Introduction to Side Channel Attacks

through Power Analysis

FdLSifu @ Ph0wn 2024

Who am I?

ID: Nabil Hamzi - FdLSifu

Hobbies

- CTF (soudure au beurre), Football player & coach, cryptography, security, programming languages

Exp

- Embedded crypto developer @ Gemalto (now Thales) [8 years]
- Crypto and Payment expert @ Ingenico [4 years]
- Now Head of Product Security since 5 years

Feel free to ask any questions!

What will not be covered?

- Cryptography internals
- Hardware, Signal processing
- Power acquisition, Oscilloscopes
- EM, Fault injection
- High order attacks/masking
- Statistical tools
- Deep dive into counter measures
- Secure elements, Certification

What will be covered?

- Leakage model
- Hamming weight
- Simple Power Analysis
- Differential Power Analysis
- Correlation Power Analysis
- AES internals
- RSA modular exponentiation
- Key recovery techniques

Agenda

Power Analysis

- Grey box attack model
- Leakage model
  - Hamming weight

Simple Power Analysis

- RSA and modular exponentiation
- Hands on

Differential Power Analysis

- AES internals
- Hands on

Correlation Power Analysis

- Hands on

Set up for the workshop

1. Install `docker` and `docker-compose`

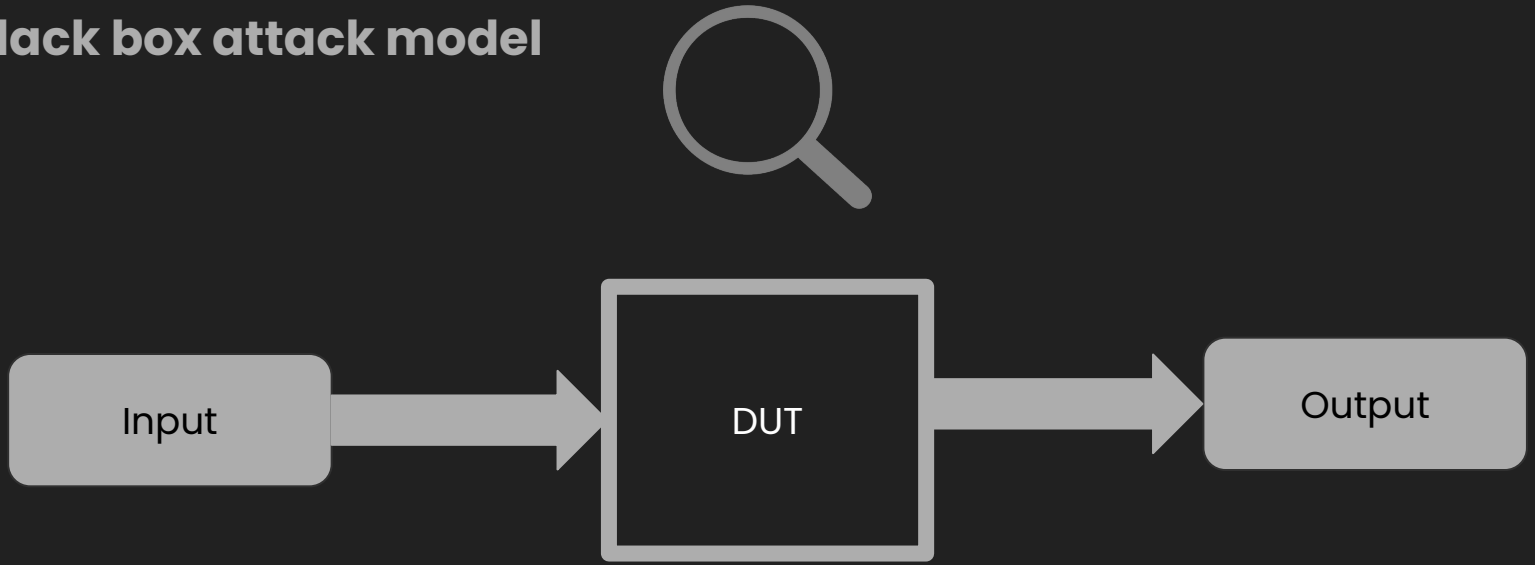
2. Clone this git repo:
   https://github.com/FdLSifu/2024_ph0wn_intro_sca

3. Open terminal and set as current directory the cloned repo
   a. Recommended: Review `launch_workshop.sh`
4. Run: `$ sh launch_workshop.sh rsa`

You should have a prompt! ➜ ~ _

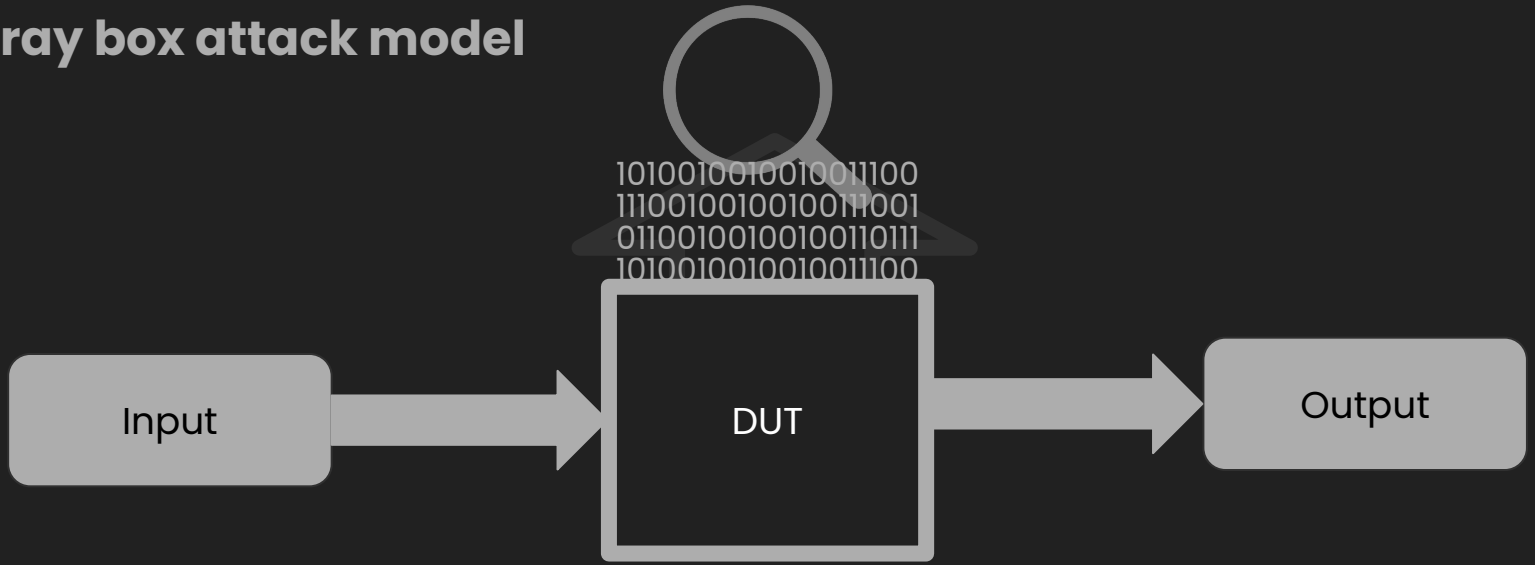
Power Analysis

Black box attack model



Power Analysis

Gray box attack model



Power Analysis

What can be observed?

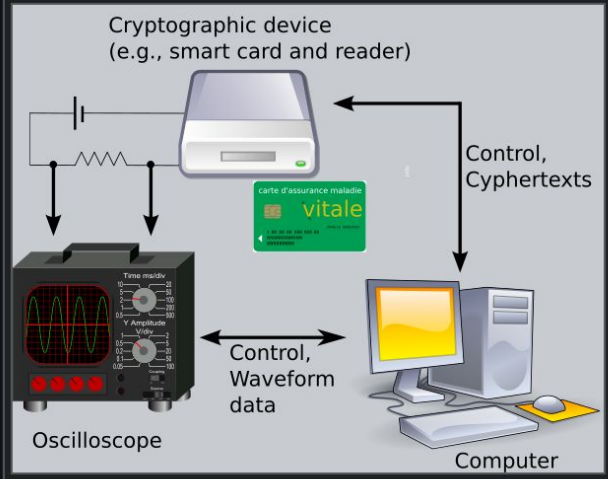
- Power consumption
- Electromagnetic emanation
- Radio signals
- Temperature change
- Timing

...

Power Analysis

Power analysis is a form of side channel attack in which the attacker studies the power consumption of a cryptographic hardware device. These attacks rely on basic physical properties of the device: semiconductor devices are governed by the laws of physics, which dictate that changes in voltages within the device require very small movements of electric charges (currents). By measuring those currents, it is possible to learn a small amount of information about the data being manipulated.

Simple power analysis (**SPA**) involves visually interpreting power *traces*, or graphs of electrical activity over time. **Differential power analysis** (**DPA**) is a more advanced form of power analysis, which can allow an attacker to compute the intermediate values within cryptographic computations through statistical analysis of data collected from multiple cryptographic operations. SPA and DPA were introduced to the open cryptography community in 1998 by Paul Kocher, Joshua Jaffe and Benjamin Jun.[1]



A diagram of differential power analysis.

Power Analysis

Leakage model

Hamming weight

=> Sum of the bits set

Hamming distance

=> Difference of the bits set between 2 bytes/words

Time

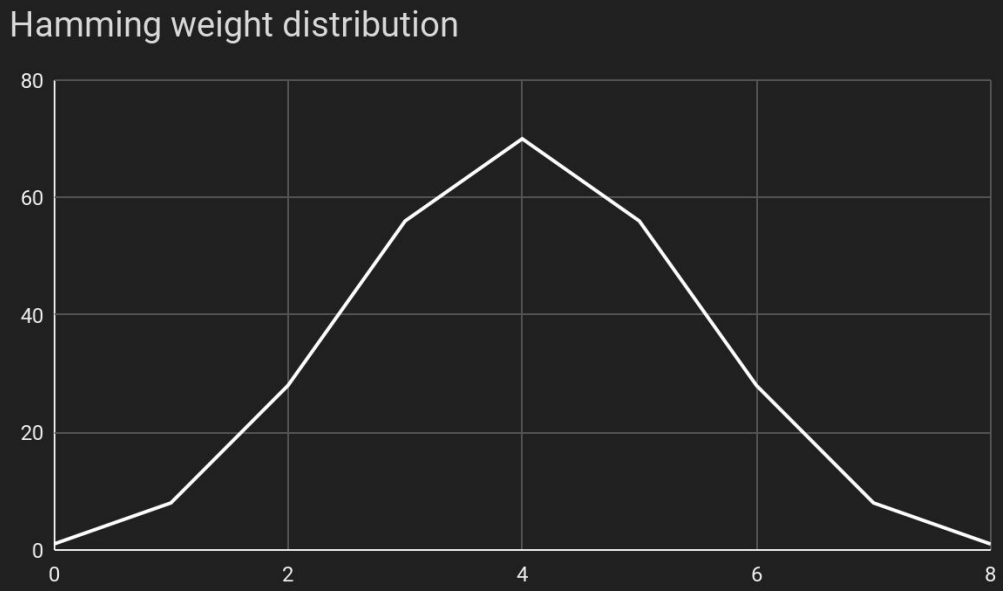
=> Difference timing analysis

Value

=> Value of the byte/word

Power Analysis

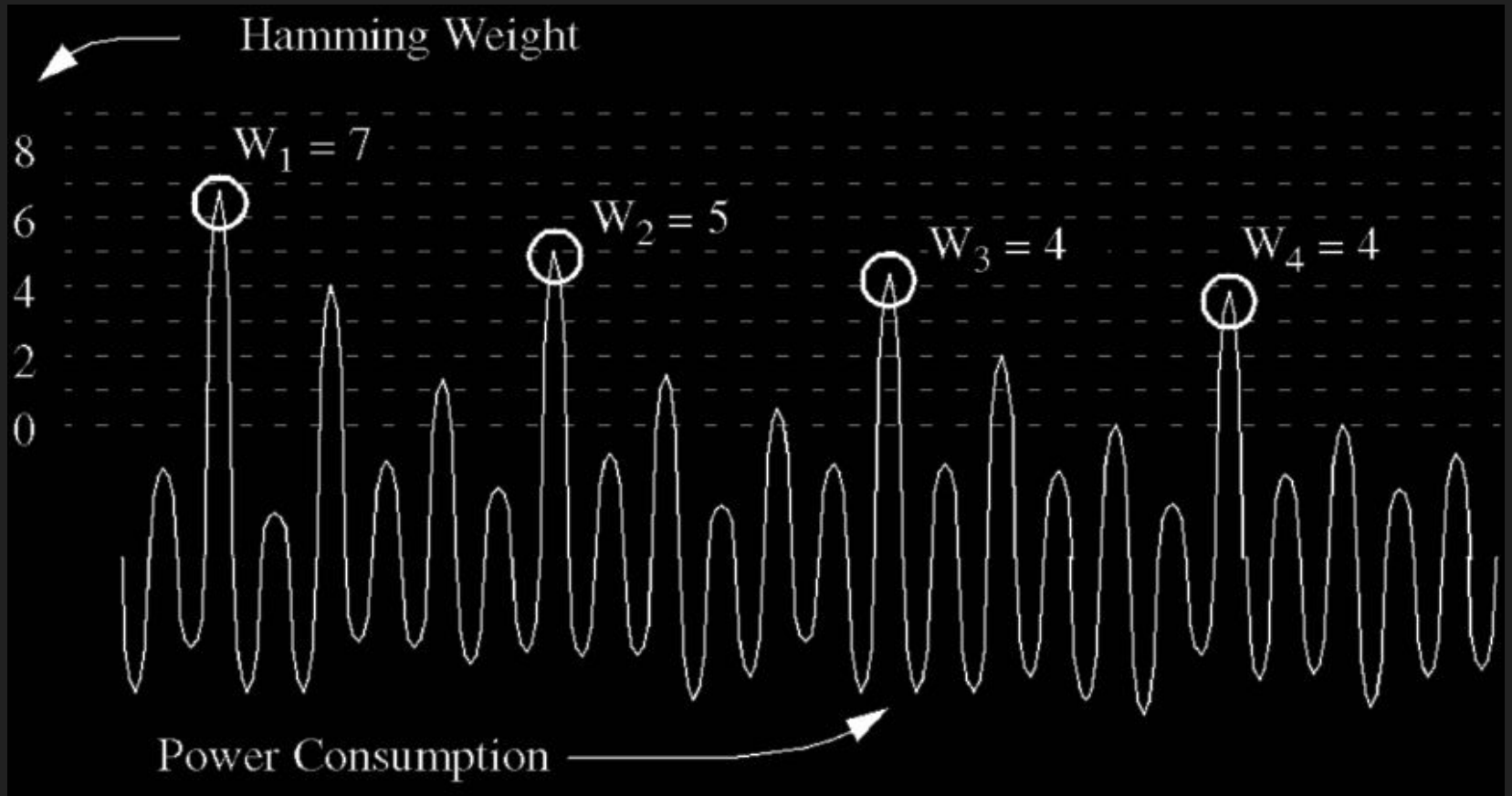
Hamming weight



56 = 0x38 = 0011 1000

HW(56) = 3

Power Analysis



Simple Power Analysis

Differential timing analysis

Code PIN: x000 => 5us

Code PIN: 1000 => 6us

Code PIN: 12x0 => 7us

Code PIN: 1230 => 8us

Code PIN 1x00 => 6us

Code PIN 1200 => 7us

Code PIN 123x => 8us

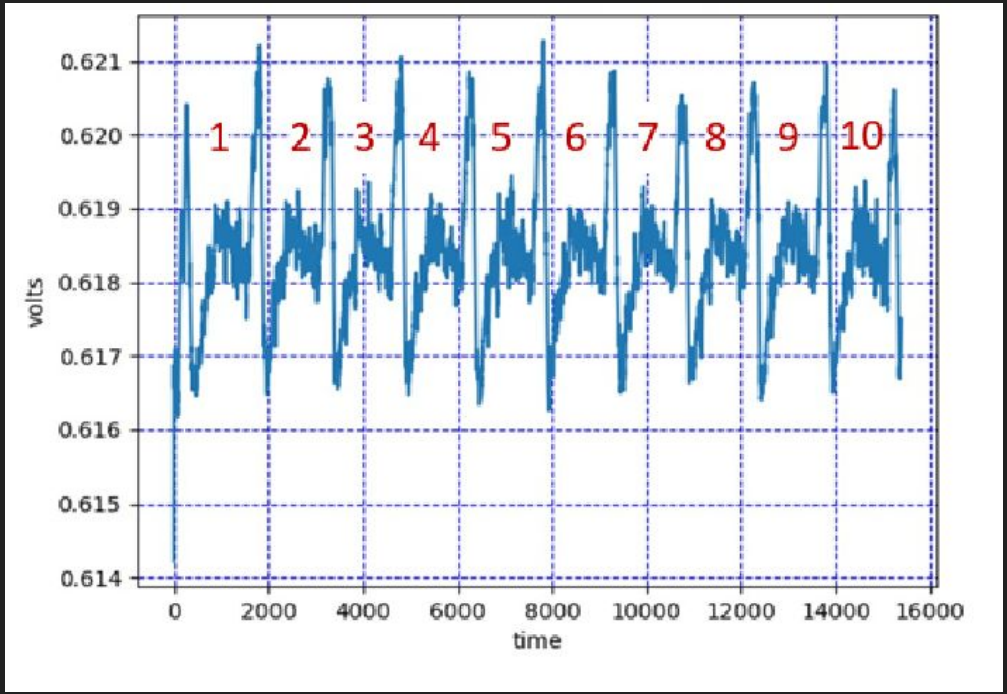
Code PIN 1234 => 9us => UNLOCK

40 attempts!!

Simple Power Analysis

Simple Power Analysis

Power consumption leaks information



Simple Power Analysis

RSA

Public Key: (n, e)

Private Key: (d) or (dp, dq, p, q, iq)

Encryption/Decryption

$c = m^e$ / $m = c^d$

Signature/Verification

$s = hash(m)^d$ / $hash(m) == s^e$

Simple Power Analysis

RSA

Public Key: (n, e)

Private Key: (d) or (dp, dq, p, q, iq)

Encryption/Decryption

$c = m^e$ / $m = c^d$

Signature/Verification

$s = hash(m)^d$ / $hash(m) == s^e$

How a modular exponentiation is performed?

Simple Power Analysis

Modular exponentiation

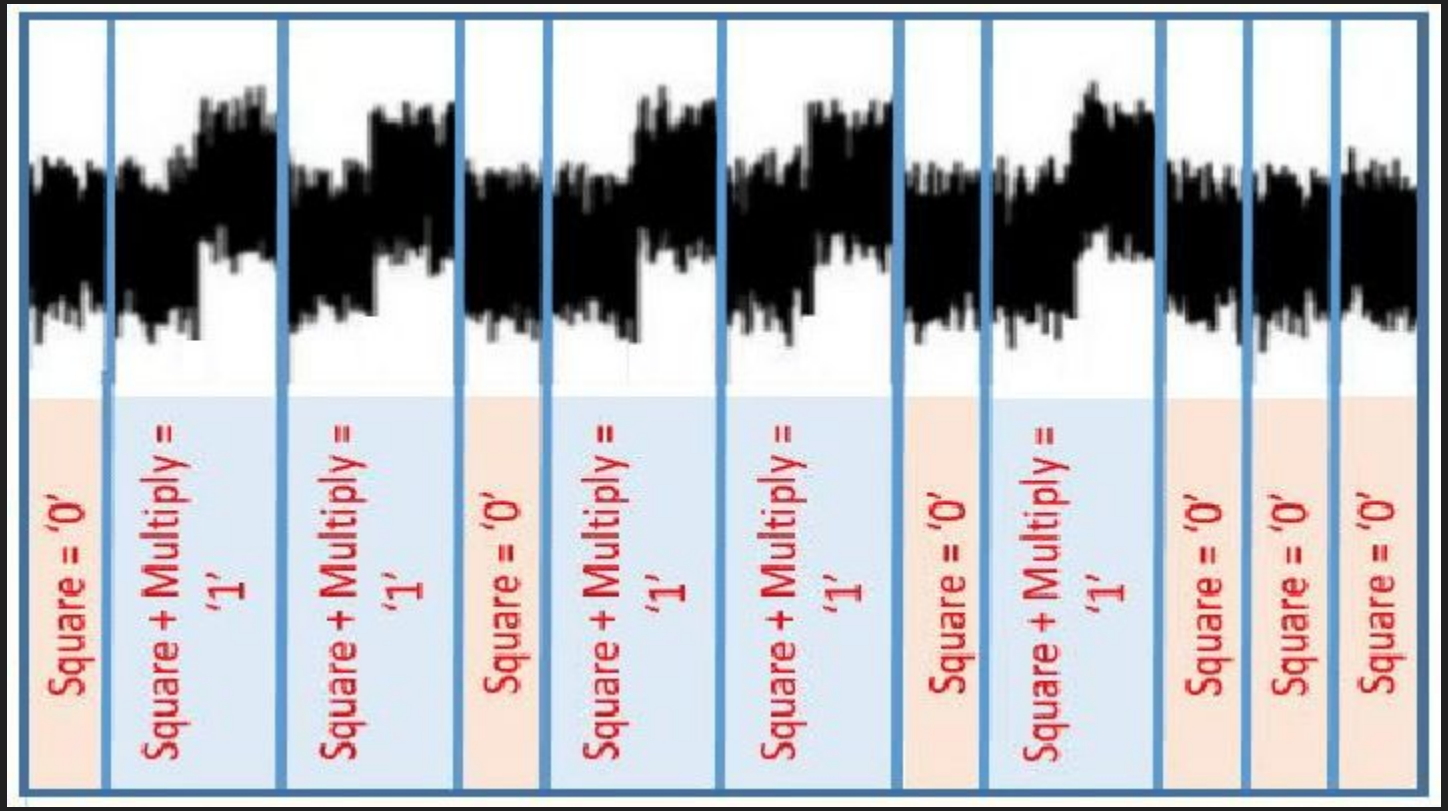
```
func square_and_multiply(m,exp,n)
    x = 1
    for bit in bit_array_msb(exp)
        x = x*x mod n  # do square
        if bit == 1
            x = x*m        # do multiplication
    return x
```

Simple Power Analysis

Modular exponentiation

```
func square_and_multiply(m,exp,n)
    x = 1
    for bit in bit_array_msb(exp)
      x = x*x mod n  # do square
      if bit == 1
          x = x*m        # do multiplication
    return x
```

Simple Power Analysis



Simple Power Analysis

Hands on

$ sh launch_workshop.sh rsa

cd /attack/

python3 spa.py

Recover the key

Differential Power Analysis

Differential Power Analysis

Observation

- SPA requires visually identifying patterns in power traces
- This is difficult with noisy power signals

Enter DPA

- DPA is a statistical technique overcoming noise limitations
- It analyzes power traces from multiple inputs

Differential Power Analysis

How it works

1. Choose an intermediate value that depends on the secret key
2. Make a guess about the secret key
3. Divide power traces into two groups based on the predicted bit of the intermediate value
4. Calculate the average power trace for each group
5. Subtract the averages to find the differential trace

Why it works

- If key is the right one, the device will manipulate the predicted bit
- Average of bit 1 power trace minus average of bit 0 power trace will give a **spike**

Differential Power Analysis

QUICK LIVE DEMO

```
import random
def power(bit):
    power_0 = 50
    power_1 = 60
    if bit == 0:
        return power_0 + random.randint(0,2)
    else:
        return power_1 + random.randint(0,2)


trace0 = 0
trace1 = 0
for i in range(10000):
    guess = random.randint(0,1)
    if guess == 0:
        trace0+= power(guess)
    else:
        trace1+= power(guess)

print(trace1-trace0)
```

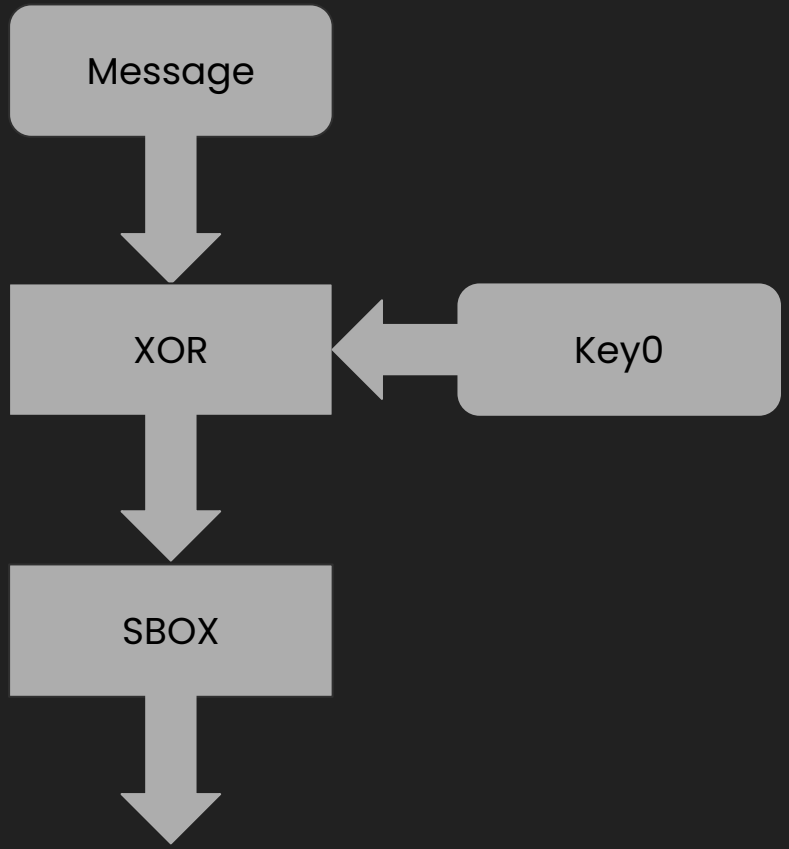
Differential Power Analysis

Let’s target AES

- Message is 16 bytes long
- Key0 is the actual AES key for AES-128
- SBOX is a byte-substitution table

```
for i in range(16):

  state[i] = sbox[m[i] ^ k[i]]
```



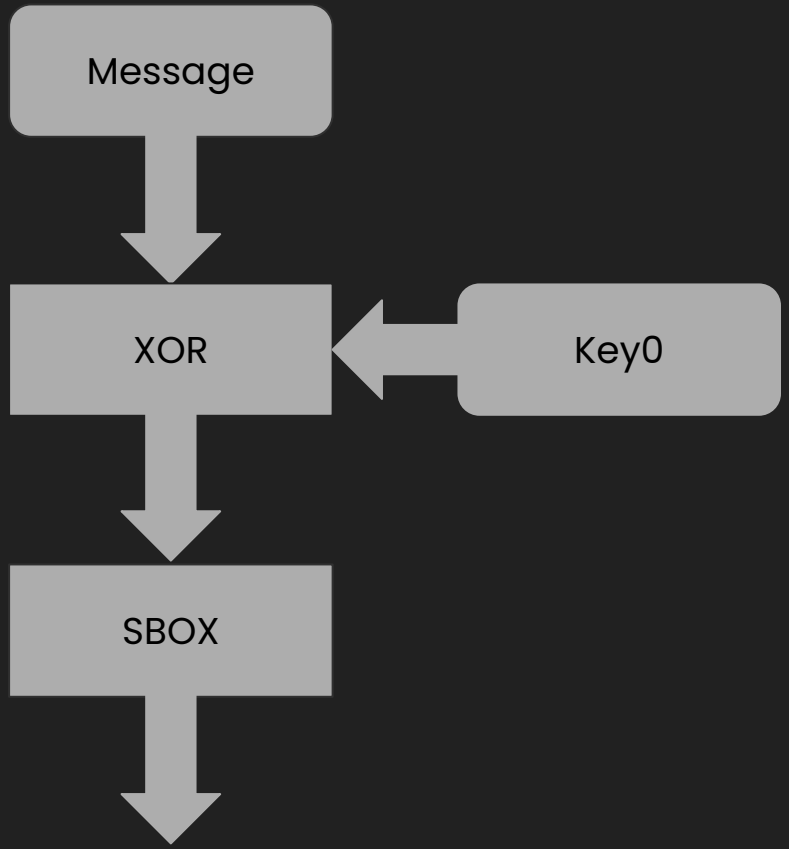
Differential Power Analysis

Let's target AES

1. Choose an intermediate value that depends on the secret key => **One of the Sbox output bit**

2. Make a guess about the secret key => **Try all key bytes**

3. Perform the DPA



Differential Power Analysis

Hands on

$ sh launch_workshop.sh aes

cd /attack/

python3 acquisition.py

python3 dpa.py

Recover the key

Correlation Power Analysis

Correlation Power Analysis

- DPA focuses on one bit leakage
- However, an operation on byte involves 8 bits.
- To simplify, the power consumption is dependent on the number of 1s to be processed
- Said differently:

The power consumption is correlated with the Hamming Weight

Correlation Power Analysis

How it works

1. Choose an intermediate value that depends on the secret key

2. Make a guess about the secret key

3. We compute the **correlation coefficient** between the hamming weight of intermediate values and the power traces at a given point of time

4. If the correlation is high => the key guess is correct

5. Else => the key guess is incorrect

6. If all key guesses are incorrect, the point of time is incorrect

Correlation Power Analysis

Correlation coefficient: **Pearson correlation coefficient**

It measures linear correlation between two sets of data.

r ϵ [−1 , 1]

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

r = correlation coefficient

x_i = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

y_i = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable

Correlation Power Analysis

Hands on