

# Implementation Security In Cryptography

Lecture 11: Entering the World of Attacks

# Recap

- In the last lecture
  - Compact design of AES

# Today

- Few more words about implementations
- Entering the world of attacks.

# AES Once Again

- Well, we have seen how it is done in hardware..
- But what about software?
- A popular, extremely fast, yet terrible approach — T tables
  - Used quite a lot in OpenSSL
    - Not used anymore due to several **cache timing attacks**
- But table based secure implementations also do exist



# AES T Tables

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Let us consider the MixColumns operation
- In software, each 32-bit AES column is a uint32 variable.

$$s_0 = 2c_0 \oplus 3c_1 \oplus 1c_2 \oplus 1c_3$$

$$s_1 = 1c_0 \oplus 2c_1 \oplus 3c_2 \oplus 1c_3$$

$$s_2 = 1c_0 \oplus 1c_1 \oplus 2c_2 \oplus 3c_3$$

$$s_3 = 3c_0 \oplus 1c_1 \oplus 1c_2 \oplus 2c_3$$

# AES T Tables

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Let us consider the MixColumns operation
- In software, each 32-bit AES column is a uint32 variable.
- So we can store  $s$  in a uint32 variable.
- Here each  $s_i$  is 8-bit, so we can simply do concatenation to get 32-bits.

$$s_0 = 2c_0 \oplus 3c_1 \oplus 1c_2 \oplus 1c_3$$

$$s_1 = 1c_0 \oplus 2c_1 \oplus 3c_2 \oplus 1c_3$$

$$s_2 = 1c_0 \oplus 1c_1 \oplus 2c_2 \oplus 3c_3$$

$$s_3 = 3c_0 \oplus 1c_1 \oplus 1c_2 \oplus 2c_3$$

$$s = s_0 \mid s_1 \mid s_2 \mid s_3$$

$$s = 2*c_0 \wedge 3*c_1 \wedge 1*c_2 \wedge 1*c_3 \mid$$

$$1*c_0 \wedge 2*c_1 \wedge 3*c_2 \wedge 1*c_3 \mid$$

$$1*c_0 \wedge 1*c_1 \wedge 2*c_2 \wedge 3*c_3 \mid$$

$$3*c_0 \wedge 1*c_1 \wedge 1*c_2 \wedge 2*c_3$$

# AES T Tables

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Here each  $s_i$  is 8-bit, so we can simply do concatenation to get 32-bits.
- Now we can also rearrange the terms.
  - Why this is beneficial??

$$\begin{aligned} s &= s_0 \mid s_1 \mid s_2 \mid s_3 \\ s &= 2*c_0 \wedge 3*c_1 \wedge 1*c_2 \wedge 1*c_3 \mid \\ &\quad 1*c_0 \wedge 2*c_1 \wedge 3*c_2 \wedge 1*c_3 \mid \\ &\quad 1*c_0 \wedge 1*c_1 \wedge 2*c_2 \wedge 3*c_3 \mid \\ &\quad 3*c_0 \wedge 1*c_1 \wedge 1*c_2 \wedge 2*c_3 \end{aligned}$$

$$\begin{aligned} s &= (2*c_0 \mid 1*c_0 \mid 1*c_0 \mid 3*c_0) \wedge \\ &\quad (3*c_1 \mid 2*c_1 \mid 1*c_1 \mid 1*c_1) \wedge \\ &\quad (1*c_2 \mid 3*c_2 \mid 2*c_2 \mid 1*c_2) \wedge \\ &\quad (1*c_3 \mid 1*c_3 \mid 3*c_3 \mid 2*c_3) \end{aligned}$$

# AES T Tables

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Here each  $s_i$  is 8-bit, so we can simply do concatenation to get 32-bits.
- Now we can also rearrange the terms.
  - Why this is beneficial??
  - Observe that you can compute each term being XORed only from a  $c_i$

$$\begin{aligned} s &= s_0 \mid s_1 \mid s_2 \mid s_3 \\ s &= 2*c_0 \wedge 3*c_1 \wedge 1*c_2 \wedge 1*c_3 \mid \\ &\quad 1*c_0 \wedge 2*c_1 \wedge 3*c_2 \wedge 1*c_3 \mid \\ &\quad 1*c_0 \wedge 1*c_1 \wedge 2*c_2 \wedge 3*c_3 \mid \\ &\quad 3*c_0 \wedge 1*c_1 \wedge 1*c_2 \wedge 2*c_3 \end{aligned}$$

$$\begin{aligned} s &= (2*c_0 \mid 1*c_0 \mid 1*c_0 \mid 3*c_0) \wedge \\ &\quad (3*c_1 \mid 2*c_1 \mid 1*c_1 \mid 1*c_1) \wedge \\ &\quad (1*c_2 \mid 3*c_2 \mid 2*c_2 \mid 1*c_2) \wedge \\ &\quad (1*c_3 \mid 1*c_3 \mid 3*c_3 \mid 2*c_3) \end{aligned}$$

# AES T Tables

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Why this is beneficial??
  - Observe that you can compute each term being XORed only from a  $c_i$
  - Since  $c_i$  is 8-bit, so we can compute all possible 256 values and store them in. A table.
  - Same can be done for all  $c_i$ 
    - One table for each
  - **Catch:** Table lookup is much faster than a finite field operation.

$$\begin{aligned} s &= s_0 \mid s_1 \mid s_2 \mid s_3 \\ s &= 2*c_0 \wedge 3*c_1 \wedge 1*c_2 \wedge 1*c_3 \mid \\ &\quad 1*c_0 \wedge 2*c_1 \wedge 3*c_2 \wedge 1*c_3 \mid \\ &\quad 1*c_0 \wedge 1*c_1 \wedge 2*c_2 \wedge 3*c_3 \mid \\ &\quad 3*c_0 \wedge 1*c_1 \wedge 1*c_2 \wedge 2*c_3 \end{aligned}$$

$$\begin{aligned} s &= (2*c_0 \mid 1*c_0 \mid 1*c_0 \mid 3*c_0) \wedge \\ &\quad (3*c_1 \mid 2*c_1 \mid 1*c_1 \mid 1*c_1) \wedge \\ &\quad (1*c_2 \mid 3*c_2 \mid 2*c_2 \mid 1*c_2) \wedge \\ &\quad (1*c_3 \mid 1*c_3 \mid 3*c_3 \mid 2*c_3) \end{aligned}$$

# AES T Tables

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Why this is beneficial??
  - We denote these tables as te0, te1, te2, and te3.
  - The operation is as follows:

```
te0[i] = 2*i | 1*i | 1*i | 3*i
```

```
te1[i] = 3*i | 2*i | 1*i | 1*i
```

```
te2[i] = 1*i | 3*i | 2*i | 1*i
```

```
te3[i] = 1*i | 1*i | 3*i | 2*i
```

```
s = te0[c0] ^ te1[c1] ^ te2[c2] ^ te3[c3]
```

```
s = s0 | s1 | s2 | s3
```

```
s = 2*c0 ^ 3*c1 ^ 1*c2 ^ 1*c3 |
```

```
1*c0 ^ 2*c1 ^ 3*c2 ^ 1*c3 |
```

```
1*c0 ^ 1*c1 ^ 2*c2 ^ 3*c3 |
```

```
3*c0 ^ 1*c1 ^ 1*c2 ^ 2*c3
```

```
s = (2*c0 | 1*c0 | 1*c0 | 3*c0) ^
```

```
(3*c1 | 2*c1 | 1*c1 | 1*c1) ^
```

```
(1*c2 | 3*c2 | 2*c2 | 1*c2) ^
```

```
(1*c3 | 1*c3 | 3*c3 | 2*c3)
```

# It does not ends here...

$$\begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 1 \\ 1 & 3 & 2 & 1 \\ 1 & 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix}$$

- Can we do better than this?

- 

```
te0[i] = 2*i | 1*i | 1*i | 3*i
te1[i] = 3*i | 2*i | 1*i | 1*i
te2[i] = 1*i | 3*i | 2*i | 1*i
te3[i] = 1*i | 1*i | 3*i | 2*i
```

```
s = te0[c0] ^ te1[c1] ^ te2[c2] ^ te3[c3]
```

```
s = s0 | s1 | s2 | s3
s = 2*c0 ^ 3*c1 ^ 1*c2 ^ 1*c3 |
    1*c0 ^ 2*c1 ^ 3*c2 ^ 1*c3 |
    1*c0 ^ 1*c1 ^ 2*c2 ^ 3*c3 |
    3*c0 ^ 1*c1 ^ 1*c2 ^ 2*c3
```

```
s = (2*c0 | 1*c0 | 1*c0 | 3*c0) ^
     (3*c1 | 2*c1 | 1*c1 | 1*c1) ^
     (1*c2 | 3*c2 | 2*c2 | 1*c2) ^
     (1*c3 | 1*c3 | 3*c3 | 2*c3)
```

# It does not ends here...

$$\begin{aligned} s = & (2*c0 \mid 1*c0 \mid 1*c0 \mid 3*c0) \wedge \\ & (3*c1 \mid 2*c1 \mid 1*c1 \mid 1*c1) \wedge \\ & (1*c2 \mid 3*c2 \mid 2*c2 \mid 1*c2) \wedge \\ & (1*c3 \mid 1*c3 \mid 3*c3 \mid 2*c3) \end{aligned}$$

- Can we do better than this?
- Observe that:  $c0|c1|c2|c3$  can be
  - $s[b0]|s[b5]|s[b10]|s[b15]$  or,  $s[b4]|s[b9]|s[b14]|s[b3]$  or  
 $s[b8]|s[b13]|s[b2]|s[b7]$   $s[b12]|s[b1]|s[b6]|s[b11]$
  - So we can merge subtypes and shift rows in a table

$$\begin{aligned} R0 = & (2*S[b0] \mid 1*S[b0] \mid 1*S[b0] \mid 3*S[b0]) \wedge \\ & (3*S[b5] \mid 2*S[b5] \mid 1*S[b5] \mid 1*S[b5]) \wedge \\ & (1*S[b10] \mid 3*S[b10] \mid 2*S[b10] \mid 1*S[b10]) \wedge \\ & (1*S[b15] \mid 1*S[b15] \mid 3*S[b15] \mid 2*S[b15]) \end{aligned}$$

$$\begin{aligned} R1 = & (2*S[b4] \mid 1*S[b4] \mid 1*S[b4] \mid 3*S[b4]) \wedge \\ & (3*S[b9] \mid 2*S[b9] \mid 1*S[b9] \mid 1*S[b9]) \wedge \\ & (1*S[b14] \mid 3*S[b14] \mid 2*S[b14] \mid 1*S[b14]) \wedge \\ & (1*S[b3] \mid 1*S[b3] \mid 3*S[b3] \mid 2*S[b3]) \end{aligned}$$

$$\begin{aligned} R2 = & (2*S[b8] \mid 1*S[b8] \mid 1*S[b8] \mid 3*S[b8]) \wedge \\ & (3*S[b13] \mid 2*S[b13] \mid 1*S[b13] \mid 1*S[b13]) \wedge \\ & (1*S[b2] \mid 3*S[b2] \mid 2*S[b2] \mid 1*S[b2]) \wedge \\ & (1*S[b7] \mid 1*S[b7] \mid 3*S[b7] \mid 2*S[b7]) \end{aligned}$$

$$\begin{aligned} R3 = & (2*S[b12] \mid 1*S[b12] \mid 1*S[b12] \mid 3*S[b12]) \wedge \\ & (3*S[b1] \mid 2*S[b1] \mid 1*S[b1] \mid 1*S[b1]) \wedge \\ & (1*S[b6] \mid 3*S[b6] \mid 2*S[b6] \mid 1*S[b6]) \wedge \\ & (1*S[b11] \mid 1*S[b11] \mid 3*S[b11] \mid 2*S[b11]) \end{aligned}$$



# It does not ends here...

- So finally

```
R0 = te0[b0] ^ te1[b5] ^ te2[b10] ^ te3[b15]
R1 = te0[b4] ^ te1[b9] ^ te2[b14] ^ te3[b3]
R2 = te0[b8] ^ te1[b13] ^ te2[b2] ^ te3[b7]
R3 = te0[b12] ^ te1[b1] ^ te2[b6] ^ te3[b11]
```

# It doe

- So finall

```
R0 = te0[b0]
R1 = te0[b4]
R2 = te0[b8]
R3 = te0[b12]
```

```
// Initialize the state, stored in s0, s1, s2 and s3
s0 := uint32(src[0])<<24 | uint32(src[1])<<16 | uint32(src[2])<<8 | uint32(src[3])
s1 := uint32(src[4])<<24 | uint32(src[5])<<16 | uint32(src[6])<<8 | uint32(src[7])
s2 := uint32(src[8])<<24 | uint32(src[9])<<16 | uint32(src[10])<<8 | uint32(src[11])
s3 := uint32(src[12])<<24 | uint32(src[13])<<16 | uint32(src[14])<<8 | uint32(src[15])

// Add the first round key to the state
s0 ^= xk[0]
s1 ^= xk[1]
s2 ^= xk[2]
s3 ^= xk[3]

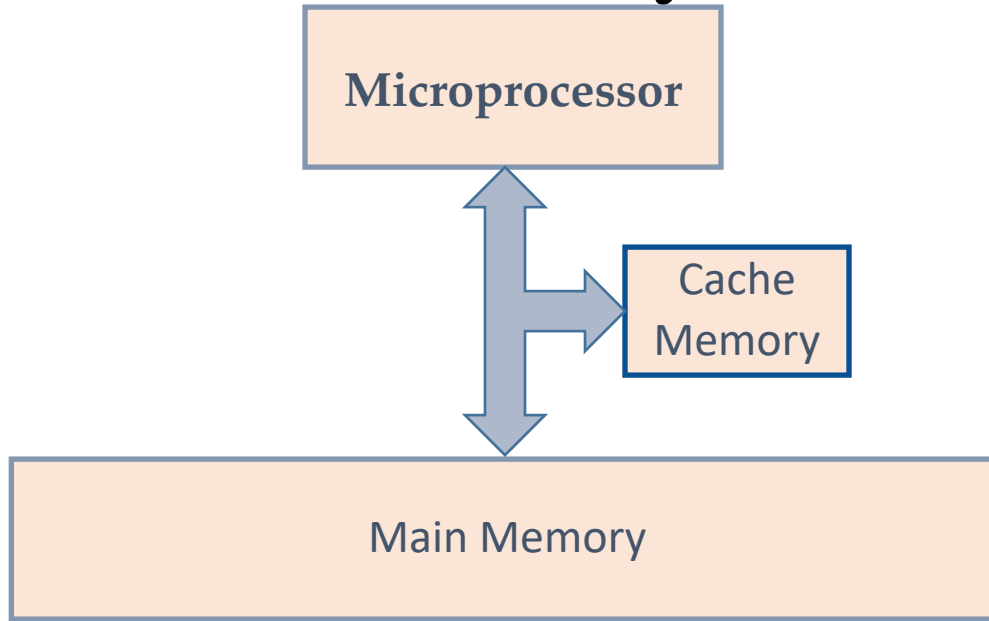
for i:= 1; i < nr; i++ {
    // This performs SubBytes + ShiftRows + MixColumns + AddRoundKey
    tmp0 = te0[s0>>24] ^ te1[s1>>16&0xff] ^ te2[s2>>8&0xff] ^ te3[s3&0xff] ^ xk[4*i]
    tmp1 = te0[s1>>24] ^ te1[s2>>16&0xff] ^ te2[s3>>8&0xff] ^ te3[s0&0xff] ^ xk[4*i+1]
    tmp2 = te0[s2>>24] ^ te1[s3>>16&0xff] ^ te2[s0>>8&0xff] ^ te3[s1&0xff] ^ xk[4*i+2]
    tmp3 = te0[s3>>24] ^ te1[s0>>16&0xff] ^ te2[s1>>8&0xff] ^ te3[s2&0xff] ^ xk[4*i+3]

    s0, s1, s2, s3 = tmp0, tmp1, tmp2, tmp3
}
```

# But As We Said...

- The tables are stored in cache memory of your system.
- AES accesses this table depending on the secret key values
- An adversary, who is able to measure the time for each encryption operation, and also using the same cache can do something so that it can recover the secret key!!!
- Such attacks are called cache timing attacks...

# Attacks due to Memory Wall

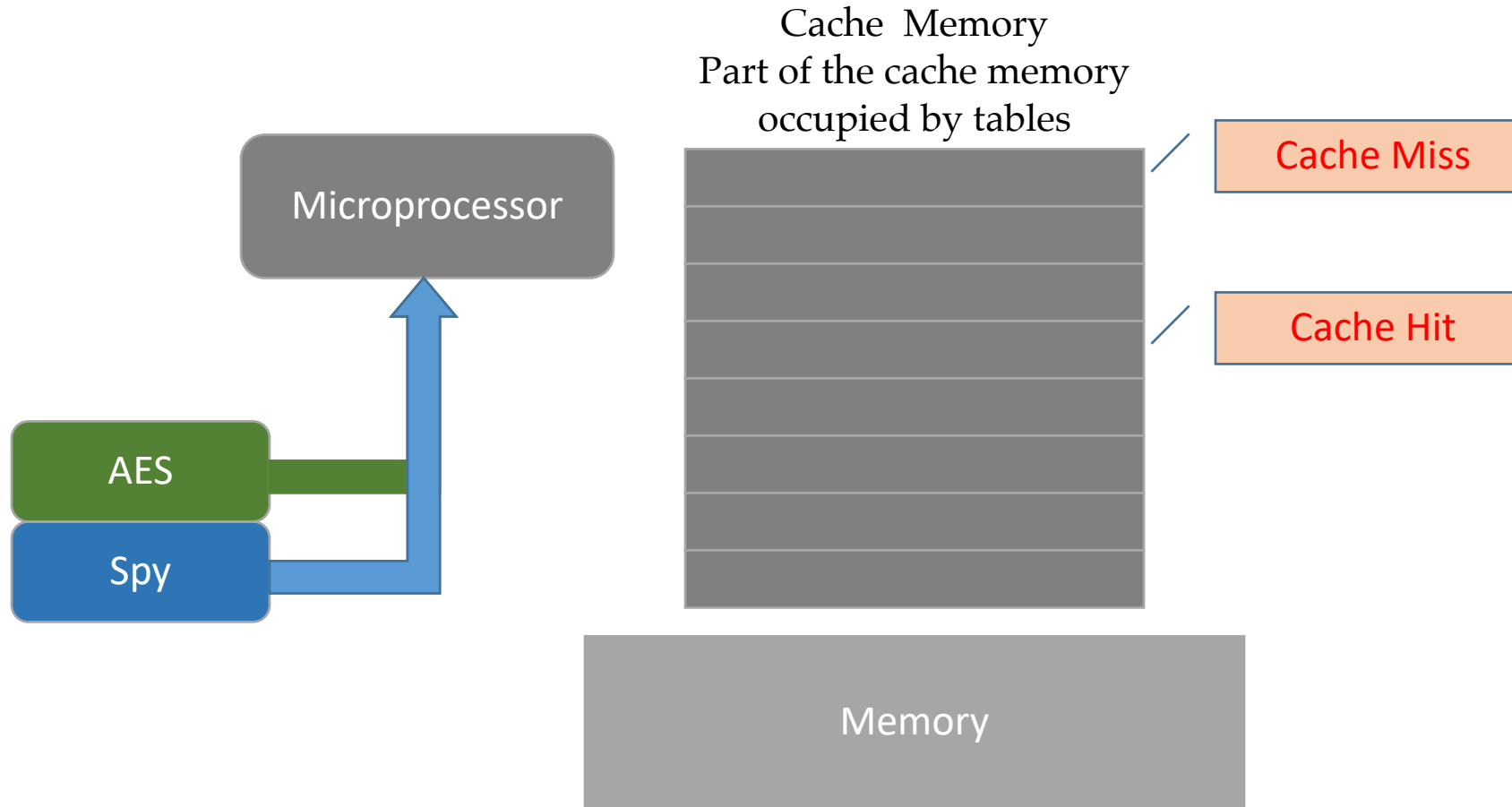


- If there is a *Cache Hit*
  - Access time is less
  - Power Consumption is less

- If there is a *Cache Miss*
  - Access time is more
  - Power Consumption is more

# Timing Attacks due to Cache Memory

- Uses a spy program to determine cache behavior



# Bitslicing

- Simply speaking, implement the software like hardware
- Results in constant-time crypto
- Idea: Let's say you are running on a 32-bit machine.
  - 32-bit registers
  - Logical AND, OR, NOT, XOR.
  - Also consider your block cipher in terms of these gates.

# Bitslicing: A Simple Example

- Let us consider the first equation only.
  - It can be written as follows:

and  $t1, x1, x4$

and  $t2, t1, x2$

and  $t3, t1, x3$

xor  $t4, t2, t3$

And so on...

$$y_1 = x_1x_2x_4 + x_1x_3x_4 \\ + x_1 + x_2x_3x_4 + x_2x_3 + x_3 + x_4 + 1$$

$$y_2 = x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1x_4 + \\ x_1 + x_2 + x_3x_4 + 1$$

$$y_3 = x_1x_2x_4 + x_1x_2 + x_1x_3x_4 + x_1x_3 + \\ x_1 + x_2x_3x_4 + x_3$$

$$y_4 = x_1 + x_2x_3 + x_2 + x_4$$

# Bitslicing: A Simple Example

- Let us consider the first equation only.
  - It can be written as follows:

and  $t1, x1, x4$

and  $t2, t1, x2$

and  $t3, t1, x3$

xor  $t4, t2, t3$

And so on...

$$\begin{aligned}y_1 &= x_1x_2x_4 + x_1x_3x_4 \\ &\quad + x_1 + x_2x_3x_4 + x_2x_3 + x_3 + x_4 + 1 \\ y_2 &= x_1x_2x_4 + x_1x_3x_4 + x_1x_3 + x_1x_4 + \\ &\quad x_1 + x_2 + x_3x_4 + 1 \\ y_3 &= x_1x_2x_4 + x_1x_2 + x_1x_3x_4 + x_1x_3 + \\ &\quad x_1 + x_2x_3x_4 + x_3 \\ y_4 &= x_1 + x_2x_3 + x_2 + x_4\end{aligned}$$

- Now, each of  $t1, x1, x2, \dots$  are mapped to 32 bit registers; but actually they are processing 1-bit values
  - So, what to do?



# Bitslicing: A Simple Example

- Let us consider the first equation only.

- It can be written as follows:

`and t1, x1, x4`

`and t2, t1, x2`

`and t3, t1, x3`

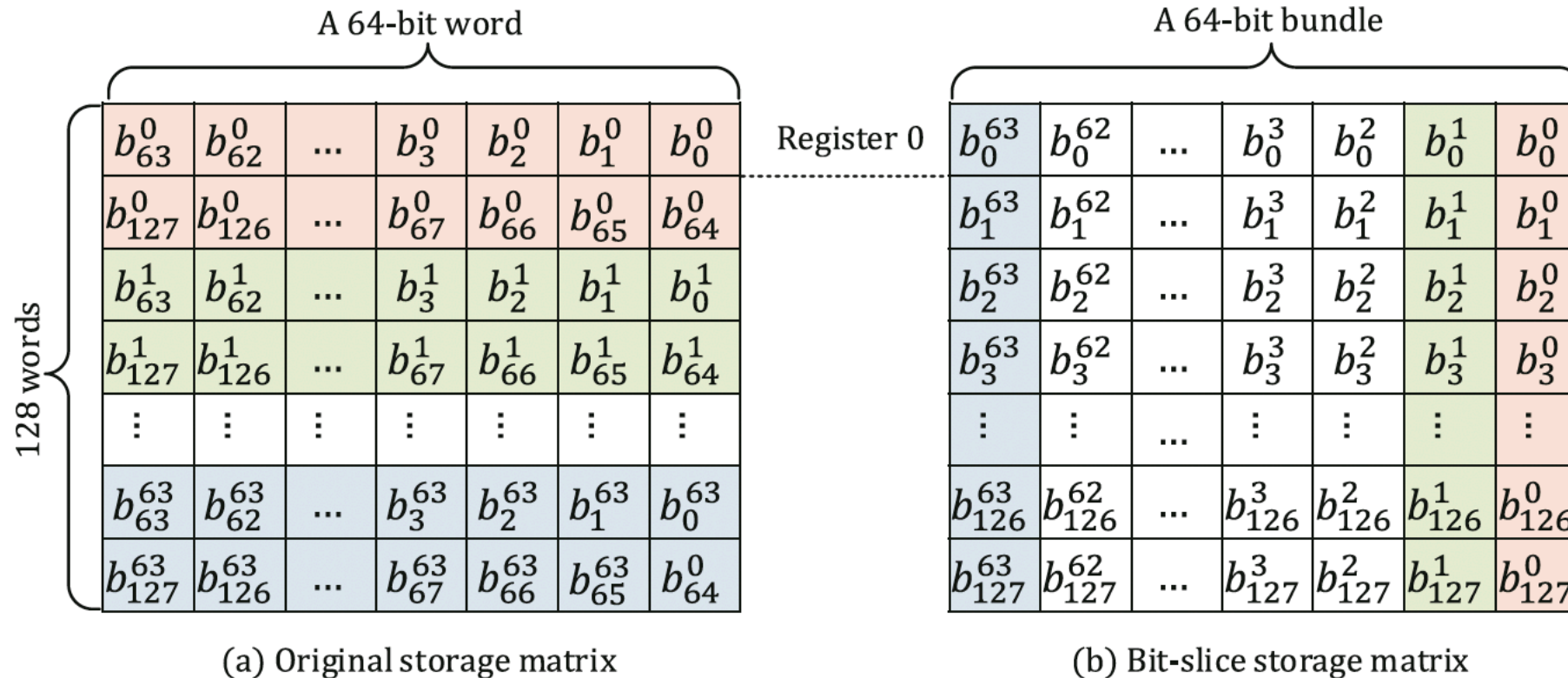
`xor t4, t2, t3`

And so on...

- Pack each register with independent values and process them using the same instruction.!!!
  - Easiest case: you can encrypt 32 plaintext together

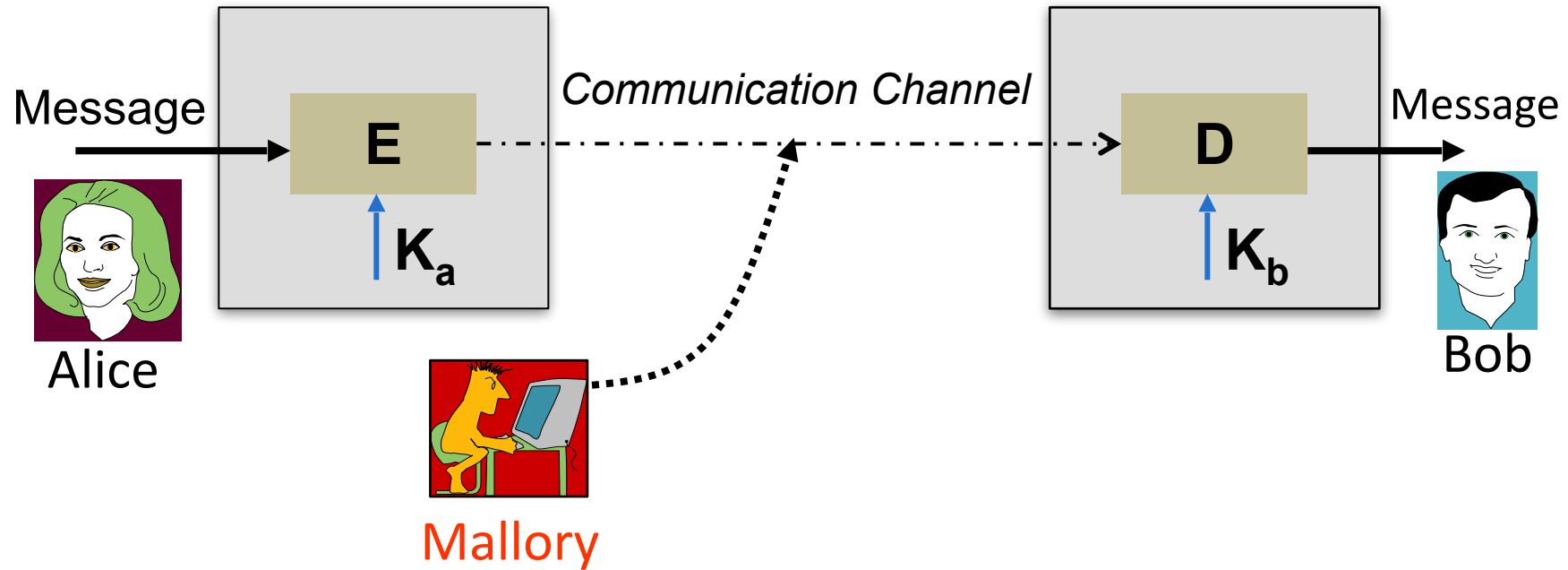
# Bitslicing: A Simple Example

- Easiest case: you can encrypt 32 plaintext together
- You can parallelize S-Box computations for one plaintext

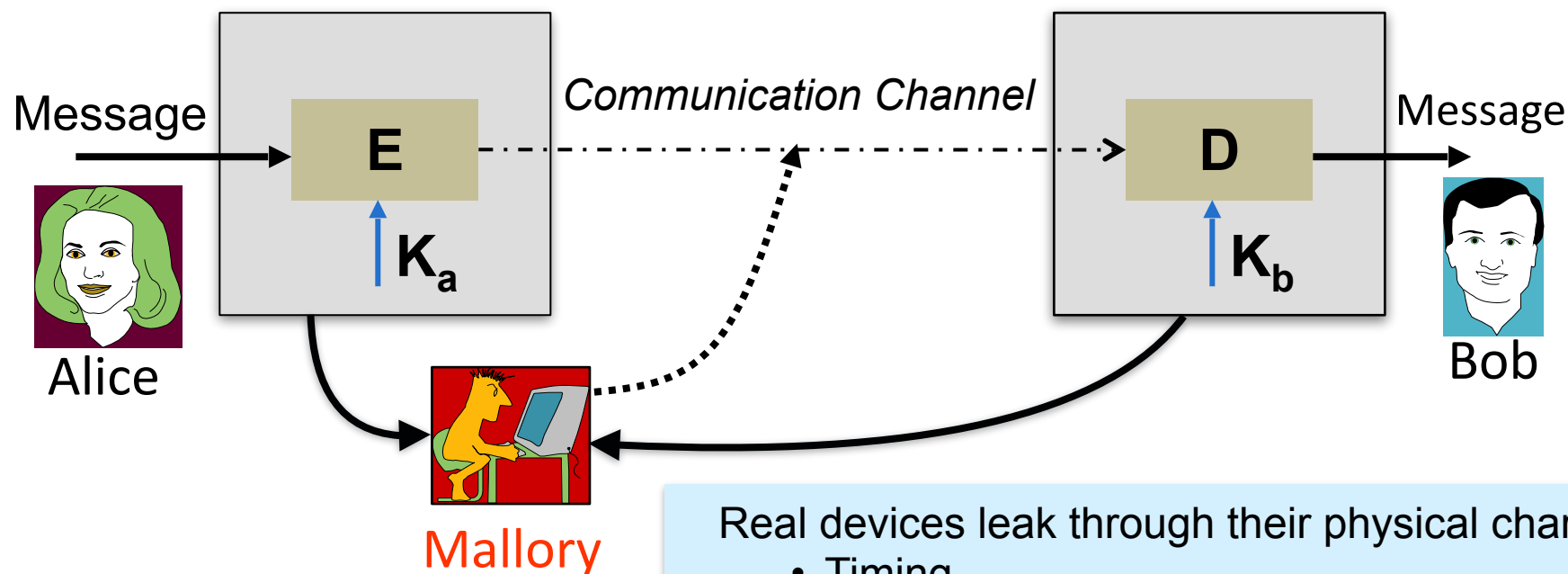


# Side Channel Attacks

# Why do Cryptographers Need Engineers?



# Cryptographic Security: Real World



Strong cryptographic algorithms  
are only the beginning

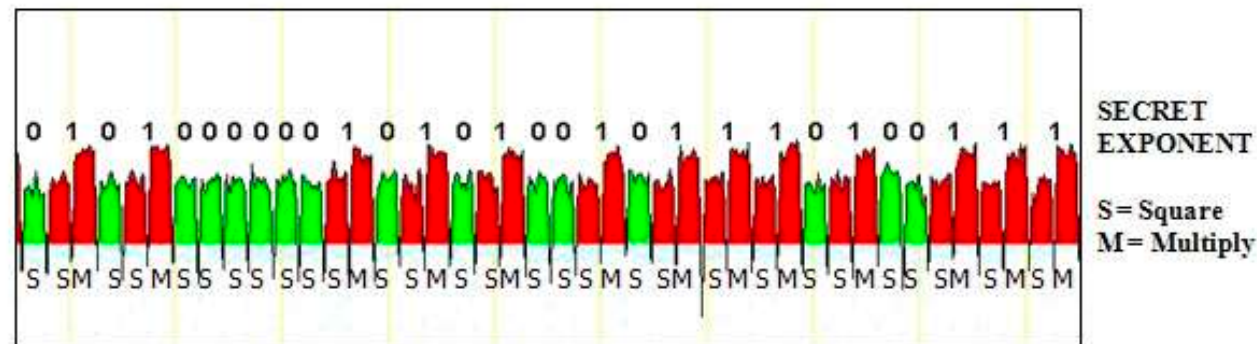
Real devices leak through their physical characteristics

- Timing
- Power consumption
- Electromagnetic Radiation
- Sound
- Faults

Analysis and mitigation of physical attacks are cryptographic as  
well as engineering problems

# Side-Channel Attacks (SCA)

- The physical channels are correlated with the information being processed
- Fundamental cause: power consumption is correlated with switching of CMOS transistors (0→1, 1→0)
  - Typically it is assumed that power consumption is correlated with the Hamming Weight/Distance.
- If some internal state is exposed, the secret key can be recovered in seconds.



Source: Internet

## Encryption

Plaintext

$$M < n$$

Ciphertext

$$C = M^e \pmod{n}$$

## Decryption

Ciphertext

C

Plaintext

$$M = C^d \pmod{n}$$

## Square and Multiply Algorithm

# Goal: Compute  $a^e \pmod n$

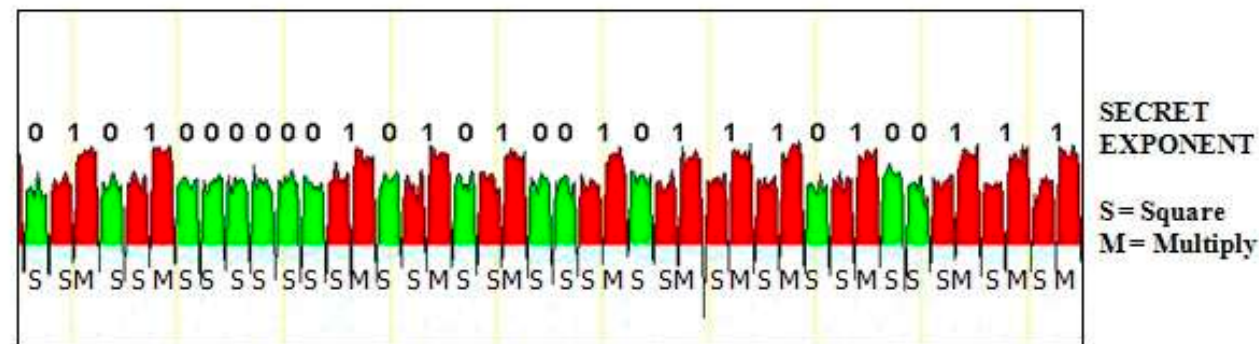
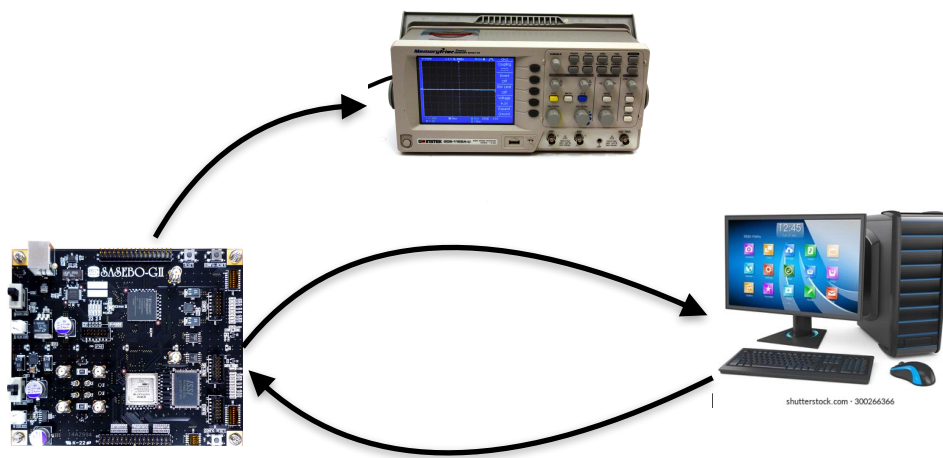
- ```

1. convert e to binary:  $k_s k_{s-1} \dots k_1 k_0$ 
2.  $b = 1$ ;
3. for ( $i = s$ ;  $i \geq 0$ ;  $i--$ )
4. {  $b = b * b \pmod{n}$ ;
5.   if ( $k_i == 1$ )
6.      $b = b * a \pmod{n}$ 
7. }
8. return b;

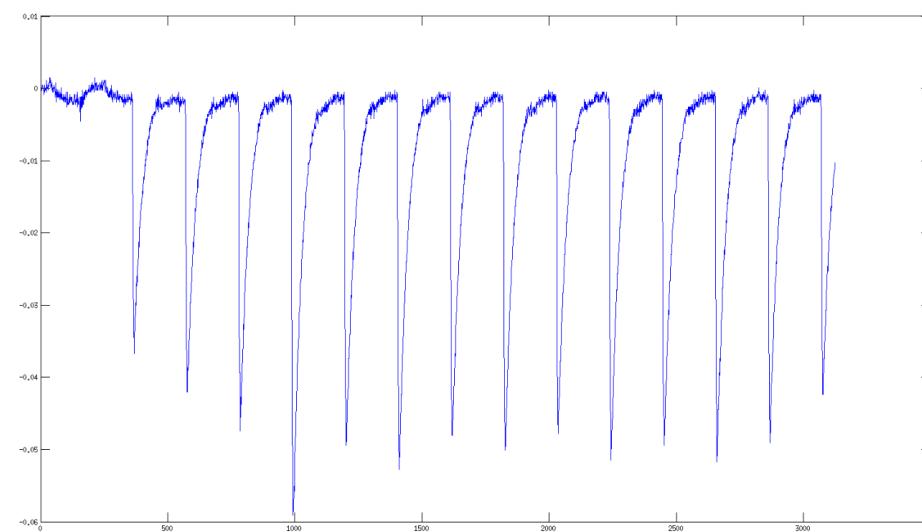
```

# Side-Channel Attacks (SCA)

- The physical channels are correlated with the information being processed
- Fundamental cause: power consumption is correlated with switching of CMOS transistors (0→1, 1→0)
  - Typically it is assumed that power consumption is correlated with the Hamming Weight/Distance.
- If some internal state is exposed, the secret key can be recovered in seconds.



Source: Internet

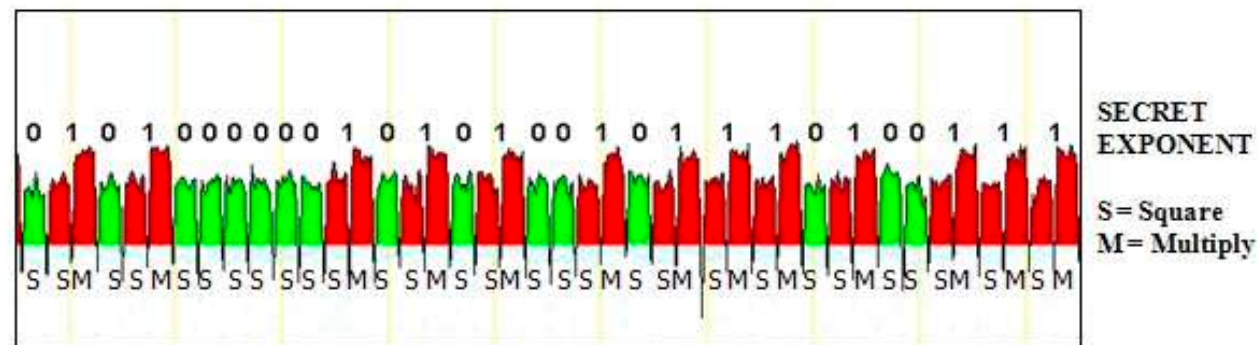


Source: Testbed for Side Channel analysis and security evaluation

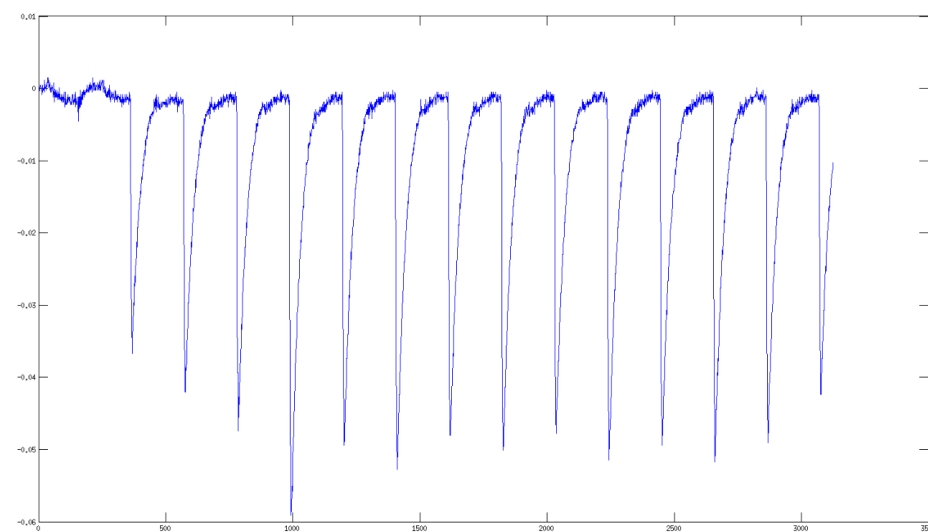
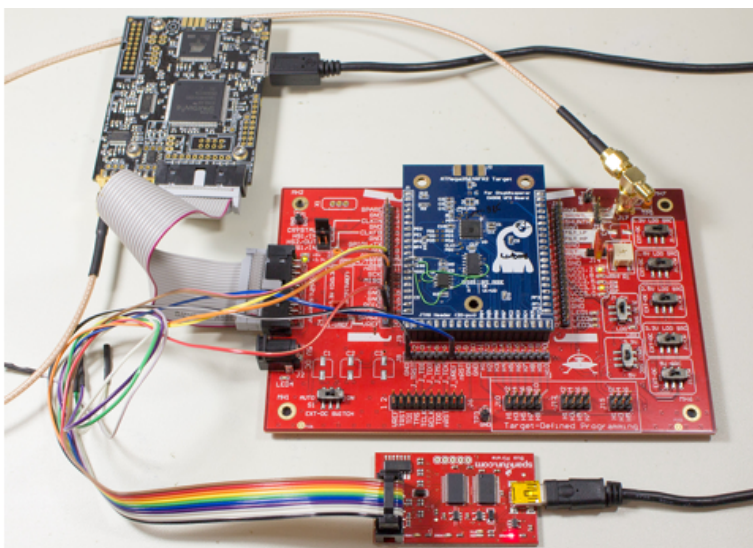


# Side-Channel Attacks (SCA)

- The physical channels are correlated with the information being processed
- Fundamental cause: power consumption is correlated with switching of CMOS transistors (0- $\rightarrow$ 1, 1- $\rightarrow$ 0)
  - Typically it is assumed that power consumption is correlated with the Hamming Weight/Distance.
- If some internal state is exposed, the secret key can be recovered in seconds.



Source: Internet



Source: Testbed for Side Channel analysis and security evaluation



# Side-Channel Vs. Classical Cryptanalysis

- Cryptanalysis: Purely mathematical
  - Take example of AES
  - Cryptanalysis means, you only have access to plaintext, ciphertext — a lot of them
  - You have to
    - Find the key
    - Or, at least, show that it is distinguishable from uniform randomness

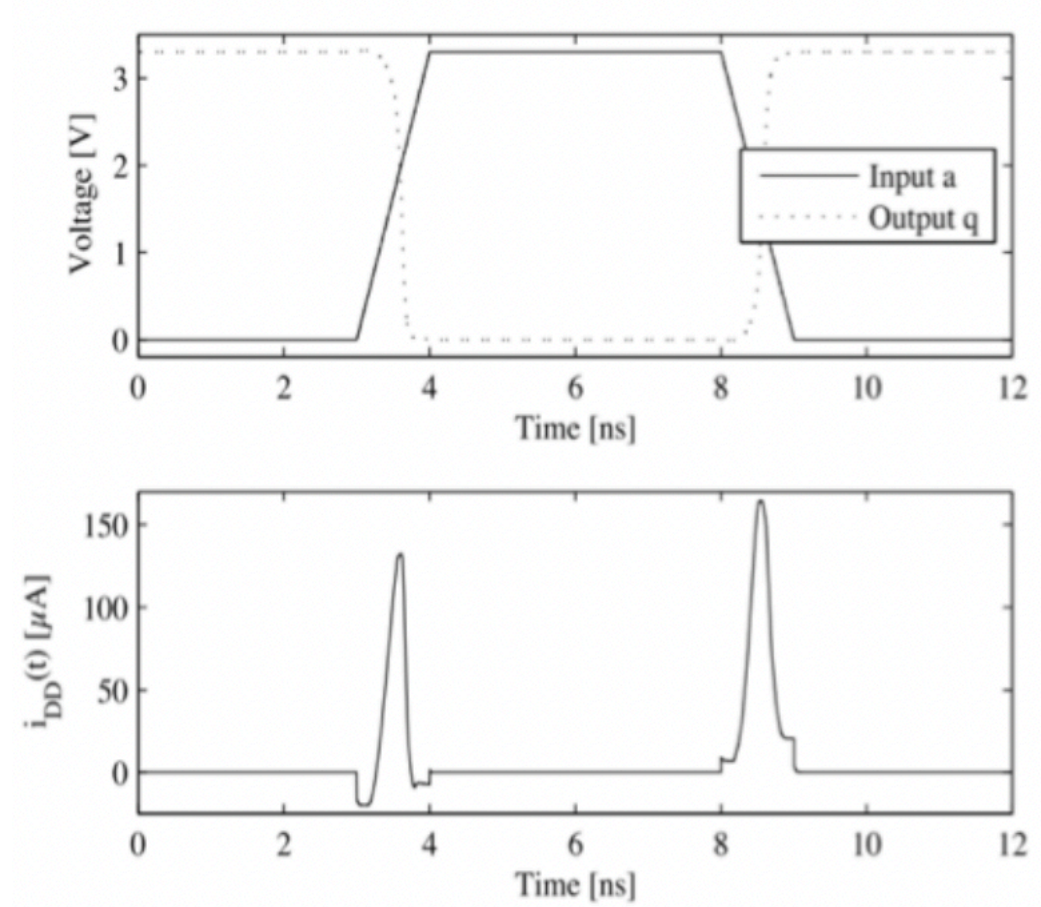
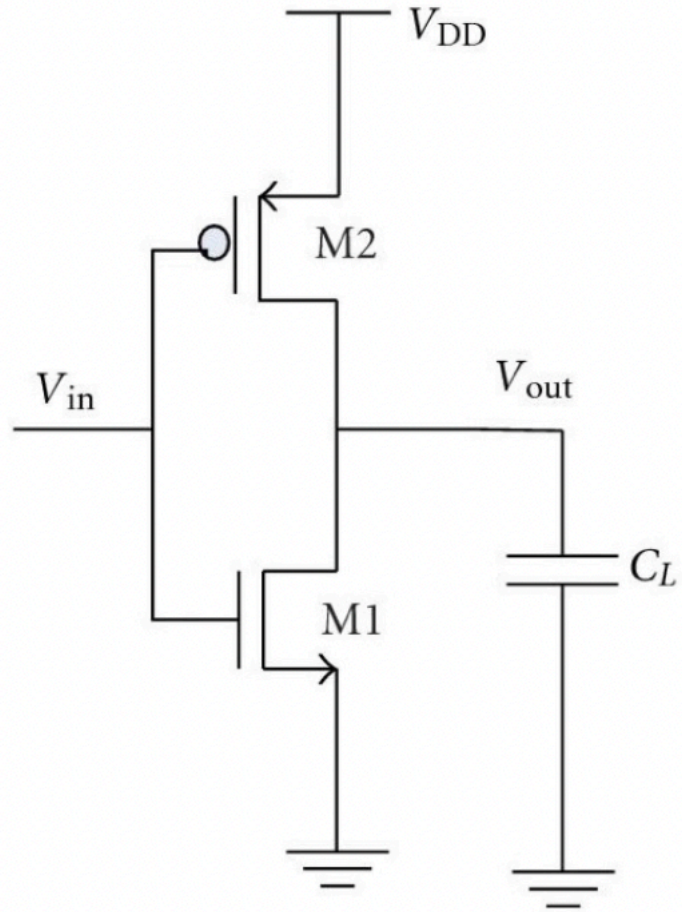
# Side-Channel Vs. Classical Cryptanalysis

- Cryptanalysis: Purely mathematical
  - Take example of RSA/ECC/PQC
  - Cryptanalysis means, you only have access to plaintext, ciphertext — a lot of them
  - You have to
    - Find the key
      - Maybe you need to solve the underlying hard problem in some (mathematical) way.

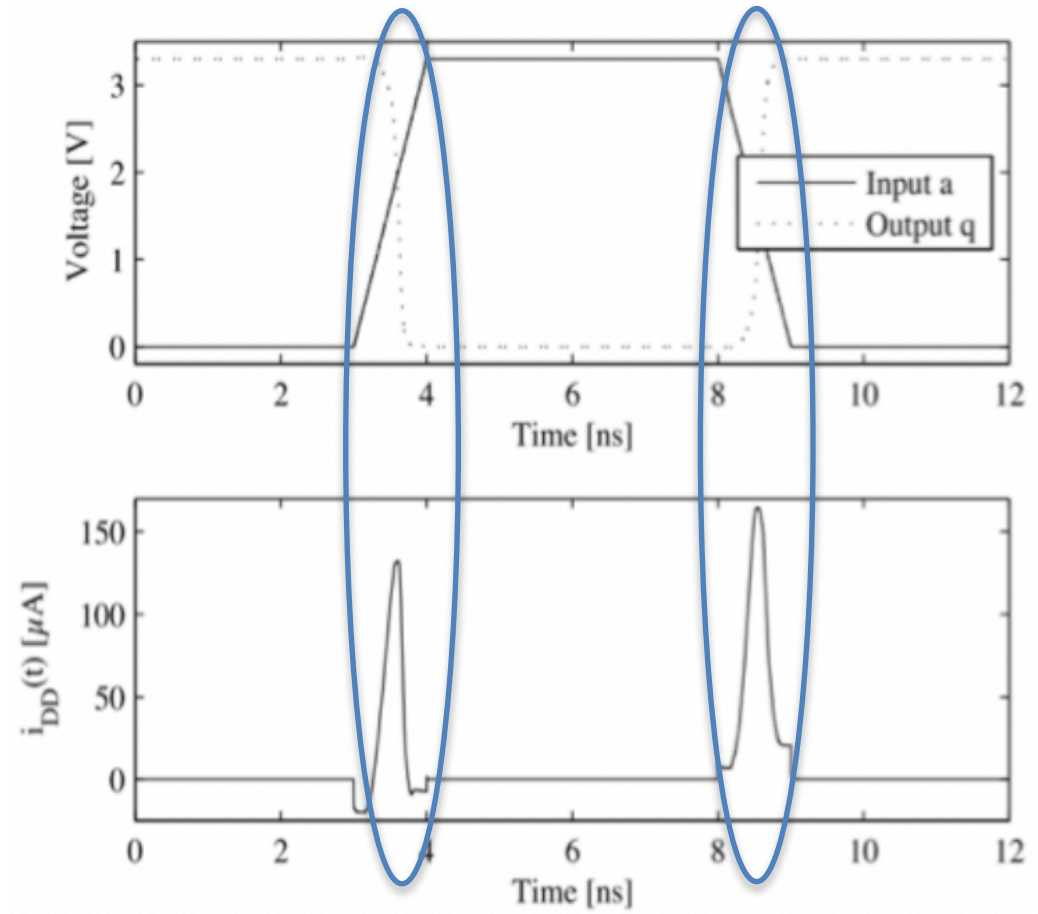
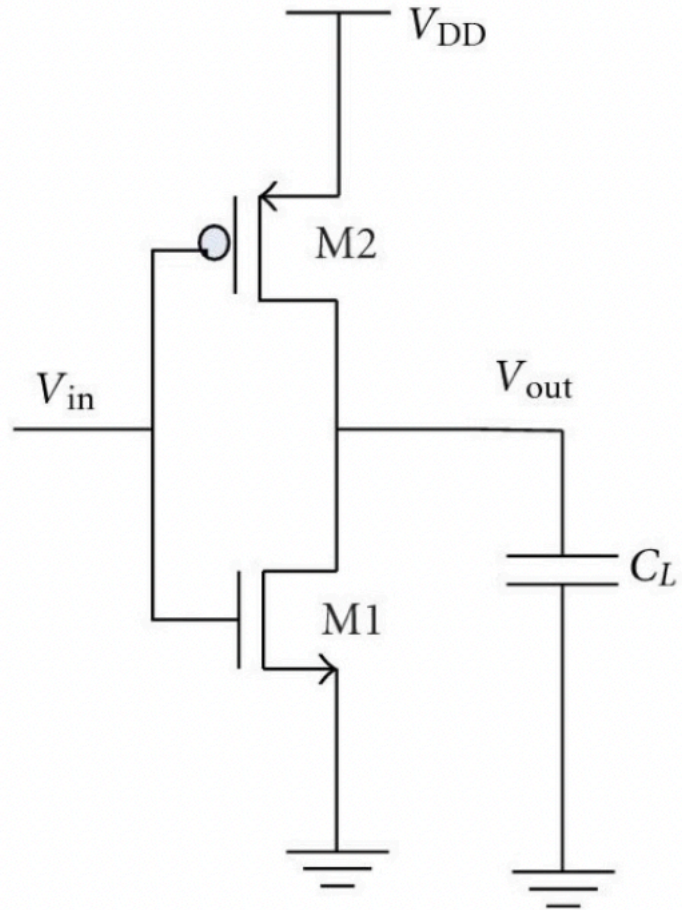
# Side-Channel Vs. Classical Cryptanalysis

- Side-Channel Cryptanalysis: Mathematics + Physics + Statistics
  - The goal is mostly to recover key
    - But also signature forgery, confidentiality breach
  - Ranges beyond crypto...
    - Kernel information extraction
    - Unprivileged access
    - Neural network reverse engineering

# The Root Cause



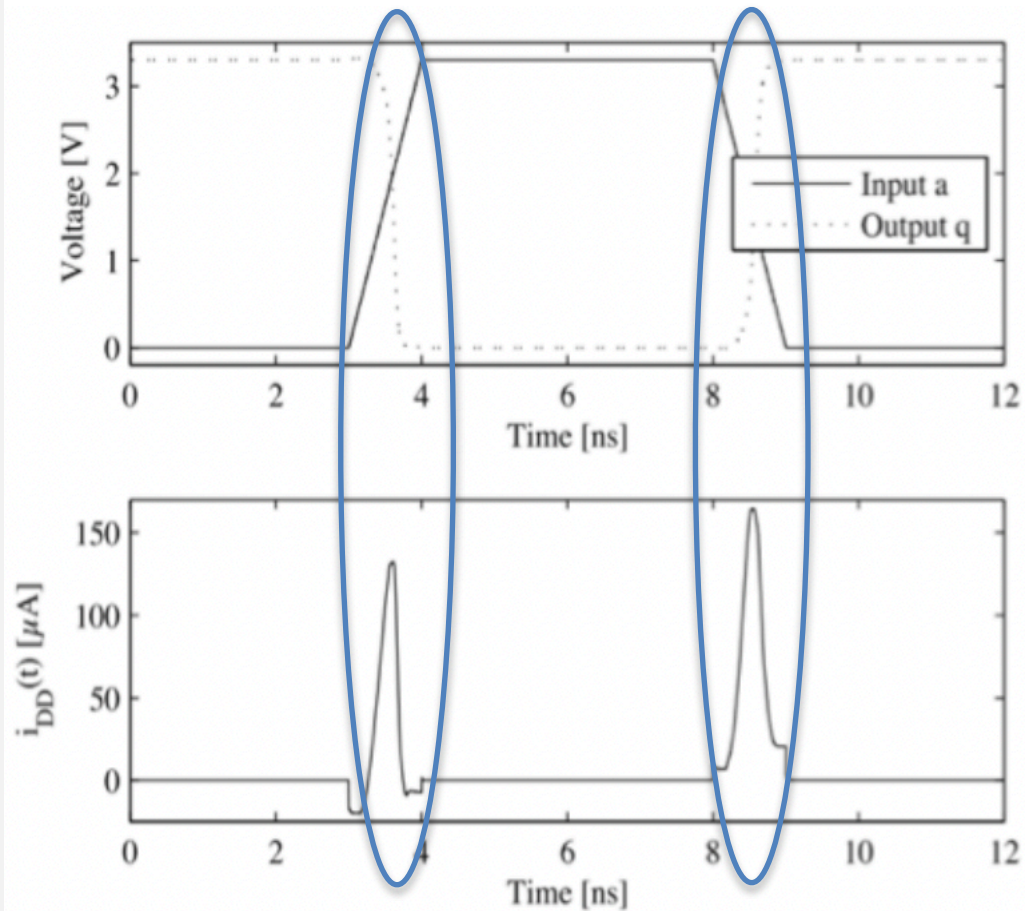
# The Root Cause



# The Root Cause

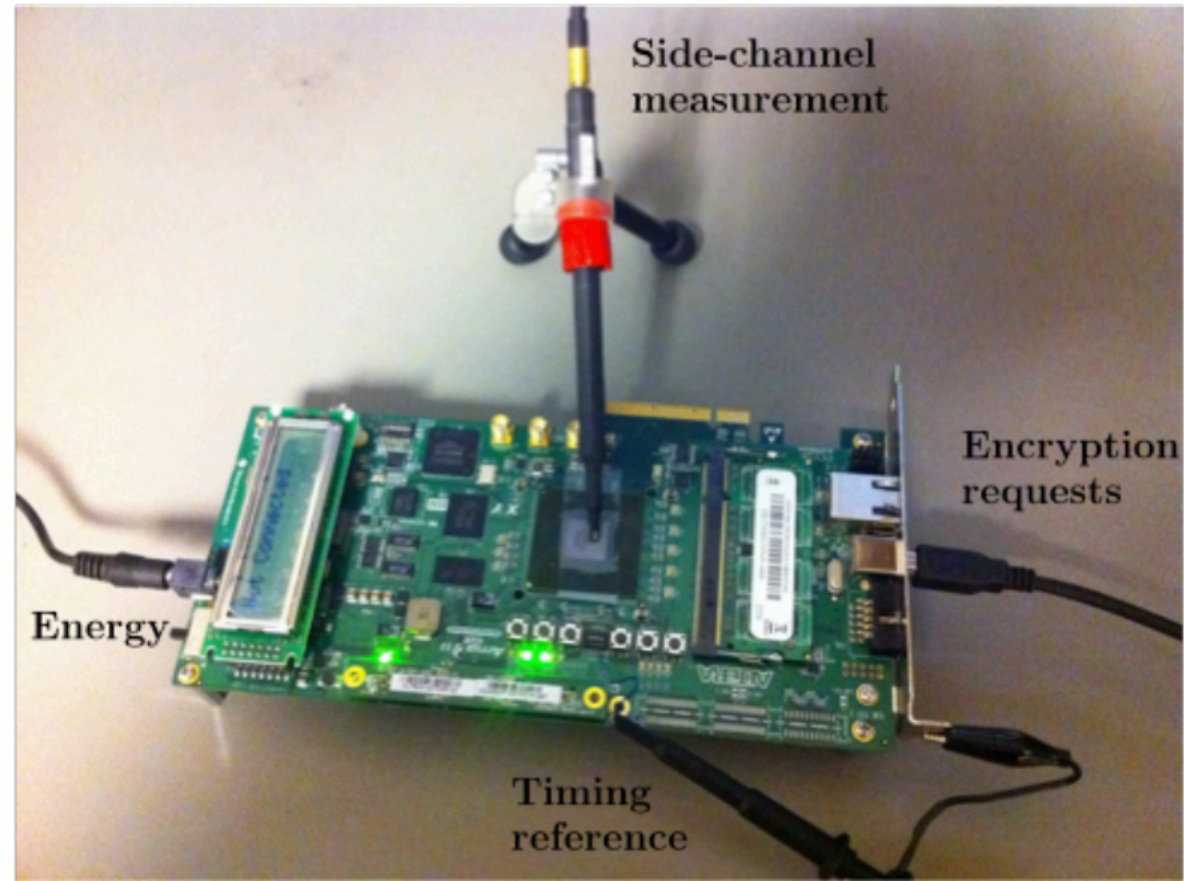
## What is exploited?

- The state change of a gate is proportional to the power dissipated.
- Think about a circuit with millions of gates.
- **How to measure**
  - Power dissipation can be measured by putting a resistor in series with Vdd or Vss and the true source/ground.
  - Roughly, 1 Ohm resistors work well for many microcontrollers, but it is highly target dependent
  - We actually measure current.
    - Differential probes.
  - **The best approach is to use a near-field H-probe and measure EM signal**
    - Less noisy than global power measurement





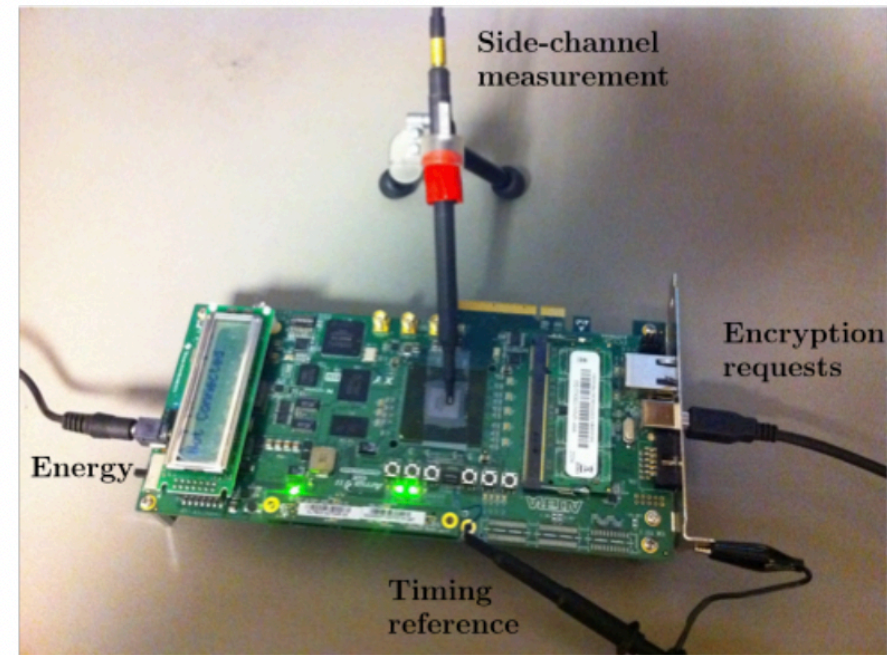
# The Root Cause



# What to “Measure”?

## The Crypto Running on a Microcontroller/ FPGA/ASIC

- End of the day everything is CMOS!!!
- Since power consumption is proportional to the switching activity, *so we can get some idea about the internal computation of the crypto*
  - **The crypto is no more black box**
- In this talk we will be specifically focusing on symmetric key algorithms
  - **AES**
- What do we mean by attacking AES?
  - **Finding out it's secret key**





# Looking Inside AES

## State

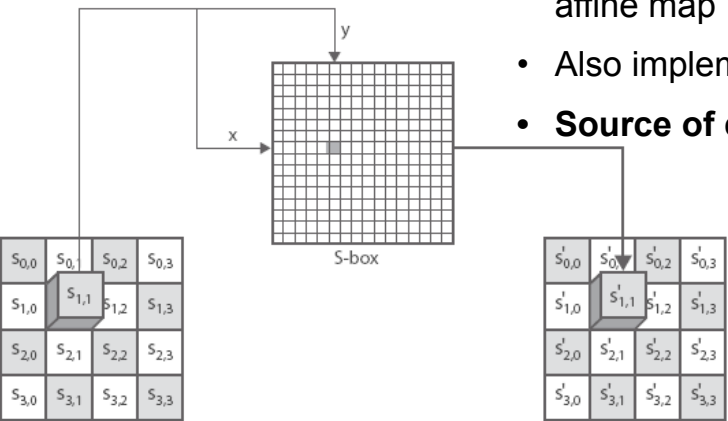
1 byte

|     |     |     |     |
|-----|-----|-----|-----|
| s00 | s01 | s02 | s03 |
| s10 | s11 | s12 | s13 |
| s20 | s21 | s22 | s23 |
| s20 | s31 | s32 | s33 |

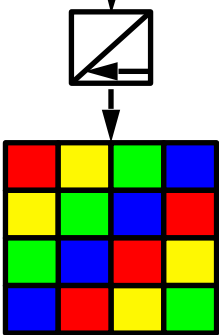
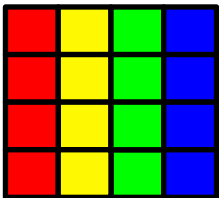
|     |     |     |     |
|-----|-----|-----|-----|
| k00 | k01 | k02 | k03 |
| k10 | k11 | k12 | k13 |
| k20 | k21 | k22 | k23 |
| k20 | k31 | k32 | k33 |

## 1. SubBytes

- Nonlinear Boolean Function
- Finite field inversion followed by affine map
- Also implemented as a table
- **Source of confusion**

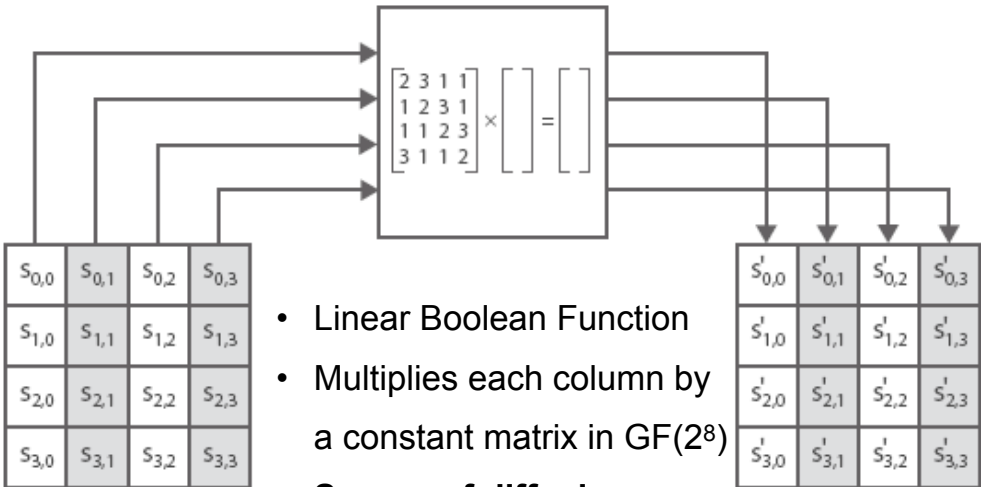


## 2. ShiftRows



- Linear Boolean Function
- Left circular shift of rows
- **Source of diffusion**

## 3. MixColumns



- Linear Boolean Function
- Multiplies each column by a constant matrix in  $GF(2^8)$
- **Source of diffusion**

## 4. AddRoundKey

|     |     |     |     |
|-----|-----|-----|-----|
| s00 | s01 | s02 | s03 |
| s10 | s11 | s12 | s13 |
| s20 | s21 | s22 | s23 |
| s20 | s31 | s32 | s33 |



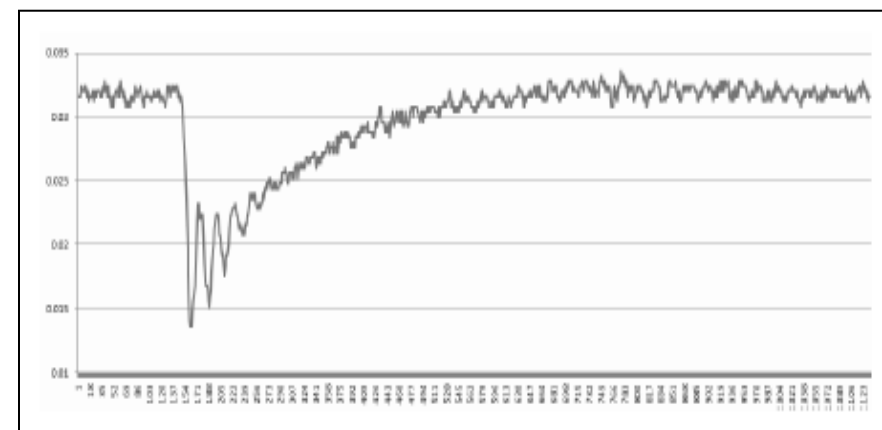
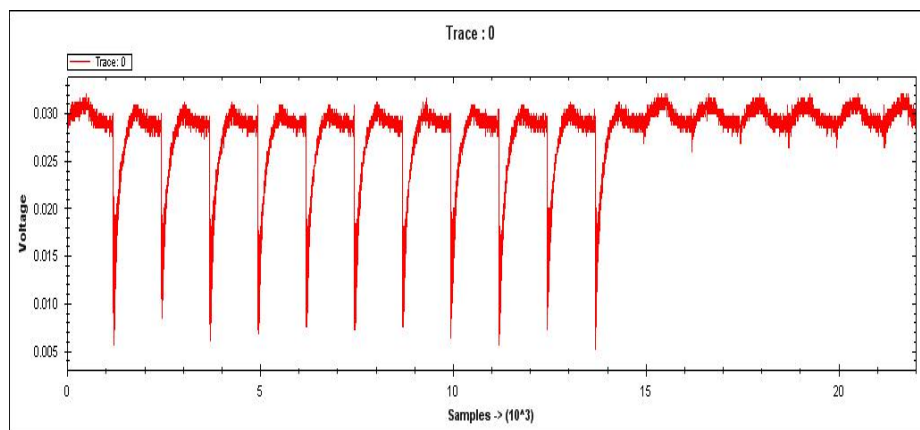
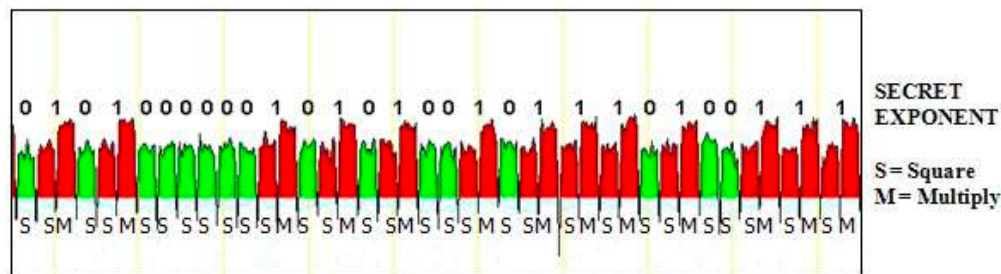
|     |     |     |     |
|-----|-----|-----|-----|
| k00 | k01 | k02 | k03 |
| k10 | k11 | k12 | k13 |
| k20 | k21 | k22 | k23 |
| k20 | k31 | k32 | k33 |

- Linear Boolean Function
- XOR the state with a round key

# Through the Looking Glass

**There can be two kinds of power analysis attacks:**

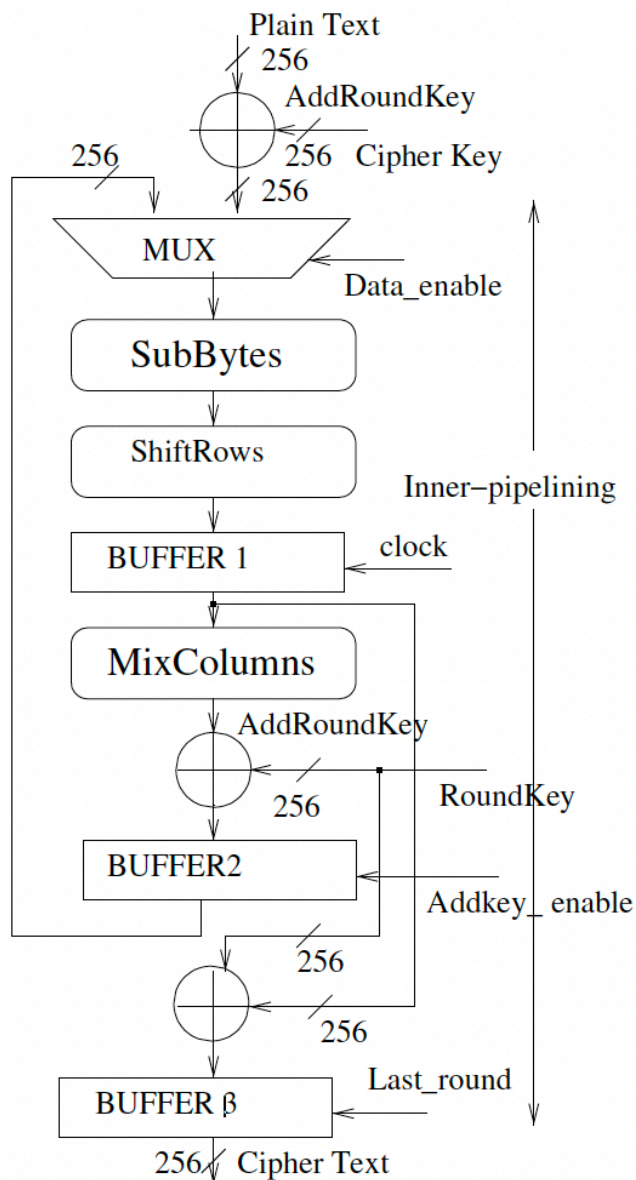
- **Simple power analysis (SPA):** Exploits the operation dependence of power consumption
  - Remember the RSA example from the beginning...
- **Differential Power Analysis (DPA):** Exploits the data dependence of power consumption
  - We will see now for AES
  - **Fact: there is no secret dependent operation in AES, everything is uniform.**



# Differential Power Analysis

- **Power Trace:** A set of power consumptions across a cryptographic process
  - 1 millisecond operation sampled at 5MHz yield a trace with 5000 points.
- **Leakage Model:** Hypothetical model relating the leakage with the internal states of the target algorithm.
  - For AES the internal state is a 128-bit value.
  - **Hamming Weight Model:** The power consumption is proportional to the Hamming weight (count of 1's) of the state.
  - **Hamming Distance Model:** The power consumption is proportional to the Hamming distance between the state in two consecutive clock cycles. — FPGA/ASICs
  - More complex models are possible...
  - Used to simulate leakage and also in some attacks.

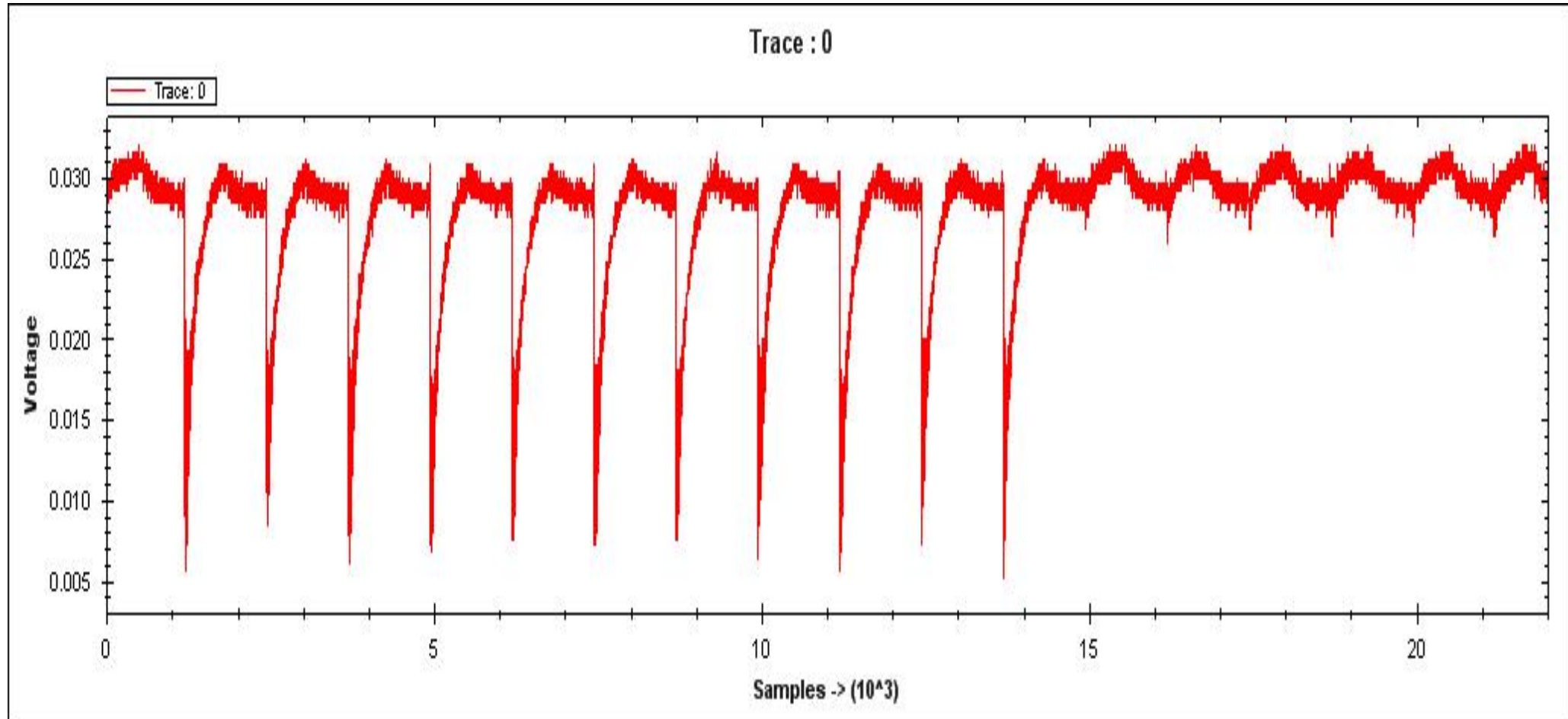
# Differential Power Analysis



- Common hardware implementation of a block cipher
- We consider the state of the circuit at time instance  $t$  — you can consider it as one time point in the x-axis of the trace.
- Let this state be  $v_t$
- Hamming weight is number of 1's in  $v_t$ .
- Hamming distance is  $\text{HW}(v_t \oplus v_{t-1})$ .
  - Why?

# Differential Power Analysis

- **Power Trace:** A set of power consumptions across a cryptographic process
  - 1 millisecond operation sampled at 5MHz yield a trace with 5000 points.



# Differential Power Analysis: The Idea

| $s$  | $\text{HW}(s)$ | Target bit (LSB) |
|------|----------------|------------------|
| 0000 | 0              | 0                |
| 0001 | 1              | 1                |
| 0010 | 1              | 0                |
| 0011 | 2              | 1                |
| 0100 | 1              | 0                |
| 0101 | 2              | 1                |
| 0110 | 2              | 0                |
| 0111 | 3              | 1                |
| 1000 | 1              | 0                |
| 1001 | 2              | 1                |
| 1010 | 2              | 0                |
| 1011 | 3              | 1                |
| 1100 | 2              | 0                |
| 1101 | 3              | 1                |
| 1110 | 3              | 0                |
| 1111 | 4              | 1                |

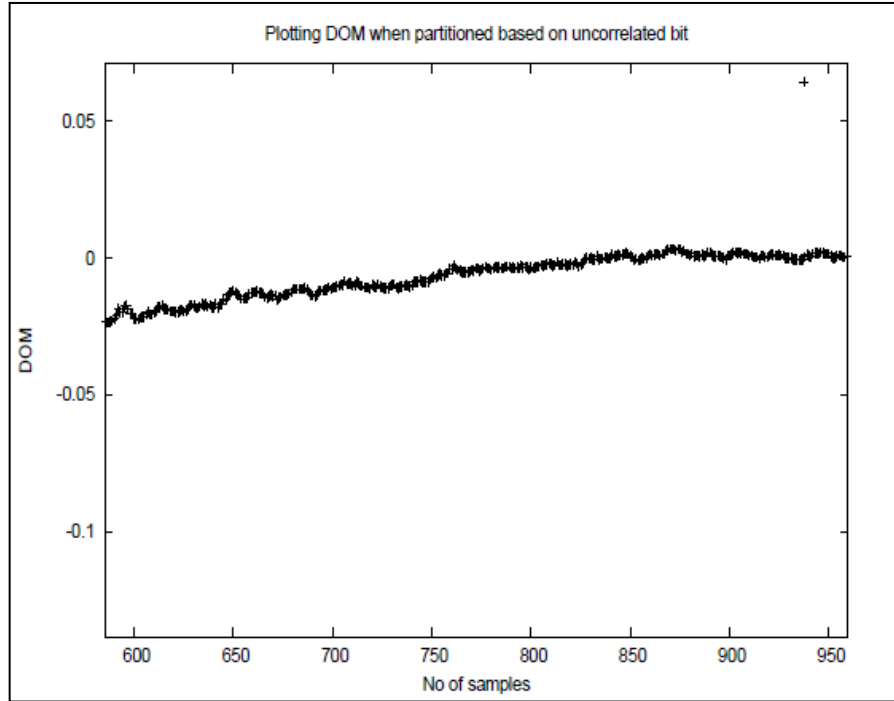
- Assume power leakage follows Hamming Weight.
- Divide the  $\text{HW}(s)$  into two bins:
  - 0 bin: when LSB is 0
  - 1 bin: when LSB is 1

# Differential Power Analysis: The Idea

| $s$  | $\text{HW}(s)$ | Target bit (LSB) |
|------|----------------|------------------|
| 0000 | 0              | 0                |
| 0001 | 1              | 1                |
| 0010 | 1              | 0                |
| 0011 | 2              | 1                |
| 0100 | 1              | 0                |
| 0101 | 2              | 1                |
| 0110 | 2              | 0                |
| 0111 | 3              | 1                |
| 1000 | 1              | 0                |
| 1001 | 2              | 1                |
| 1010 | 2              | 0                |
| 1011 | 3              | 1                |
| 1100 | 2              | 0                |
| 1101 | 3              | 1                |
| 1110 | 3              | 0                |
| 1111 | 4              | 1                |

- Assume power leakage follows Hamming Weight.
- Divide the  $\text{HW}(s)$  into two bins:
  - 0 bin: when LSB is 0
  - 1 bin: when LSB is 1
- Difference-of-Mean (DoM) =  $20/8 - 12/8 = 1$

# When the Partitioning is Random



- Partitioning done by bits simulated using *rand* function in C.
- Observe the DoM is close to 0, as expected!
- **Note:** Instead of computing the difference, we can use some statistical hypothesis test, such as t-test.
- The hypothesis will be — whether the two trace distributions are the same or different
- For a random uncorrelated bit, the two distributions are the same.

- Moral of the story: If the bin partitioning is based on a bit from the actual state, there will be significant difference in the mean values of the bins. This is because, the traces are data dependent.



# Attacking AES

## A very very important point

- We haven't seen AES key schedule in detail — but it has somewhat similar operations as the rounds — has SBoxes, shifts XORs etc.
- **Important:** key schedule is invertible
  - That is, if you recover one round key , you can recover all, and the master key

# Attacking AES

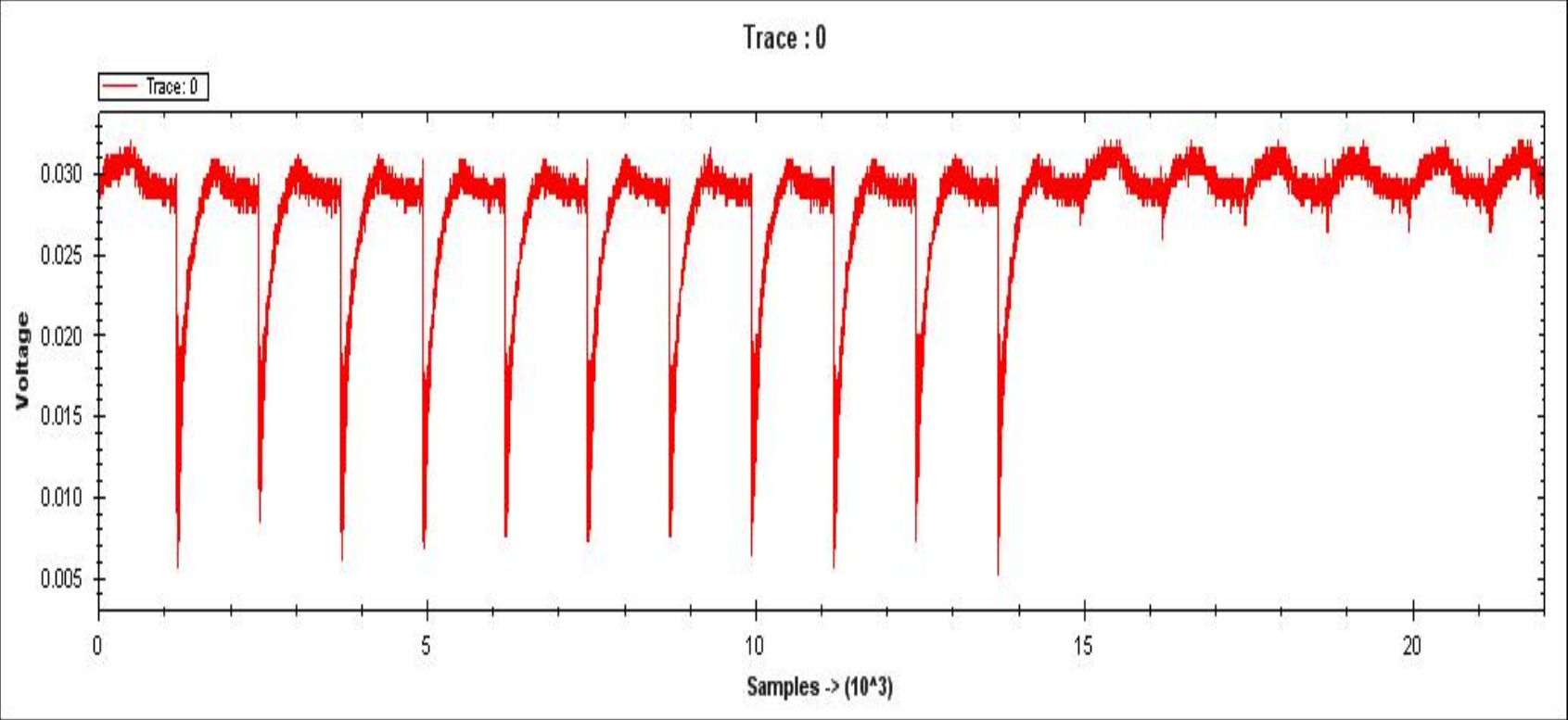
## 1. Measurement:

- Make power consumption measurement of about 1000 AES operations, 100000 data points/trace,
- Save (Ciphertext<sub>i</sub>, trace<sub>i</sub>)

## 2. Attack:

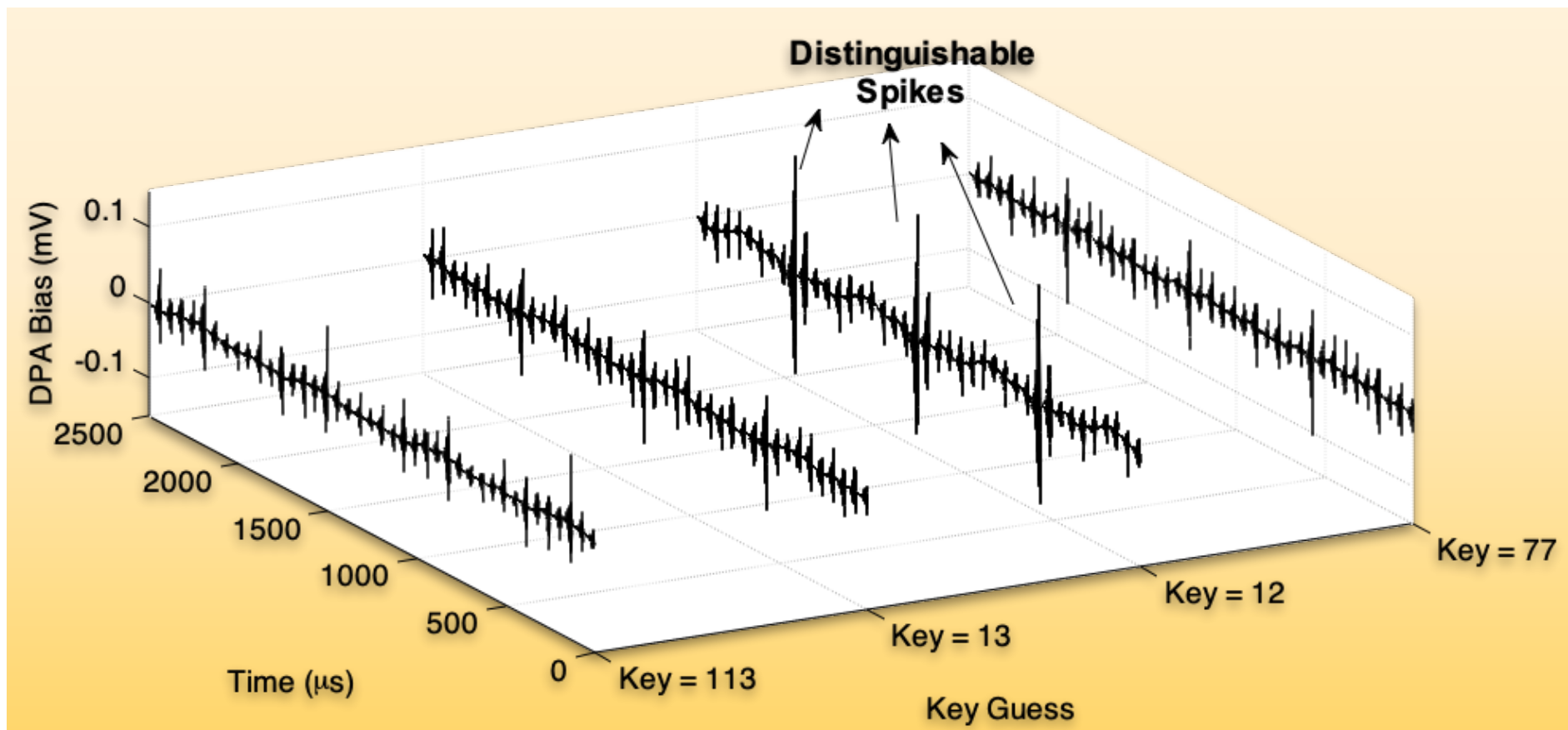
- Target an S-Box in the last round (say the  $j$ -th S-Box)
  - A. Guess a key for an S-box of last round (8 bit key, so total 256 guesses possible)
  - B. Partially decrypt one byte of each ciphertext with the guessed key till the input of the last round S-Box. That is compute:  $S_j = S^{-1}(C_j \oplus k_j^g)$
  - C. Divide the traces into 2 groups based on the LSB of  $S_j$
  - D. Calculate the **average trace** of each group
  - E. Calculate the difference of two **average traces**
  - F. **Correct key guess** → **spikes in the differential curve**
- Repeat A-F for other S-boxes

# Attacking AES



|       |         |         |     |         |
|-------|---------|---------|-----|---------|
| CT[0] | T[0][0] | T[0][1] | ... | T[0][m] |
| CT[1] | T[1][0] | T[1][1] | ... | T[1][m] |
| CT[2] | T[2][0] | T[2][1] | ... | T[2][m] |
| ...   | ...     | ...     | ... | ...     |
| CT[n] | T[n][0] | T[n][1] | ... | T[n][m] |

# Attacking AES



SBOX – 11

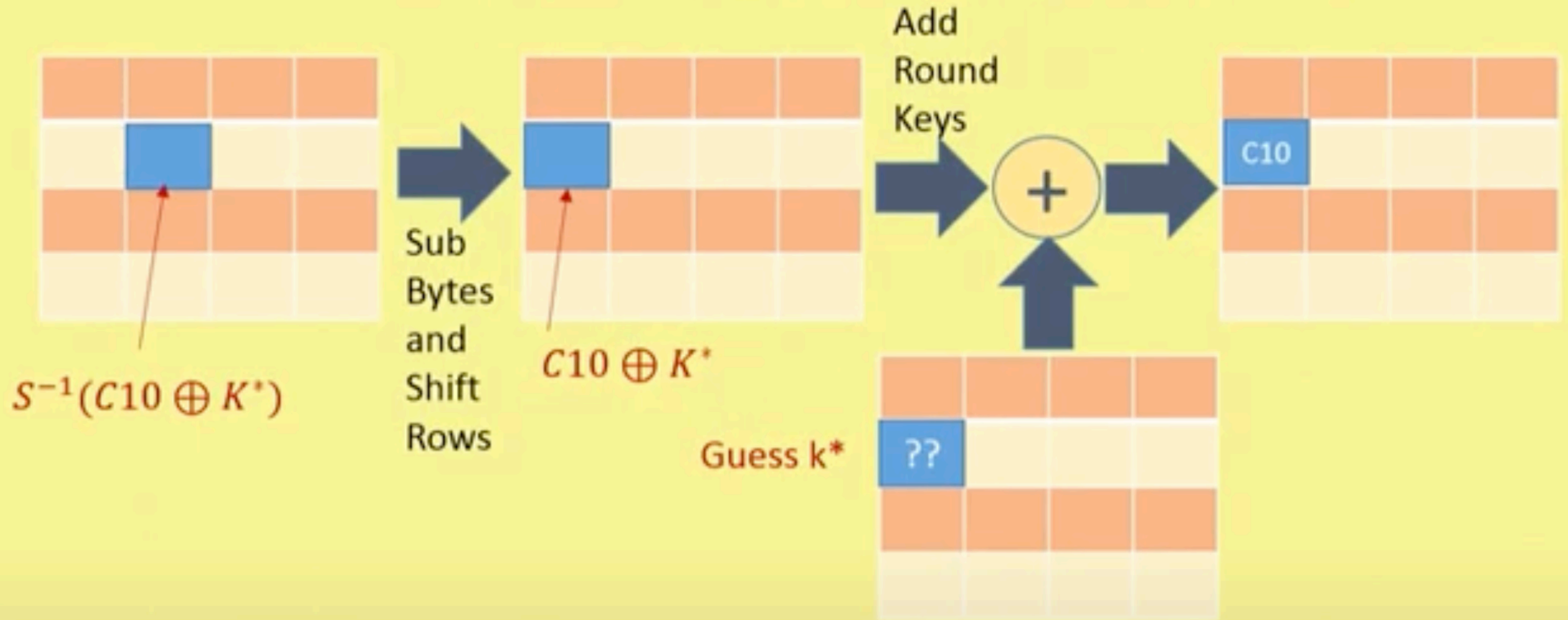
BIT – 8 TRACE COUNT = 15,000, FPGA implementation

# Attacking AES

- **DPA selection function:**  $D(C, b, k^g)$  is defined as computing the value of the
  - $b^{\text{th}}$  output bit, depending upon
    - $C$ : Ciphertext
    - $k^g$  is the guessed key for the S-Box
- In the attack,  $D = S^{-1}(C_j \oplus k_j^g)$
- If  $k^g$  is a wrong guess then  $b$  is correctly evaluated only for half of the ciphertexts (randomly).
  - **Thus for large number of points, the difference between average traces is close to 0**
  - **In other words, distribution of both the bins will be the same**
- But if  $k^g$  is a correct guess, then  $b$  is correctly evaluated for all the ciphertexts.

**Principle:** If  $K_s$  is wrongly guessed,  $D$  behaves like a random guess. Thus for a large number of sample points,  $\Delta[1..k]$  tends to zero. But if its correct, the differential will be non-zero and show spikes when  $D$  is correlated with the value being processed.

# Attacking AES



$$D(C_{10}, b = 0, K_{10}) = S^{-1}(C_{10} \oplus K^*)|_{b=0}$$

# Attacking AES

- **Differential Trace:** It is a  $m$  sample trace denoted as  $\Delta_D$ , where,

$$\Delta_D[j] = \frac{\sum_{i=0}^{n-1} D(C_i, b, K_g) T[i][j]}{\sum_{i=0}^{n-1} D(C_i, b, K_g)} - \frac{\sum_{i=0}^{n-1} (1 - D(C_i, b, K_g)) T[i][j]}{\sum_{i=0}^{n-1} (1 - D(C_i, b, K_g))}$$

- **Note:**  $C_i$  is a particular byte of the  $i$ -th ciphertext.

# Attacking AES

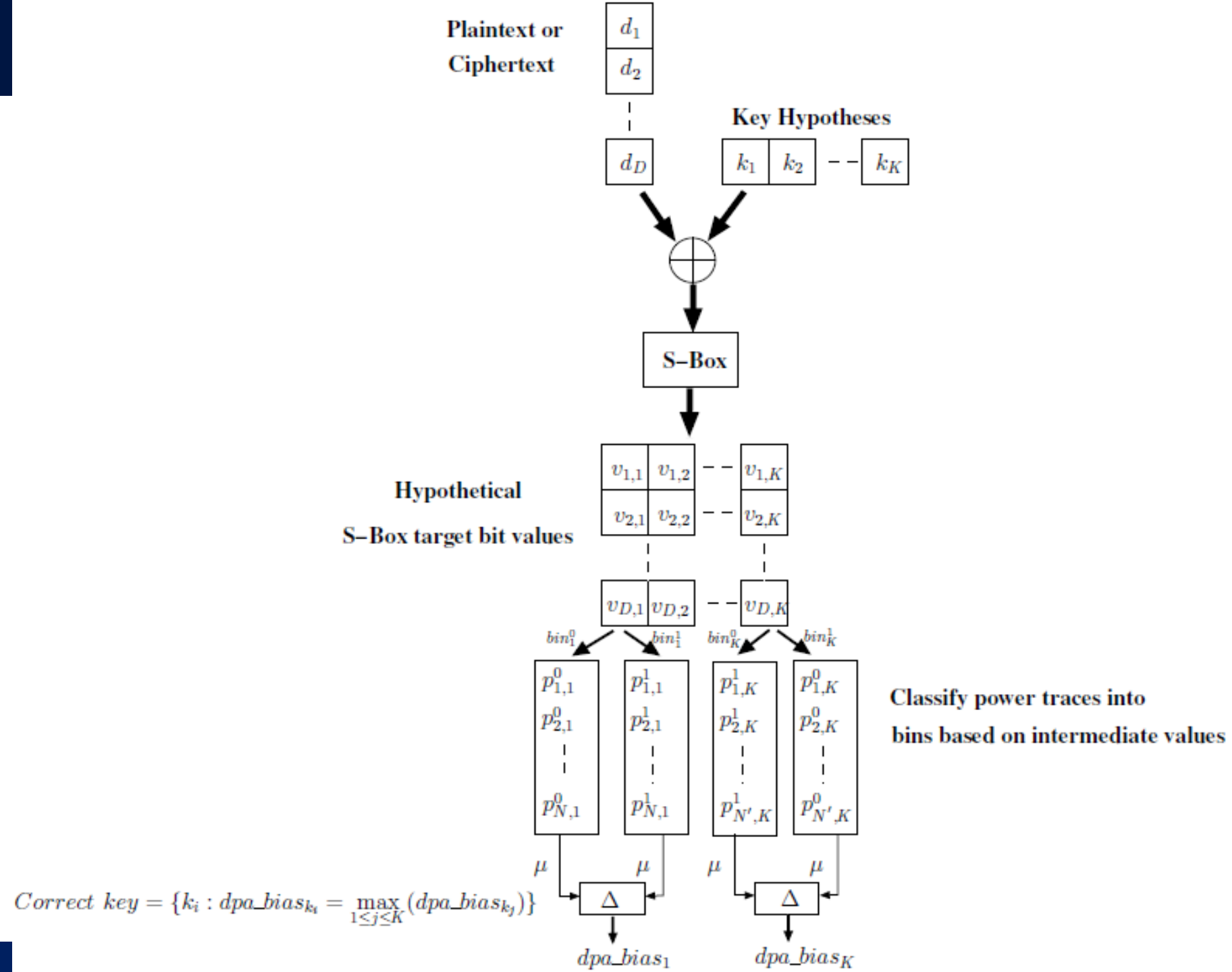
- Why does the attack work?

- It's not only the data dependency. But it also depends on the mathematics of AES
- **DPA selection function:**  $D(C, b, k^g)$  is defined as computing the value of the
  - $b^{\text{th}}$  output bit, depending upon
    - $C$ : Ciphertext
    - $k^g$  is the guessed key for the S-Box
- In the attack,  $D = S^{-1}(C_j \oplus k_j^g)$
- If  $k^g$  is a wrong guess then  $b$  is correctly evaluated only for half of the ciphertexts (randomly).
  - **Thus for large number of points, the difference between average traces is close to 0**
  - **In other words, distribution of both the bins will be the same**
- But if  $k^g$  is a correct guess, then  $b$  is correctly evaluated for all the ciphertexts.
- **Note:** The non-linearity of the S-Boxes play an important role here.

**Principle:** If  $K_s$  is wrongly guessed,  $D$  behaves like a random guess. Thus for a large number of sample points,  $\Delta[1..k]$  tends to zero. But if its correct, the differential will be non-zero and show spikes when  $D$  is correlated with the value being processed.



# Attacking AES



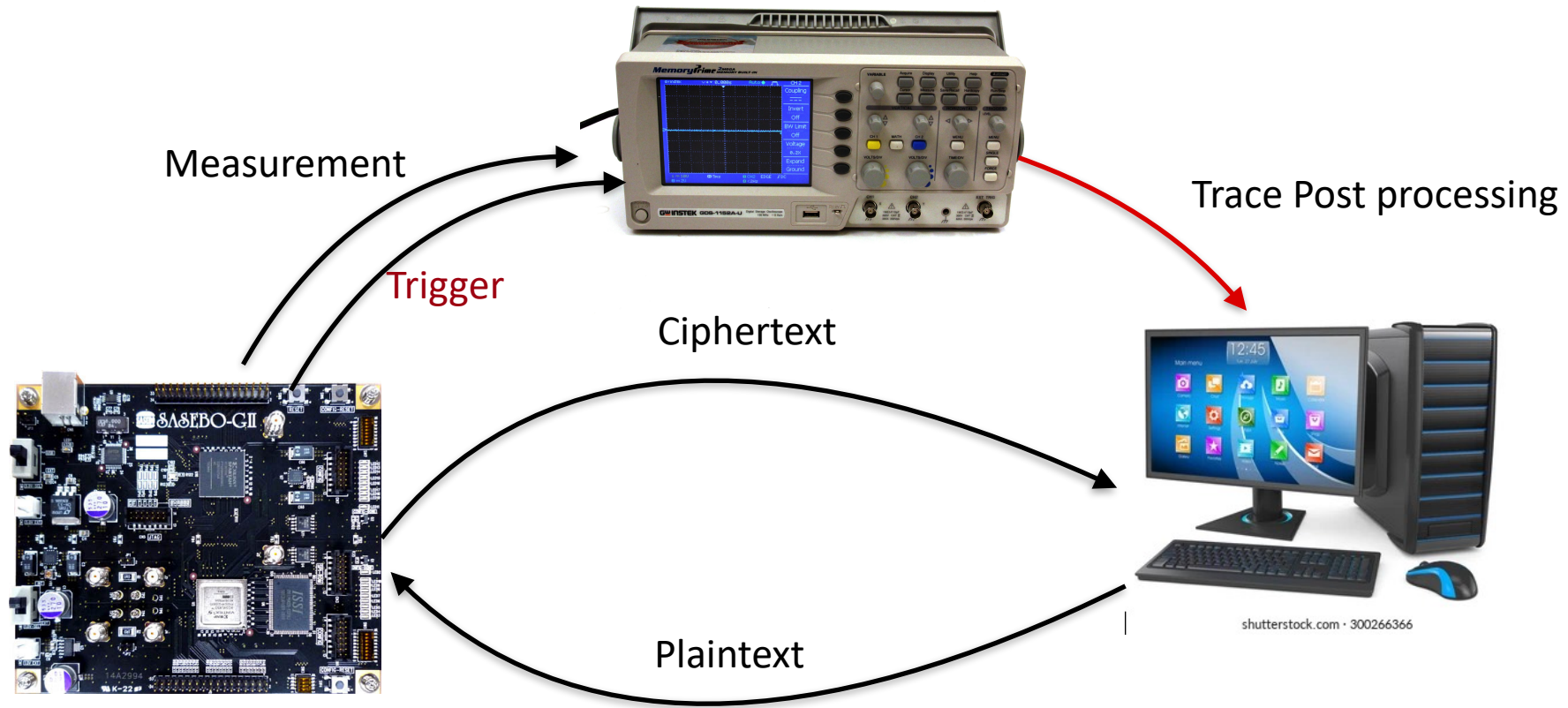
# Attacking AES: Attack Complexity

- **What is the attack complexity**
  - Say you are given  $n$  number of traces
  - How much further computation you need to perform?

# Attacking AES: Attack Complexity

- **Observe:** We are targeting the last round of AES and attacking each S-Box separately.
  - Divide and Conquer
  - Attack complexity: always  $2^8$
  - Trace complexity: depends
- **Observe:** The attack don't need plaintext knowledge

# Leakage Models



# Differential Power Analysis

- **Power Trace:** A set of power consumptions across a cryptographic process
  - 1 millisecond operation sampled at 5MHz yield a trace with 5000 points.
- **Measurement:**
  - Usually requires to adhere a signal bandwidth and frequency.
    - We do quite high frequency sampling — e.g. for a 65 Mhz microcontroller or 150 Mhz FPGA implementation, we do sampling at 5 GS/sec.
  - Number of samples is an important point. More samples results in better attack
  - **Trigger:** An important constraint
    - Without reliable trigger, the traces will be **misaligned**. — attack will be difficult
    - Most of the practical implementations does not provide a very reliable trigger
      - So, some realignment of traces are needed
        - Today, DL algorithms are great for this purpose — does automatic realignment.

# Attacking AES

## 1. Measurement:

- Make power consumption measurement of about 1000 AES operations, 100000 data points/trace,
- Save (Ciphertext<sub>i</sub>, trace<sub>i</sub>)

## 2. Attack:

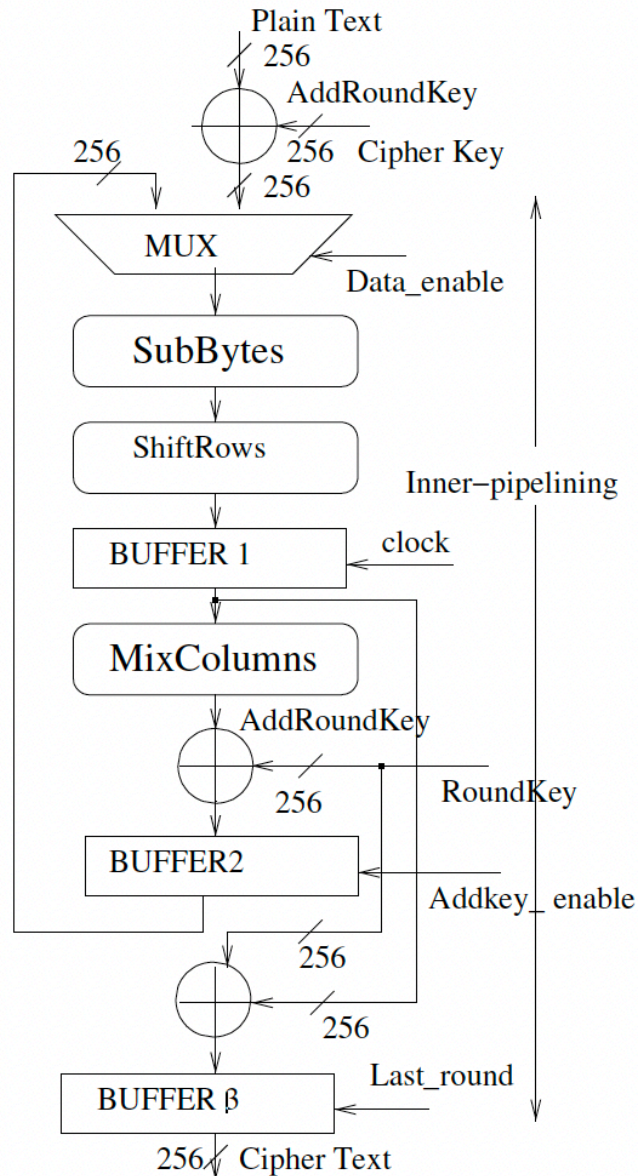
- Target an S-Box in the last round (say the  $j$ -th S-Box)
  - A. Guess a key for an S-box of last round (8 bit key, so total 256 guesses possible)
  - B. Partially decrypt one byte of each ciphertext with the guessed key till the input of the last round S-Box. That is compute:  $S_j = S^{-1}(C_j \oplus k_j^g)$
  - C. Divide the traces into 2 groups based on the LSB of  $S_j$
  - D. Calculate the **average trace** of each group
  - E. Calculate the difference of two **average traces**
  - F. **Correct key guess** → **spikes in the differential curve**
- Repeat A-F for other S-boxes

# Differential Power Analysis

- **Power Trace:** A set of power consumptions across a cryptographic process
  - 1 millisecond operation sampled at 5MHz yield a trace with 5000 points.
- **Leakage Model:** Hypothetical model relating the leakage with the internal states of the target algorithm.
  - For AES the internal state is a 128-bit value.
  - **Hamming Weight Model:** The power consumption is proportional to the Hamming weight (count of 1's) of the state.
  - **Hamming Distance Model:** The power consumption is proportional to the Hamming distance between the state in two consecutive clock cycles. — FPGA/ASICs
  - More complex models are possible...
  - Used to simulate leakage and also in some attacks.

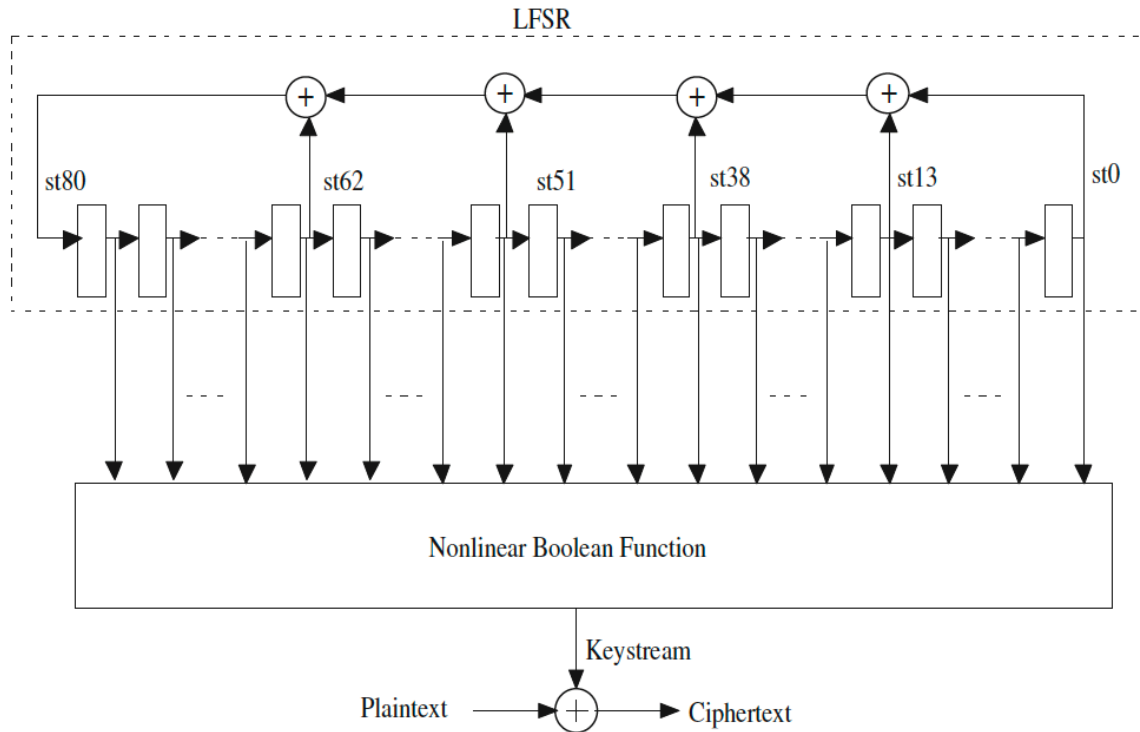


# Differential Power Analysis

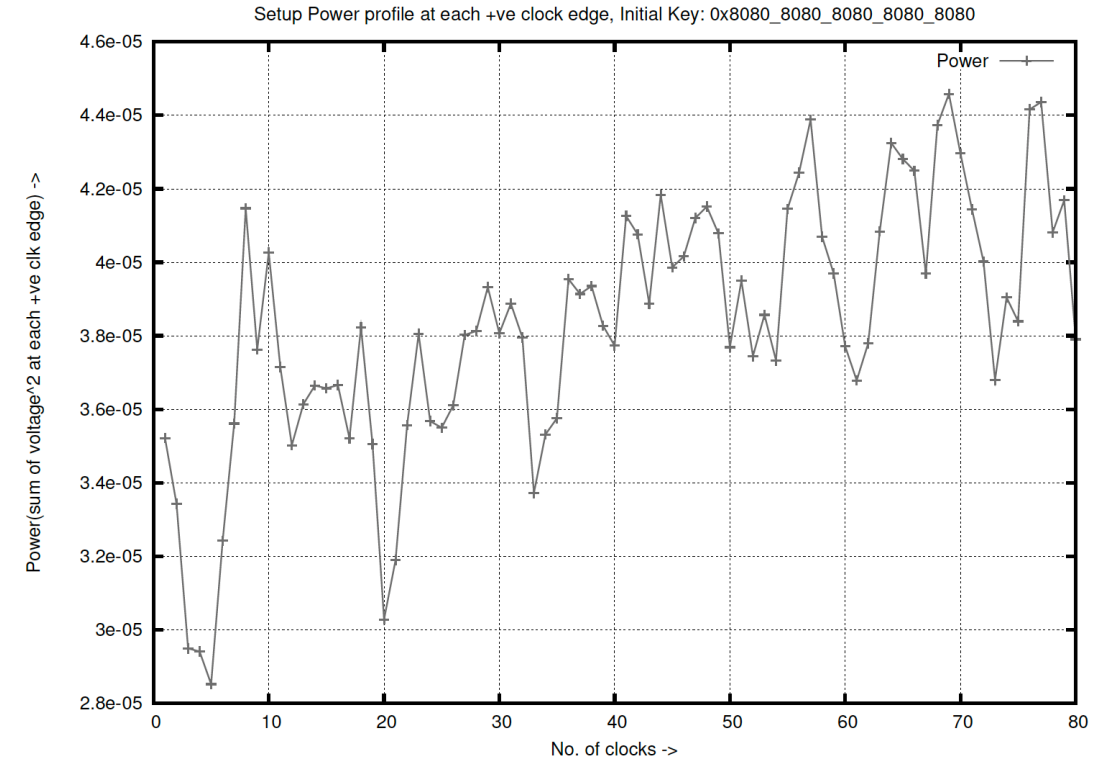


- Common hardware implementation of a block cipher
- We consider the state of the circuit at time instance  $t$  — you can consider it as one time point in the x-axis of the trace.
- Let this state be  $v_t$
- Hamming weight is number of 1's in  $v_t$ .
- Hamming distance is  $\text{HW}(v_t \oplus v_{t-1})$ .
  - Why?
  - After all power consumption is a result of transition  $v_{t-1} \rightarrow v_t$
  - So , HW is relatively inaccurate, but works well for software
    - Registers often start from a specific initialisation value.

# Differential Power Analysis

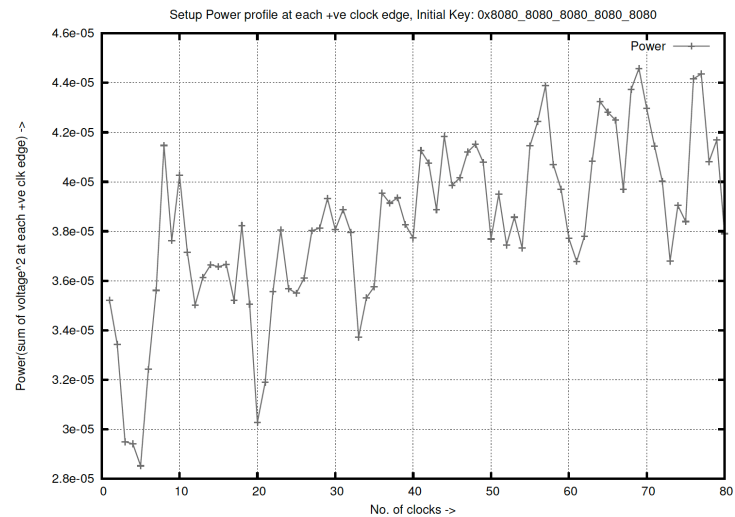


Linear Feedback Shift Register

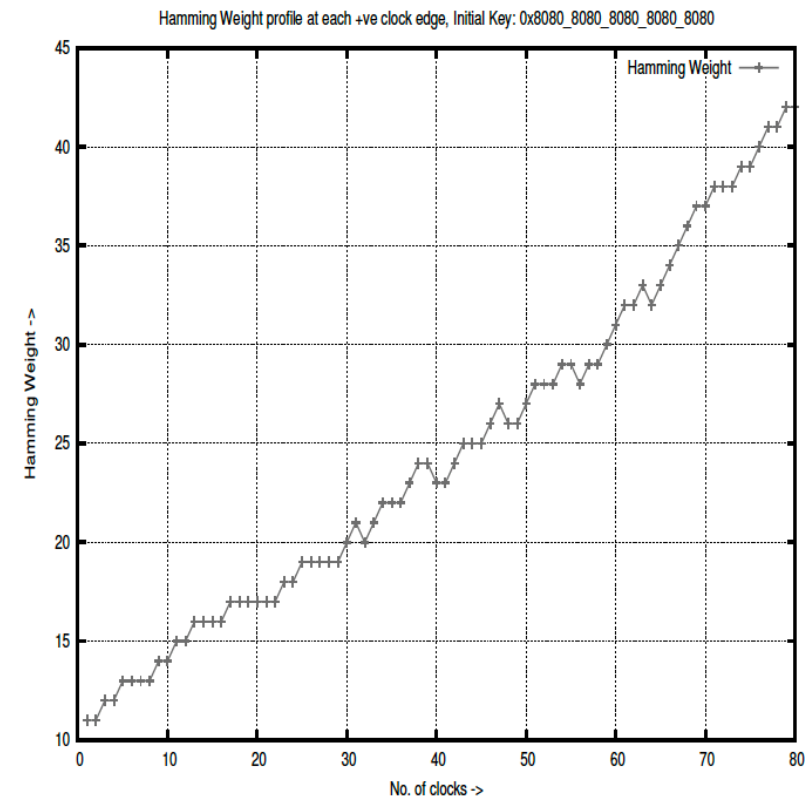
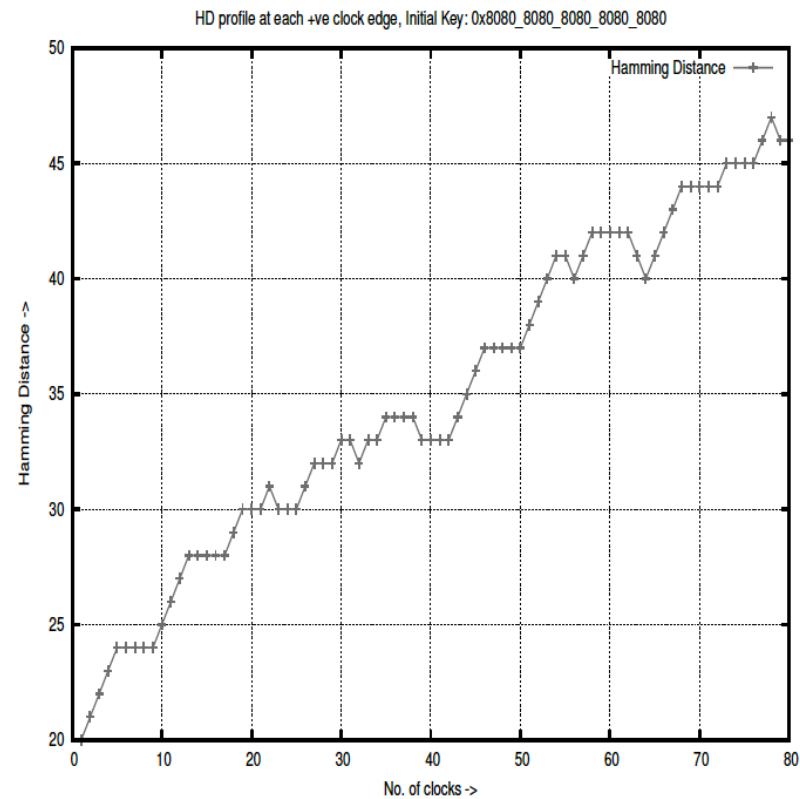


Actual Trace from an FPGA

# Differential Power Analysis



Actual Trace from :



Actual Trace from an FPGA

# Why do Gates Leak?

| a   | b   | y   | Energy    |
|-----|-----|-----|-----------|
| 0→0 | 0→0 | 0→0 | $E_{0→0}$ |
| 0→0 | 0→1 | 0→0 | $E_{0→0}$ |
| 0→0 | 1→0 | 0→0 | $E_{0→0}$ |
| 0→0 | 1→1 | 0→0 | $E_{0→0}$ |
| 0→1 | 0→0 | 0→0 | $E_{0→0}$ |
| 0→1 | 0→1 | 0→1 | $E_{0→1}$ |
| 0→1 | 1→0 | 0→0 | $E_{0→0}$ |
| 0→1 | 1→1 | 0→1 | $E_{0→1}$ |
| 1→0 | 0→0 | 0→0 | $E_{0→0}$ |
| 1→0 | 0→1 | 0→0 | $E_{0→0}$ |
| 1→0 | 1→0 | 1→0 | $E_{1→0}$ |
| 1→0 | 1→1 | 1→0 | $E_{1→0}$ |
| 1→1 | 0→0 | 0→0 | $E_{0→0}$ |
| 1→1 | 0→1 | 0→1 | $E_{0→1}$ |
| 1→1 | 1→0 | 1→0 | $E_{1→0}$ |
| 1→1 | 1→1 | 1→1 | $E_{1→1}$ |

- Consider an AND gate  $y = ab$
- 4 different energy levels due to transition of the gate  $E_{0→0}$ ,  $E_{0→1}$ ,  $E_{1→0}$ ,  $E_{1→1}$
- We estimate  $E(q = 0)$ , and  $E(q = 1)$  which are the average energy levels when  $q = 0$ , and  $q = 1$ , respectively.

$$E(q = 0) = \frac{9E_{0→0} + 3E_{1→0}}{12}$$

$$E(q = 1) = \frac{3E_{0→1} + E_{1→1}}{4}$$

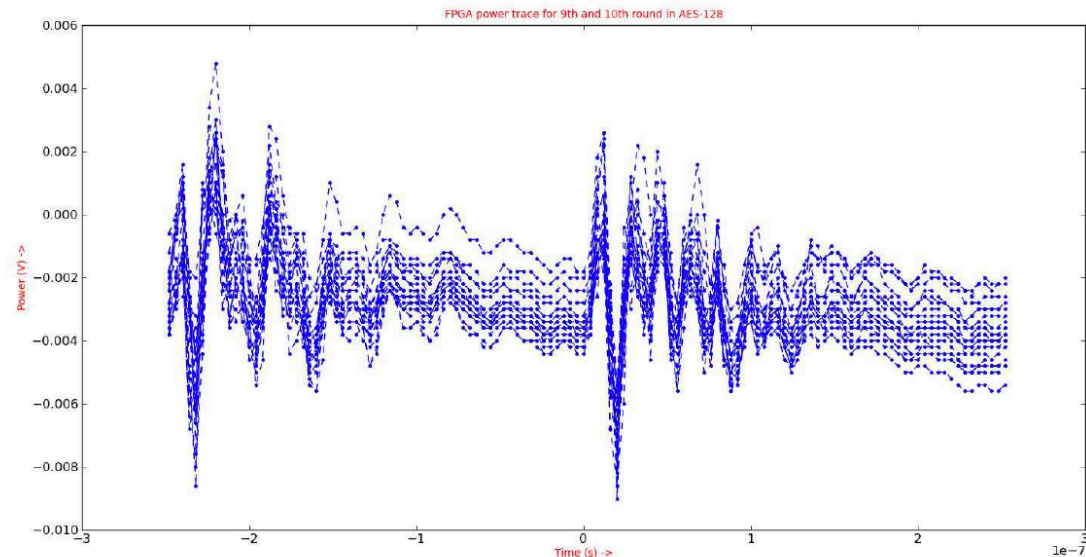
- So  $E(q = 0) \neq E(q = 1)$

Make Some Noise

# Measurement Noise

- **Importance of high-quality measurement**
  - Noise increases the required trace count.
- These fluctuations are due to **electrical noise**, caused due to power supply, clock generator, conduction and radiation emissions from the components

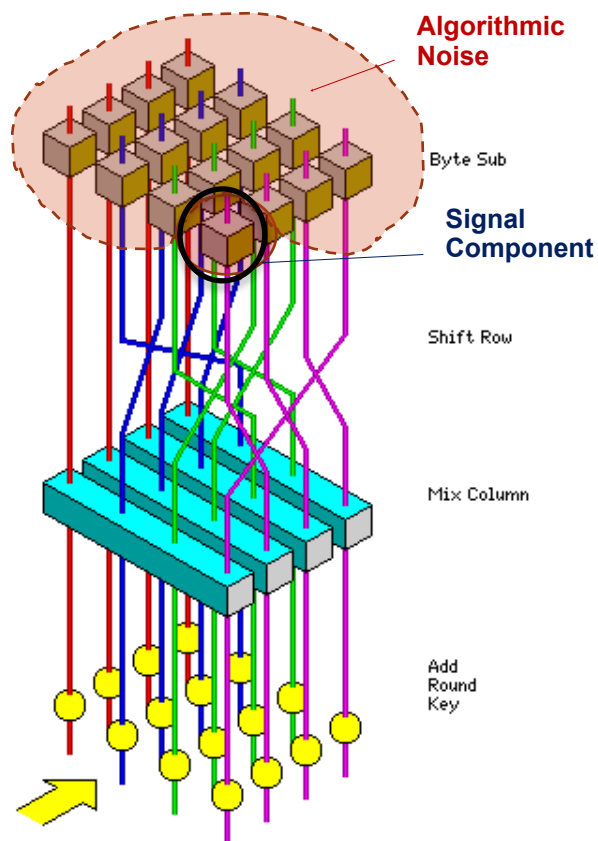
AES-128 Encryption with same plaintext and key, but resulting in different power traces.



*Simple Power Model.* Let  $t$  denote the time, and  $\mathcal{N}(t)$  be a normal distributed random variable which represents the noise components. Let  $f(g, t)$  denote the power consumption of gate  $g$  at the time  $t$ . Then a simplified power model for the power consumption is the function

$$P(t) = \sum_g f(g, t) + \mathcal{N}(t)$$

# Algorithmic Noise



- Algorithmic or switching noise occurs because of contributions of logic cells to the power consumption, which are not under attack.
- The power trace corresponds to the total power consumption of the circuit.
- However, in the attack we target only a small part (see black circle in fig.) to reveal a portion of the key.
- The power consumption from all the other parts (see blue shaded portion in fig.) form the algorithmic noise.
- This would be more in a parallel implementation, compared to a serialized implementation.

**Fortunately, our statistical attack can handle both**



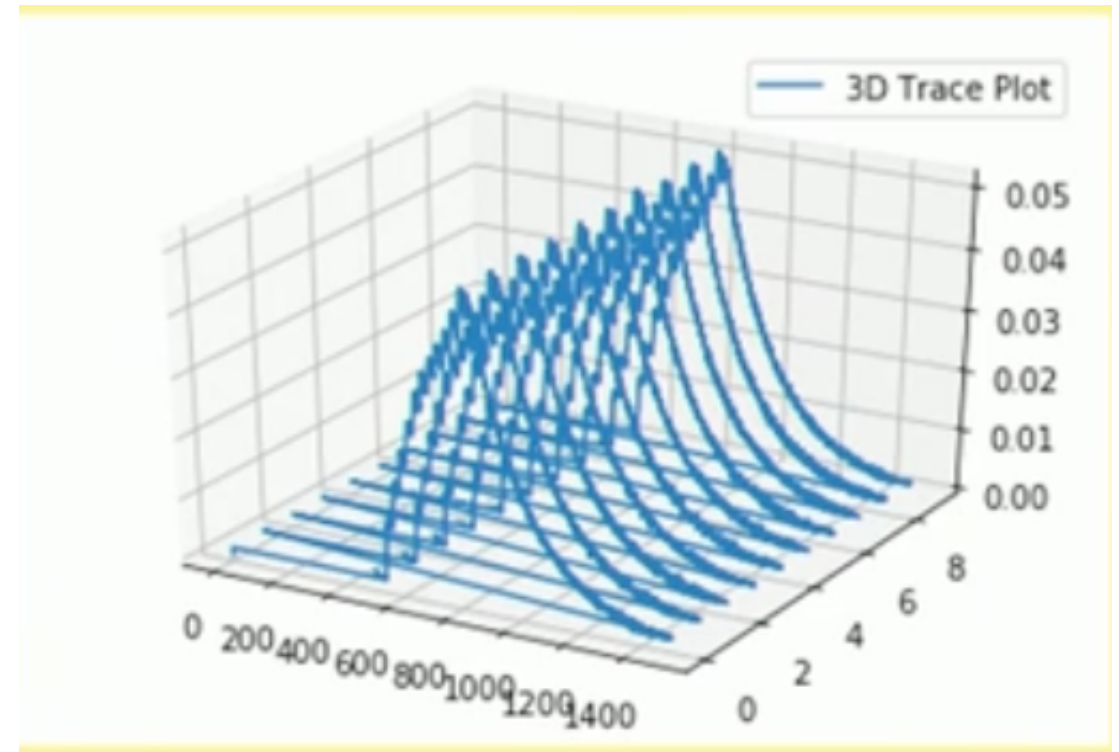
# Statistical Analysis of Power Traces

- **Fact:** For fixed operation and fixed value, the power consumption is fixed. But there will be some additive noise.
- **Fortunately, this noise is Gaussian.**
  - Let us assume that the mean is  $\mu$  and variance is  $\sigma^2$
  - Noise mainly results in the variance...
- **Recall:** The gaussian probability distribution function

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}} \quad x \in \mathbb{R}$$

$$X \sim \mathcal{N}(\mu, \sigma)$$

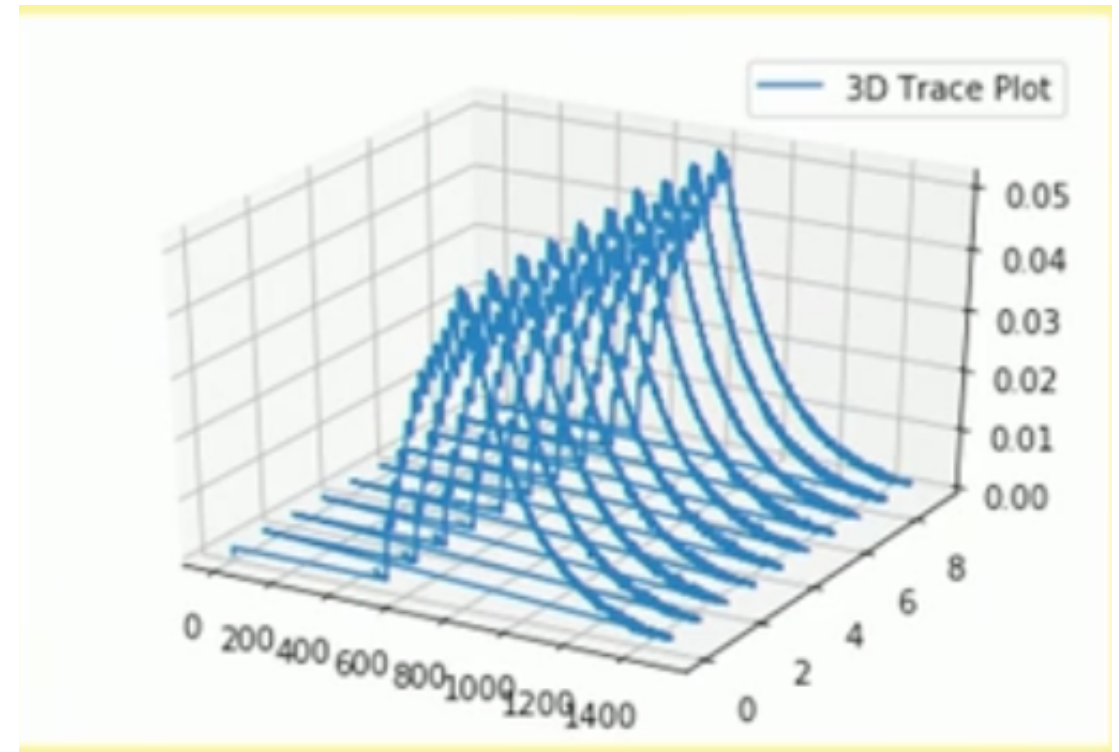
- If  $\mu = 0$ ,  $\sigma = 1$ , we call it a standard normal distribution.
- The CDF of standard normal is denoted as  $\phi(x)$ .



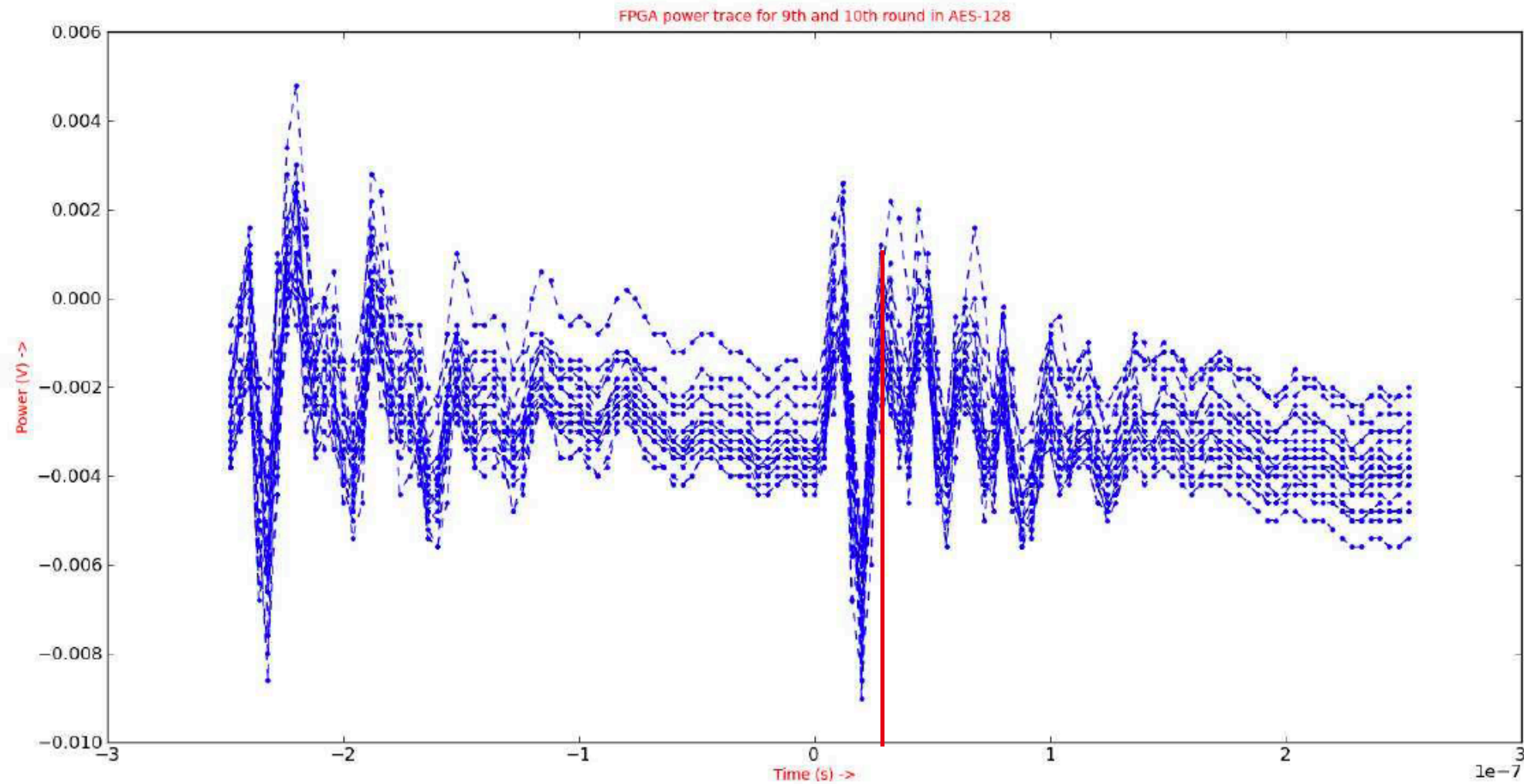


# Statistical Analysis of Power Traces

- **Recall:** Sampling distribution
  - An *unbiased estimator* for  $\mu$  is  $\bar{x}$  — the sample mean
  - An unbiased estimator for  $\sigma^2$  is  $s^2$  — the sample variance (with n-1 correction factor, of course)
  - **Unbiased estimator:** The expected value of the estimator is equal to the true value of the parameter; e.g.  $E(\bar{x}) = \mu$



# Statistical Analysis of Power Traces

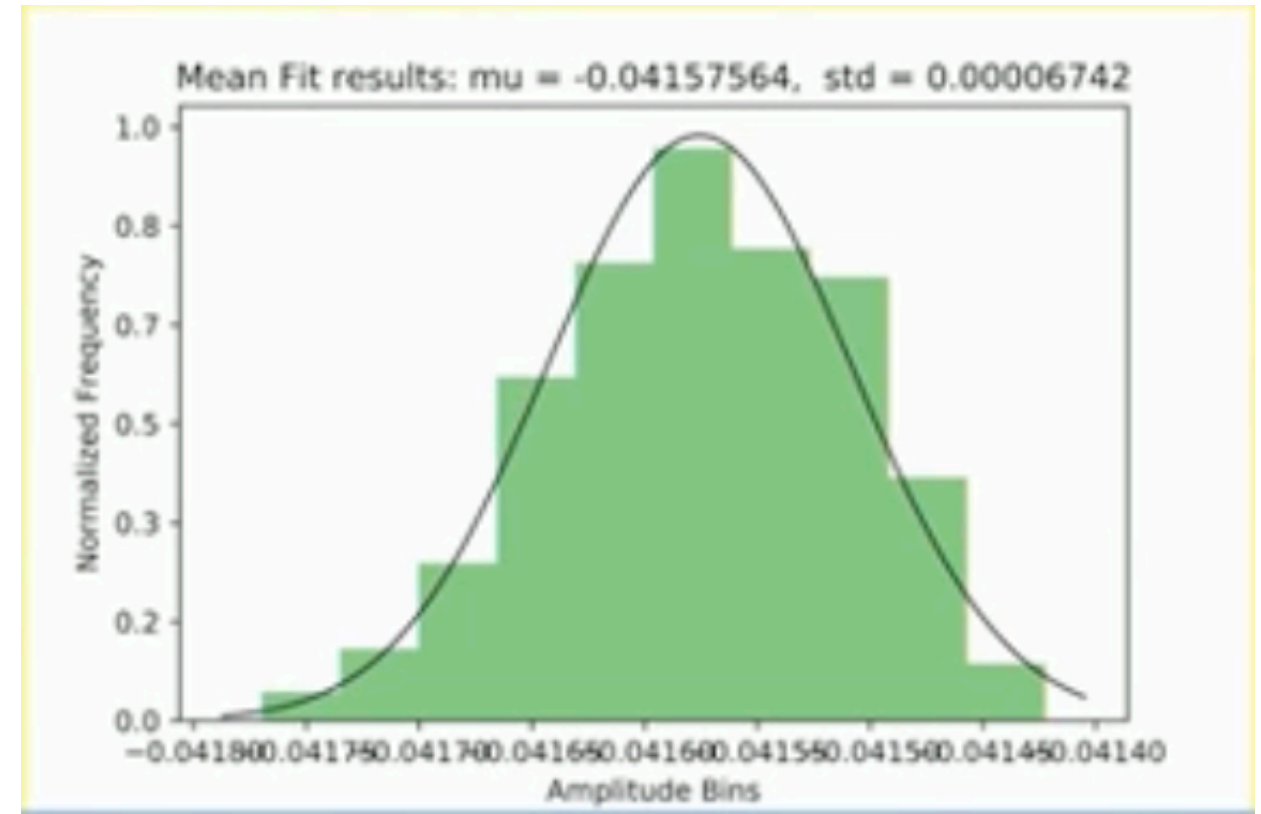
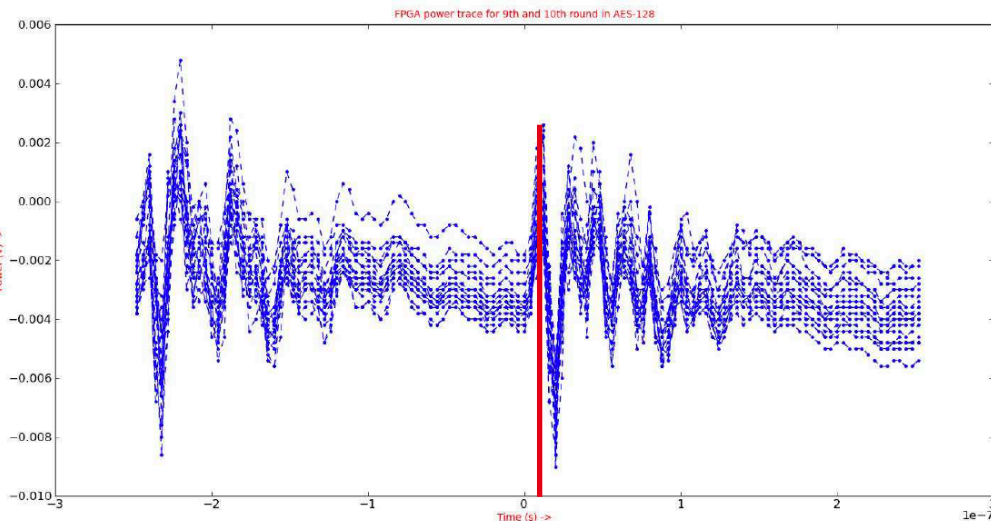


# Statistical Analysis of Power Traces

- **A point in the power trace is represented as:**
  - $P(t) = \sum_g f(g, t) + \mathcal{N}(\mu, \sigma, t)$
- **POI (point of interest) in the trace:** The point(s) in the trace which are the best for distinguishing the correct key from the wrong one.

*Simple Power Model.* Let  $t$  denote the time, and  $\mathcal{N}(t)$  be a normal distributed random variable which represents the noise components. Let  $f(g, t)$  denote the power consumption of gate  $g$  at the time  $t$ . Then a simplified power model for the power consumption is the function

$$P(t) = \sum_g f(g, t) + \mathcal{N}(t)$$



Histogram of a point in power trace for same pt and key

# Sampling Distribution and Estimation of Parameters

- A point in the power trace is represented as:

$$P(t) = \sum_g f(g, t) + \mathcal{N}(\mu, \sigma, t)$$

- The question is, how does it affect the attacks??
  - **You cannot determine the exact HW/HD**
- How to manage this noise??
  - **By modelling it**
  - **Taking mean is a good idea as it is done for DoM**
- **Sampling distribution:**
  - **How do we estimate  $\mu, \sigma$ ?**
  - We can only access some samples from the true distribution and therefore we can only compute the sample mean, variance  $\bar{x}, s^2$
  - They are indeed unbiased estimators.
  - But then, there are a few “statistical” points..

# Sampling Distribution and Estimation of Parameters

- **Sampling distribution:**
  - **How do we estimate  $\mu, \sigma$ ?**
  - We can only access some samples from the true distribution and therefore we can only compute the sample mean, variance  $\bar{x}, s^2$
  - They are indeed unbiased estimators.
  - But then, there are a few “statistical” points..
  - Each of  $\bar{x}, s^2$  has their own distribution... the **sampling distributions**

# Sampling Distribution and Estimation of Parameters

- **Sampling distribution:**

- Let us consider the sample mean  $\bar{X}$  as a random variable
- Note that the population is normally distributed.
- It can be shown that  $\bar{X} \sim \mathcal{N}(\mu, \sigma/\sqrt{n})$ 
  - You can prove it !! — try at home
- **Central Limit Theorem:** Let  $X_1, X_2, \dots, X_n$  be independent and identically distributed random variables with  $E(X) = \mu < \infty$ , and variance  $0 < \sigma^2 < \infty$ . Then:

$$\lim_{n \rightarrow \infty} Pr[Z_n \leq x] = \phi(x), \forall x \in \mathbb{R}$$

$$Z_n = \frac{(X_1 + X_2 + X_3 + \dots + X_n) - n\mu}{\sqrt{n}\sigma} = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

This means, whatever be the population's distribution, the sample mean will approximately follow the normal distributions for sufficiently large number of sample ( $n > 30$ , typically).

- **Law of large numbers:**  $\lim_{n \rightarrow \infty} \frac{X_1 + X_2 + \dots + X_n}{n} \rightarrow \mu$ , that is  $\bar{X}$  is unbiased estimator.

# Sampling Distribution and Estimation of Parameters

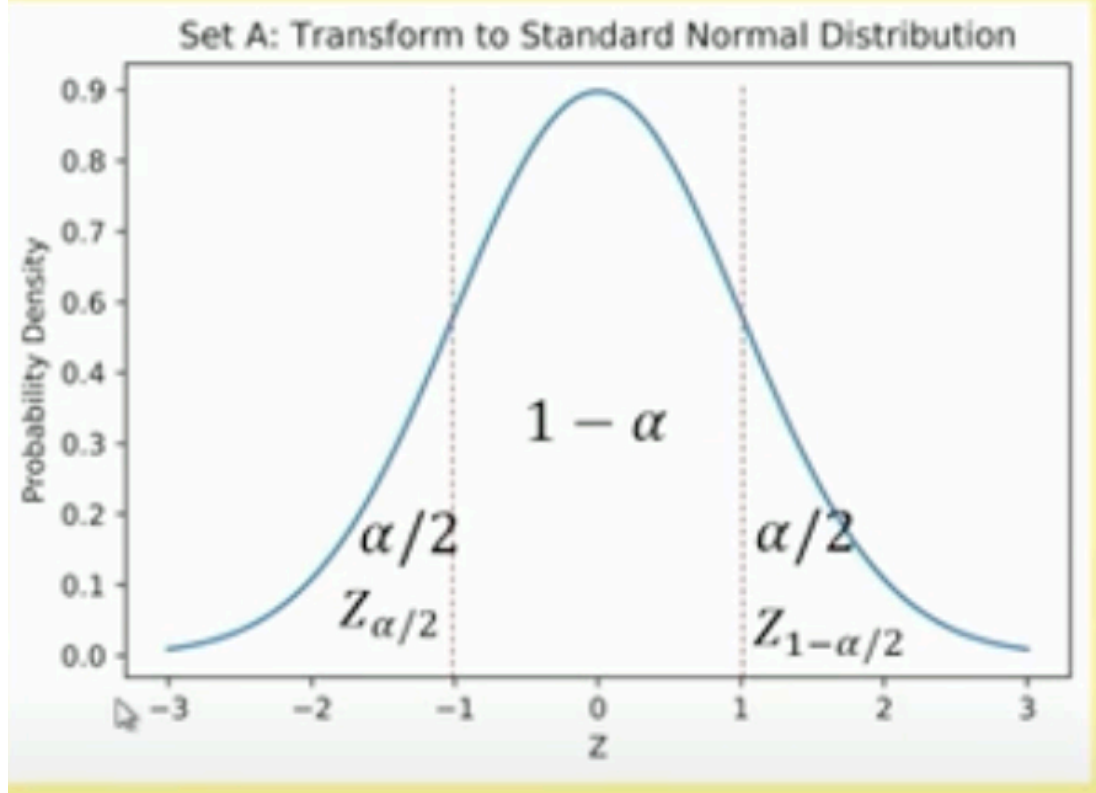
- Sampling distribution:
  - So, the more traces you take, the average becomes closer to the actual mean.
  - That means, if you have noise with  $\mu = \text{const}$ , with more traces you can significantly reduce its impact.
  - **But, how do you know that how many traces take you closest to the mean???**

# Hypothesis Testing

- **Confidence Interval**
  - **Determines, how close a certain parameter estimate is to its actual value**
  - When we say that we have a .99 confidence interval for  $\mu$ , actual  $\mu$  lies in that interval, and that happens 99% of the time you calculate  $\bar{x}$  and the interval.
  - **How do we find out these intervals?**
    - We shall see soon..
- **Hypothesis Testing**
  - We define a hypothesis and test whether it is true or not depending on the samples
  - Suppose we hypothesize that  $\mu = \mu_0$ 
    - You set a **null hypothesis**:  $H_0 : \mu = \mu_0$
    - You set an **alternative hypothesis**:  $H_1 : \mu \neq \mu_0$
    - **Two hypotheses must be exclusive.**
    - It can be something else as well, such as  $H_0 : \mu_1 \neq \mu_2$
- **Normal to Standard Normal**: If  $X \sim \mathcal{N}(\mu, \sigma)$  then  $Z = \frac{X - \mu}{\sigma} \sim \mathcal{N}(0,1)$



# Standard Normal Distribution



- Let us define an as an interval  $[-Z_{\alpha/2}, Z_{\alpha/2}]$
- $P(Z \leq Z_{\alpha}) = \phi(Z_{\alpha}) = \alpha$
- **Also, we can see that:**  $Z_{\alpha} = -Z_{1-\alpha}$
- The probability that a randomly sampled value of  $Z$  will lie in this interval is  $1 - \alpha$  —  
 $P(-Z_{\alpha/2} \leq Z \leq Z_{\alpha/2}) = 1 - \alpha,$
- The normal distribution tables provide numerically calculated values for  $\phi(Z_{\alpha})$  for different  $\alpha$

# Confidence Interval for $\mu$

$$\bar{X} \sim \mathcal{N}(\mu, \sigma/\sqrt{n})$$
$$\implies Z = (\bar{X} - \mu) \frac{\sqrt{n}}{\sigma}$$

- Therefore,

- $P(Z_{\alpha/2} \leq (\bar{X} - \mu) \cdot \frac{\sqrt{n}}{\sigma} \leq Z_{1-\alpha/2})$   
 $= P(\frac{\sigma}{\sqrt{n}} Z_{\alpha/2} \leq (\bar{X} - \mu) \leq \frac{\sigma}{\sqrt{n}} Z_{1-\alpha/2})$   
 $= P(\bar{X} - \frac{\sigma}{\sqrt{n}} Z_{1-\alpha/2} \leq \mu \leq \bar{X} - \frac{\sigma}{\sqrt{n}} Z_{\alpha/2}).$
- Using,  $Z_{\alpha/2} = -Z_{1-\alpha/2}$ , thus we continue:  
 $= P(\bar{X} - \frac{\sigma}{\sqrt{n}} Z_{1-\alpha/2} \leq \mu \leq \bar{X} + \frac{\sigma}{\sqrt{n}} Z_{1-\alpha/2}).$

- Given  $\bar{X} \sim \mathcal{N}(\mu, \sigma/\sqrt{n})$
- We need to compute “how good is the estimate of  $\mu$  for a given number of samples (traces)  $n$ ”
- If we can estimate  $\mu$  well, then we actually average out the noise and get close to the actual power model (HW/HD/ or something else).
- **Overall, we need to compute a confidence interval of  $\mu$ , based on the sample mean  $\bar{X}$ .**
- Informally, that means, we do not know the exact value of  $\mu$ , but we can say (based on  $\bar{X}$ ), what range it lies.  
•  $[\bar{X} - \frac{\sigma}{\sqrt{n}} Z_{1-\frac{\alpha}{2}}, \bar{X} + \frac{\sigma}{\sqrt{n}} Z_{1-\frac{\alpha}{2}}]$  is the  $(1 - \alpha)$  **confidence interval** for  $\mu$ .
- $\alpha$  is called the **error probability**

# Confidence Interval for $\mu$

- Overall,  $\mu$  lies in the interval

$$\left[\bar{X} - \frac{\sigma}{\sqrt{n}}Z_{1-\frac{\alpha}{2}}, \bar{X} + \frac{\sigma}{\sqrt{n}}Z_{1-\frac{\alpha}{2}}\right] \text{ with}$$

probability  $1 - \alpha$

- Say  $1 - \alpha = 0.99$ , then it means that if you take say 100 sets of n-samples,  $\mu$  will lie in the aforementioned range 99 times.
- It is the “confidence” on the interval finding mechanism, not in the found interval.

# Hypothesis Test

- Suppose, we want to test if  $\mu = \mu_0$ 
  - **null hypothesis:**  $H_0 : \mu = \mu_0$
  - **alternative hypothesis:**  $H_1 : \mu \neq \mu_0$
- $\mu$  is estimated from the sample mean  $\bar{X}$
- We check
  - If  $|\bar{X} - \mu_0|$  is greater than some predefined constant  $c$ , we reject  $H_0 : \mu = \mu_0$ .
  - Else, we accept  $H_0 : \mu = \mu_0$
- **False positive (Type I error):**
  - The null hypothesis is correct but got rejected;
  - $P(|\bar{X} - \mu_0| > c) = \alpha$  — is therefore called the error probability and  $1 - \alpha$  is the level of confidence.
- **False negative (Type II error):** The null hypothesis is wrong but got accepted.

# Estimating the Number of Traces to Estimate $\mu$

- Suppose, we want to estimate the mean with precision  $c = 0.01$

$$P(|\bar{X} - \mu_0| > c) = \alpha,$$

$$P(|\bar{X} - \mu_0| > c) = P\left(|\bar{X} - \mu_0| \frac{\sqrt{n}}{\sigma} > c \frac{\sqrt{n}}{\sigma}\right) = P\left(|Z| > c \frac{\sqrt{n}}{\sigma}\right) = 2P\left(Z > c \frac{\sqrt{n}}{\sigma}\right) = \alpha$$

$$\Rightarrow P\left(Z > c \frac{\sqrt{n}}{\sigma}\right) = \frac{\alpha}{2} \Rightarrow \mathbf{c \frac{\sqrt{n}}{\sigma} = Z_{1-\alpha/2}}$$

- Finally,

$$\mathbf{n = \frac{\sigma^2}{c^2} Z_{1-\alpha/2}^2}$$

# Estimating the Number of Traces for DoM

- Suppose,  $X \sim \mathcal{N}(\mu_X, \sigma)$ ,  $Y \sim \mathcal{N}(\mu_Y, \sigma)$  — let us assume the variance is the same. Let us also assume that we have  $n$  traces.

- $\bar{X} - \bar{Y} \sim \mathcal{N}(\mu_X - \mu_Y, \sigma \sqrt{\frac{n + n}{n^2}})$

- $Z = \frac{\bar{X} - \bar{Y} - (\mu_X - \mu_Y)}{\sigma \sqrt{\frac{2}{n}}}$  is the corresponding standard normal variable.

# Estimating the Number of Traces for DoM

- Confidence interval for  $(\mu_X - \mu_Y)$  for known  $\sigma$ :

$$P(Z_{\alpha/2} \leq Z \leq Z_{1-\alpha/2}):$$
$$\left[ \bar{X} - \bar{Y} - \frac{\sigma}{\sqrt{n/2}} Z_{1-\alpha/2}, \bar{X} - \bar{Y} + \frac{\sigma}{\sqrt{n/2}} Z_{1-\alpha/2} \right]$$

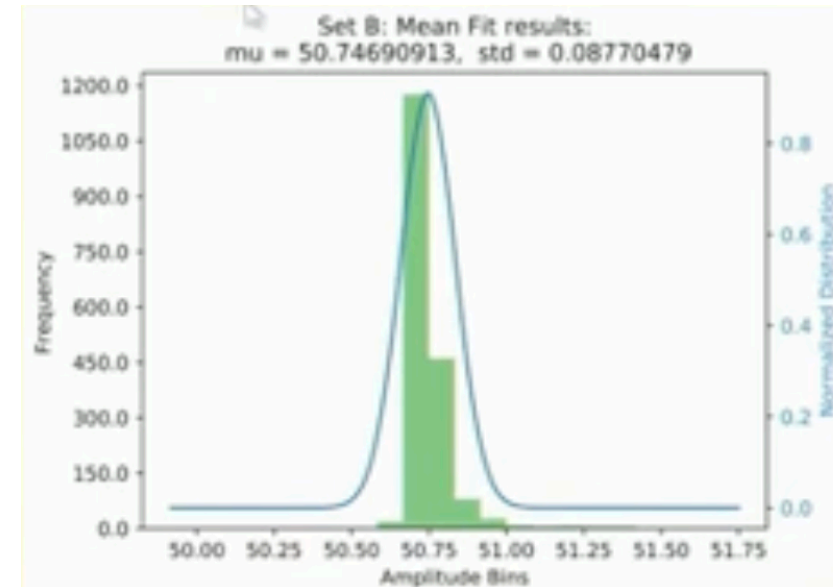
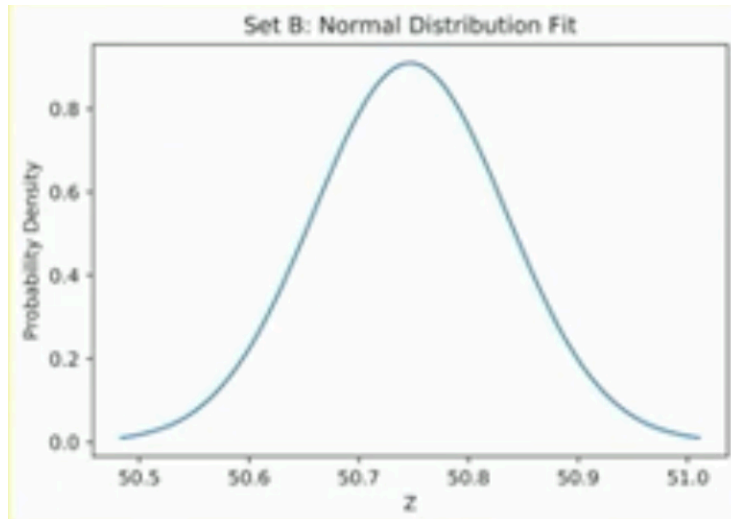
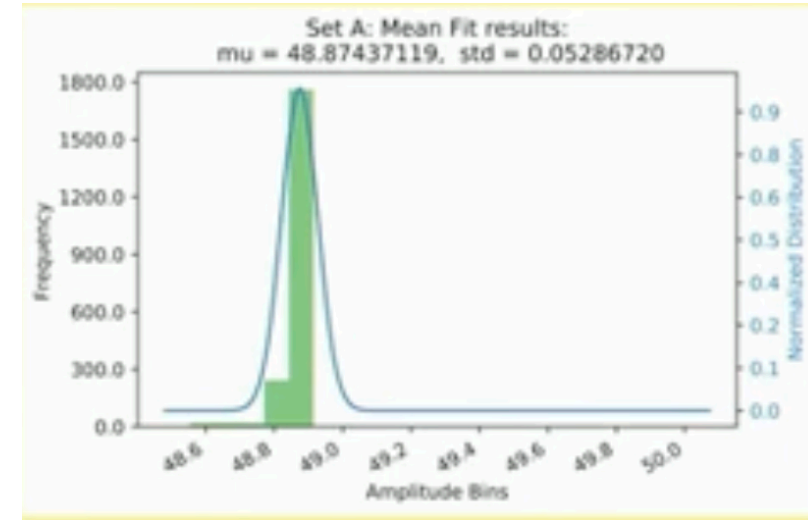
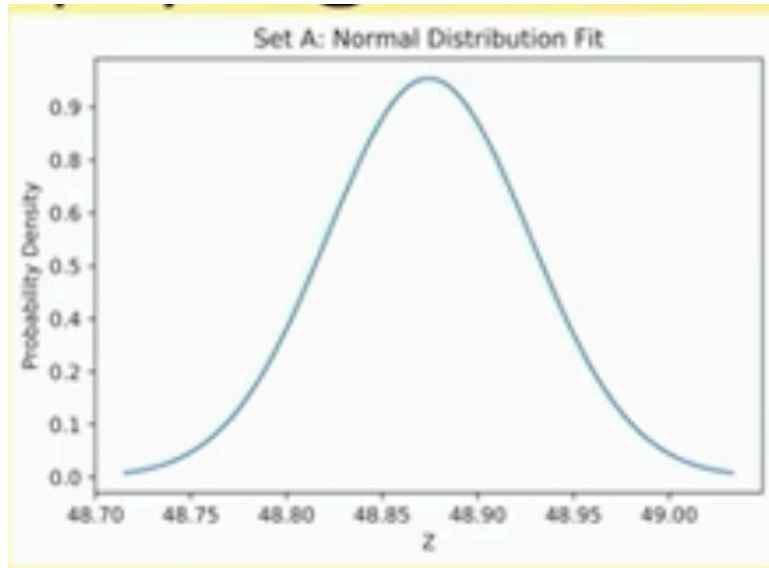
- **Null Hypothesis:**  $(\mu_X - \mu_Y) = 0$  , **Alternate Hypothesis:**  $(\mu_X - \mu_Y) \neq 0$

- We test if  $Z = \frac{\bar{X} - \bar{Y} - (\mu_X - \mu_Y)}{\sigma \sqrt{\frac{2}{n}}}$  is in the critical region.

- Therefore,

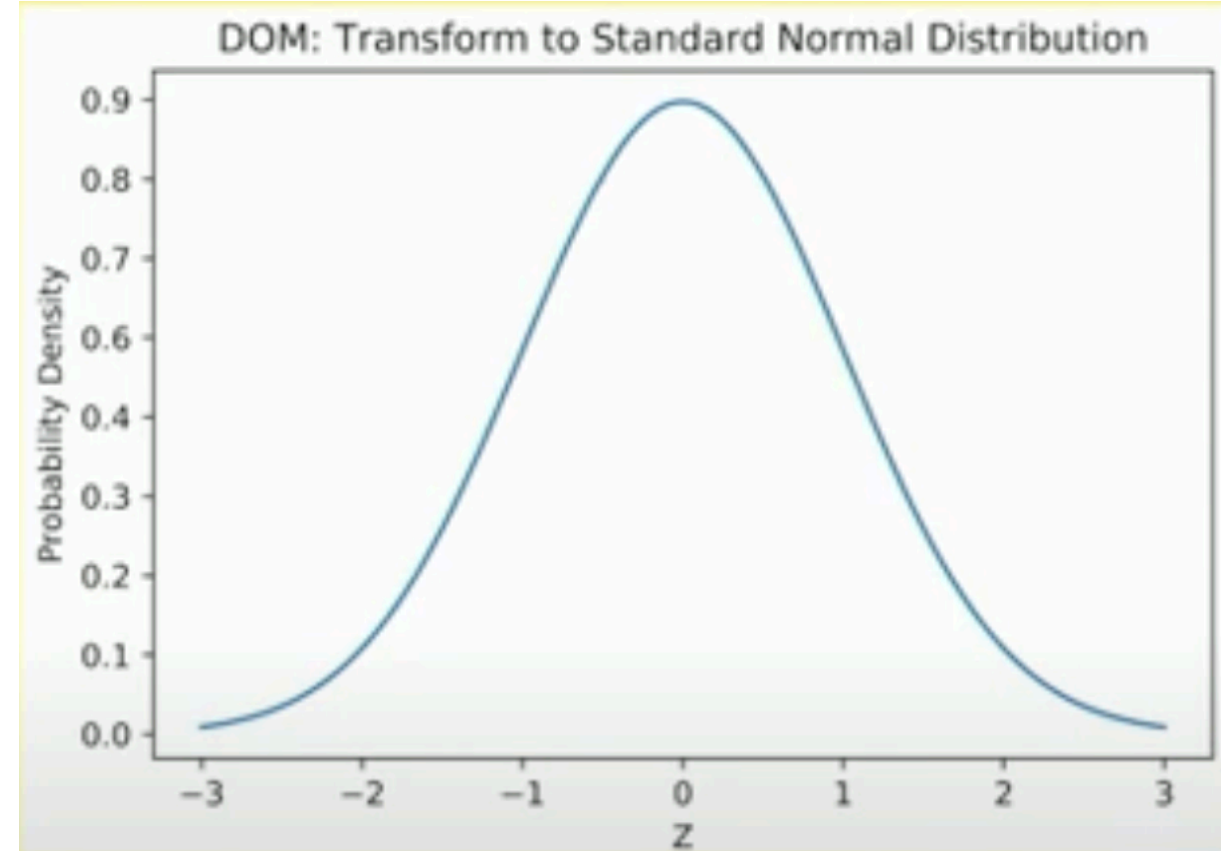
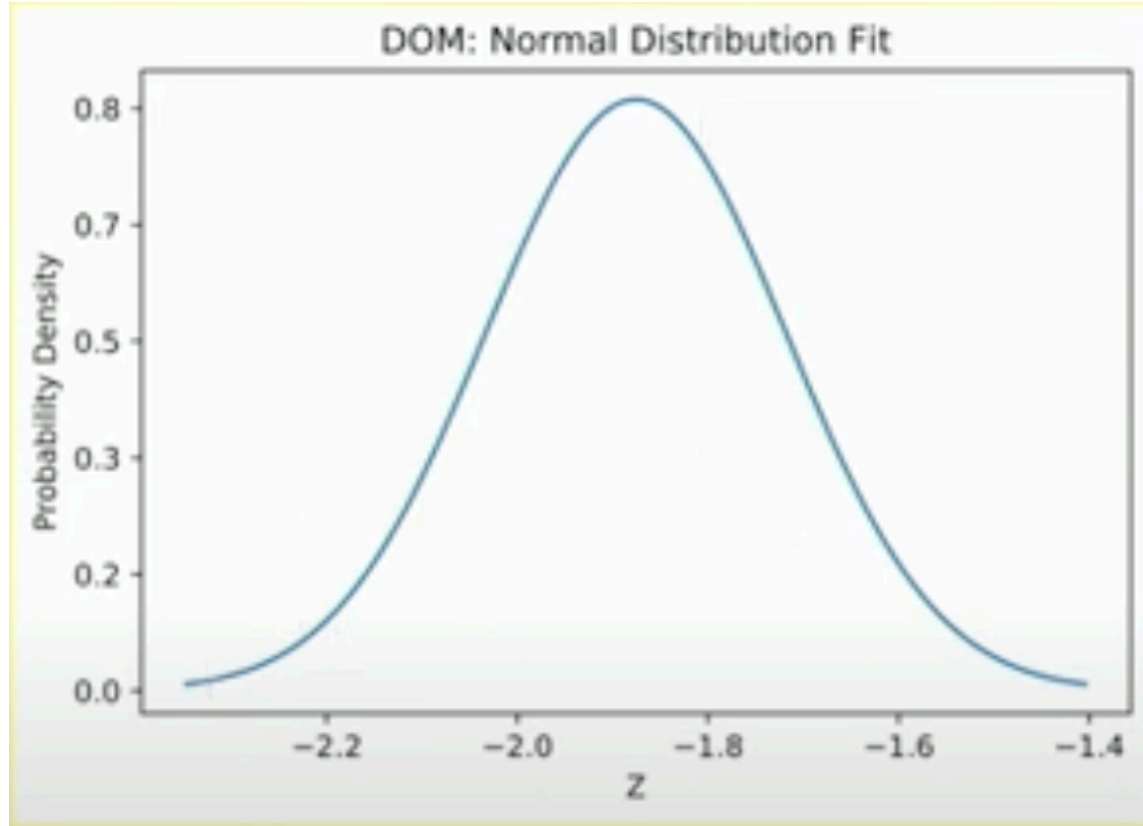
$$2P \left[ \frac{\bar{X} - \bar{Y}}{\sigma \sqrt{\frac{2}{n}}} > \frac{c}{\sigma \sqrt{\frac{2}{n}}} \right] = \alpha \Rightarrow \frac{c}{\sigma \sqrt{\frac{2}{n}}} = Z_{1-\alpha/2} \Rightarrow \mathbf{n} = \frac{2\sigma^2}{c^2} \mathbf{Z}_{1-\alpha/2}^2$$

# Estimating the Number of Traces for DoM





# Estimating the Number of Traces for DoM



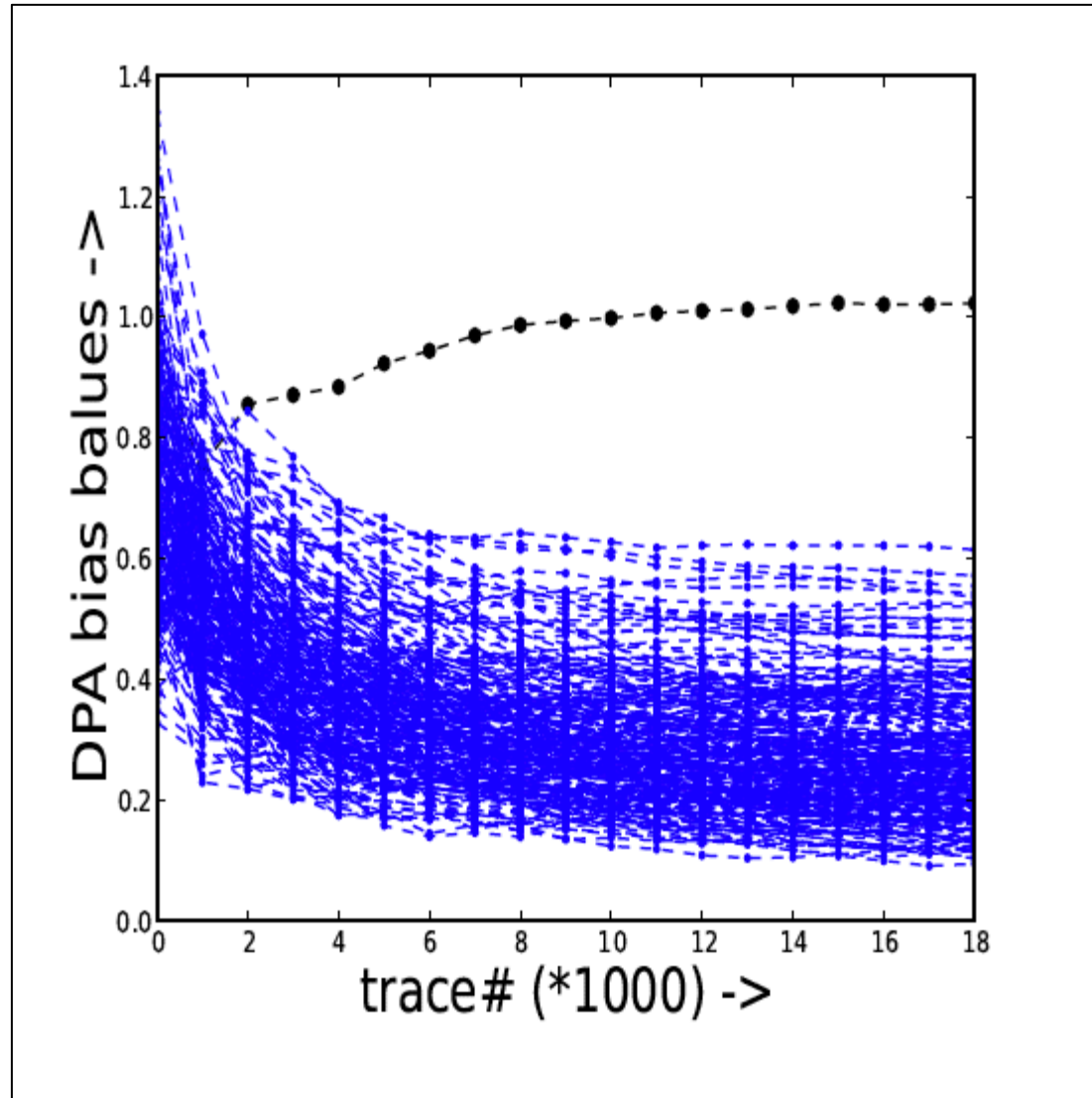
# Estimating the Number of Traces for DoM

With around 4296 traces, with equal proportions going to Set X and Y, we have the following observations:

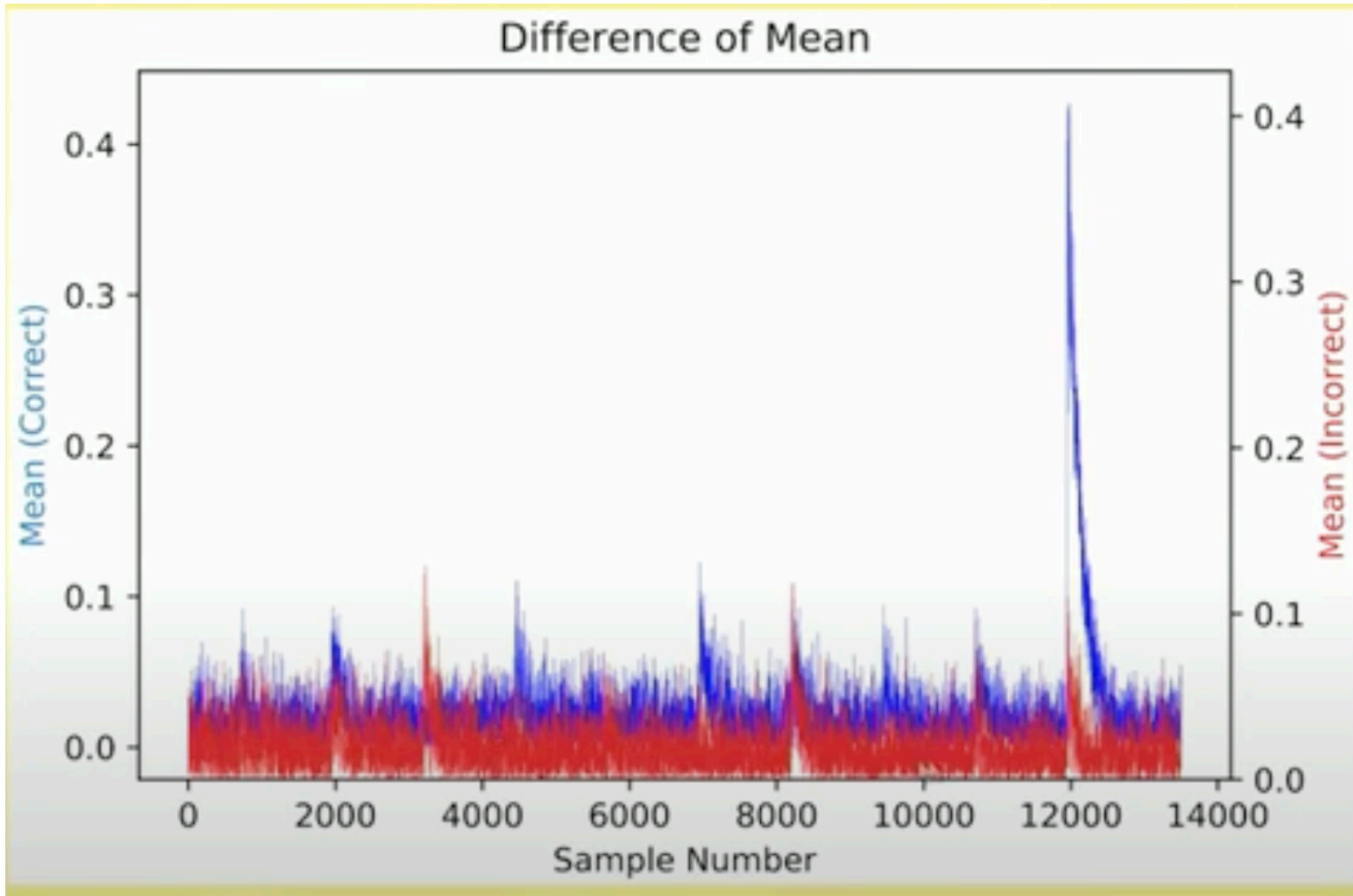
$\sigma = 0.07$ . We set  $c = 0.01$ . For a confidence of  $(1 - \alpha) = 0.95 \Rightarrow \alpha = 0.05 \Rightarrow Z_{1-\alpha/2} = Z_{0.975} = 1.96$ .

Thus, estimated number of traces =  $\frac{2(0.07)^2}{(0.01)^2} \mathbf{1.96^2 = 377}$ .

# Estimating the Number of Traces for DoM



# Estimating the Number of Traces for DoM



# Simulating the Power Traces

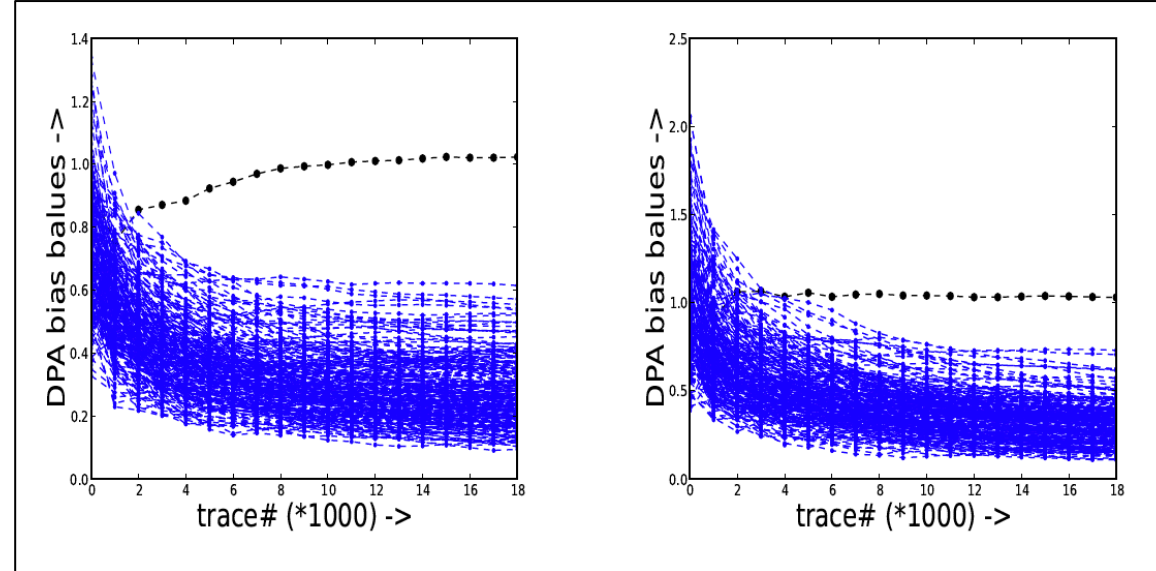
- Let **NSample** denote the number of randomly chosen plaintexts to be encrypted using wlog. the rolled AES cipher.
- In such an architecture, a register is updated by the output of the AES round every encryption.
- The power trace is stored in the array **sample[NSample][NPoint]**, where **NPoint** is corresponding to the power consumption after each round of encryption.
- For each power trace, we also store the corresponding ciphertexts in the array **Ciphertext[NSample]**
- We simulate the power traces by using the power model (Hamming weight or Hamming Distance) on the state of the register before the encryption, then followed by that after initial key addition, then after each of the 9 rounds, and finally the cipher state.
- Thus NPoint is 12 and we choose NSample as 8000.

# Algorithm for Performing DPA

```

Input: sample[NSample][NPoint],Ciphertext[NSample]
Output: biasKey[NKey],biasIndex[NKey]
1 for (cipher = 0; cipher < NCipher; cipher++) do
2   partialCipher = inverseSBox[cipher  $\oplus$  key]
3   if (partialCipher & 1) then
4     for (j = 0; j < NPoint; j++) do
5       sumBin1[j] += sample[cipher][j]
6       countBin1 += freqSample[cipher]
7     end
8   end
9   else
10    for (j = 0; j < NPoint; j++) do
11      sumBin0[j] += sample[cipher][j]
12      countBin0 += freqSample[cipher]
13    end
14  end
15 end
16 bias = 0
17 for (j = 0; j < NPoint; j++) do
18   meanDiff[j] = sumBin0[j]/countBin0 - sumBin1[j]/countBin1
19   if (bias < abs(meanDiff[j])) then
20     bias = abs(meanDiff[j])
21     index = j
22   end
23   biasKey[key] = bias
24   biasIndex[key] = index
25 end

```



DPA on first key byte of  
10<sup>th</sup> Round of AES  
Power is Simulated by  
Hamming Weight of the  
Registers after each  
Round

Power is Simulated by  
Hamming Weight of the  
Registers after each  
Round, with superimposed  
Gaussian Noise.

# DOM and Power Model

- The above attack was demonstrated using Hamming Weight Power Model.
- However, it can be easily adapted for a Hamming Distance power model.
  - Then the classification of the power traces would be based on the change of transitions of the target bit across two successive clock cycles.
  - Rest of Attack is same.

# Stochastic Power Models

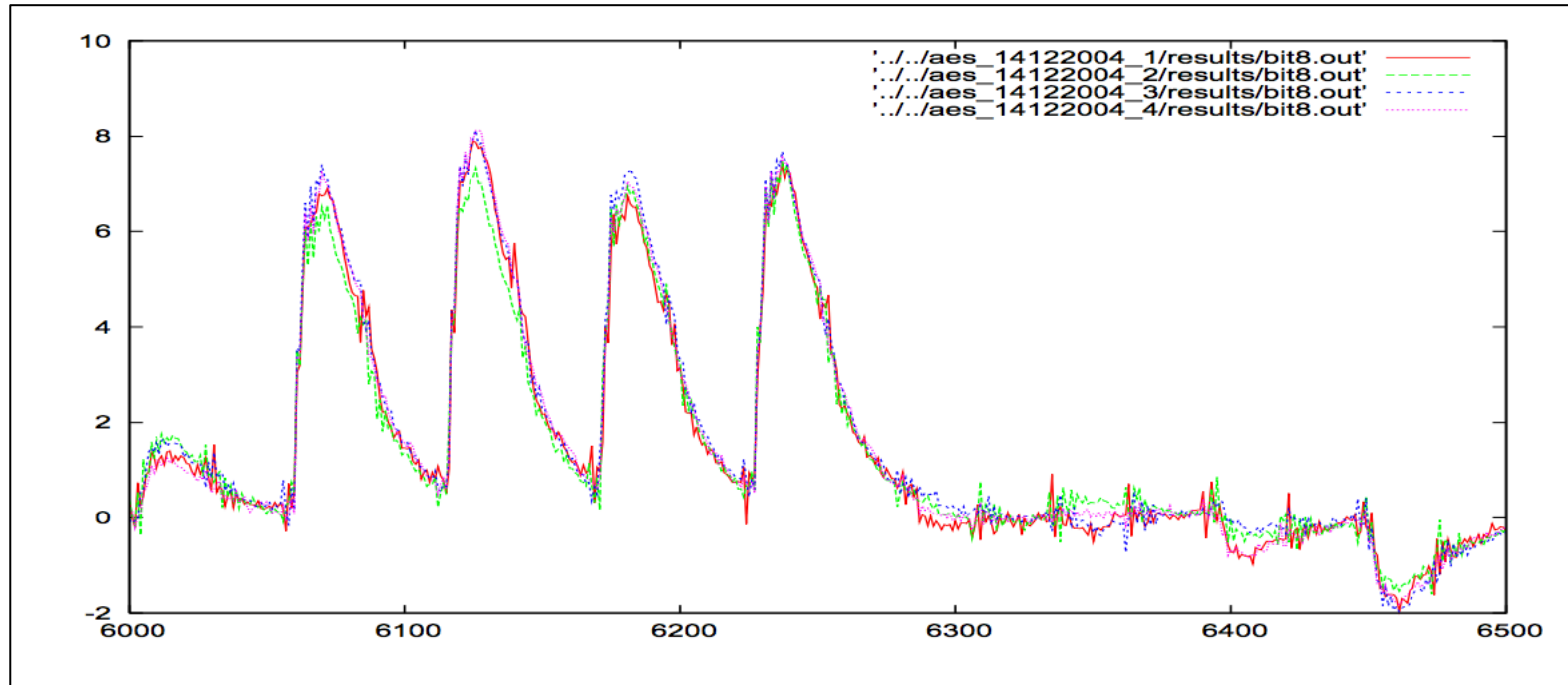
- The DoM technique for DPA is non-profiled.
- On the contrary, there could be another kind of side channel attacks which fall into the category of profiled attacks.
- In profiled attacks, one needs to have access to a pair of identical devices:
  - Target: limited control and running cipher with fixed key.
  - Profiler: full knowledge and control of the input and keys.
- Profiling can help to stochastically learn the power model more accurately.



# An Example of Such a Model

- Consider, a target byte say in the last AES round.
  - $\mathcal{P}(\phi(x, k)) = b_0 + \sum_{i=1}^8 b_i(g_i(\phi(x, k)))$
- Here  $\phi(x, k) = S^{-1}(x \oplus k)$ ,
  - $x, k$  are the portions of the cipher and the portion of the key guessed in the last round
  - $S^{-1}$  stands for Inverse Sub Bytes (or inverse of the last round S-Box)
  - $g_i$  extracts the  $i^{th}$  bit of the state  $\phi(x, k)$ .
- In the profiling phase, we keep the deterministic part defined by  $\phi(x, k)$  constant.
- However, because of noise we have to solve an over-defined system of equations to compute the values of the  $b'_i$ s.

# Coefficient $b_8$ varying with time t.



4 different keys used.

The deviation of the coefficient with time shows that Hamming Distance is not the best power model.

However, we need a profiling step.

In this case, 2000 equations were solved (ie experiments were done with 2000 values of x)

Ref: Werner Schindler, Kerstin Lemke, Christof Paar, A Stochastic Model for Differential Side Channel Cryptanalysis, CHES 2005.

# Correlation Power Attack (CPA)

- Like DoM based DPA, CPA also relies on targeting an intermediate computation, typically the input or output of an S-Box.
- These input values are computed from a known value, say the ciphertext and a portion of the key, which is guessed.
- The power model is then subsequently applied to develop a hypothetical power trace of the device for a given input to the cipher.
- These hypothetical power values are then stored in a matrix for several inputs and can be indexed by the known value of the ciphertext and the guessed key byte.
- This matrix is denoted as  $H$ , the hypothetical power matrix.

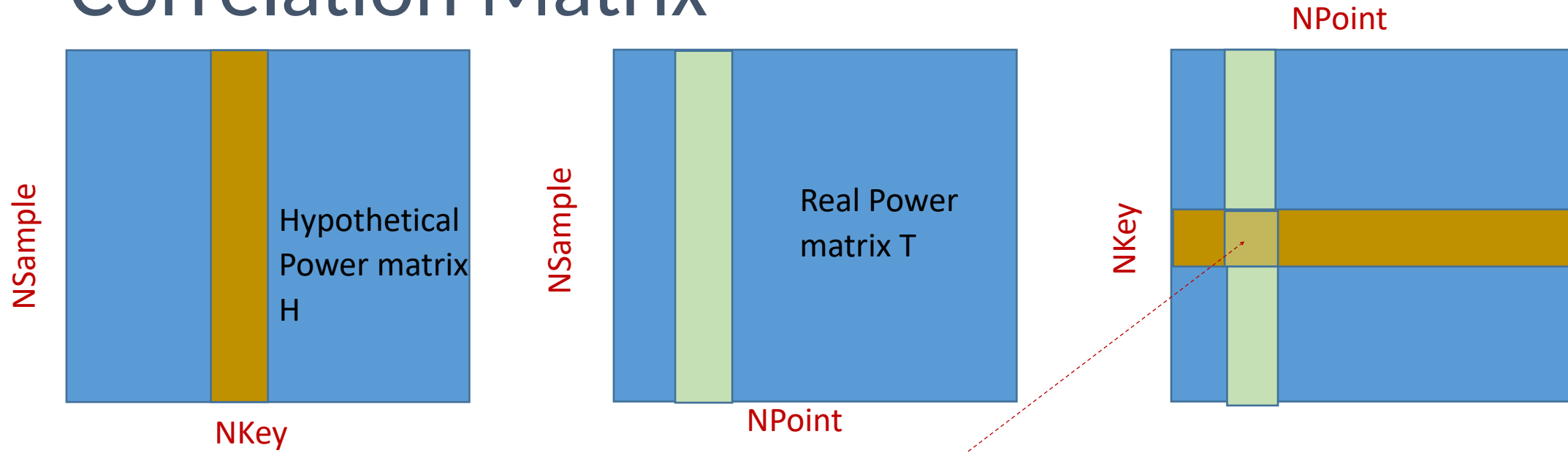
# Correlation Power Attack (CPA)-Contd.

- The attacker also observes the actual power traces, and stores them in a matrix for several inputs.
- The actual power values can be indexed by the known value of the ciphertext and the time instance when the power value was observed.
- This matrix is denoted by  $T$ , the real power matrix.
- It may be observed that one of the columns of the matrix  $H$  corresponds to the correct key  $k^*$ .
- CPA tries to compute the similarity between the columns of the matrix  $H$  and the columns of the matrix  $T$ , to distinguish  $k^*$  from rest: similarity computed by Pearson's Correlation, usually.

# Computing Correlation Coefficient for Simulated Power Traces for AES

- Like before, we simulate the power profile for the iterative AES, this time by using Hamming Distance power model applied on the state registers updated after each round.
- The real power matrix is stored in the array `trace[NSample][NPoint]`
  - NSample: Number of Power Traces acquired
  - NPoint: Time instances for which the power values are observed. Here NPoint is 12.

# Correlation Matrix

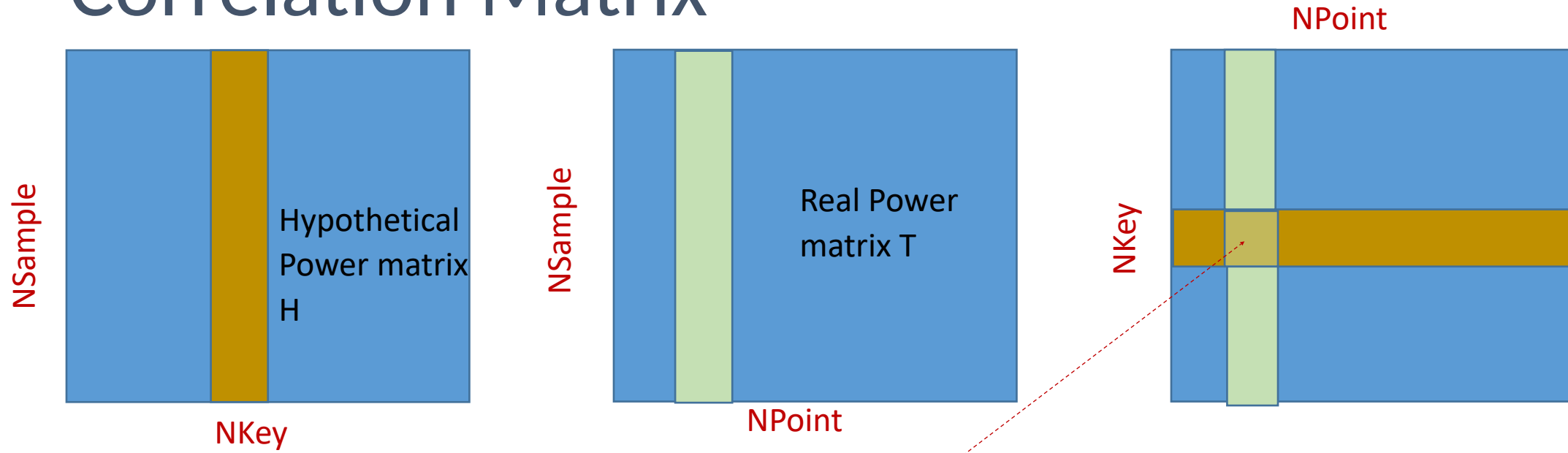


$$C[i][j] = \frac{\sum_{k=0}^{N_{Sample}} (hPower[i][k] - meanH[i])(trace[j][k] - meanTrace[j])}{\sum_{k=0}^{N_{Sample}} (hPower[i][k] - meanH[i])^2 \sum_{k=0}^{N_{Sample}} (trace[j][k] - meanTrace[j])^2}$$

# Computing the Correlation Matrix

- Actual Power values for all the NSample encryptions are stored in the array `trace[NSample][NPoint]`.
  - However, reflect as we are calculating hypothetical power using HD, the trick which we applied before for storing the traces compactly will not work.
  - Attacker first scans each column of this array and computes the average, and stores in `meanTrace[NPoint]`.
- Likewise, the hypothetical power is stored in the array `hPower[NSample][NKey]`.
  - Attacker scans each column and stores in the array `meanH[NKey]`

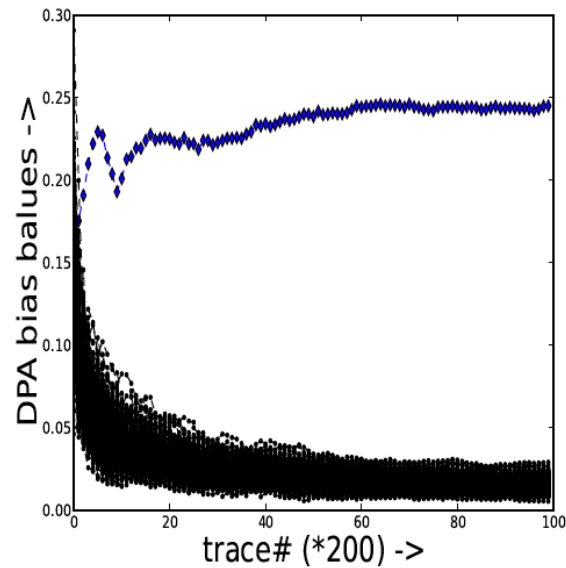
# Correlation Matrix



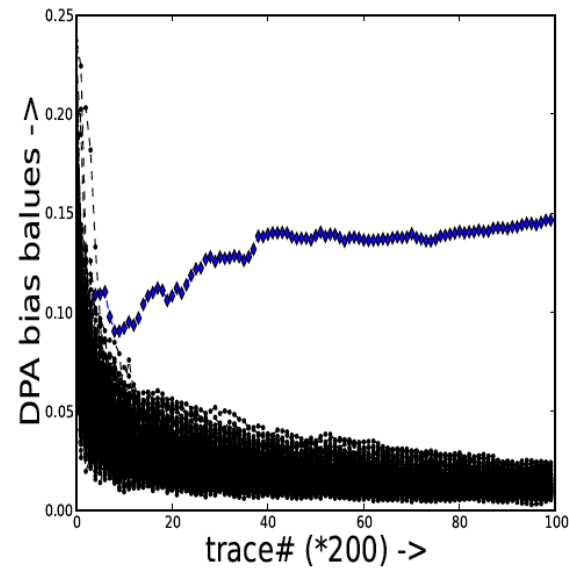
$$C[i][j] = \frac{\sum_{k=0}^{N_{Sample}} (hPower[i][k] - meanH[i])(trace[j][k] - meanTrace[j])}{\sum_{k=0}^{N_{Sample}} (hPower[i][k] - meanH[i])^2 \sum_{k=0}^{N_{Sample}} (trace[j][k] - meanTrace[j])^2}$$



# Experimental Results on Simulated Traces

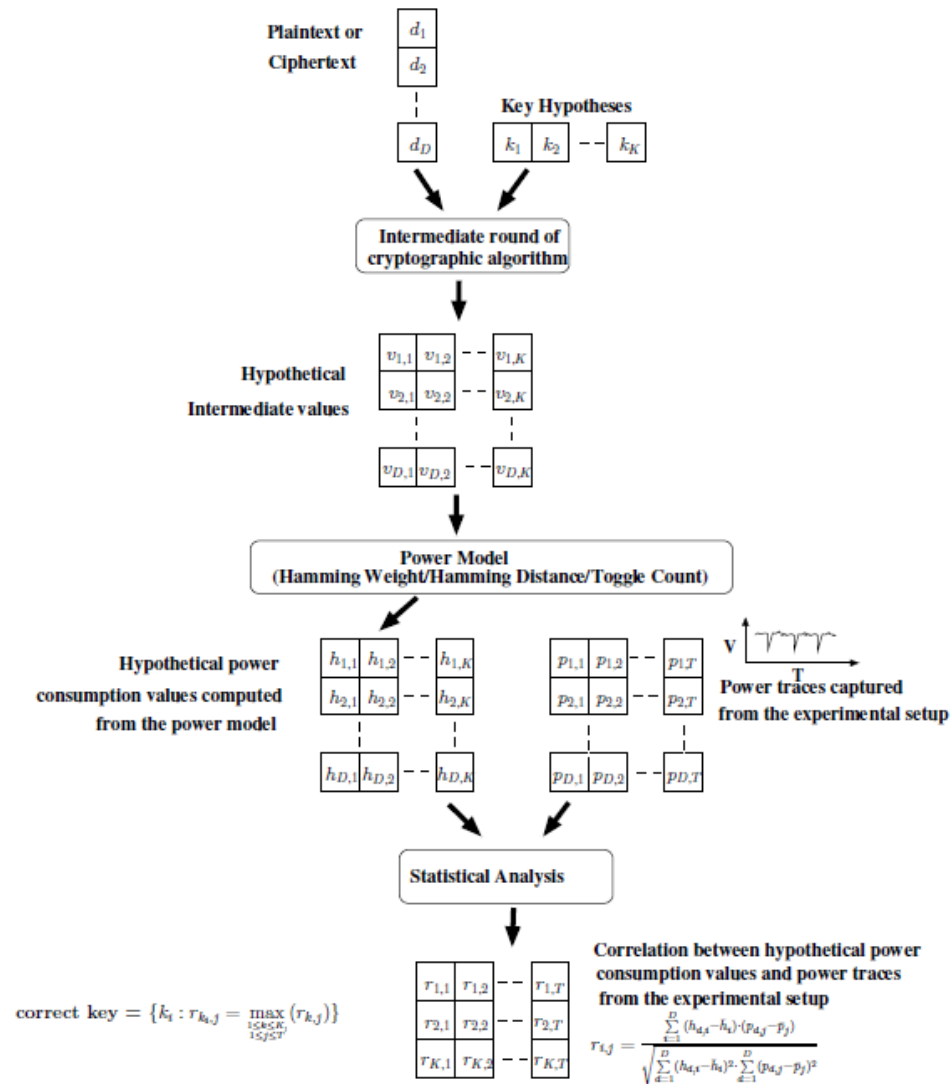


CPA on first key byte of 10<sup>th</sup> Round of AES  
Power is Simulated by Hamming Distance of the Registers after each Round



Power is Simulated by Hamming Distance of the Registers after each Round, with superimposed Gaussian Noise.

# Summary of Correlation Power Analysis



# Metrics for SCA

- In order to evaluate the attacks and also to compare the crypto-implementations wrt. the SCAs we need quantifiable metrics.
- Useful Metrics:
  - Success Rate of a SCA adversary
  - Guessing Entropy of an Adversary

# Success Rate of an Adversary

- SCA works as a divide and conquer strategy, where the key space is divided into several equivalent classes.
- The attack normally does not distinguish between keys which belong to the same class or partition.
- We can formalize the cryptographic implementation as  $E_K$ , where  $K$  is the key space.
- The adversary assumes a leakage model, denoted by  $L$ 
  - The leakage model provides some information about the key or some other desirable information.
- The adversary is an efficient algorithm denoted by  $A_{E_K, L}$ : bounded by time complexity,  $\tau$ , memory complexity,  $m$ , and  $q$  queries.

# Formal Definition of Success Rate

- The leakage exploited by the adversary , maps a key  $k \in K$ , to a set, within which it cannot distinguish.
- We define these partitions by  $S$ , and the mapping by a function  $\gamma$ , st.  $s = \gamma(k)$ ,  $k \in K$ .
- Typically,  $|S| \ll |K|$ .
- The objective of the attack is to determine the class  $s$  to which a target  $k$  belongs with non-negligible probability.
- As an analogy, consider the Hamming Weight class, which divides the key space into equivalence classes or partitions.

# Formal Definition of Success Rate

- As an analogy, consider the Hamming Weight class, which divides the key space into equivalence classes or partitions.
- The output of the adversary is based on the ciphertexts (black-box information) and the leakage (side channel information).
- The output is a guess-vector, which are the key classes sorted in terms of descending order of being a likely candidate.
- Thus we can define a order- $o$  ( $o \leq |S|$ ) adversary when the adversary produces the guessing-vector as  $g = [g_1, g_2, \dots, g_o]$

# Formal Definition of Success Rate

**Input:**  $K, L, E_K$

**Output:** 0 (failure), 1 (success)

```
1  $k \in_R K$ 
2  $s = \gamma(k)$ 
3  $g = [g_1, \dots, g_o] \leftarrow A_{E_k, L}$ 
4 if  $s \notin g$  then
5     return 0
6 end
7 else
8     return 1
9 end
```

The experiment is a success if  $s \in g$

$$Succ_{A_{E_K, L}(\tau, m, q)} = \Pr[Exp_{A_{E_K, L}} = 1]$$

# Guessing Entropy of an Adversary

- The above metric for an  $o^{th}$  order attack implies the success rate for an attack where the remaining load is  $o$ -key classes.
- The attacker has a maximum of  $o$ -key classes to which the required  $k$  may belong.
- While the above definition is fixed wrt. the remaining work load, we define guessing-entropy to provide a more flexible definition for the remaining work load.



# Formalizing Guessing Entropy

---

**Input:**  $K, L, E_K$

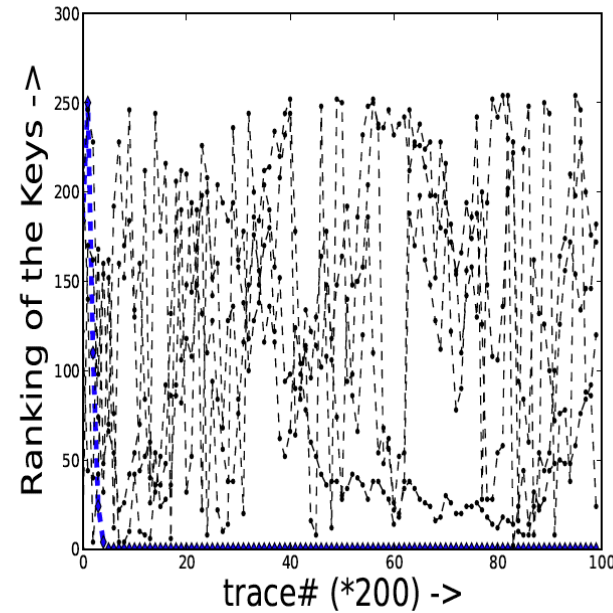
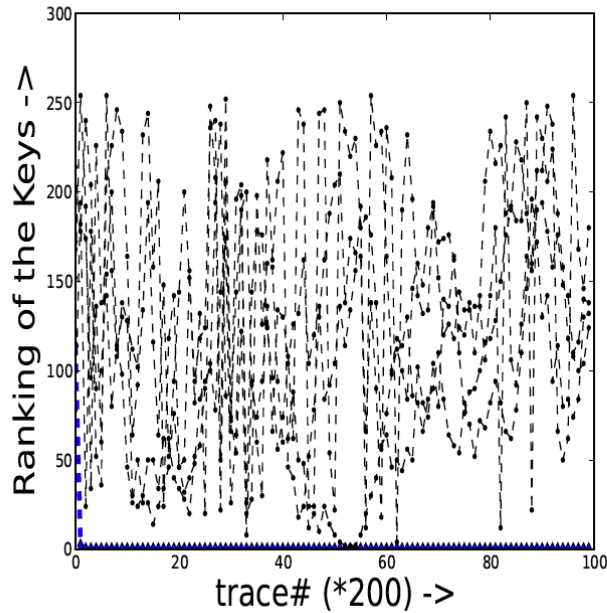
**Output:** Key class  $i$

- 1  $k \in_R K$
- 2  $s = \gamma(k)$
- 3  $\mathbf{g} = [g_1, \dots, g_o] \leftarrow A_{E_k, L}$
- 4 **return**  $i$  such that  $g_i = s$

- Formal definition of Guessing Entropy of the adversary against a key class variable  $S$

$$GE_{A_{E_K, L}(\tau, m, q)} = E[Exp_{A_{E_K, L}}]$$

# Guessing Entropy Plots



CPA on Simulated Power Traces with and without noise.

Blue indicates that the guessing entropy for the correct key byte drops to 0 faster than the wrong keys.

# CPA on Real Power Traces

- When attacking real power traces, the effect of electrical and algorithmic noise comes into effect.
- Architecture often has a strong effect on the algorithmic noise.
- It determines the Signal-to-Noise Ratio (SNR) which quantifies the quality of the power traces.
- This in turn affects the Success rate and the Guessing-Entropy of the attacks.

# SNR of a Power Trace

- Consider a Cipher, like AES, with  $r$ -rounds and let  $S$  be a random variable representing the key dependent intermediate variable of  $E$ .
- $S$  is called the target and satisfies  $S = F_{k^*}(X)$ , where  $X$  is a random variable representing a part of known plaintext or ciphertext.
- $F_{k^*}$  is a function which depends on a part of the cipher and the output depends on a part of the secret key, denoted as  $k^*$ .
- The function  $F_{k^*}$  also depends on the leakage function of the hardware device.
- This component is also denoted as

$$L_t = aF_{k^*}(X) + N_t = aS + N_t$$

Here  $a$  is some real constant.  $N_t$  is some Gaussian Distribution  $N_t \sim N(0, \sigma_t)$

# Distinguishers: Univariate Vs. Multivariate

- Leakage has a deterministic part  $aS$ .
  - *Can be simulated by the adversary knowing the value of  $X$  and each key hypothesis.*
- Distinguisher: A mathematical function that the attacker applies at each point of the trace to distinguish the correct key from the wrong key.
- If the observed leakage is denoted by  $F_k(X)$  then the distinguisher is a vector  $D_t = \{d_k \mid k \in K\}_t$ , where  $d_k = D(aF_{k^*}(X) + N_t, F_k(X))$
- So far, we have discussed distinguishers at one point  $t$ .

$$L_t = aF_{k^*}(X) + N_t = aS + N_t$$

Here  $a$  is some real constant.  $N_t$  is some Gaussian Distribution  $N_t \sim N(0, \sigma_t)$

# Distinguishers: Univariate Vs. Multivariate

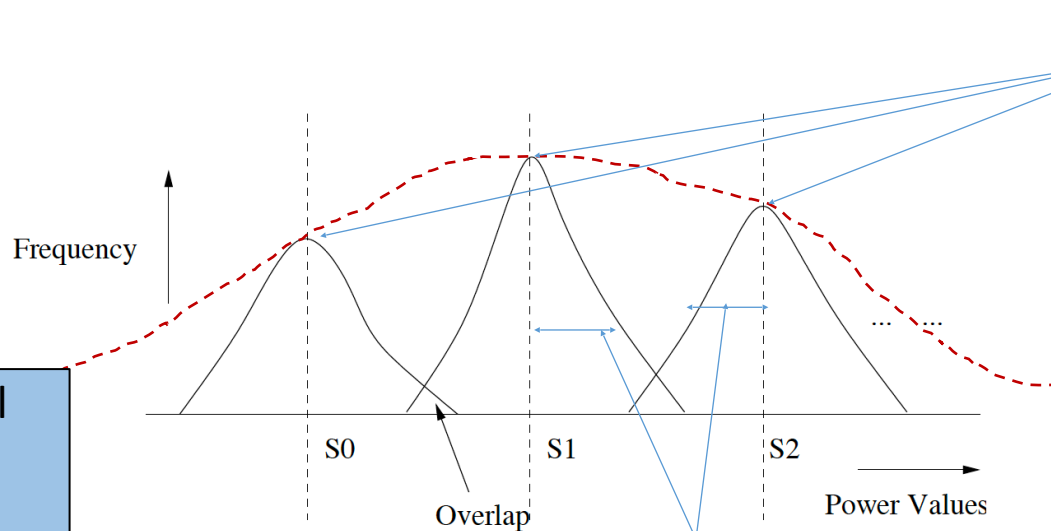
- So far, we have discussed distinguishers at one point  $t$ .
- But multiple trace points can be combined within a distinguisher.
  - Generally, done by calculating higher-order moments:
    - $(x - \bar{x})(y - \bar{y})$ , where  $x, y$  are points on the trace at time  $t_1$  and  $t_2$
    - This is basically co-variance!!!
- Multivariate attacks are good..but
  - Both trace points have some noise, (some SNR)
  - The noise amplifies along with the signal.
  - So, have to be careful

# Definition of SNR

- A register  $X$  is being updated by an AES circuit depending on the inputs.
  - We observe the power traces by varying the inputs and creating a frequency plot at some point of interest,  $t$
  - Our power model tells that the power values would depend on the Hamming weights, say  $S$

$Var(E[L_t | S])$  indicates the variations in leakage due to the target  $S$  at sample point  $t$ .

This contributes to the signal component, in aiding the attack.  
More this value, the  $S$ -values are distinguishable better



$$E[L_t | S]$$

$Var(L_t - E[L_t | S])$  contributes to noise.  
More this variance, more is the overlap between the leakage due to the target variable  $S$ .  
This hinders the attack.

$$L_t - E[L_t | S]$$

# Definition of SNR

- $SNR_t = \frac{Var(E[L_t | S])}{Var(L_t - E[L_t | S])}$ 
  - The SNR is a very useful metric for assessing the threat of a power attack on a given implementation.
- How can it be computed in experiments?
  - Say we collect 10,000 traces.
  - We target a byte, say denoted as S.
  - Depending on the Hamming Weight (say) we split the traces into 9 bins.
  - At a time t, thus the  $i^{th}$ -bin consists of power values,  $P_i^0, \dots, P_i^{n_i-1}$ 
    - The average of this gives a point in the distribution  $E[L_t | S]$ . We calculate the variance for the signal.
  - From each of the power values in the  $i^{th}$ -bin we subtract the average of the bin.
    - The variance of this gives the noise component.