Power Analysis Attacks

Side-Channel Security

Rishub Nagpal

May 16, 2024

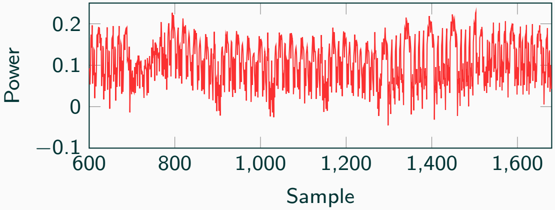
IAIK – Graz University of Technology

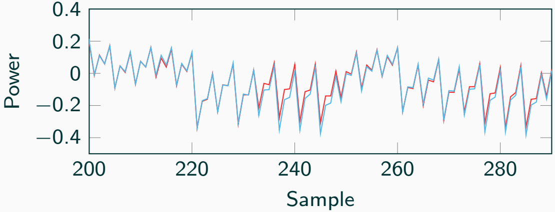
Overview

Recap

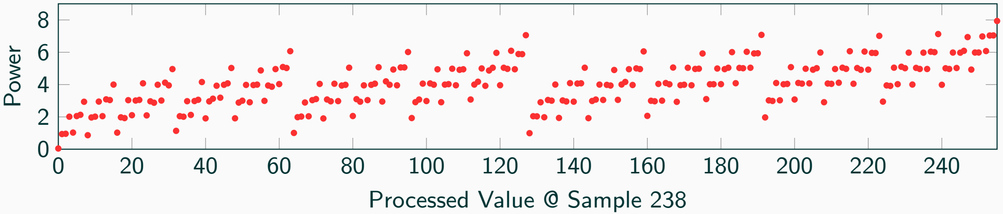
Last Time...

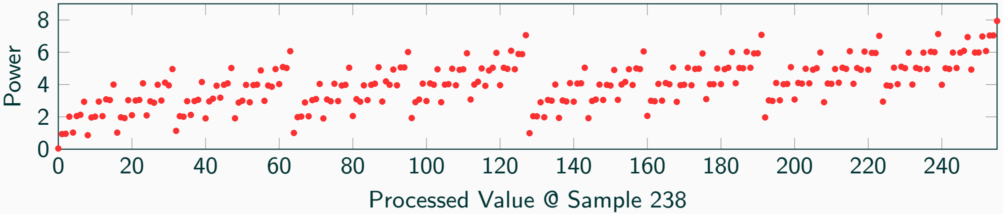
- Power consumption depends on
    - Executed operation
    - Processed data
- Now: Exploitation





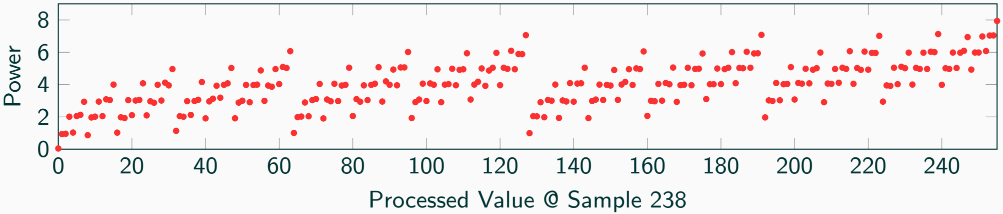
(Rough) Taxonomy





- How many measurements (traces) do we have?
    - One: Only a single execution of the cryptographic algorithm
    - Many: Record many executions, each using different inputs

(Rough) Taxonomy



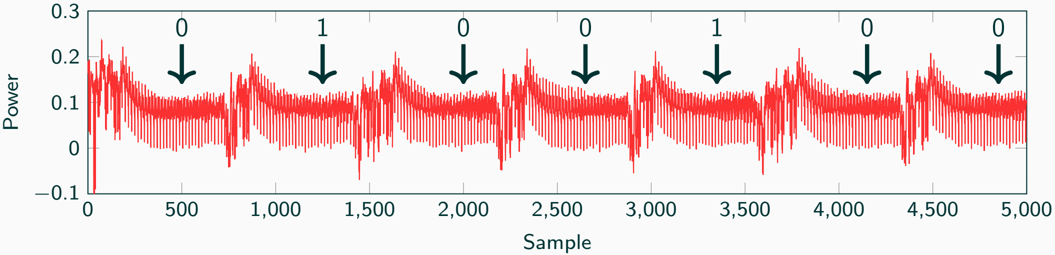
- How many measurements (traces) do we have?
  - One: Only a single execution of the cryptographic algorithm
  - Many: Record many executions, each using different inputs
- Do we perform profiling?
  - YES: Value x causes power consumption p
  - NO: We use a model e.g. $p(x) \approx$ Hamming weight(x)

(Rough) Taxonomy

	Non-profiled Attacks	**Profiled Attacks**
One or few observations with fixed data	**Simple SCA**	Profiled simple SCA
Many observations with varying data	Differential SCA	Profiled differential SCA

Non-Profiled Attacks

"Simple" Side-Channel Analysis



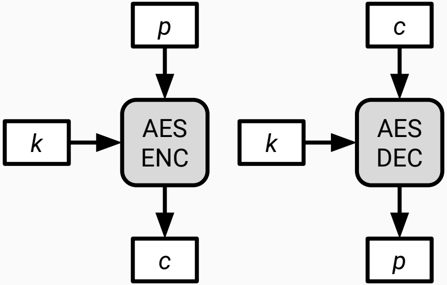
- Derive key directly from one or very few power traces
- Often requires detailed knowledge about the implementation and more complex statistical models
- No profiling
- But what about symmetric crypto?
  - Constant control flow, only data leakage

(Rough) Taxonomy

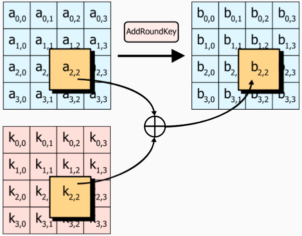
	Non-profiled Attacks	**Profiled Attacks**
One or few observations with fixed data	Simple SCA	Profiled simple SCA
Many observations with varying data	**Differential SCA**	Profiled differential SCA

Preliminaries: AES

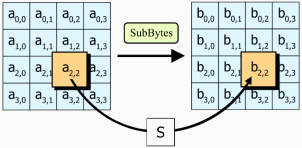
- Advanced Encryption Standard
- Block cipher with key size: **128**/192/256 bit
- Symmetric
- State size: 128 bit
  - Organized as 4×4 bytes
- 4 round functions
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddRoundKey
- 10 rounds in total (+ initial round)



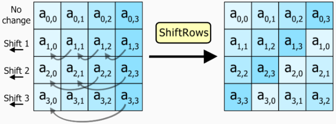
Preliminaries: AES



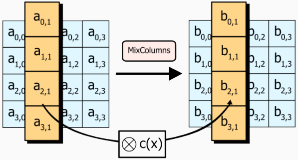
1. AddRoundKey



2. SubBytes

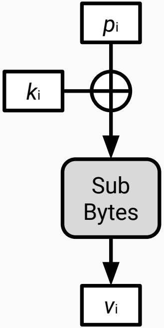


3. ShiftRows



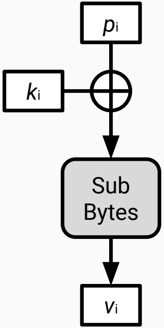
4. MixColumns

- Initial/first round
- Round key $= k$
- Other roundkeys are derived from AES key schedule



for i in $0 \ldots 15$

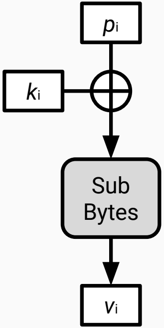
- Lets assume we “attack” an AES implementation with a known key...



for i in $0 \ldots 15$

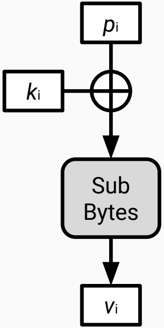
Preliminaries: Predicting Power Consumption

- Lets assume we "attack" an AES implementation with a known key...
- We can:
  - can request the encryption of a known plaintext
  - calculate intermediate values of corresponding AES computations
    - For example v_0 with $v_0 = \text{SubBytes}(p_0 \oplus k_0)$
  - predict the power consumption of, e.g., $\texttt{MOV}(v_0)$ with a power model



for i in $0 \dots 15$

- Lets assume we "attack" an AES implementation with a known key...
- We can:
  - can request the encryption of a known plaintext
  - calculate intermediate values of corresponding AES computations
    - For example v_0 with $v_0 = \text{SubBytes}(p_0 \oplus k_0)$
  - predict the power consumption of, e.g., $\texttt{MOV}(v_0)$ with a power model
- Repeat these steps x-times using different plaintexts
  - $\rightarrow x$ power traces with x corresponding predictions for the power consumption of $\texttt{MOV}(v_0)$



for i in $0 \ldots 15$

- For each point in time we have in total x samples
  - Corresponding to the x power traces

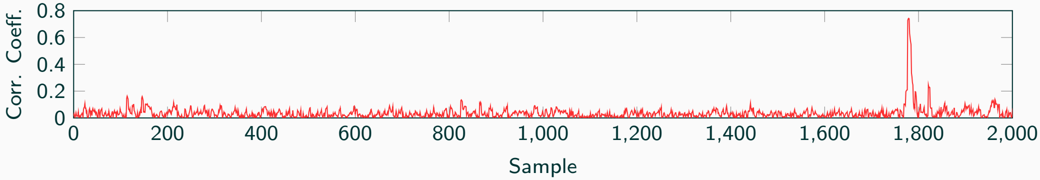
Preliminaries: Testing Predictions of Power Consumption

- For each point in time we have in total x samples
  - Corresponding to the x power traces
- We also have in total x power predictions of $\texttt{MOV}(v_0)$

- For each point in time we have in total x samples
  - Corresponding to the x power traces
- We also have in total x power predictions of $\texttt{MOV}(v_0)$
- Let's correlate them!

- For each point in time we have in total x samples
    - Corresponding to the x power traces
- We also have in total x power predictions of $\texttt{MOV}(v_0)$
- Let's correlate them!
- But when does the $\texttt{MOV}(v_0)$ occur in the power trace?
    - ¯_(ツ)_/¯

- For each point in time we have in total x samples
  - Corresponding to the x power traces
- We also have in total x power predictions of $\texttt{MOV}(v_0)$
- Let's correlate them!
- But when does the $\texttt{MOV}(v_0)$ occur in the power trace?
  - ¯_(ツ)_/¯
- Let's just try all possible points in time:



- We can model the power consumption of the processing of certain intermediate values!

- We can model the power consumption of the processing of certain intermediate values!
- But we require knowledge of the key to calculate v_0 in first place...
    - $v_0 = \text{SubBytes}(p_0 \oplus k_0)$

- We can model the power consumption of the processing of certain intermediate values!
- But we require knowledge of the key to calculate v_0 in first place...
  - $v_0 = \text{SubBytes}(p_0 \oplus k_0)$
  - So far, this is not useful for an attack...

- We can model the power consumption of the processing of certain intermediate values!
- But we require knowledge of the key to calculate v_0 in first place...
  - $v_0 = \text{SubBytes}(p_0 \oplus k_0)$
  - So far, this is not useful for an attack...
- Maybe there is a way to test parts of the key using power side-channels...

- Enumerating all 2^{128} possible keys of AES-128?
  - @ 1 billion keys / second $\Rightarrow$ (1 trillion) $\times$ (age of universe)

Preliminaries: Divide and Conquer

- Enumerating all 2^{128} possible keys of AES-128?
  - @ 1 billion keys / second $\Rightarrow$ (1 trillion) $\times$ (age of universe)
- Instead: Recover key parts individually
  - 2^8 possibilities per key byte
  - 16 bytes $\rightarrow$ 4 096 values to test
  - But we can't test just using plain/ciphertexts...

Preliminaries: Divide and Conquer

- Enumerating all 2^{128} possible keys of AES-128?
  - @ 1 billion keys / second $\Rightarrow$ (1 trillion) $\times$ (age of universe)
- Instead: Recover key parts individually
  - 2^8 possibilities per key byte
  - 16 bytes $\rightarrow$ 4 096 values to test
  - But we can't test just using plain/ciphertexts...
- Test them using side-channels!
  - Use information on intermediate values that depend on 1 byte of key!

Differential Power Analysis (DPA) Overview

1. Select target operation in the AES algorithm
    - Dependence on inputs and small number of key bits (e.g. 8-bit subkey)

Differential Power Analysis (DPA) Overview

1. Select target operation in the AES algorithm
   - Dependence on inputs and small number of key bits (e.g. 8-bit subkey)
2. Query device with different inputs and measure power consumption

Differential Power Analysis (DPA) Overview

1. Select target operation in the AES algorithm
   - Dependence on inputs and small number of key bits (e.g. 8-bit subkey)
2. Query device with different inputs and measure power consumption
3. Enumerate all possible values of one subkey
   - $2^8 = 256$ possibilities (hypotheses)

Differential Power Analysis (DPA) Overview

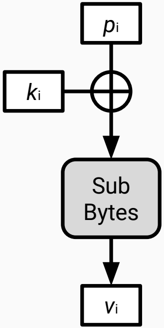
1. Select target operation in the AES algorithm
   - Dependence on inputs and small number of key bits (e.g. 8-bit subkey)
2. Query device with different inputs and measure power consumption
3. Enumerate all possible values of one subkey
   - $2^8 = 256$ possibilities (hypotheses)
4. Predict power consumption of targeted operation based on all inputs and the current key hypothesis
   - Use power model such as Hamming weight

Differential Power Analysis (DPA) Overview

1. Select target operation in the AES algorithm
   - Dependence on inputs and small number of key bits (e.g. 8-bit subkey)
2. Query device with different inputs and measure power consumption
3. Enumerate all possible values of one subkey
   - $2^8 = 256$ possibilities (hypotheses)
4. Predict power consumption of targeted operation based on all inputs and the current key hypothesis
   - Use power model such as Hamming weight
5. Compare predictions with real measurements
   - Key hypothesis that fits best is most likely correct
   - What "fits best"? $\rightarrow$ Correlation!

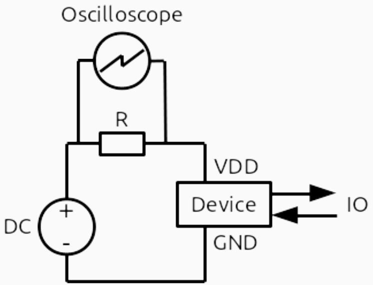
Step 1: Select target operation in the AES algorithm

- Should depend on:
  - Small number of key bits (enumerable, e.g. 8)
  - Known and varying data (plain/ciphertext)
- Common choice is SubBytes output of first round
  - Why not output of AddRoundKey? → later

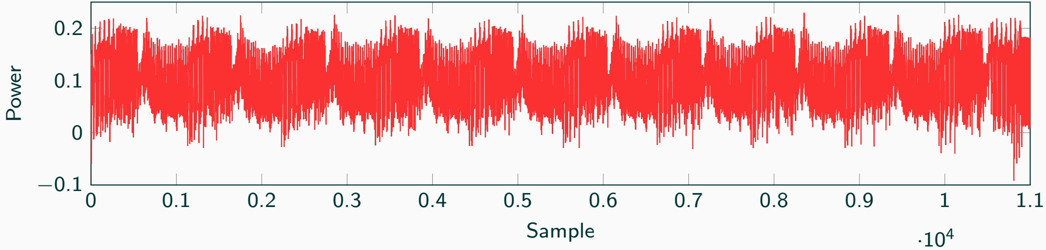


Step 2: Measure Power Consumption

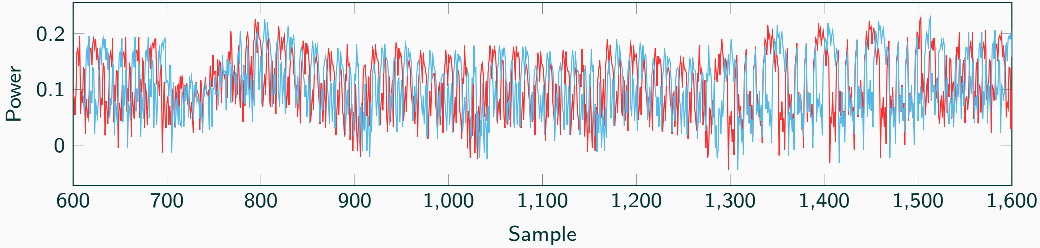
- Query device
- Gather IO plain/ciphertext
- Measure power consumption of en/decryption
  - Voltage over R (shunt resistor) ≈ current
  - Oscilloscope measures voltage
  - At least 1 sample per clock cycle
  - Measurement must include the targeted operation



Step 2: Measure Power Consumption

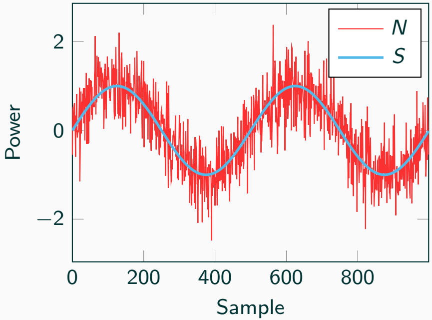


- How to know what part is measured?
  - Visual inspection, trial&error, experience,...

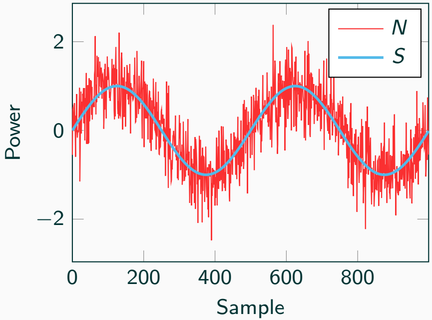


- Traces should be aligned → same operation at same instant in trace
- → Trigger on communication
- → Trigger on trace feature (distinctive pattern)
  (with oscilloscope support or through post-alignment)

- Side-channels are noisy
  1. Exploitable signal S
  2. Noise N

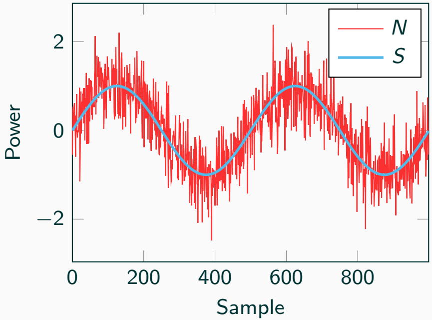


- Side-channels are noisy
  1. Exploitable signal S
  2. Noise N
- Common metric:
  Signal-to-Noise-Ratio $\mathrm{SNR} = \sigma_S^2 / \sigma_N^2$



Step 2: Measure Power Consumption

- Side-channels are noisy
  1. Exploitable signal S
  2. Noise N
- Common metric:
  Signal-to-Noise-Ratio $\text{SNR} = \sigma_S^2/\sigma_N^2$
- Higher SNR $\rightarrow$ Better attacks



- Averaging
    - Run device multiple times with same inputs
    - Average power consumption $\rightarrow$ reduce noise
    - σ_N^2 goes down with 1 / #traces
      (only for electrical / random noise)

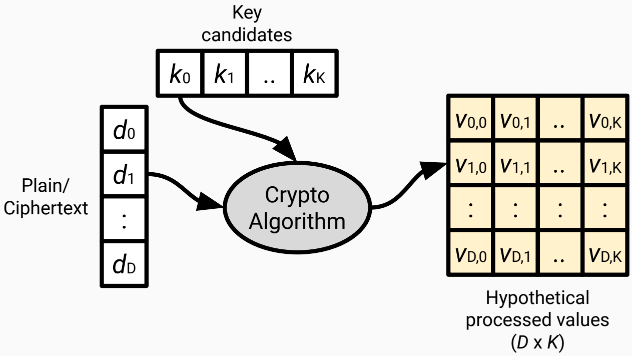
Step 2: Measure Power Consumption

- Averaging
  - Run device multiple times with same inputs
  - Average power consumption $\rightarrow$ reduce noise
  - σ_N^2 goes down with 1 / #traces
    (only for electrical / random noise)
- Filtering
  - Power side-channel is slow, but sampling can be fast
  - Lower frequencies more informative, higher frequencies more noisy
  - Low-pass filtering analog/digital

Step 2: Measure Power Consumption

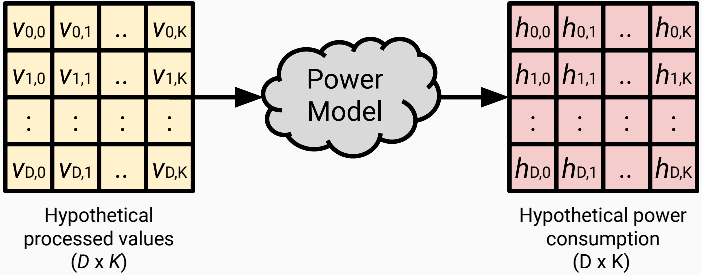
- Averaging
  - Run device multiple times with same inputs
  - Average power consumption $\rightarrow$ reduce noise
  - σ_N^2 goes down with 1 / #traces
    (only for electrical / random noise)
- Filtering
  - Power side-channel is slow, but sampling can be fast
  - Lower frequencies more informative, higher frequencies more noisy
  - Low-pass filtering analog/digital
- Lots of other signal-processing options...

Step 3: Enumerate Subkeys



- D inputs (#measurements)
- K key hypotheses ($K = 2^8$)
- $D \times K$ hypothetical processed values

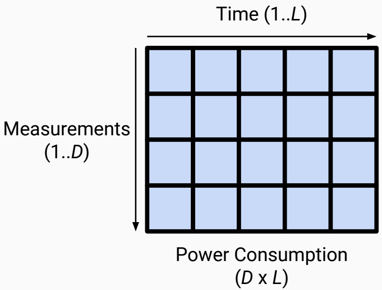
Step 4: Predict Power Consumption



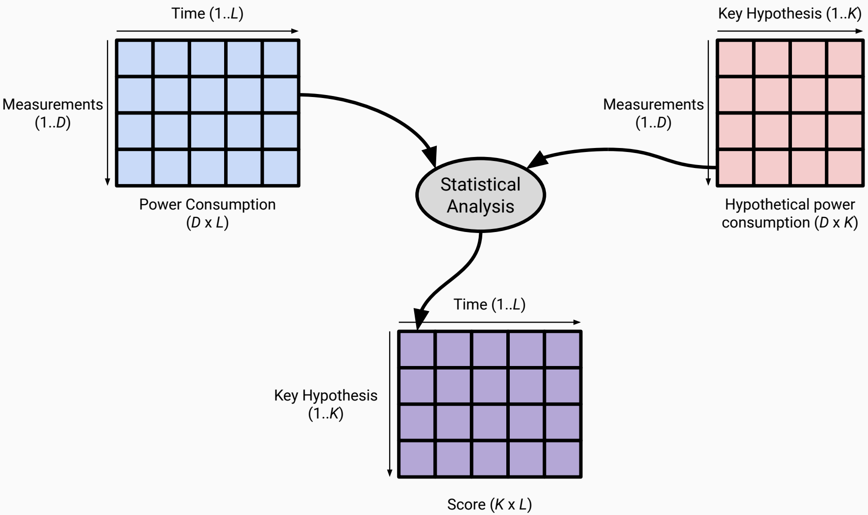
- Common power models
  - Hamming weight (number of set bits)
  - Hamming distance (Hamming weight of XOR difference between two values)

Step 5: Comparison

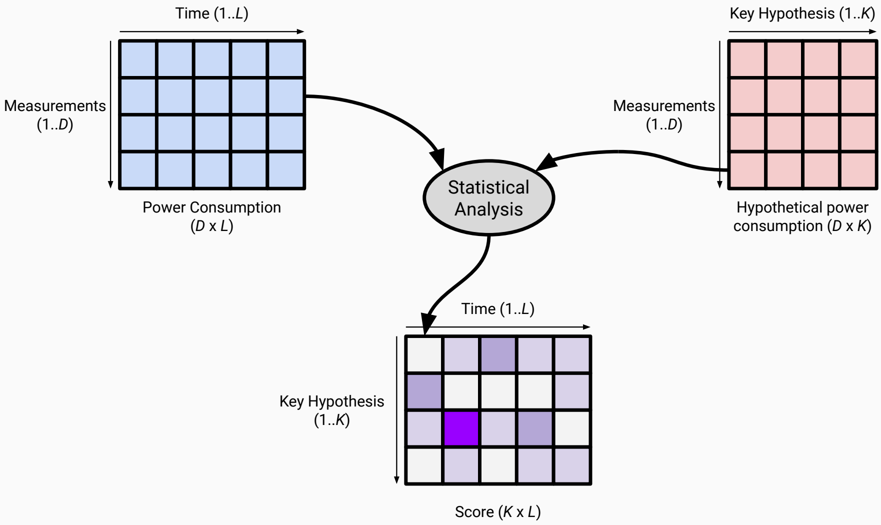
- Trace matrix
  - Each measurement has L samples
- Problems:
  - L can be large
  - We have no idea when targeted operation occurs
- Simply test all locations!



Step 5: Comparison



Step 5: Comparison



Step 5: Comparison

- Statistical Analysis via Pearson Correlation Coefficient ρ
  - Linear relationship between 2 random variables
    (how much do they change together)
  - X: predictions corresponding to one key hypothesis
  - Y: measured samples corresponding to one point in time

$$\rho = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}} = \frac{\mathrm{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_x \sigma_y}$$

Cov = Covariance,
Var = Variance,
E = Expected value,
σ = Standard deviation,
μ = Mean

- Statistical Analysis via Pearson Correlation Coefficient ρ
  - Linear relationship between 2 random variables (how much do they change together)
  - X: predictions corresponding to one key hypothesis
  - Y: measured samples corresponding to one point in time

$$\rho = \frac{\mathrm{Cov}(X, Y)}{\sqrt{\mathrm{Var}(X) \cdot \mathrm{Var}(Y)}} = \frac{\mathrm{E}[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$$

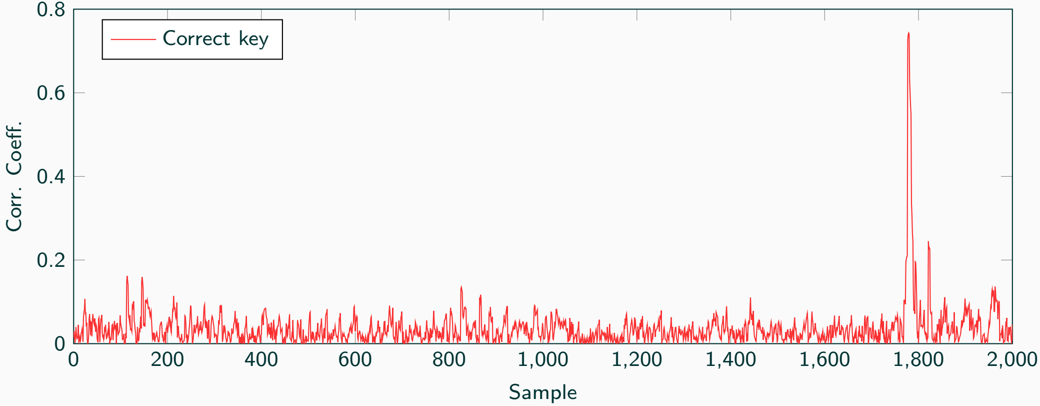
Cov = Covariance,
Var = Variance,
E = Expected value,
σ = Standard deviation,
μ = Mean

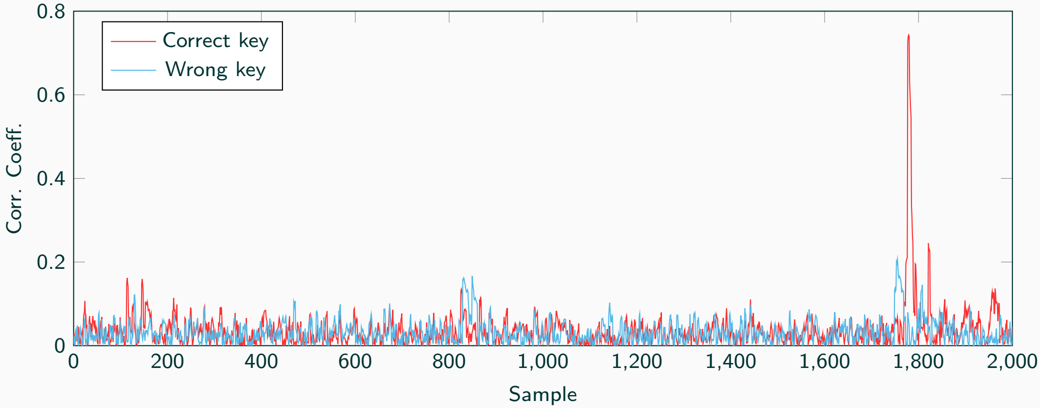
- Estimate:

$$r = \frac{\sum_i (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_i (x_i - \overline{x})^2} \sqrt{\sum_i (y_i - \overline{y})^2}}$$

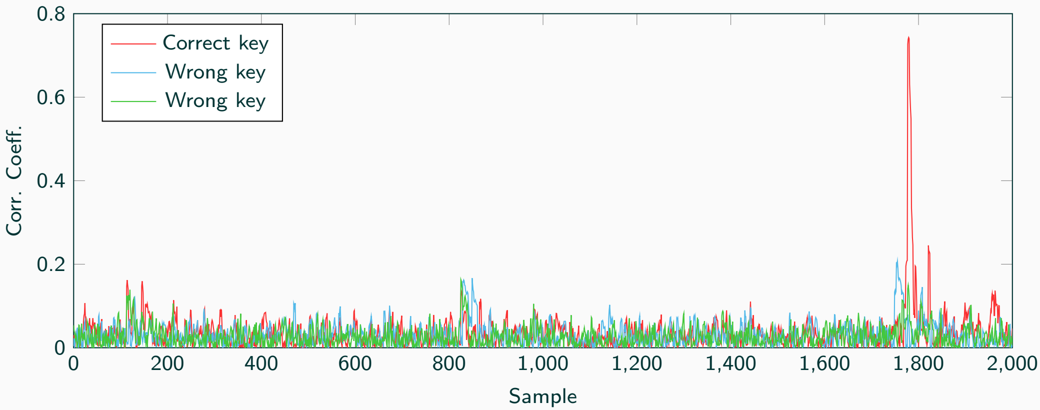
$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

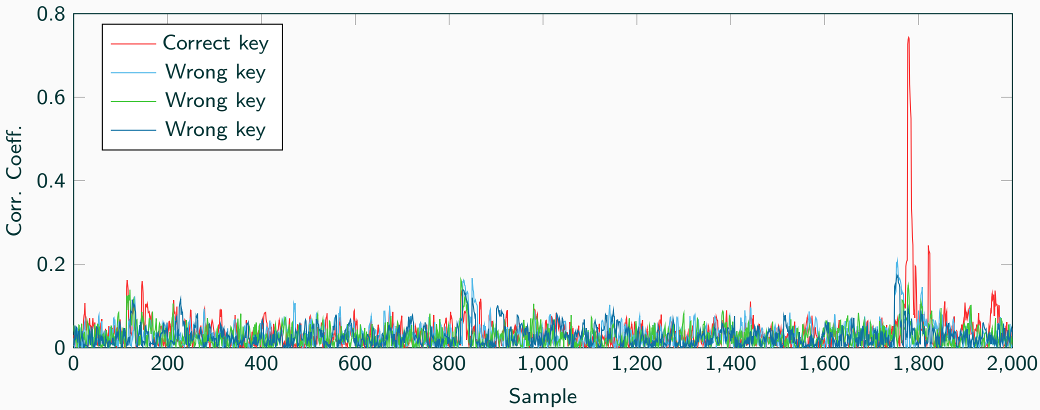
Exemplary Outcome



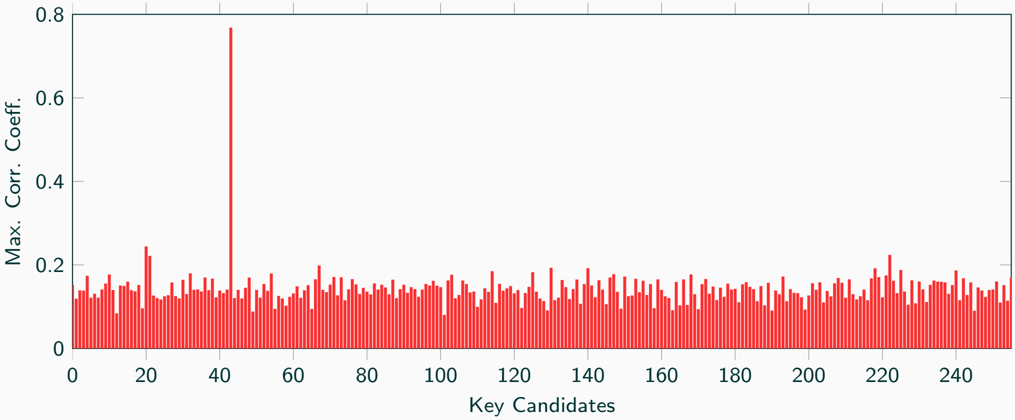

Correct key
Wrong key
0.8
0.6
0.4
0.2
0
Corr. Coeff.
0
200
400
600
800
1,000
1,200
1,400
1,600
1,800
2,000
Sample

Exemplary Outcome




Correct key
Wrong key
Wrong key
Wrong key
Corr. Coeff.
0.8
0.6
0.4
0.2
0
200
400
600
800
1,000
1,200
1,400
1,600
1,800
2,000
Sample

Exemplary Outcome

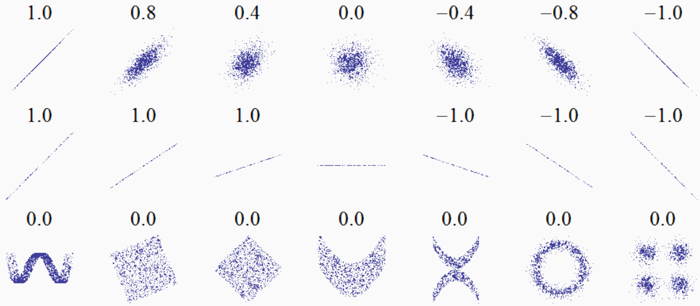


Some Notes on DPA

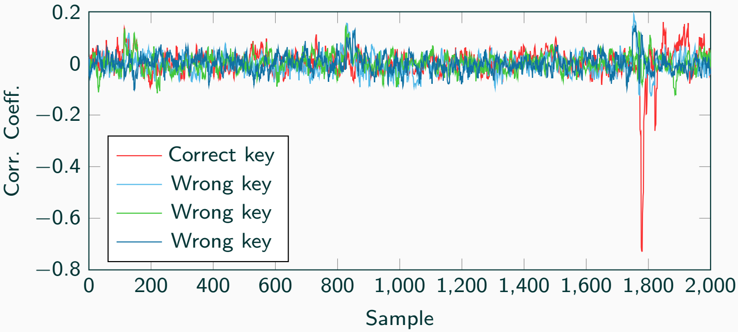
Correlation Coefficient ρ

- $-1 \leq \rho(X, Y) \leq 1$
- If ρ is -1 or 1 then X is a "linearly scaled version" of Y
- Leakage behaves mostly linear
- ρ is simple and converges fast
- If X and Y are independent then $\rho(X, Y) = 0$
    - Not necessarily true in other direction...

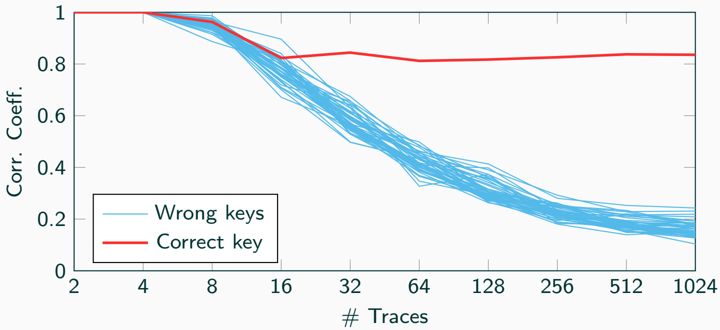
Zero Correlation ≠ Independence



Correlation Coefficient ρ



- We care about the *absolute* correlation coefficient
  $\rightarrow$ Strong negative correlation is also good!



- Estimating value of ρ requires certain amount of traces
  - Wrong keys approach 0, correct key the real ρ_c
  - Intuitive: The lower ρ_c the more traces are required

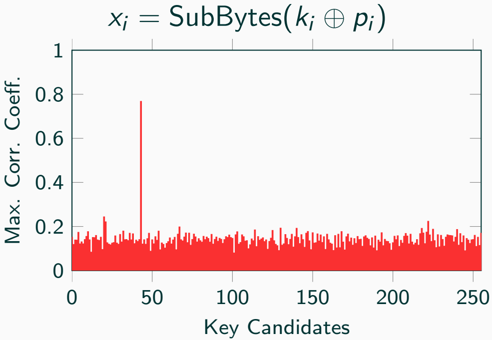
Estimation of ρ

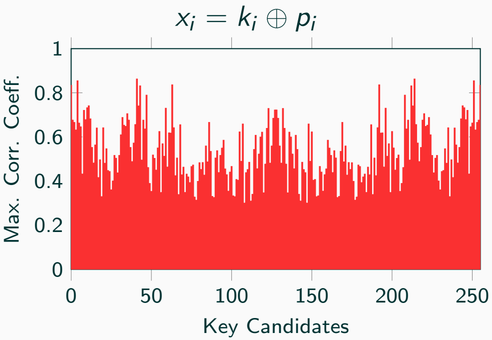
- There exists some fancy maths to determine #traces
  - E.g., based on SNR
  - Which we will not go into now...

Estimation of ρ

- There exists some fancy maths to determine #traces
  - E.g., based on SNR
  - Which we will not go into now...
- Simple rules for #traces
  - Inversely quadratic in ρ_c : $\rho_c/2 \rightarrow \#\text{traces} \times 4$
  - Linear in noise: Noise variance $\times\ 2 \rightarrow \#\text{traces} \times 2$

On the Targeted Operation

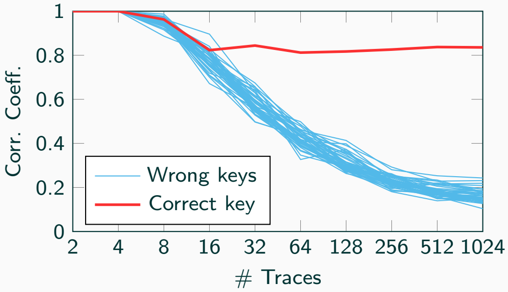




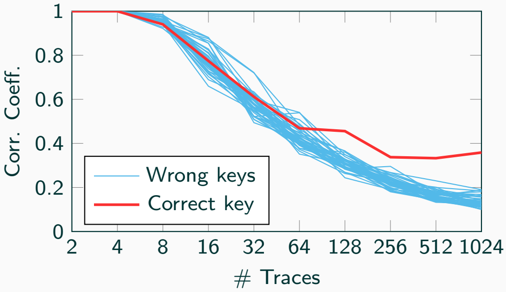
- Intuition: "Similar" keys have similar $x_i = k_i \oplus p_i$
  - Change 1 bit in $k_i \rightarrow$ HW only changes by 1
  - Flip all bits in $k_i \rightarrow$ Correlation in other direction
- SubBytes: Changing 1 input bit affects all output bits in non-linear way

On the Power Model

$h_i = \mathrm{HW}(\mathrm{SubBytes}(k_i \oplus p_i))$



$h_i = \mathrm{LSB}(\mathrm{SubBytes}(k_i \oplus p_i))$



- Choose power model that describes reality best
- Higher correlation $\rightarrow$ fewer traces

DPA: Recap

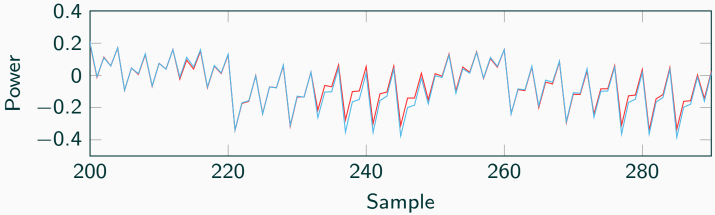
- Requires little assumptions...
  - On the attacked device (power models)
  - On the concrete implementation (when does it leak?)
  - Yet still effective
- But there are also downsides
  - Simplifications that affect performance
  - Not applicable to single traces or multiple traces with constant input
  - Only target operations that depend on few key bits

(Rough) Taxonomy

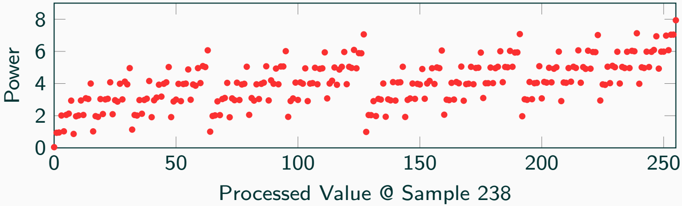
	Non-profiled Attacks	Profiled Attacks
One or few observations with fixed data	Simple SCA	**Profiled simple SCA**
Many observations with varying data	Differential SCA	Profiled differential SCA

- Characterize (profile) power consumption of target device

Profiled Attacks



Power
0.4
0.2
0
−0.2
−0.4
200
220
240
260
280
Sample



Power
8
6
4
2
0
0
50
100
150
200
250
Processed Value @ Sample 238

Why Profiling?

- DPA uses simplifications → not all information in trace is exploited
  - Power models, leakage on single point in time, etc.
  - Profiled attacks are more powerful → worst-case security evaluations

Why Profiling?

- DPA uses simplifications $\rightarrow$ not all information in trace is exploited
  - Power models, leakage on single point in time, etc.
  - Profiled attacks are more powerful $\rightarrow$ worst-case security evaluations
- DPA cannot be run on a single measurement
  - "Differential" information between traces

Why Profiling?

- DPA uses simplifications → not all information in trace is exploited
  - Power models, leakage on single point in time, etc.
  - Profiled attacks are more powerful → worst-case security evaluations
- DPA cannot be run on a single measurement
  - "Differential" information between traces
- DPA requires prediction of values
  - In some scenarios not possible (unknown or low amount inputs/outputs)

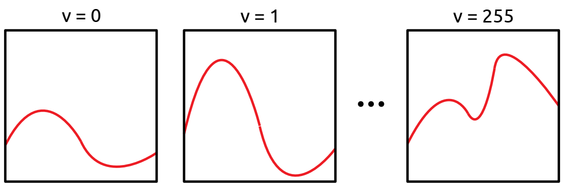
Why Profiling?

- DPA uses simplifications → not all information in trace is exploited
  - Power models, leakage on single point in time, etc.
  - Profiled attacks are more powerful → worst-case security evaluations
- DPA cannot be run on a single measurement
  - "Differential" information between traces
- DPA requires prediction of values
  - In some scenarios not possible (unknown or low amount inputs/outputs)
- Downsides
  - Assumes attacker has access to same or similar device
  - Can run it with known inputs (including key)
  - Many profiling traces needed

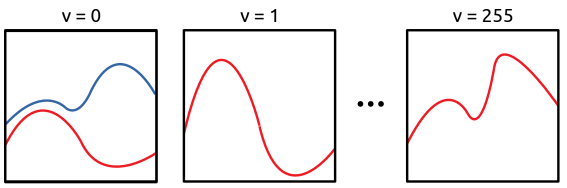
1. Pick an operation (e.g. `MOV`)

Basic Steps of a Profiled Attack

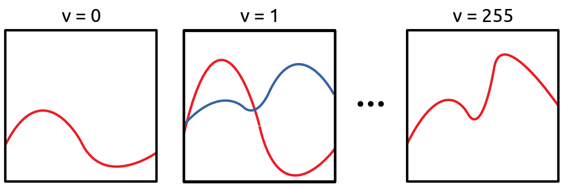
1. Pick an operation (e.g. `MOV`)
2. Characterize leakage
   - Profile power consumption for each possible processed value v
   - Record traces with all inputs known, group according to v
   - We call a profile a "template"



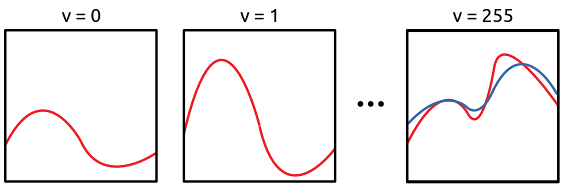
3. Attack phase
   - Compare (match) measured traces to all templates
   - Use v which best fits, process probabilities...



3. Attack phase
    - Compare (match) measured traces to all templates
    - Use v which best fits, process probabilities...



3. Attack phase
    - Compare (match) measured traces to all templates
    - Use v which best fits, process probabilities...



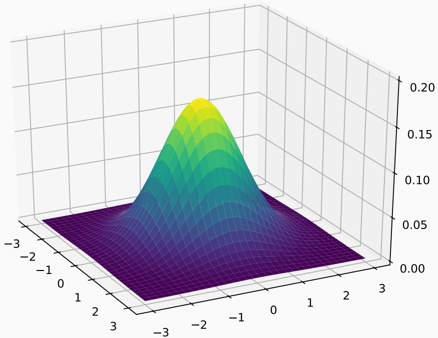
- Profiling: Estimation/learning of a Probability Density Function (PDF)
  - For each v and each trace t: estimate $P(T = t|v)$

Templates and Probability Theory

- Profiling: Estimation/learning of a Probability Density Function (PDF)
  - For each v and each trace t: estimate $P(T = t|v)$
- Attack: Evaluation of PDFs at measured samples
  - Record trace t_a
  - Compute $P(T = t_a|v)$ for each v (probability that t is measured given v)

Templates and Probability Theory

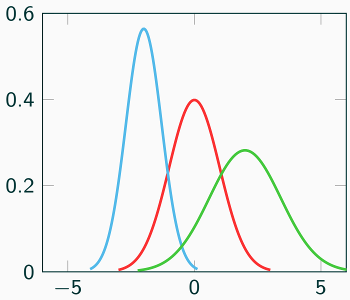
- Profiling: Estimation/learning of a Probability Density Function (PDF)
  - For each v and each trace t: estimate $P(T = t|v)$
- Attack: Evaluation of PDFs at measured samples
  - Record trace t_a
  - Compute $P(T = t_a|v)$ for each v (probability that t is measured given v)
- **Aka: Machine Learning**



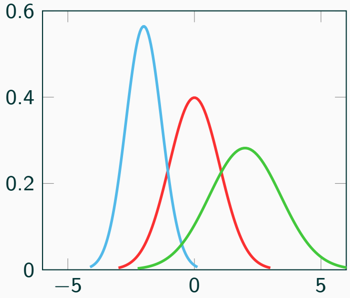
- We still need some assumptions to allow efficient profiling
- Good approximation:
  Traces follow (multivariate) Gaussian (i.e., normal) distribution $\mathcal{N}$

Univariate Case

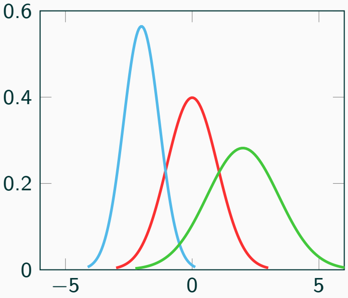
- PDF characterized by
  - Mean μ
  - Std. dev. σ, variance σ^2



- PDF characterized by
  - Mean μ
  - Std. dev. σ, variance σ^2
- For each possible value v estimate:
  - Means $\mu_0, \mu_1, \ldots, \mu_v$
  - Std. dev. $\sigma_0, \sigma_1, \ldots, \sigma_v$



- PDF characterized by
  - Mean μ
  - Std. dev. σ, variance σ^2
- For each possible value v estimate:
  - Means $\mu_0, \mu_1, \ldots, \mu_v$
  - Std. dev. $\sigma_0, \sigma_1, \ldots, \sigma_v$
- Intuitive interpretation
  - μ_i = true power consumption of data i
  - σ_i = noise



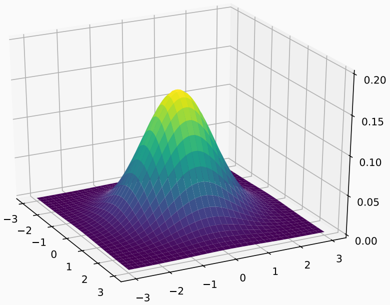
$$f(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-(x-\mu)^2/2\sigma^2}$$

Estimation: $\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2$

Multivariate Case

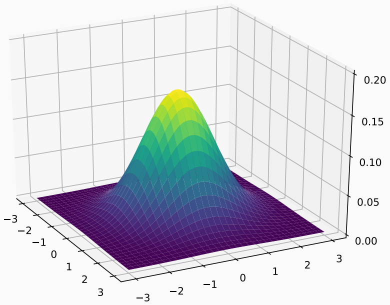
- Considers multiple samples
- PDF characterized by:
  - Mean vector $\mathbf{m} = (m_1, m_2, \ldots)^\mathsf{T}$
  - Cov matrix $\mathbf{C} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots \\ c_{2,1} & c_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$

Bivariate Gaussian



- Considers multiple samples
- PDF characterized by:
  - Mean vector $\mathbf{m} = (m_1, m_2, \dots)^\mathsf{T}$
  - Cov matrix $\mathbf{C} = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots \\ c_{2,1} & c_{2,2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$
- Again for each possible value:
  - Means $\mathbf{m}_0, \mathbf{m}_1, \dots, \mathbf{m}_v$
  - Cov Matrixes $\mathbf{C}_0, \mathbf{C}_1, \dots, \mathbf{C}_v$

Bivariate Gaussian



$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\mathbf{C})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^\mathsf{T} \cdot \mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

- Can't model the entire trace as multivariate Gaussian
  - **C** is $(L \times L)$ matrix ...
  - **C** tends to be badly conditioned (it is close to being singular)
    $\rightarrow$ numerical problems with matrix inversions

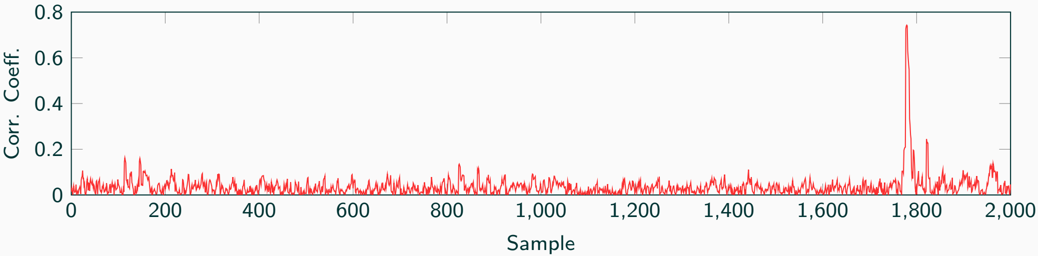
Practical Limitations in Template Attacks

- Can't model the entire trace as multivariate Gaussian
  - **C** is $(L \times L)$ matrix ...
  - **C** tends to be badly conditioned (it is close to being singular)
    $\rightarrow$ numerical problems with matrix inversions
- Solution 1: Dimensionality reduction
  - Generic techniques such as Principal Component Analysis (PCA)
  - Selecting a subset of samples: Points-Of-Interest (POI)

Practical Limitations in Template Attacks

- Can't model the entire trace as multivariate Gaussian
  - **C** is $(L \times L)$ matrix ...
  - **C** tends to be badly conditioned (it is close to being singular)
    $\rightarrow$ numerical problems with matrix inversions
- Solution 1: Dimensionality reduction
  - Generic techniques such as Principal Component Analysis (PCA)
  - Selecting a subset of samples: Points-Of-Interest (POI)
- Solution 2: Reduced templates
  - Assume "independent" samples

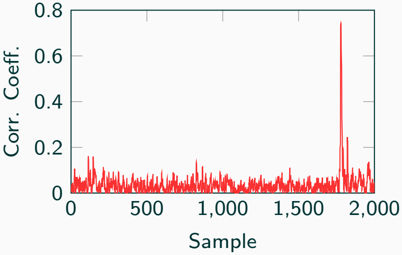
Selecting Points of Interest (POI)



- Only small set of samples has information about v
  - As seen during DPA
- "Feature Selection" in Machine Learning

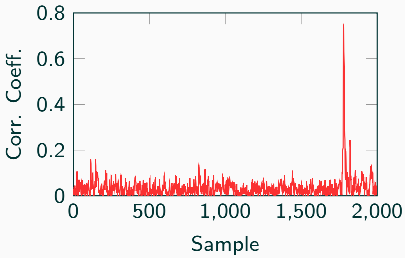
Selecting Points of Interest (POI)

1. Use points of highest correlation
    - Does not capture non-HW leakages
    - Does not capture leakages for $f(v)$



Selecting Points of Interest (POI)

1. Use points of highest correlation
   - Does not capture non-HW leakages
   - Does not capture leakages for $f(v)$



2. Welch t-test
   - Statistical test if two populations have same mean
   - Use points where means significantly differ

$$\frac{m_i - m_j}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}}$$

Selecting POIs using t-tests

- Create multiple groups of traces corresponding to different cipher inputs
  - Each group consists of the same amount of traces
  - E.g.: 2 groups: random inputs, some fixed input
  - E.g.: 256 groups: $\mathtt{0x0000}\ldots$, $\mathtt{0x0100}\ldots$, $\mathtt{0x0200}\ldots$, $\ldots$
- For each group of traces, and each point in time, pre-compute mean m and std. dev. σ

Selecting POIs using t-tests

- Create multiple groups of traces corresponding to different cipher inputs
  - Each group consists of the same amount of traces
  - E.g.: 2 groups: random inputs, some fixed input
  - E.g.: 256 groups: `0x0000...`, `0x0100...`, `0x0200...`, ...
- For each group of traces, and each point in time, pre-compute mean m and std. dev. σ
- For each point in time:
  - Perform pair-wise t-tests between all groups
  - Sum up the squares of t-scores → SOST
  - If you only use 2 groups this boils down to the t-test formula from before

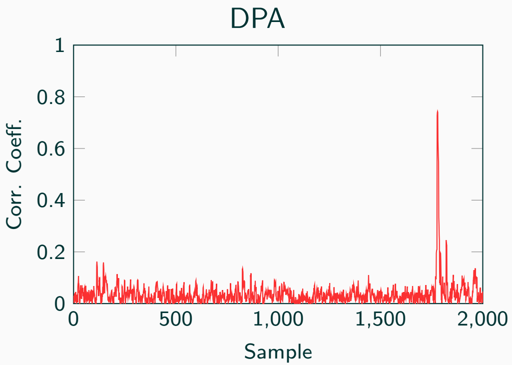
$$\sum_{i,j=1}^{\#\text{groups}} \left(\frac{m_i - m_j}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}} \right)^2 \quad \text{for} \quad i \geq j$$

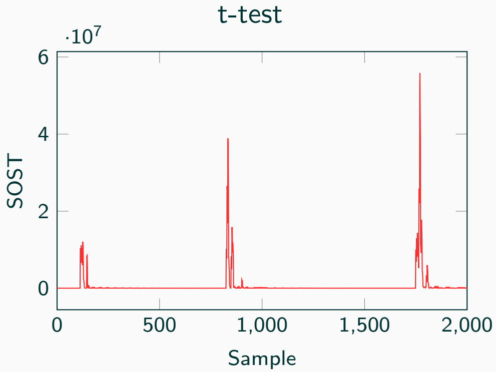
Selecting POIs using t-tests

- Create multiple groups of traces corresponding to different cipher inputs
  - Each group consists of the same amount of traces
  - E.g.: 2 groups: random inputs, some fixed input
  - E.g.: 256 groups: `0x0000...`, `0x0100...`, `0x0200...`, ...
- For each group of traces, and each point in time, pre-compute mean m and std. dev. σ
- For each point in time:
  - Perform pair-wise t-tests between all groups
  - Sum up the squares of t-scores $\rightarrow$ SOST
  - If you only use 2 groups this boils down to the t-test formula from before

$$\sum_{i,j=1}^{\#\text{groups}} \left(\frac{m_i - m_j}{\sqrt{\frac{\sigma_i^2}{n_i} + \frac{\sigma_j^2}{n_j}}} \right)^2 \quad \text{for} \quad i \geq j$$

- Captures all first-moment (mean) leakage
- Automatically captures f(v)

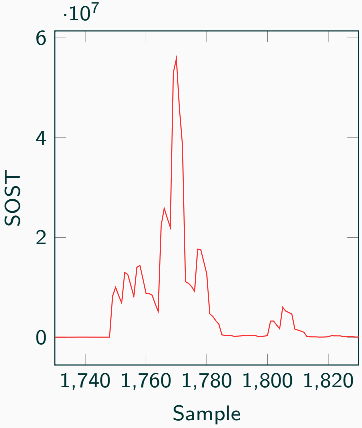




- t-test also captures key addition $= f(v)$
- t-test has similar peaks, but different relative heights

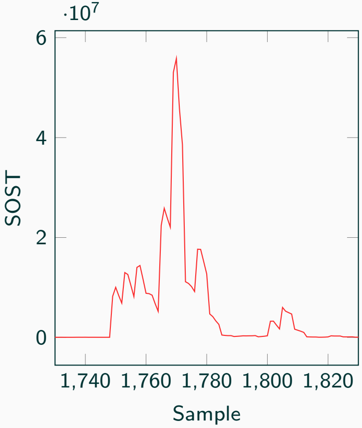
A Word of Caution

- Don't select too many points!



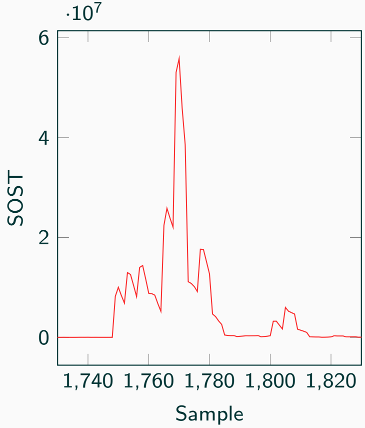
A Word of Caution

- Don't select too many points!
- Don't use points close to each other
  - High linear dependency
    $\rightarrow$ badly conditioned **C**



A Word of Caution

- Don't select too many points!
- Don't use points close to each other
  - High linear dependency
    $\rightarrow$ badly conditioned **C**
- Some simple guides for power SCA
  - Only 1 point per clock cycle (power is slow)
  - Only use distinctive peaks of t-score



Further Simplifications

- Reduced Templates
  - Assume samples are linearly independent $\rightarrow$ all covariances are 0
  - Normalize traces: Divide by σ at each point in time $\rightarrow \sigma = 1$ at all times $\rightarrow$ **C** becomes identity matrix **I**
  - Reduces complexity of profiling and attacking
  - But somewhat worse performance (or complete failure)

Further Simplifications

- Reduced Templates
  - Assume samples are linearly independent $\rightarrow$ all covariances are 0
  - Normalize traces: Divide by σ at each point in time $\rightarrow \sigma = 1$ at all times $\rightarrow$ **C** becomes identity matrix **I**
  - Reduces complexity of profiling and attacking
  - But somewhat worse performance (or complete failure)
- Combine templates with power models
  - Build templates for Hamming weights instead of values
  - e.g., 9 instead of 256 templates

Attack Phase

- Goal: Evaluate PDFs at observed power

- Goal: Evaluate PDFs at observed power
- Evaluate Gaussian, where $\mathbf{x}$ is the power, for each template:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\mathbf{C})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^\mathsf{T} \cdot \mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

- Goal: Evaluate PDFs at observed power
- Evaluate Gaussian, where $\mathbf{x}$ is the power, for each template:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\mathbf{C})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^\mathsf{T} \cdot \mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

- Receive $p(t|v_i)$ for $i = 1 \ldots V \rightarrow$ likelihood

- Goal: Evaluate PDFs at observed power
- Evaluate Gaussian, where $\mathbf{x}$ is the power, for each template:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot \det(\mathbf{C})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{m})^\mathsf{T} \cdot \mathbf{C}^{-1}(\mathbf{x}-\mathbf{m})\right)$$

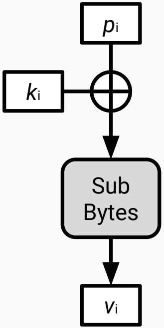
- Receive $p(t|v_i)$ for $i = 1 \dots V \rightarrow$ likelihood
- Alternatively compute $\ln(p(t|v_i))$, i.e., the log-likelihood:

$$\ln p(\mathbf{t}|(v,k)) = -\frac{1}{2}\left(\ln((2\pi)^n \cdot \det(\mathbf{C})) + (\mathbf{t}-\mathbf{m})^\mathsf{T} \cdot \mathbf{C}^{-1} \cdot (\mathbf{t}-\mathbf{m})\right)$$

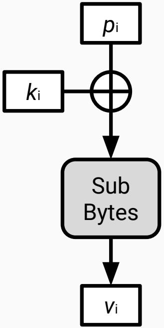
- v_i with highest likelihood = most likely value
- Reduced templates: minimal $||(\mathbf{x} - \mathbf{m})||^2$ = most likely value
  vector norm: $||\mathbf{x}||^2 = x_1^2 + x_2^2 + x_3^2 + \dots$

Attack Phase

- v_i with highest likelihood = most likely value
- Reduced templates: minimal $||(\mathbf{x} - \mathbf{m})||^2$ = most likely value
  vector norm: $||\mathbf{x}||^2 = x_1^2 + x_2^2 + x_3^2 + \ldots$
- From v to k
  - p is known
  - Each possible value of k $\rightarrow$ exactly one value of v
  - $p(t|k) = p(t|v = \text{SubBytes}(k \oplus p))$



- v_i with highest likelihood = most likely value
- Reduced templates: minimal $||(\mathbf{x}-\mathbf{m})||^2$ = most likely value
  vector norm: $||\mathbf{x}||^2 = x_1^2 + x_2^2 + x_3^2 + \dots$
- From v to k
  - p is known
  - Each possible value of k $\rightarrow$ exactly one value of v
  - $p(t|k) = p(t|v = \text{SubBytes}(k \oplus p))$
- Caution: Likelihood $\neq$ probability
  - We might want $p(v_i|t)$ or $p(k_i|t)$



$p(k|t)$ with Bayes Theorem

- Bayes: Update probabilities of A given new observation B
- General form
  - $P(A|B)$ = posterior probability
  - $P(B|A)$ = likelihood
  - $P(A)$ = prior probability

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

$p(k|t)$ with Bayes Theorem

- Bayes: Update probabilities of A given new observation B
- General form
  - $P(A|B)$ = posterior probability
  - $P(B|A)$ = likelihood
  - $P(A)$ = prior probability

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- In our case:
  - $p(t|k_j)$ = likelihood from before
  - $p(k_j)$ = prior (uniform)
  - denom = "normalization"

$$p(k_j|\mathbf{t}'_i) = \frac{p(\mathbf{t}'_i|k_j) \cdot p(k_j)}{\sum_{l=1}^{K}(p(\mathbf{t}'_i|k_l) \cdot p(k_l))}$$

(Rough) Taxonomy

	Non-profiled Attacks	**Profiled Attacks**
One or few observations with fixed data	Simple SCA	Profiled simple SCA
Many observations with varying data	Differential SCA	**Profiled differential SCA**

- Thus far we used a single attack trace
- Extension to multi-trace setting: Bayesian Updating
    - Update beliefs given new information
    - Use Bayes theorem iteratively
    - Posterior after previous trace $=$ prior for next trace
    - Update key probabilities for each new trace

$$p(k_j|\mathbf{t}'_i) = \frac{p(\mathbf{t}'_i|k_j) \cdot p(k_j)}{\sum_{l=1}^{K}(p(\mathbf{t}'_i|k_l) \cdot p(k_l))}$$

Closed Form Expression

$$p(k_j|\mathbf{t}_i') = \frac{p(\mathbf{t}_i'|k_j)\cdot p(k_j)}{\sum_{l=1}^{K}(p(\mathbf{t}_i'|k_l)\cdot p(k_l))} \implies p(k_j|\mathbf{T}) = \frac{\left(\prod_{i=1}^{D} p(\mathbf{t}_i'|k_j)\right)\cdot p(k_j)}{\sum_{l=1}^{K}\left(\left(\prod_{i=1}^{D} p(\mathbf{t}_i'|k_l)\right)\cdot p(k_l)\right)}$$

Closed Form Expression

$$p(k_j|\mathbf{t}'_i) = \frac{p(\mathbf{t}'_i|k_j) \cdot p(k_j)}{\sum_{l=1}^{K}(p(\mathbf{t}'_i|k_l) \cdot p(k_l))} \implies p(k_j|\mathbf{T}) = \frac{\left(\prod_{i=1}^{D} p(\mathbf{t}'_i|k_j)\right) \cdot p(k_j)}{\sum_{l=1}^{K}\left(\left(\prod_{i=1}^{D} p(\mathbf{t}'_i|k_l)\right) \cdot p(k_l)\right)}$$

- Caution: numerical problems
  - Use log-likelihood
  - Or do iterative updates

Reduced Template-Based “DPA”

- For reduced templates ($\mathbf{C} = \mathbf{I}$) simplification possible
    - Determine most likely key using least-square test
    - Single trace: Most likely key $\rightarrow$ minimal $||(\mathbf{x} - \mathbf{m})||^2$
      vector norm: $||\mathbf{x}||^2 = x_1^2 + x_2^2 + x_3^2 + \ldots$
    - Multiple traces: Minimal sum $||(\mathbf{x} - \mathbf{m})||^2$ over all traces

Questions?