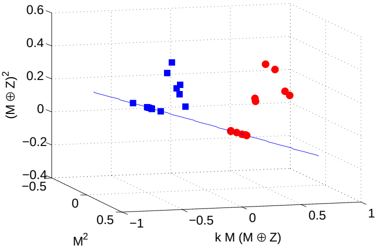
KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)





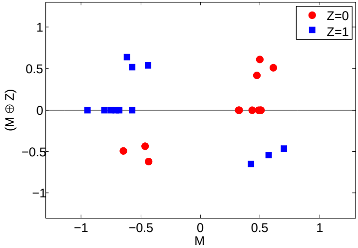
$\Phi \longrightarrow$ LDA

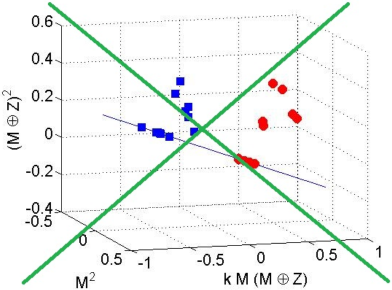
KDA: an intuition

Toy example: 2 time samples, 1-bit data

t_1: $M + n$, $n \sim \mathcal{N}(0, 0.1)$

t_2: $M \oplus Z + n$ (Boolean masking)





KDA
remains in $\mathbb{R}^D$

Kernel Function

$$\begin{tikzcd}[column sep=large]
\mathbb{R}^D \arrow[r, "\Phi"] \arrow[rr, bend left=50, "\epsilon^{\mathrm{KPCA}}"] \arrow[rr, bend right=50, "\epsilon^{\mathrm{KDA}}"'] & \mathcal{F} \arrow[r, shift left, "\epsilon^{\mathrm{PCA}}"] \arrow[r, shift right, "\epsilon^{\mathrm{LDA}}"'] & \mathbb{R}^C
\end{tikzcd}$$

Φ non-linear function

Kernel Function

$K : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \tag{1}$$

dth-degree Polynomial Kernel Function

$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d \quad \leftrightarrow \quad \Phi : \mathbb{R}^D \rightarrow \mathcal{F} \subset \mathbb{R}^{\binom{D+d-1}{d}}$ all dth-degree monomials

KDA - the training

"Fisher Discriminant Analysis with Kernels"([SM99])

Between-class (inter-class) Covariance Matrix

LDA

- $\mathbf{S}_\mathrm{B} = \sum_{z \in \mathcal{Z}} N_z (\vec{\mu}_s - \overline{\vec{x}})(\vec{\mu}_s - \overline{\vec{x}})^\intercal$

KDA

- $\mathbf{M} = \sum_{z \in \mathcal{Z}} N_z (\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
-

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by:
$\vec{M}_z[j] = \frac{1}{N_z} \sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\quad \vec{M}_T[j] = \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

"Fisher Discriminant Analysis with Kernels"([SM99])

Within-class (intra-class) Covariance Matrix

LDA

- $\mathbf{S_B} = \sum_{z\in\mathcal{Z}} N_z(\vec{\mu}_s - \vec{\overline{x}})(\vec{\mu}_s - \vec{\overline{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z\in\mathcal{Z}} \sum_{i=1}^{N_z}(\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$

KDA

- $\mathbf{M} = \sum_{z\in\mathcal{Z}} N_z(\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z\in\mathcal{Z}} \mathbf{K}_z(\mathbf{I} - \mathbf{I}_{N_z})\mathbf{K}_z^\intercal$ [2]
-

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by:
$\vec{M}_z[j] = \frac{1}{N_z}\sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\quad \vec{M}_T[j] = \frac{1}{N}\sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

"Fisher Discriminant Analysis with Kernels"([SM99])

Eigenvector problem

Computational Complexity $O(D^3)$

LDA

- $\mathbf{S_B} = \sum_{z\in\mathcal{Z}} N_z(\vec{\mu}_s - \vec{\overline{x}})(\vec{\mu}_s - \vec{\overline{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z\in\mathcal{Z}} \sum_{i=1}^{N_z}(\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$
- $\vec{\alpha}_i$ eigenvectors of $\mathbf{S_W}^{-1}\mathbf{S_B}$ $[D \times D]$

Computational Complexity $O(N^3)$

KDA

- $\mathbf{M} = \sum_{z\in\mathcal{Z}} N_z(\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z\in\mathcal{Z}} \mathbf{K}_z(\mathbf{I} - \mathbf{I}_{N_z})\mathbf{K}_z^\intercal$ [2]
- $\vec{\nu}_i$ eigenvectors of $\mathbf{N}^{-1}\mathbf{M}$ $[N \times N]$
-

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by:
$\vec{M}_z[j] = \frac{1}{N_z}\sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\quad \vec{M}_T[j] = \frac{1}{N}\sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

KDA - the training

"Fisher Discriminant Analysis with Kernels"([SM99])

New trace projection

Computational Complexity $O(D^3)$

LDA

- $\mathbf{S_B} = \sum_{z \in \mathcal{Z}} N_z (\vec{\mu}_s - \overline{\vec{x}})(\vec{\mu}_s - \overline{\vec{x}})^\intercal$
- $\mathbf{S_W} = \sum_{z \in \mathcal{Z}} \sum_{i=1}^{N_z} (\mathbf{x}_i^z - \vec{\mu}_s)(\mathbf{x}_i^z - \vec{\mu}_s)^\intercal$
- $\vec{\alpha}_i$ eigenvectors of $\mathbf{S_W}^{-1}\mathbf{S_B}$ $[D \times D]$
- $\epsilon_\ell^{LDA}(\vec{x}) = \sum_{i=1}^{D} \vec{\alpha}_\ell[i]\vec{x}[i]$

Computational Complexity $O(N^3)$

KDA

- $\mathbf{M} = \sum_{z \in \mathcal{Z}} N_z (\vec{M}_s - \vec{M}_T)(\vec{M}_s - \vec{M}_T)^\intercal$ [1]
- $\mathbf{N} = \sum_{z \in \mathcal{Z}} \mathbf{K}_z(\mathbf{I} - \mathbf{I}_{N_z})\mathbf{K}_z^\intercal$ [2]
- $\vec{\nu}_i$ eigenvectors of $\mathbf{N}^{-1}\mathbf{M}$ $[N \times N]$
- $\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})$

---

[1] $\vec{M}_s$ and $\vec{M}_T$ are two N-sized column vectors whose entries are given by:
$\vec{M}_z[j] = \frac{1}{N_z} \sum_{i:z_i=z}^{N_z} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$, $\quad \vec{M}_T[j] = \frac{1}{N} \sum_{i=1}^{N} K(\mathbf{x}_j^{z_j}, \mathbf{x}_i^{z_i})$.

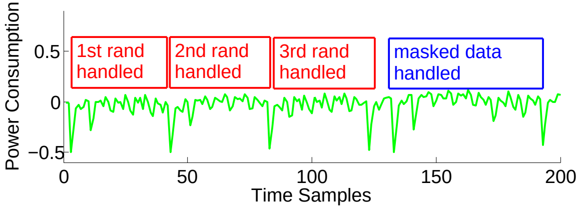
[2] $\mathbf{I}$ is a $N_z \times N_z$ identity matrix, $\mathbf{I}_{N_z}$ is a $N_z \times N_z$ matrix with all entries equal to $\frac{1}{N_z}$ and $\mathbf{K}_z$ is the $N \times N_z$ sub-matrix of $\mathbf{K} = (K(\mathbf{x}_i^{z_i}, \mathbf{x}_j^{z_j}))_{\substack{i=1,\dots,N \\ j=1,\dots,N}}$ storing only columns indexed by the indices i such that $z_i = z$

Experimental setup

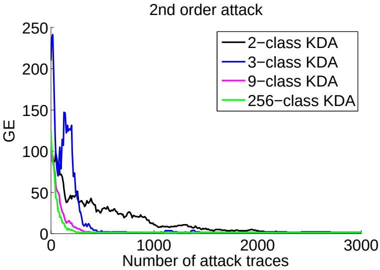
Target device and acquisitions:

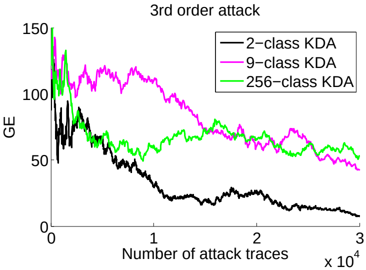
- 8-bit AVR microprocessor Atmega328P
- power-consumption acquired via the ChipWhisperer [OC14] platform
- $D = 200$, 4 clock-cycles are selected
- $9,000$ KDA training traces

Sensitive variable: $Z = \mathrm{Sbox}_{\mathrm{AES}}(P \oplus K^{\star})$
One byte: 256 classes



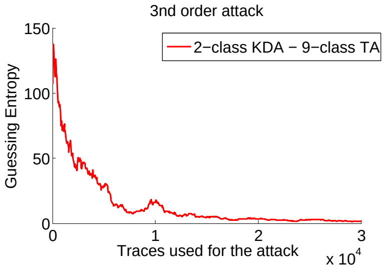
Second and third order

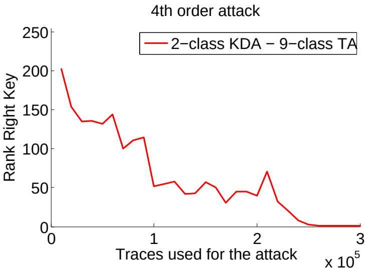




GE = Guessing Entropy (mean rank of the right key candidate)

Third and Fourth Order





- $d = 2 \quad \longrightarrow \quad \mathcal{F} = \mathbb{R}^{\binom{200+2-1}{2}} \Rightarrow$ 20,100 implicit coefficients
- $d = 3 \quad \longrightarrow \quad \mathcal{F} = \mathbb{R}^{\binom{200+3-1}{3}} \Rightarrow$ 1,353,400 implicit coefficients
- $d = 4 \quad \longrightarrow \quad \mathcal{F} = \mathbb{R}^{\binom{200+4-1}{4}} \Rightarrow$ 68,685,050 implicit coefficients

Same time of execution of the KDA algorithm!

Conclusions on KDA

Strong points

- KDA with d-th degree polynomial kernel function is suitable to attack $(d-1)$th-order masking
- KDA computational complexity is independent from the order d
- Tested and effective on a real case, positively compared to PP

	2nd order	3-rd order	4th order
KDA	✓	✓	✓
PP	✓	–	–

Limits and drawbacks

- Memory-based $[\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})]$

Conclusions on KDA

Strong points

- KDA with d-th degree polynomial kernel function is suitable to attack $(d-1)$th-order masking
- KDA computational complexity is independent from the order d
- Tested and effective on a real case, positively compared to PP

	2nd order	3-rd order	4th order
KDA	✓	✓	✓
PP	✓	–	–

Limits and drawbacks

- Memory-based $[\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})] + O(N^3)$ complexity $\rightarrow$ Non-scalability to big training set

Conclusions on KDA

Strong points

- KDA with d-th degree polynomial kernel function is suitable to attack $(d-1)$th-order masking
- KDA computational complexity is independent from the order d
- Tested and effective on a real case, positively compared to PP

	2nd order	3-rd order	4th order
KDA	✓	✓	✓
PP	✓	–	–

Limits and drawbacks

- Memory-based $[\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})] +$ $O(N^3)$ complexity $\rightarrow$ Non-scalability to big training set
- Regularization hyper-parameter μ: $\mathbf{N} = \sum_{z \in \mathcal{Z}} \mathbf{K}_z(\mathbf{I} - \mathbf{I}_{N_z})\mathbf{K}_z^{\intercal} + \mu \mathbf{I}$

Conclusions on KDA

Strong points

- KDA with d-th degree polynomial kernel function is suitable to attack $(d-1)$th-order masking
- KDA computational complexity is independent from the order d
- Tested and effective on a real case, positively compared to PP

	2nd order	3-rd order	4th order
KDA	✓	✓	✓
PP	✓	–	–

Limits and drawbacks

- Memory-based $[\epsilon_\ell^{\mathrm{KDA}}(\vec{x}) = \sum_{i=1}^{N} \vec{\nu}_\ell[i] K(\mathbf{x}_i^{z_i}, \mathbf{x})] + O(N^3)$ complexity $\rightarrow$ Non-scalability to big training set
- Regularization hyper-parameter μ: $\mathbf{N} = \sum_{z \in \mathcal{Z}} \mathbf{K}_z(\mathbf{I} - \mathbf{I}_{N_z})\mathbf{K}_z^{\intercal} + \mu\mathbf{I}$
- No localisation of PoIs

Contents

Motivations

Profiling phase

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\rho(\vec{X}))\mid Z=z}, p_{\epsilon(\rho(\vec{X}))}, p_Z$ (generative model)
    - Gaussian hypothesis (**Template Attack**)[CRR03]
- $p_{Z\mid\epsilon(\rho(\vec{x}))}$ (discriminative model)

Many independent preprocessing steps and assumptions

Motivations

Profiling phase

DEEP LEARNING

- manage de-synchronization problem $[\mathcal{D}_{\text{train}} \longrightarrow \rho\colon \mathbb{R}^D \to \mathbb{R}^D]$
- mandatory dimensionality reduction $[\mathcal{D}_{\text{train}} \longrightarrow \epsilon\colon \mathbb{R}^D \to \mathbb{R}^C]$
- estimate
  - $p_{\epsilon(\rho(\vec{X}))\mid Z=z}$, $p_{\epsilon(\rho(\vec{X}))}$, p_Z (generative model)
    - Gaussian hypothesis (**Template Attack**)[CRR03]
- $p_{Z\mid\vec{X}}$ (discriminative model)
  by means of a neural network $\hat{p}(\vec{x}, W) \approx p_{Z\mid\vec{X}=\vec{x}}$

Many independent preprocessing steps and assumptions
$\longleftrightarrow$ integrated and agnostic approach

Multi-Layer Perceptron

In SCA litterature [MHM13; MZ13; MMT15; MDM16]

Multi-Layer Perceptron (MLP)

$$\hat{p}(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx p_{Z \mid \vec{X}=\vec{x}}$$

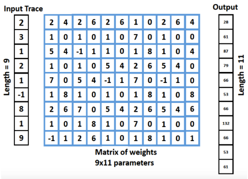
Multi-Layer Perceptron

In SCA litterature [MHM13; MZ13; MMT15; MDM16]

Multi-Layer Perceptron (MLP)

$$\hat{p}(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx p_{Z \mid \vec{X}=\vec{x}}$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W



Multi-Layer Perceptron

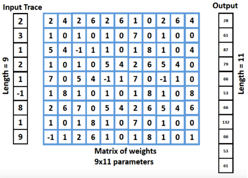
In SCA litterature [MHM13; MZ13; MMT15; MDM16]

Multi-Layer Perceptron (MLP)

$$\hat{p}(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx p_{Z \mid \vec{X}=\vec{x}}$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

σ_i non-linear *activation* functions



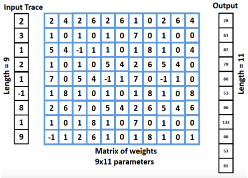
Multi-Layer Perceptron

In SCA litterature [MHM13; MZ13; MMT15; MDM16]

Multi-Layer Perceptron (MLP)

$$\hat{p}(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx p_{Z \mid \vec{X}=\vec{x}}$$

- λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W
- σ_i non-linear *activation* functions
- s normalizing *softmax* function



Multi-Layer Perceptron

In SCA litterature [MHM13; MZ13; MMT15; MDM16]

Multi-Layer Perceptron (MLP)

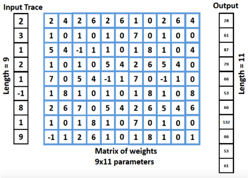
$$\hat{p}(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx p_{Z \mid \vec{X}=\vec{x}}$$

λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

σ_i non-linear *activation* functions

s normalizing *softmax* function

Architecture hyper-parameters



Multi-Layer Perceptron

In SCA litterature [MHM13; MZ13; MMT15; MDM16]

Multi-Layer Perceptron (MLP)

$$\hat{p}(\vec{x}, W) = s \circ \lambda_n \circ \sigma_{n-1} \circ \lambda_{n-1} \circ \cdots \circ \lambda_1(\vec{x}) = \vec{y} \approx p_{Z \mid \vec{X}=\vec{x}}$$

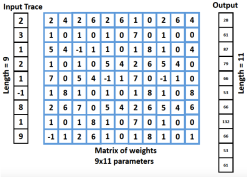
λ_i linear functions (linear combinations of time samples) depending on some **trainable weights** W

σ_i non-linear *activation* functions

s normalizing *softmax* function

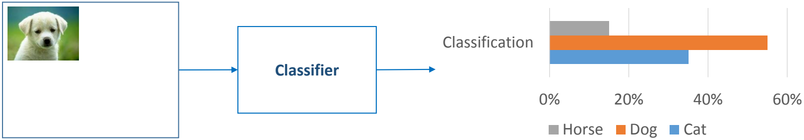
Architecture hyper-parameters

Universal approximation theorem



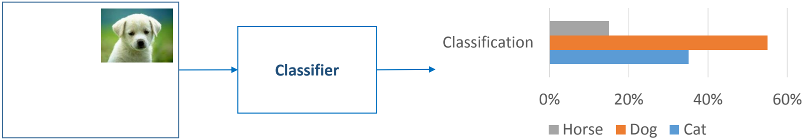
Convolutional Neural Networks

Translation-Invariance



Convolutional Neural Networks

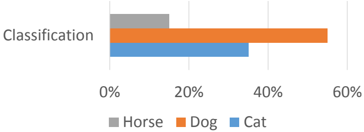
Translation-Invariance



Convolutional Neural Networks

Translation-Invariance

Classifier



Convolutional Neural Networks

Translation-Invariance

Convolutional Neural Networks

Translation-Invariance

Convolutional Neural Networks

Translation-Invariance

Convolutional Layers

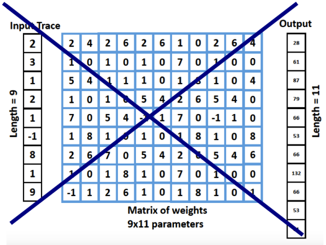


Figure: Linear layer in an MLP.

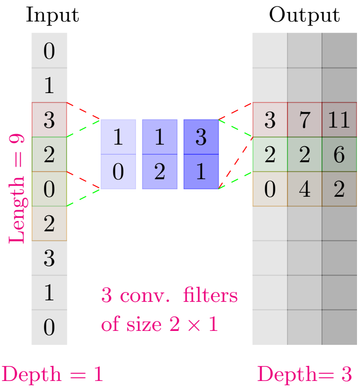
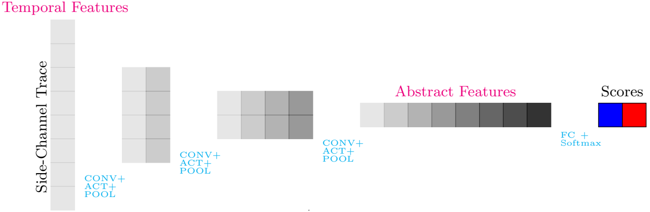


Figure: Convolutional layer in a CNN.

A kind of CNN architecture



Architecture inspired by AlexNet [KSH12], VGG [SZ14], ResNet [He+16] design rules:

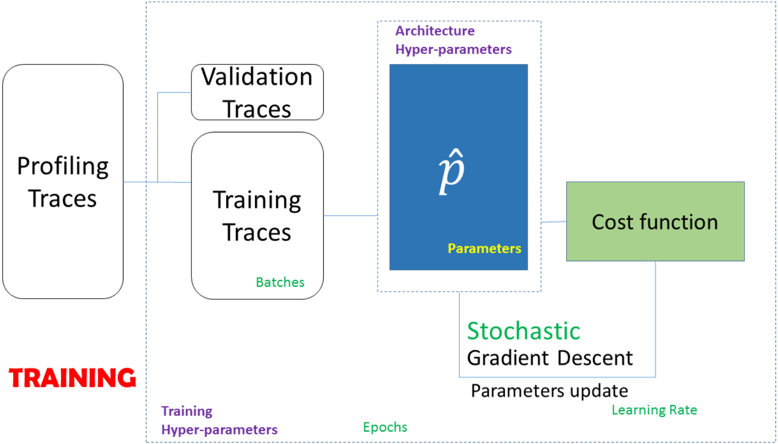
- Reduce temporal features to only one
- Maintain time complexity of each layer (one-half pooling when number of feature maps is doubled)

Model used in our experiments

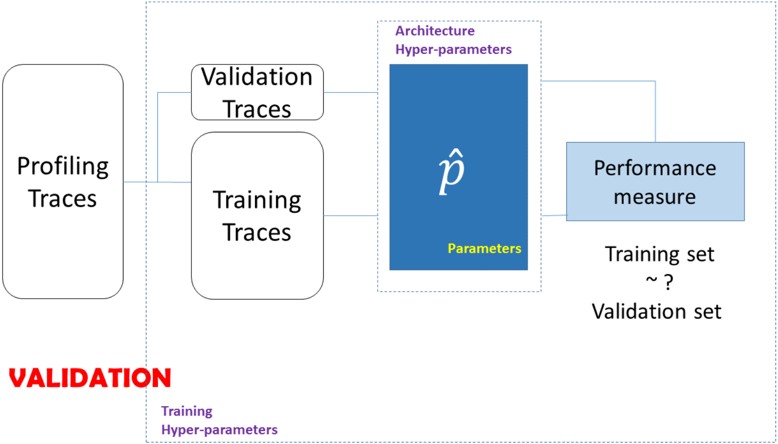
- 4 Conv + Pool layers
- tanh activations
- batch normalisation [IS15]
- 1 *fully connected layer* + softmax

Training and Validation

Training and Validation



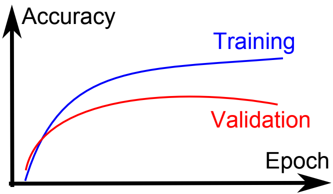
Training and Validation



Overfitting

Evaluate and compare training and validation accuracy

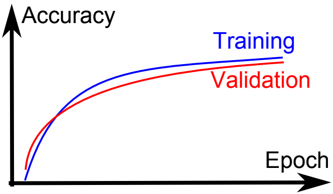
Learn by heart (**OVERFITTING**)



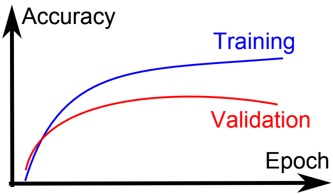
Overfitting

Evaluate and compare training and validation accuracy

Understand significant features



Learn by heart (**OVERFITTING**)

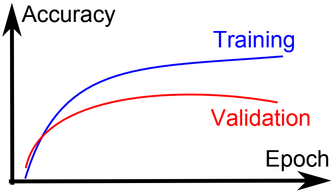


Overfitting

Evaluate and compare training and validation accuracy

- Why?
  - Too complex model
  - Not enough training data
- Solution?
  - Reduce model capacity
  - Regularization
  - Dropout
  - Data augmentation

Learn by heart (**OVERFITTING**)



Overfitting

Evaluate and compare training and validation accuracy

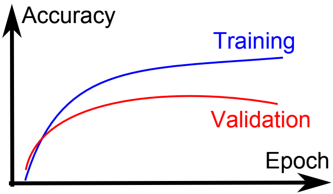
Why?

- Too complex model
- Not enough training data

Solution?

- Reduce model capacity
- Regularization
- Dropout
- Data augmentation

Learn by heart (**OVERFITTING**)



Data Augmentation

Data Augmentation

Artificially generate new training data by deforming those previously acquired,
Applying transformations that preserve the label Z

Countermeasure Emulation Idea

Emulate the effects of misaligning countermeasures to generate new traces

SHIFTING

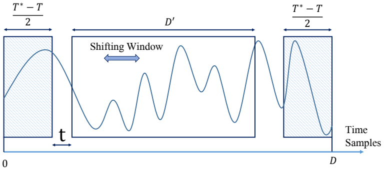


Figure: SH_T

ADD-REMOVE

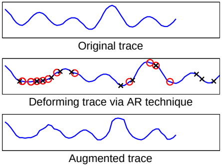


Figure: AR_R

Parameter T: $\sharp$ of possible positions
Parameter R: $\sharp$ of added and removed points
Data Augmentation techniques are applied online during training phase.

Experimental Results

- Random delays (software countermeasure)
- Artificial Jitter (simulated hardware countermeasure)
- Real Jitter (hardware countermeasure)

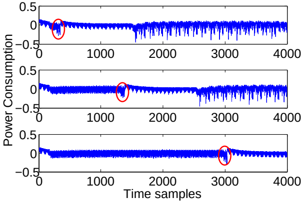
Keras 1.2.1 library with Tensorflow backend [Cho+15] (open source, today 2.2.4)

Experimental Results

- Random delays (software countermeasure)
- Artificial Jitter (simulated hardware countermeasure)
- Real Jitter (hardware countermeasure)

Keras 1.2.1 library with Tensorflow backend [Cho+15] (open source, today 2.2.4)

Random delays



(a) One leaking operation

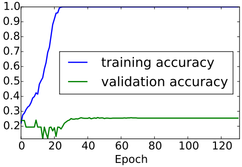
Setup

- Target Chip: Atmega328P
- Target Variable: $Z = \mathrm{HW}(\mathrm{Sbox}(P \oplus K))$
- Acquisition: through ChipWhisperer[OC14] platform, $\approx 4,000$ time samples
- Countermeasure: Random Delays - insertion of r *nop* operations, $r \in [0, 127]$ uniform random
- $1,000$ training traces

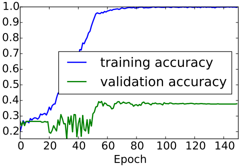
Random delays

Data augmentation vs overfitting

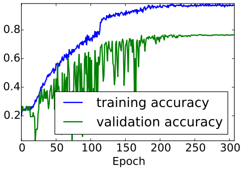
Training



SH_0



SH_{100}

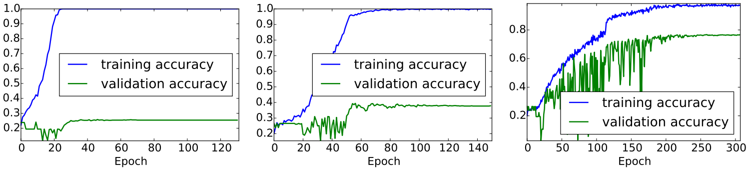


SH_{500}

Random delays

Data augmentation vs overfitting

Training



SH_0 SH_{100} SH_{500}

Attack

		SH_0		SH_{100}		SH_{500}	
Accuracy	$N^\star$	27.0%	$> 1,000$	31.8%	101	**78%**	**7**

Table: $N^\star$ = number of attack traces to have GE $= 1$.

Conclusions about CNN

- CNNs provide an integrated approach to construct a discriminative model from misaligned data

Conclusions about CNN

- CNNs provide an integrated approach to construct a discriminative model from misaligned data
- CNN models may have high capacity and require plenty of data to be trained

Conclusions about CNN

- CNNs provide an integrated approach to construct a discriminative model from misaligned data
- CNN models may have high capacity and require plenty of data to be trained
- Side-Channel-adapted Data Augmentation techniques

Conclusions about CNN

- CNNs provide an integrated approach to construct a discriminative model from misaligned data
- CNN models may have high capacity and require plenty of data to be trained
- Side-Channel-adapted Data Augmentation techniques
- Effectiveness/efficiency of the CNN+Data Augmentation approach experimentally verified

Contents

Conclusions

- A wide part of Side-Channel literature consider leakages localised in small and known portions of signal
- In practical context, curse of dimensionality affects the potential optimality of profiling attacks
- In many domains Machine Learning solutions are used to tackle it
- Profiling attacks $\approx$ classification task
- Generative model approach:
  - Classification-oriented techniques for dimensionality reduction
  - LDA and KDA generalization to tackle masking countermeasure
- Discriminative model approach:
  - Neural Networks, big data scalability
  - CNN to integrate resynchronization in a unique model construction process

Conclusions

- A wide part of Side-Channel literature consider leakages localised in small and known portions of signal
- In practical context, curse of dimensionality affects the potential optimality of profiling attacks
- In many domains Machine Learning solutions are used to tackle it
- Profiling attacks $\approx$ classification task
- Generative model approach:
  - Classification-oriented techniques for dimensionality reduction
  - LDA and KDA generalization to tackle masking countermeasure
- Discriminative model approach:
  - Neural Networks, big data scalability
  - CNN to integrate resynchronization in a unique model construction process

Today and in the future

- ASCAD database and SCA/DL community
- From CNN to PoIs, visualizing techniques
- Advanced-attack-oriented machine learning task (instead of multiple classification)
- Collision attacks $\approx$ verification task (siamese network)

Conclusions

- A wide part of Side-Channel literature consider leakages localised in small and known portions of signal
- In practical context, curse of dimensionality affects the potential optimality of profiling attacks
- In many domains Machine Learning solutions are used to tackle it
- Profiling attacks $\approx$ classification task
- Generative model approach:
  - Classification-oriented techniques for dimensionality reduction
  - LDA and KDA generalization to tackle masking countermeasure
- Discriminative model approach:
  - Neural Networks, big data scalability
  - CNN to integrate resynchronization in a unique model construction process

Today and in the future

- ASCAD database and SCA/DL community
- From CNN to PoIs, visualizing techniques
- Advanced-attack-oriented machine learning task (instead of multiple classification)
- Collision attacks $\approx$ verification task (siamese network)

Thank You!

References I

[Arc+06] C. Archambeau et al. "Template Attacks in Principal Subspaces". English. In: *Cryptographic Hardware and Embedded Systems - CHES 2006*. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 1–14. ISBN: 978-3-540-46559-1. DOI: 10.1007/11894063_1. URL: http://dx.doi.org/10.1007/11894063_1.

[BDP10] Martin Bär, Hermann Drexler, and Jürgen Pulkus. "Improved template attacks". In: *COSADE2010* (2010).

[BHW12] Lejla Batina, Jip Hogenboom, and Jasper G.J. van Woudenberg. "Getting More from PCA: First Results of Using Principal Component Analysis for Extensive Power Analysis". English. In: *Topics in Cryptology CT-RSA 2012*. Ed. by Orr Dunkelman. Vol. 7178. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 383–397. ISBN: 978-3-642-27953-9. DOI: 10.1007/978-3-642-27954-6_24. URL: http://dx.doi.org/10.1007/978-3-642-27954-6_24.

References II

[Bat+11] Lejla Batina et al. "Mutual information analysis: a comprehensive study". In: *Journal of Cryptology* 24.2 (2011), pp. 269–291.

[Bha+14] Shivam Bhasin et al. "Side-channel leakage and trace compression using normalized inter-class variance". In: *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*. ACM. 2014, p. 7.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation power analysis with a leakage model". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2004, pp. 16–29.

[Bru+15] Nicolas Bruneau et al. "Less is more". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2015, pp. 22–41.

References III

[CRR03] Suresh Chari, JosyulaR. Rao, and Pankaj Rohatgi. "Template Attacks". English. In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Ed. by Burton S. Kaliski, Cetin K. Koc, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2003, pp. 13–28. ISBN: 978-3-540-00409-7. DOI: 10.1007/3-540-36400-5_3. URL: http://dx.doi.org/10.1007/3-540-36400-5_3.

[Cho+15] François Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[CK14] Omar Choudary and Markus G Kuhn. "Efficient template attacks". In: *Smart Card Research and Advanced Applications*. Springer, 2014, pp. 253–270.

[Dur+15] François Durvaux et al. "Efficient selection of time samples for higher-order DPA with projection pursuits". In: *Constructive Side-Channel Analysis and Secure Design*. Springer, 2015, pp. 34–50.

References IV

[GLRP06] Benedikt Gierlichs, Kerstin Lemke-Rust, and Christof Paar. "Templates vs. stochastic methods". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2006, pp. 15–29.

[Gie+08] Benedikt Gierlichs et al. "Mutual information analysis". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2008, pp. 426–442.

[He+16] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

References V

[HZ12] Annelie Heuser and Michael Zohner. “Intelligent Machine Homicide”. English. In: *Constructive Side-Channel Analysis and Secure Design*. Ed. by Werner Schindler and SorinA. Huss. Vol. 7275. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 249–264. ISBN: 978-3-642-29911-7. DOI: 10.1007/978-3-642-29912-4_18. URL: http://dx.doi.org/10.1007/978-3-642-29912-4_18.

[Hos+11] Gabriel Hospodar et al. “Machine learning in side-channel analysis: a first study”. English. In: *Journal of Cryptographic Engineering* 1.4 (2011), pp. 293–302. ISSN: 2190-8508. DOI: 10.1007/s13389-011-0023-x. URL: http://dx.doi.org/10.1007/s13389-011-0023-x.

[IS15] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). URL: http://arxiv.org/abs/1502.03167.

References VI

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. "Differential power analysis". In: *Annual International Cryptology Conference*. Springer. 1999, pp. 388–397.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.* 2012, pp. 1106–1114.

[LBM15] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. "A machine learning approach against a masked AES". In: *Journal of Cryptographic Engineering* 5.2 (2015), pp. 123–139.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. "Power analysis attack: an approach based on machine learning". In: *International Journal of Applied Cryptography* 3.2 (2014), pp. 97–115.

References VII

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*. Vol. 31. Springer Science & Business Media, 2008.

[MDM16] Zdenek Martinasek, Petr Dzurenda, and Lukas Malina. "Profiling power analysis attack based on MLP in DPA contest V4. 2". In: *Telecommunications and Signal Processing (TSP), 2016 39th International Conference on*. IEEE. 2016, pp. 223–226.

[MHM13] Zdenek Martinasek, Jan Hajny, and Lukas Malina. "Optimization of power analysis using neural network". In: *International Conference on Smart Card Research and Advanced Applications*. Springer. 2013, pp. 94–107.

[MMT15] Zdenek Martinasek, Lukas Malina, and Krisztina Trasy. "Profiling power analysis attack based on multi-layer perceptron network". In: *Computational Problems in Science and Engineering*. Springer, 2015, pp. 317–339.

References VIII

[MZ13] Zdenek Martinasek and Vaclav Zeman. "Innovative method of the power analysis". In: *Radioengineering* 22.2 (2013), pp. 586–594.

[OC14] Colin O'Flynn and Zhizhang David Chen. "ChipWhisperer: An open-source platform for hardware embedded security research". In: *Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.

[Osw+06] Elisabeth Oswald et al. "Practical Second-Order DPA Attacks for Masked Smart Card Implementations of Block Ciphers". English. In: *Topics in Cryptology CT-RSA 2006*. Ed. by David Pointcheval. Vol. 3860. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 192–207. ISBN: 978-3-540-31033-4. DOI: 10.1007/11605805_13. URL: http://dx.doi.org/10.1007/11605805_13.

References IX

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. “A stochastic model for differential side channel cryptanalysis”. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2005, pp. 30–46.

[SM99] Bernhard Schölkopf and Klaus-Robert Mullert. “Fisher discriminant analysis with kernels”. In: *Neural networks for signal processing IX* 1 (1999), p. 1.

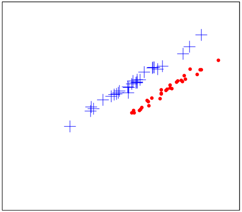
[SZ14] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

References X

[SA08] François-Xavier Standaert and Cedric Archambeau. "Using Subspace-Based Template Attacks to Compare and Combine Power and Electromagnetic Information Leakages". English. In: *Cryptographic Hardware and Embedded Systems CHES 2008*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Vol. 5154. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 411–425. ISBN: 978-3-540-85052-6. DOI: 10.1007/978-3-540-85053-3_26. URL: http://dx.doi.org/10.1007/978-3-540-85053-3_26.

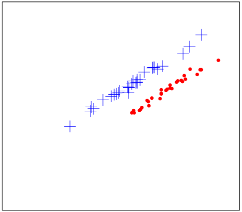
[TM97] Mitchell T. M. *Machine Learning*. McGraw-Hill, New York, 1997.

LDA: an optimal binary linear classifier



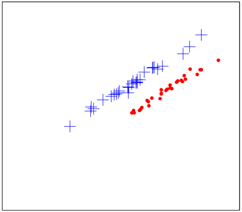
- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$

LDA: an optimal binary linear classifier



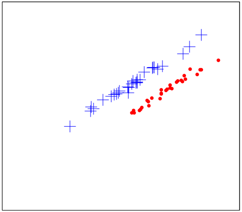
- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$

LDA: an optimal binary linear classifier



- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

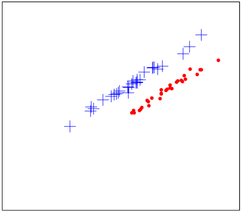
LDA: an optimal binary linear classifier



- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log \left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j

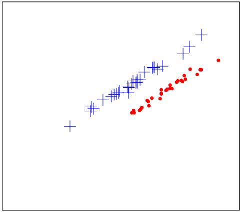
LDA: an optimal binary linear classifier



- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j
- $\Rightarrow a = \vec{w}^\intercal \vec{x} + w_0$ (linear decision boundary, $\vec{w}$ and w_0 functions of Σ, μ_j)

LDA: an optimal binary linear classifier



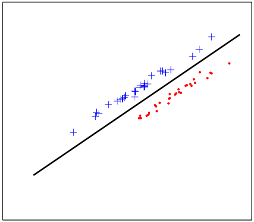
- Classify data $\vec{x}$ into 2 classes $\mathcal{Z} = \{s_1, s_2\}$
- Generative model: $p_{\vec{X} \mid Z=s_j}(\vec{x})$, $p_Z(s_j)$ and $p_{\vec{X}}(\vec{x})$
- Posterior probabilities (via Bayes' theorem), then classify through the *log-likelihood ratio*: $a = \log\left[\frac{\Pr(s_1 \mid \vec{x})}{\Pr(s_2 \mid \vec{x})}\right]$ (boundary surface $a = 0$)

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j
- $\Rightarrow a = \vec{w}^\intercal \vec{x} + w_0$ (linear decision boundary, $\vec{w}$ and w_0 functions of Σ, μ_j)

Generalised linear discriminative model

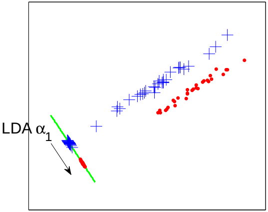
$$\Pr(s_1 \mid \vec{x}) = \sigma(\vec{w}^\intercal \vec{x} + w_0) \text{ ,where } \sigma(a) = \frac{1}{1+e^{-a}} \textit{ logistic sigmoid} \qquad (2)$$

LDA and Fisher Criterion



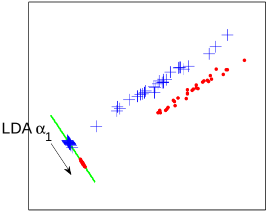
- LDA: linear decision boundary
  $a = \vec{w}^\intercal \vec{x} + w_0$

LDA and Fisher Criterion



- LDA: linear decision boundary
  $a = \vec{w}^\intercal \vec{x} + w_0$
- Equivalently, project data onto $\vec{w}^\intercal \vec{x}$ (orthogonally to the decision boudary), than classify by a real threshold (optimally w_0).

LDA and Fisher Criterion



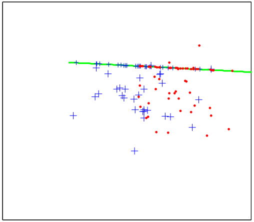
- LDA: linear decision boundary $a = \vec{w}^\intercal \vec{x} + w_0$
- Equivalently, project data onto $\vec{w}^\intercal \vec{x}$ (orthogonally to the decision boudary), than classify by a real threshold (optimally w_0).

- Two assumptions about class-conditional densities:
  - Gaussian distributions with parameters μ_j, Σ_j
  - Homoscedasticity: $\Sigma_j = \Sigma$ for all j

Fact, abuse and preference for the dimensionality reduction formulation

- When LDA assumptions are met, the solution $\vec{\alpha}_1$ of the Fisher's criterion is orthogonal to $\vec{w}$.
- assumption not required
- naturally multi-class

Linear separability

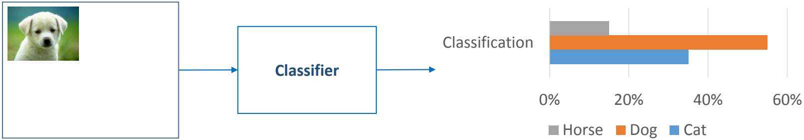


LDA: linear decision boundary $a = \vec{w}^\intercal \vec{x} + w_0$ $(\vec{w} = \Sigma^{-1}(\mu_1 - \mu_2))$

What if $\mu_1 = \mu_2$?

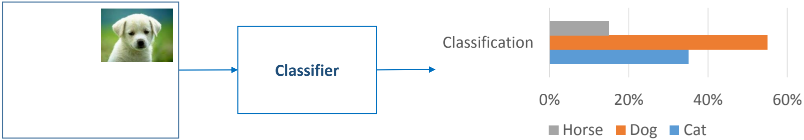
Convolutional Neural Networks

Translation-invariance



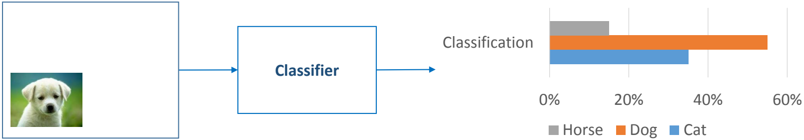
Convolutional Neural Networks

Translation-invariance



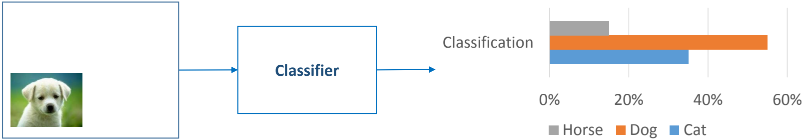
Convolutional Neural Networks

Translation-invariance



Convolutional Neural Networks

Translation-invariance



It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

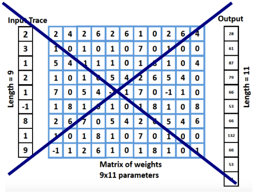


Figure: Linear layer in an MLP (*Fully-Connected*

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

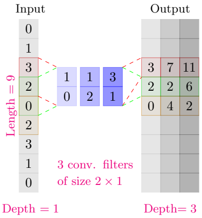


Figure: Linear layer in a ConvNet (*Convolutional*

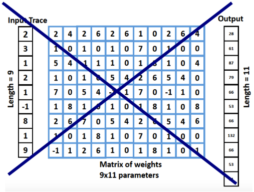


Figure: Linear layer in an MLP (*Fully-Connected*

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

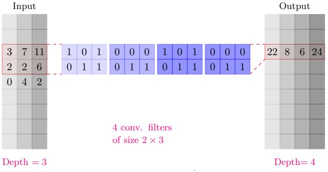


Figure: Linear layer in a ConvNet (*Convolutional Layer*)

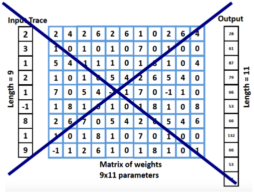


Figure: Linear layer in an MLP (*Fully Connected*

Convolutional Neural Networks

Translation-invariance

It is important to explicit the data translation-invariance
Convolutional Neural Networks: share weights across space

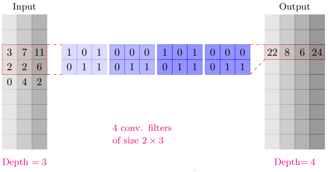


Figure: Linear layer in a ConvNet (*Convolutional Layer*)

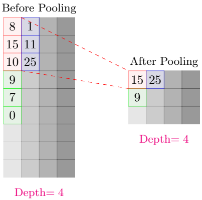


Figure: Max Pooling Layer

Cost function - Cross-entropy

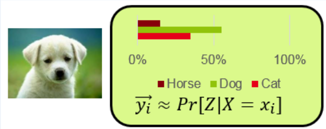
- batch of training data $(\vec{x}_i, z_i)_{i\in I}$, outputs of the current model $(\vec{y}_i)_{i\in I}$
- labels $z_i = s_j$ are *one-hot encoded*: $\vec{z}_i = \vec{s}_j = (0,\dots,0,\underbrace{1}_{j},0,\dots,0)$

Loss function

$$\mathcal{L} = -\frac{1}{|I|}\sum_{i\in I}\sum_{t=1}^{|\mathcal{Z}|}\vec{z}_i[t]\log\vec{y}_i[t] \tag{3}$$

Maximum-*a-posteriori* or Cross-entropy

- $\vec{y}_i \approx \Pr[Z \mid \vec{X} = \vec{x}_i]$



Cost function - Cross-entropy

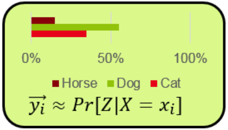
- batch of training data $(\vec{x}_i, z_i)_{i \in I}$, outputs of the current model $(\vec{y}_i)_{i \in I}$
- labels $z_i = s_j$ are *one-hot encoded*: $\vec{z}_i = \vec{s}_j = (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)$

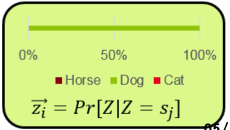
Loss function

$$\mathcal{L} = -\frac{1}{|I|} \sum_{i \in I} \sum_{t=1}^{|\mathcal{Z}|} \vec{z}_i[t] \log \vec{y}_i[t] \quad (3)$$

Maximum-*a-posteriori* or Cross-entropy

- $\vec{y}_i \approx \Pr[Z \mid \vec{X} = \vec{x}_i]$
- $\vec{z}_i \approx \Pr[Z \mid Z = \vec{s}_j]$
- $\mathbb{H}(\vec{z}_i, \vec{y}_i) = \mathbb{H}(\vec{z}_i) + D_{KL}(\vec{z}_i || \vec{y}_i) = \mathbb{E}_{\vec{z}_i}[-\log \vec{y}_i] = -\sum_{t=1}^{|\mathcal{Z}|} \vec{z}_i[t] \log \vec{y}_i[t]$





Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

MSE_train=44.228280, MSE_test=330.984916

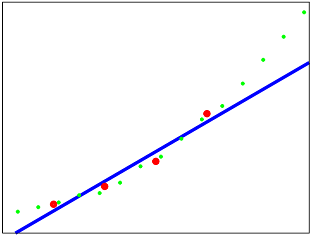


Figure: Linear regression → underfitting

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

MSE_train=44.228280, MSE_test=330.984916

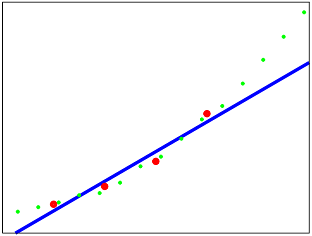


Figure: Linear regression → underfitting

MSE_train=2.243097, MSE_test=61.891672

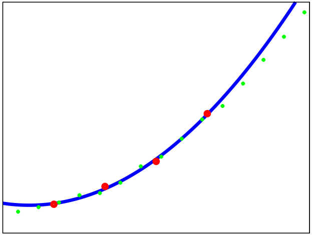


Figure: Quadratic regression → fits

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

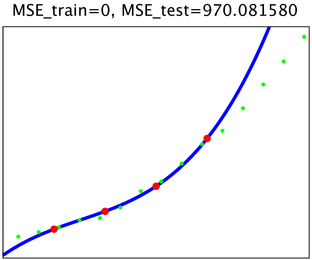


Figure: Cubic regression → overfitting

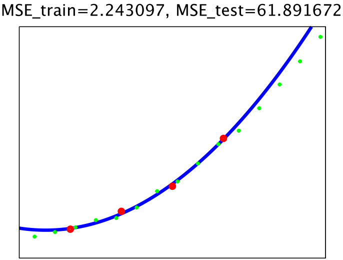


Figure: Quadratic regression → fits

Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

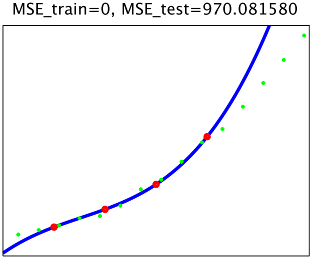


Figure: Cubic regression → overfitting

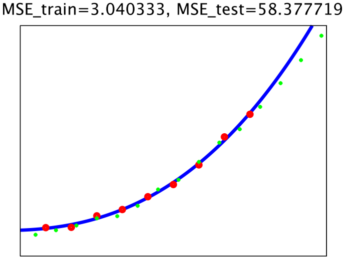


Figure: Cubic regression with more training data

Capacity-Overfitting-Regularization

Regression example

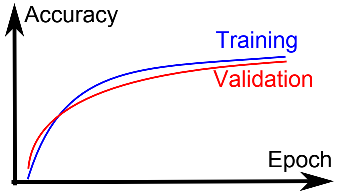
Performance metric: Mean Square Error (MSE)

Classification via Neural Network

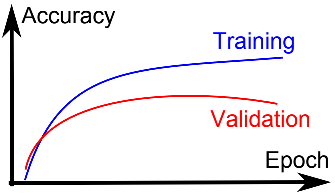
Performance measure: Accuracy (Classification rate)

Evaluate and compare training and validation accuracy

Understand significant features



Learn by heart (**OVERFITTING**)



Capacity-Overfitting-Regularization

Regression example

Performance metric: Mean Square Error (MSE)

Classification via Neural Network

Performance measure: Accuracy (Classification rate)
Evaluate and compare training and validation accuracy

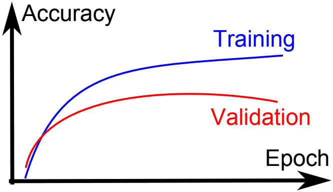
Learn by heart (**OVERFITTING**)

Why?

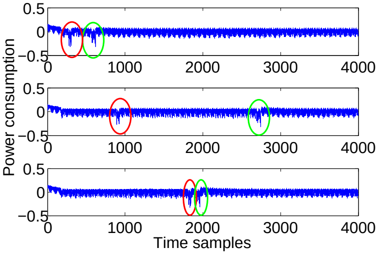
- Too complex model
- Not enough training data

Solution?

- Data augmentation



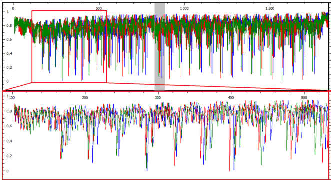
Random Delays - Two Leaking Operations



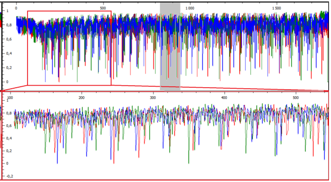
Two leaking operations

First operation - Test acc: 76.8%, $N^{\star} = 7$
Second operation - Test acc: 82.5%, $N^{\star} = 6$

Artificial Jitter



(a) Low artificial jitter



(b) High artificial jitter

Target

- Target Variable: $Z = \mathrm{HW}(\mathrm{Sbox}(P \oplus K))$
- $\approx$ 2000 time samples
- Countermeasure: artificial signal treatment simulating clock jitter
- 10000 training traces

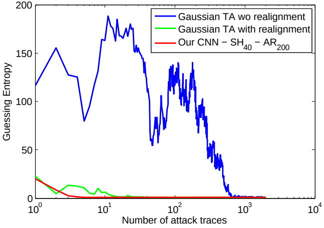
Artificial Jitter (2)

Low_jitter

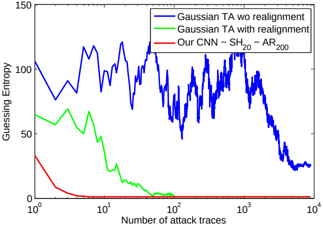
Acc	$N^\star$	SH_0		SH_{20}		SH_{40}	
AR_0		57.4%	14	82.5%	6	83.6%	6
AR_{100}		86.0%	6	87.0%	5	87.5%	6
AR_{200}		86.6%	6	85.7%	6	87.7%	5

High_jitter

Acc	$N^\star$	SH_0		SH_{20}		SH_{40}	
AR_0		40.6%	35	51.1%	9	62.4%	11
AR_{100}		50.2%	15	72.4%	11	73.5%	9
AR_{200}		64.0%	11	75.5%	8	74.4%	8



(c) Low Jitter



(d) High Jitter

Artificial Jitter

DS_low_jitter									
a / c	b / d	SH_0		SH_{20}		SH_{40}		SH_{200}	
AR_0		100.0% / 57.4%	68.7% / 14	99.8% / 82.5%	86.1% / 6	98.9% / 83.6%	84.1% / 6		
AR_{100}		87.7% / 86.0%	88.2% / 6	82.4% / 87.0%	88.4% / 5	81.9% / 87.5%	89.6% / 6		
AR_{200}		83.2% / 86.6%	88.6% / 6	81.4% / 85.7%	86.9% / 6	80.6% / 87.7%	88.9% / 5		
AR_{500}								85.0% / 86.2%	88.6% / 5

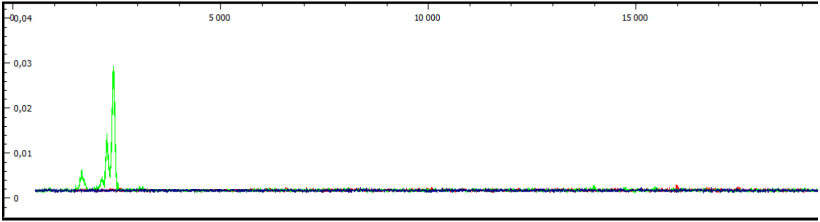
DS_high_jitter									
a / c	b / d	SH_0		SH_{20}		SH_{40}		SH_{200}	
AR_0		100% / 40.6%	45.0% / 35	100% / 51.1%	60.0% / 9	98.5% / 62.4%	67.6% / 11		
AR_{100}		90.4% / 50.2%	57.3% / 15	76.6% / 72.4%	73.6% / 11	78.5% / 73.5%	76.4% / 9		
AR_{200}		83.1% / 64.0%	67.7% / 11	82.0% / 75.5%	77.1% / 8	82.6% / 74.4%	77.0% / 8		
AR_{500}								83.6% / 68.2%	73.4% / 11

Real Jitter (1)

Target

- AES hardware implementation
- strong jitter effect
- Target Variable: $Z = \mathrm{Sbox}(P \oplus K)$
- $2,500$ selected time samples
- $99,000$ training traces

SNR first Sbox

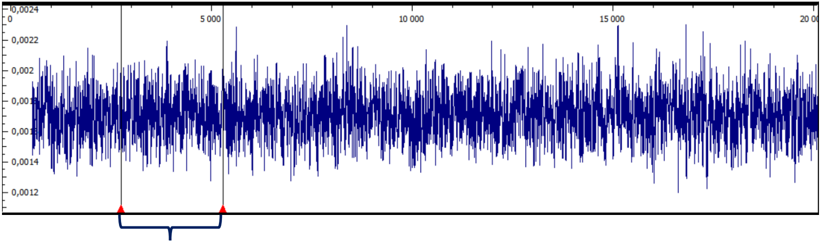


Real Jitter (1)

Target

- AES hardware implementation
- strong jitter effect
- Target Variable: $Z = \mathrm{Sbox}(P \oplus K)$
- $2,500$ selected time samples
- $99,000$ training traces

SNR second Sbox without realignment



Entry region for CNN (2,500 pts)

Real Jitter (2)

		SH_0AR_0		$SH_{10}AR_{100}$		$SH_{20}AR_{200}$	
Acc	$N^\star$	1.2%	137	1.3%	89	1.8%	54

Real Jitter (2)

		$SH_0 AR_0$		$SH_{10} AR_{100}$		$SH_{20} AR_{200}$	
Acc	$N^\star$	1.2%	137	1.3%	89	1.8%	54

SNR second Sbox with realignment