



Cédric DELAUNAY, ESIEA, (C + V) ^ O Lab, France

Alexander Alexandrovich ISTOMIN, “NPP “GAMMA”, Russia

Eric FILIOL, ESIEA, (C + V) ^ O Lab, Laval, France

Делоне Седрик, ESIEA, (C+V)^O Lab, Франция

Истомин Александр Александрович, ФГУП «НПП «ГАММА»,
Россия

Фийол Эрик, ESIEA, (C+V)^O Lab, Лаваль, Франция

XXI - RUSCRYPTO 2019

20.03.2019

COOPERATION / КООПЕРАЦИЯ

ESIEA

(C + V) ^ O Lab.

Laboratory of Operational
Cryptology and Virology,
France



ESIEA

(C + V) ^ O Lab.

Лаборатория оперативной
Криптологии и Вирусологии,
Франция



FSUE “SIE “GAMMA”

Federal State Unitary Enterprise
Scientific and Industrial Enterprise
“GAMMA”,
Russian Federation



ФГУП «НПП «ГАММА»

Федеральное Государственное
Унитарное Предприятие
«Научно-производственное предприятие
«ГАММА»,
Россия



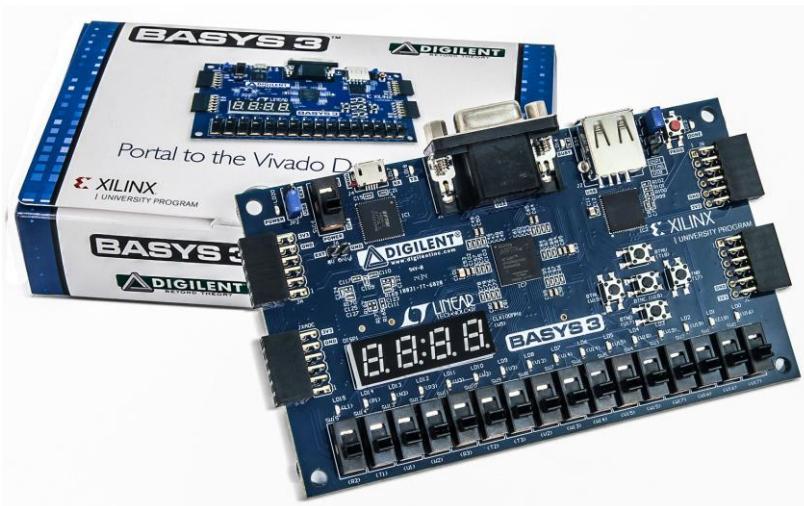
Presentation Outline



1. “Kuznyechik” et al.
2. Securing the Implementation
3. CPA and DPA – Attacks and Resistance
4. Results and Conclusions
5. Where to next ?

Initial context and stakes

- Study the symmetric **GOST R 34.12-2015** aka “**Grasshopper**” Block cipher algorithm.
- Provide different implementations for **FPGA** and **Microcontrollers**
- Check the resistance of the algorithm and its implementations to **CPA** and **DPA** attacks.



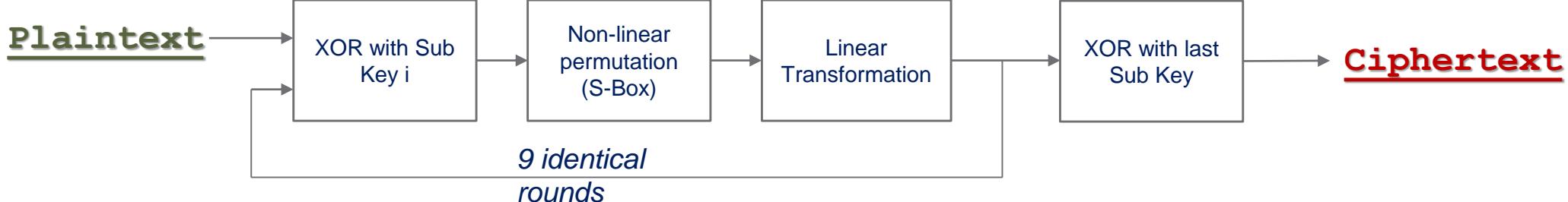
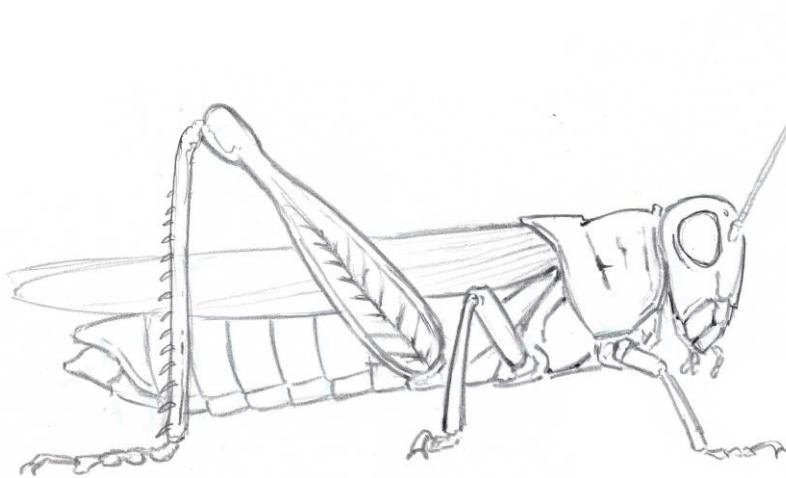
« Kuznyechik » et al.

About Grasshoppers and Waffles



“Kuznyechik” (GOST R 34.12-2015)*

- **Russian Standard**, adopted 01/2016
- **Symmetric Block Cipher** algorithm
- **128 bits** Block size
- **256 bits** Key length
- **10 sub-keys of 128 bits**
- Substitution-Permutation Network
- **Feistel Network** for Key Schedule
- **10 Rounds / 3 functions per round**



$$C = X[k_{10}]LSX[k_9]LSX[k_8]\dots LSX[k_2]LSX[k_1](P)$$

(*): for all those in hibernation for the last 5 years



“Kuznyechik” (GOST R 34.12-2015)*

- Linear permutation L :

(γ)

where: $L(x) = R^{16}(x)$

$$\begin{aligned} R(x) &= R(x_{15} || \dots || x_0) & x &= x_{15} || \dots || x_0 \\ &= l(x_{15}, \dots, x_0) || x_{15} || x_1 \end{aligned}$$

and

where :
$$\begin{aligned} l(x_{15}, \dots, x_0) &= 148x_{15} + 32x_{14} + 133x_{13} + 16x_{12} \\ &\quad + 194x_{11} + 192x_{10} + 1x_9 + 251x_8 \\ &\quad + 1x_7 + 192x_6 + 194x_5 + 16x_4 \\ &\quad + 133x_3 + 32x_2 + 148x_1 + 1x_0 \end{aligned}$$

where all multiplications are performed in $GF(2)[X]/p(x)$, with

$$p(x) = x^8 + x^7 + x^6 + x + 1 \in GF(2)[X]$$

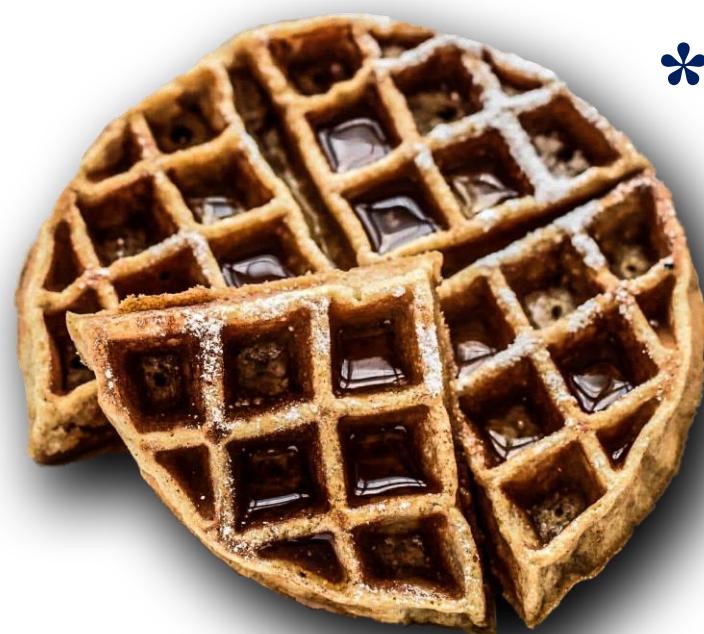
(*): for all those in hibernation for the last 5 years



The AES Cipher as brief reminder



- **NIST Standard**, adopted 2001
- **Symmetric Block Cipher** algorithm
- **128 bits** Block size
- **128, 192 or 256 bits** Key length
- **14 sub-keys of 128 bits**
- Substitution-Permutation Network
- **14 Rounds / 4 functions** per round
(for 256 bit key length)

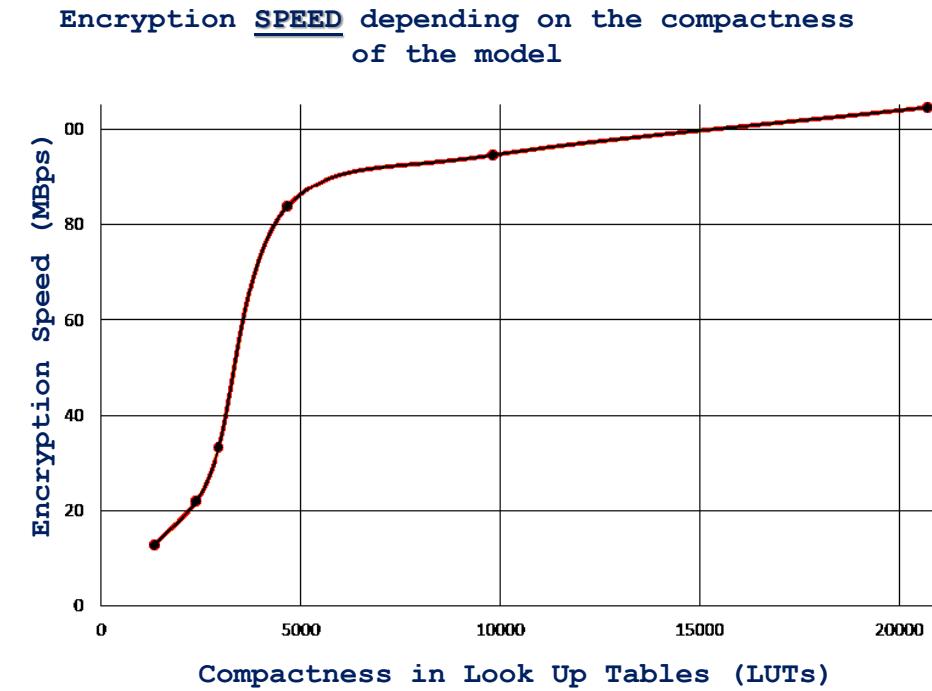
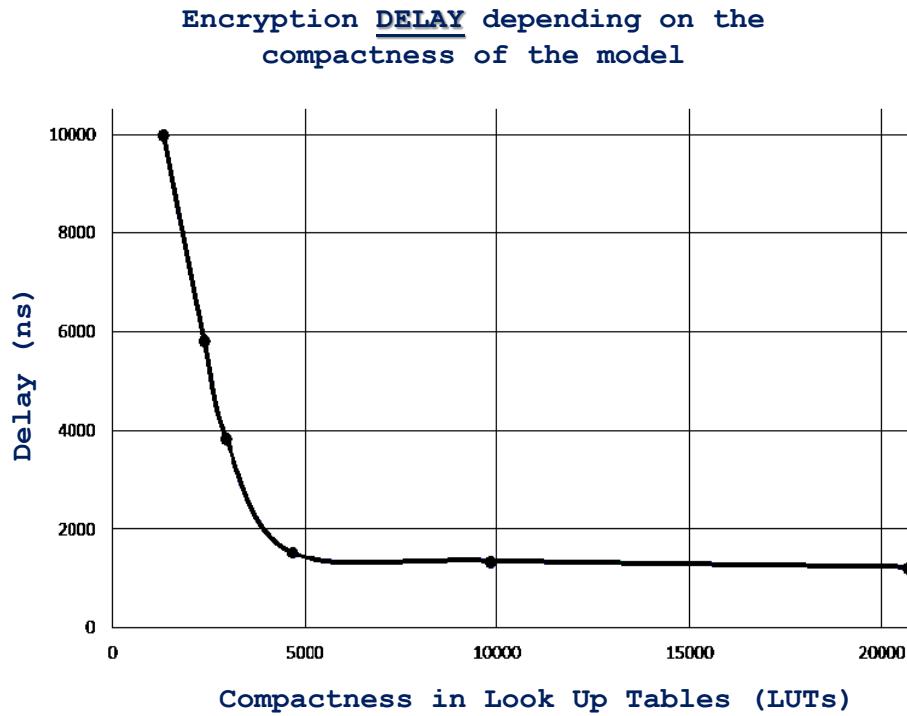


*

(*): Belgian Waffle – our avatar for the AES Cipher

Kuznyechik on FPGA

- Several variants where conceptualized and implemented
- Here the results of the first implementations:



Securing the Implementation

Вежливые Зелёные Кузнечики

« It's not easy being green »

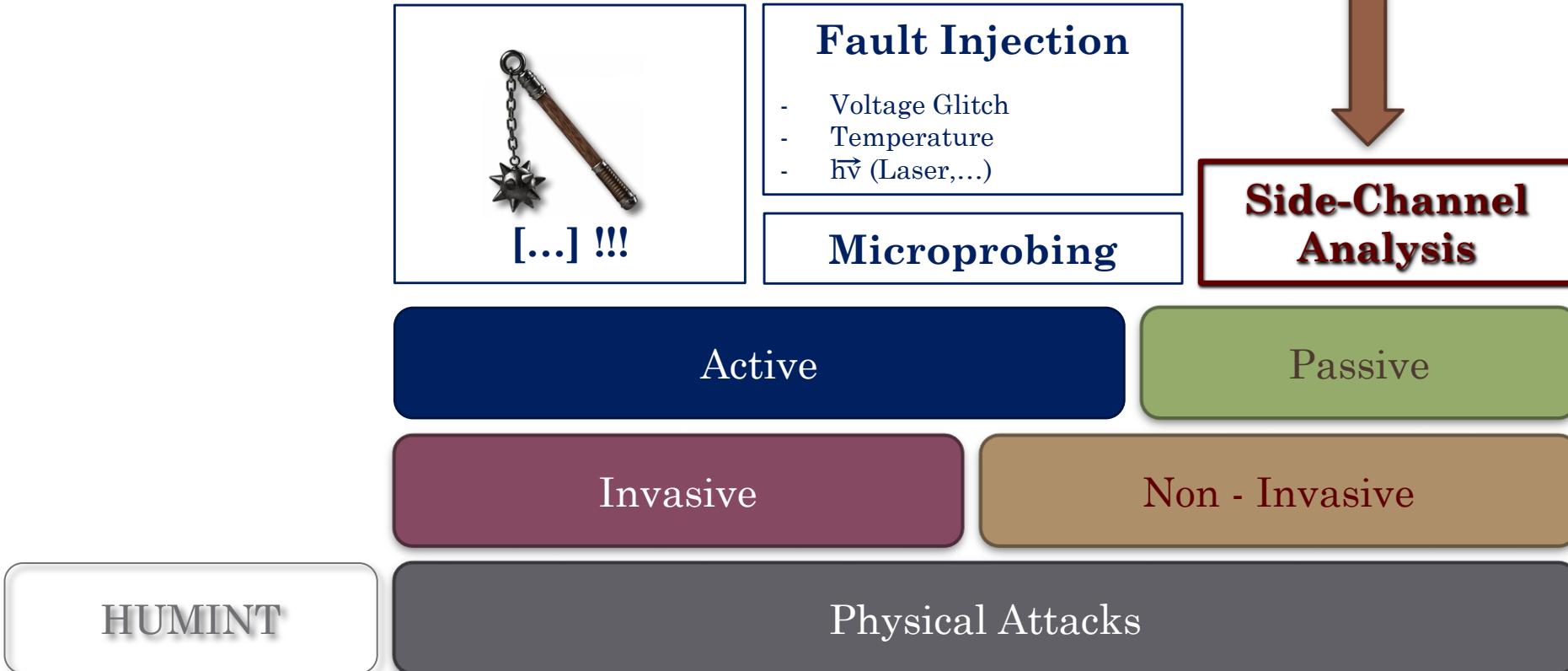


Threat Matrix

- What do we face ?

SIGINT / MEASINT

→ **Power Consumption Analysis - Attack**



Securing the Implementation

How can we protect ourselves against these attacks?



Enacting
HARDWARE and SOFTWARE
Countermeasures

- Random Delay Insertion
- Noise Generator
- Shuffling

• Masking

• Dual-Rail with Precharge Logic



Chosen Solution !

Securing the Implementation

How does the masked “Kuznyechik”-Implementation work ?

- Random Mask Generation
- XOR between the Plaintext and the mask before encryption (or decryption)
- Unmasking at the end of the encryption (or decryption)



$$\begin{aligned} LS_{m_1}X[k](x + m) &= LS_{m_1}(X[k](x) + m) \\ &= L(S(X[k](x)) + m)) \\ &= LSX[k](x) + L(m) \end{aligned}$$

$$X[k_{10}]LS_{m_9}X[k_9]...LS_{m_1}X[k_1](x + m) = X[k_{10}]LSX[k_9]...LSX[k_1](x) + L^9(m)$$

Securing the Implementation

- Masking AES-256 works the same way

$$\begin{aligned} A_{rk} M_c S_r S'_b(x + m) &= A_{rk} M_c S_r(S_b(x) + m) \\ &= A_{rk} M_c (S_r S_b(x) + S_r(m)) \\ &= A_{rk} (M_c S_r S_b(x) + M_c S_r(m)) \\ &= A_{rk} M_c S_r S_b(x) + M_c S_r(m) \end{aligned}$$

A_{rk} : *AddRoundKey*

M_c : *MixColumns*

S_r : *ShiftRows*

S_b : *SubBytes*



Securing the Implementation

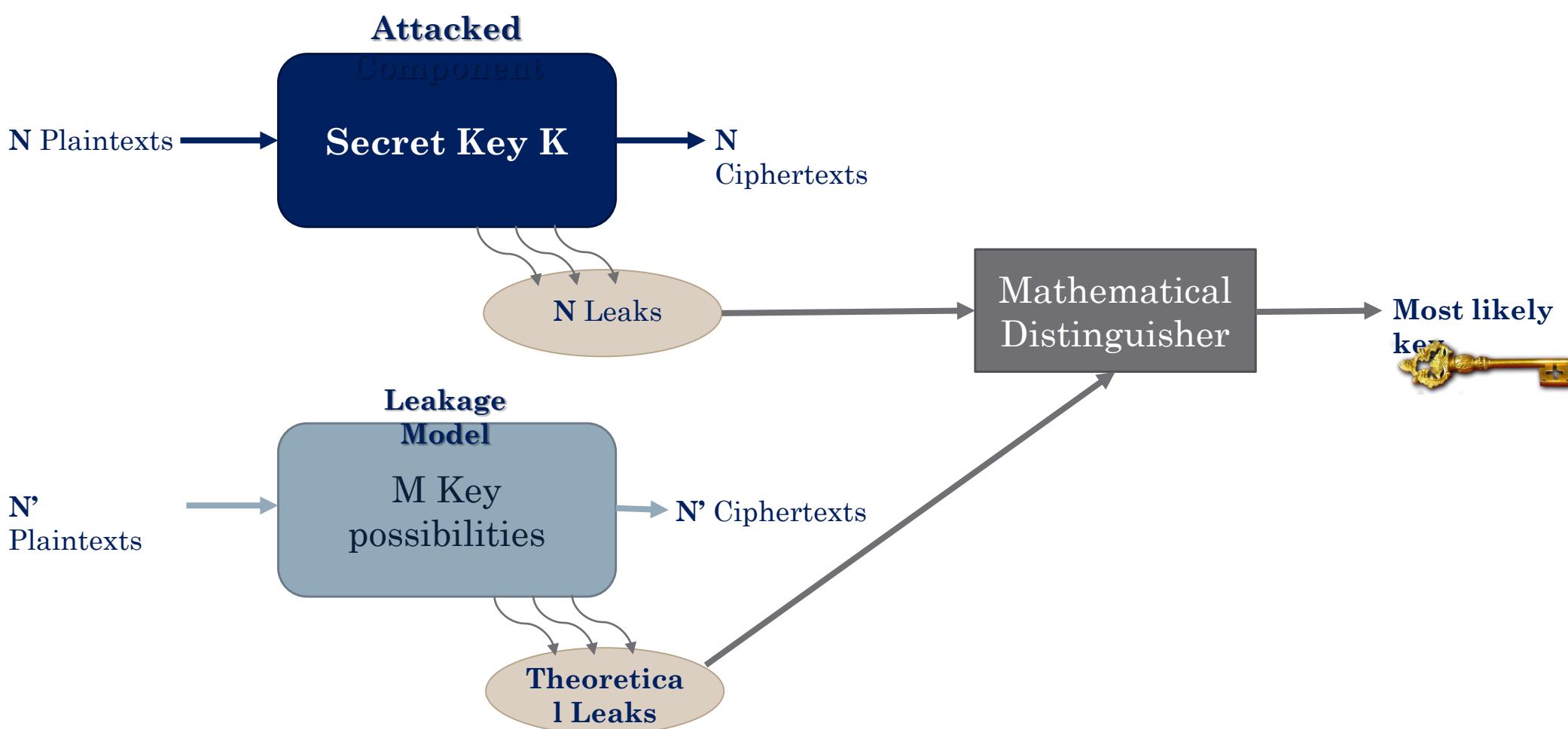
- **Comparison between implementations:**
(Reference Platform XILINX Artix7 (Speedgrade -1))

Implementation:	“Kuznyechik” Standard	“Kuznyechik” Masked	AES-256 Standard
LUTs:	8810	10106	4846
Latches:	2167	2697	4540
Maximum Frequency:	28.5 MHz	28.5 MHz	28.5 MHz
Delay:	1526.2 ns	1596.2 ns	1920.0 ns
Encryption rate	83.9 Mbps	80.8 Mbps	66.7 Mbps



Preview of the CPA Theory

- Leakage Model



CPA and DPA Attacks and Resistance

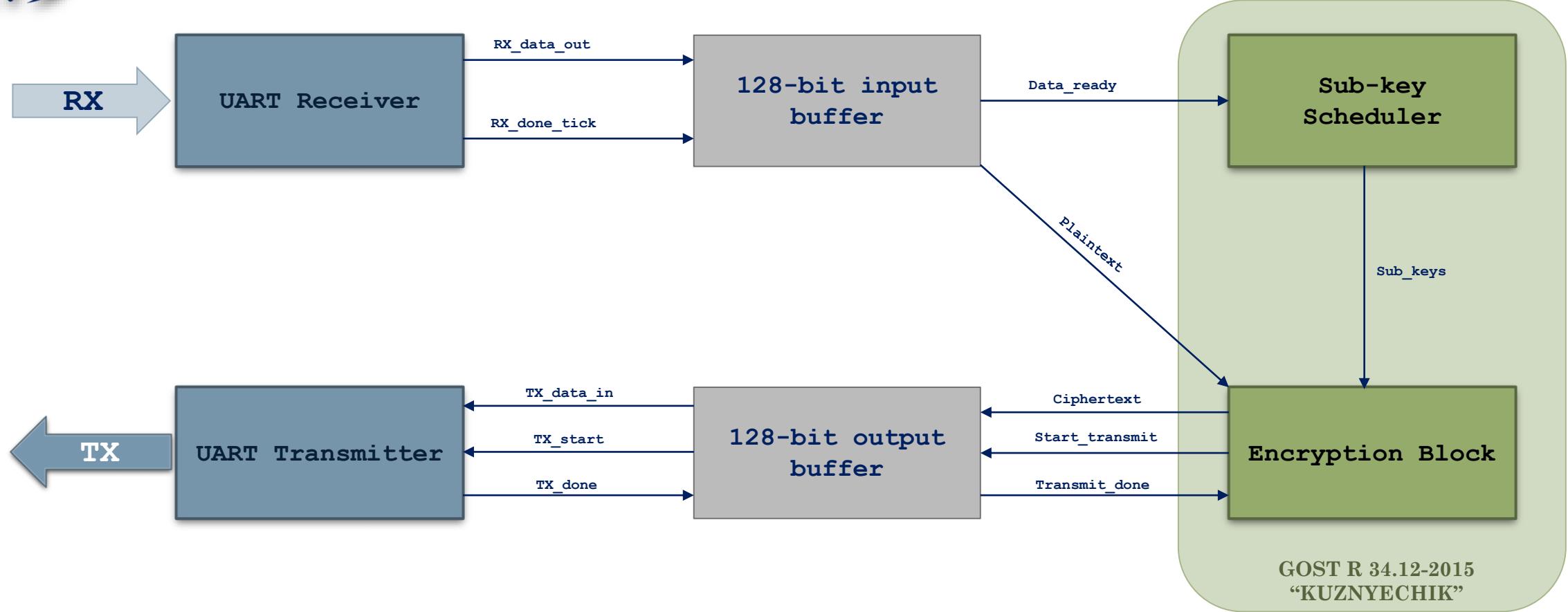
...after that, the NSA, the CIA,...



DPA Setup in FPGA

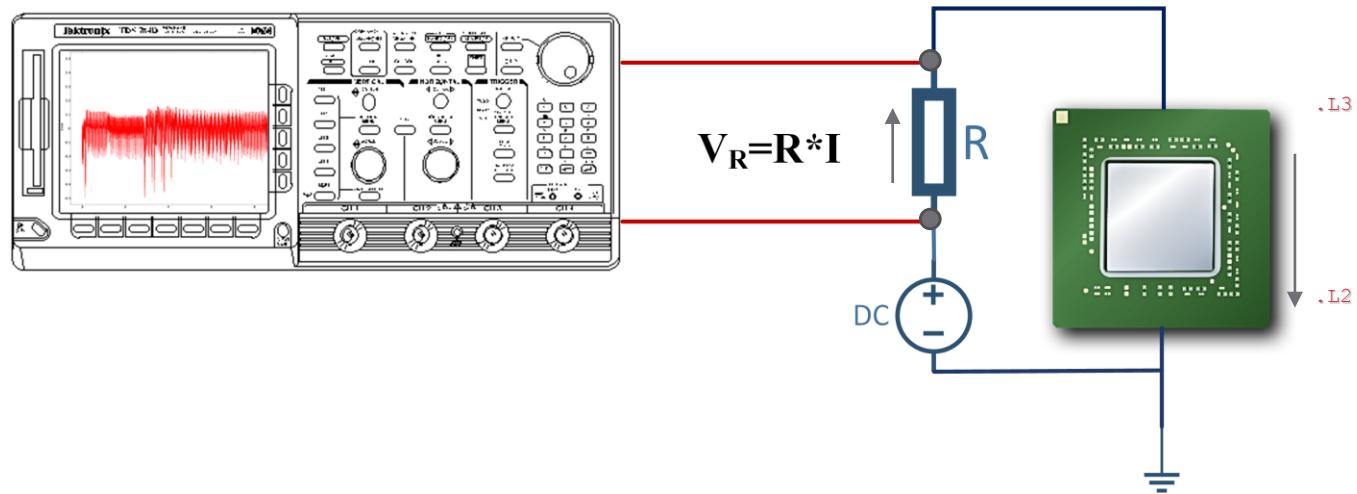
- XILINX Artix 7 XC7A35T & XC7A100T

(γ)



Power Measurement

- Direct measurement at the source
- Usually through a shunt resistor in the Power supply line
 - The voltage measured at the resistor is linear proportional to the current through the circuit



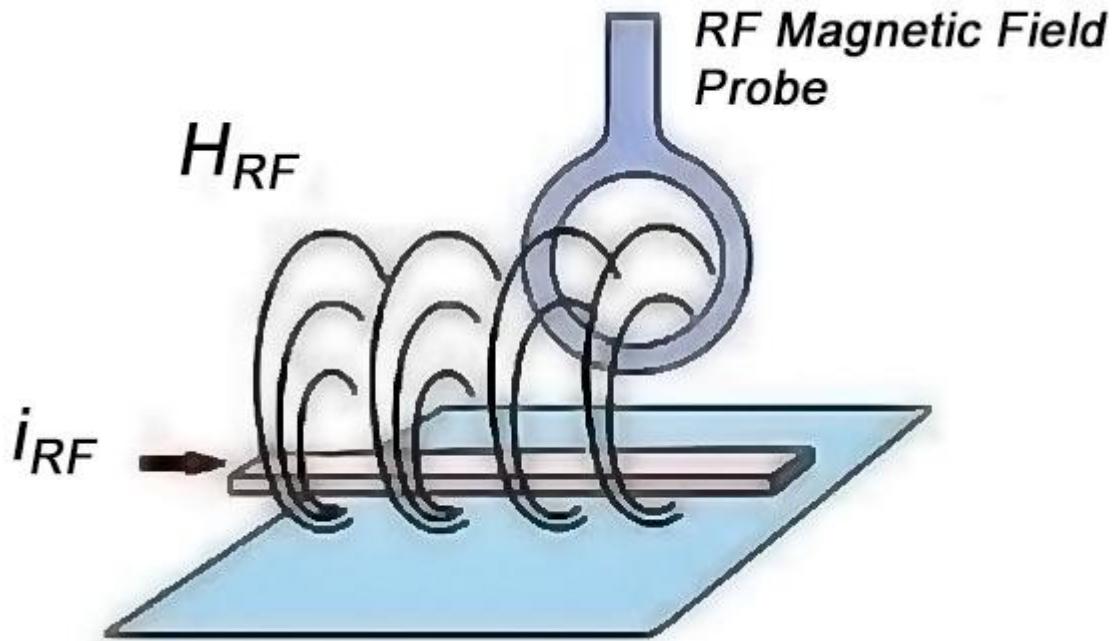
```
[...]
add fp, sp, #0
sub sp, sp, #20
str r0, [fp, #-16]
mov r3, #0
str r3, [fp, #-8]
b .L2

ldr r3, [fp, #-8]
add r3, r3, #1
str r3, [fp, #-8]
ldr r3, [fp, #-16]
mov r3, r3, asr #1
str r3, [fp, #-16]

ldr r3, [fp, #-16]
cmp r3, #0
bgt .L3
ldr r3, [fp, #-8]
mov r0, r3
add sp, fp, #0
ldmfd sp!, {fp}
bx lr
[...]
```

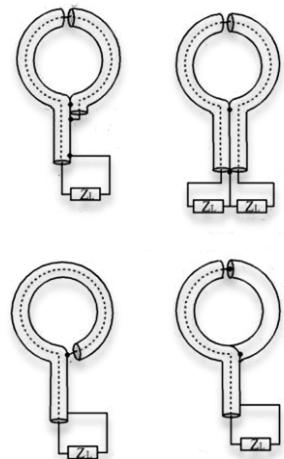
EM-Field Measurement

- Indirect measurement
- Current through conductor = magnetic field
- Measured through an **H-Probe**



H-Probe Constructions

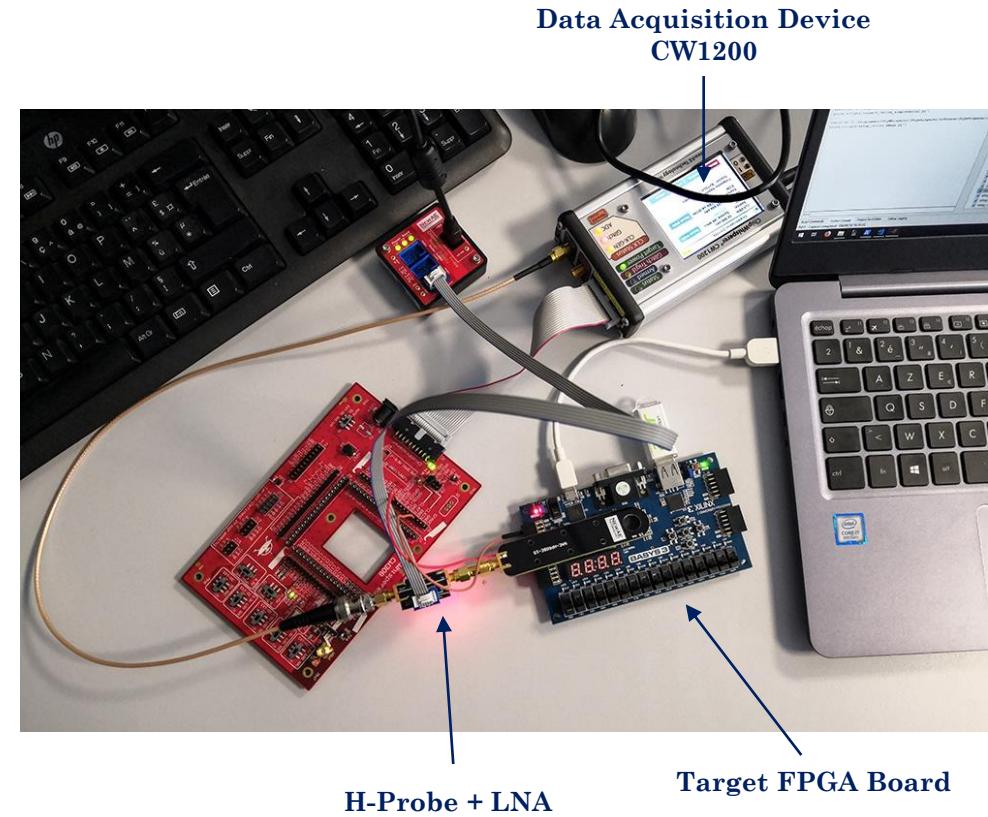
..



Hardware Setup

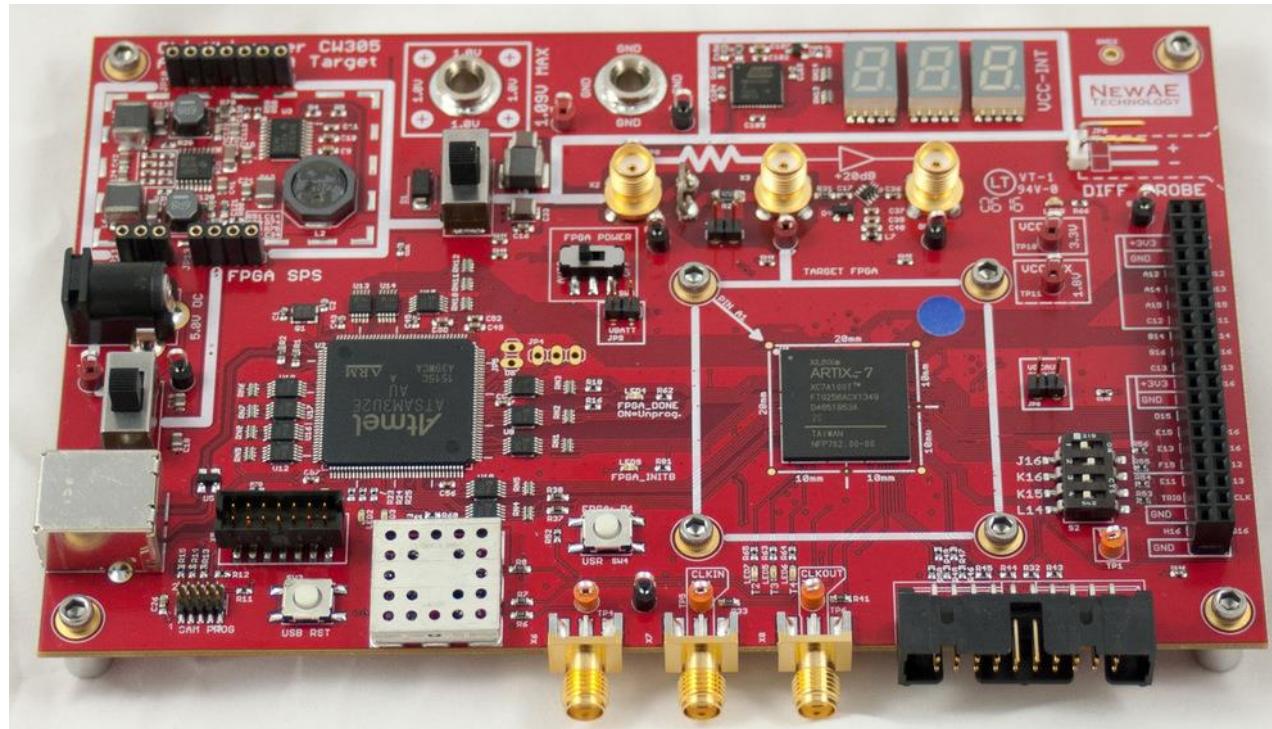
- Data Acquisition Equipment (France)
 - Power measurement through shunt on Microcontroller
 - Power measurement through H-Probe on Artix Devboard

ChipWhisperer CW1200



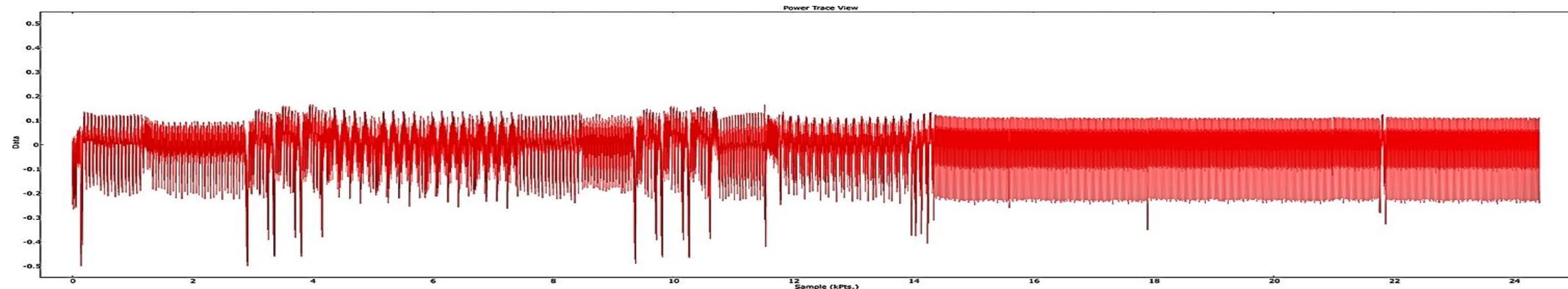
Hardware Setup

- Data Acquisition Equipment – Russia
- Two Labs with ChipWhisperer CW1173 and special FPGA Target Board
 - Power measurement through shunt on Microcontroller
 - Power measurement through shunt on low noise FPGA Board XC7A100T

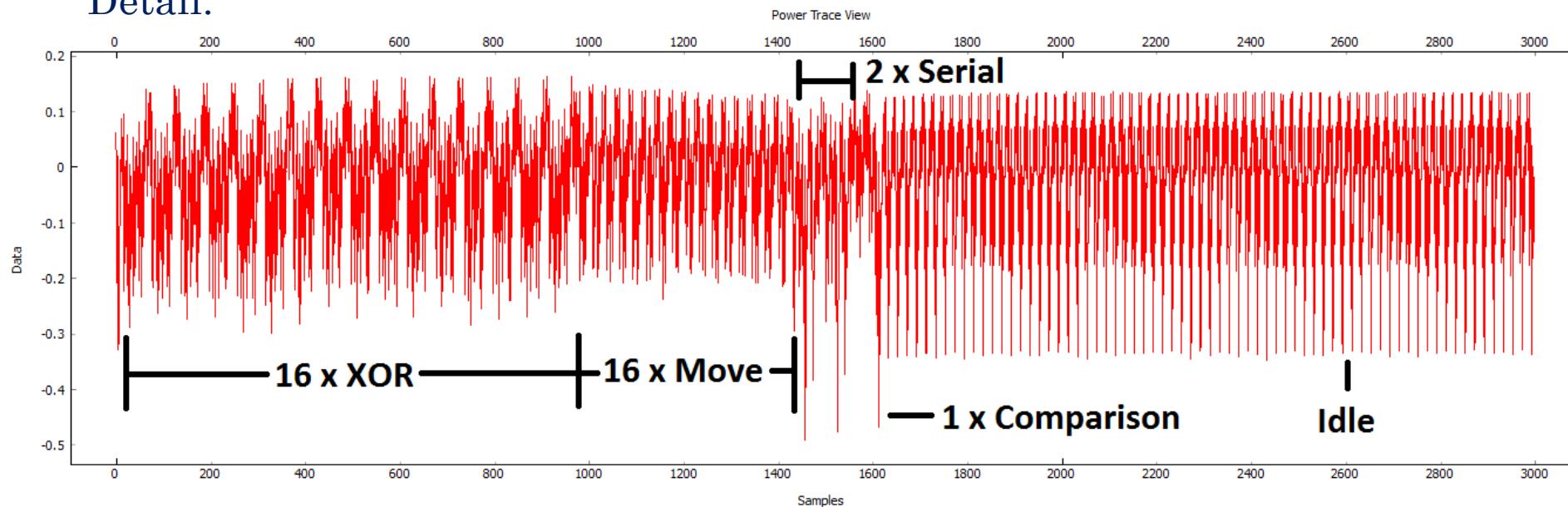


Standard CPA with AES (trivial)

Complete AES Signature (1 trace):



Detail:





Standard CPA with AES (trivial)

- An algorithm can be mathematically strong but very weak in hardware implementations without countermeasures

Attack Settings

Parameter	Value
Attack	None

Results Table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
PGE	0	2B 0.7677	7E 0.8317	15 0.8839	16 0.8012	28 0.7966	AE 0.8188	D2 0.8278	A6 0.8053	AB 0.7662	F7 0.8589	15 0.8207	88 0.8639	09 0.7543	CF 0.8361	4F 0.8707	3C 0.9210
0	00	40	40	10	20	00	00	00	00	00	00	00	00	00	00	00	
1	0.6249	0.6114	0.6465	0.6345	0.6561	0.6612	0.6385	0.6246	0.6099	0.6486	0.6349	0.6031	0.6157	0.6623	0.6242	0.6551	
2	F4 0.5968	5E 0.6040	FF 0.6334	56 0.6305	01 0.6075	A6 0.5944	14 0.6008	45 0.6088	E5 0.5898	0A 0.6411	76 0.6251	AB 0.5968	3B 0.6067	CE 0.6565	73 0.6117	21 0.6339	
3	2A 0.5943	17 0.5960	94 0.6040	86 0.6055	05 0.5994	10 0.5928	24 0.5894	AE 0.6050	0.5852	0.5976	0.6007	0.5918	0.6020	0.6468	0.5950	0.6303	
4	68 0.5817	6C 0.5883	47 0.5870	AD 0.6006	40 0.5896	18 0.5858	A4 0.5809	C2 0.5889	CA 0.5691	E6 0.5804	6A 0.5925	BD 0.5844	2F 0.5931	53 0.6288	74 0.5917	86 0.6135	
5	C0 0.5812	24 0.5869	4F 0.5816	3A 0.5790	85 0.5861	AF 0.5836	EC 0.5788	90 0.5888	4C 0.5646	C1 0.5794	B6 0.5881	31 0.5833	08 0.5923	85 0.6248	4D 0.5828	EF 0.6050	
6	F7 0.5769	F5 0.5799	86 0.5736	98 0.5778	9C 0.5781	68 0.5803	7E 0.5692	4C 0.5816	77 0.5633	6E 0.5725	23 0.5860	1F 0.5826	1E 0.5722	4E 0.5918	EB 0.5820	7C 0.5898	
7	22 0.5750	6E 0.5797	93 0.5626	0C 0.5766	3A 0.5773	DA 0.5744	A1 0.5574	15 0.5786	12 0.5606	A6 0.5667	70 0.5820	D6 0.5794	B5 0.5695	D5 0.5866	57 0.5786	19 0.5898	
8	14 0.5711	26 0.5784	ED 0.5610	C5 0.5736	7C 0.5757	43 0.5648	34 0.5528	7A 0.5682	9A 0.5605	02 0.5663	03 0.5807	51 0.5790	A2 0.5653	2C 0.5866	C2 0.5777	C3 0.5867	
9	7C 0.5682	7F 0.5757	F4 0.5568	31 0.5623	15 0.5706	19 0.5605	43 0.5497	97 0.5668	95 0.5884	0E 0.5608	99 0.5798	AC 0.5753	3A 0.5573	8B 0.5685	01 0.5777	E1 0.5849	
10	AF 0.5594	S0 0.5691	DA 0.5554	BE 0.5608	A0 0.5672	B6 0.5592	1F 0.5459	BD 0.5649	76 0.5560	25 0.5595	47 0.5752	F9 0.5741	EA 0.5558	BF 0.5628	49 0.5770	98 0.5848	
11	47 0.5549	5D 0.5668	32 0.5517	03 0.5587	76 0.5661	B1 0.5571	C6 0.5401	34 0.5627	93 0.5540	17 0.5532	D4 0.5662	97 0.5706	E8 0.5558	6D 0.5619	60 0.5735	96 0.5831	
12	9E 0.5542	4C 0.5649	CB 0.5504	C4 0.5578	FB 0.5660	CD 0.5560	B5 0.5385	9A 0.5603	E7 0.5504	CF 0.5516	95 0.5660	72 0.5693	1F 0.5557	65 0.5612	7A 0.5723	C2 0.5818	
13	1F 0.5536	74 0.5647	7C 0.5488	F5 0.5571	2E 0.5611	71 0.5555	5F 0.5385	46 0.5582	E6 0.5488	46 0.5505	9F 0.5626	D8 0.5568	F0 0.5533	96 0.5574	87 0.5720	A3 0.5791	
14	63 0.5497	E8 0.5617	78 0.5487	F7 0.5561	A4 0.5590	0B 0.5522	2D 0.5366	A7 0.5555	13 0.5488	96 0.5583	77 0.5630	E5 0.5548	8C 0.5571	A6 0.5702	BE 0.5768		
15	99 0.5489	1D 0.5572	9D 0.5482	BB 0.5537	50 0.5576	09 0.5469	FB 0.5319	DB 0.5530	D7 0.5464	50 0.5470	FB 0.5581	00 0.5626	CF 0.5540	29 0.5554	A5 0.5697	5E 0.5685	
16	D6 0.5485	D2 0.5564	03 0.5482	17 0.5519	8C 0.5535	30 0.5465	8D 0.5306	2D 0.5519	12 0.5462	4E 0.5446	4A 0.5564	56 0.5527	CA 0.5531	C0 0.5498	B3 0.5654	A8 0.5678	

Key found with only 50 traces

Python Console

```

lib\logging\__init__.py, line 845, in flush
    self.stream.flush()
IOError: [Errno 9] Bad file descriptor
Logged from file ProgressBar.py, line 63
Traceback (most recent call last):
  File "D:\Programmes\ChipWhisperer\WinPython-64bit-2.7.13.1Zero\python-2.7.13.amd64\lib\logging\__init__.py", line 885, in emit
    self.flush()
  File "D:\Programmes\ChipWhisperer\WinPython-64bit-2.7.13.1Zero\python-2.7.13.amd64\lib\logging\__init__.py", line 845, in flush
    self.stream.flush()
IOError: [Errno 9] Bad file descriptor
>>>

```

Project Text Editor

Script Commands

Debug Logging

ChipWhisperer Project File System Recent

Name

- __init__.py
- attack_cpa.py
- attack_cpa_decryptaes.py
- attack_des.py

Script Preview (Read Only)

```

'''CPA attack script.

Assumes that a project with XMEGA software AES traces is already open.

'''

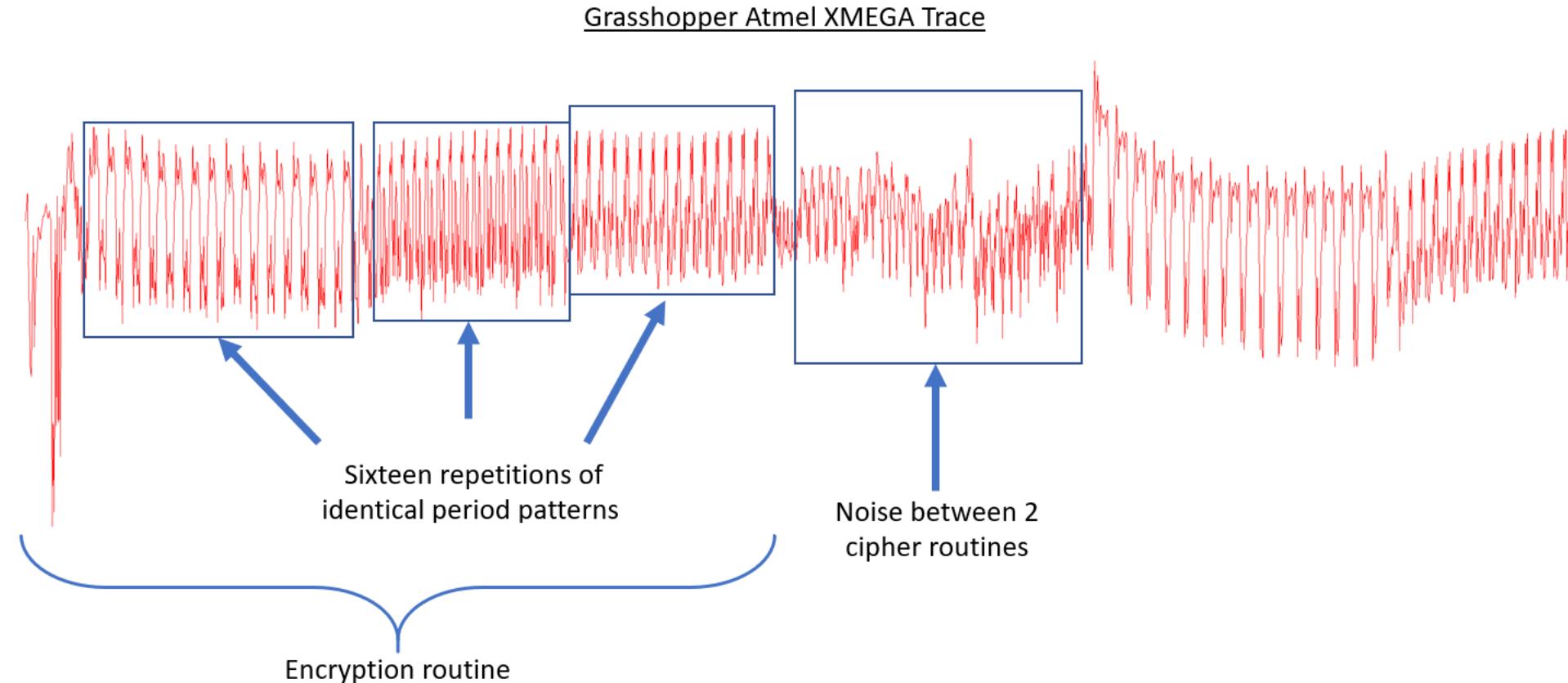
import chipwhisperer as cw
from chipwhisperer.analyzer.attacks.cpa import CPA
from chipwhisperer.analyzer.attacks.cpa_algorithms.progressive import CPAProgressive
from chipwhisperer.analyzer.attacks.models.AES128_8bit import AES128_8bit, SBox_output
from chipwhisperer.analyzer.preprocessing.add_noise_random import AddNoiseRandom

```

Run Edit Edit Copy

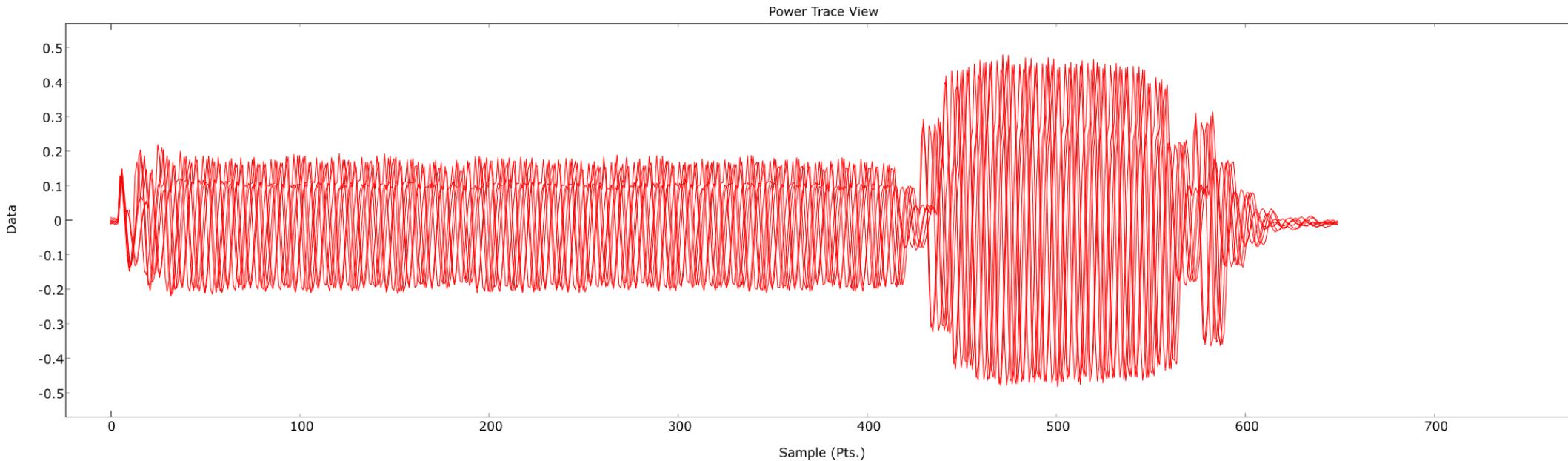
“Kuznyechik” Traces on Microcontroller

- Full trace



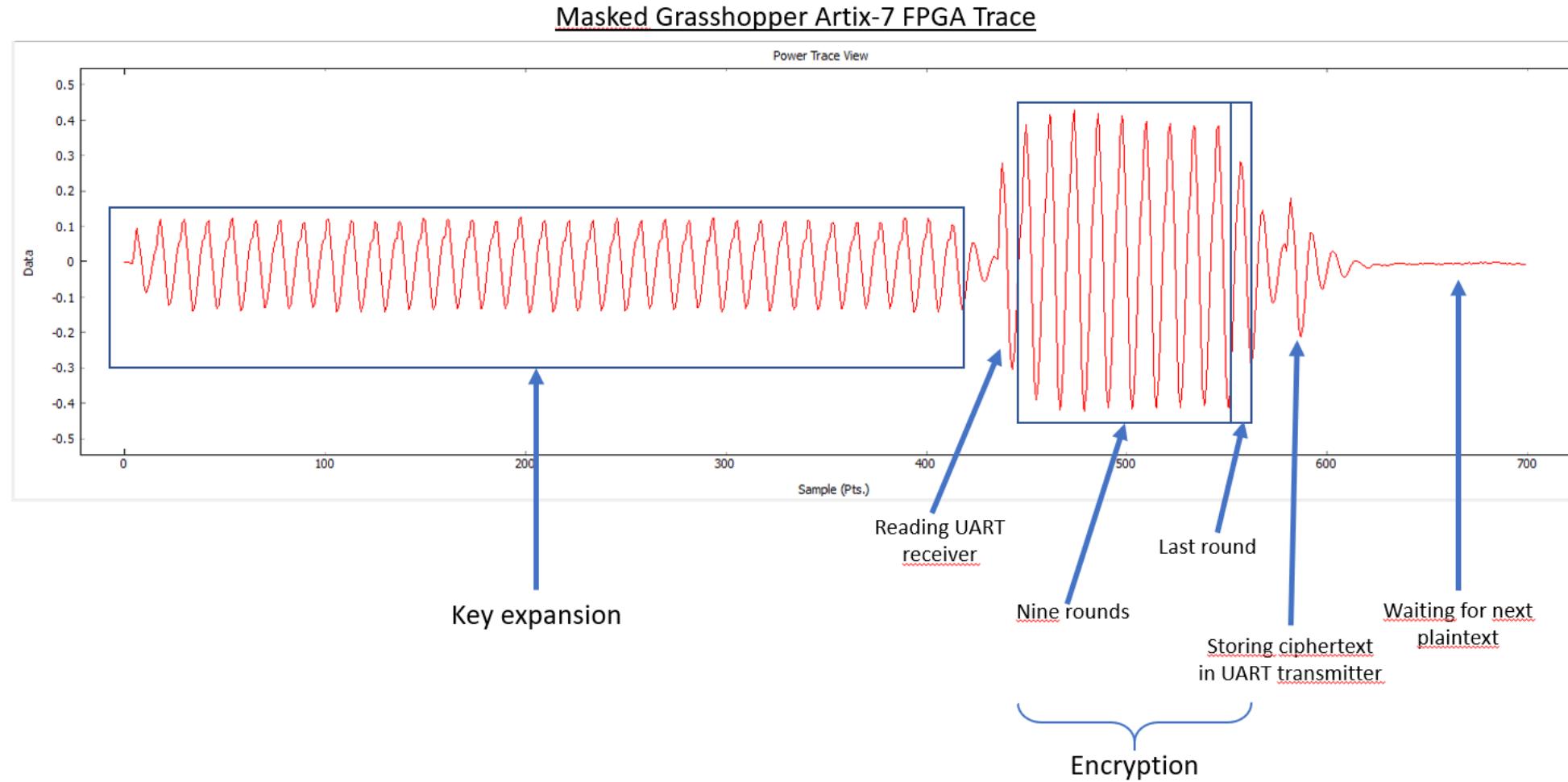
Masked – Kuznyechik on FPGA

- Masked / Non- Masked implementations have very similar signatures
- Here are 10+ traces superimposed (unsynchronized)



Masked – Kuznyechik on FPGA

- Annotated Single Trace



Results & Conclusions



Results



- Working set: 100 000 traces (4000 sample points / trace) acquired.
- Tested attacks:
 - **Rounds :** 1st round / 10th round / 9th round with 10th known key
 - **Leakage models :** Hamming Distance (DPA) & Hamming Weight (CPA)
 - **Mathematical Distinguisher :** Pearson correlation coefficient

Results

Despite the acquisition of a large number of traces, no subkeys could be extracted from a “Kuznyechik” encryption process. No "usual" model, working on **AES**, **DES**, **3DES** or **RSA** made it possible to determine one of the subkeys.

Even if a sub key were discovered, there would still be a problem:

- Subkeys are generated in pairs,
so **2 subkeys are needed to recover the master key** (instead of one for AES !)

$$(K_{2i+1}, K_{2i+2}) = F[C_{8(i-1)+8}] \dots F[C_{8(i-1)+1}](K_{2i-1}, K_{2i}), \quad i = 1, 2, 3, 4$$

$$F[k](a_1, a_0) = (LSX[k](a_1) \oplus a_0, a_1)$$

Where

Results



- Four optimized implementations (2x FPGA + 2x Microcontroller) have been developed, allowing to compare the “Kuznyechik” (GOST R 34.12-2015) algorithm and AES.
- However, even if security elements have been addressed, it is impossible to say whether or not this algorithm is sensitive to DPA / CPA attacks.
- Less “traditional” methods, such as attacks involving the use of **Machine Learning (ML)**, should be considered.
- Or maybe ...

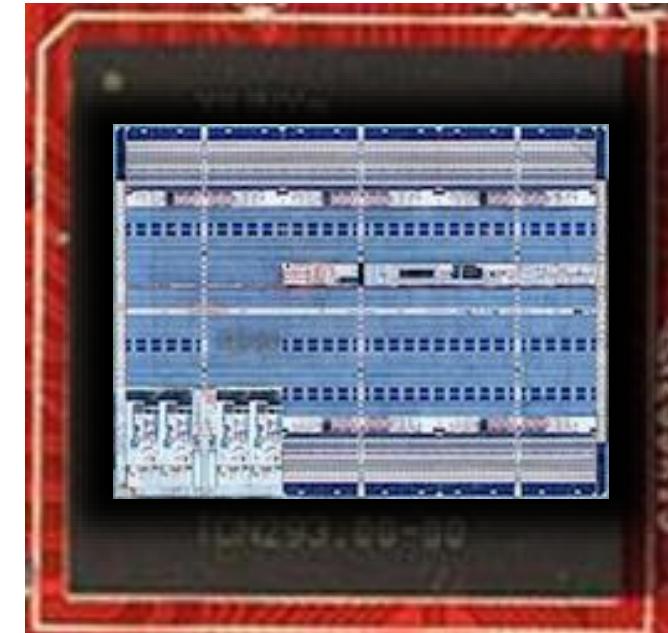
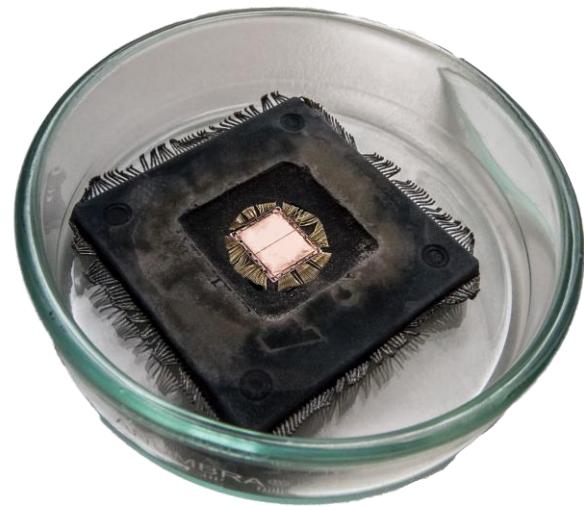
Where to next ?

Кто виноват? Чем делать?



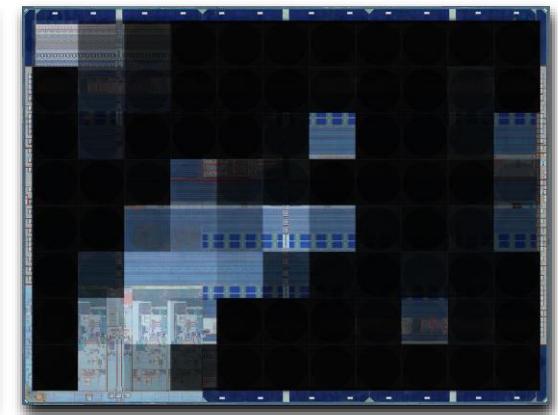
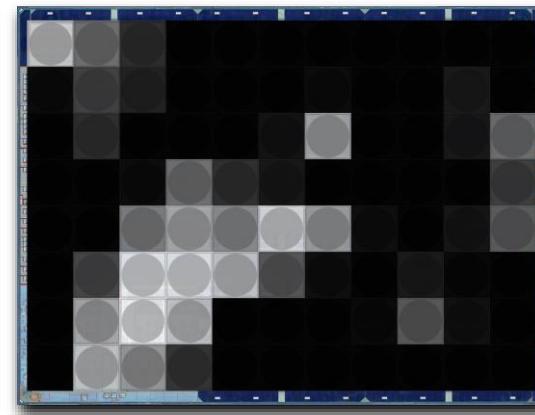
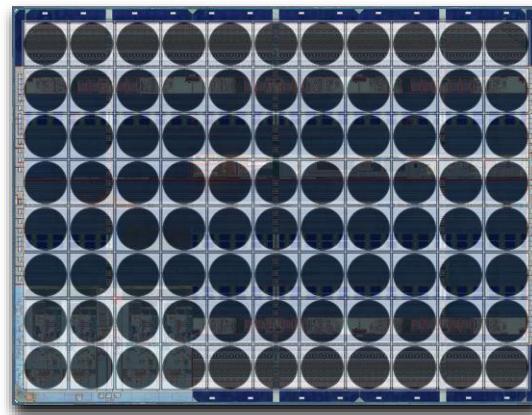
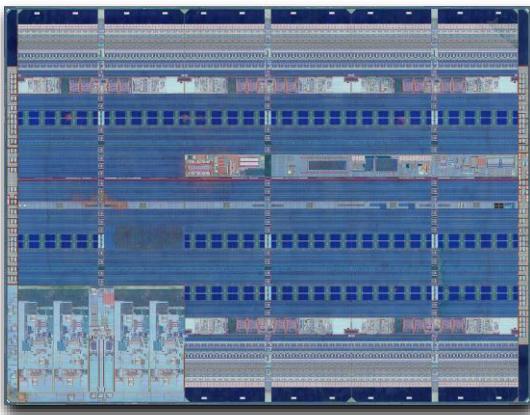
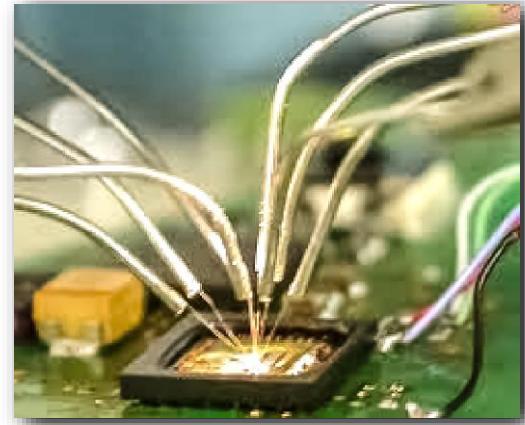
3D - Leakage Cartography

- 2D + Time !
- Works better with chip decapping !



3D - Leakage Cartography

- Very precise X-Y Plotter Table (special Construction)
- Micro H-Probe / Micro H-Probe Array (ev. E-Probe)
- Work in progress !



Caveats ?

- Very High Analysis Cost ! Pessimistic : $(X^*Y : X^*Y) * \text{Traces}$
- Not very usable for small implementations
 - → Small attack surface
- Targeted for high speed implementations using most FPGA gate area
- Semi-invasive / full invasive technique (a little !)



Thank you for your attention !

Cédric DELAUNAY,
ESIEA, (C + V) ^ O Lab, France
cedric.delaunay@ensta-bretagne.org

Делоне Седрик,
ESIEA, (C+V)^O Lab, Франция
cedric.delaunay@ensta-bretagne.org

Alexander Alexandrovich ISTOMIN,
FSUE “SIE “GAMMA”, Russia
istomin.a@nppgamma.ru

Истомин Александр Александрович,
ФГУП «НПП «ГАММА», Россия
istomin.a@nppgamma.ru

Eric FILIOL,
ESIEA, (C + V) ^ O Lab, Laval, France
e.filiol@esiea.fr

Фийоль Эрик,
ESIEA, (C+V)^O Lab, Лаваль, Франция
e.filiol@esiea.fr



XXI - RUSCRYPTO 2019

20.03.2019



36