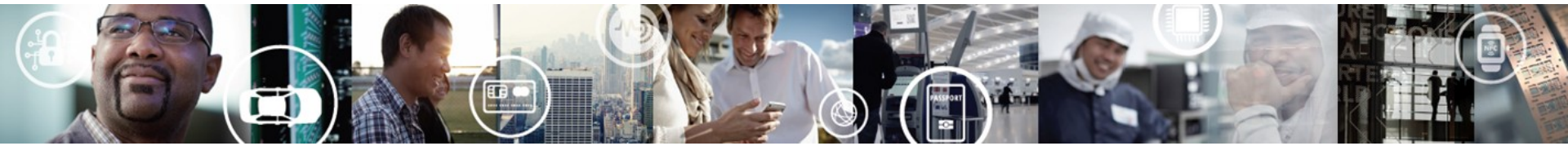
Hiding your White-Box Designs is Not Enough

#CYBSEC15

Philippe Teuwen

05/11/2015



SECURE CONNECTIONS
FOR A SMARTER WORLD

whoami

Philippe Teuwen aka @doegox aka yobibe

- @ Philips / NXP since Y2K
- ♥ free software, security, CTFs, photography
- 웹 http://wiki.yobi.be

Principal Researcher

- Innovation Center Crypto & Security
- Business Unit Security & Connectivity
- NXP Semiconductors

Outline

- Introduction to white-box cryptography
- Software execution traces
- Differential Computation Analysis

Mes excuses mais...

Credits:

Ecole française:

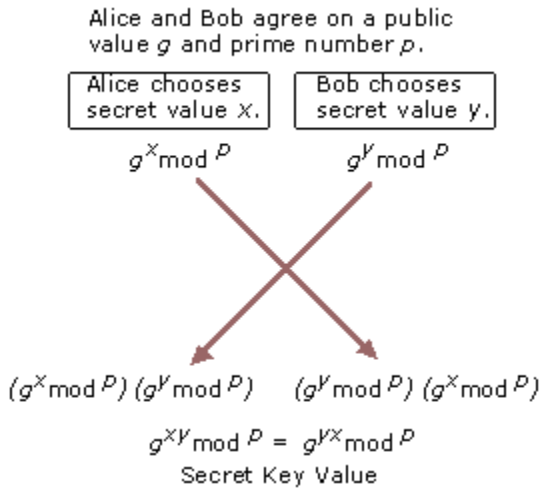
on dit “chiffrer”,
pas “crypter”

Ecole belge:

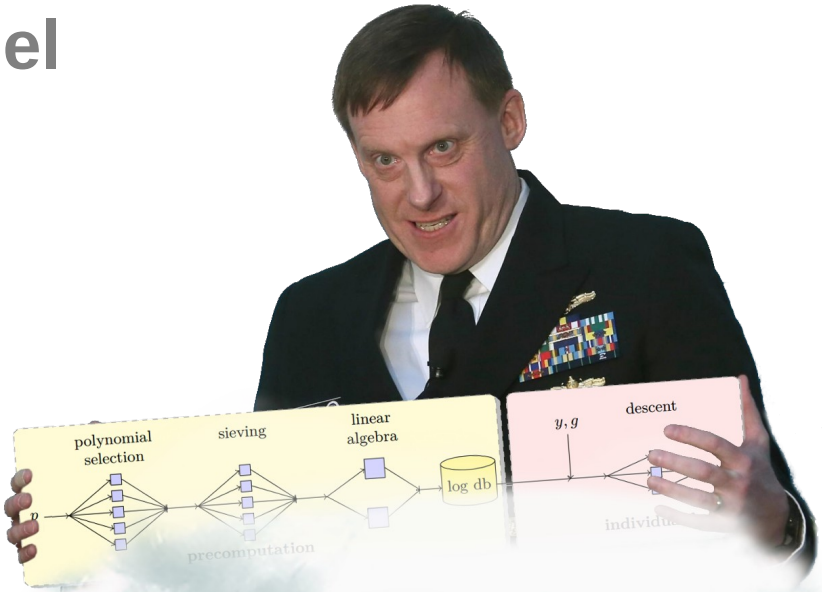
“décrypte le cipher”

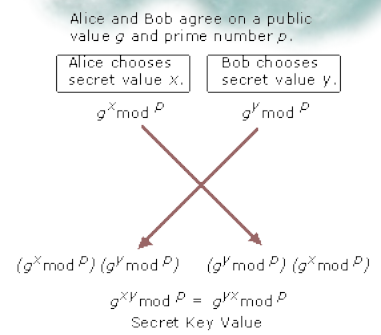
INTRODUCTION TO WHITE-BOX CRYPTOGRAPHY

Black box model

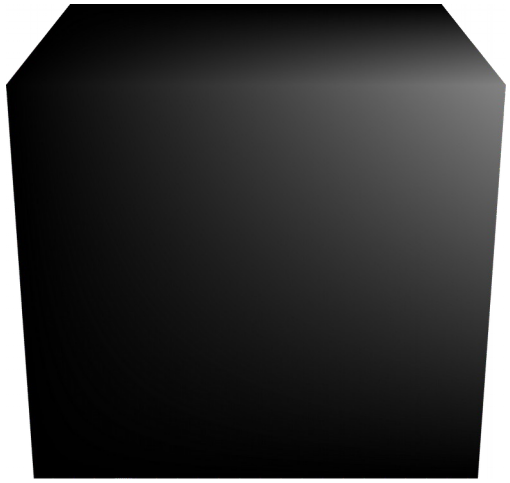


Black box model



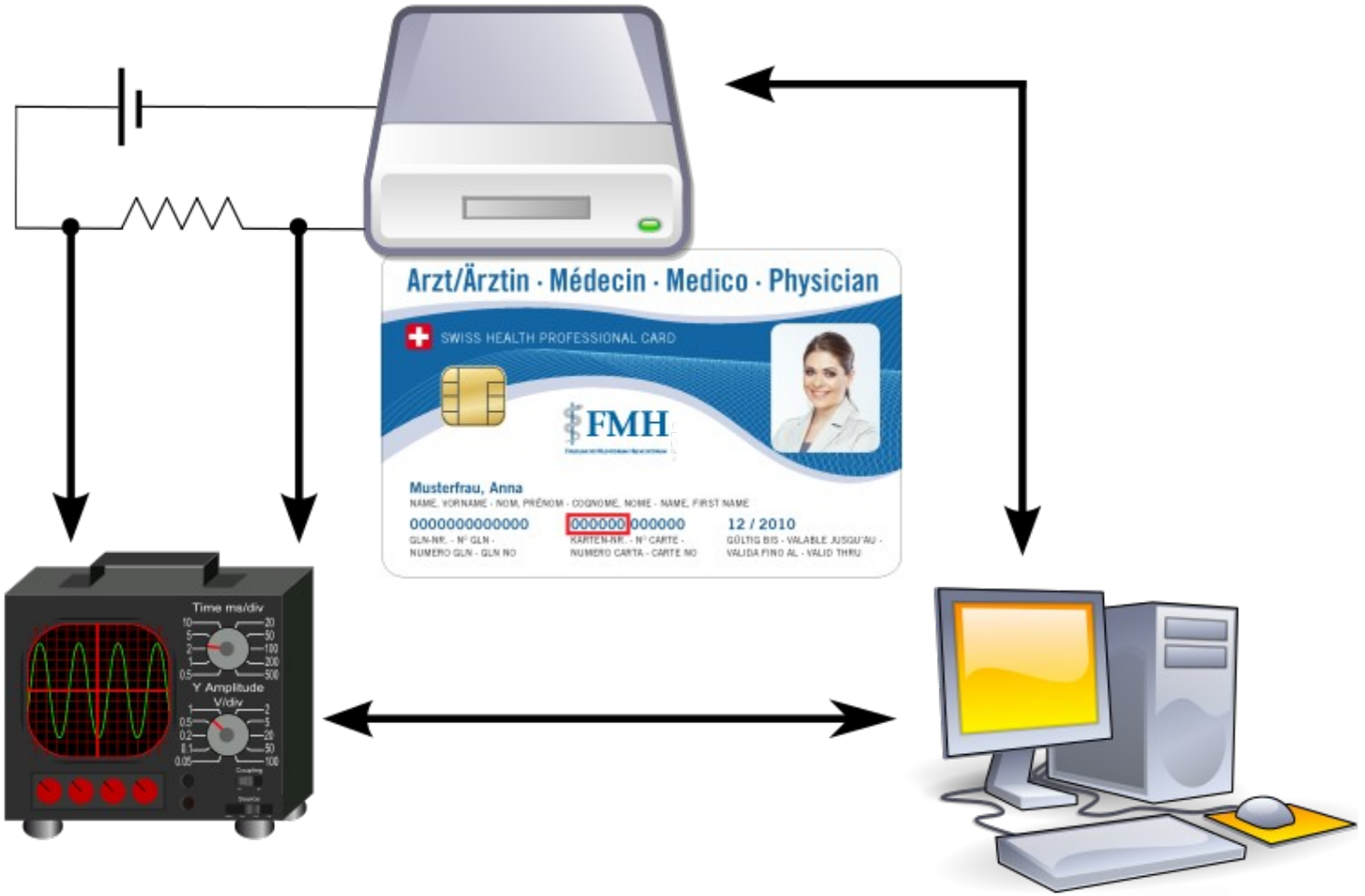


Black box model

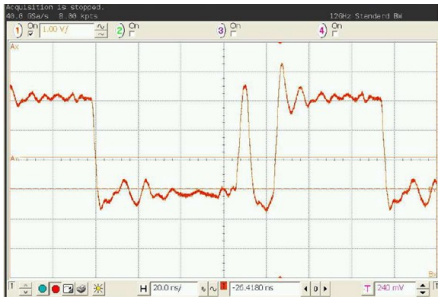


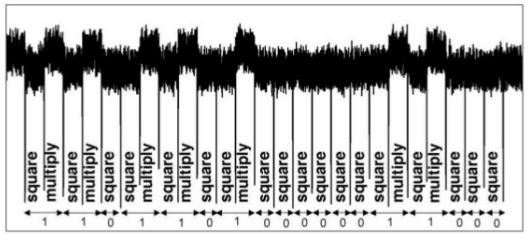
Grey box model

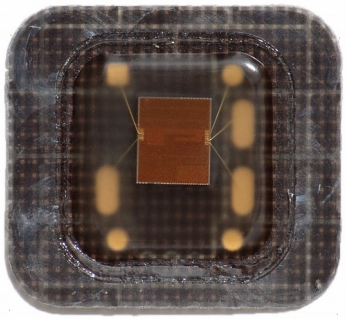
Grey box model

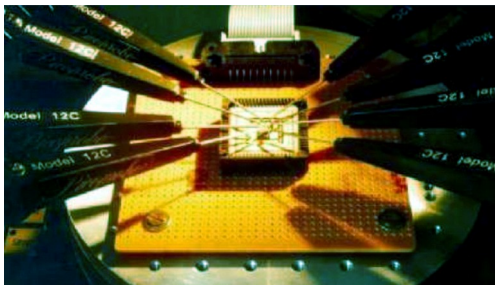


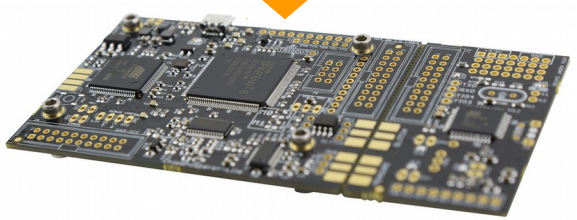
Grey box model





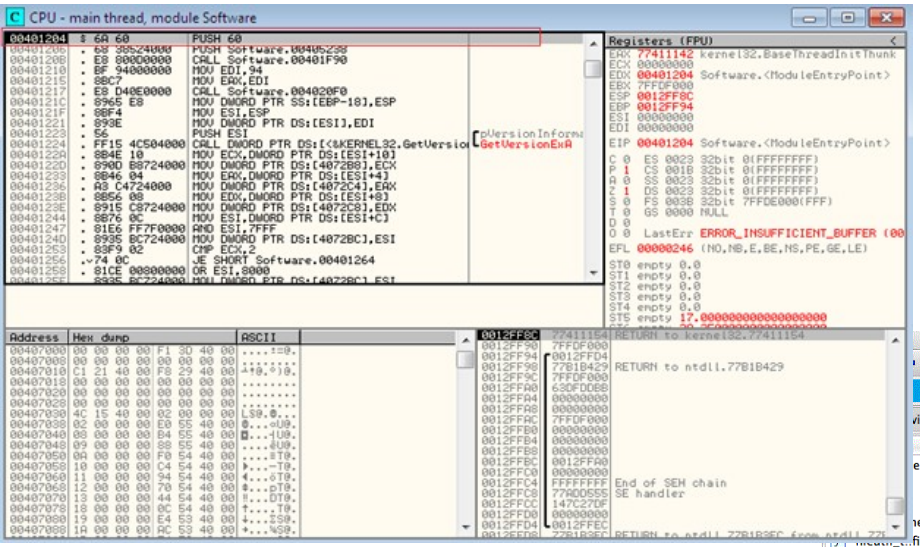


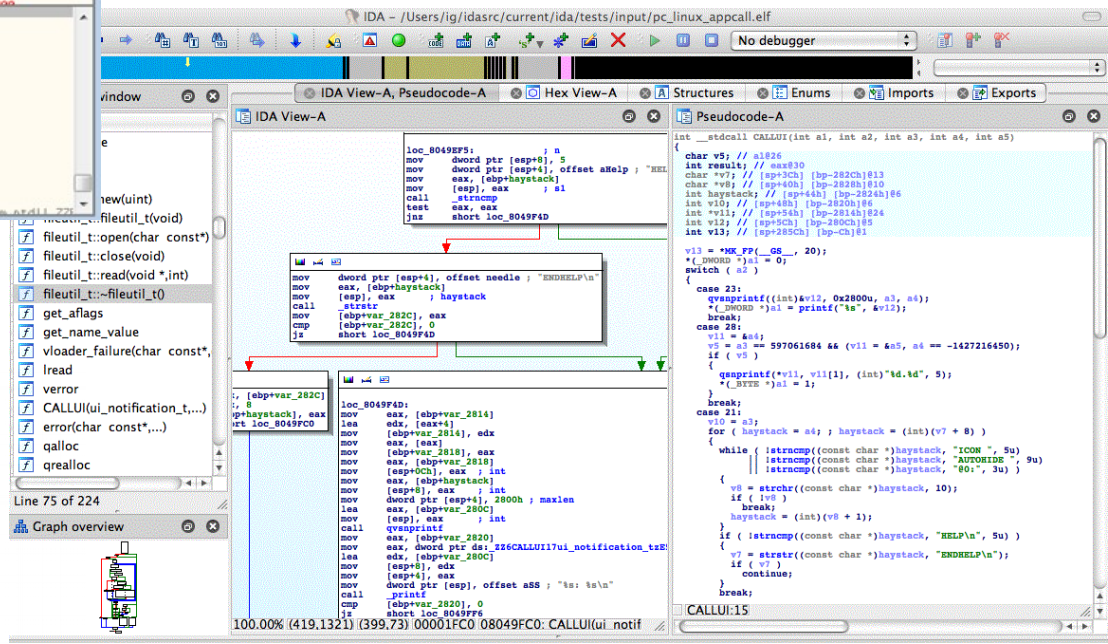




White box model

White box model





NXP

White box model

Attacker: full access to the execution environment

- Read/alter binary
- Step execution
- Read/alter CPU registers, memory, I/Os, …

Your sole line of defense:

- The implementation

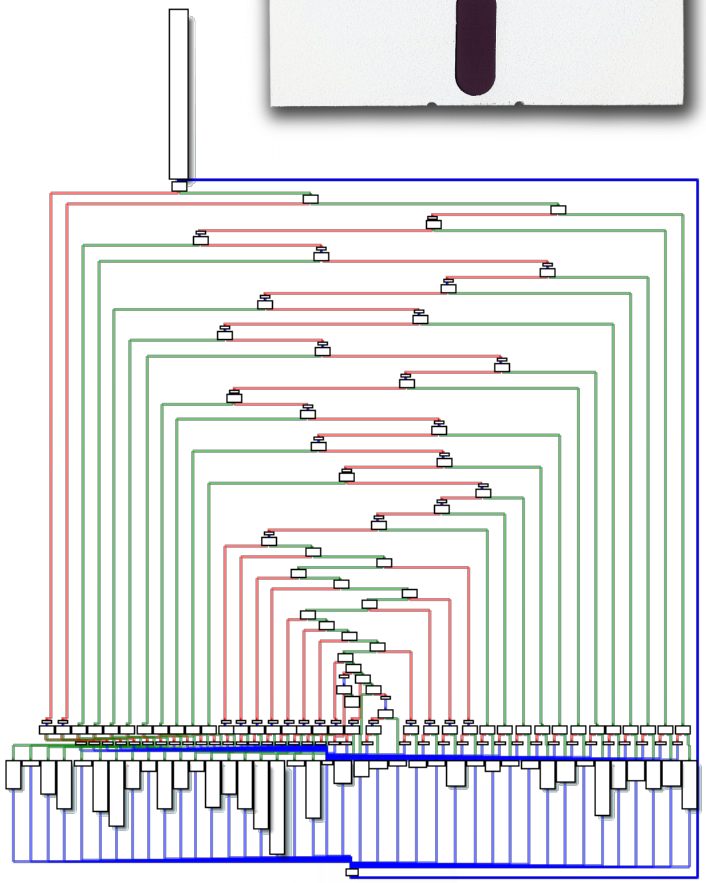
If you're brave, Kerckhoff tells you to expose your design as well

Usual countermeasures

Code obfuscation

Integrity checks

Anti-debug tricks



```
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]
```

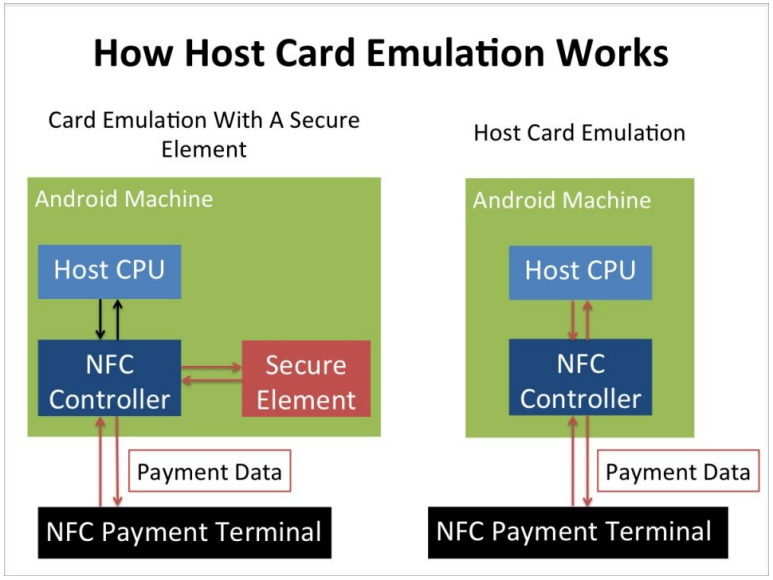
Illustrations: by @xoreaxeaxeax about M/o/Vfuscator and blog.quarkslab.com about OLLVM

Cryptography under White-box model

What if you need to do some crypto in such hostile environment?

- DRM schemes ↔ ~~criminals~~ users
- Mobile payment, HCE ↔ malwares

Source: “l'industrie du film”



Source: Business Insider

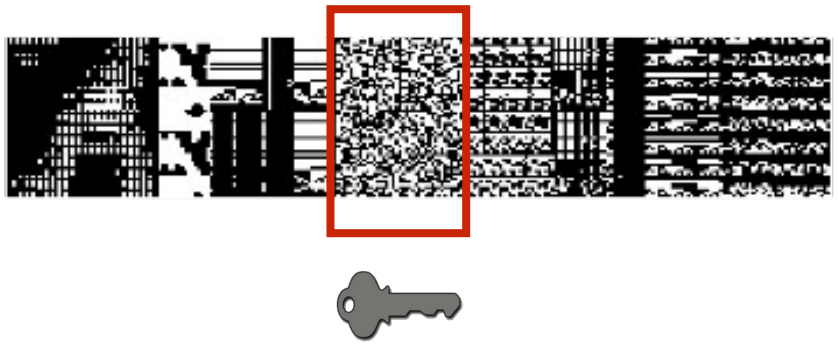
Cryptography under White-box model

What if you need to do some crypto in such hostile environment?

- DRM schemes ↔ ~~criminals~~ users
- Mobile payment, HCE ↔ malwares

Obfuscation techniques alone are mostly insufficient

- Obfuscation mainly about securing code but here:
  standard crypto algo in need for strong key protection
- E.g. entropy attack on RSA by Shamir and Van Someren (1999)



White-box cryptography

Chow et al. (2002)

- “Ideal” WB AES implementation:
  One big lookup table
  4.94×10^{27} TB
- Practical WB AES:
  Network of smaller tables
  752kB
  Encoding on intermediate values

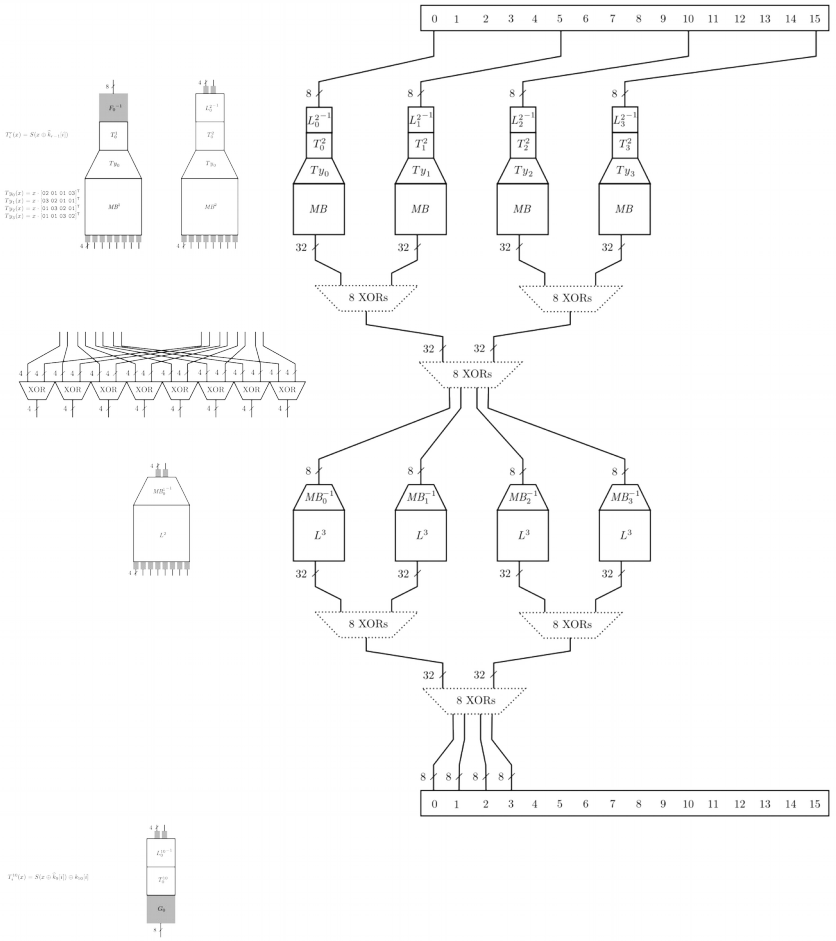


Illustration from “A Tutorial on White-box AES” by James A. Muir

White-box cryptography

History:

- Academic attacks
- New academic designs
- New academic attacks, etc.
- Today, all academic schemes have been broken

Industry response:

- Keep white-box designs secret
- Bury white-box implementation under layers of code obfuscation, integrity checks, anti-debug tricks
- Claim to be equivalent to a Secure Element

“Academic” attacks?

Require reversing of all the obfuscation layers
Require knowledge on the design
Then apply attack:

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*

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

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

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

$$\begin{cases} \mathtt{SR} \circ \bigoplus_{K'_{10}} \circ \{S_{10,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_9} \circ \Lambda_9'^{-1} & \text{for } R_{10} \ , \\ \mathtt{MC} \circ \mathtt{SR} \circ \Lambda_r'' \circ \{S_{r,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_{r-1}} \circ \Lambda_{r-1}'^{-1} & \text{for } R_r|_{2\leq r\leq 9} \ , \\ \mathtt{MC} \circ \mathtt{SR} \circ \Lambda_1'' \circ \{S_{1,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_0} & \text{for } R_1 \ , \end{cases} \quad (5)$$

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:

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

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

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

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

$$\begin{aligned} \mu_r(x_0x_1\ldots x_{47}, y_0\ldots y_{47}) &= y_0\ldots y_5 x_{\mu_r^{-1}(0)} x_{\mu_r^{-1}(1)} y_6\ldots y_{11} x_{\mu_r^{-1}(2)} x_{\mu_r^{-1}(3)} \cdots y_{42}\ldots y_{47} x_{\mu_r^{-1}(22)} x_{\mu_r^{-1}(23)} \cdots x_{\mu_r^{-1}(47)} \\ \gamma_r(z_0z_1\ldots z_{95}) &= z_{\gamma_r^{-1}(0)}\cdots z_{(\gamma_r^{-1}(0)+5)} z_6 z_7 \ldots z_{\gamma_r^{-1}(11)}\cdots z_{(\gamma_r^{-1}(11)+5)} z_{94} z_{95} \end{aligned}$$

The obfuscated t-box is

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

Hence the transformed function is:

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

with

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

By setting

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

the resulting encryption operation is

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

Excerpts:

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

“Academic” attacks?

= a lot of reverse-engineering effort then, anyway, for me:

Definition 3. *The mapping* $\overline{AES}_{enc}^{(r,j)} : (\mathbf{F}_2^8)^4 \to (\mathbf{F}_2^8)^4$ *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*

$$\Pi_2^{(r,j)} \circ \overline{MS}^{(r,\pi^{(r)}(j))} \circ \Pi_1^{(r,j)} \circ \overline{MC}^{(r,j)} \circ (S,S,S,S) \circ \oplus_{[\bar{k}_i^{(r,j)}]}$$

$$[\bar{k}_i^{(r,j)}]_{0\leq i\leq 3} \qquad ([k_i^{(r,\pi^{(r)}(j))}]_{0\leq i\leq$$

$$\mathtt{MC}^{(r,j)} \qquad \mathtt{MC} \circ \Pi_1^{(r,j)} \ .$$

As a r… algorithm mentioned … white-box adv…ary … black-box acces… the following structures of each round $R_r|_{r=1,\ldots,1}$

$$\begin{cases} \mathtt{SR} \circ \bigoplus_{K'_{10}} \circ \{S_{10,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_9} \circ \Lambda_9'^{-1} & \text{for } R_{10} \ , \\ \mathtt{MC} \circ \mathtt{SR} \circ \Lambda_r'' \circ \{S_{r,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_{r-1}} \circ \Lambda_{r-1}'^{-1} & \text{for } R_r|_{2\leq r\leq 9} \ , \\ \mathtt{MC} \circ \mathtt{SR} \circ \Lambda_1'' \circ \{S_{1,i}\}|_{i=0,\ldots,15} \circ \bigoplus_{K_0} & \text{for } R_1 \ , \end{cases} \quad (5)$$

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:

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

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

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

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

$$\mu_r(x_0x_1 \ldots y_0 \ldots y_0 \ldots y_5 x_{\mu_r^{-1}(0)} x_{\mu_r^{-1}(1)} \ldots x_{\mu_r^{-1}(2)} x_{\mu_r^{-1}(3)} \ldots y_{42} \ldots x_{\mu_r^{-1}(47)}$$
$$\gamma_r(\ldots)z_1 \ldots = z_{\gamma_r^{-1}(0)} \ldots z_{(\gamma_r^{-1}(0)+5)} \ldots z_{(\gamma_r^{-1}(11)+5)} z_{94} z_{95}$$

The obfusca… $T_r'^k(x) = (\ldots)(x)$.

…nce the t…nsform… function is:

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

with

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

By setting

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

the resulting encryption operation is

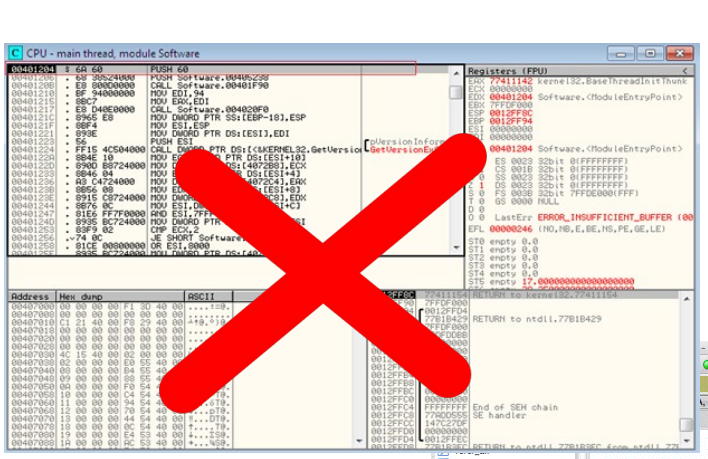
$$E^k(x) = \left[\tau_{n+1}'^k \cdot \left(\tau_n'^k \cdot T_n'^k \right) \cdot \ldots \cdot \left(\tau_1'^k \cdot T_1'^k \right) \cdot \tau_0'^k \right] (x)$$

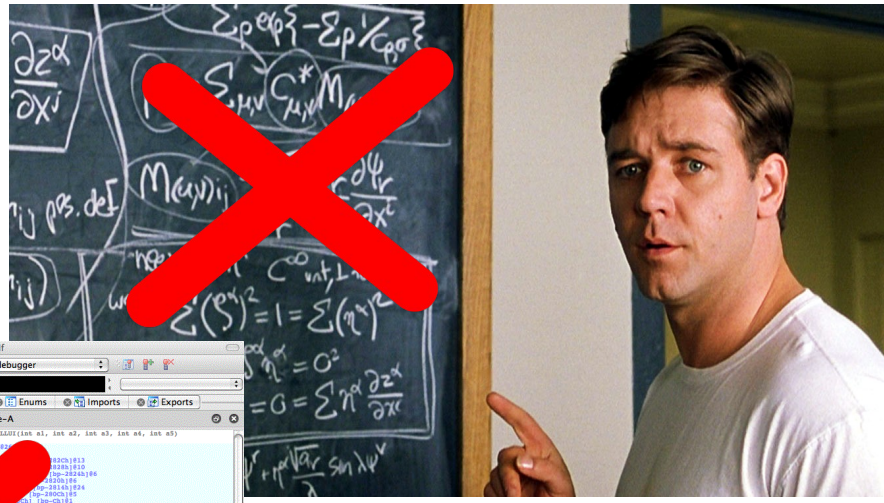
看起來像天書

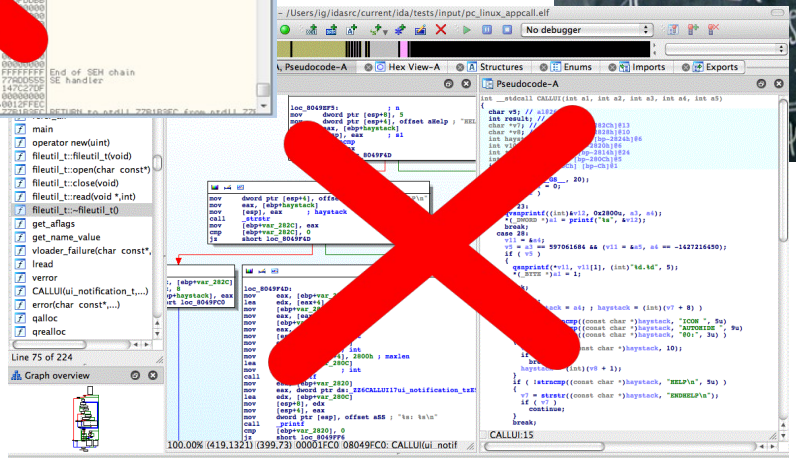
Our goal

Recover white-box keys

- without much reverse-engineering effort
- without much intellectual effort ^^







SOFTWARE EXECUTION TRACES

Tracing binaries

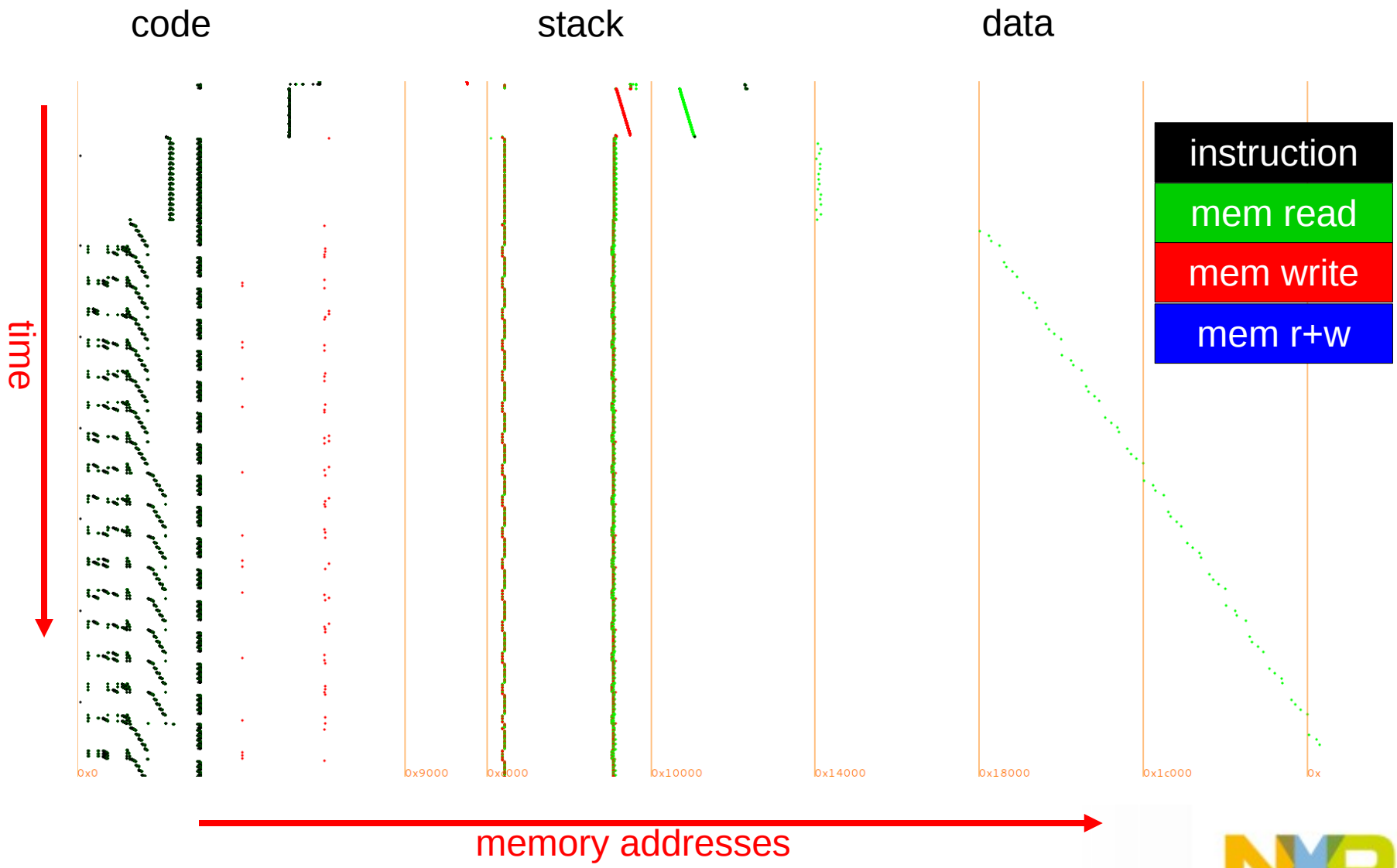
Software execution trace

- Record all instructions and memory accesses
- Using dynamic binary instrumentation or hooking into emulators
- Transparent even in case of integrity checks or anti-debug tricks

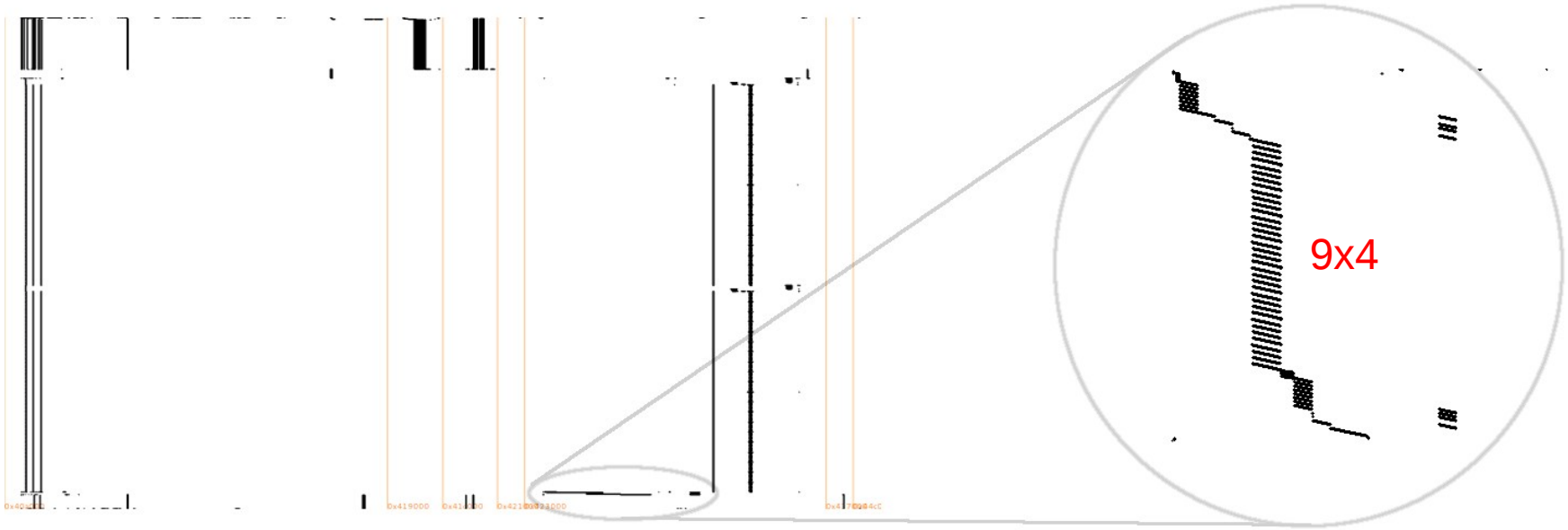
Examples of what we did:

- Intel PIN (x86, x86-64, Linux, Windows, Wine/Linux)
- Valgrind (idem+ARM, Android)
- Add hooks to VM (Java, Python,…)
- Add hooks to emulators (for exotic platforms)

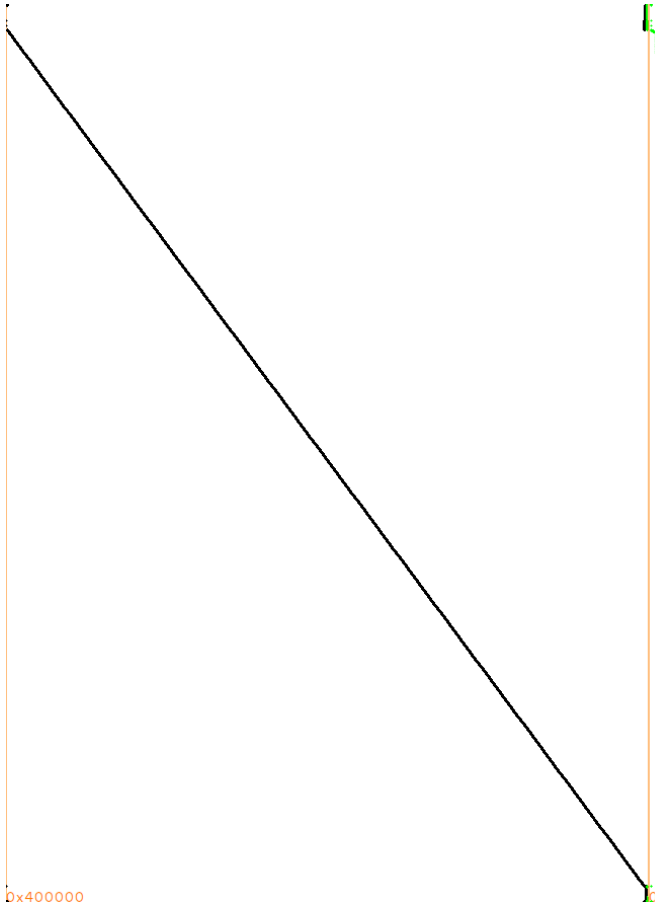
Trace visualisation convention: pTra waterfall



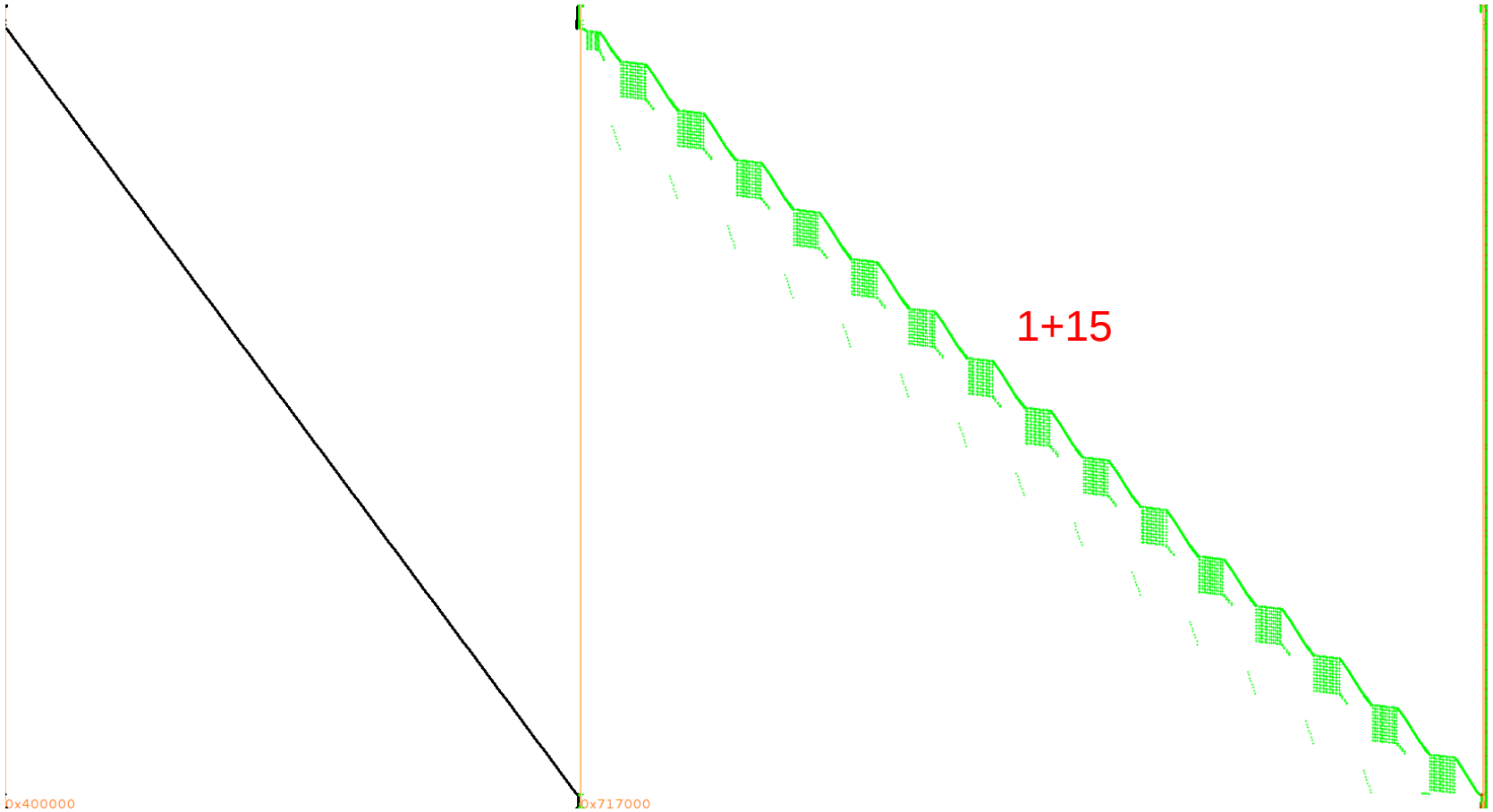
Visual crypto identification: code



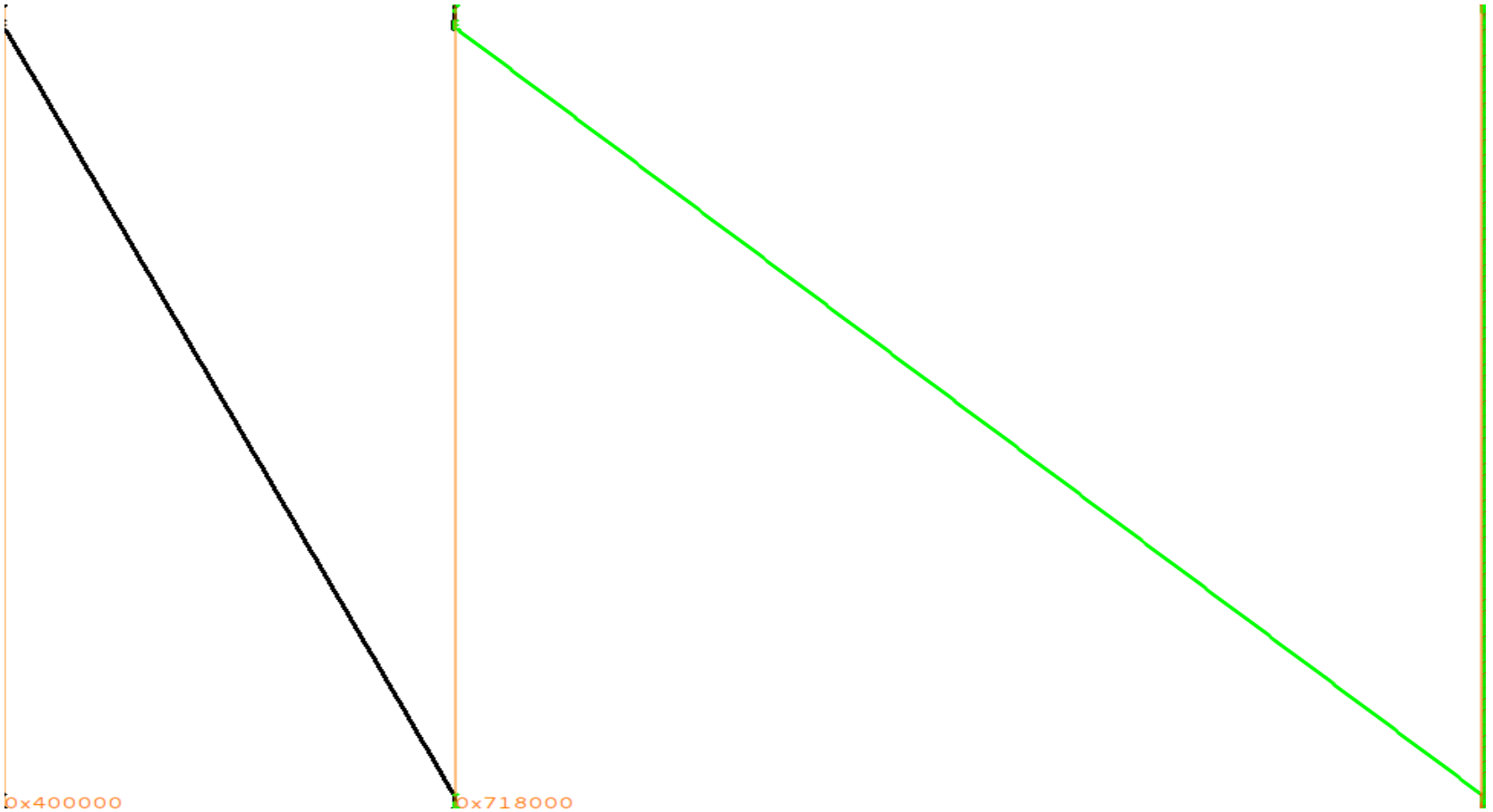
Visual crypto identification: code?



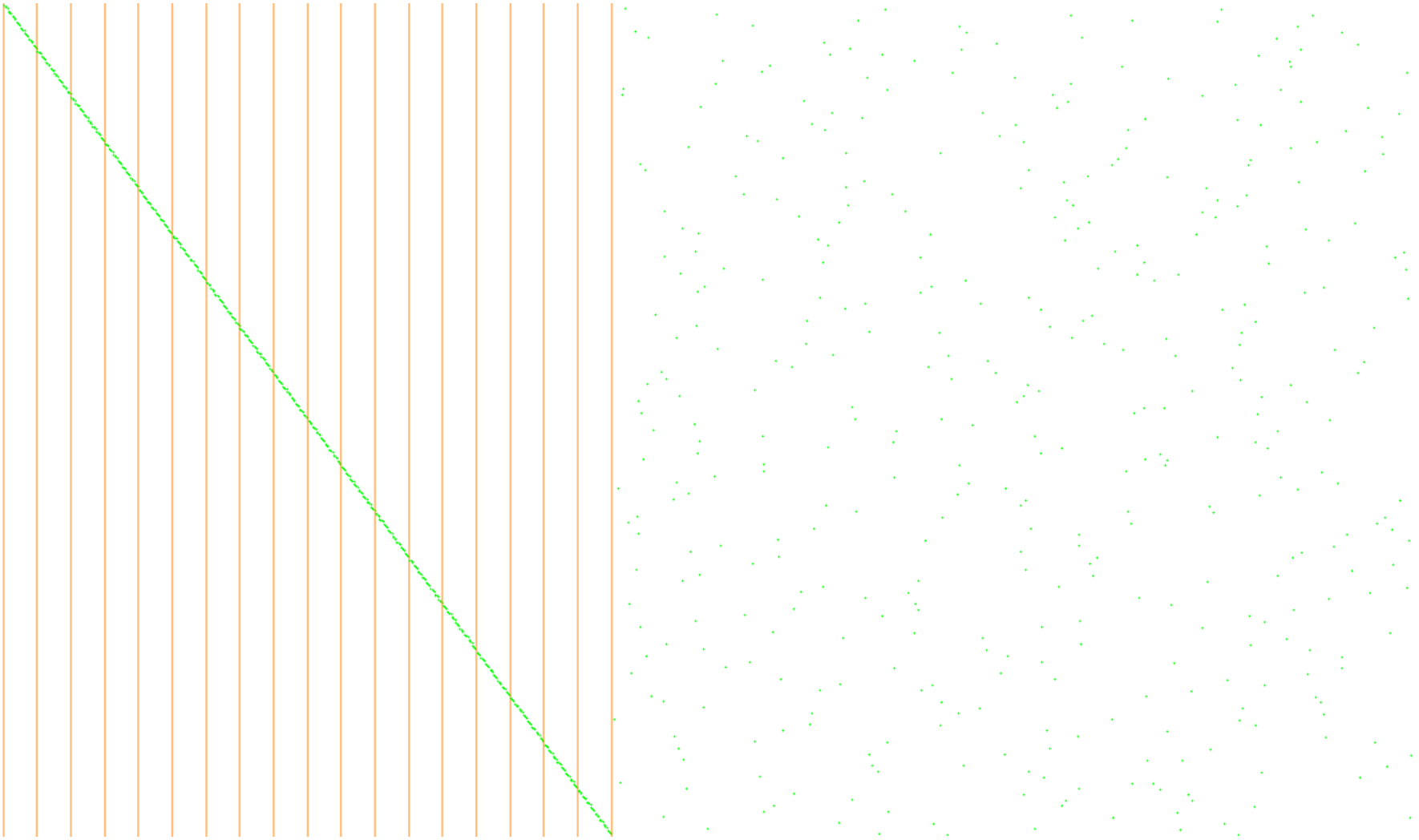
Visual crypto identification: code? data!



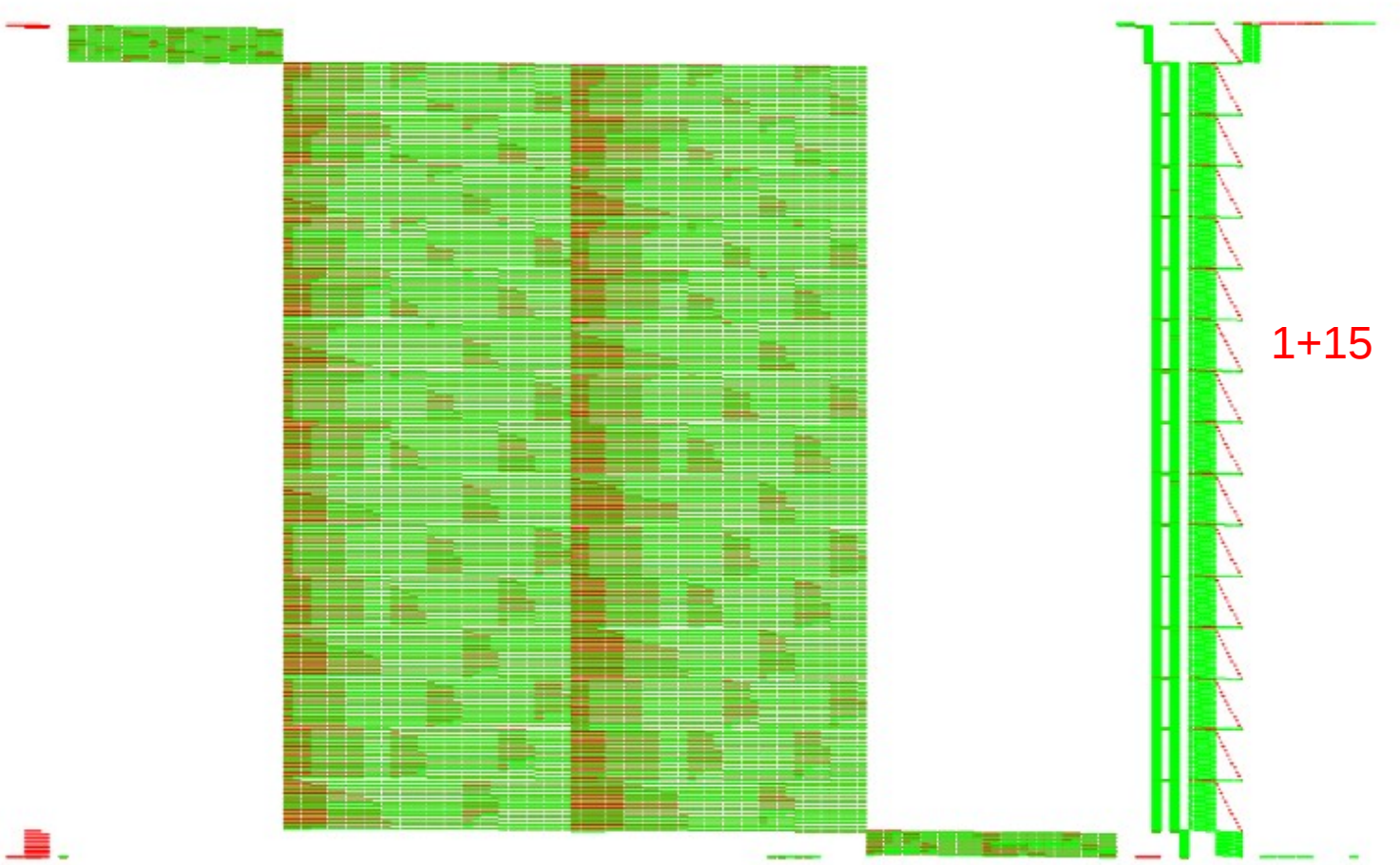
Visual crypto identification: code? data?



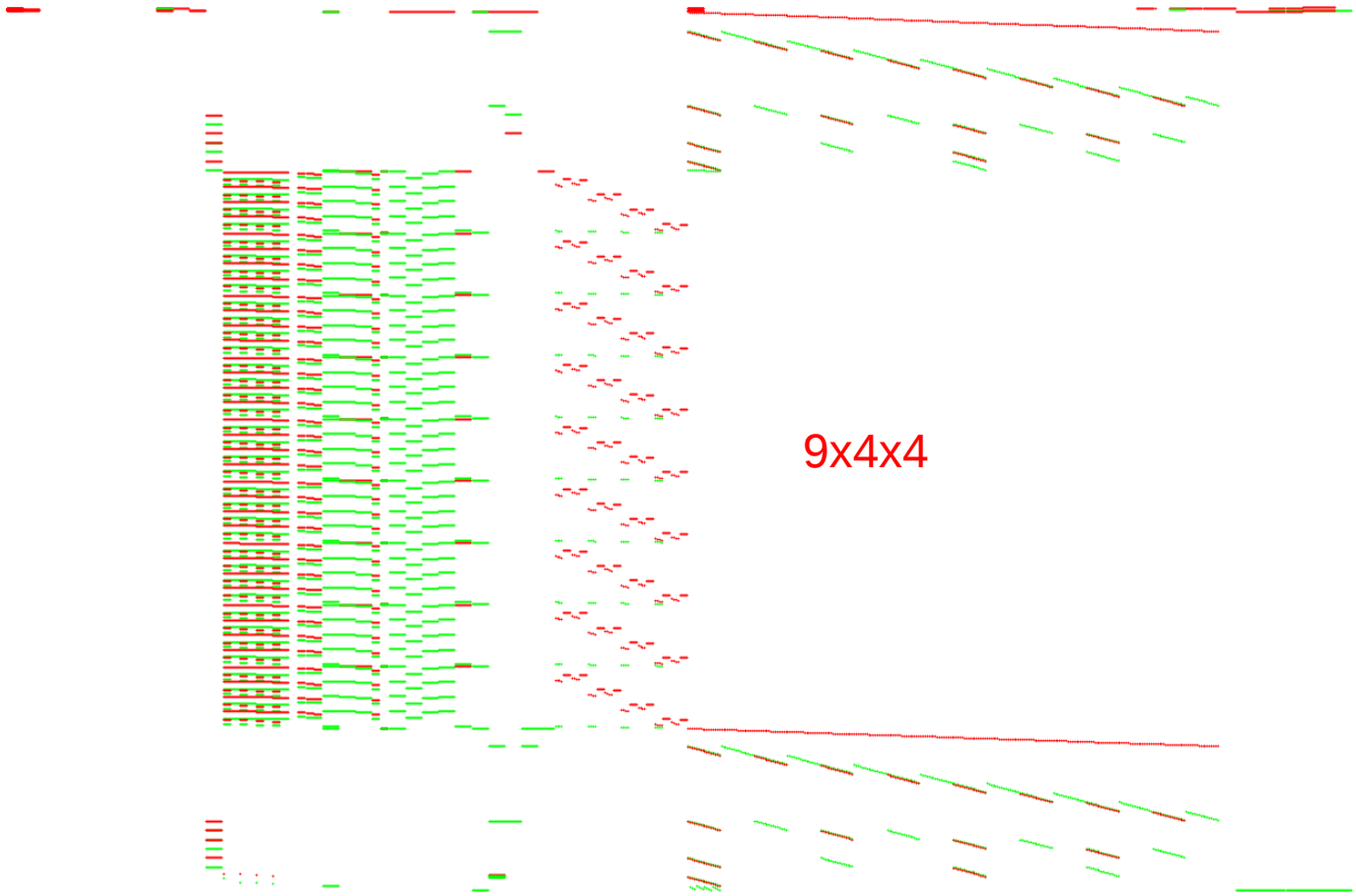
Visual crypto identification: data?



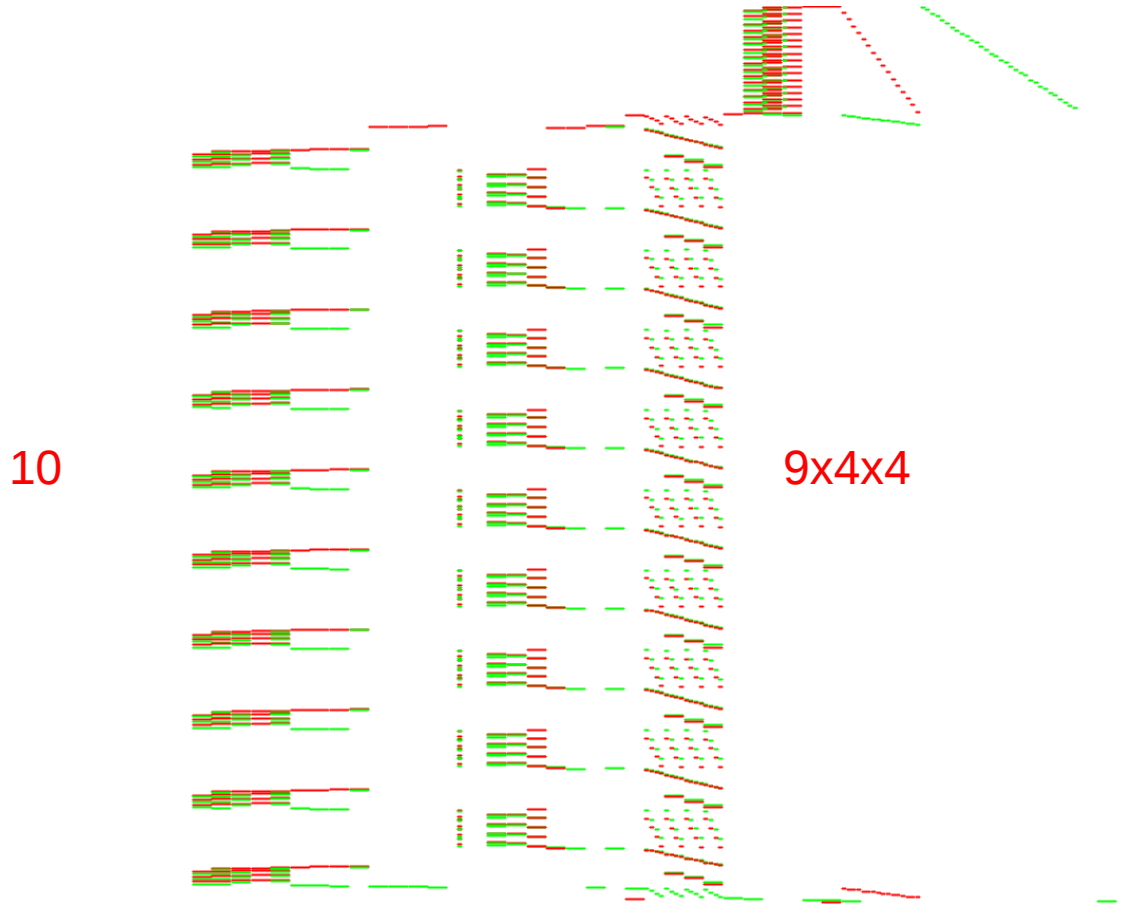
Visual crypto identification: stack!



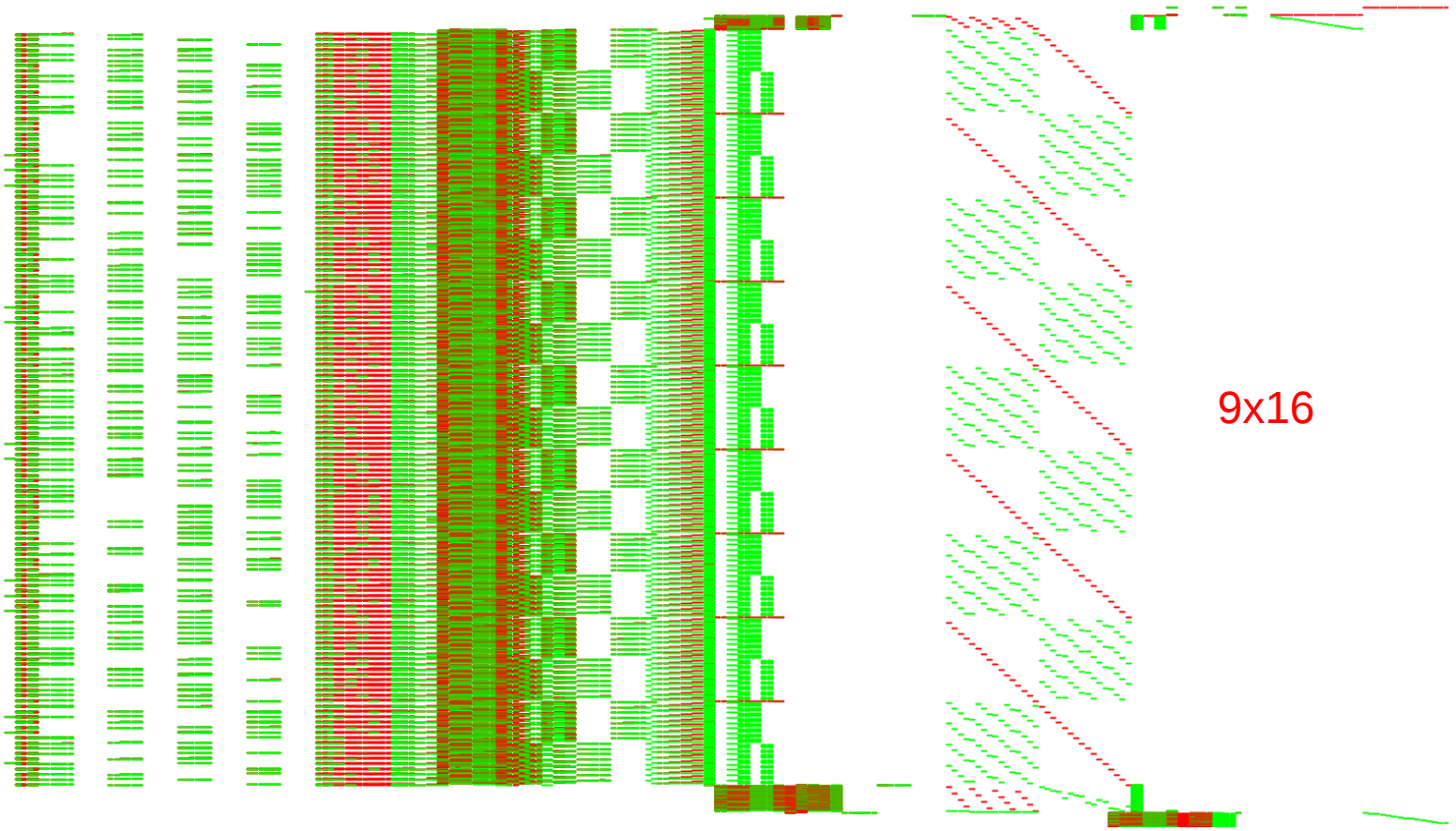
Visual crypto identification: stack!



Visual crypto identification: stack!



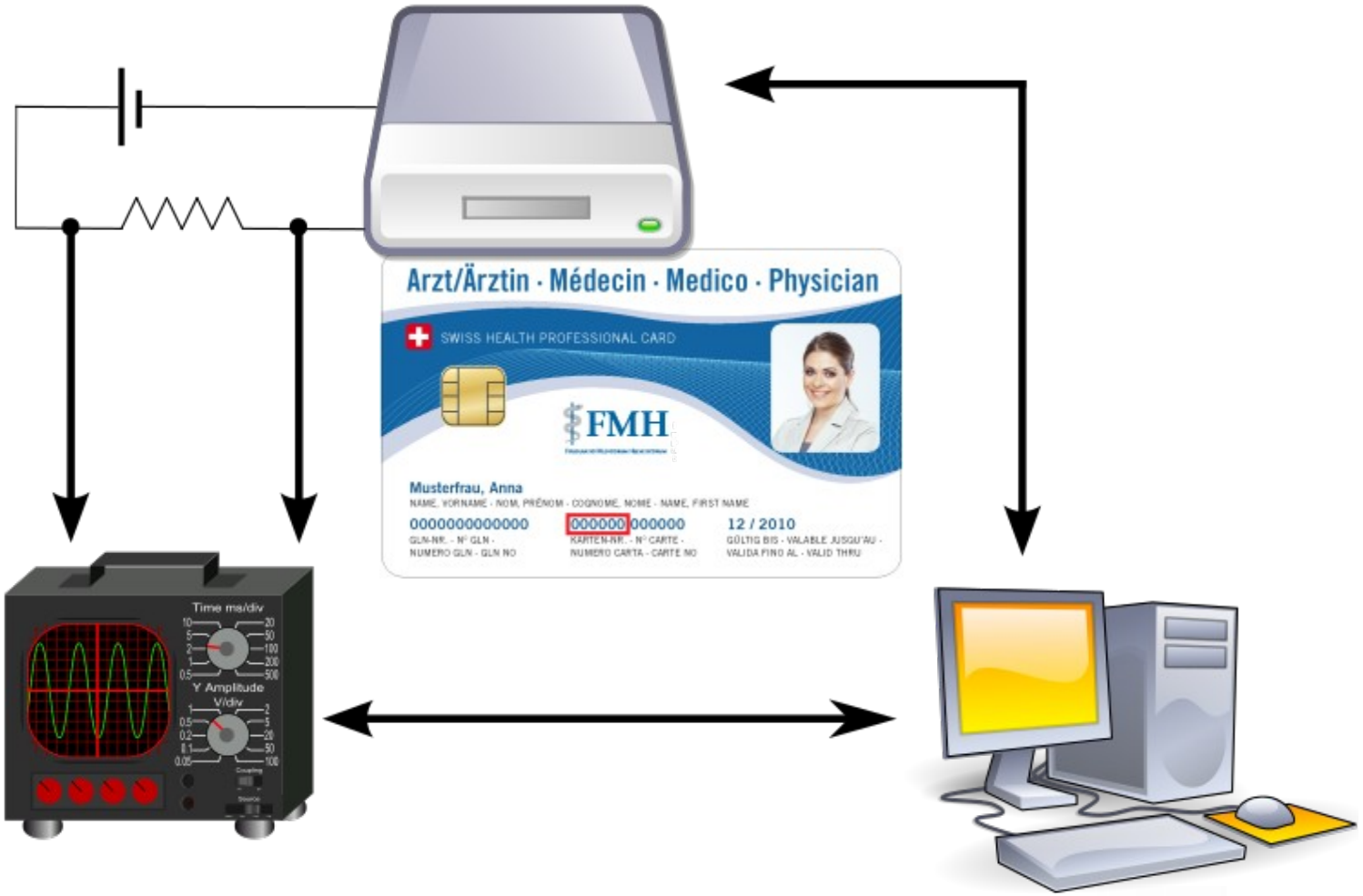
Visual crypto identification: stack!



Where is my key?

DIFFERENTIAL COMPUTATION ANALYSIS

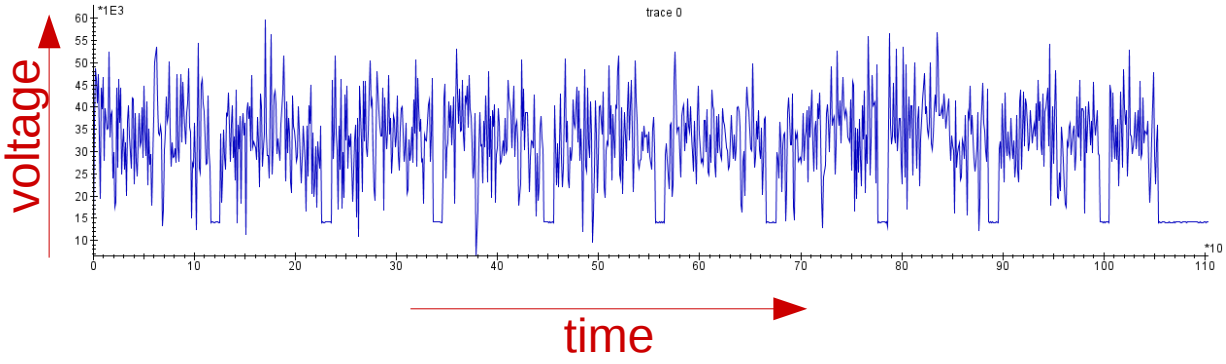
Remember?

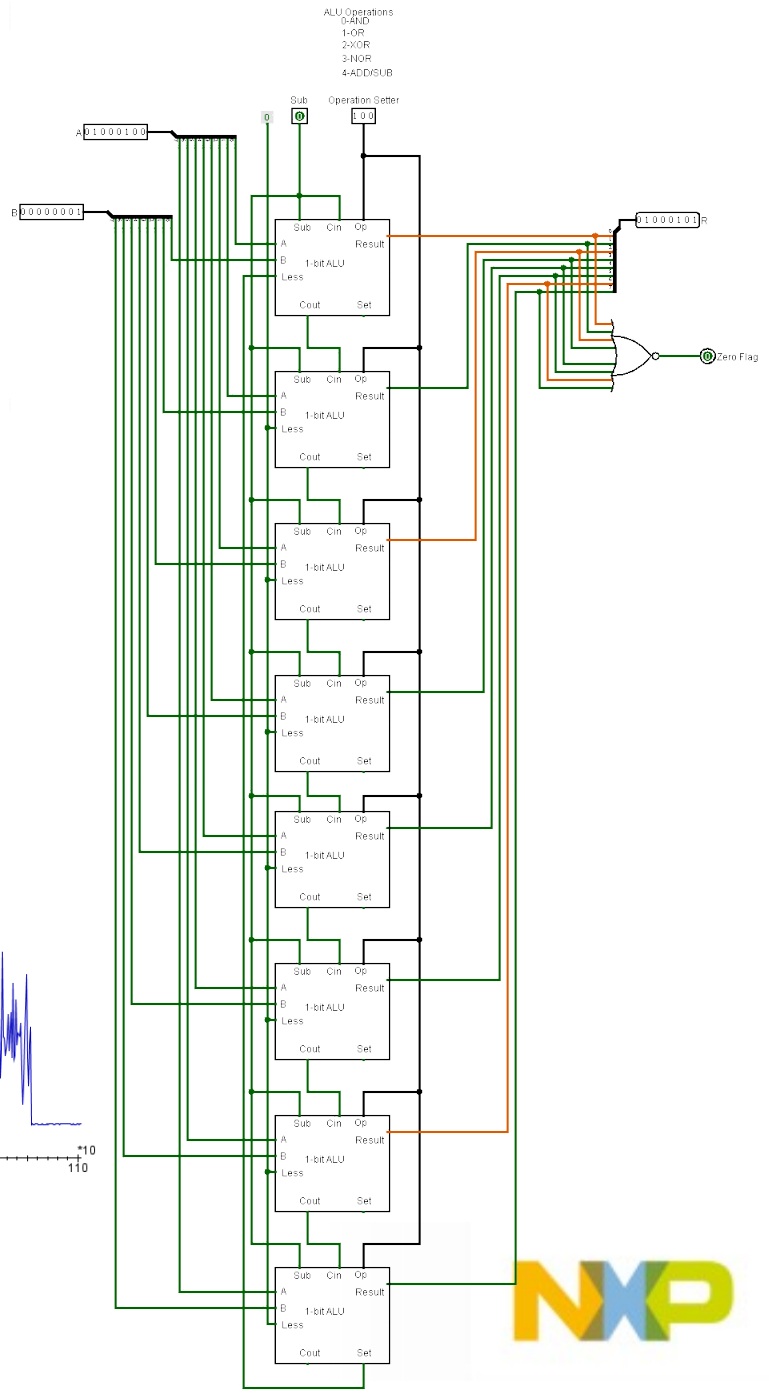

Arzt/Ärztin · Médecin · Medico · Physician
SWISS HEALTH PROFESSIONAL CARD
FMH
Musterfrau, Anna
NAME, VORNAME · NOM, PRÉNOM · COGNOME, NOME · NAME, FIRST NAME
0000000000000
GLN-NR. · N° GLN · NUMERO GLN · GLN NO
000000 000000
KARTEN-NR. · N° CARTE · NUMERO CARTA · CARTE NO
12 / 2010
GÜLTIG BIS · VALABLE JUSQU'AU · VALIDA FINO AL · VALID THRU
Time ms/div
Y Amplitude
V/div

All started with Differential Power Analysis

by P. Kocher et al. (1998)

- Probable correlations:
  power consumption vs.
  Hamming weight of internal values
- Record many traces
  while providing different inputs

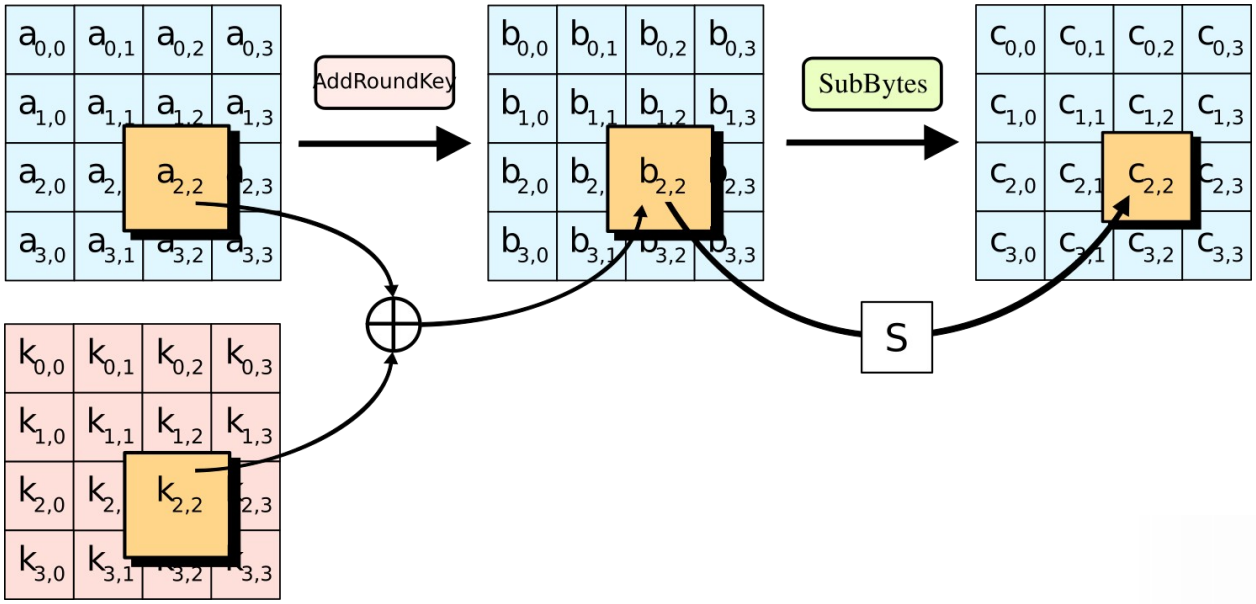




Differential Power Analysis

Some intermediate values in first (or last) round depend only on known data and a fraction of the round key

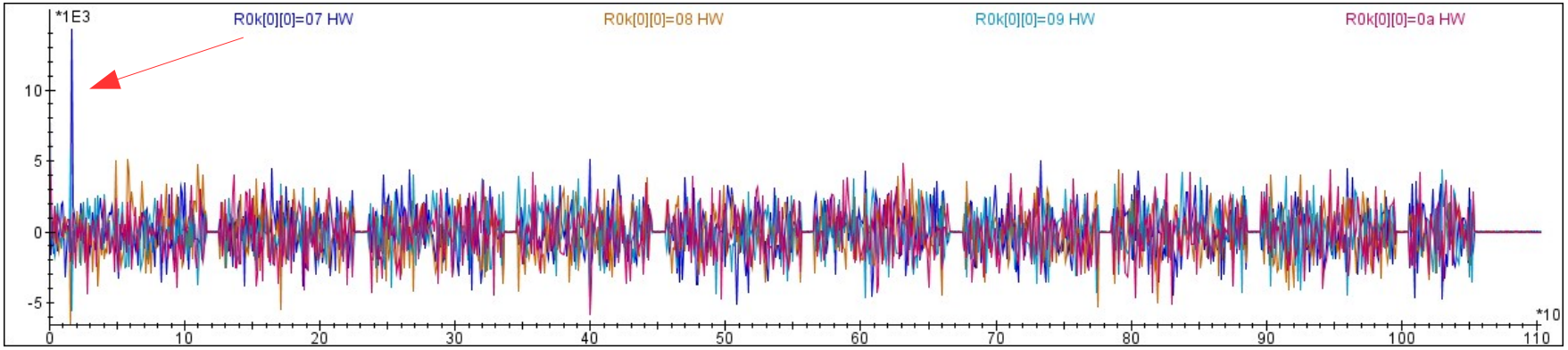
E.g. for AES:



Differential Power Analysis

1) Make a guess on that fraction of key
2) Evaluate targeted intermediate value for each plaintext: 0 or 1?
3) Sort traces accordingly in two buckets and average them
4) Compute differences between those averages

If the key guess is correct, it'll show up:



Differential Power Analysis

Very powerful grey box attack!

Requirements:

- Either known input or known output
- Ability to trace power consumption (or EM radiations)
- Some leakage

Differential Computation Analysis

Port the white-box to a smartcard and measure power consumption

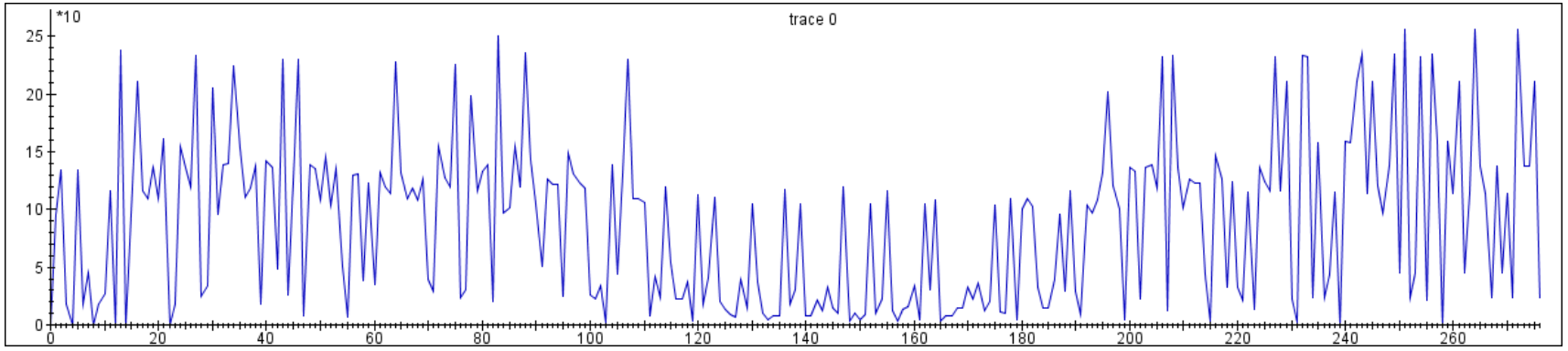
Differential Computation Analysis

~~Port the white-box to a smartcard and measure power consumption~~

Make pseudo power traces from our software execution traces

Those are lists of memory accesses / data / stack writes / …

E.g. build a trace of all 8-bit data reads:

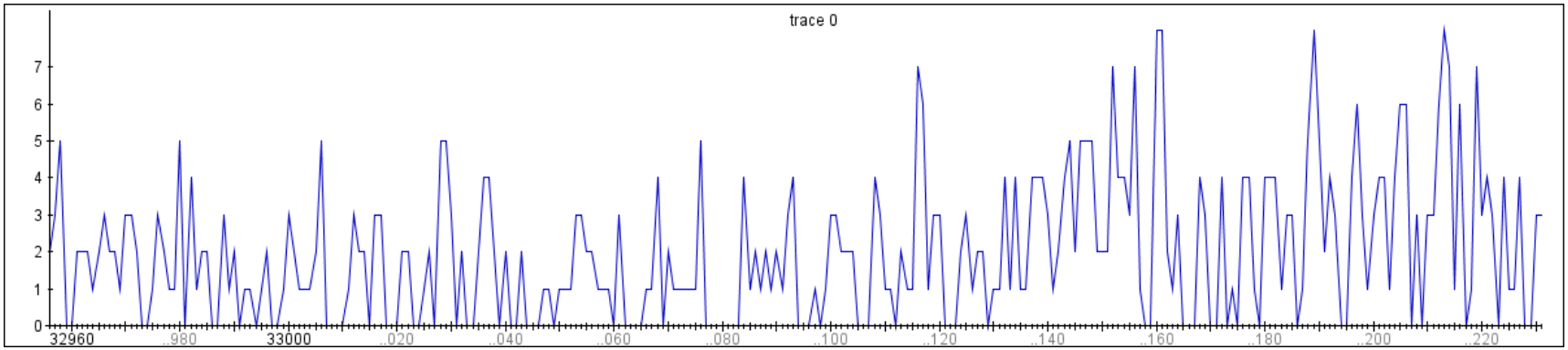


→ 256 possible discrete values

Differential Computation Analysis

256 possible discrete values but bit values quite dominated by the MSB

→ Build Hamming weight traces?



→ 8 possible discrete values

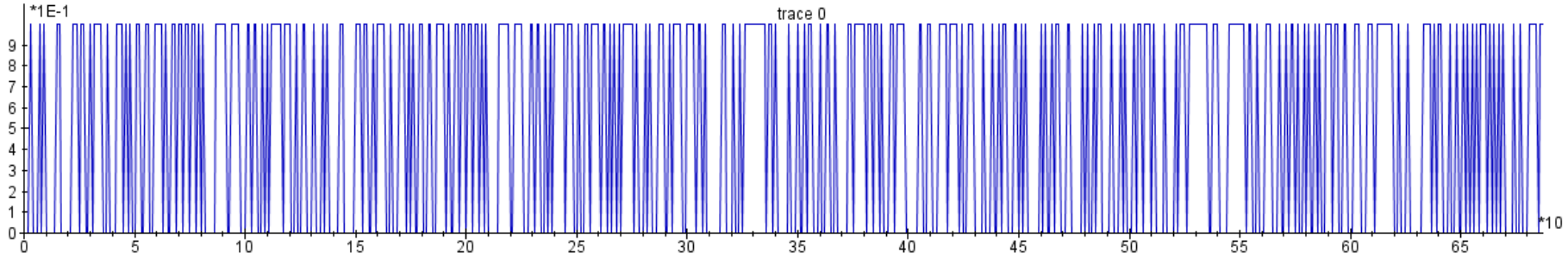
That works but we can do better…

Hamming weight was a hardware model for combined bit leaks

Differential Computation Analysis

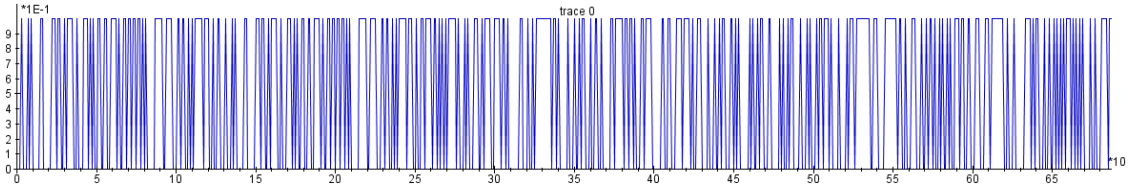
Each bit may leak individually
E.g. each address bit represent a different axis to split the look-up tables

→ Serialize bytes in a succession of bits



→ 2 possible discrete values: 0's and 1's

Differential Computation Analysis



Looks weird but works great!

As if we were probing individual bus lines:

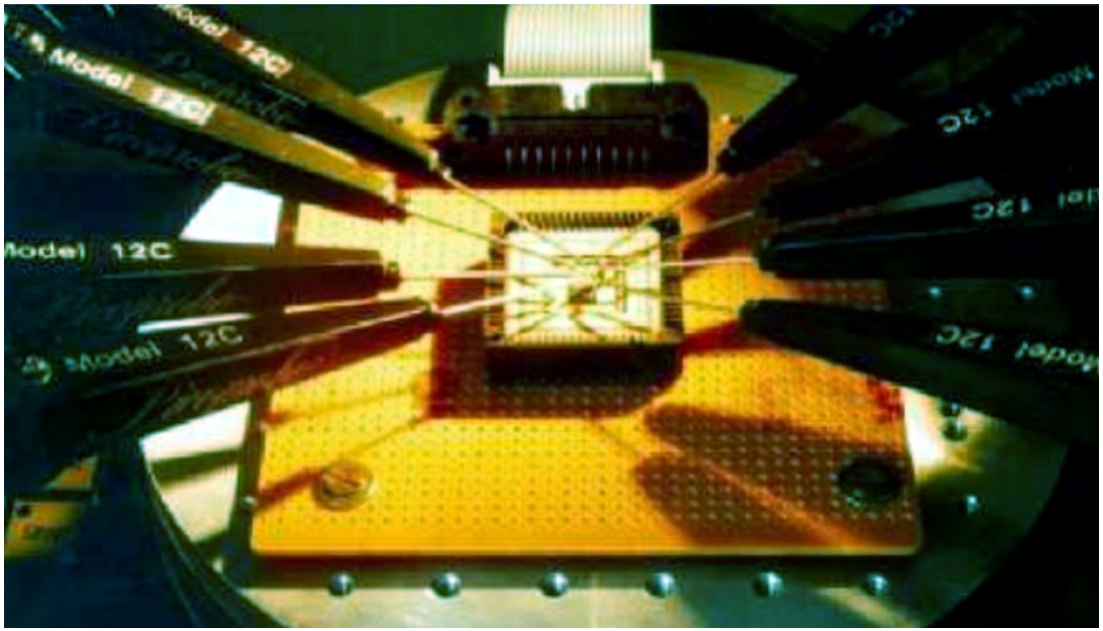


Image source: Erik Poll

Next step

Feed traces in your favorite DPA tool

- Riscure Inspector SCA software
- ChipWhisperer opensource software
- Matlab...

Tips

What to trace?

- Stack writes (reads are redundant leakages)
- Data reads (usually only bytes, not larger reads)
- Accessed addresses (usually just the lowest byte)

Combine them all if you wish

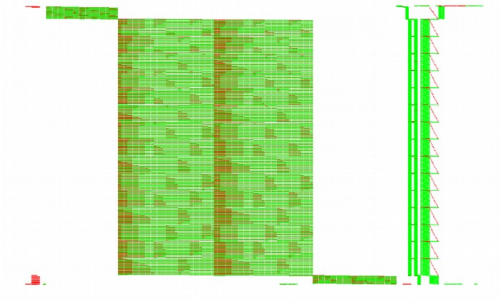
You'll need corresponding plaintexts and/or ciphertexts

- May require binary instrumentation, so far regular I/O or faking kbd/mouse and reading screen did the job

Large white-box? Minimize amount of traced information

- Trace only first (or last) round
- Standard deviation analysis to compress the trace

Wyseur challenge



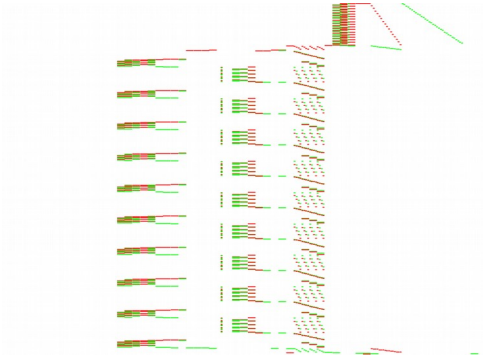
by Brecht Wyseur, 2007

DES implementation based on Chow “plus some personal improvements”

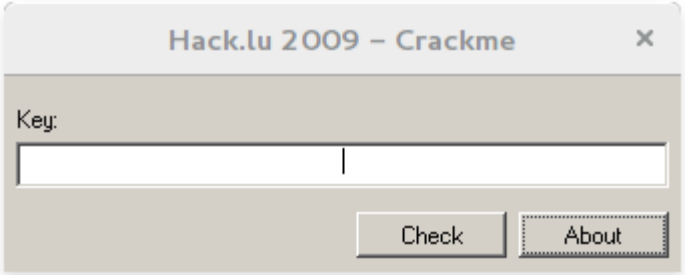
Downloading Linux binary…

1h and 65 traces later (of a full binary execution), key got broken!

Hack.lu 2009 challenge



Windows *crackme* by Jean-Baptiste Bédrune
AES implementation based on Chow



I was lazy porting our instrumentation under Windows

→ Wine/Linux + xdotool (kbd+mouse emulation)

16 traces (of a full bin. exec.) to break the key

No surprise, it's a CTF challenge, no internal encodings

SSTIC 2012 challenge

Python white-box by Axel Tillequin
DES implementation in a marshalled object

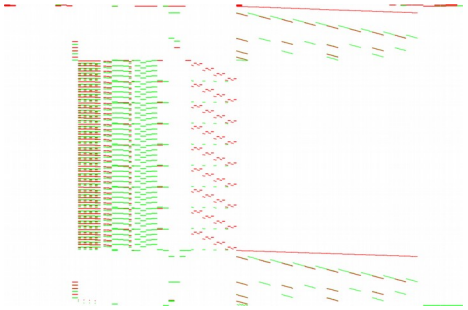
Tracing Python interpreter with PIN is really really not a good idea

→ Instrumenting “Bits” helper class to record all new instances

Again, 16 traces

Again, no internal encodings

Karroumi



Latest academic attempt to “fix” Chow (2011)

Idea is to interleave Dual Ciphers, i.e. isomorphic AES ciphers:
you can move from one to the other one via invertible transformations of the key, plaintext and ciphertext

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

It got academically broken too

We made our own binary challenge… and broke it with our DCA

2000 traces, 500 traces after some tuning

Some proprietary white-boxes

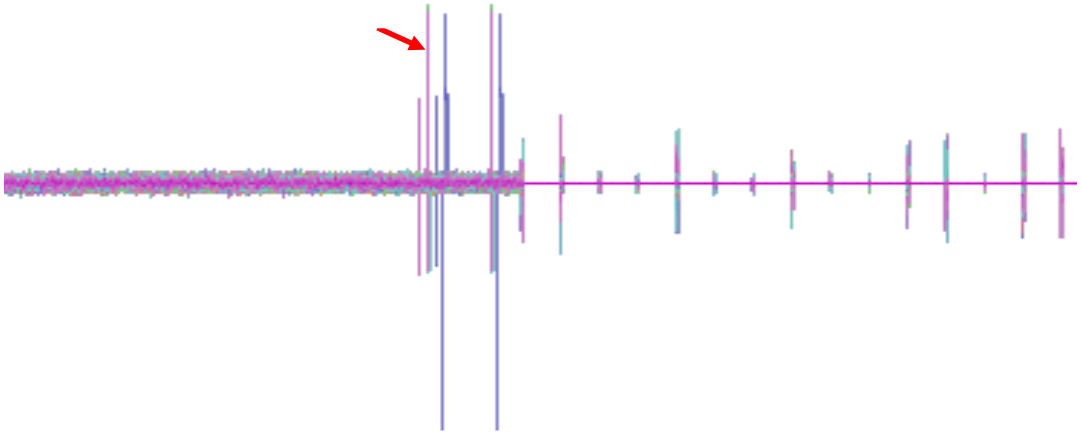
DES & AES

Broken in 200 to 2500 traces

Sorry, can't tell you much more ;)

Back to White-Box design

Known key analysis



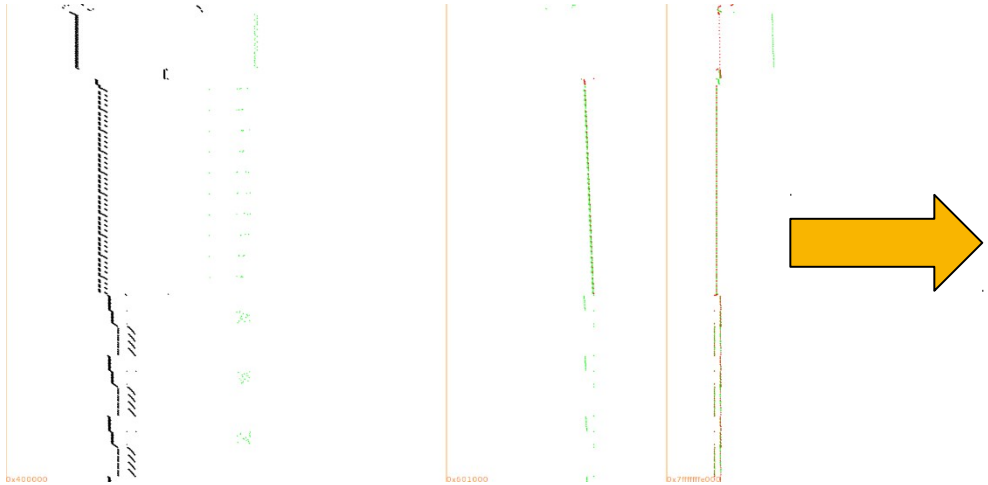
1) Identify first leaking samples (the original source)
2) Find the corresponding instruction
3) Find the corresponding source code line

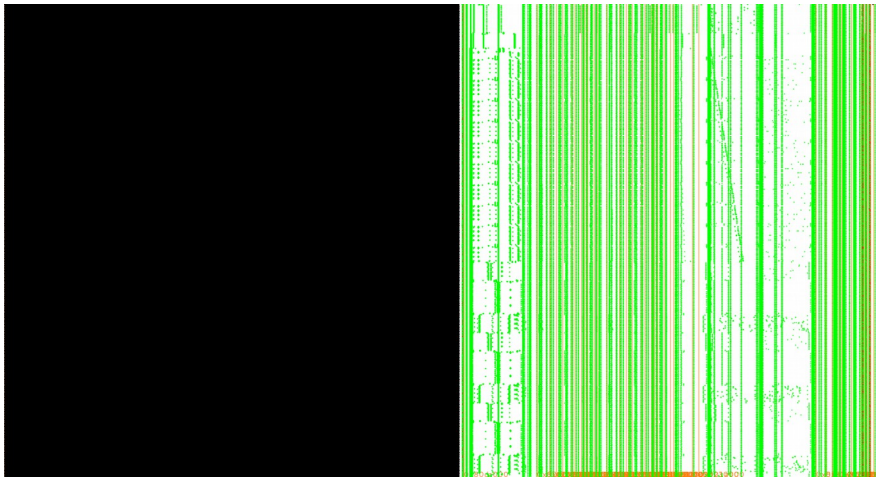
Works also on obfuscated VMs: M/o/Vfuscator2

```
mov eax,0x0
mov ax,[0x80a0451]
mov byte [eax+0x80e17bc],0x0
mov al,[eax+0x80e17bc]
mov [0x80a0451],al
mov eax,[0x80a0556]
mov edx,[eax+0x80a058e]
mov eax,[0x80a0451]
mov eax,[eax+edx]
```

Single Instruction (MOV) C compiler, because MOV is Turing-complete!

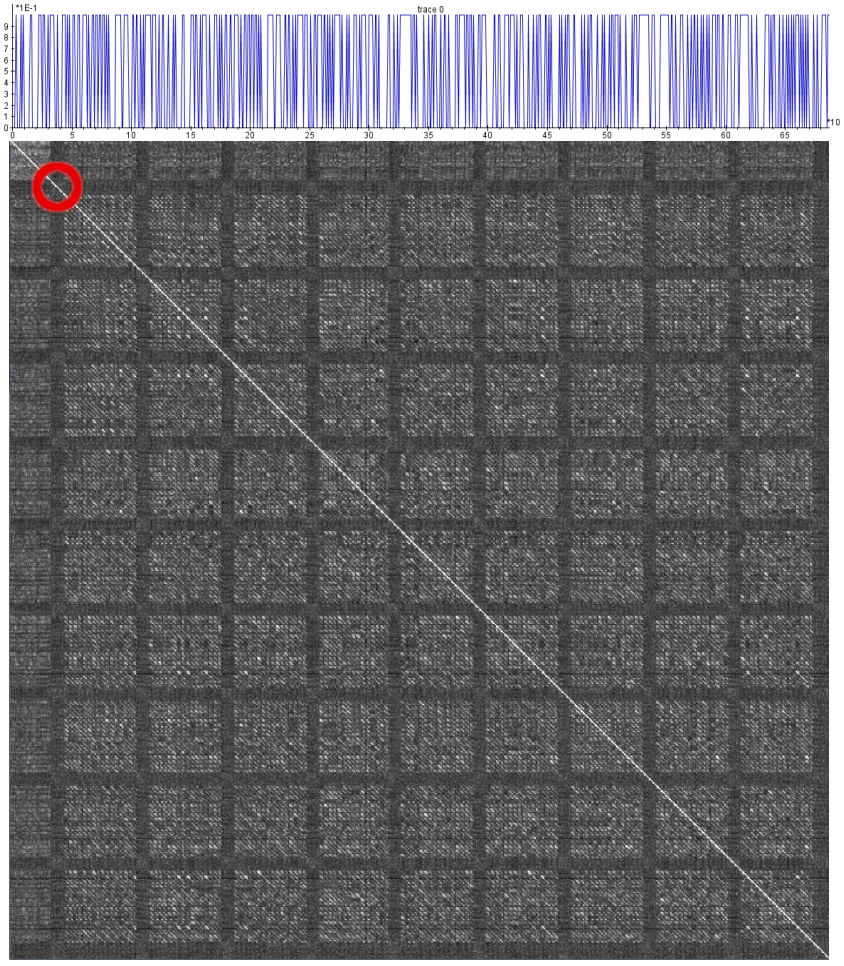
Applied on a standard AES implementation





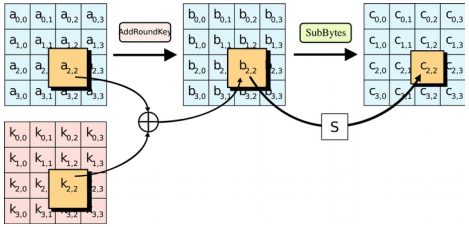
M/o/Vfuscator2 on AES

Auto-correlation reveals structure:



Huge traces, compressed by looking at standard deviation
4Mb -> 6.6kb

First round Sbox output



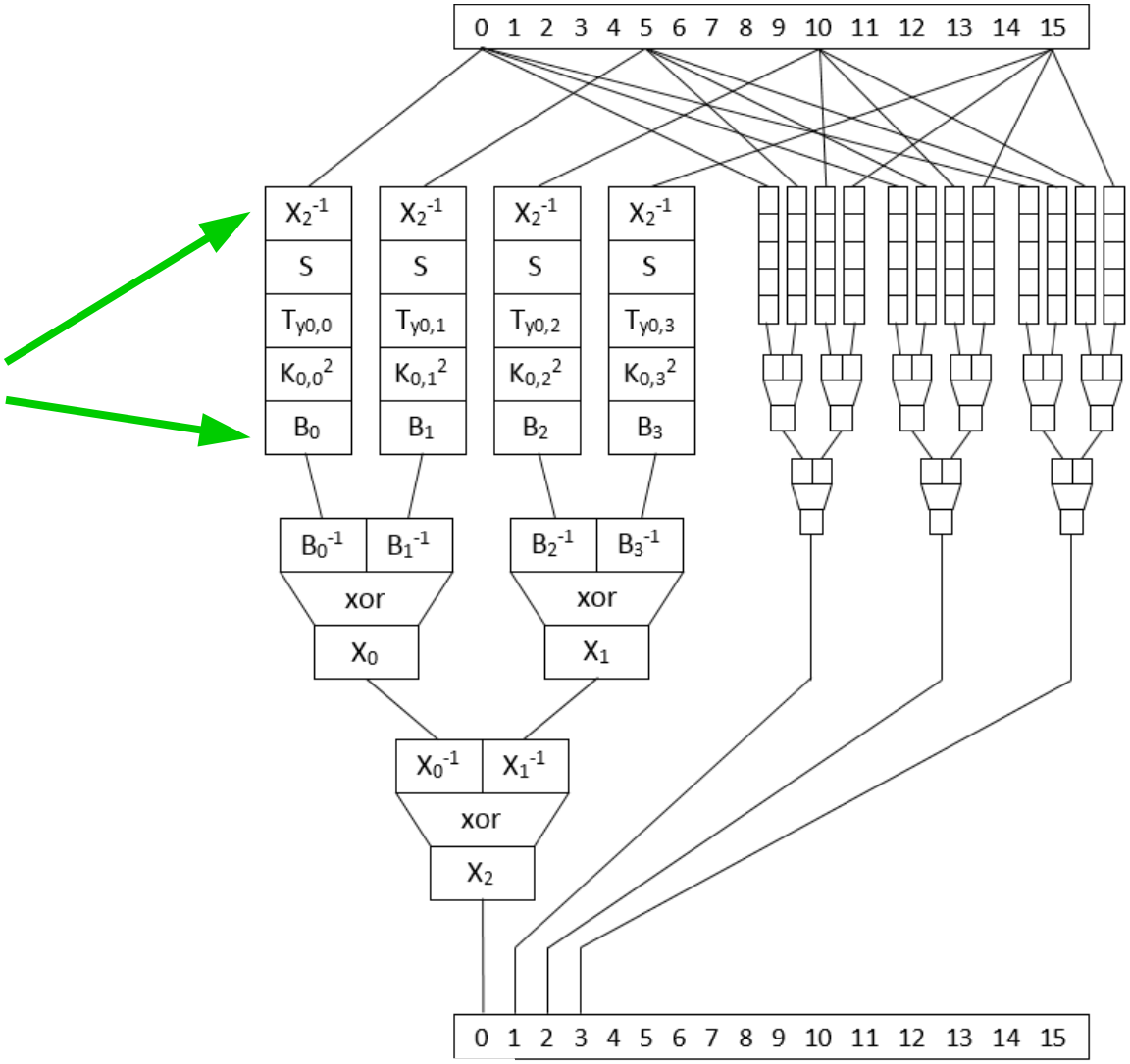
20 – 30 traces

http://wiki.yobi.be/wiki/MoVfuscator_Writeup

Can DCA fail?

Yes!

Wide intermediate
non-linear encodings (8x8)
blind the SBox non-linearity



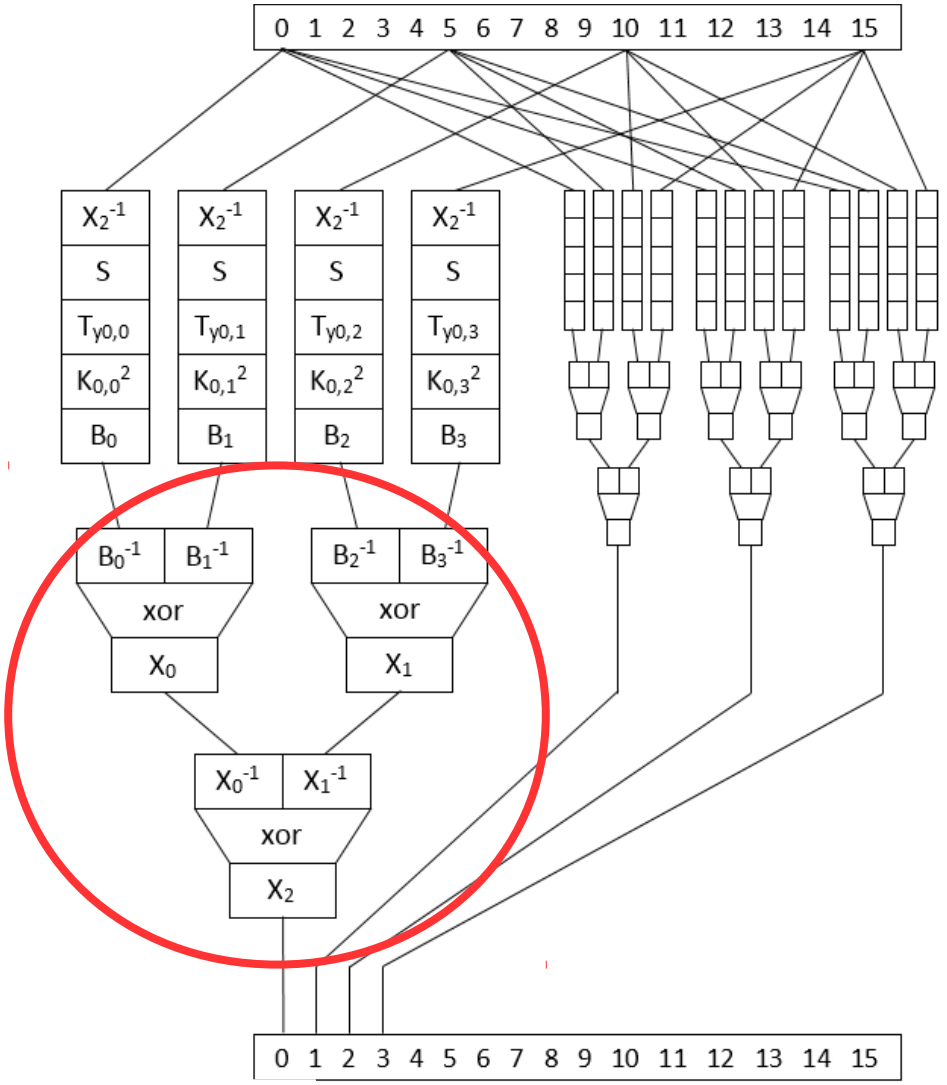
Can DCA fail?

Yes!

Wide intermediate
non-linear encodings (8x8)
blind the SBox non-linearity

But manipulating wide-encoded
data require very large tables!

→ Trend to reuse those tables
→ reuse encodings
→ other types of attack



cf my write-ups of
NoSuchCon 2013 and CHES 2015
http://wiki.yobi.be/wiki/CHES2015_Writeup

Other countermeasures?

Typical hardware countermeasures are based on runtime randomness

- But here, no trustworthy TRNG available

Runtime random delays?

- Trace also instructions and use them to realign memory accesses

Building proper white-box technology is a delicate matter…
Forget about “perfect” security, but if cost of an attack is larger than the benefit for the attacker, you achieved your goal.

Oops, it seems our cheap attack raised the bar...

Other grey box attacks within reach: Higher order DPA, CPA, DFA,…

Take also care of code lifting, inversing f(),...

“Now this is not the end.
It's not even the beginning of the end.
But it is, perhaps, the end of the beginning.”

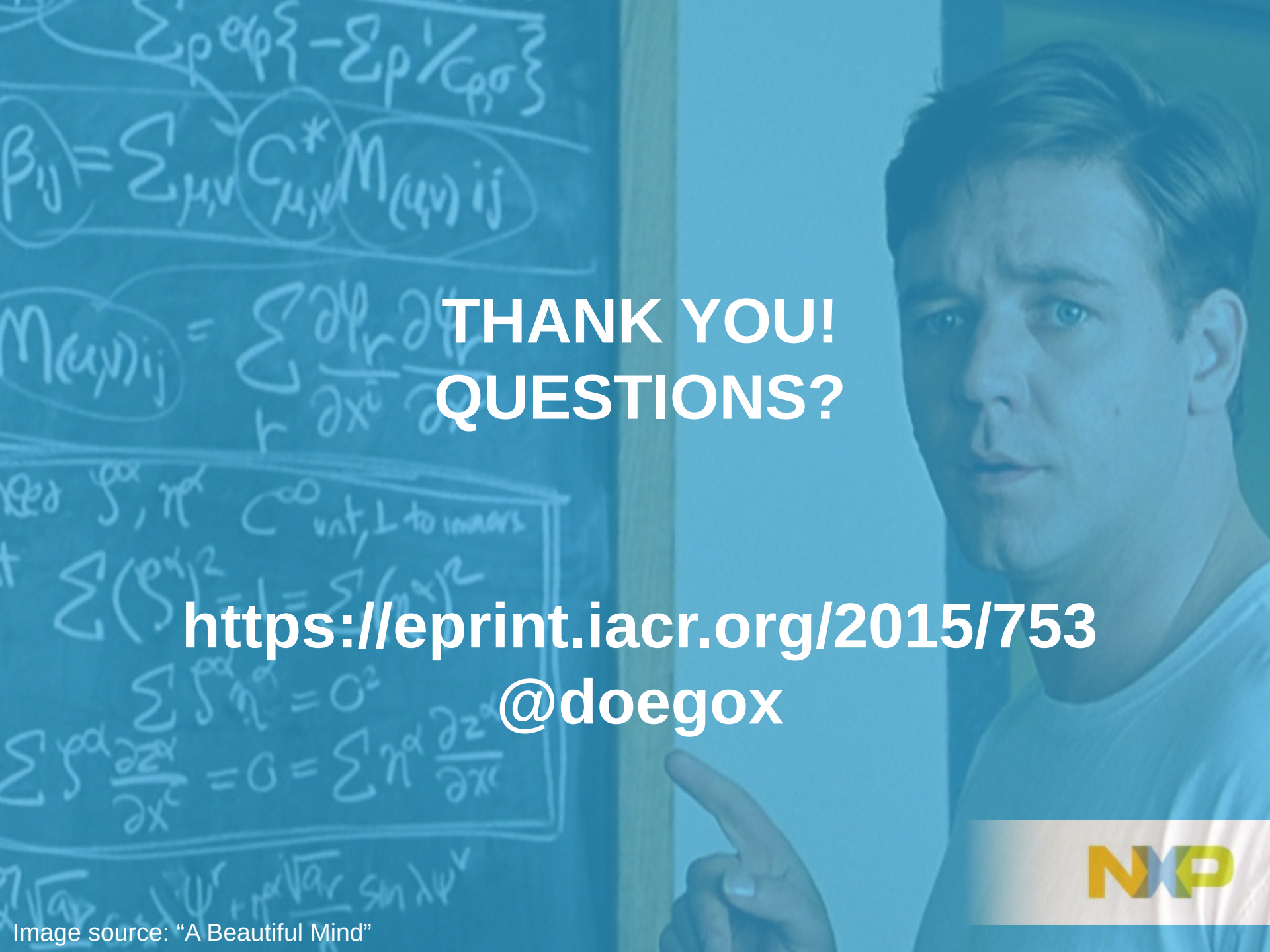


Image source: “A Beautiful Mind”

SECURE CONNECTIONS
FOR A SMARTER WORLD