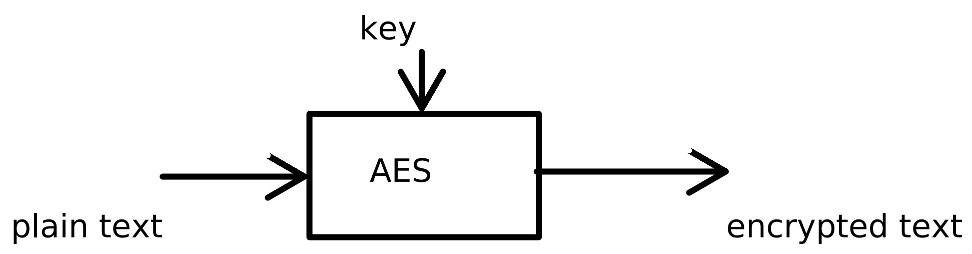
DPA attack on AES encryption algorithm

Paul C. Kocher, Joshua Jaffe, Benjamin Jun:
Differential Power Analysis. CRYPTO 1999: 388-397

AES reminders:

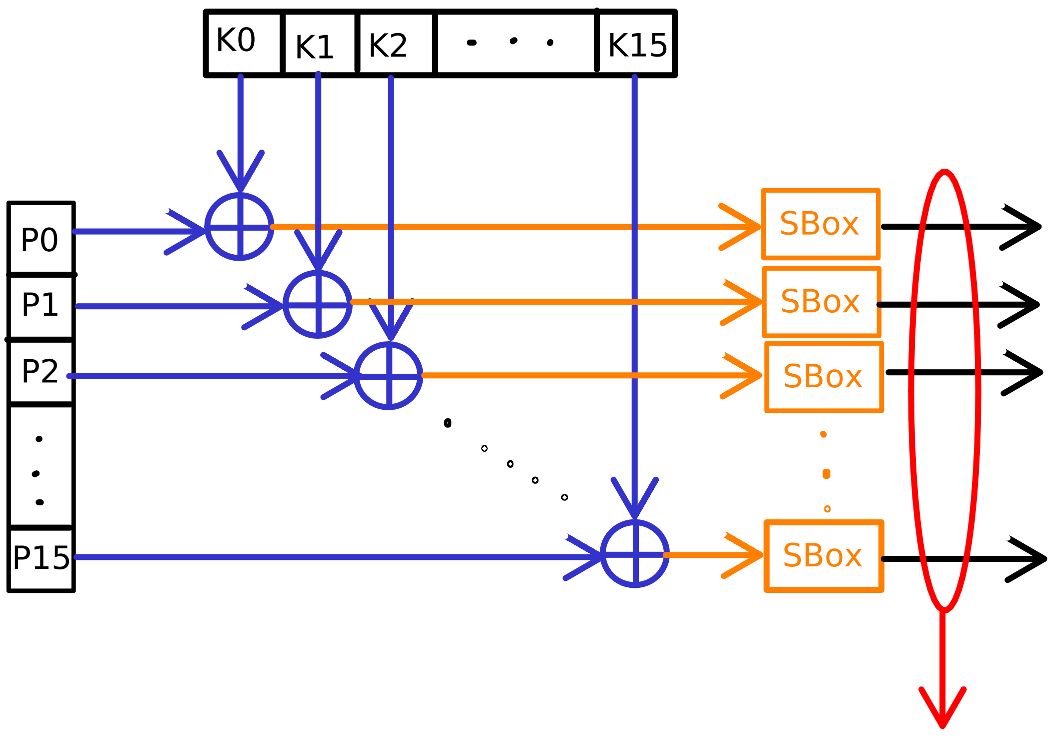


Operations :

$$\left.\begin{array}{l}\text{AddRoundKey}\\ \text{SubBytes}\\ \text{ShiftRows}\\ \text{MixColumns}\end{array}\right\}\ \text{9 rounds}$$

AddRoundKey
SubBytes
ShiftRows

AddRoundKey



16 Intermediate values (bytes)
-> "Divide and conquer"

Attack :

- Plain text: variable and known
- key : fixed and unknown

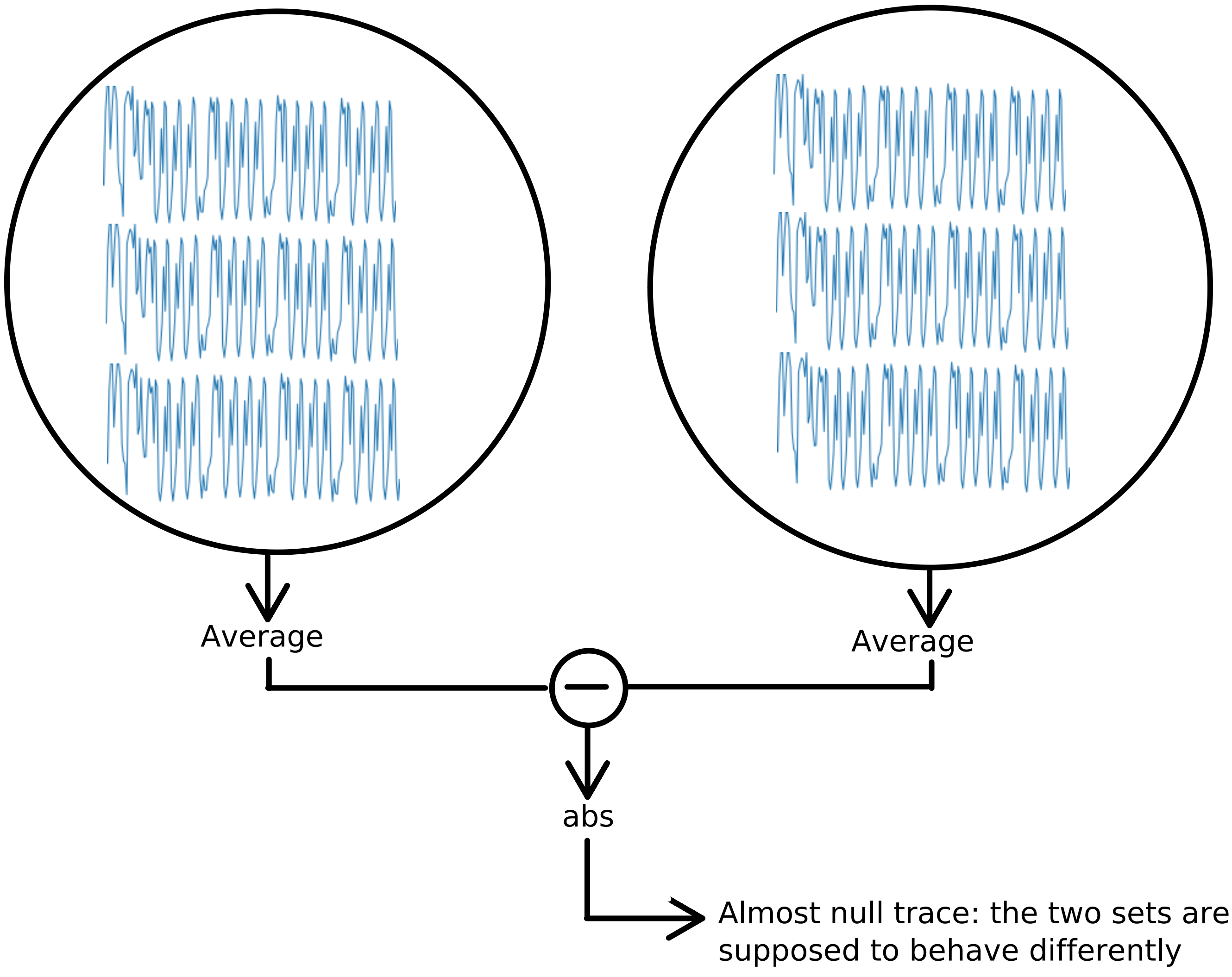
Underlying principle:

Power consumption of micro-controller depends on manipulated data.

Trace partitioning:

Differential : We need to partition traces into two sets in order to observe a different behavior between group 1 and group 2

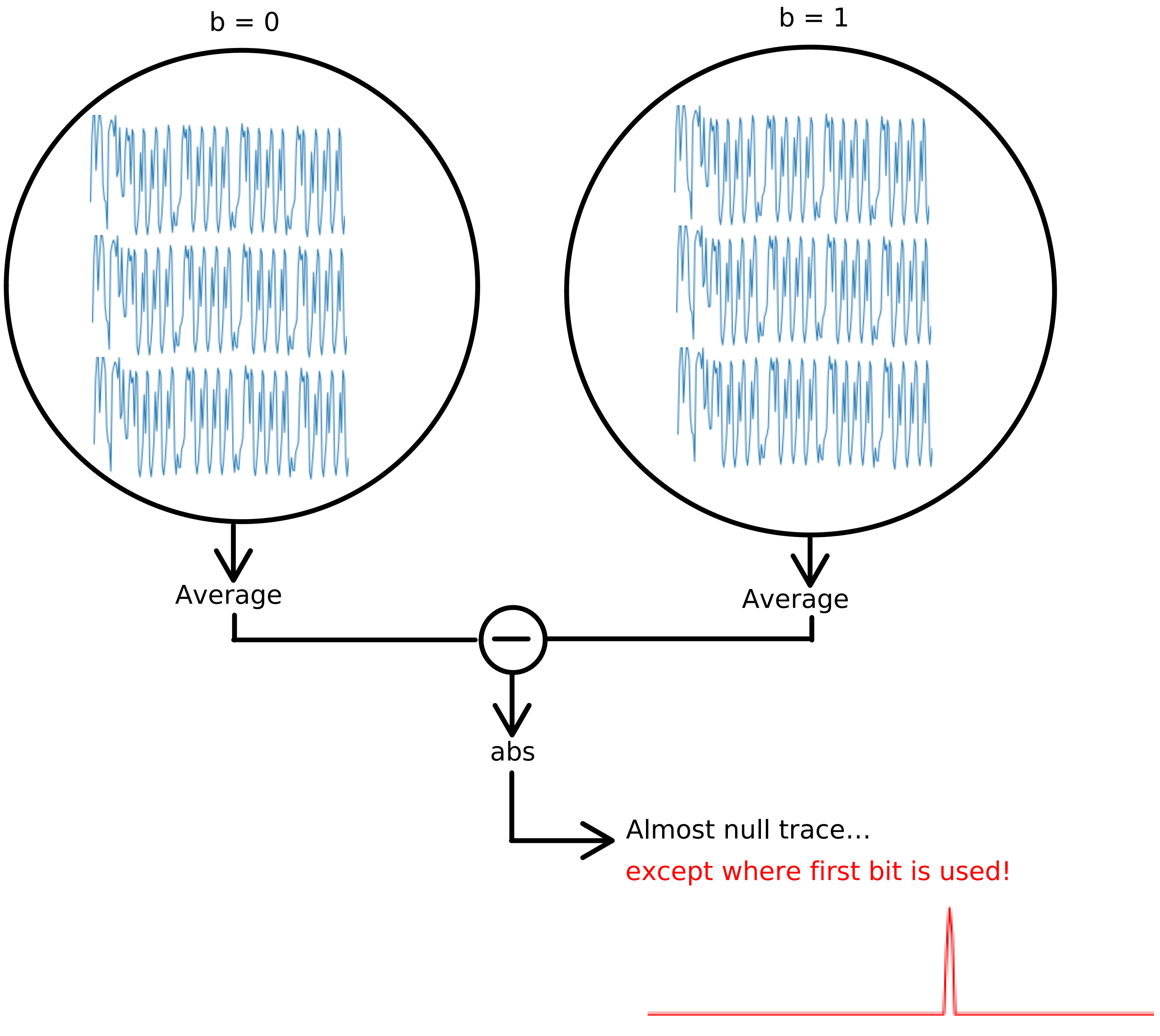
Random partitioning:


Average
Average
abs
Almost null trace: the two sets are not supposed to behave differently

"Correct" partitioning :

Leakage model

We partition according to 1st output bit of 1st SBox
- we know the plain text
- we guess the key



Hence, if the partitioning is well chosen, we can see a peak in the differential trace

"Divide and conquer" :

Algorithm :

```
For each byte of the key:
    For each possible value (guess) (among 256 possibilities):
        For each pair (trace, plain text)
            We compute the output bit of SBox (leakage bit)
        We classify traces into two sets, according to the value of leakage bit
        We compute the differential trace
        We store the max absolute peak of the differential trace
    We find the guess which has the highest peak → We guess the most likely key
```

To go further:

* Measure the number of traces neede for the attack to succeed,
* Change acquisition parameters to reduce the number of traces,
* Improve presentation of attack results,
** Try different leakage models,
** Identify, within the trace, the most leaking sample,
** Speed up the attack using numpy’s matricial acceleration,
** Attack on known cipher text (instead of known plain text),

*** Tracer, pour chaque octet, le rang de la bonne hypothèse en fonction du nombre de traces considéré. Ce rang tend vers 1.

Help Numpy :

Data structures:

- traces : matrix (N_traces x N_samples)
- key : vector (16)
- plaintexts : matrix (N_traces x 16)

Useful functions (import numpy as np) :

- np.bitwise_xor : binary exclusive OR
- np.mean : average
- np.abs : absolute value
- np.max : maximum
- np.argmax : index of maximum in a vector
- np.argsort : indexes of sorted vector
- reversed(tab) : inverted vector tab
- random.randint(A, B) : random integer between A and B