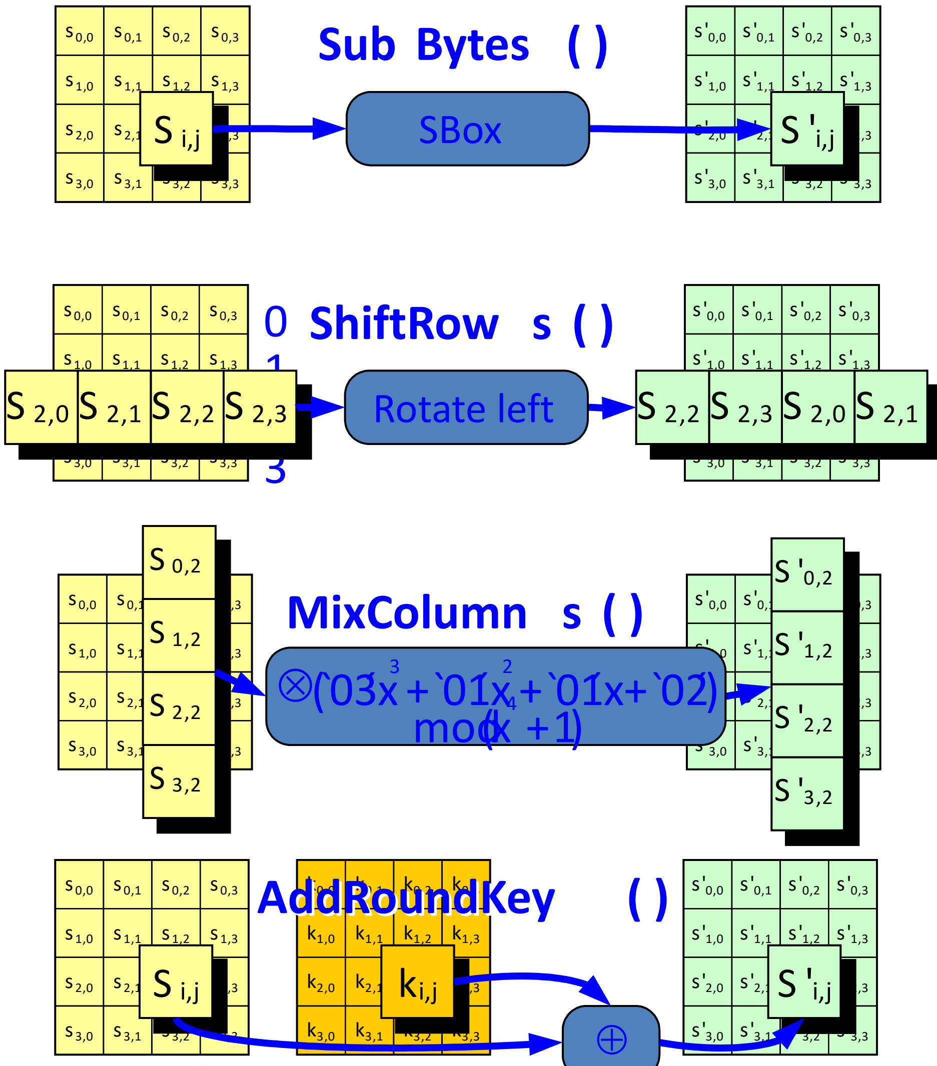
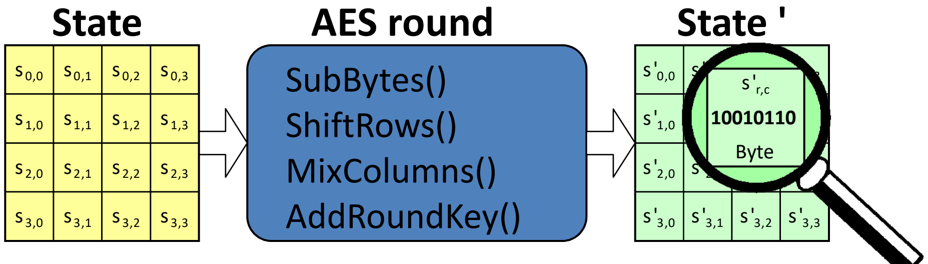
Applied Security

SPA on DES/AES (1)

AES

- AES-128
  - State: 128-bit block
    - Matrix of 4*4 bytes
  - Round function
    - 10 Iterations
  - Key scheduling
    - 10 128-bit round keys
- In software:
  - ARK, SB, SR, then MC


Sub Bytes ( )
SBox
$S_{i,j}$
$S'_{i,j}$
0 ShiftRows ( )
1
3
Rotate left
$S_{2,0}$ $S_{2,1}$ $S_{2,2}$ $S_{2,3}$
$S_{2,2}$ $S_{2,3}$ $S_{2,0}$ $S_{2,1}$
MixColumns ( )
$S_{0,2}$ $S_{1,2}$ $S_{2,2}$ $S_{3,2}$
$\otimes$(`03$x^3$ + `01x^2 + `01x + `02) mod (x^4 + 1)
$S'_{0,2}$ $S'_{1,2}$ $S'_{2,2}$ $S'_{3,2}$
AddRoundKey ( )
$k_{i,j}$
$\oplus$



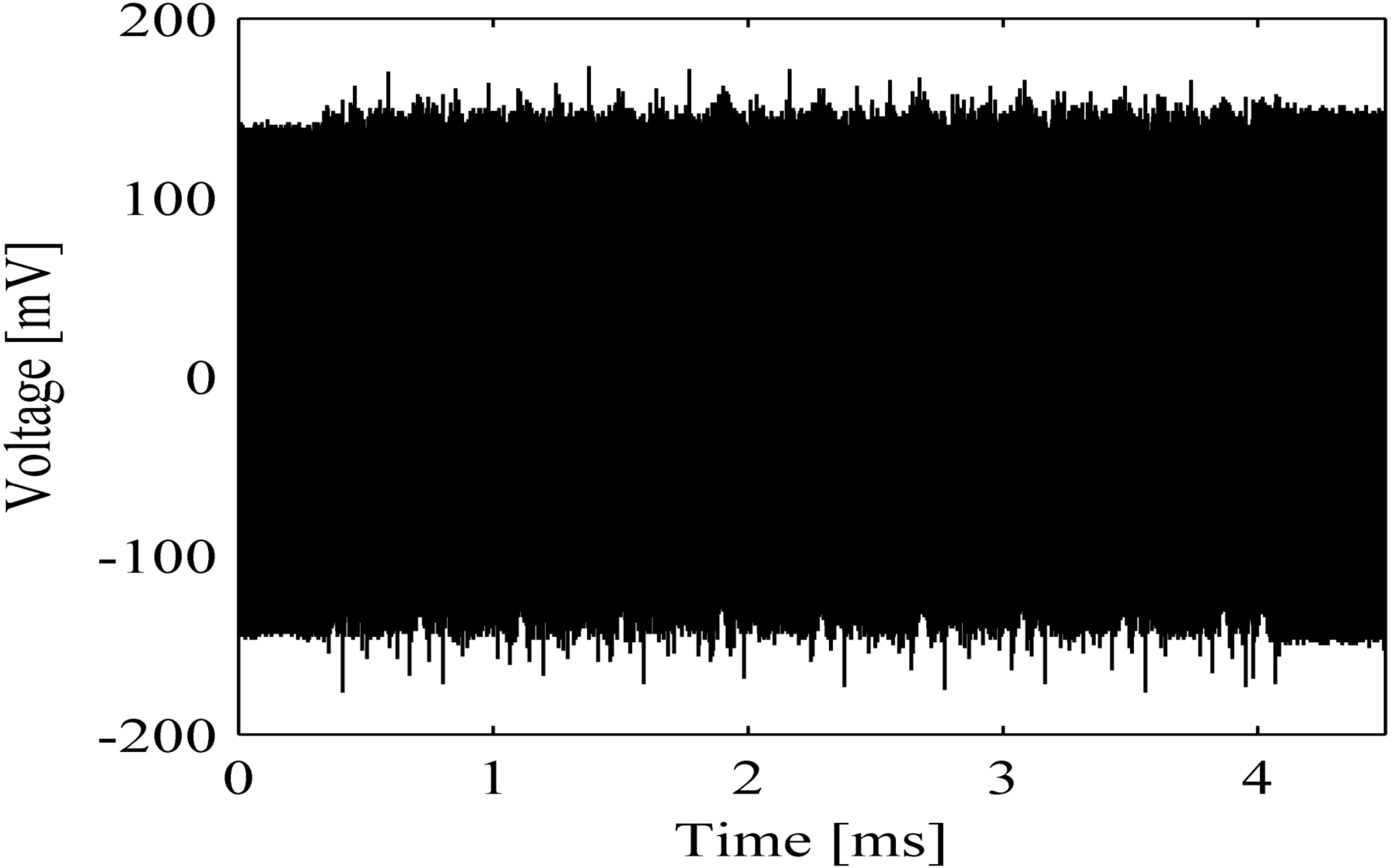
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

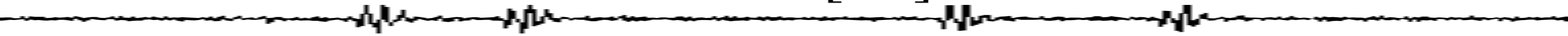
SubBytes()
ShiftRows()
MixColumns()
AddRoundKey()

Case study: SPA on AES (1/6), AES implementation

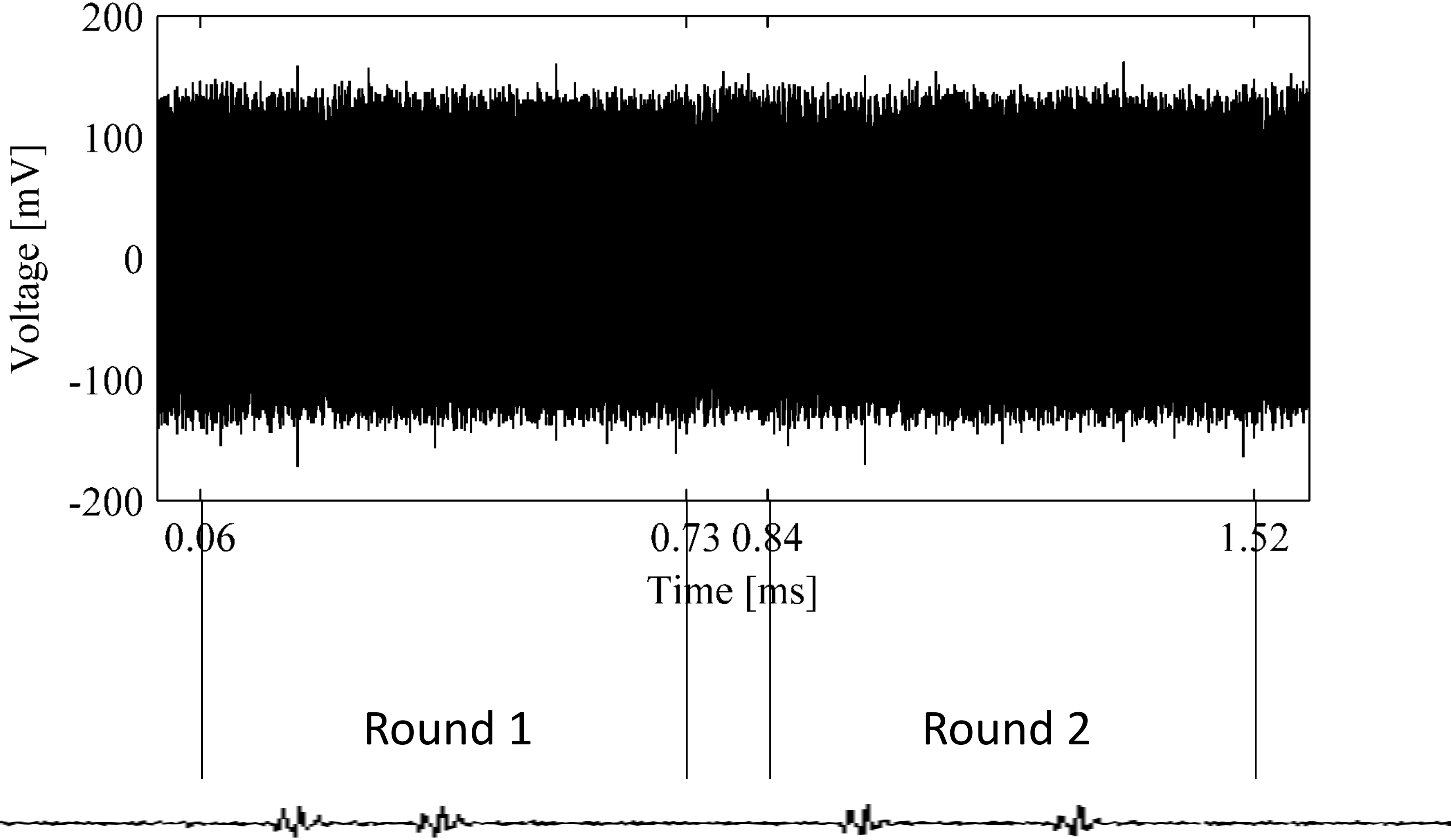
- Diff. transformations require diff. instr.:
  - AddRoundKey
    - MOV (*RAM* → *register*)
    - XOR (*ALU*)
  - SubBytes
    - MOV (*RAM* → *register* or *ROM* → *register*)
  - ShiftRows
    - MOV (*register* → *register* or *register* → *RAM*)
  - MixColumns
    - XOR, AND (*ALU*)
    - MOV (*register* → *register*)

Case study: SPA on AES (2/6), one full AES encryption

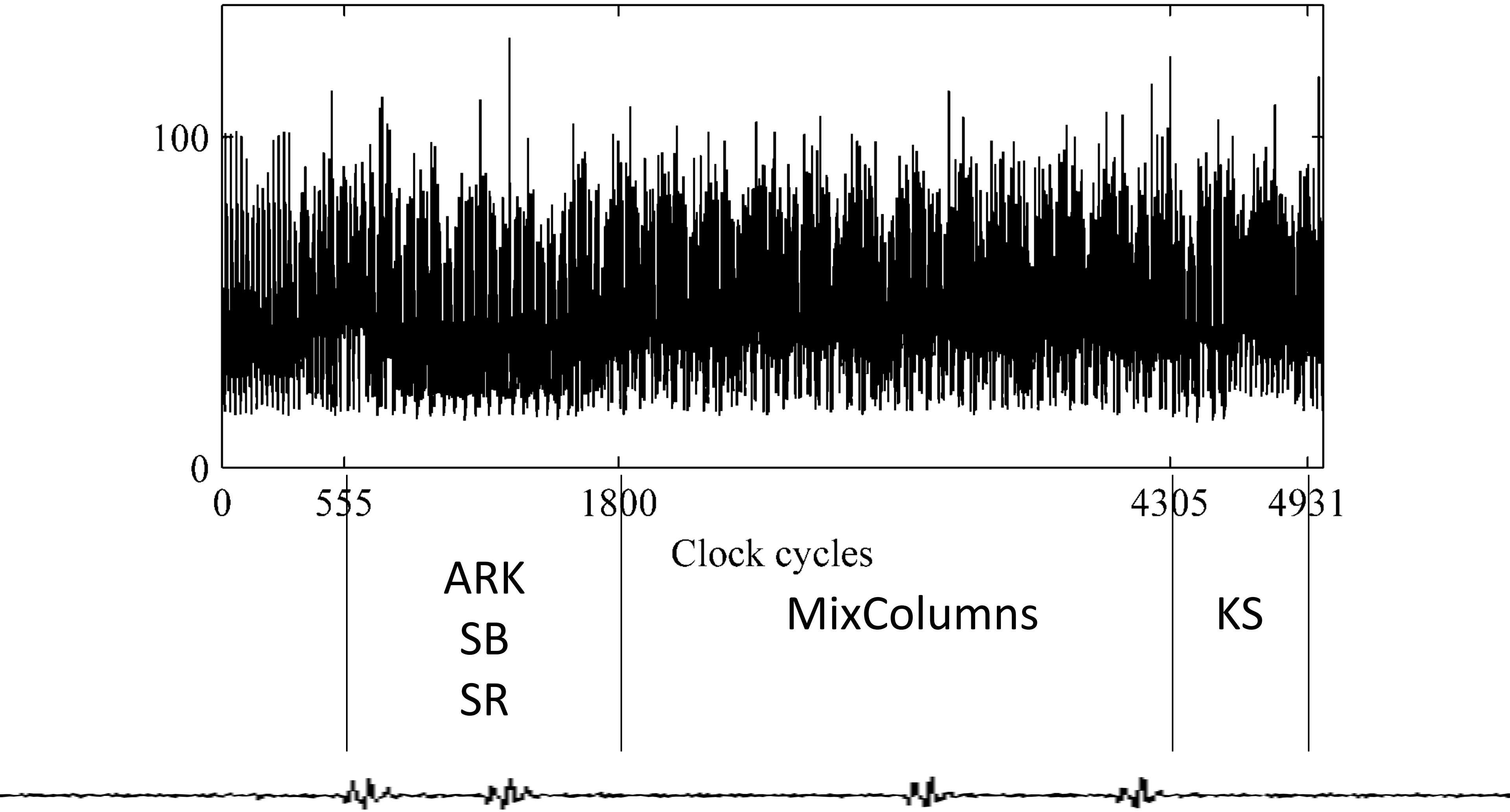




Case study: SPA on AES (3/6), zoom in on 2 AES rounds



Case study: SPA on AES (4/6), zoom in on single AES round

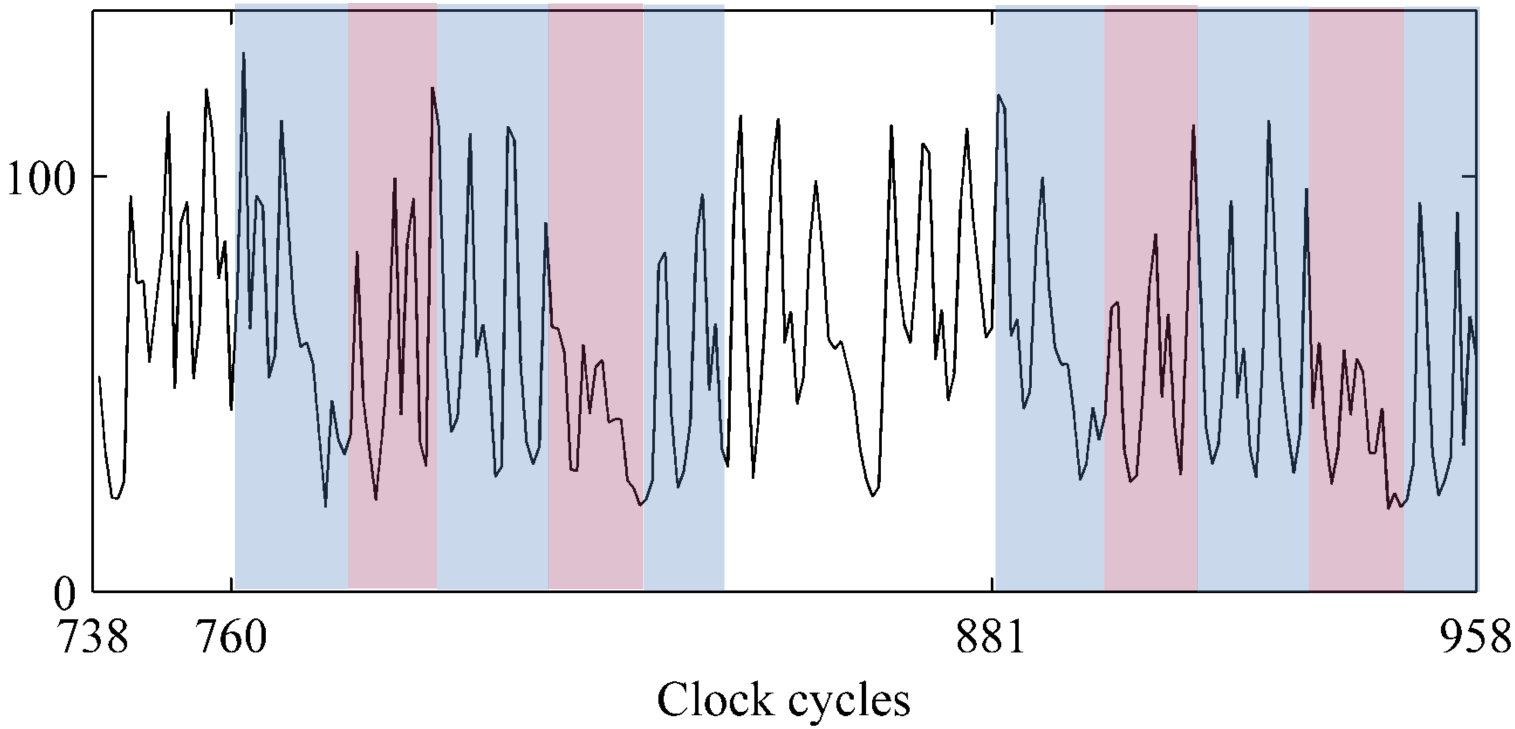


ARK
SB
SR

MixColumns

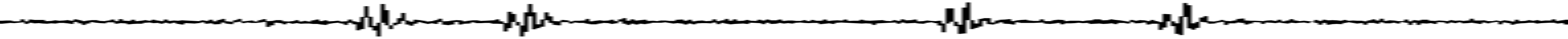
KS

Case study: SPA on AES (5/6), zoom in on processing of bytes 1 and 2

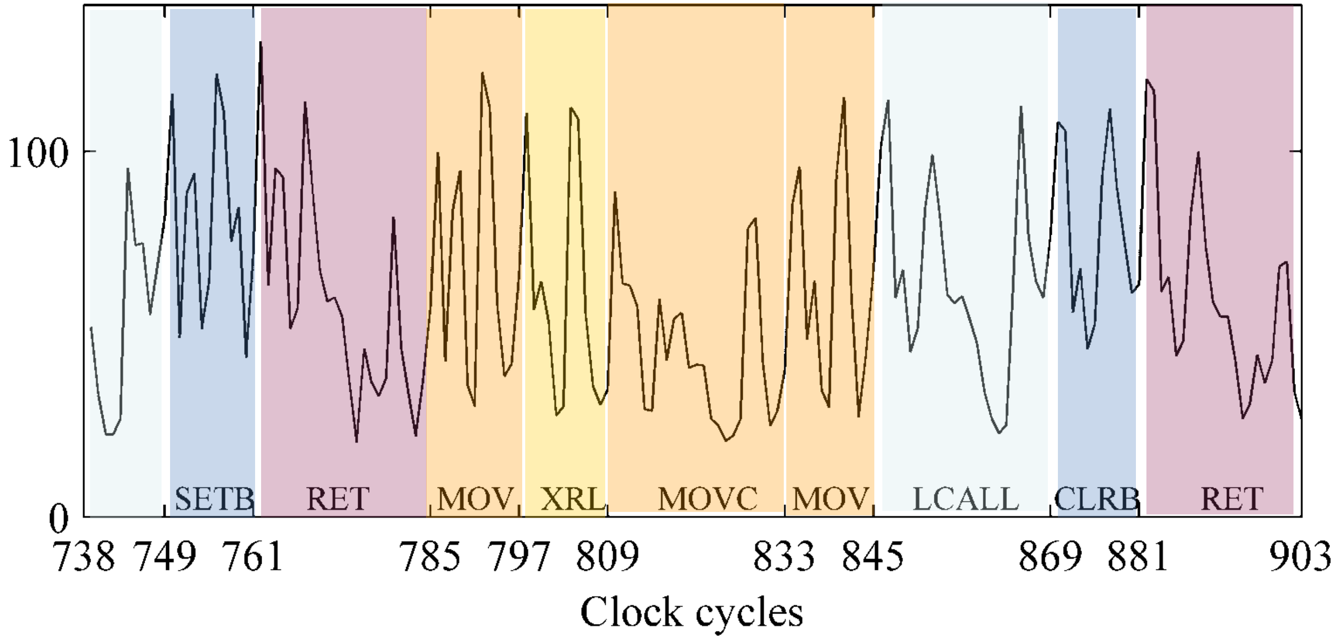


ARK, SB, SR
for byte 1

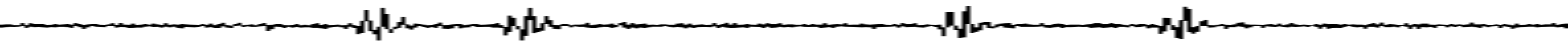
ARK, SB, SR
for byte 2



Case study: SPA on AES (6/6), zoom in on processing of byte 1

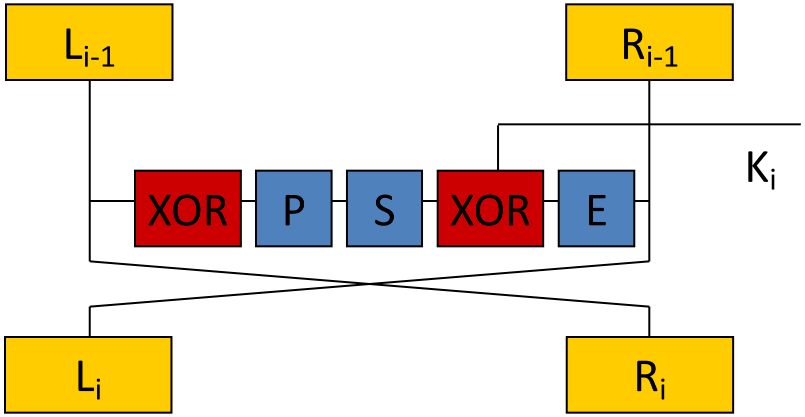


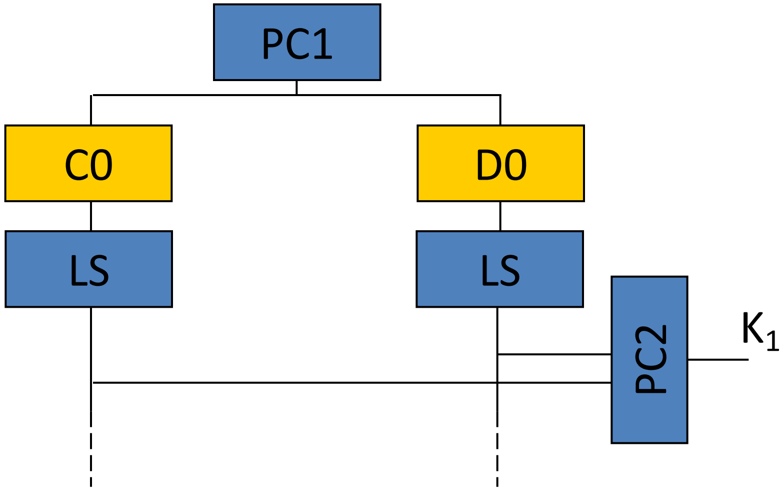
- **LCALL SET_ROUND_TRIGGER ; SETB + RET**
- **MOV A, ASM_input+0 ; load State byte**
- **XRL A, ASM_key+0 ; AddRoundKey**
- **MOVC A, @A+DPTR ; SubBytes**
- **MOV ASM_input, A ; store State byte (incl. ShiftRows)**
- **LCALL CLEAR_ROUND_TRIGGER ; CLRB + RET**

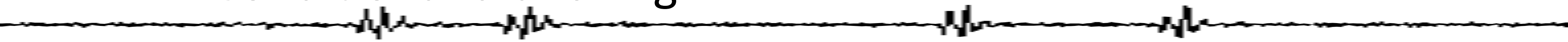


SPA Example, DES

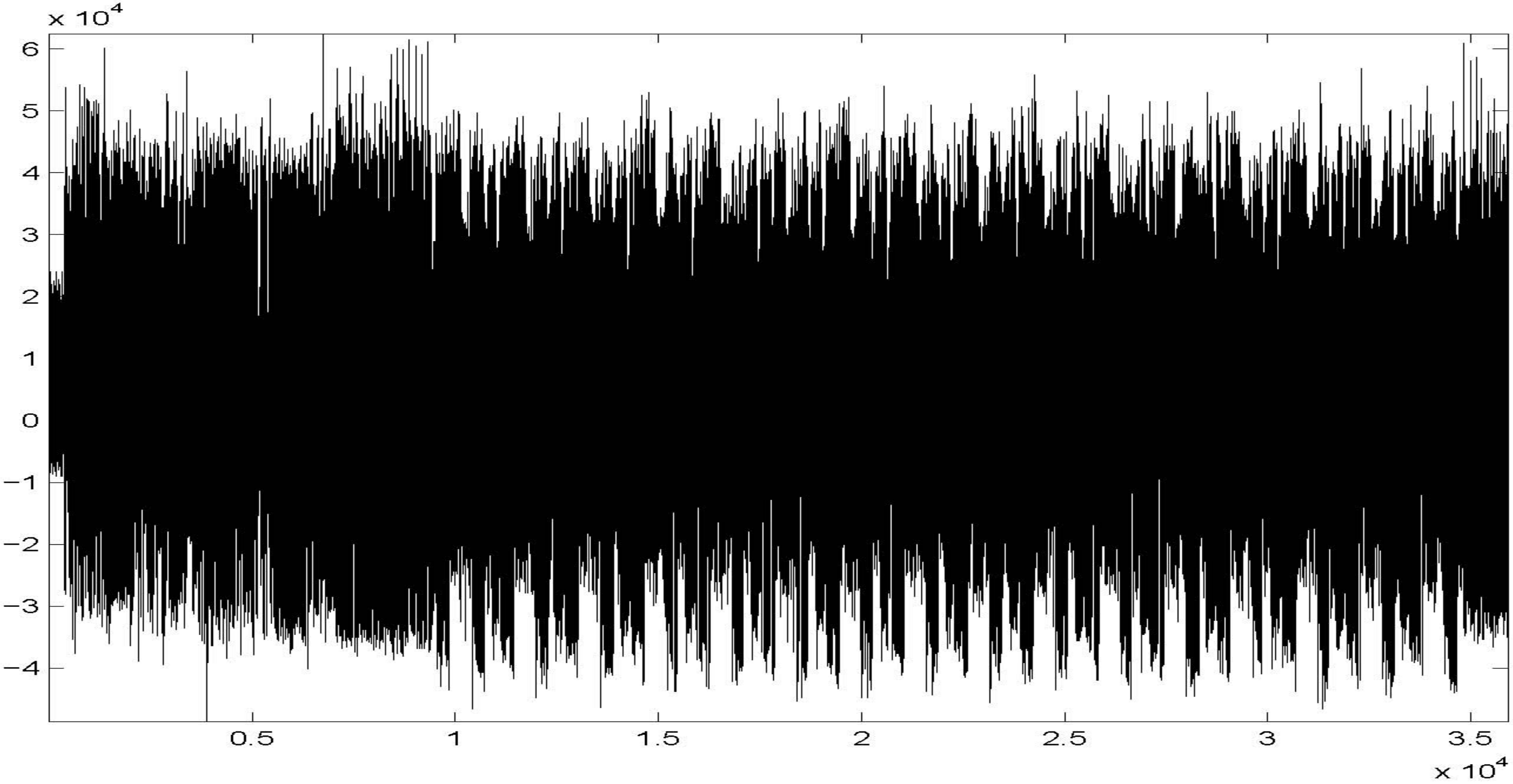
- DES (Data Encryption Standard)
  - 16 Rounds
    - Feistel structure
    - Roundfunction: E, S, P, XOR
  - Key schedule
    - PC1, PC2
- Uses many bit-level permutations
  - When implemented in software:
    - Conditional branching!



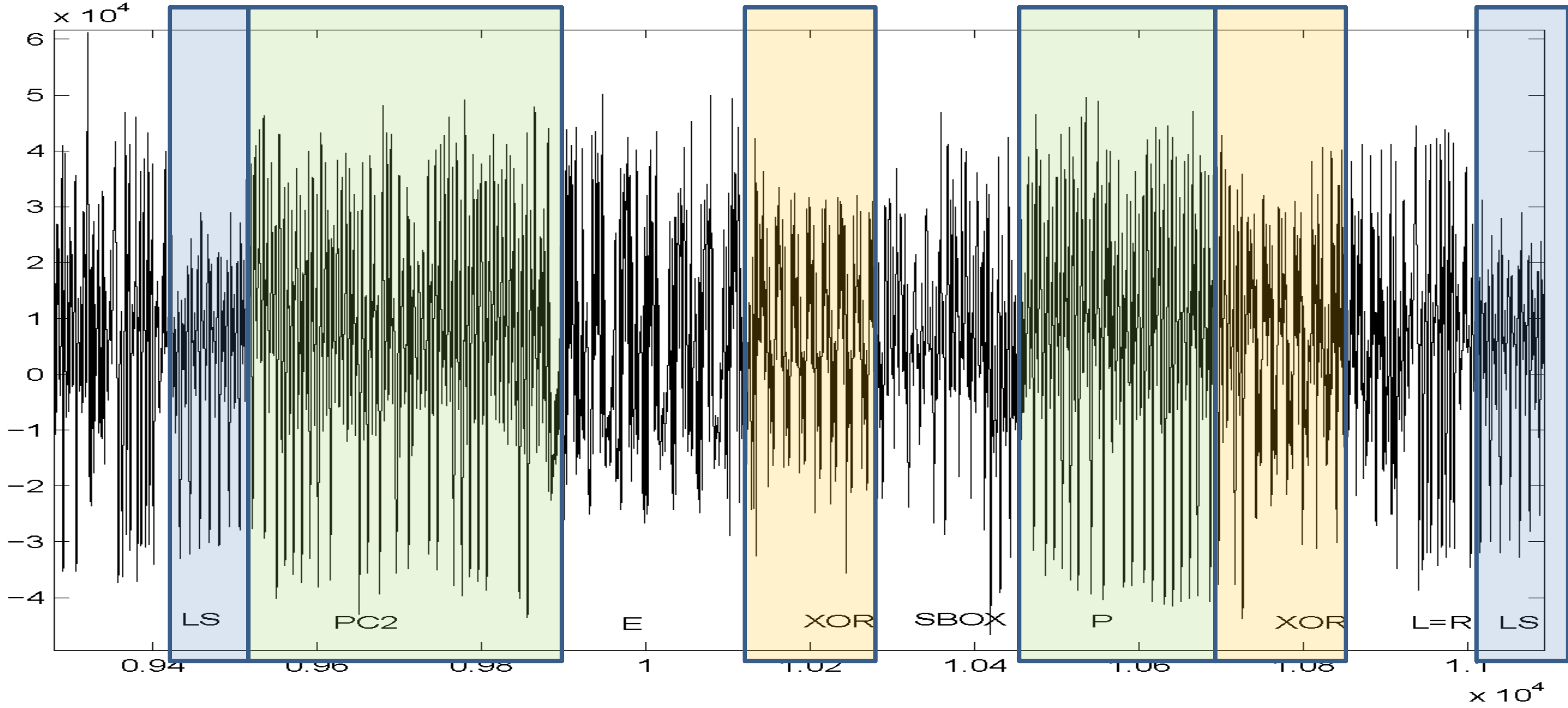


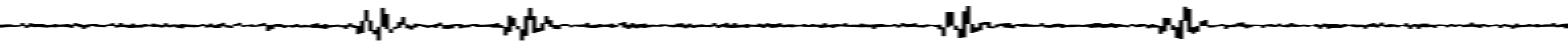


Trace of DES implementation

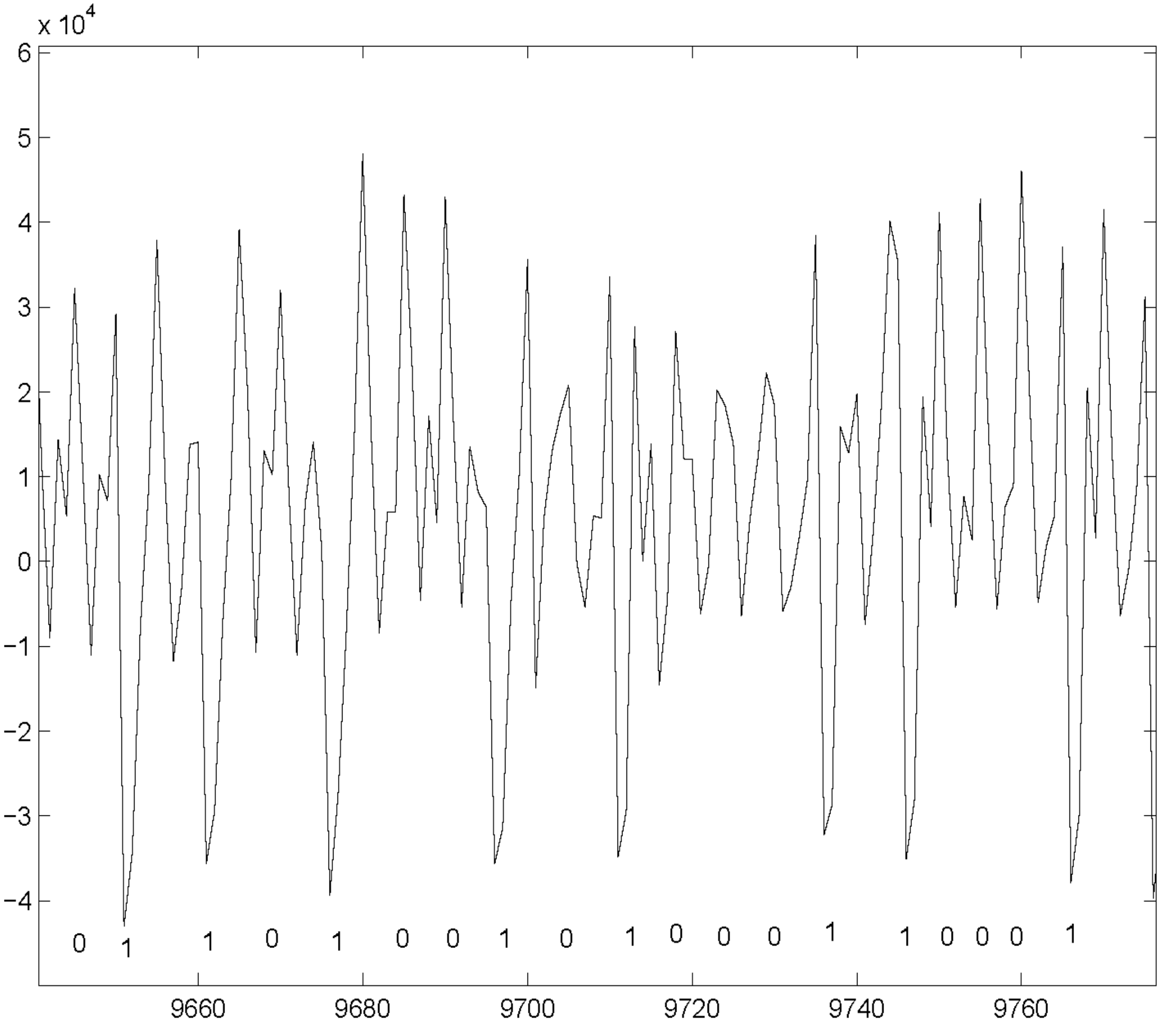


Zooming in: 1 round only





Zooming in on PC2

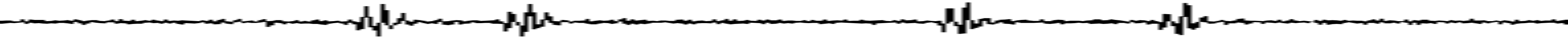


PC2 implementation:

```
input: x = x[0]…x[55]
       t =  t[0]…t[47]
output: y=PC2(x)=y[0]…y[47]

for i=0 to 47
  y[i]=0
  if x[i] =1 then y[t[i]]=1
end
```

Only if x[i]=1 the conditional branching takes place!



SPA on DES Conclusion

- The challenge is to
  - Filter traces such as to have the clearest view on the underlying instructions
  - Find the part which relates to key dependent operations
  - To spot the difference between different peaks
- Clearly the presented example only works because the programmer chose to be efficient
  - And was forced to implement DES in software!
  - Most hardware implementations would not give this kind of leakage as permutations are simple rewiring

Summary

- SPA attacks exploit information within a trace and hence sometimes succeed with a single observation
- One needs to accurately measure and understand ‚what happens when‘
  - Often detailed knowledge about the underlying implementation is required
- But any naive implementation will probably be vulnerable to SPA