

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
- 웹 <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”



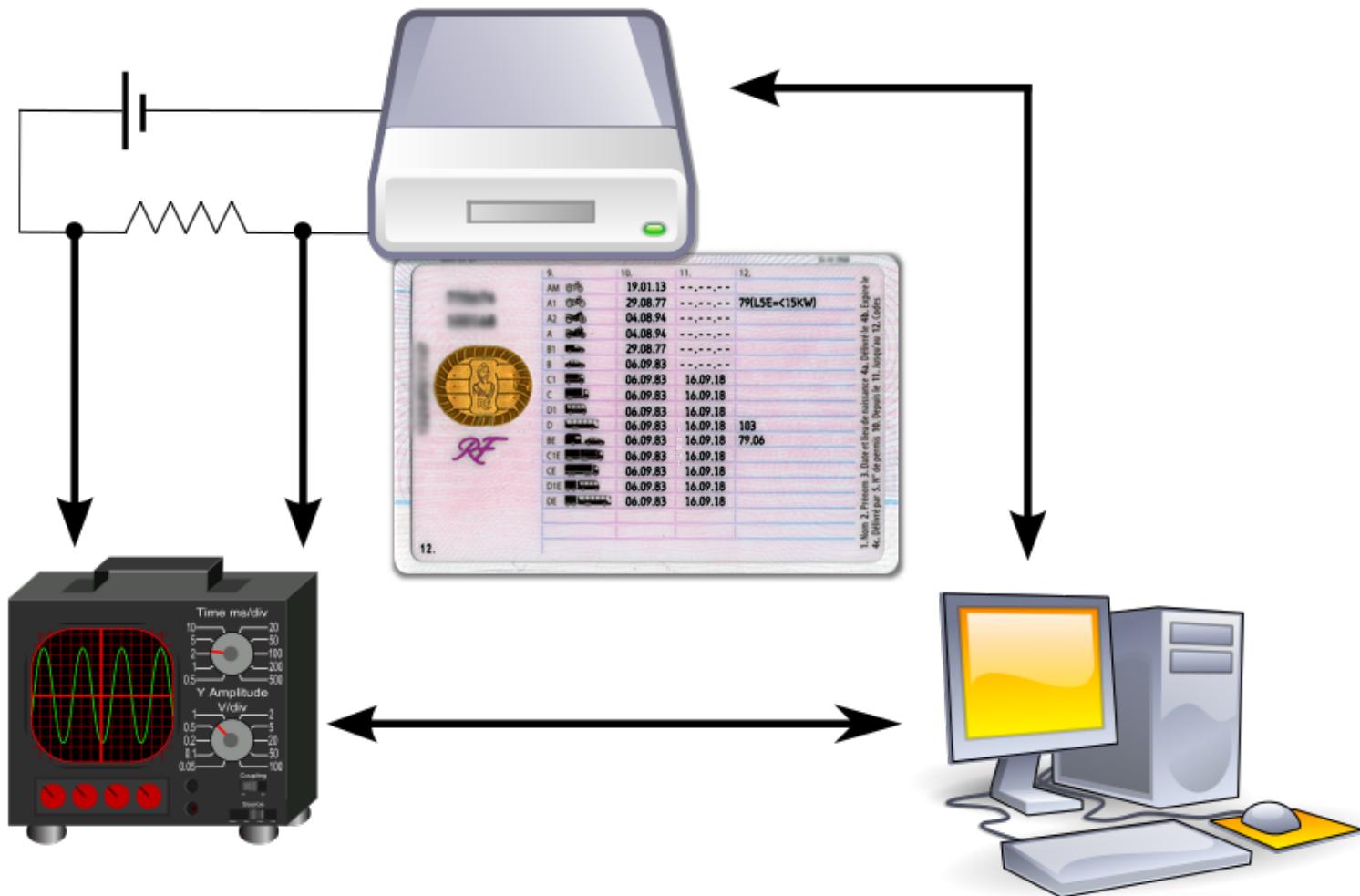
BREVE INTRO A LA CRYPTO WHITE-BOX



Rappel: modèle “Black box” => modèle “Grey box”

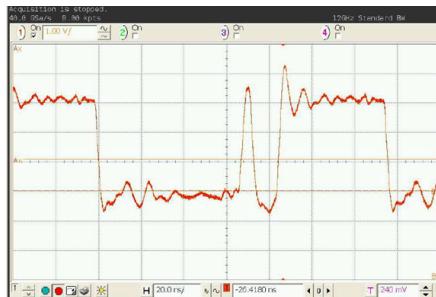


Modèle “Grey box”

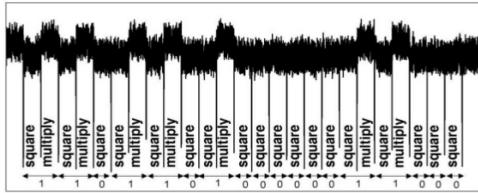


NXP

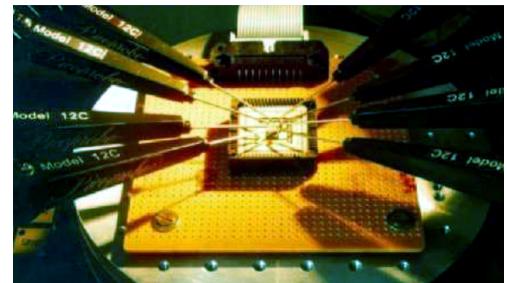
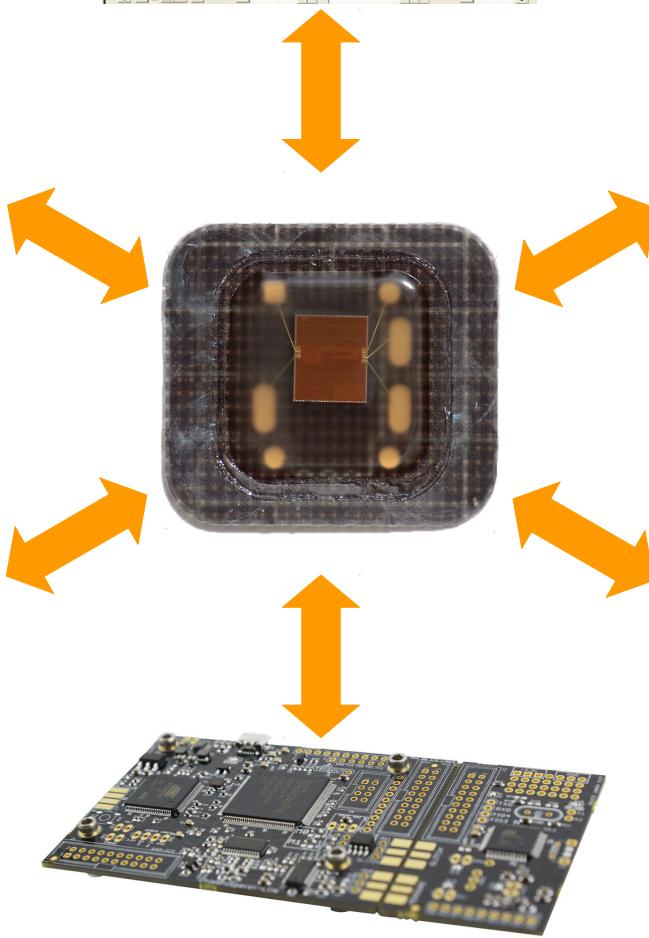
Modèle “Grey box”



Simple Power Analysis: Directly analyze (few) traces, for example RSA:



24



Modèle “White box”



Modèle “White box”



C CPU - main thread, module Software

00401204	\$	6A 60	PUSH 60
00401206		FF 10 40 00	CALL Software.00401F90
00401210		BF 94000000	MOV EDI, 94
00401215		8B C7	MOV ECX, EDI
0040121C		BF D40E0000	MOV EDX, PTR DS:[4872D0]
00401220		89 E9	MOV ECX, PTR SS:[EBP-18], ESP
00401225		89 F4	MOV EST, ESP
00401229		89 E6	MOV EDW, PTR DS:[ESI+1], EDI
0040122D		89 E8	MOV EDW, PTR DS:[ECX+1], EDI
00401231		FF 15 4C504000	CALL DWORD PTR DS:[4C504000]
00401234		89 E4 10	MOV ECX, DMOD PTR DS:[ESI+10]
0040123A		89 00 80724000	MOV DMOD PTR DS:[4872B8], ECX
0040123E		89 E8	MOV EDW, PTR DS:[ECX+1], EDW
0040123F		89 E8	MOV EDW, PTR DS:[4872C8], EDX
00401240		89 E6	MOV ECX, PTR DS:[ESI+1], EDX
00401241		89 E8	MOV EDW, PTR DS:[ECX+1], EDX
00401247		89 F9 FF7F0000	MOV ECX, PTR DS:[4872C8], ESI
00401248		89 F9 BC724000	MOV DMOD PTR DS:[4872B8], ESI
00401250		89 F9 02	CMPS ECX, ECX
00401254		89 E8	MOV ECX, PTR DS:[Software.00401264]
00401258		89 E8	MOV ECX, PTR DS:[4872B8], ESI
0040125E		89 E8	MOV ECX, PTR DS:[4872B8], ESI

Registers (CPU)

ECX	77411142	kernel32.BaseInitInithunk
EDX	00000000	
EBP	00401204	Software.<ModuleEntryPoint>
ESP	0012FF90	
EBP	0012FF94	
ESI	00000000	
EDI	00000000	

Stack

C	0023 32bit 0xFFFFFFFF
P	1 0018 32bit 0xFFFFFFFF
R	0 SS 0023 32bit 0xFFFFFFFF
S	0 FS 003B 32bit 0xFFFFFFFF
T	0 GS 0000 NULL
D	0 0

LastError ERROR_INSUFFICIENT_BUFFER (000000246) (N0,NB,E,BE,MS,PE,GE,LE)

Registers (CPU) (continued)

ST0	empty 0.0
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
STS	empty 17.000000000000000000000000000000

IDB - /Users/ig/idasrc/current/ida/tests/input/pc_linux_appcall.elf

IDA View-A, Pseudocode-A

```

int __stdcall CALLUI(ui_notification_t..., int a1, int a2, int a3, int a4, int a5)
{
    char v5; // _a126
    int result; // _a280
    int v6; // _a300 [bp-28Chh]@13
    char *v7; // _a304 [bp-28Chh]@10
    char v8; // _a308 [bp-28Chh]@8
    char v9; // _a312 [bp-28Chh]@6
    int v10; // _a316 [bp-28Chh]@6
    int v11; // _a318 [bp-28Chh]@4
    int v12; // _a31C [bp-28Chh]@4
    int v13; // _a31E [bp-28Chh]@2
    int v14; // _a320 [bp-28Chh]@1
    v14 = 0;
    if (v14 == 0)
        v14 = _getpid();
    switch (v14)
    {
        case 23:
            v15 = _snprintf((char *)v12, 0x2000u, a3, a5);
            *(C_DWORD *)v11 = printf("%s", v15);
            break;
        case 28:
            v11 = a4;
            if (v11 == 597061684 && (v11 = a5, a4 == -1427216450))
                if ((v5 = _snprintf((char *)v11, (int)v14, 5));
                    *v5 != '\0')
                    break;
            case 21:
                for (*haystack = a4; haystack = (int)(v7 + 8); )
                    while (!strcmp((const char *)haystack, "ICON")) {
                        if (strcmp((const char *)haystack, "AUTOWINE"))
                            continue;
                        if (*v7 == '\0') {
                            haystack = -(int)v7 + 10;
                            if (*v7)
                                haystack = -(int)v7 + 8;
                            if (!strcmp((const char *)haystack, "HELPn"))
                                if (*v7 == strret((const char *)haystack, "ENDHELPn")))
                                    continue;
                            break;
                        }
                    }
            case 13:
                if (*v5 == '\0') {
                    v5 = strret((const char *)haystack, "HELPn");
                    if (*v7)
                        continue;
                    break;
                }
            case 15:
                if (*v5 == '\0') {
                    v5 = strret((const char *)haystack, "AUTOWINE");
                    if (*v7)
                        continue;
                    break;
                }
    }
    CALLUI:15
}

```

Hex View-A

Graph overview

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>

```

5,25" Mini-Disk
DS/DD
25/20
soft
48 TPI



NXP

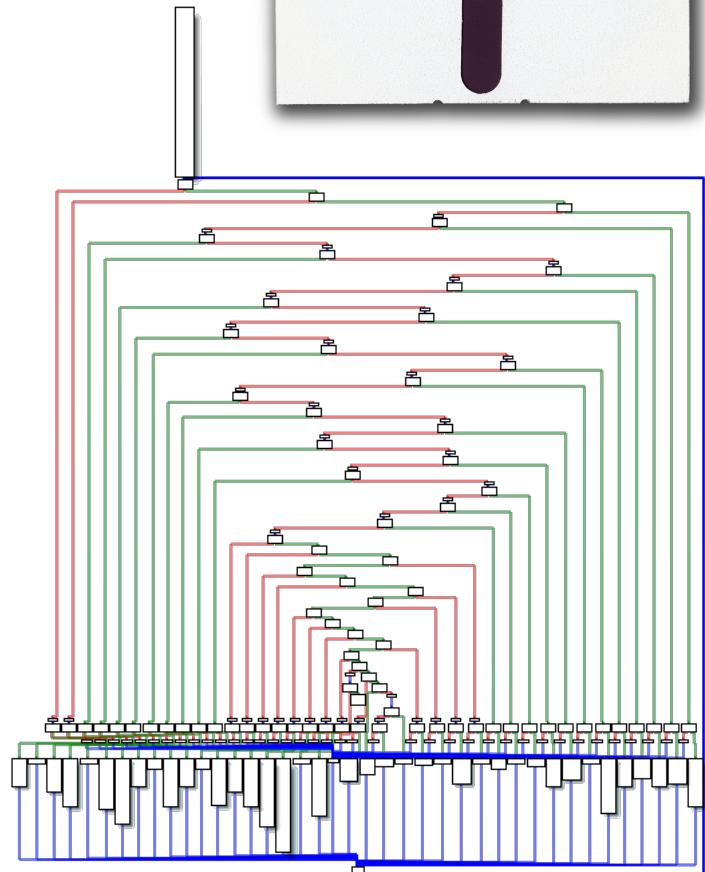
Ligne de défense = implémentation

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



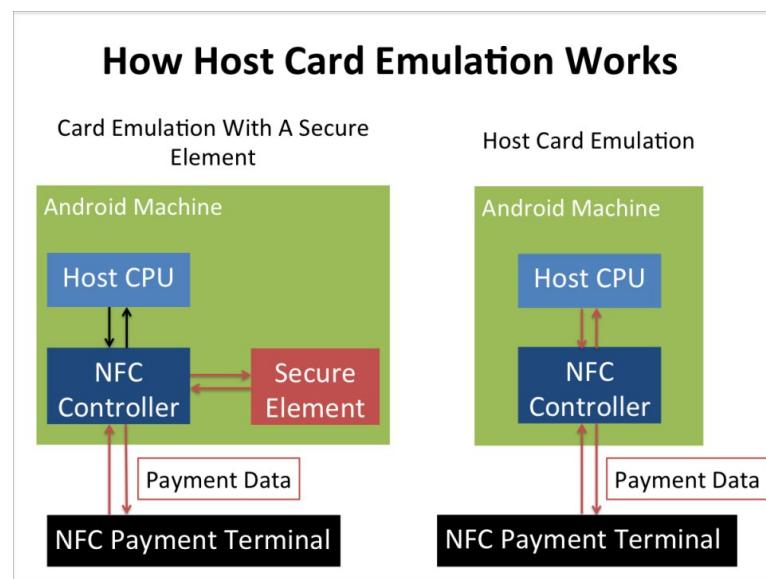
Cryptographie dans le modèle “White box”

Pourquoi faire?

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



Source: "l'industrie du film"



Source: Business Insider

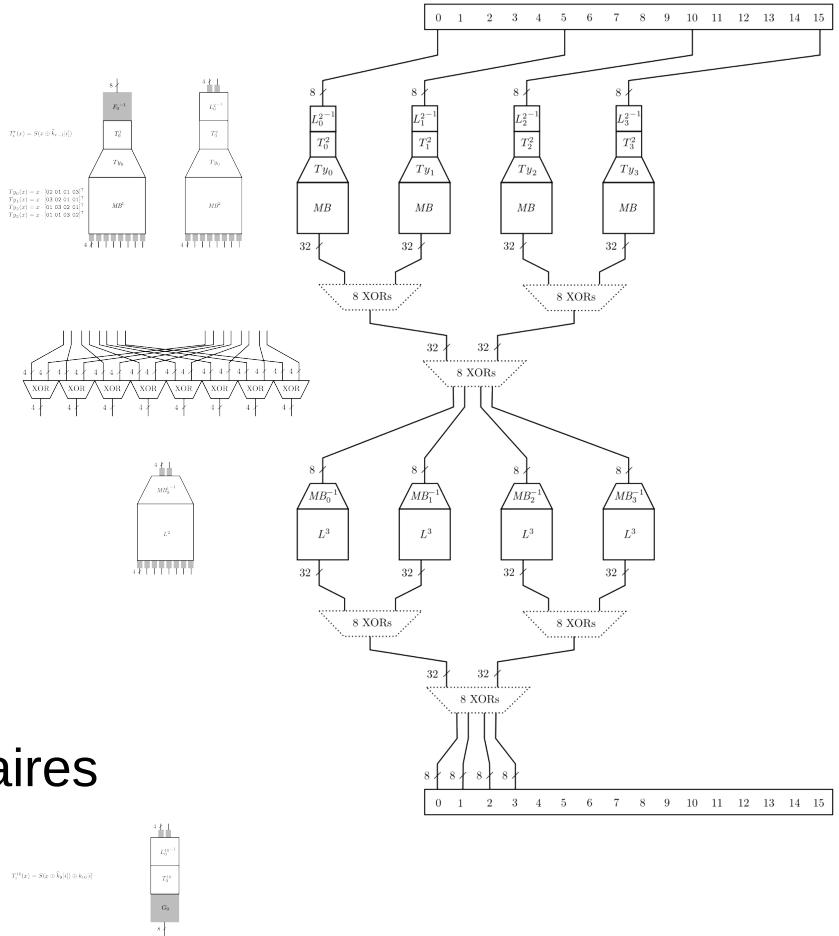


Cryptographie “White box”

Obscurcissement de code: inadéquat

→ Chow et al. (2002)

- Implémentation “idéale”: 4.94×10^{27} 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

$$\Pi_2^{(r,j)} \circ AES^{(r,\pi^{(r)}(j))} \circ \Pi_1^{(r,j)} = \text{MC}^{(r,j)} \circ (S, S, S, S) \circ \oplus_{[\bar{k}_i^{(r,j)}]_{0 \leq i \leq 3}} ,$$

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

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

$$\left\{ \begin{array}{ll} \text{SR} \circ \bigoplus_{K'_{10}} \circ \{S_{10,i}\}_{i=0,\dots,15} \circ \bigoplus_{K_9} \circ A_9'^{-1} & \text{for } R_{10} , \\ \text{MC} \circ \text{SR} \circ A''_r \circ \{S_{r,i}\}_{i=0,\dots,15} \circ \bigoplus_{K_{r-1}} \circ A_{r-1}'^{-1} & \text{for } R_r |_{2 \leq r \leq 9} , \\ \text{MC} \circ \text{SR} \circ A''_1 \circ \{S_{1,i}\}_{i=0,\dots,15} \circ \bigoplus_{K_0} & \text{for } R_1 , \end{array} \right. \quad (5)$$

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:

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

ψ_r and ϕ_r are different non-linear bijective encodings on 4-bit blocks, and δ_r

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

$$\begin{aligned} \mu_r(x_0x_1\dots x_{47}, y_0\dots y_{47}) &= y_0\dots y_5x_{\mu_r^{-1}(0)}x_{\mu_r^{-1}(1)}y_6\dots y_{11}x_{\mu_r^{-1}(2)}x_{\mu_r^{-1}(3)}\dots y_{42}\dots y_{47}x_{\mu_r^{-1}(22)}x_{\mu_r^{-1}(23)}\dots x_{\mu_r^{-1}(47)} \\ \gamma_r(z_0z_1\dots z_{95}) &= z_{\gamma_r^{-1}(0)}\dots z_{(\gamma_r^{-1}(0)+5)}z_6z_7\dots z_{\gamma_r^{-1}(11)}\dots z_{(\gamma_r^{-1}(11)+5)}z_{94}z_{95} \end{aligned}$$

The obfuscated t-box is

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

Hence the transformed function is:

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

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

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

Extraits de:

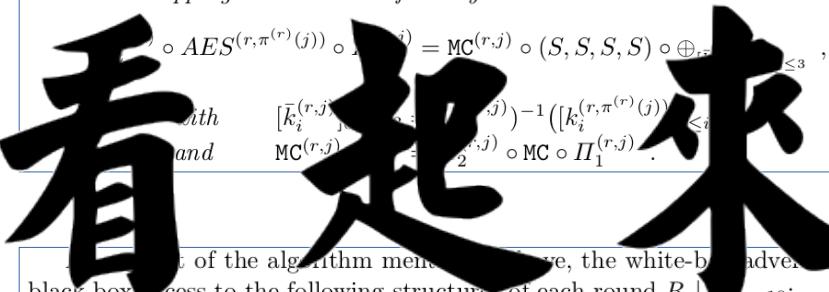
- “Two Attacks on a White-Box AES”
- “Cryptanalysis of a Perturbated White-Box AES Implementation”
- “Attacking an obfuscated cipher by injecting faults”

Attaques “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



As part 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:

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

ψ_r and ϕ_r are different non-linear bijective encodings on 4-bit blocks, and δ_r

$$\begin{aligned} \mu_r(x_0x_1x_2x_3y_0y_1y_2y_3) &= y_0 \dots y_5 x_{\mu_r^{-1}(0)} x_{\mu_r^{-1}(1)} y_{\mu_r^{-1}(2)} x_{\mu_r^{-1}(3)} \dots y_{42} \dots y_{47} \\ \gamma_r(z_0z_1z_2z_3) &= z_{\gamma_r^{-1}(0)} \dots z_{(\gamma_r^{-1}(0)+5)} z_{\gamma_r^{-1}(1)} \dots z_{(\gamma_r^{-1}(11)+5)} z_{94} z_{95} \end{aligned}$$

The obfuscation is

$$T_r^k(x) = (\beta_r^k \circ \tau_r^k \circ \lambda)(x).$$

Hence the transformed function is:

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

with

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

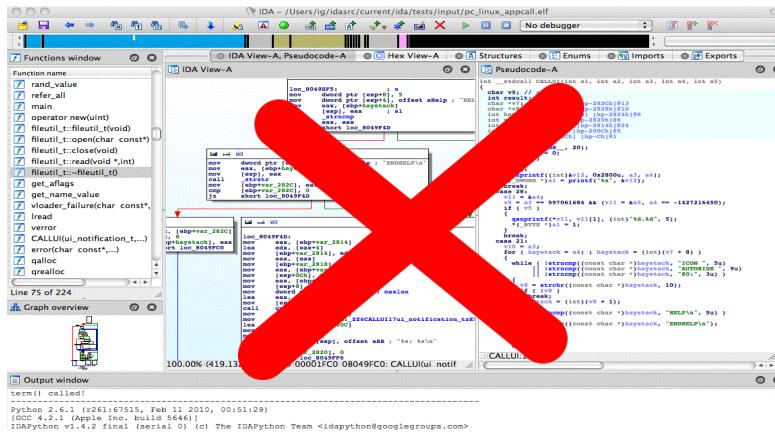
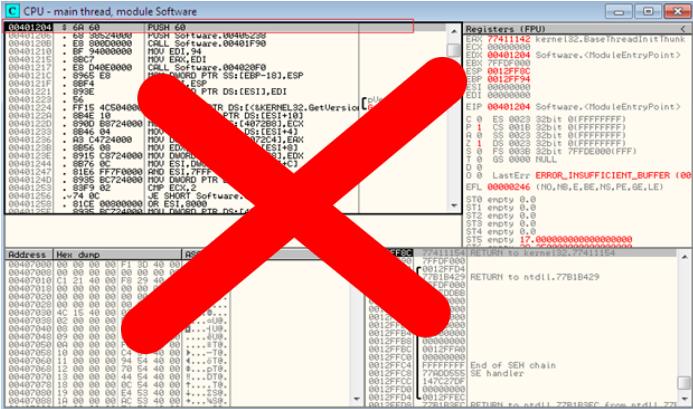
By setting

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

the resulting encryption operation is

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

Notre but:



NXP

TRACES D'EXECUTION

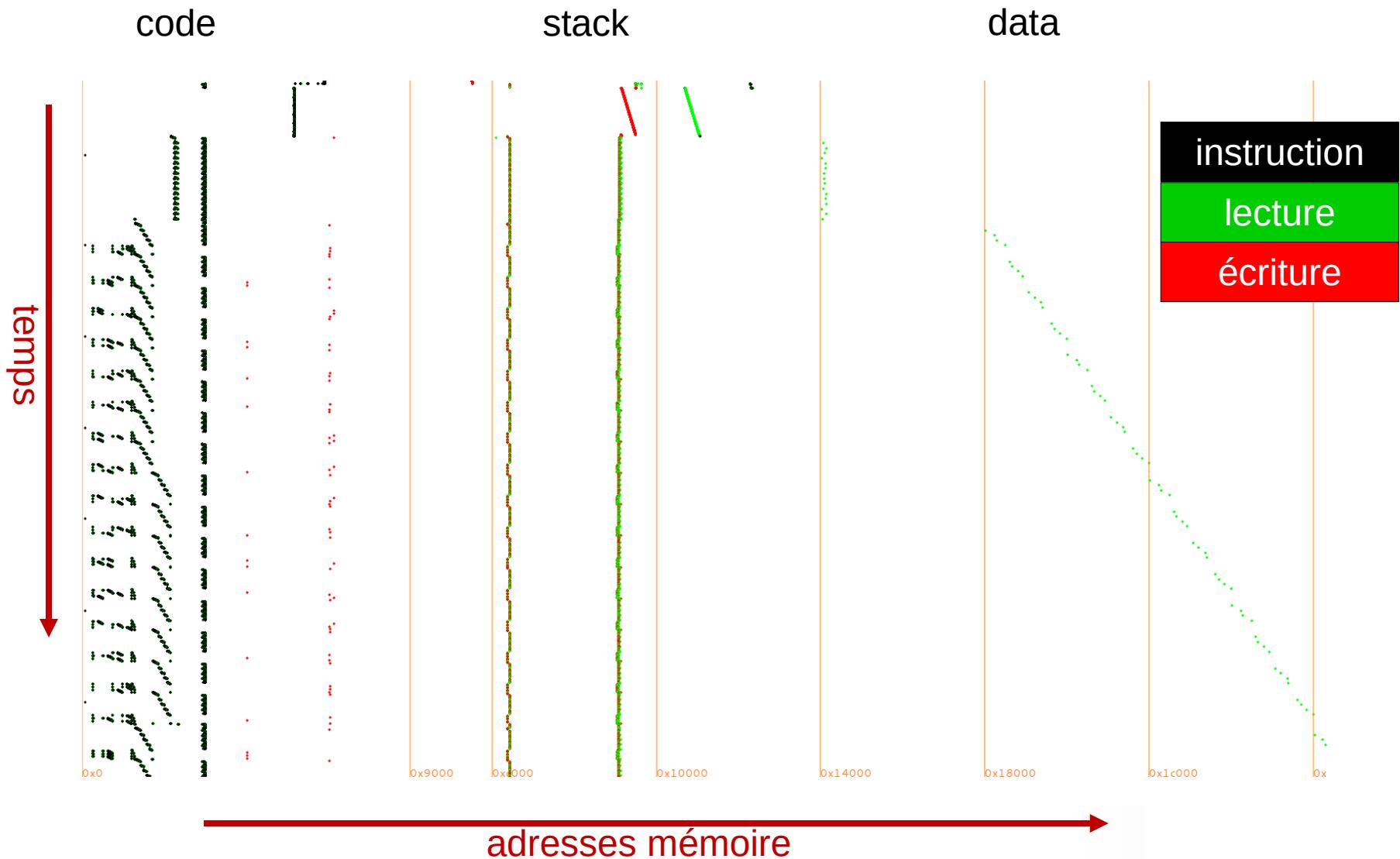
Tracer un exécutable

Enregistrer toutes les instructions et les accès mémoire

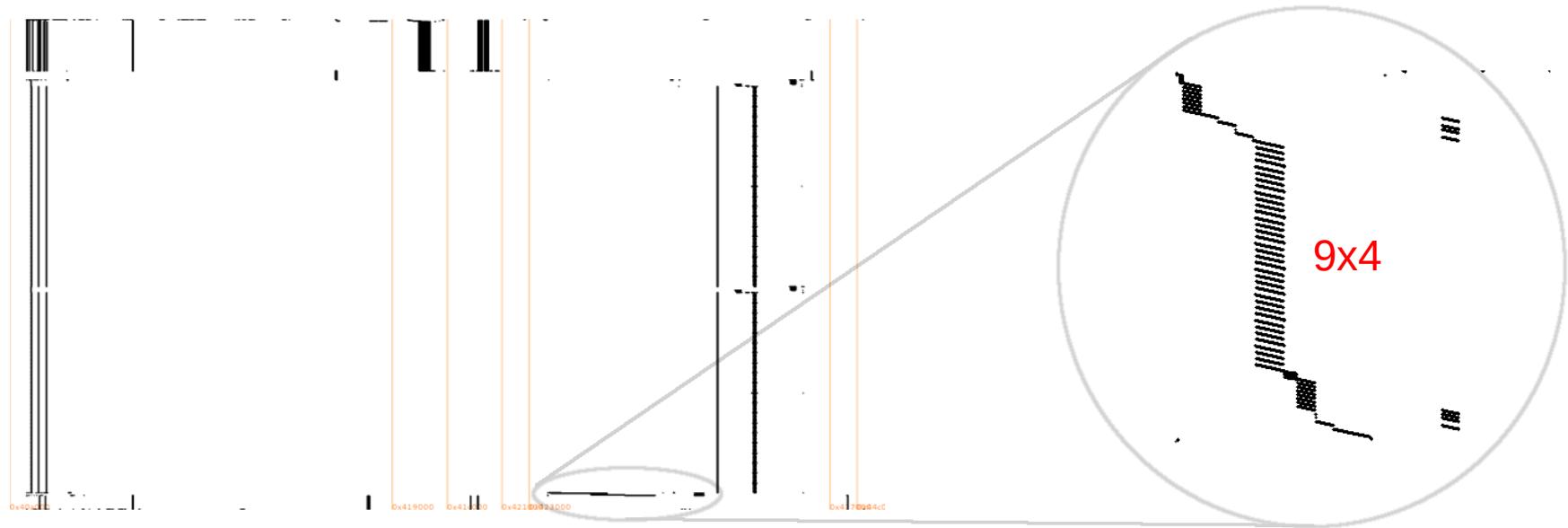
Exemples:

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

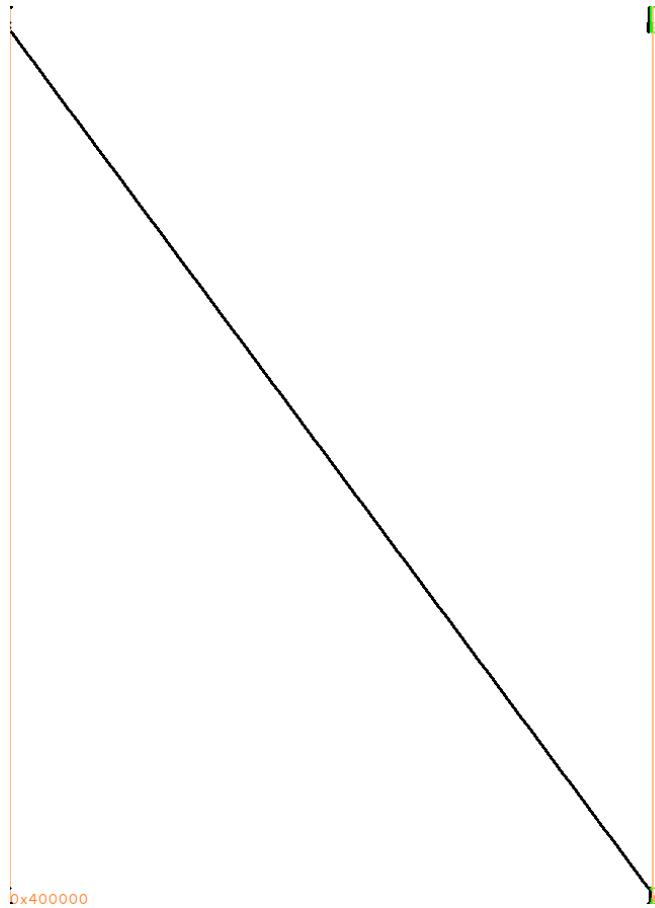
Même convention que pTra de Quarkslab



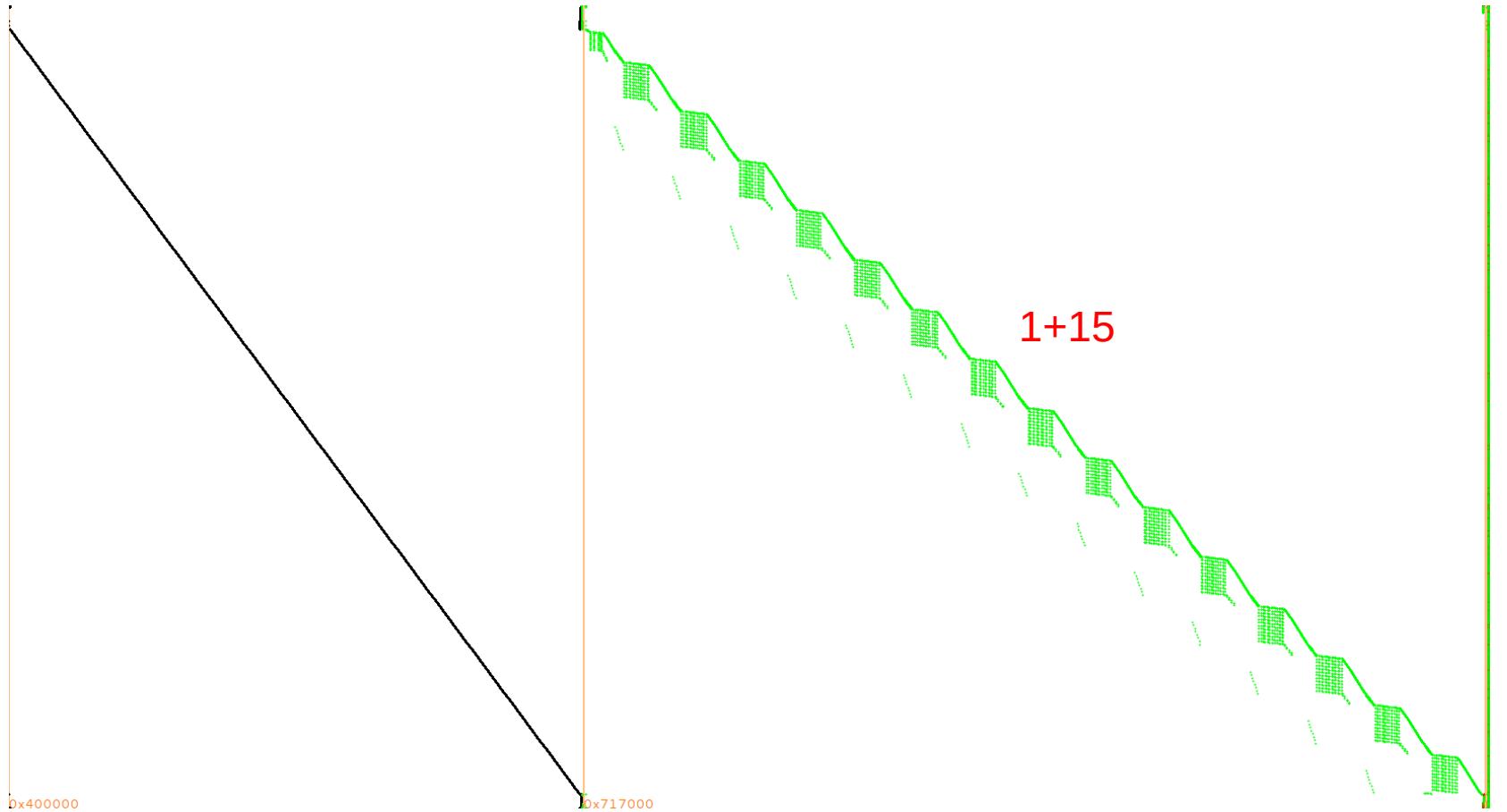
Identification visuelle de crypto: par le code



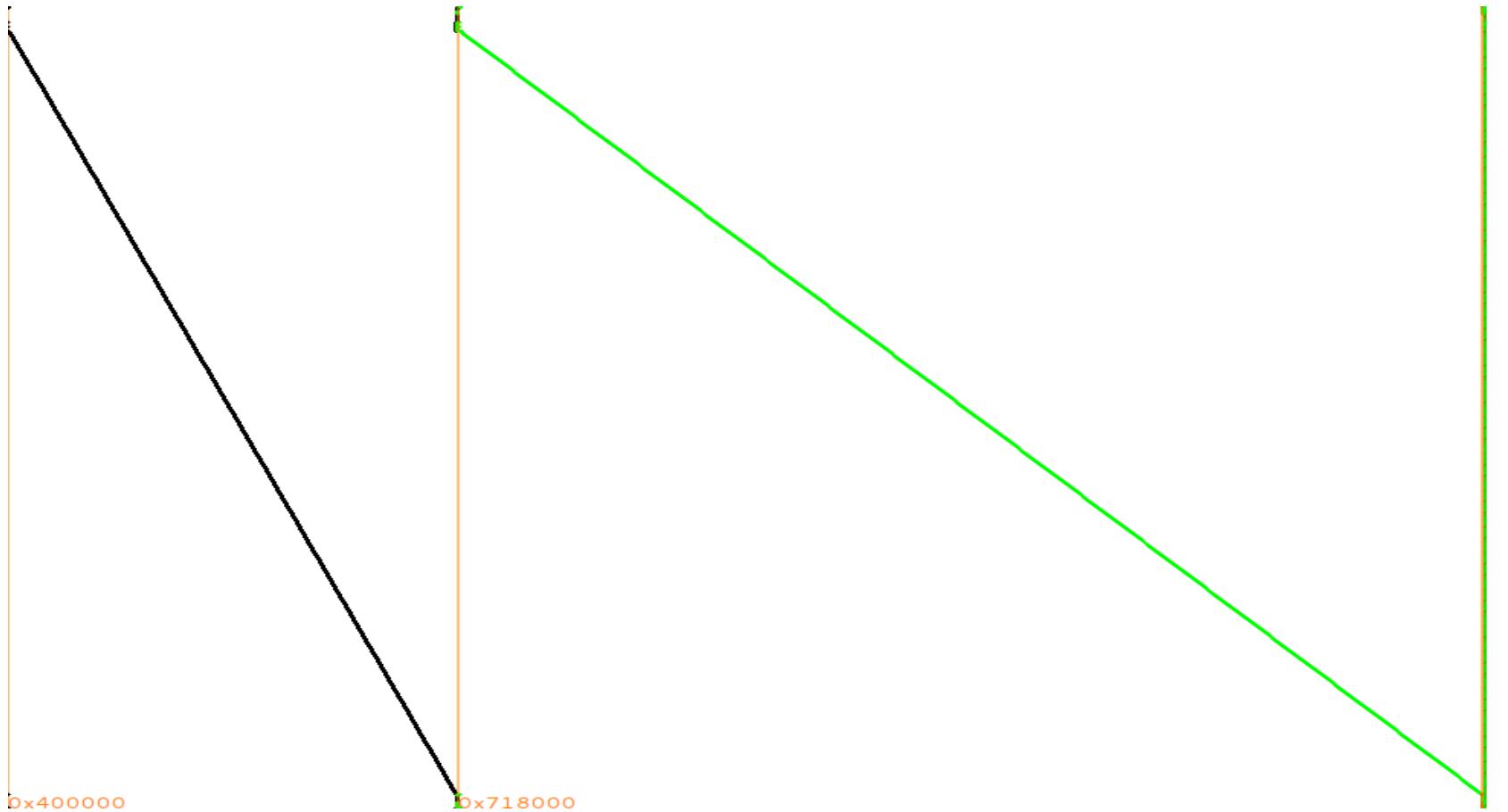
Identification visuelle de crypto: par le code ?



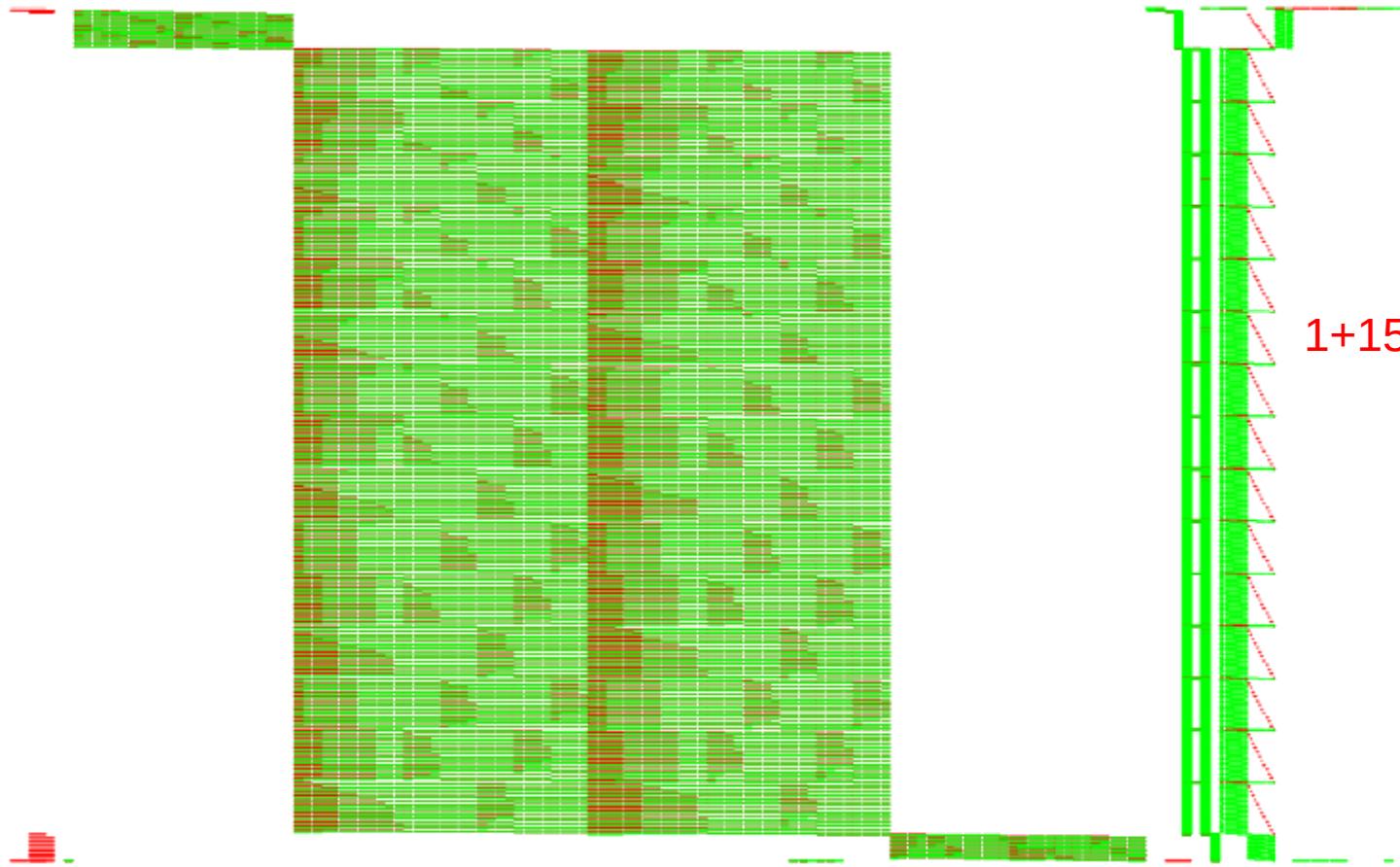
Identification visuelle de crypto: par les données !



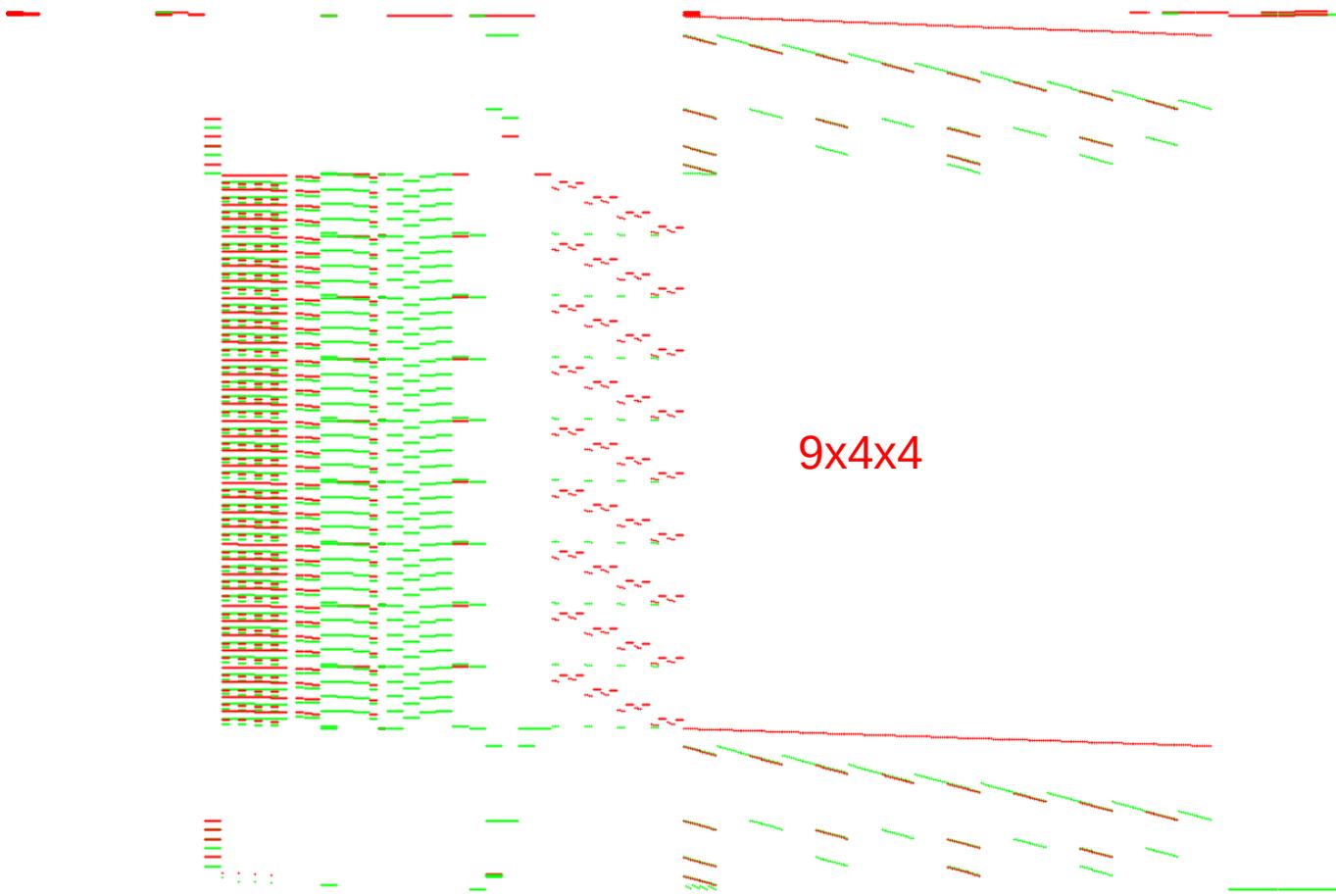
Identification visuelle de crypto: par les données ?



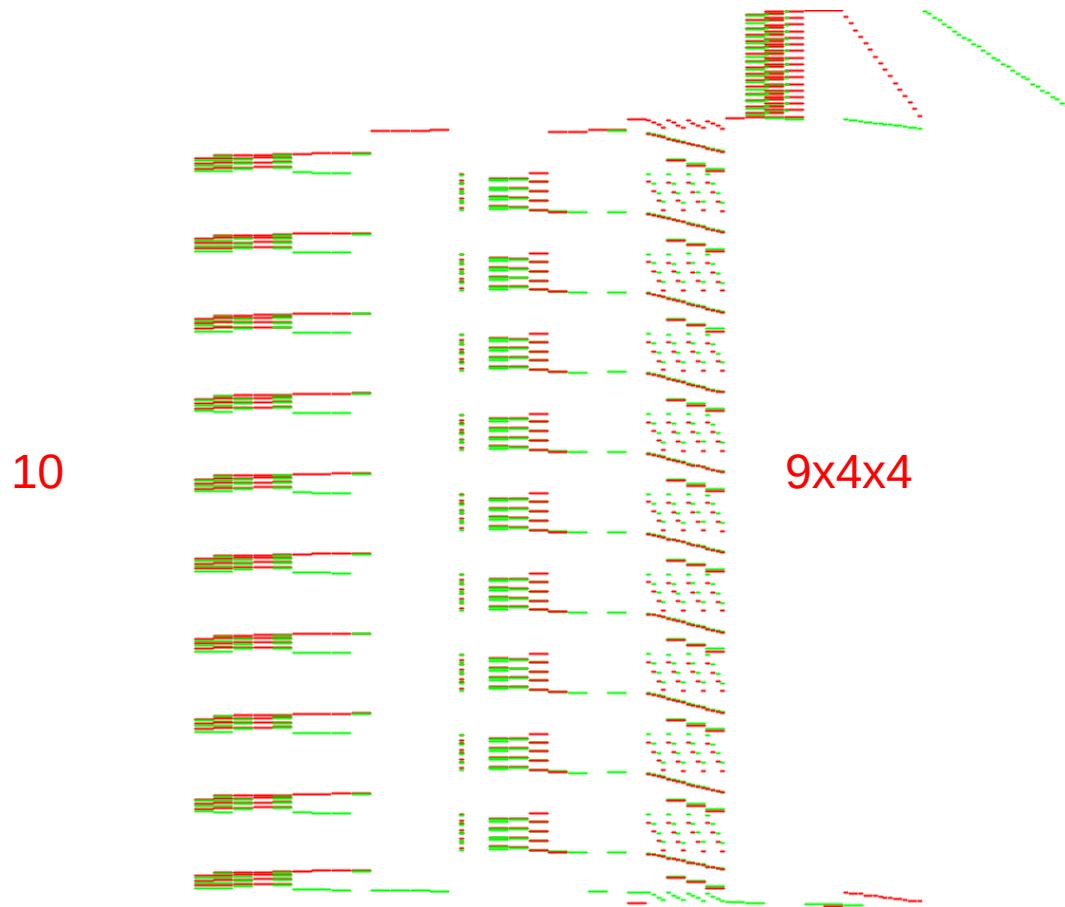
Identification visuelle de crypto: par la pile !



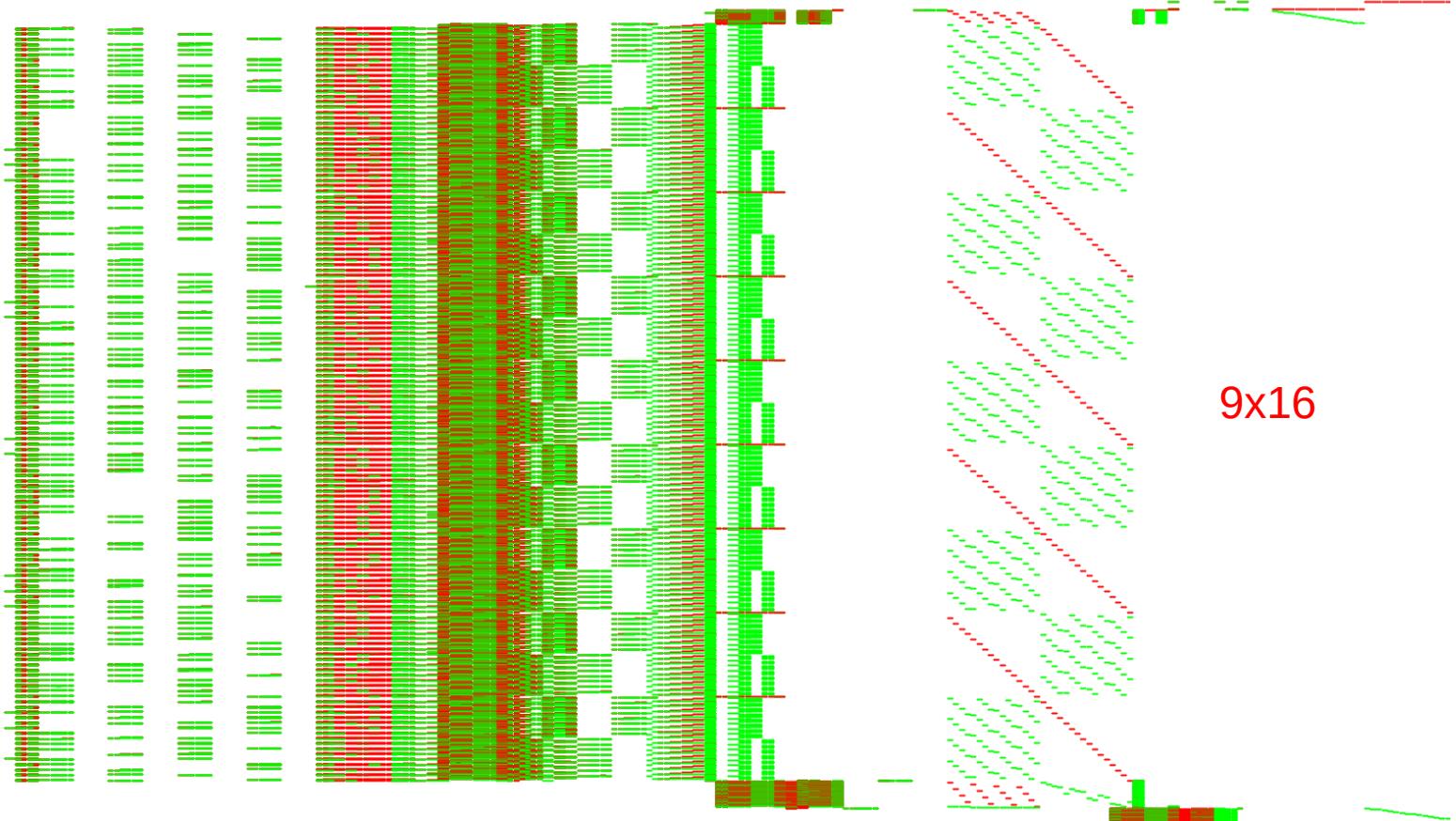
Identification visuelle de crypto: par la pile !



Identification visuelle de crypto: par la pile !



Identification visuelle de crypto: par la pile !





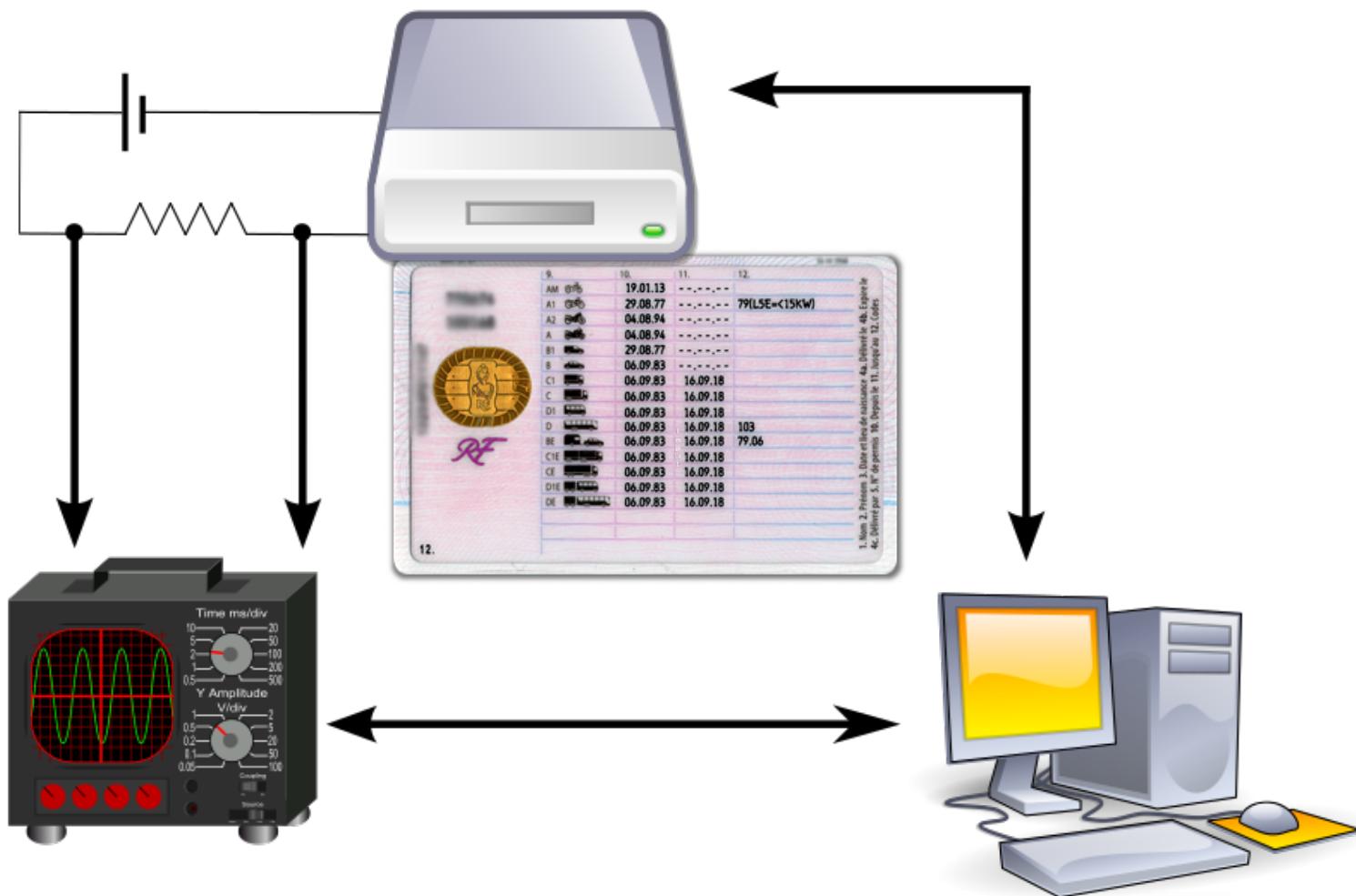
 You and 82 others don't give a fuck.

Et ma clé dans tout ça ?

DIFFERENTIAL COMPUTATION ANALYSIS



Rappelez-vous...

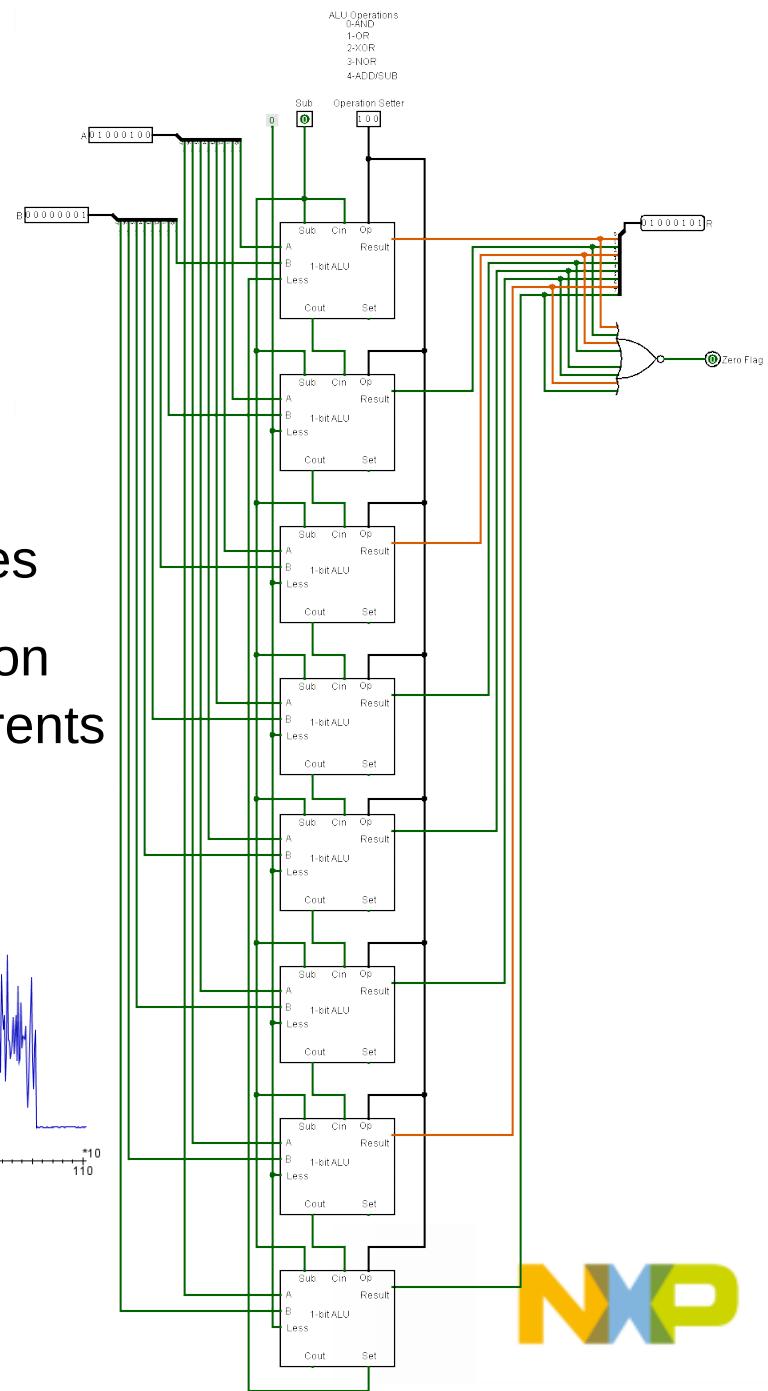
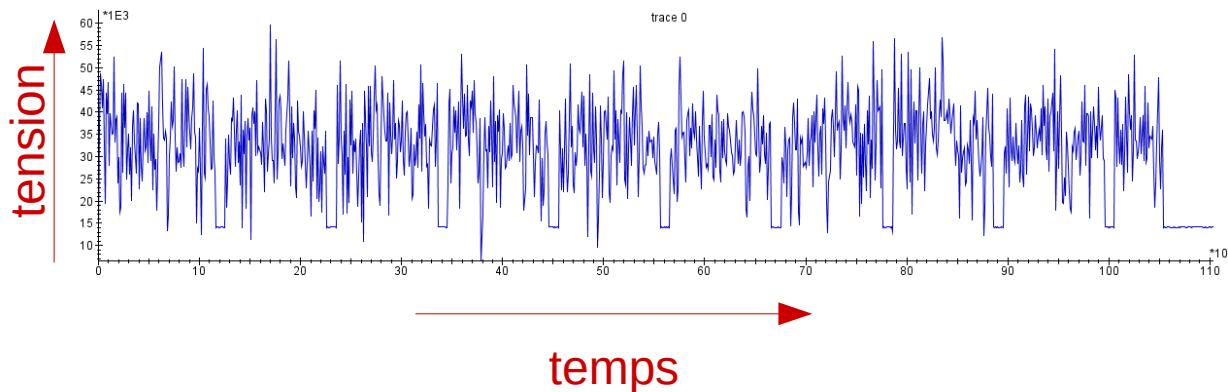


NXP

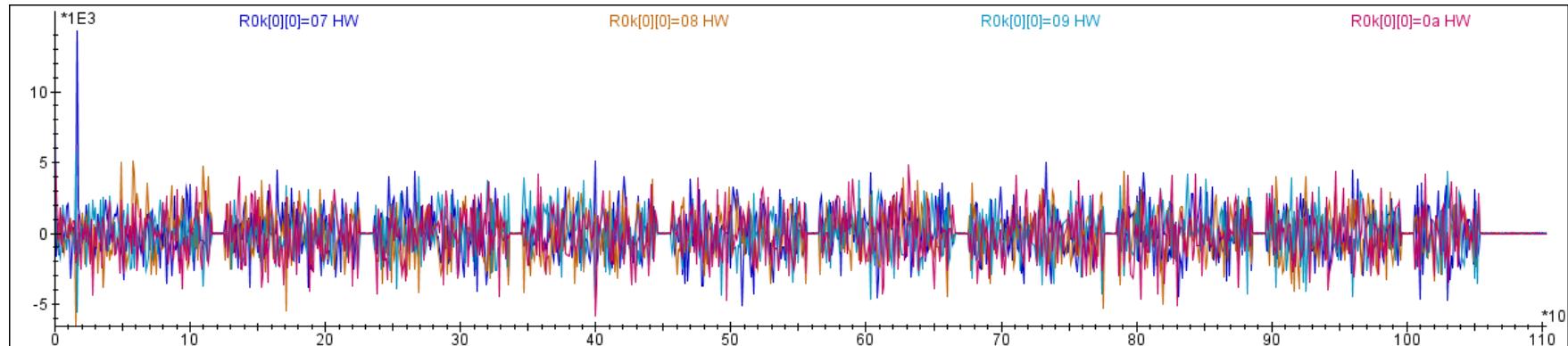
Differential Power Analysis

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:

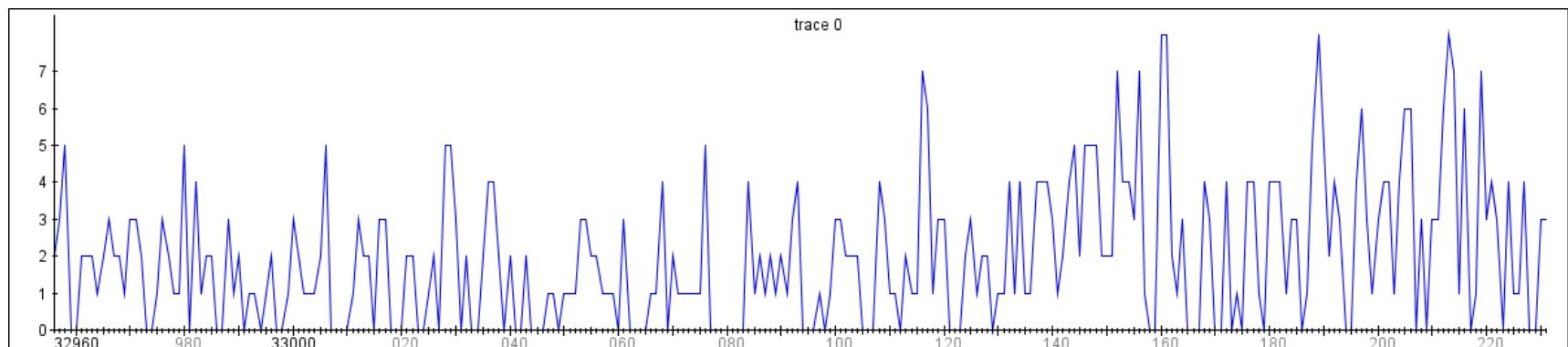
- Connaître l'entrée **ou** la sortie du chiffrement par blocs
- Pouvoir mesurer la consommation (ou les radiations EM)
- Présence de “fuites” (corrélations)

Differential Computation Analysis

Traces d'exécution → “traces de consommation” ?

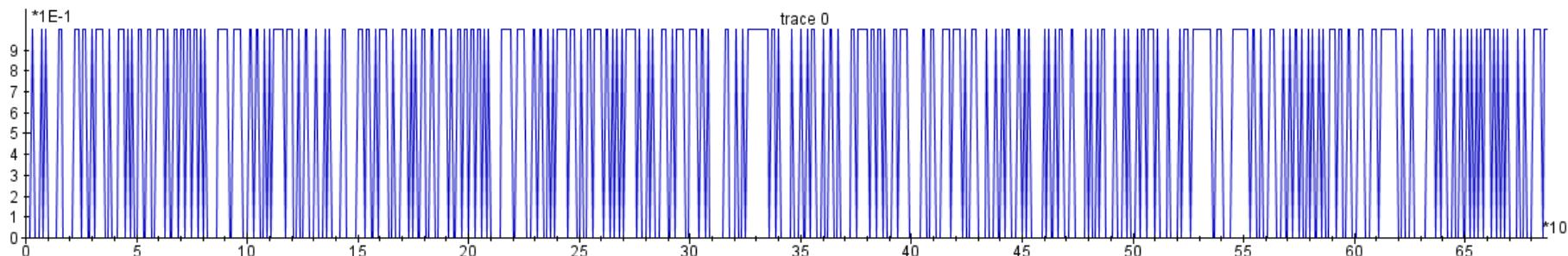
Adresses ou données des accès mémoire, écritures sur la pile,...

Exemple: octets lus → poids de Hamming ?



Differential Computation Analysis

Octets → bits

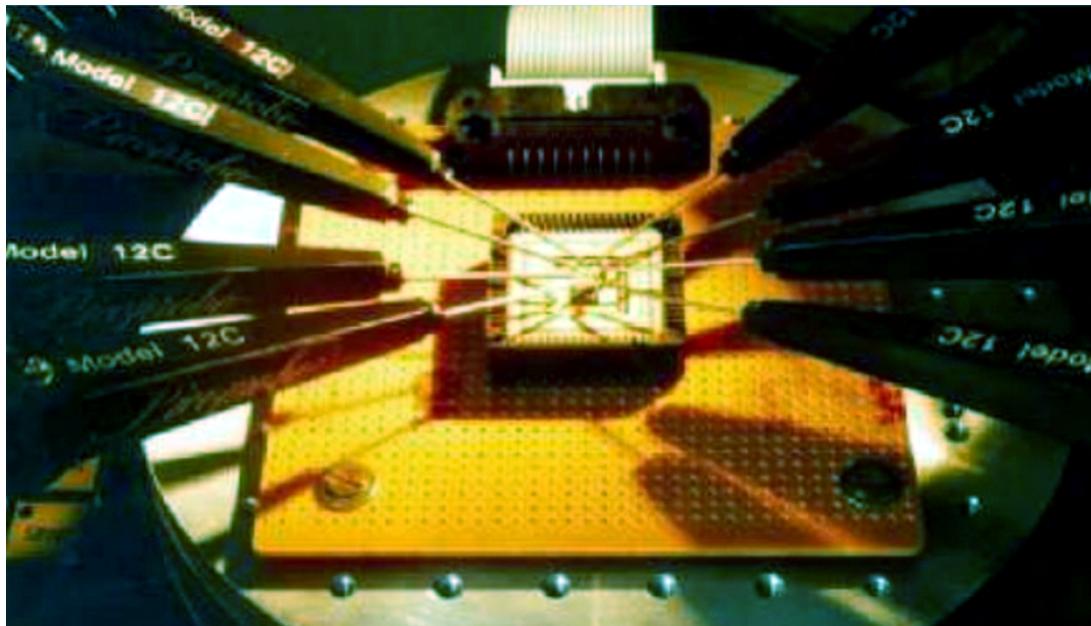
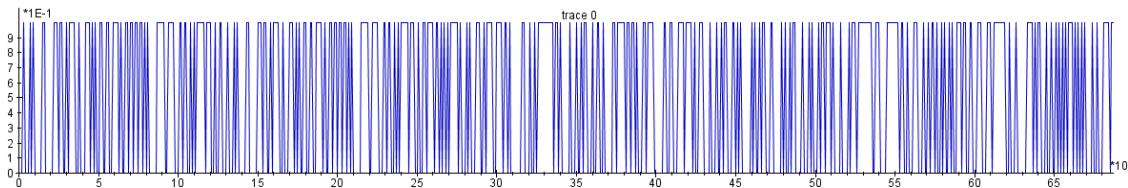


Yapluka → outils de DPA traditionnelle

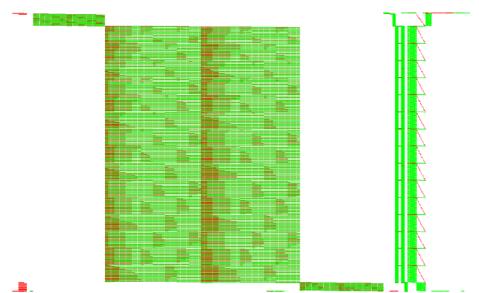
Riscure Inspector, ChipWhisperer, Matlab ou... Daredevil!



Differential Computation Analysis



Challenge Wyseur



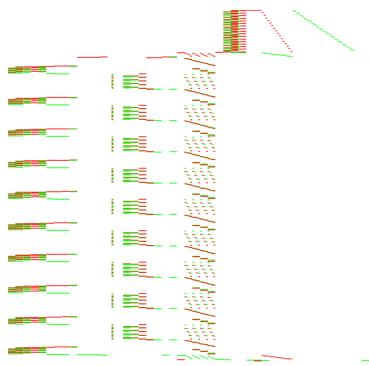
par Brecht Wyseur, 2007

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

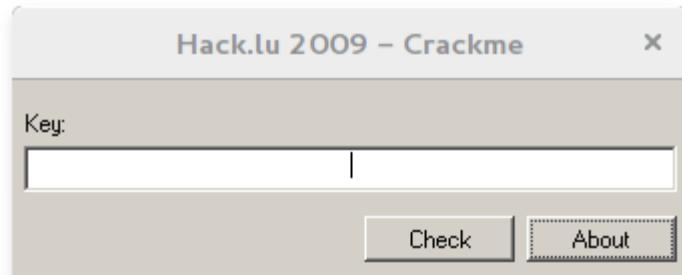
On télécharge...

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

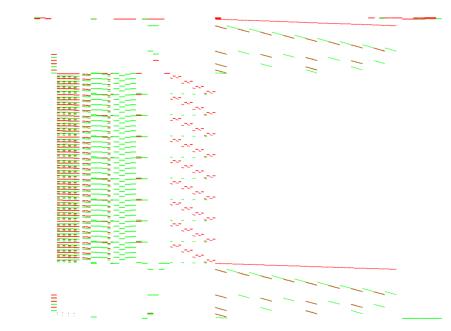
“white-box” en Python par Axel Tillequin
Implémentation WB-DES dans un objet “marshall”

→ 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.

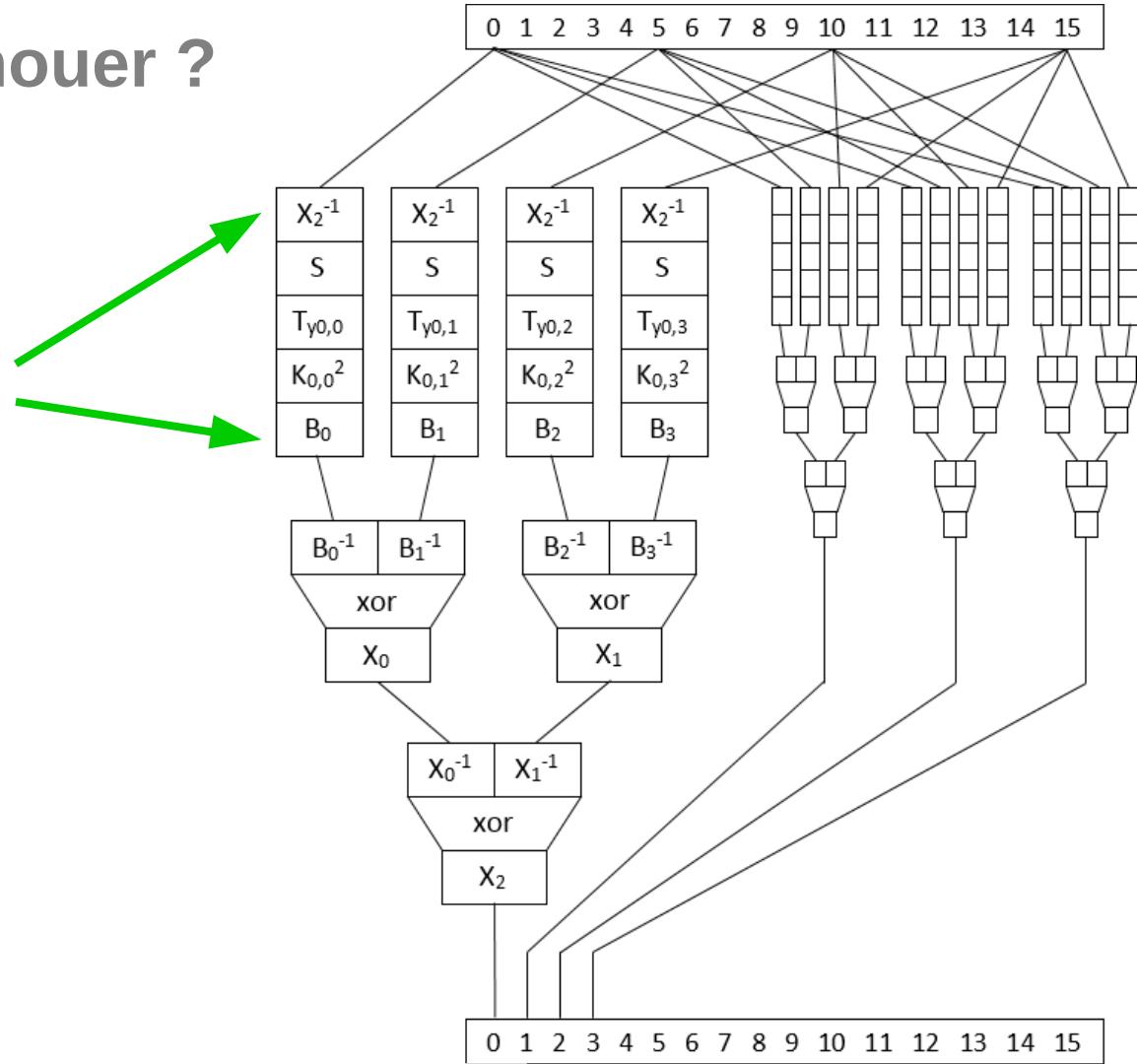
Oups, c'est la chute des prix...



La DCA peut-elle échouer ?

Oui !

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



La DCA peut-elle échouer ?

Oui !

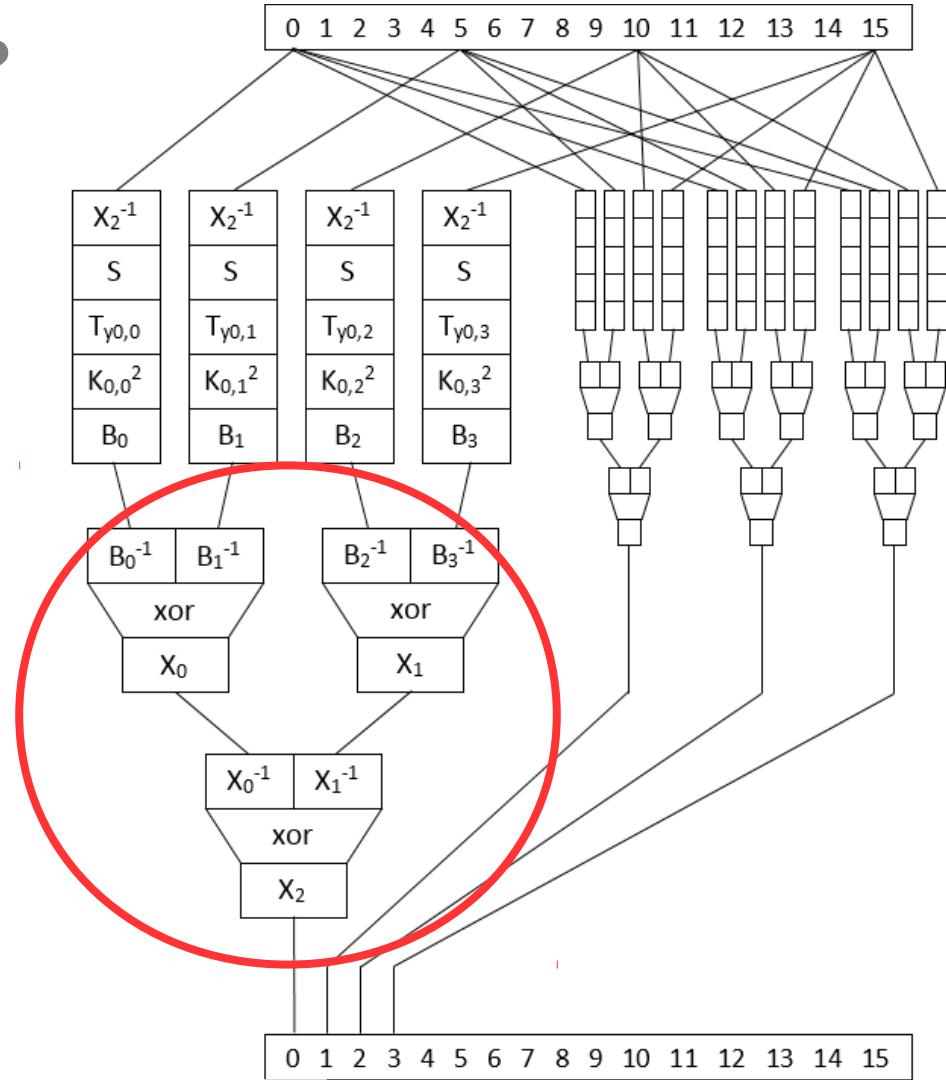
Encodages “larges” (8x8)
masquant la non-linéarité
de la SBOX

- 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



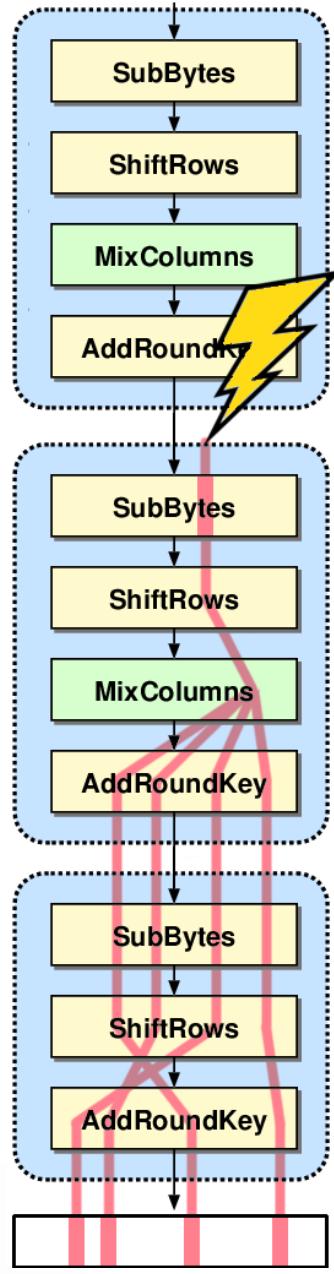
Autres attaques “grey box” à portée de main : Higher order DPA, CPA, DFA,....



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)

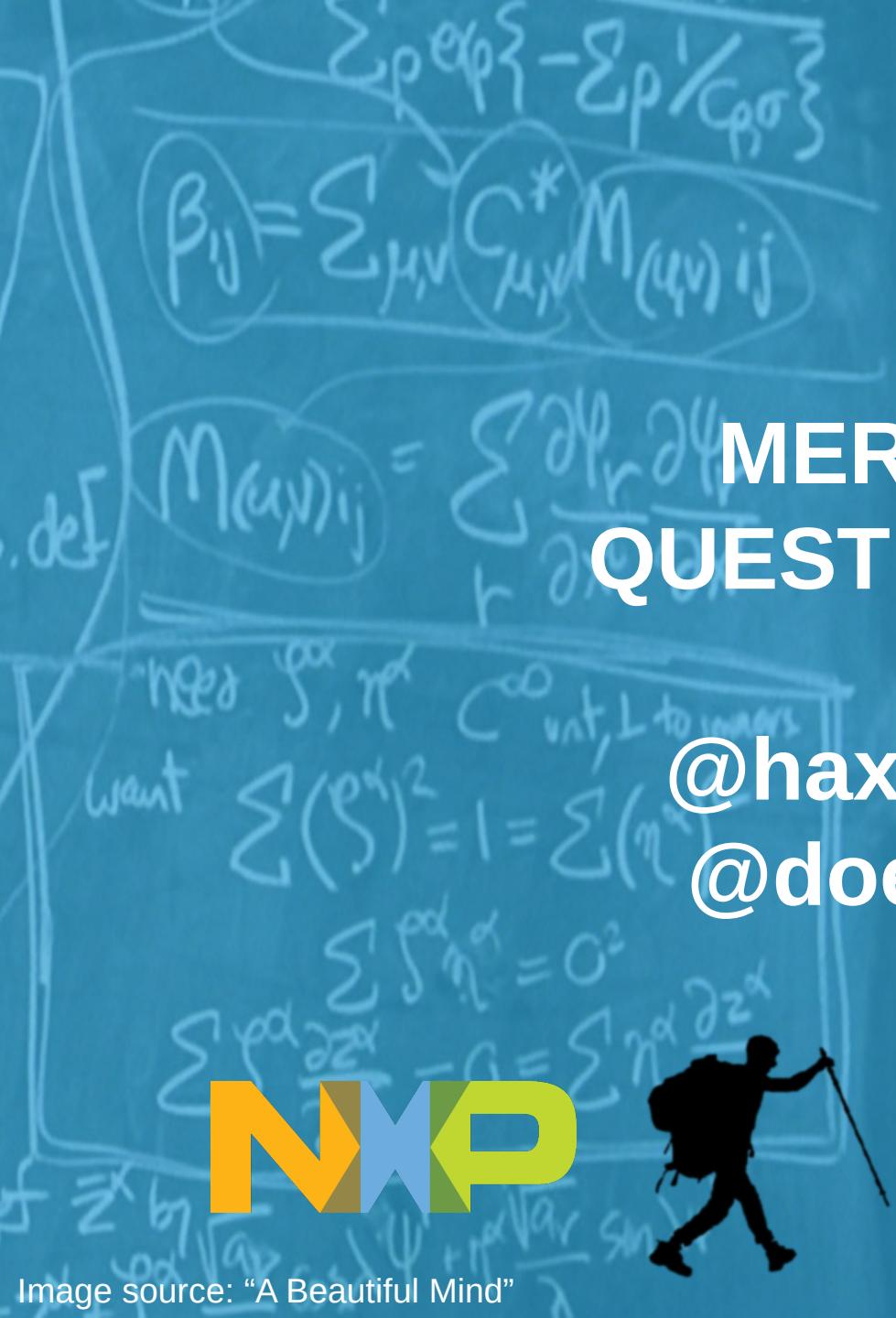
Van Huynh Le (U.Twente, NXP)

Wil Michiels (NXP, TU/e)

Sans oublier...



Orka
- images Docker



MERCI !
QUESTIONS ?

@haxelion
@doegox



Quarkslab

DEMO

