

# Design de cryptographie white-box : et à la fin, c'est Kerckhoffs qui gagne

SSTIC 2016

Charles Hubain & Philippe Teuwen  
3 juin 2016



# \$ quisommesnous

Charles Hubain alias @haxelion

Philippe Teuwen alias @doegox

- @ **Quarkslab**
- ♥ le libre, la sécu, les Capture-Ze-Flag
-  <http://wiki.yobi.be> <http://haxelion.eu/>

Note:

*Les résultats présentés ici sont principalement le fruit de recherches menées lorsque nous travaillions pour NXP Semiconductors*



## Nos excuses mais...

Ecole de pensée  
française:

on dit “chiffrer”,  
pas “crypter”



Ecole de pensée  
belge:

“décrypte le cipher”



# BREVE INTRO A LA CRYPTO WHITE-BOX



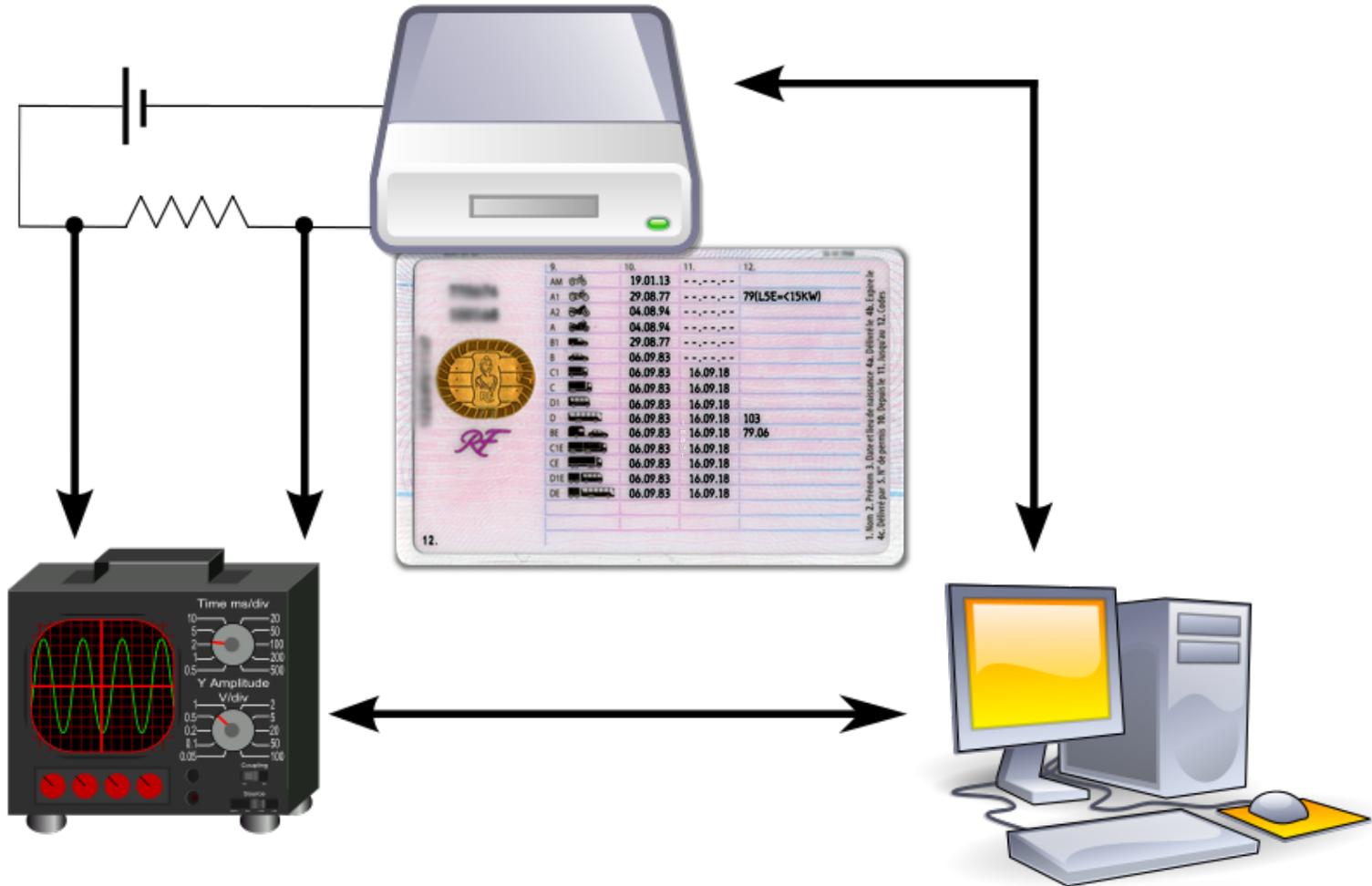
# Rappel: modèle "Black box" => modèle "Grey box"

9.	10.	11.	12.
AM	19.01.13	---	---
A1	29.08.77	---	79(LSE<15KW)
A2	04.08.94	---	---
A	04.08.94	---	---
B1	29.08.77	---	---
B	06.09.83	---	---
C1	06.09.83	16.09.18	---
C	06.09.83	16.09.18	---
D1	06.09.83	16.09.18	---
D	06.09.83	16.09.18	103
BE	06.09.83	16.09.18	79.06
CE	06.09.83	16.09.18	---
CF	06.09.83	16.09.18	---
DIE	06.09.83	16.09.18	---
DE	06.09.83	16.09.18	---

12.

1. Nom 2. Prénoms 3. Date et lieu de naissance 4a. Délivrance 4b. Expire le 4c. Délivré par 5. N° de permis 10. Depuis le 11. Jusqu'au 12. Code

# Modèle "Grey box"





# Modèle “White box”



# Modèle "White box"



CPU - main thread, module Software

Address	Hex dump	ASCII	Disassembly
00401204	6A 60		PUSH 60
00401206	68 38524000		CALL Software.00405238
00401208	E8 80000000		CALL Software.00401F90
00401210	BF 94000000		MOV EDI, 94
00401212	8BC7		MOV EAX, EDI
00401214	E8 D40E0000		CALL Software.004020F0
00401216	9965 E8		MOV DWORD PTR DS:[EBP-18], ESP
00401218	8BF4		MOV ESI, ESP
0040121A	899E		MOV DWORD PTR DS:[ESI], EDI
0040121C	56		PUSH ESI
0040121E	FF15 4C504000		CALL DWORD PTR DS:[<KERNEL32.GetVersionExA
00401220	884E 10		MOV ECX, DWORD PTR DS:[ESI+10]
00401222	9900 88724000		MOV DWORD PTR DS:[487288], ECX
00401224	8846 04		MOV EAX, DWORD PTR DS:[ESI+4]
00401226	8B C4724000		MOV DWORD PTR DS:[4872C4], EAX
00401228	8856 08		MOV EDI, DWORD PTR DS:[ESI+8]
0040122A	8915 C8724000		MOV DWORD PTR DS:[4872C8], EDI
0040122C	8876 0C		MOV ESI, DWORD PTR DS:[ESI+C]
0040122E	91E8 FF700000		RND ESI, 7FF
00401230	8995 BC724000		MOV DWORD PTR DS:[4872BC], ESI
00401232	83F9 02		CMF ECX, 2
00401234	74 0C		JE SHORT Software.00401264
00401236	81CE 00000000		OR ESI, 8000
00401238	8935 BC724000		MOV DWORD PTR DS:[48723C], ESI

Registers (FPU)

EAX: 77411142 kernel32.BaseThreadInitThunk  
ECX: 00000000  
EDX: 00401204 Software.(ModuleEntryPoint)  
EBX: 77F0F900  
ESP: 0012FF8C  
EBP: 0012FF94  
ESI: 00000000  
EDI: 00000000  
EIP: 00401204 Software.(ModuleEntryPoint)

C 0 ES 0023 32bit 01FFFFFFF  
P 1 CS 0018 32bit 01FFFFFFF  
A 0 SS 0023 32bit 01FFFFFFF  
Z 1 DS 0023 32bit 01FFFFFFF  
I 0 FS 0038 32bit 7F000000FFF  
T 0 G0 0000 NULL  
D 0  
O 0  
0 0  
0 0  
ST0 empty 0.0  
ST1 empty 0.0  
ST2 empty 0.0  
ST3 empty 0.0  
ST4 empty 0.0  
ST5 empty 17.00000000000000000000

Address	Hex dump	ASCII	Disassembly
00407000	00 00 00 00 F1 30 40 00	....?..0.	0012FF94 7FF0F000 RETURN to kernel32.77411154
00407002	00 00 00 00 00 00 00 00		0012FF96 77B18429 RETURN to ntdll.77B18429
00407004	C1 21 40 00 F8 29 40 00	..*..*..*..	0012FF98 77F0F900
00407006	00 00 00 00 00 00 00 00	.....	0012FF9C 65DF0D8E
00407008	00 00 00 00 00 00 00 00	.....	0012FFA0 00000000
0040700A	00 00 00 00 00 00 00 00	.....	0012FFA4 00000000
0040700C	4C 15 40 00 02 00 00 00	..L..0..	0012FFA8 77F0F900
0040700E	02 00 00 00 00 00 00 00	.....	0012FFAC 00000000
00407010	08 00 00 00 00 00 00 00	.....	0012FFB0 00000000
00407012	00 00 00 00 00 00 00 00	.....	0012FFB4 00000000
00407014	09 00 00 00 00 00 00 00	.....	0012FFB8 00000000
00407016	0A 00 00 00 F9 54 40 00	.....T..	0012FFBC 00000000
00407018	10 00 00 00 C4 54 40 00	.....T..	0012FFC0 00000000
0040701A	12 00 00 00 70 54 40 00	.....T..	0012FFC4 00000000
0040701C	00 00 00 00 00 00 00 00	.....	0012FFC8 00000000
0040701E	13 00 00 00 44 54 40 00	.....T..	0012FFCC 147C2055 End of SEH chain
00407020	18 00 00 00 0C 54 40 00	.....T..	0012FFD0 00000000 SE handler
00407022	19 00 00 00 E4 53 40 00	.....T..	0012FFD4 00000000
00407024	00 00 00 00 00 00 00 00	.....	0012FFD8 00000000
00407026	1A 00 00 00 FC 53 40 00	.....NS..	0012FFDC 77B18429 RETURN to ntdll.77B18429

IDA - /Users/jg/idasrc/current/ida/tests/input/pc\_linux\_apcall.elf

IDA View-A, Pseudocode-A, Hex View-A, Structures, Enums, Imports, Exports

loc\_8049F95:  
mov dword ptr [esp+0], 0  
mov dword ptr [esp+4], offset aHELP\n  
mov [esp], eax ; haystack  
call strcat ; strcat  
mov [ebp+var\_280C], eax  
cmp [ebp+var\_280C], 0  
je short loc\_8049FAD

loc\_8049FAD:  
lea eax, [ebp+var\_2814]  
lea edx, [eax+4]  
mov [ebp+var\_2818], eax  
mov [eax], [edx]  
mov [ebp+var\_2810], eax  
mov [ebp+var\_2814], eax  
mov [ebp+var\_280C], eax  
mov dword ptr [esp+4], 2000h ; maxlen  
lea eax, [ebp+var\_280C]  
mov [esp], eax ; int  
call strlen ; strlen  
mov [esp+4], esp  
mov [ebp+var\_280C], eax  
call printf ; printf, offset aSS : "%s\n"  
cmp [ebp+var\_2820], 0  
je short loc\_8049F95

int result; // max80  
char v8; // [esp+20] [bp-282Ch]013  
char v9; // [sp+00] [bp-2820h]010  
int haystack; // [esp+40] [bp-2824h]04  
int v10; // [esp+80] [bp-2828h]04  
int v11; // [esp+84] [bp-2828h]04  
int v12; // [esp+88] [bp-2828h]04  
int v13; // [esp+8C] [bp-2828h]04

v13 = MK\_FP((0, 20));  
\*\_DWORD v14 = 0;  
switch ( a2 )  
{  
case 23:  
sprintf((int)v13, 0x2000, a3, a4);  
\*(DWORD \*)a1 = printf("%s", v13);  
if ( v5 )  
v15 = a4;  
v16 = a3 == 597061684 && (v11 == a5, a6 == -1427216450);  
if ( v5 )  
{  
sprintf("%i, %i", v11, (int)"ed,ad", 5);  
\*(BYTE \*)a1 = 1;  
break;  
case 21:  
v10 = a3;  
for ( haystack = a6; haystack = (int)(v7 + 8) ;  
while ( !strcmp(const char \*)haystack, "ICON ".5u)  
!strcmp(const char \*)haystack, "AUTORISE", 9u)  
!strcmp(const char \*)haystack, "01..3u) )  
{  
v8 = strchr(const char \*)haystack, 10);  
if ( !v8 )  
break;  
haystack = (int)(v8 + 1);  
if ( !strcmp(const char \*)haystack, "HELP\n", 5u) )  
- strcat(const char \*)haystack, "ENDHELP\n");  
if ( v7 )  
continue;  
break;  
CALLUI:15

Output window

```
term() called!
-----
Python 2.6.1 (r261:67515, Feb 11 2010, 00:51:29)
[GCC 4.2.1 (Apple Inc. build 5646)]
IDAPython v1.4.2 final (serial 0) (c) The IDAPython Team <idapython@googlegroups.com>
```



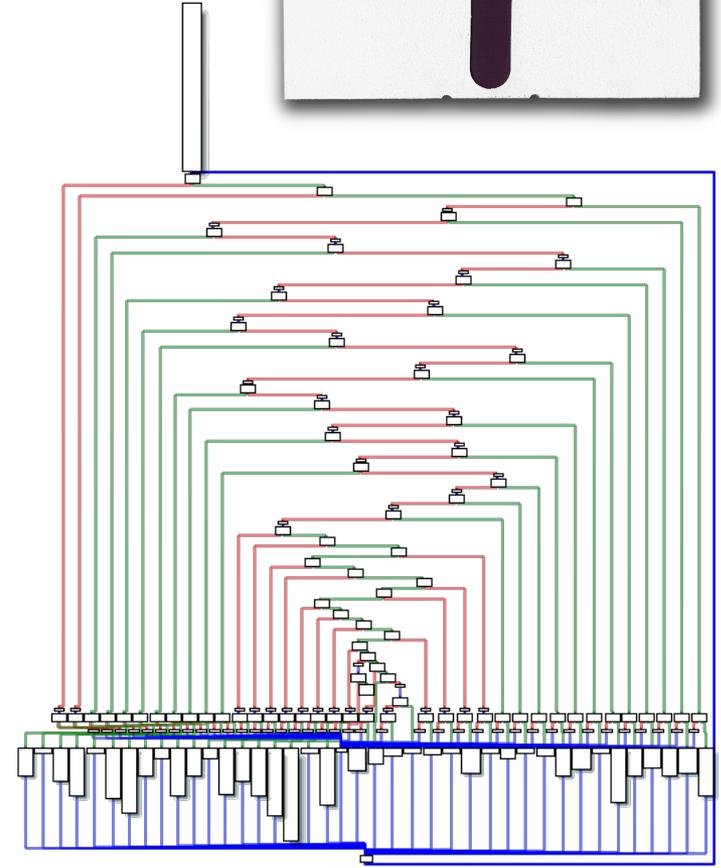
# Ligne de défense = implémentation

Obscurcissement de code

Vérifications d'intégrité

Astuces "anti-debug"

```
mov eax,0x0          mov ebx,[eax+0x80a051e]  mov edx,0x0
mov ax,[0x80a0451]   mov eax,[ebx]           mov dx,[eax+eax+0x80c0bba]
mov byte [eax+0x80e17bc],0x0  mov edx,0x0           mov [ebx],edx
mov al,[eax+0x80e17bc]  mov dx,[eax+eax+0x80c0bba]  mov eax,[0x80a0556]
mov [0x80a0451],al      mov [ebx],edx          mov ebx,[eax+0x80a051e]
mov eax,[0x80a0556]    mov eax,[0x80a0556]    mov eax,[ebx]
mov edx,[eax+0x80a058e]  mov ebx,[eax+0x80a051e]  mov edx,0x0
mov eax,[0x80a0451]    mov eax,[ebx]         mov dx,[eax+eax+0x80c0bba]
mov eax,[eax+edx]     mov edx,0x0           mov [ebx],edx
mov [0x80a044d],eax    mov dx,[eax+eax+0x80c0bba]  mov eax,[0x80a0556]
mov eax,[0x80a044d]   mov [ebx],edx        mov ebx,[eax+0x80a051e]
mov eax,[eax+0x80a054e]  mov eax,[0x80a0556]    mov eax,[ebx]
mov dword [eax],0x139  mov ebx,[eax+0x80a051e]  mov edx,0x0
mov eax,[0x80a044d]   mov eax,[ebx]        mov dx,[eax+eax+0x80c0bba]
```



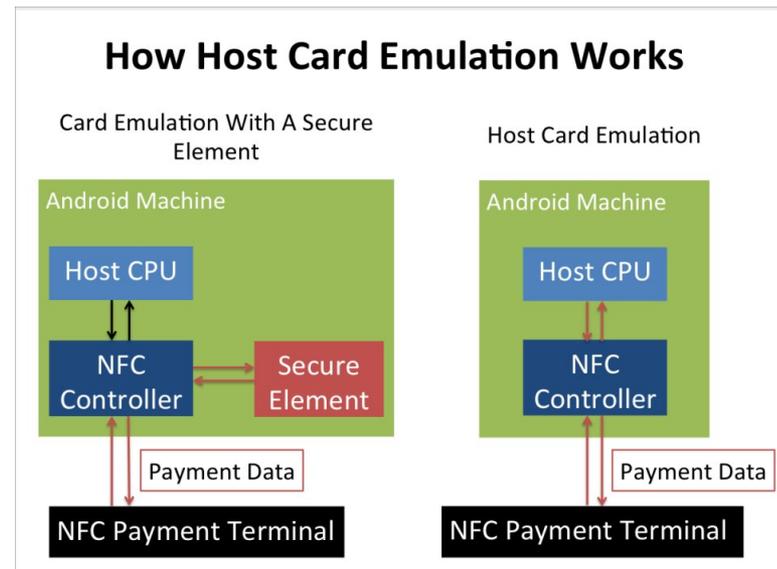
# Cryptographie dans le modèle “White box”

Pourquoi faire?

- Gestion de droits numériques (DRM) ↔ criminels utilisateurs
- Paiement mobile, HCE ↔ code malveillants



Source: “l'industrie du film”



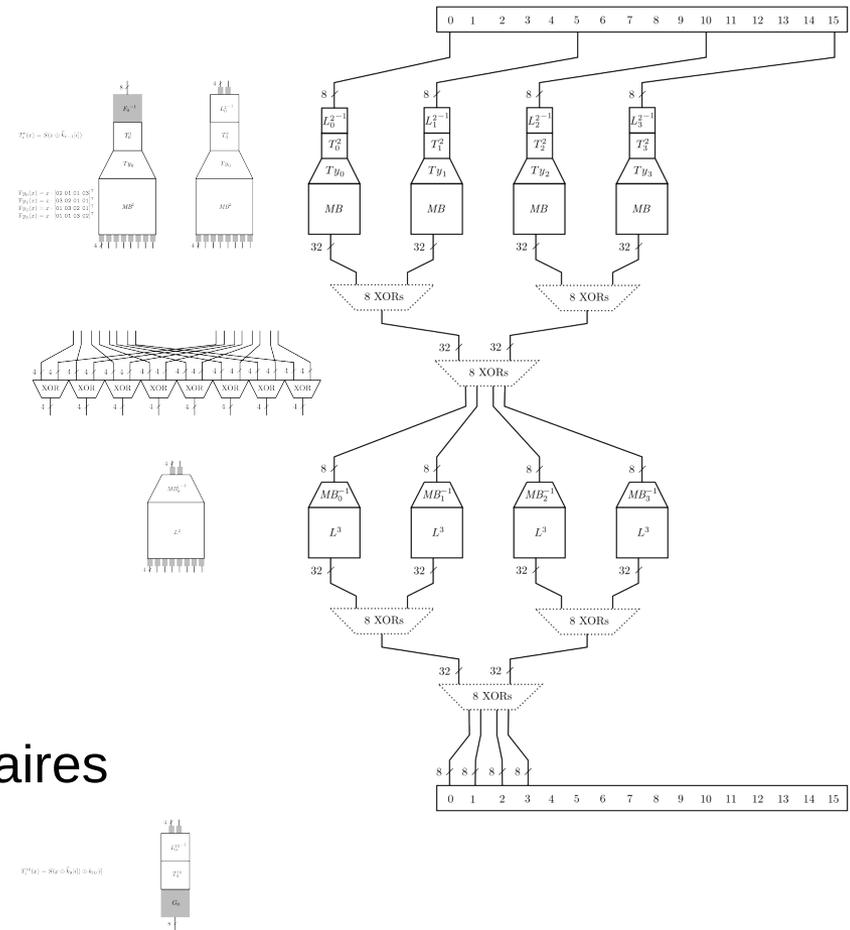
Source: Business Insider

# Cryptographie “White box”

Obscurcissement de code: inadéquat

→ Chow et al. (2002)

- Implémentation “idéale”:  
4.94 x 10<sup>27</sup> TB
- Implémentation pratique d’AES:  
752kB  
Encodage des valeurs intermédiaires



# Cryptographie “White box”

Petit résumé:

- Attaques académiques → nouveaux designs → ...
- Tous les modèles académiques ont été cassés

Réponse de l'industrie:

- Gardons nos designs secrets
- Enterrons les implémentations sous des couches d'obscurcissement, de vérif. d'intégrité, trucs “anti-debug”
- Hé ! Regardez on a un Secure Element logiciel !



# Attaques “académiques”?

1. défaire les protections
2. être *aware* du design
3. appliquer l’attaque:

**Definition 3.** The mapping  $\overline{AES}_{enc}^{(r,j)} : (\mathbf{F}_2^8)^4 \rightarrow (\mathbf{F}_2^8)^4$  for  $1 \leq r \leq 9$  and  $0 \leq j \leq 3$ , called an encoded AES subround with byte permutations, is defined by

$$\overline{AES}_{enc}^{(r,j)} = (Q_0^{(r,j)}, Q_1^{(r,j)}, Q_2^{(r,j)}, Q_3^{(r,j)}) \circ \overline{AES}^{(r,j)} \circ (P_0^{(r,j)}, P_1^{(r,j)}, P_2^{(r,j)}, P_3^{(r,j)}),$$

where the mapping  $\overline{AES}^{(r,j)}$  is defined by

$$\Pi_2^{(r,j)} \circ AES^{(r,\pi^{(r)}(j))} \circ \Pi_1^{(r,j)} = MC^{(r,j)} \circ (S, S, S, S) \circ \bigoplus_{[k_i^{(r,j)}]_{0 \leq i \leq 3}},$$

$$\begin{aligned} \text{with } [k_i^{(r,j)}]_{0 \leq i \leq 3} &= (\Pi_1^{(r,j)})^{-1} ([k_i^{(r,\pi^{(r)}(j))}]_{0 \leq i \leq 3}) \\ \text{and } MC^{(r,j)} &= \Pi_2^{(r,j)} \circ MC \circ \Pi_1^{(r,j)}. \end{aligned}$$

As a result of the algorithm mentioned above, the white-box adversary has black-box access to the following structures of each round  $R_r |_{r=1, \dots, 10}$ :

$$\begin{cases} SR \circ \bigoplus_{K'_{10}} \circ \{S_{10,i}\}_{i=0, \dots, 15} \circ \bigoplus_{K_9} \circ A_9'^{-1} & \text{for } R_{10}, \\ MC \circ SR \circ A_r'' \circ \{S_{r,i}\}_{i=0, \dots, 15} \circ \bigoplus_{K_{r-1}} \circ A_{r-1}'^{-1} & \text{for } R_r |_{2 \leq r \leq 9}, \\ MC \circ SR \circ A_1'' \circ \{S_{1,i}\}_{i=0, \dots, 15} \circ \bigoplus_{K_0} & \text{for } R_1, \end{cases} \quad (5)$$

The second step then implements the function  $\tau_{r,i}^k$  in which  $\mu_r(n)$  describes the corresponding position of the bit in the output of the t-boxes, and  $PB$  is the DES p-box operation:

$$\tau_{r,i}^k(x)(L_r^i, R_r^i) = \left( \underbrace{\alpha_{r,i}^k(x|_{8\gamma_r(i)}, x|_{8\gamma_r(i)+4}, x|_{8\gamma_r(i)+5})}_{\text{depends on } R_{r-1} \text{ only}} \right) \parallel \left( EP_i \left[ PB \left( \underbrace{x|_{\gamma_r(0)}^4 \parallel x|_{\gamma_r(1)}^4 \parallel \dots \parallel x|_{\gamma_r(11)}^4}_{\text{depends on } R_{r-1} \text{ only}} \right) \oplus \left( \underbrace{x|_{\mu_r(0)} \parallel \dots \parallel x|_{\mu_r(32)}}_{\text{depends on } L_{r-1} \text{ only}} \right) \right] \right)$$

$$\tau_r^k(x) = \tau_{r,0}^k(x) \parallel \tau_{r,1}^k(x) \parallel \dots \parallel \tau_{r,11}^k(x)$$

$\psi_r$  and  $\phi_r$  are different non-linear bijective encodings on 4-bit blocks, and  $\delta_r$

$$\delta_r(L, R') = \gamma_r(\mu_r((L \parallel 0^{24}), R'))$$

$$\begin{aligned} \mu_r(x_0 x_1 \dots x_{47}, y_0 \dots y_{47}) &= y_0 \dots y_5 x_{\mu_r^{-1}(0)} x_{\mu_r^{-1}(1)} y_6 \dots y_{11} x_{\mu_r^{-1}(2)} x_{\mu_r^{-1}(3)} \dots y_{42} \dots y_{47} x_{\mu_r^{-1}(22)} x_{\mu_r^{-1}(23)} \dots x_{\mu_r^{-1}(47)} \\ \gamma_r(z_0 z_1 \dots z_{95}) &= z_{\gamma_r^{-1}(0)} \dots z_{\gamma_r^{-1}(0)+5} z_6 z_7 \dots z_{\gamma_r^{-1}(11)} \dots z_{\gamma_r^{-1}(11)+5} z_{94} z_{95} \end{aligned}$$

The obfuscated t-box is

$$T_r^{fk}(x) = (\phi_r T_r^k \psi_{r-1}^{-1})(x).$$

Hence the transformed function is:

$$E^k(x) = [(\lambda^{-1} \delta_n^{-1} \psi_n^{-1}) \cdot (\psi_n \delta_n \tau_n^k \phi_n^{-1}) \cdot (\phi_n T_n^k \psi_{n-1}^{-1}) \cdot \dots \cdot (\psi_1 \delta_1 \tau_1^k \phi_1^{-1}) \cdot (\phi_1 T_1^k \psi_0^{-1}) \cdot (\psi_0 \delta_0 \beta \lambda)](x)$$

with

$$\beta(L, R) = L \parallel EP(R)$$

By setting

$$\tau_r^k = \begin{cases} \psi_0 \delta_0 \beta \lambda & r = 0 \\ \psi_r \delta_r \tau_r^k \phi_r^{-1} & r = 1, \dots, n \\ \lambda^{-1} \delta_n^{-1} \psi_n^{-1} & r = n + 1 \end{cases}$$

the resulting encryption operation is

$$E^k(x) = [\tau_{n+1}^k \cdot (\tau_n^k \cdot T_n^k) \cdot \dots \cdot (\tau_1^k \cdot T_1^k) \cdot \tau_0^k](x)$$

Extraits de:

- “Two Attacks on a White-Box AES”
- “Cryptanalysis of a Perturbated White-Box AES Implementation”
- “Attacking an obfuscated cipher by injecting faults”

# Attques “académiques”?

**Definition 3.** The mapping  $\overline{AES}_{enc}^{(r,j)} : (\mathbf{F}_2^8)^4 \rightarrow (\mathbf{F}_2^8)^4$  for  $1 \leq r \leq 9$  and  $0 \leq j \leq 3$ , called an encoded AES subround with byte permutations, is defined by

$$\overline{AES}_{enc}^{(r,j)} = (Q_0^{(r,j)}, Q_1^{(r,j)}, Q_2^{(r,j)}, Q_3^{(r,j)}) \circ \overline{AES}^{(r,j)} \circ (P_0^{(r,j)}, P_1^{(r,j)}, P_2^{(r,j)}, P_3^{(r,j)}),$$

where the mapping  $\overline{AES}^{(r,j)}$  is defined by

$$\overline{AES}^{(r,j)} \circ AES^{(r,\pi^{(r)}(j))} \circ \dots \circ AES^{(r,\pi^{(r)}(j))} = MC^{(r,j)} \circ (S, S, S, S) \circ \bigoplus_{i=0}^3 \dots$$

with  $[k_i^{(r,j)}]_{i=0, \dots, 15}^{-1} = ([k_i^{(r,\pi^{(r)}(j))}]_{i=0, \dots, 15})^{-1}$  and  $MC^{(r,j)} = MC \circ \Pi_1^{(r,j)}$ .

of the algorithm mentioned above, the white-box adversary has black-box access to the following structures of each round  $R_r|_{r=1, \dots, 10}$ :

$$\begin{cases} SR \circ \bigoplus_{K_{10}'} \circ \{S_{10,i}\}_{i=0, \dots, 15} \circ \bigoplus_{K_9} \circ A_9'^{-1} & \text{for } R_{10}, \\ MC \circ SR \circ A_r'' \circ \{S_{r,i}\}_{i=0, \dots, 15} \circ \bigoplus_{K_{r-1}} \circ A_{r-1}'^{-1} & \text{for } R_r|_{2 \leq r \leq 9}, \\ MC \circ SR \circ A_1'' \circ \{S_{1,i}\}_{i=0, \dots, 15} \circ \bigoplus_{K_0} & \text{for } R_1, \end{cases} \quad (5)$$

The second step then implements the function  $\tau_{r,i}^k$  in which  $\mu_r(n)$  describes the corresponding position of the bit in the output of the t-boxes, and  $PB$  is the DES p-box operation:

$$\tau_{r,i}^k(x)(L_r^i, R_r^i) = \left( \underbrace{\alpha_{r,i}^k(x|_{8\gamma_r(i)}, x|_{8\gamma_r(i)+4}, x|_{8\gamma_r(i)+5})}_{\text{depends on } R_{r-1} \text{ only}} \right) \parallel \left( EP_i \left[ PB \left( \underbrace{x|_{\gamma_r(0)}^4 \parallel x|_{\gamma_r(1)}^4 \parallel \dots \parallel x|_{\gamma_r(11)}^4}_{\text{depends on } R_{r-1} \text{ only}} \right) \oplus \left( \underbrace{x|_{\mu_r(0)} \parallel \dots \parallel x|_{\mu_r(32)}}_{\text{depends on } L_{r-1} \text{ only}} \right) \right] \right)$$

$$\tau_r^k(x) = \tau_{r,0}^k(x) \parallel \tau_{r,1}^k(x) \parallel \dots \parallel \tau_{r,11}^k(x)$$

$\psi_r$  and  $\phi_r$  are different non-linear bijective encodings on 4-bit blocks, and  $\delta_r$

$$\mu_r(x_0x_1 \dots x_4 y_0 \dots y_4 z_0 \dots z_4) = y_0 \dots y_5 x_{\mu_r^{-1}(0)} x_{\mu_r^{-1}(1)} \dots x_{\mu_r^{-1}(2)} x_{\mu_r^{-1}(3)} \dots y_4 x_{\mu_r^{-1}(4)} \dots x_{\mu_r^{-1}(9)} z_0 \dots z_4 = z_{\gamma_r^{-1}(0)} \dots z_{\gamma_r^{-1}(0)+5} \dots z_{\gamma_r^{-1}(11)+5} \dots z_{94} z_{95}$$

The obfuscation function is

$$T_r^k(x) = (\dots) \circ \tau_r^k(x) \circ (\dots)$$

Hence the transformation function is:

$$E^k(x) = [(\lambda^{-1} \delta_n^{-1} \psi_n^{-1}) \cdot (\psi_n \delta_n \tau_n^k \phi_n^{-1}) \cdot (\phi_n T_n^k \psi_{n-1}^{-1}) \cdot \dots \cdot (\psi_{10} \tau_{10}^k \phi_1^{-1}) \cdot (\phi_1 T_1^k \psi_0^{-1}) \cdot (\psi_0 \delta_0 \lambda)](x)$$

with

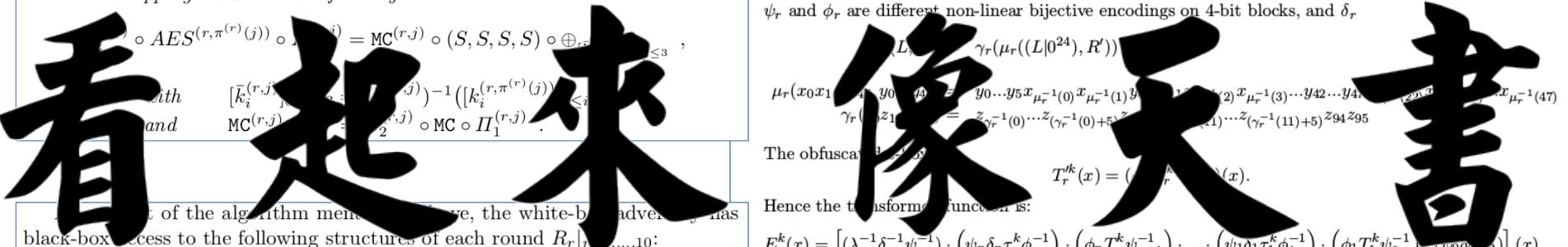
$$\beta(L, R) = L \parallel EP(R)$$

By setting

$$\tau_r^k = \begin{cases} \psi_0 \delta_0 \beta \lambda & r = 0 \\ \psi_r \delta_r \tau_r^k \phi_r^{-1} & r = 1, \dots, n \\ \lambda^{-1} \delta_n^{-1} \psi_n^{-1} & r = n + 1 \end{cases}$$

the resulting encryption operation is

$$E^k(x) = [\tau_{n+1}^k \cdot (\tau_n^k \cdot T_n^k) \cdot \dots \cdot (\tau_1^k \cdot T_1^k) \cdot \tau_0^k](x)$$





# TRACES D'EXECUTION

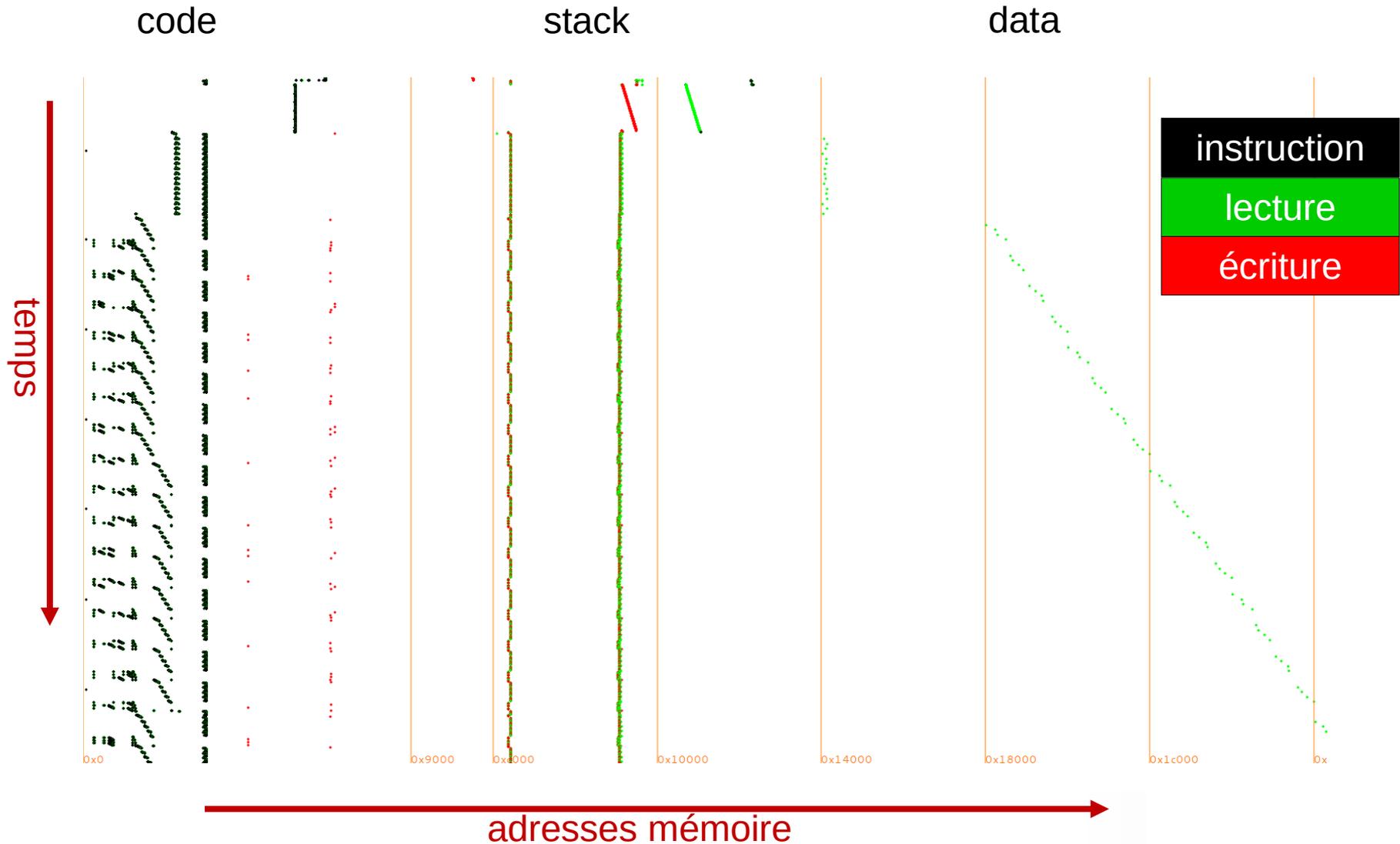
# Tracer un exécutable

Enregistrer toutes les instructions et les accès mémoire

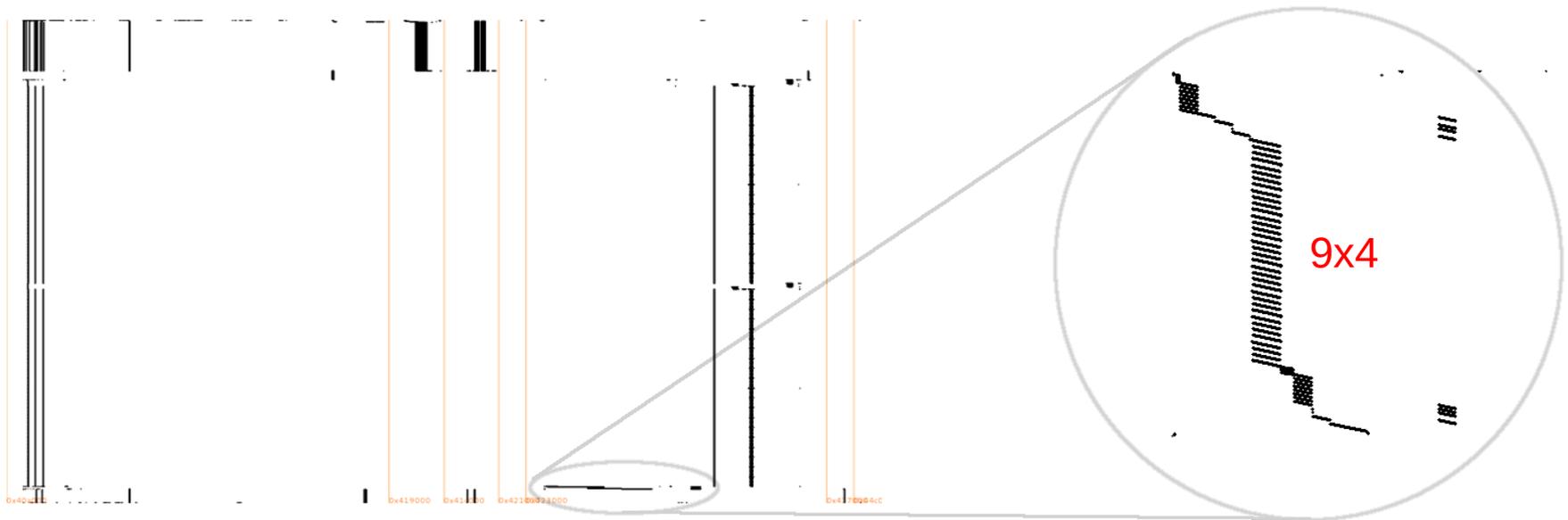
Exemples:

- Intel PIN
- Valgrind
- Instrumenter une VM (Java, Python,...)
- Instrumenter un émulateur

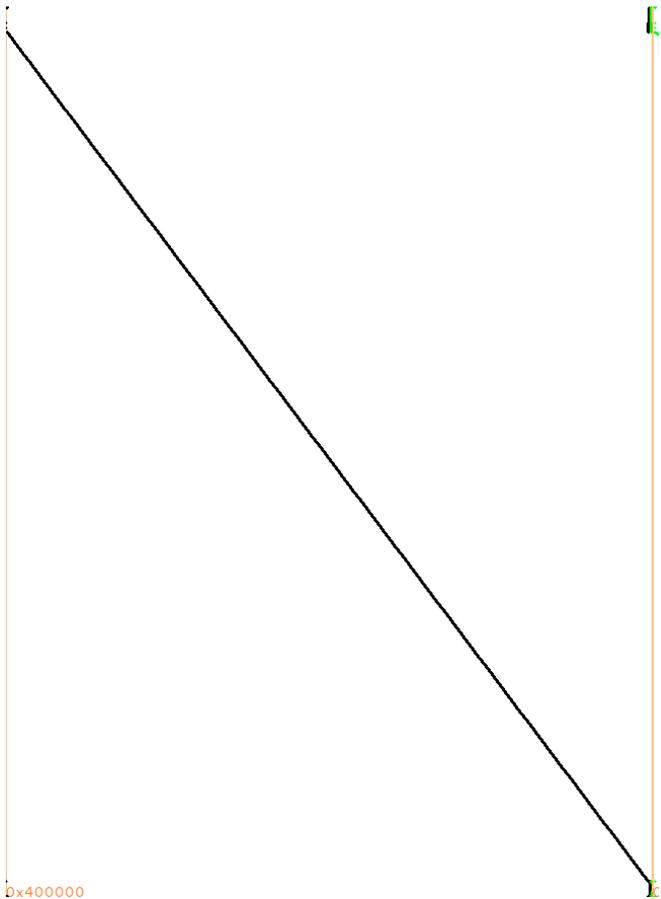
# Même convention que pTra de Quarkslab



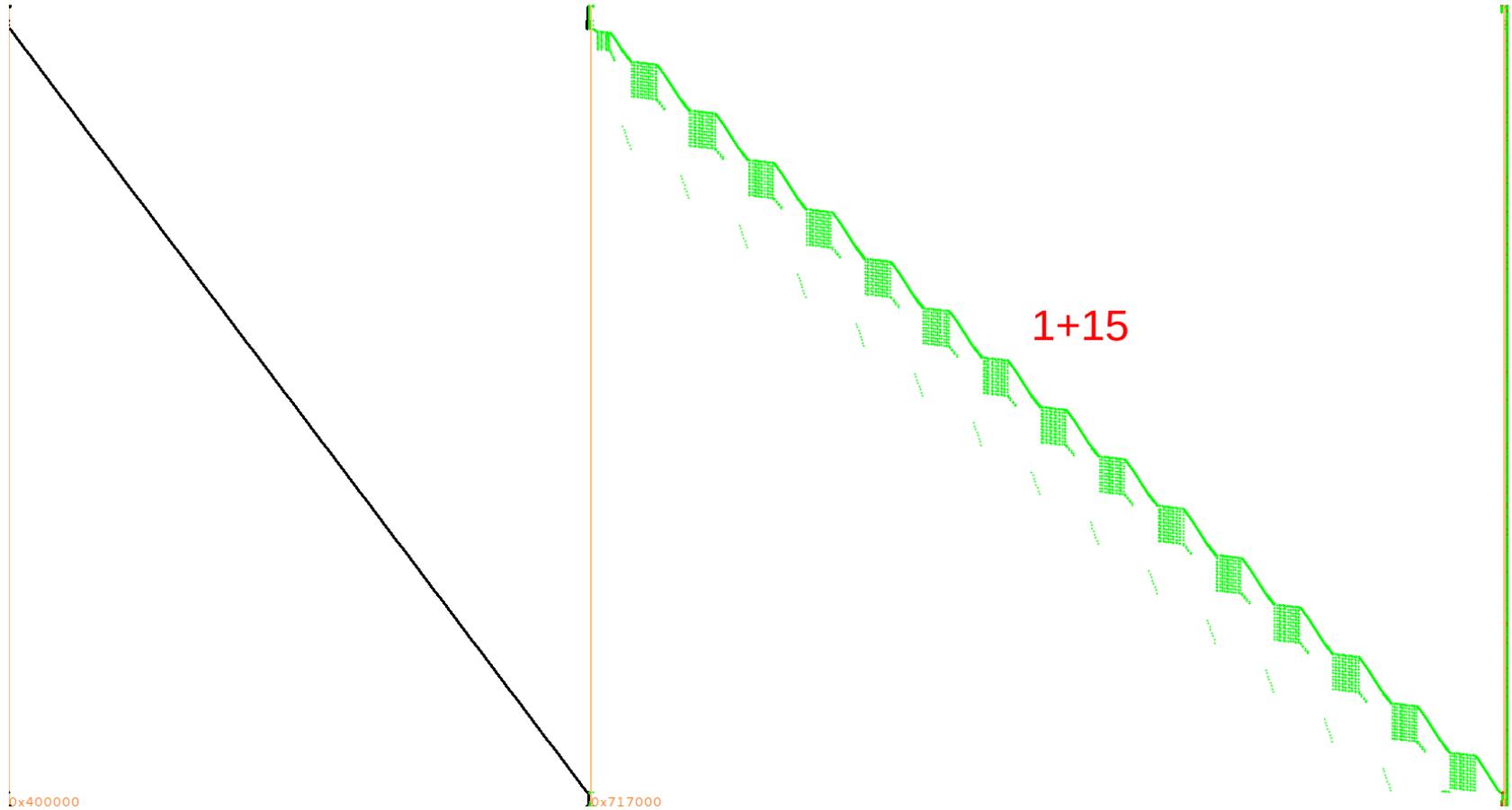
# Identification visuelle de crypto: par le code



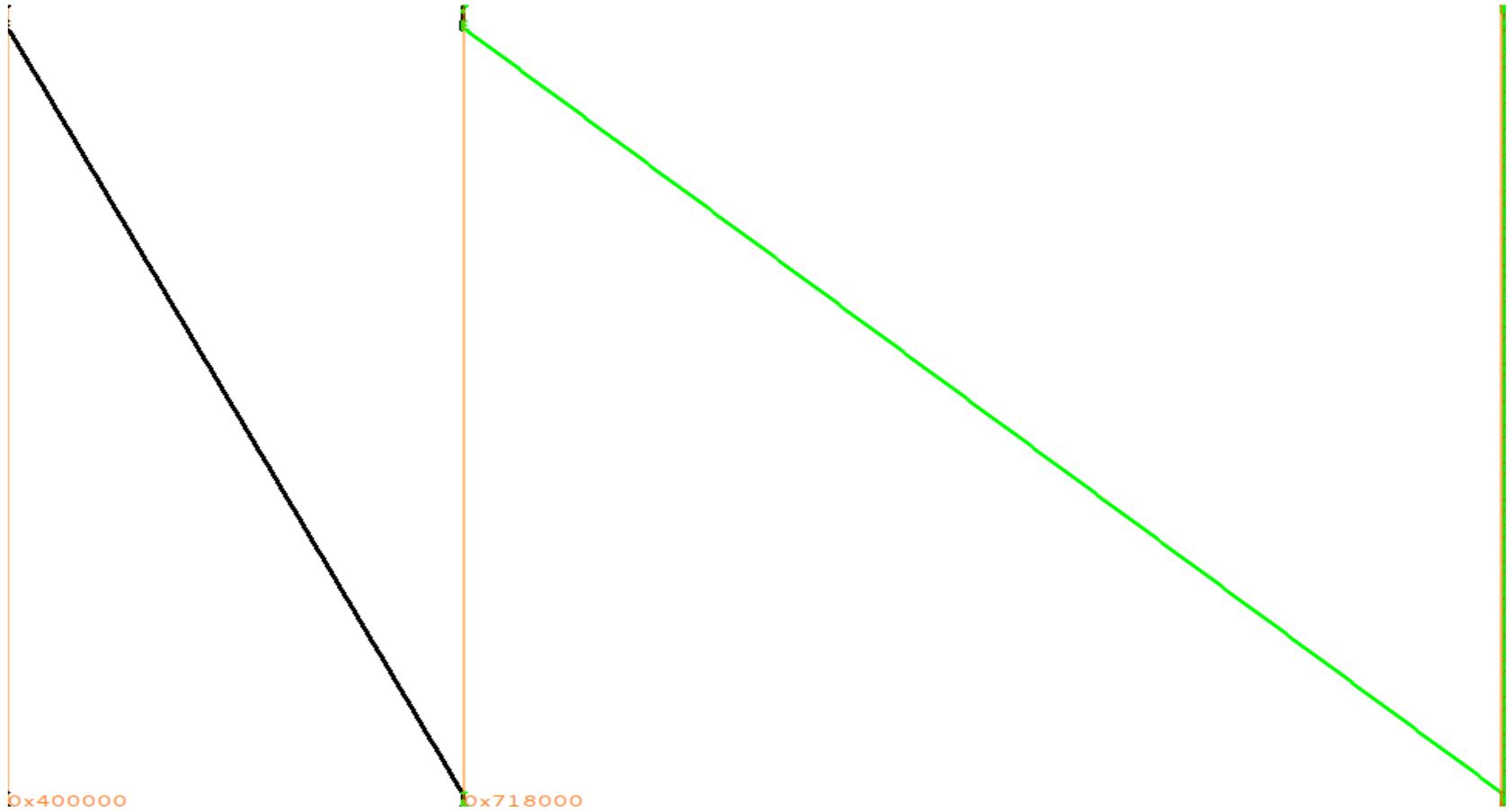
# Identification visuelle de crypto: par le code ?



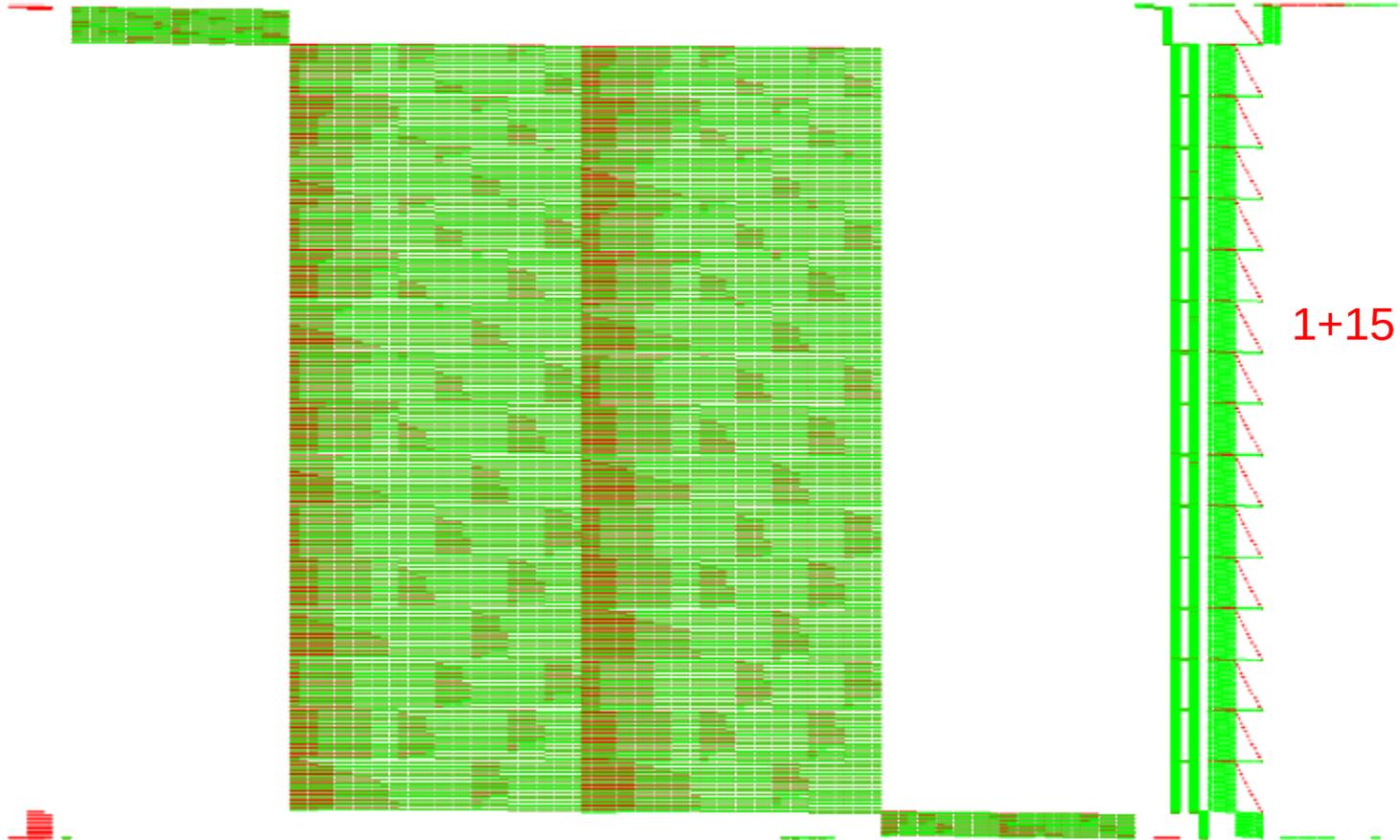
# Identification visuelle de crypto: par les données !



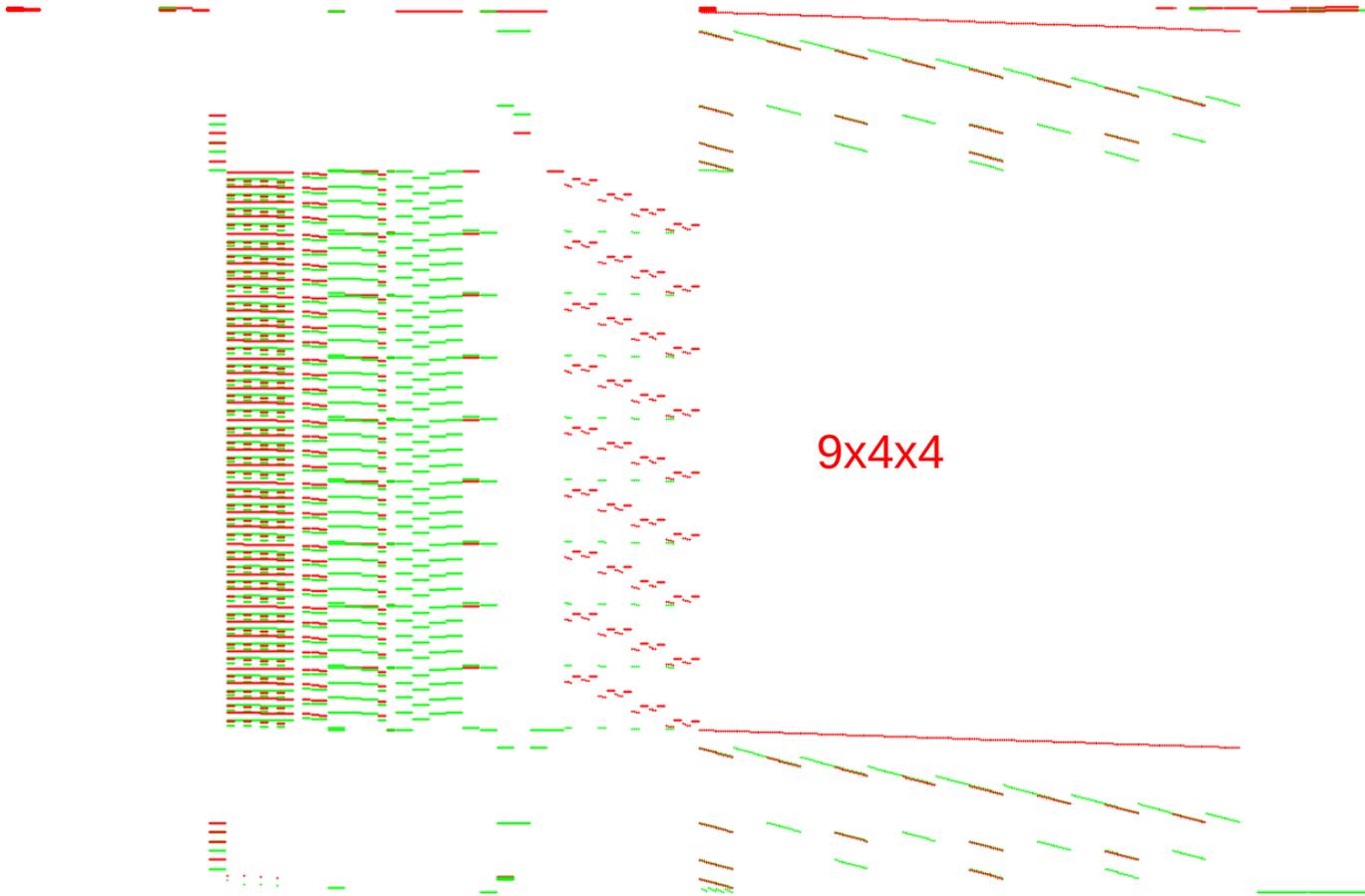
# Identification visuelle de crypto: par les données ?



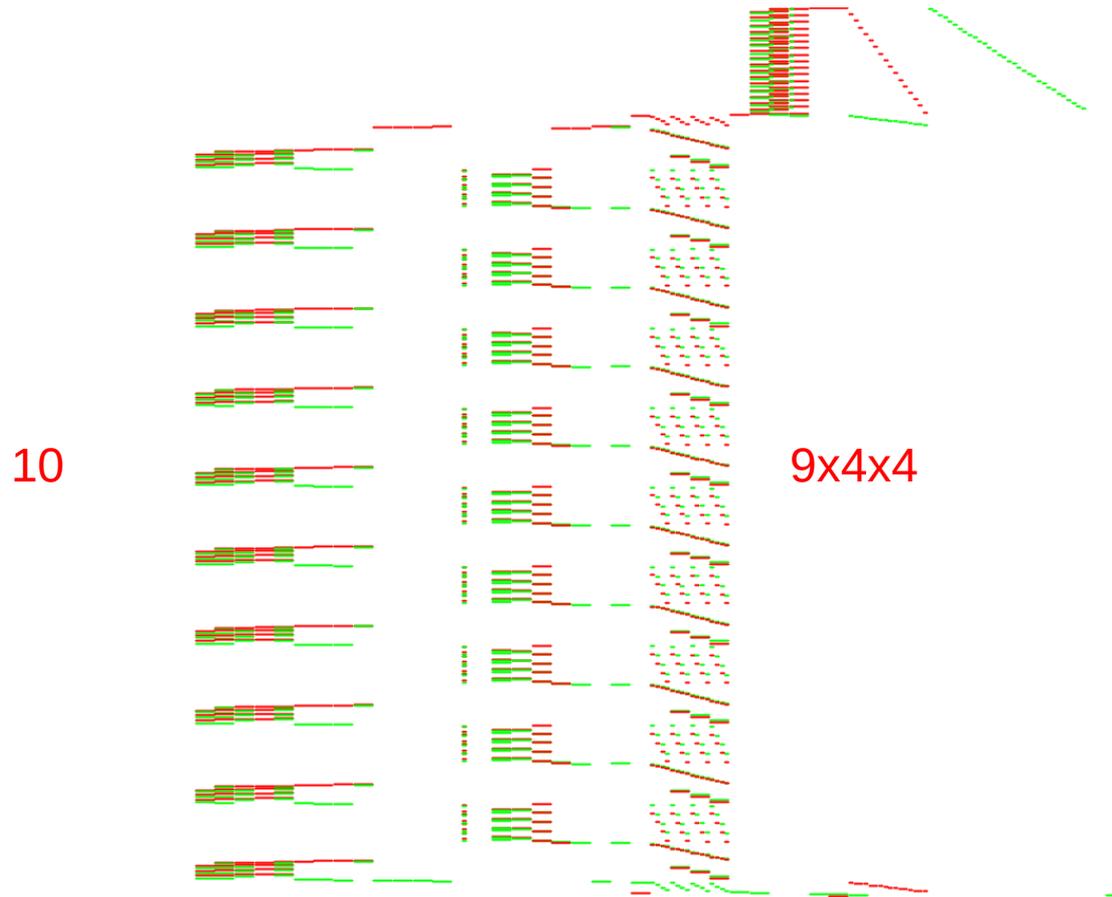
# Identification visuelle de crypto: par la pile !



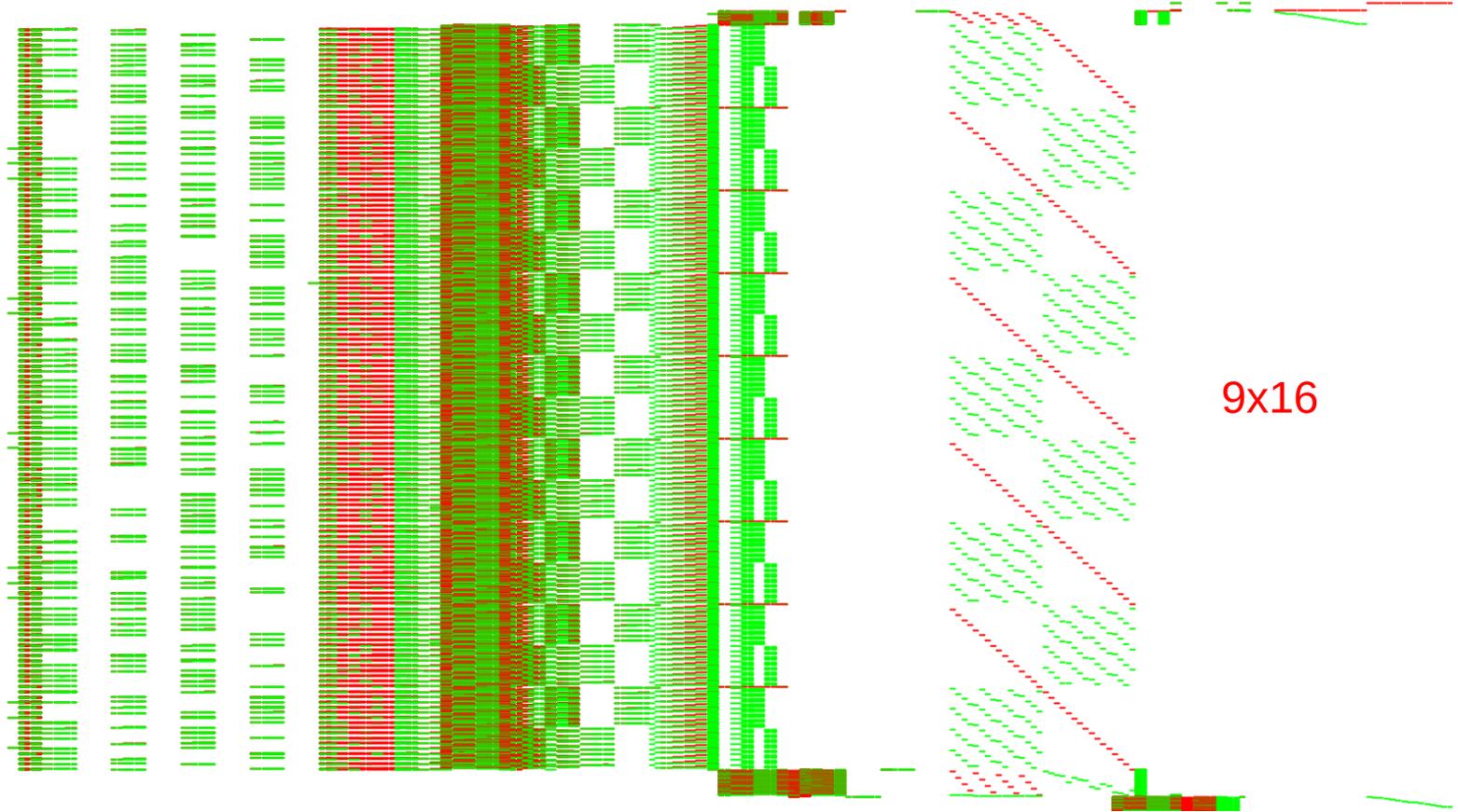
# Identification visuelle de crypto: par la pile !



# Identification visuelle de crypto: par la pile !



# Identification visuelle de crypto: par la pile !



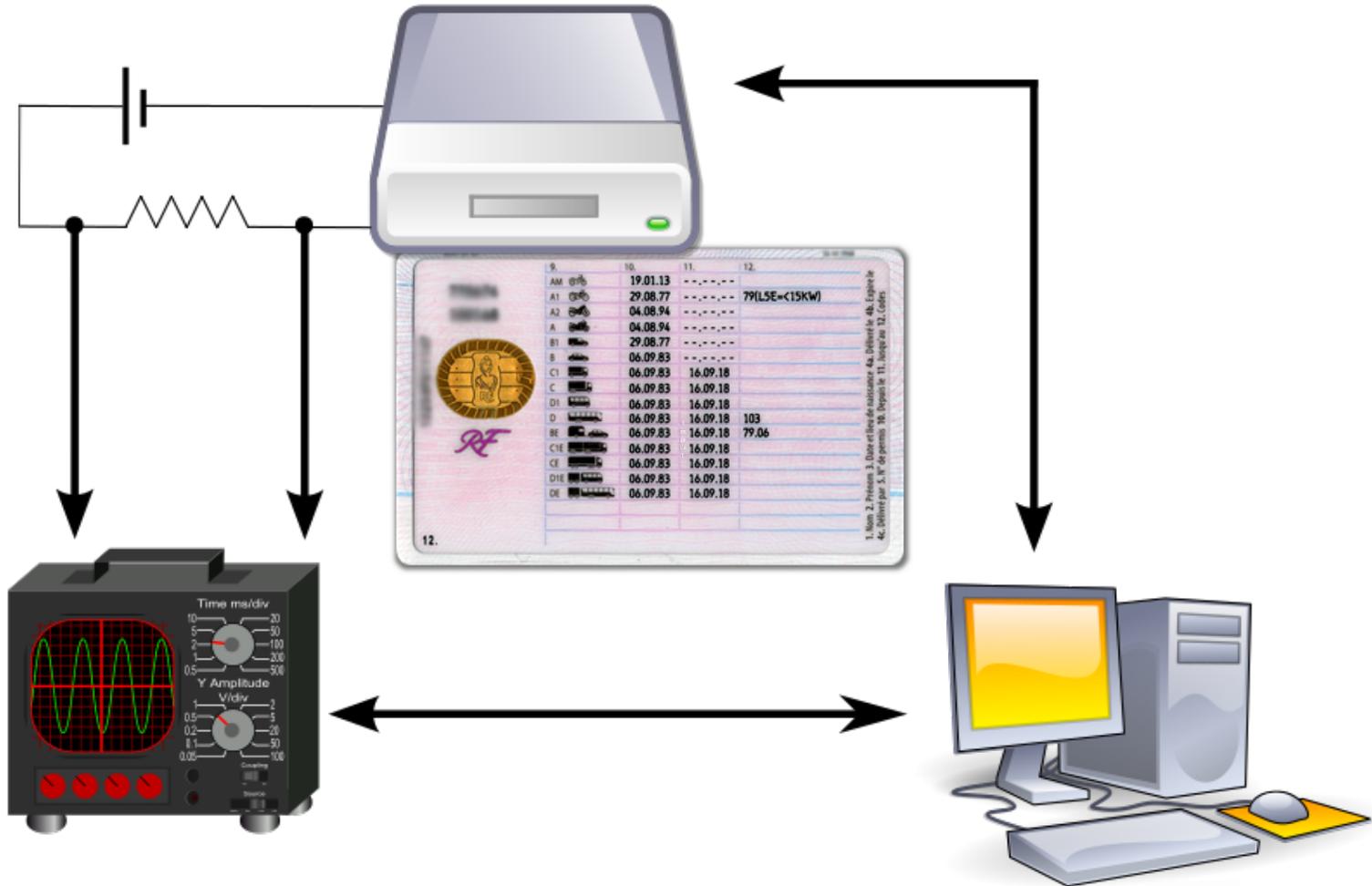
 So What? 

 You and 82 others don't give a fuck.

Et ma clé dans tout ça ?

# DIFFERENTIAL COMPUTATION ANALYSIS

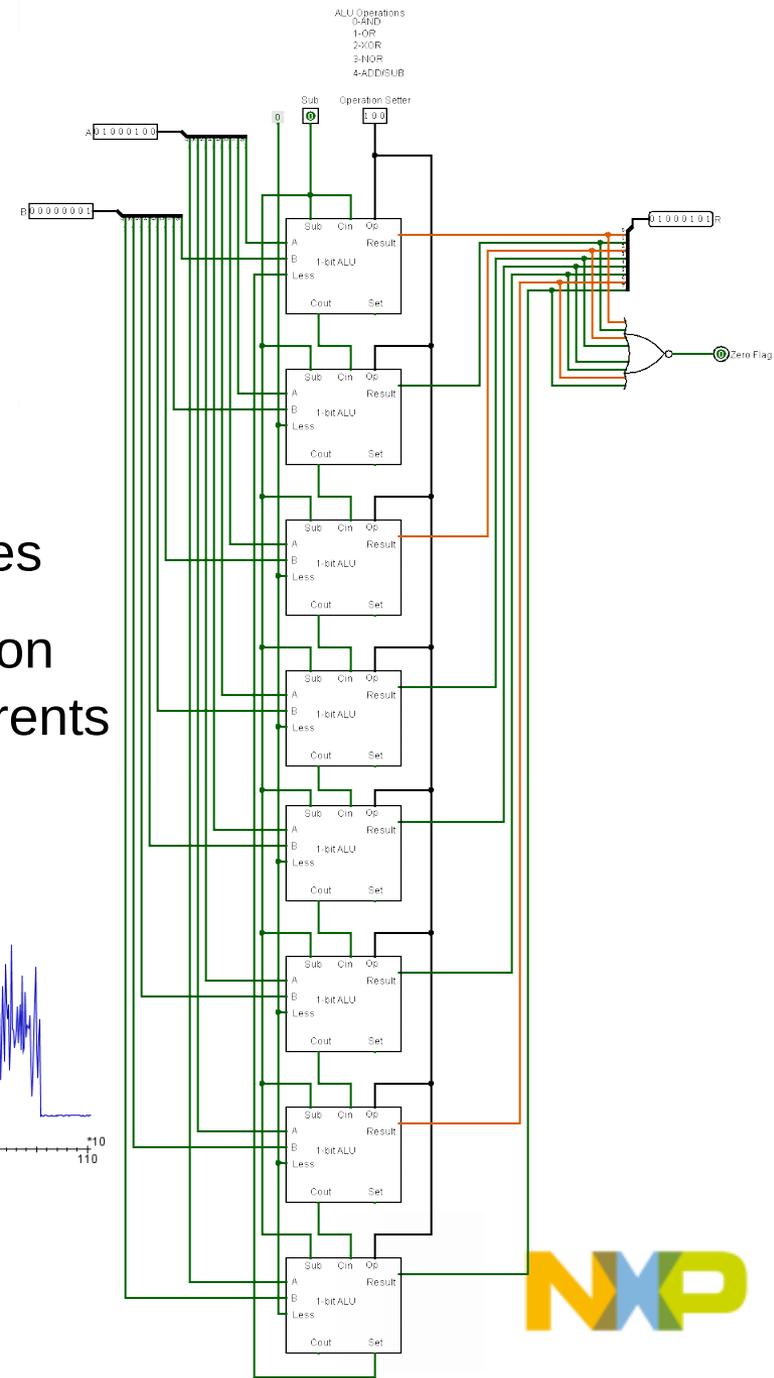
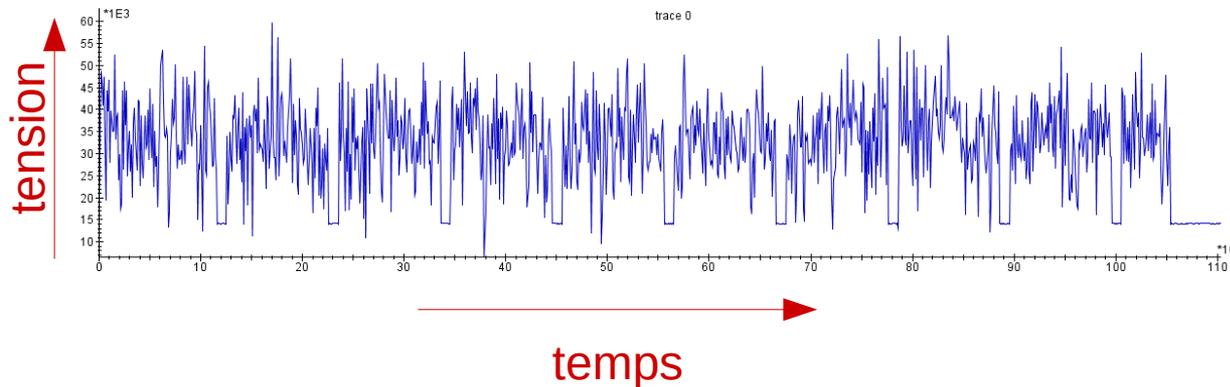
# Rappelez-vous...



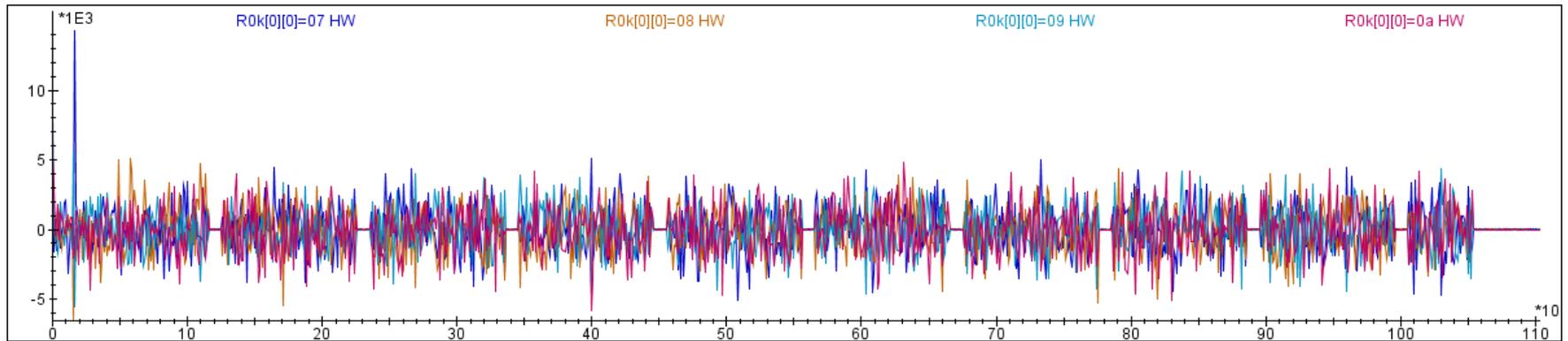
# Differential Power Analysis

par P. Kocher et al. (1998)

- Corrélations possibles entre: consommation électrique et poids de Hamming de valeurs internes
- Enregistre les traces de consommation correspondant à des messages différents



# Differential Power Analysis



Seuls prérequis:

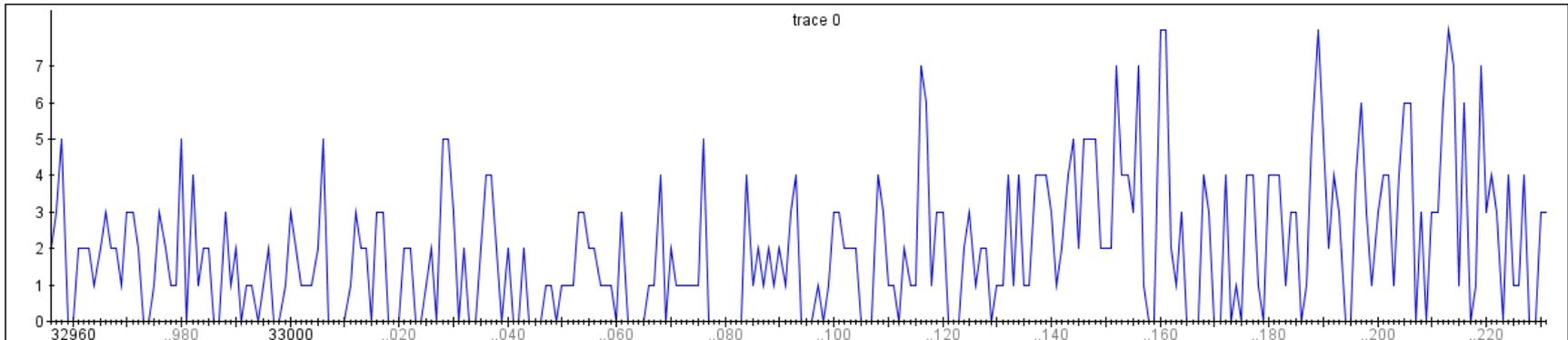
- Connaître l'entrée **ou** la sortie du chiffrement par blocs
- Pouvoir mesurer la consommation (ou les radiations EM)
- Présence de "fuites" (corrélations)

# Differential Computation Analysis

Traces d'exécution → “traces de consommation” ?

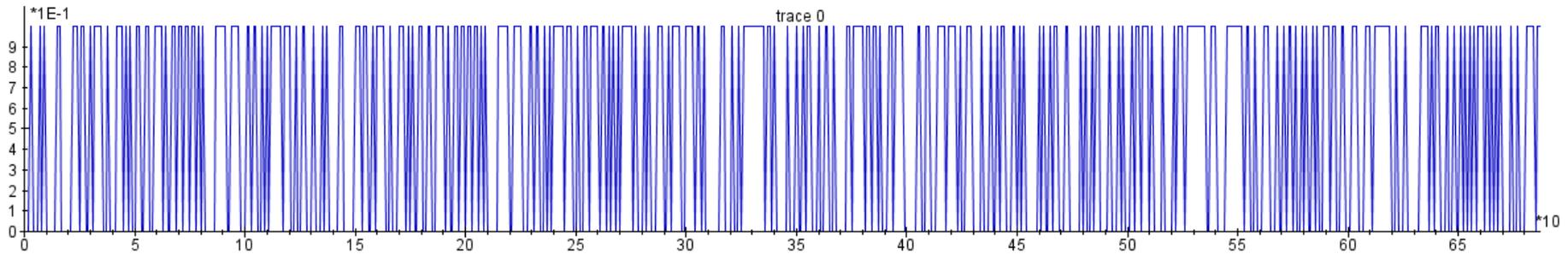
Adresses ou données des accès mémoire, écritures sur la pile,...

Exemple: octets lus → poids de Hamming ?



# Differential Computation Analysis

Octets → bits

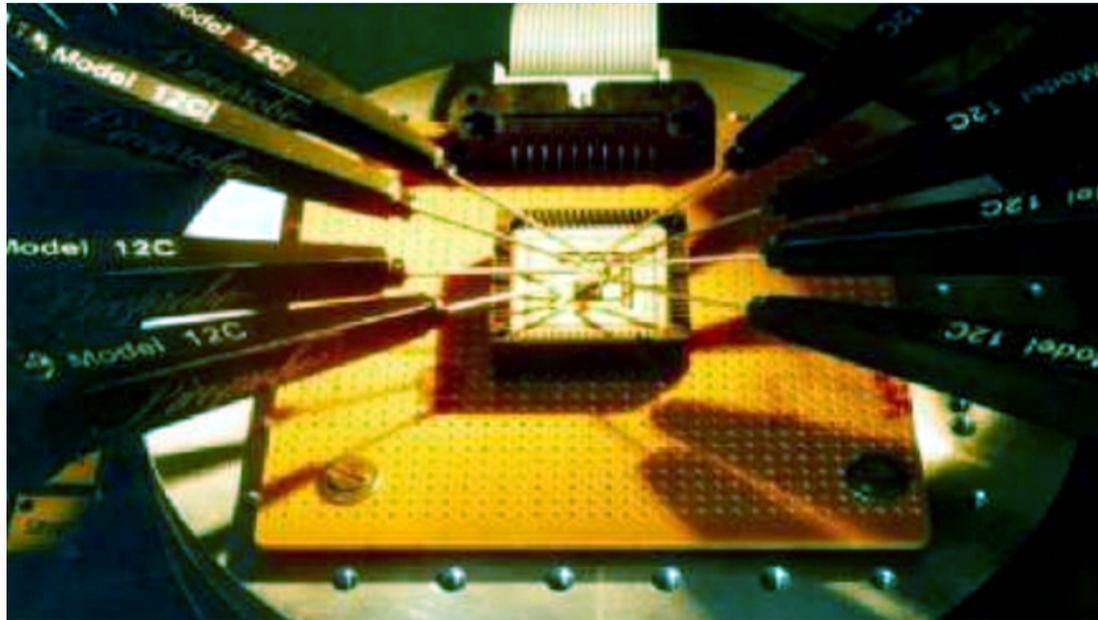
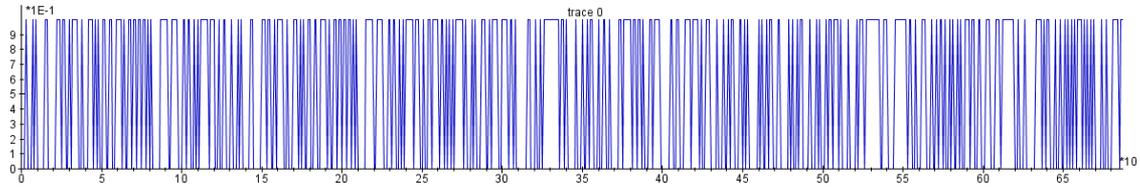


Yapluka → outils de DPA traditionnelle

Riscure Inspector, ChipWhisperer, Matlab ou... Daredevil!



# Differential Computation Analysis



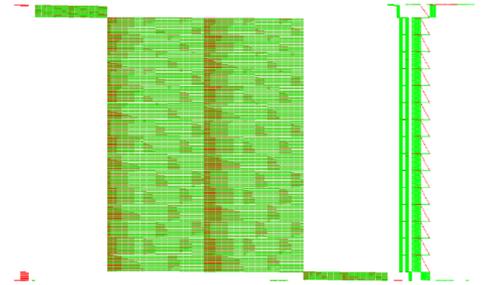
# Challenge Wyseur

par Brecht Wyseur, 2007

implémentation WB-DES : Chow “plus some personal improvements”

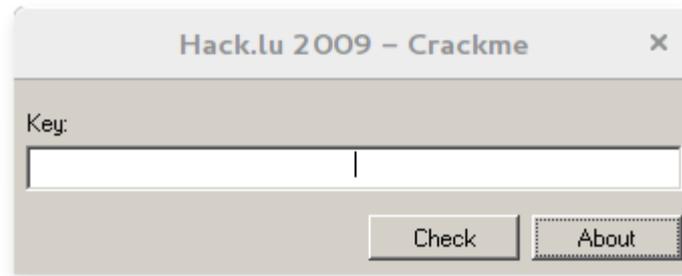
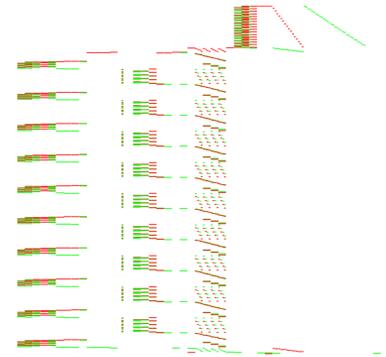
On télécharge...

1h et 65 traces plus tard, la clé est cassée !



# Challenge Hack.lu 2009

Windows *crackme* par Jean-Baptiste Bédrune  
Implémentation WB-AES : Chow



Paresse → Wine/Linux + xdotool (émulation clavier/souris)

16 traces

(challenge de CTF, pas d'encodages internes)

# Challenge SSTIC 2012

“white-box” en Python par Axel Tillequin  
Implémentation WB-DES dans un objet “marshall”

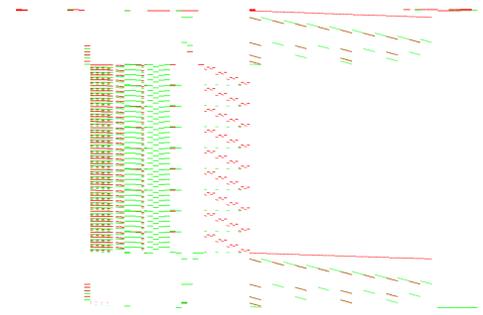
→ instrumentation d’une classe externe “Bits”

16 traces

pas d’encodages internes



# Karroumi



Dernière tentative académique de “réparer” Chow (2011)

*Dual Ciphers, i.e. isomorphic AES ciphers:*

$$\forall p, k : E_k(p) = f^{-1} \left( E'_{g(k)}(h(p)) \right)$$

김나리 생일 축하해

Notre propre challenge...

2000 traces, 500 traces après *tuning*

# Quelques implémentations “white-box” propriétaires

DES & AES

Cassées en 200 à 2500 traces



# Contre-mesures ?

Aléas dynamique?

- Pas de TRNG, juste le message d'entrée

Ajout de délais pseudo-aléatoires?

- Traces d'instructions → on réaligne

Bref, délicat...

*Une sécurité "parfaite" est illusoire, mais si le coût d'une attaque est supérieur au gain pour l'attaquant, vous avez réussi.*

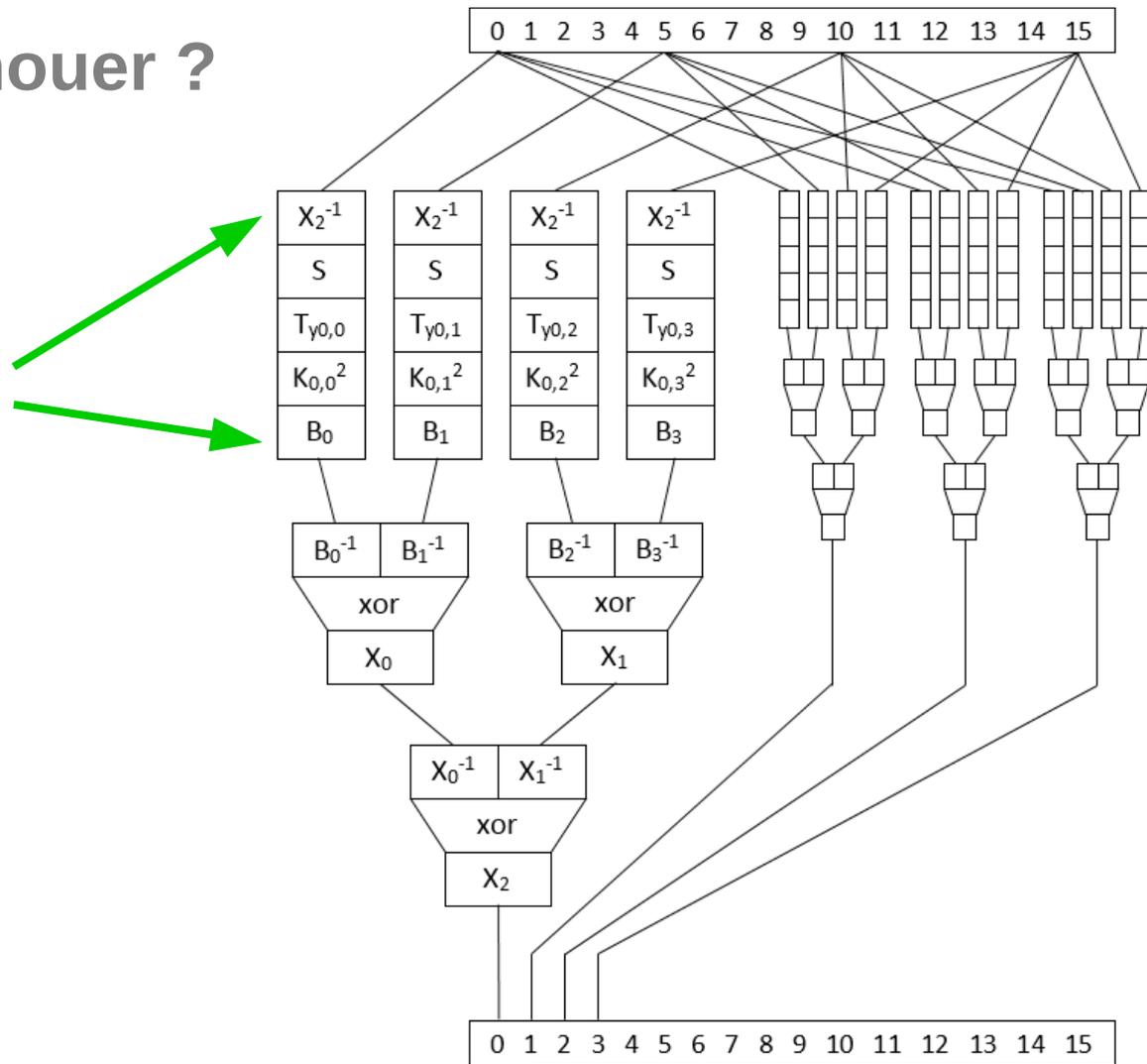
Oups, c'est la chute des prix...



# La DCA peut-elle échouer ?

Oui !

Encodages "larges" (8x8)  
masquant la non-linéarité  
de la SBOX



# La DCA peut-elle échouer ?

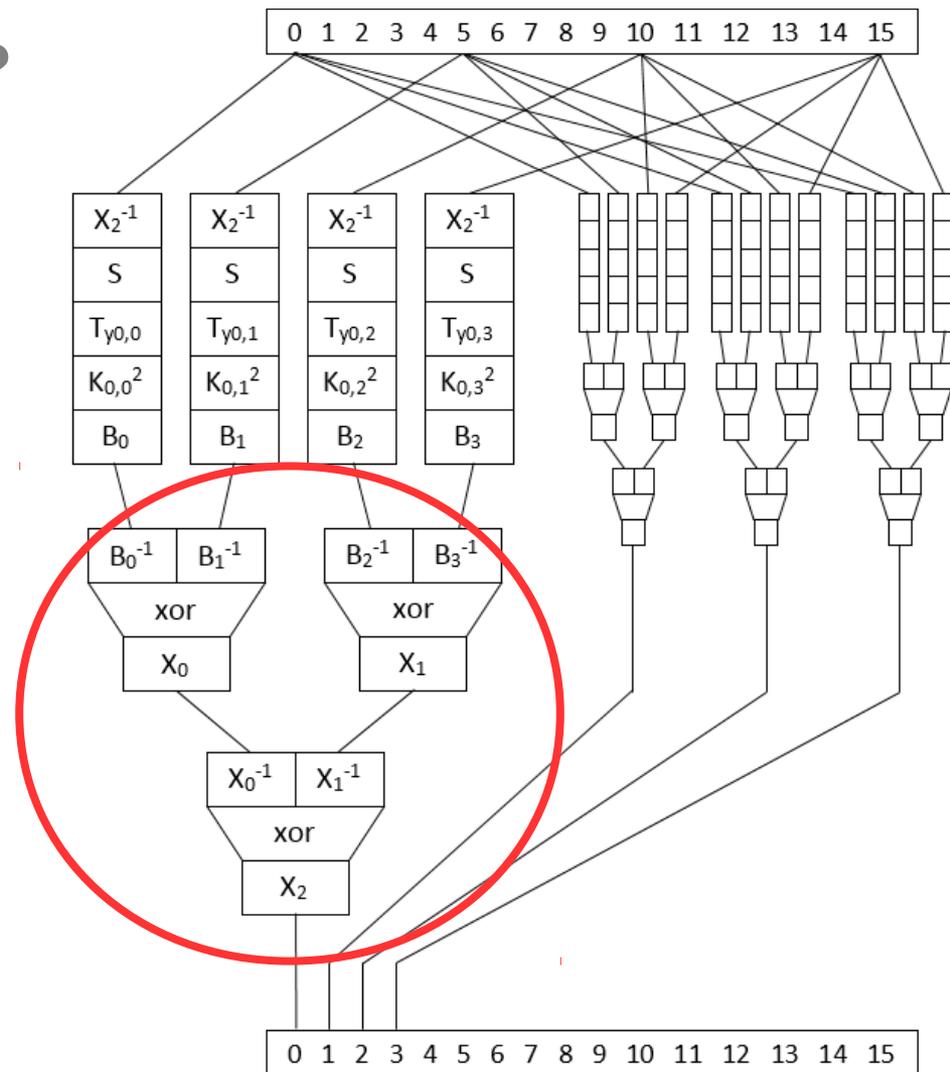
Oui !

Encodages “larges” (8x8)  
masquant la non-linéarité  
de la SBOX

→ grandes tables !

→ tendance à les réutiliser

→ autres types d'attaque



cf. write-ups

NoSuchCon 2013 et CHES 2015

[http://wiki.yobi.be/wiki/CHES2015\\_Writeup](http://wiki.yobi.be/wiki/CHES2015_Writeup)

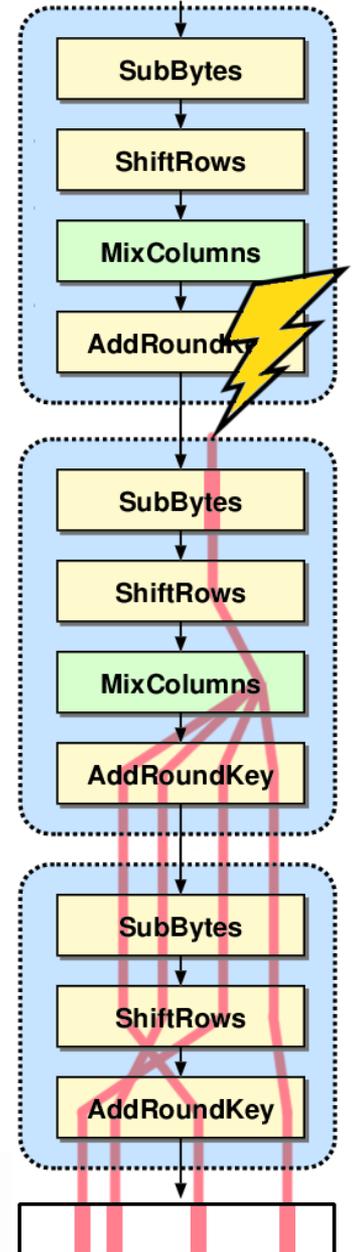
# Autres attaques “grey box” à portée de main : Higher order DPA, CPA, DFA,...



# La DFA, parlons-en...

## Differential Fault Analysis sur AES (Dusart et al. 2003)

- Prérequis: rejouer l'entrée, voir la sortie en clair
- Injection: source ; bin statique ; bin dynamique
- Injection à l'aveugle, voire à la bourrin
- 8 "bonnes" fautes suffisent (sur AES-128 enc ou dec)
- Temps d'analyse: qqs secondes





# Side-Channel Marvels

<https://github.com/SideChannelMarvels>



## Tracer

- TracerGrind
- TracerPIN
- TraceGraph



## Deadpool

- White-boxes
- Attacks automation



## Daredevil

- Side-channel analysis (CPA)



## JeanGrey

- Fault analysis (DFA)



# Side-Channel Marvels

<https://github.com/SideChannelMarvels>

Charles Hubain (Quarkslab)

Joppe Bos (NXP)

Michael Eder (TUM, Fraunhofer AISEC)

Paul Bottinelli (EPFL)

Philippe Teuwen (Quarkslab)

Van Huynh Le (U.Twente, NXP)

Wil Michiels (NXP, TU/e)

*Sans oublier...*



**Orka**

- images Docker

**MERCI !  
QUESTIONS ?**

**@haxelion  
@doegox**

**NXP**



**Quarkslab**

**DEMO**

