New Paradigms For Checking Software & Hardware Integrity of Internet of Things Devices

Pascal.Urien@telecom-paris.fr

Jackson Pollock 26A Black and white, 1948

TELECOM Paris

IP PARIS

Introduction

FORESCOUT

The Internet of Things Isn't Coming, It's Here

Webtorials surveyed IT professionals worldwide who are responsible for enterprise communications networks regarding their view about the prevalence and security of the Internet of Things (IoT). The study was sponsored by Forescout Technologies.

Here are just a few findings from the study:

- Only 30 percent are confident they really know what Things are on their network
- Respondents who initially thought they had no IoT devices on their networks, actually had eight IoT device types (when asked to choose from a list of devices)
- Only 44 percent of respondents had a known security policy for IoT

Complete the form to the right to download this informative report.

WIRED

WikiLeaks Reveals How the CIA Could Hack Your Router

SECURITY

WikiLeaks Just Dumped a Mega-Trove of CIA Hacking Secrets

LILY HAY NEWMAN

Software & Hardware Integrity: new paradigms.

US warns of supply chain cyber-attacks

By Gordon Corera
Security correspondent

26 July 2018 Share

Installing compromised software can have expensive repercussions

Search Bloomberg Sign In

Cybersecurity

Vodafone Found Hidden Backdoors in Huawei Equipment

While the carrier says the issues found in 2011 and 2012 were resolved at the time, the revelation may further damage the reputation of a Chinese powerhouse.

Pascal Urien

Jeff Bezos hack: Amazon boss's phone 'hacked by Saudi crown prince'

Exclusive: investigation suggests Washington Post owner was targeted five months before murder of Jamal Khashoggi

- **Revealed: the Saudi heir and the alleged plot to undermine Jeff Bezos**

10,000 $
TOYOTA YARIS II

USBISP USBASP USB ISP Programmer Burner 51 MCU download line for AVR ATTiny 51 AVR Board ISP Downloader ATMEGA8 ATMEGA128

1.0 1 Review 10 orders

IDEAS FOR 2020 Ends in 2 d 23 : 45 : 32

US $1.00 ~~US $1.20~~ -17%

Instant discount: US $3.00 off per US $109.00

US $3.00 New User Coupon | US $8.00 off per US $150.00 Get coupons

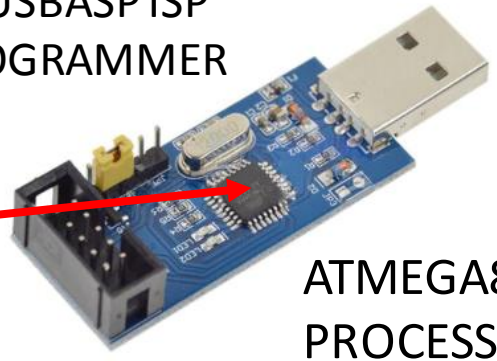
Quantity: 1 Additional 1% off (3 pieces or more) 2980 pieces available

Shipping: US $0.97
to France via Cainiao Super Economy for Special Goods

Estimated Delivery on 04/02

ECU
ENGINE

1$ USBASP ISP
PROGRAMMER



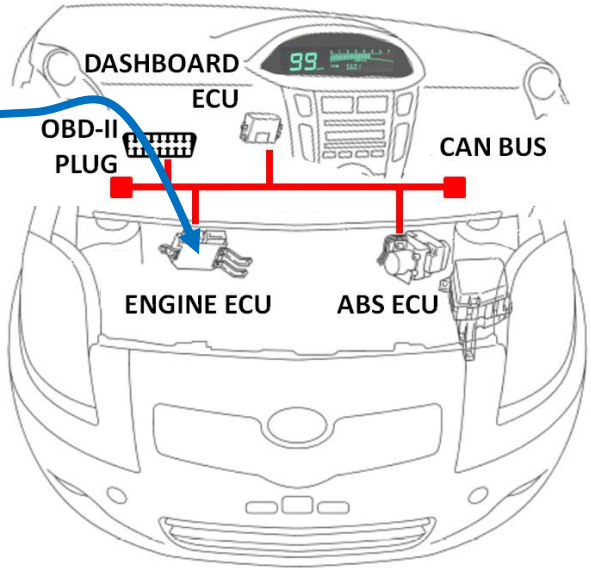
FIRMWARE

ATMEGA8
PROCESSOR

REFLASHING THE ENGINE ECU

- Miller, C., Valasek, C.; "Adventures in Automotive Networks and Control Units", DEFCON 21 2013.
- Urien, P.; " Designing Attacks Against Automotive Control Area Network Bus and Electronic Control Units ", IEEE CCNC 2019
- https://github.com/purien/CanProbe.

30$ CAN Probe



REFLASHING THE ENGINE ECU

(Security Key = 00 06 06 00)

Security Access (SID=27)

Request Seed SID=01

ID= 7E0, Len=08, Data= 02 27 01 00 00 00 00 00 (*SID=27, PID=01*)

ID= 7E8, Len=08, Data= 06 67 01 **41 87 B6 51** 00

Send Seed

Send Key SID=02

ID= 7E0, Len=08, Data= 06 27 02 **41 E7 D6 51** 00

EXOR KEY

ID= 7E8, Len=08, Data= 02 67 02 00 00 00 00 00

Positive Response

DiagnosticSessionControl (SID= 10), Programming Session (PID=02)

ID=7E0 Len=08 Data= 02 10 02 00 00 00 00 00

ID=7E8 Len=08 Data= 01 50 00 00 00 00 00 00

Positive Response

READING & REFLASHING THE USBASP TOKEN FIRMWARE

Compatible Atmel AT AVR ISP mk2 MKII ATMEL AVR Programmer USB AVRISP XPII In-System Programmer Supports AVR Studio 4/5/6/7

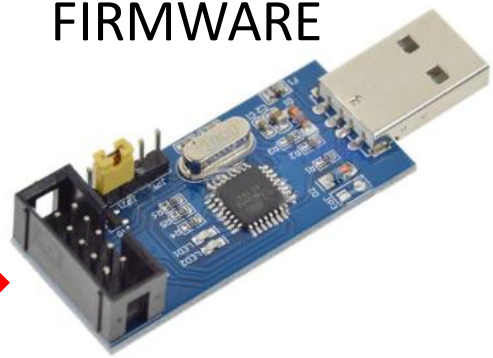
de Waveshare

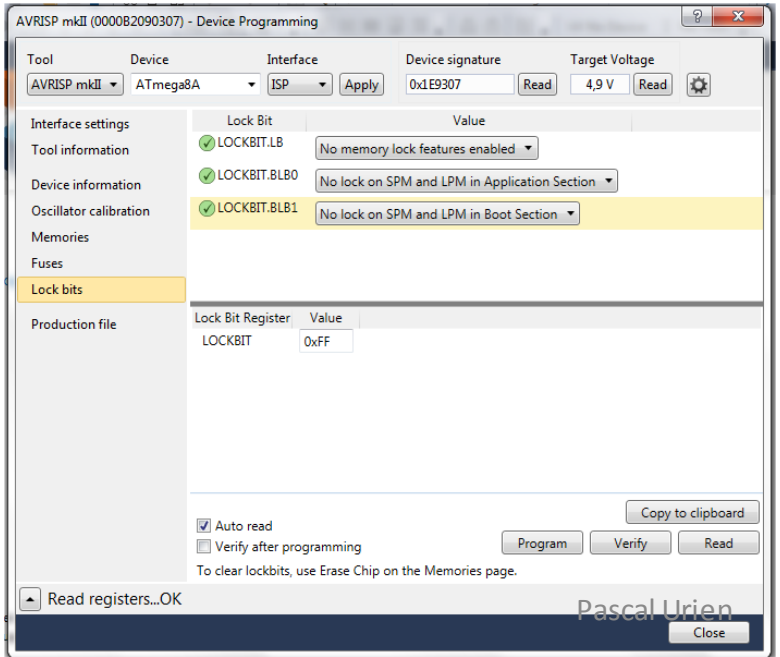
Prix : 35,32 € prime

Tous les prix incluent la TVA.

Économisez 3 % au moment de passer la commande. Détails

Livraison GRATUITE (0,01€ pour les livres) en point retrait. Détails

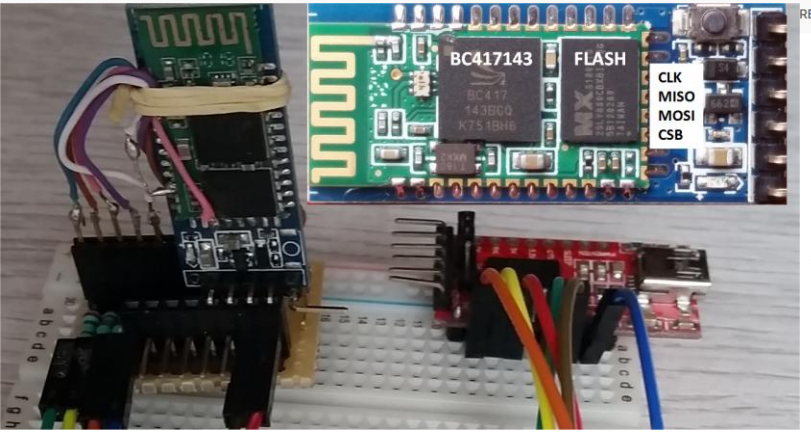




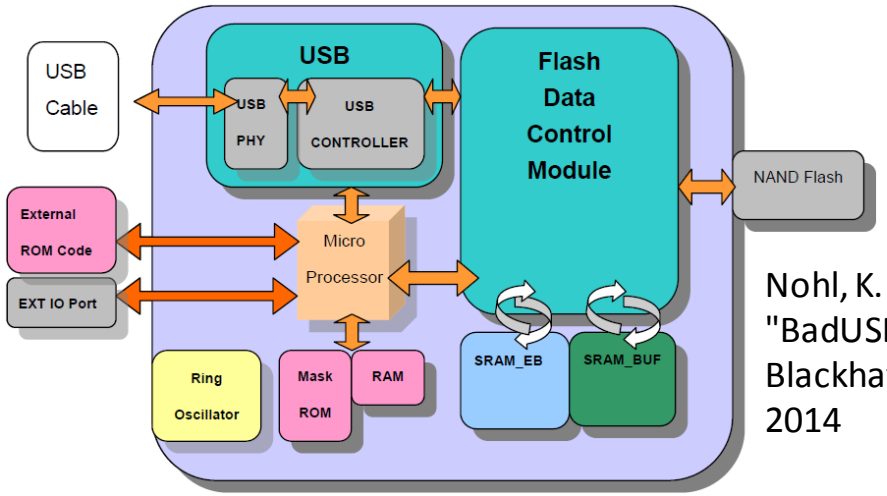
Thomas Roth boots a SNAKE game in a Ledger Nano S device. The bootloader is locked by the F00DBABE code

35C3 conference
December 2018

35C3 - wallet.fail



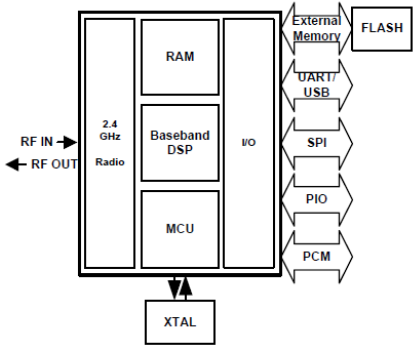
Bluetooth SoC reprogramming



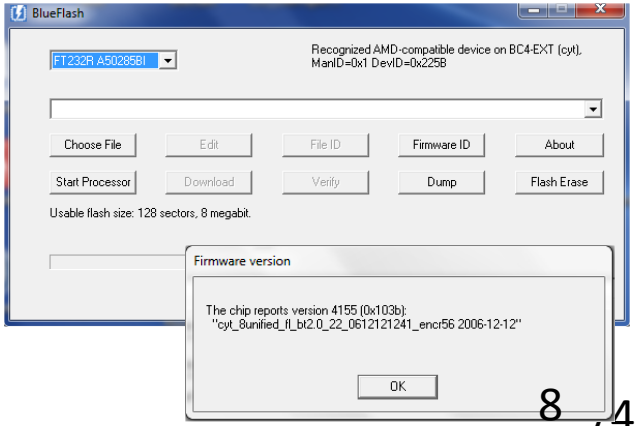
Nohl, K. & Al. "BadUSB… ", Blackhat 2014

FLASH Controller PS2251-33
Bootloader in ROM, Firmware in FLASH

CSR BC417143 (NO ROM!)

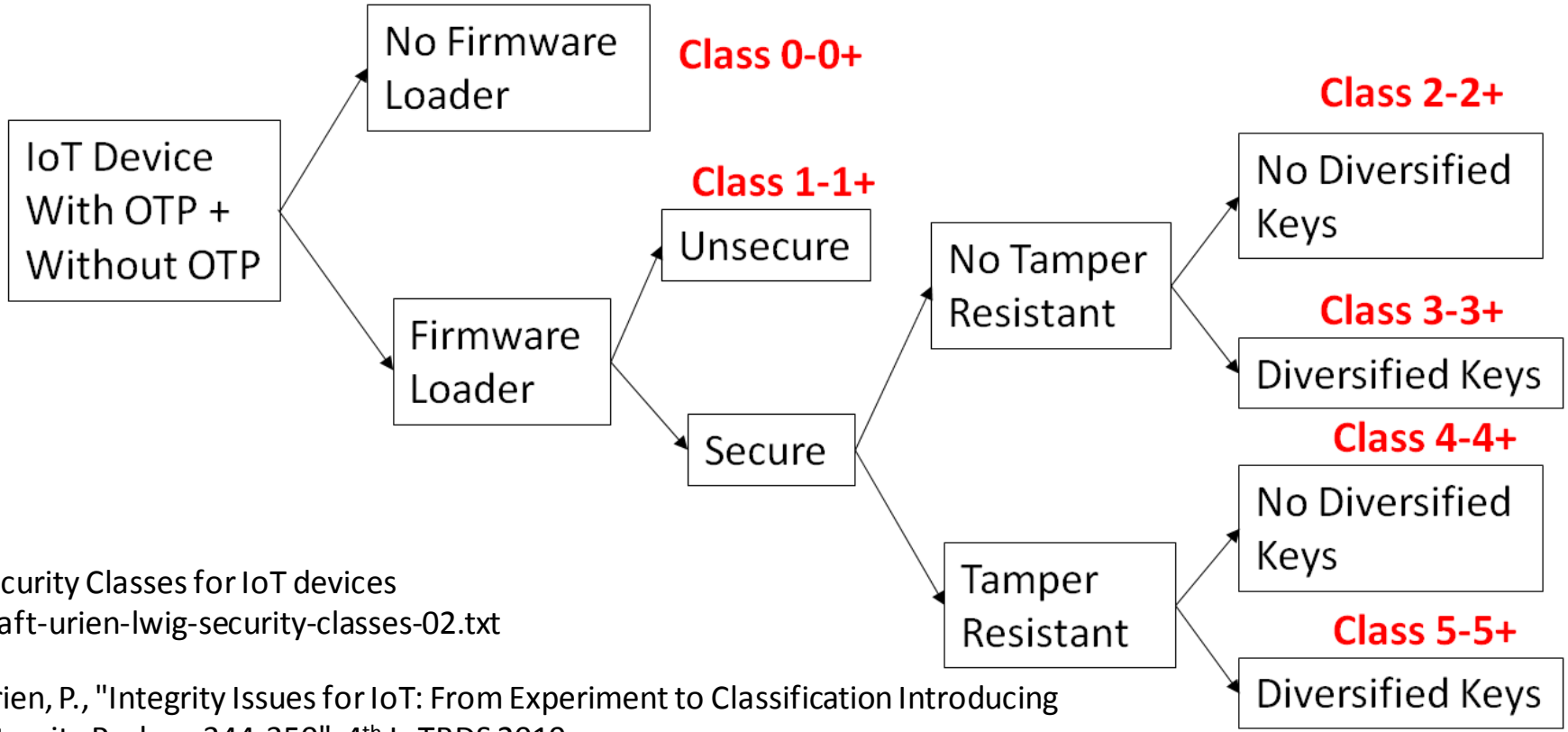


System Architecture



Pascal Urien

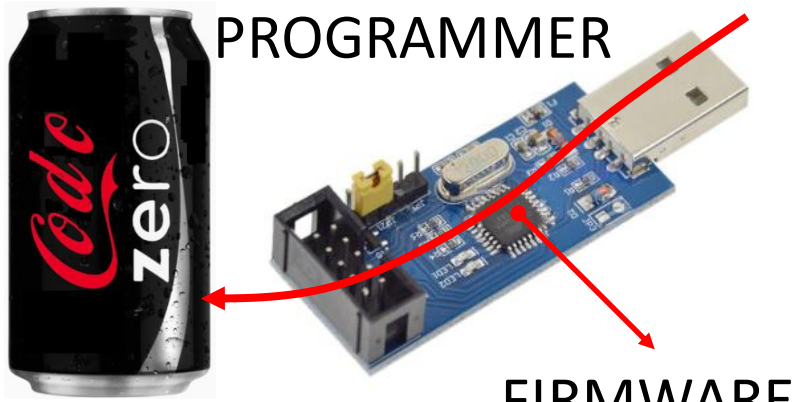
Security Classes For IoT



Security Classes for IoT devices
draft-urien-lwig-security-classes-02.txt

Urien, P., "Integrity Issues for IoT: From Experiment to Classification Introducing Integrity Probes. 344-350", 4th IoTBDS 2019

This Talk : Model & Issues To Solve



- Trust insurance in programmer firmware
  - Time Stamped Bijective MAC algorithm (bMAC)
- Trust insurance in programmer chip
  - Dynamic PUF (physical key)

Time Stamped bMAC

Finite Memory

Time

Eliud
Kipchoge
2h01mn03s

Bijective MAC

- bMAC computes a fingerprint of a set of memories (m)such as FLASH, SRAM, EEPROM, according to a pseudo random order, fixed by a permutation P

$$bMAC(P) = h(\ A(P(0)\ ||\ A(P(1)||\ \dots\ ||\ A\big(P(i)\big)\ \dots\ ||\ A(P(m-1)\ )$$

  - h= SHA2, SHA3,…
  - Given a set of memories m, m! permutations are available (35! > 2^{128}).
  - bMAC proves the knowledge of memory contents (m)

"Bijective MAC for Constraint Nodes", draft-urien-core-bmac-05.txt, https://tools.ietf.org/html/draft-urien-core-bmac-05.

Urien, P., "Integrity Probe: Using Programmer as Root of Trust for Bare Metal Blockchain Crypto Terminal", MobiSecServ'2019

bMAC & Remote Attestation

- *R~~emote attestation~~ bMAC is a process whereby a trusted entity (verifier) ~~remotely~~ measures internal state of a untrusted possible compromised device (prover).*
- *The ~~ICE~~ bMAC verification function is a self-check ~~summing~~ hash code, i.e. a sequence of instructions that compute a ~~checksum~~ fingerprint over themselves in a way that the ~~checksum~~ MAC would be wrong or the computation would be slower if the sequence if instruction is modified*

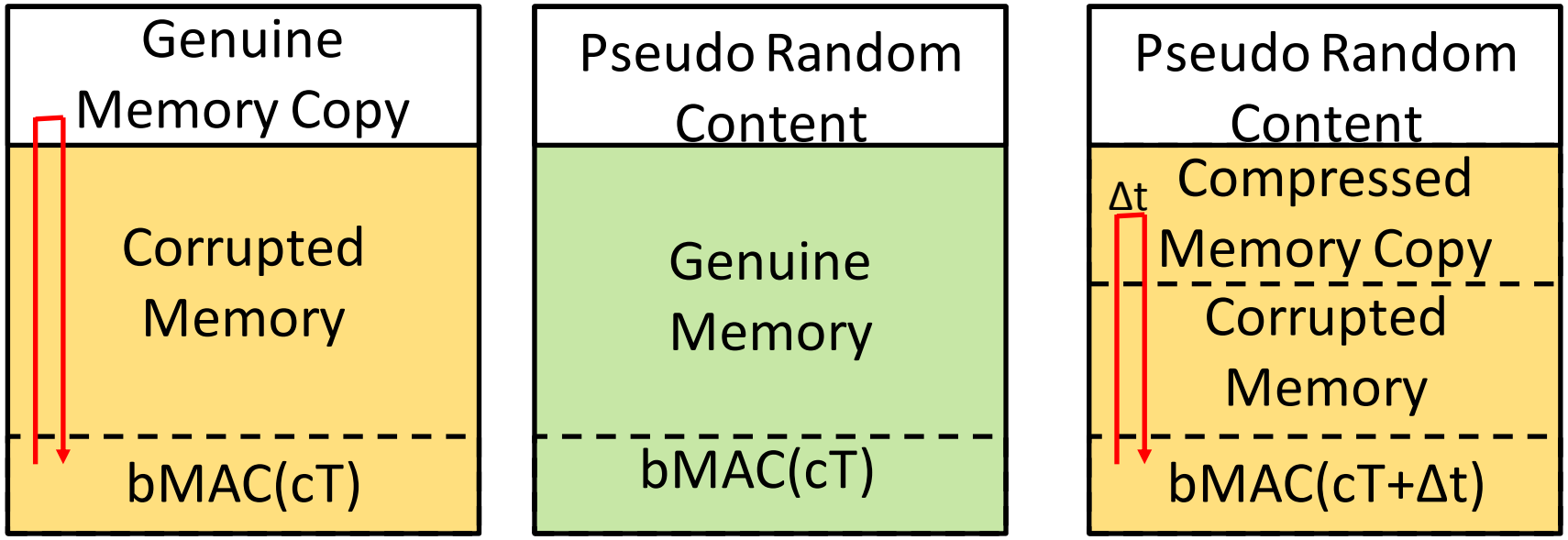
$$ICE = Checksum(\ A(P(0)\ ||\ A(P(1)||\ \dots\ ||\ A\big(P(i)\big)\ \dots\ ||\ A(P(m-1)\ )$$

Asokan, N. et al. "ASSURED: Architecture for Secure Software Update of Realistic Embedded Devices.". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37.11 (2018): 2290-2300.

Seshadri, A. et al. "SCUBA: Secure Code Update By Attestation in sensor networks.", in Radha Poovendran & Ari Juels, ed., "'Workshop on Wireless Security" , ACM, , pp. 85-94 (2006).

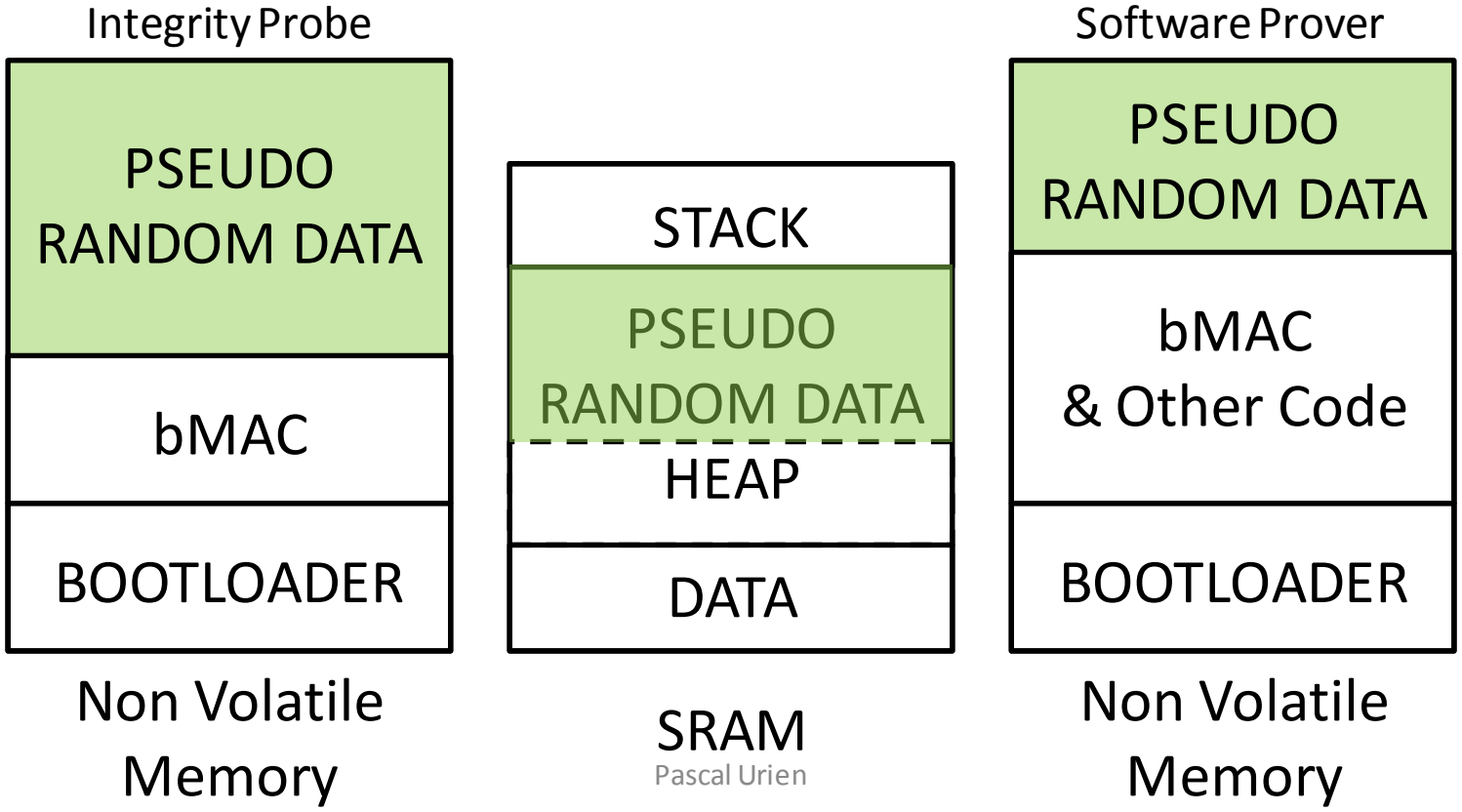
bMAC Security & Time Stamped bMAC

- bMAC fills all unused memories (FLASH, SRAM, EEPROM,…) by pseudo random content
- bMAC_TS = Time Stamped bMAC = bMAC exor ComputingTime = bMAC exor cT



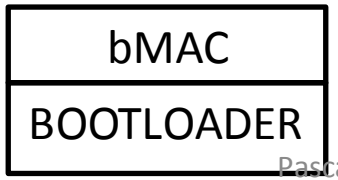
Memory Copy Attack | Time Stamped bMAC | Compressed Memory Copy Attack

bMAC usecases



Minimum (Optimal) Code Size

- A given a set of functionalities a program P has a *Minimum Code Size* (*optimal* size) :
  - This is true because a code has a finite non zero size.
  - This implies that it is not possible to compress by any means a MCS implementation.
- We assume that the P_3 MCS merge ($P_3 = P_1 \cup P_2$) of two MCS programs P_1 and P_2 with "*different functionalities*", follows the relation:
  - Size ($P_1 \cup P_2$) > Max (Size(P_1), Size(P_2))
- If a bootloader is MCS, then adding functionalities increases the code size over the physical limit.
- If the bMAC code is also MCS, then it could not be stored in the remaining FLASH space.



Permutation Choice

- bMAC works with exponential permutations based on generators in Z/pZ*, p prime (p>m)
  - x ε [1, p-1], F(x)= g^x mod p
  - y ε [0, m-1], P(y)= F(y+1)-1
- Examples of polynomial permutations in Z/pZ:

$$P(x) = x + x^2 \vee C \ mod\ 2^n$$ A. SHAMIR & AL, 2002

$$P(x) = a_0 + a_1 x + \cdots + a_d x^d \ mod\ 2^w$$ R.L. RIVEST, 2001

$$P(x) = 1 + x + x^2 + \cdots + x^d \ mod\ p^e$$ R. MATTEWS, 1994

A. Klimov, and A. Shamir. "A New Class of Invertible Mappings.", . CHES, volume 2523 of Lecture Notes in Computer Science, page 470-483. Springer, (2002)

R. L. Rivest, "Permutation polynomials modulo 2^w". Finite Fields And Their Applications, 7, 287-292 (2001).

Matthews R., "Permutation Properties Of The Polynomials 1 + x +...+x^k Over A Finite Field";. Proceedings of the American Mathematical Society, Volume 120, Number 1, January 1994

bMAC Permutation & Computing Time

$$F(x) = {g_2}^{s_1 {g_1}^{x} \bmod p} \bmod p, \quad x, s_1 \in [1, p-1]$$

(#$p^3/4$, $10^{12}/4$ for p~10^4)

g2, s1, g1 = PRNG(random)

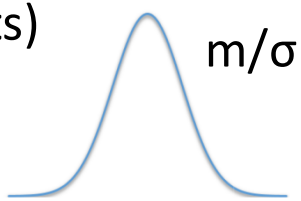
$$g^x = \prod_{i=0}^{n-1} g^{2^i b_i} \bmod p$$

Square & Multiply (S&M) Algorithm

x= $b_0 + b_1 2^1 + \ldots + b_{n-1} 2^{n-1}$ (n bits)

$$p(k.T_m) = 0{,}5^n C_n^k \;, k \in [0, n]$$

k bits set to 1, Tm modular multiplication computing time



m/σ = √n/2

S&M timing effect

$$g_k = p - (2^k \bmod p)\,, k \in [1, q-1]$$

p= Sophie Germain prime, with p=7 mod 8

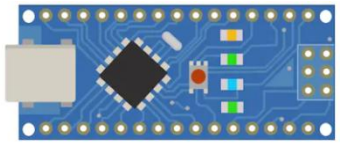
Generators g_k for p=2q+1

- q prime, p=7 mod 8

BMAC implementation (ATMEGA8)

FLASH: 8KB (2KB BOOTLOADER), SRAM: 620B (/1024B)
EEPROM: 512B
p=9887 > 8192+1024+512= 9728

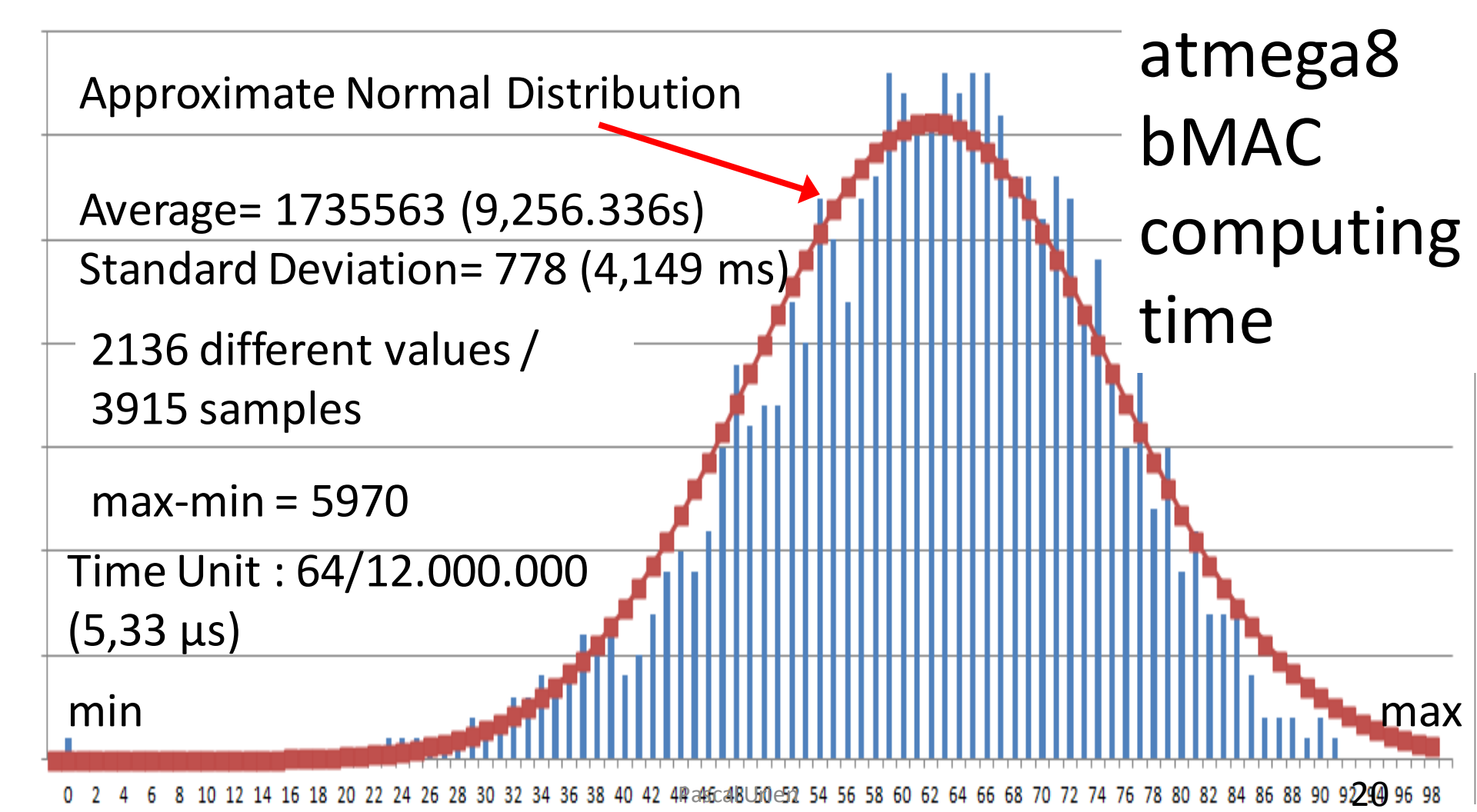
```
#define NBITS 14
uint32_t cT,x,y,bitn,v,gi[NBITS];
uint32_t s1=a_value
uint32_t
g1=a_generator,g2=a_generator;
uint16_t PRIME=9887;
uint8_t a[1],bMAC[32];
bool tohash;
disable_interrupts();
initialize_timer();
Keccak.reset();
gi[0]= g2;
for (uint8_t n=1;n<NBITS;n++)
gi[n] = (gi[n-1] * gi[n-1]) % PRIME;
x= s1;
```



bMAC for Arduino NANO
https://github.com/purien/bMAC

```
for(uint16_t i=1;i<PRIME;i++)
{ tohash = false
  x = (x*g1) % (uint32_t)PRIME;
  bitn=x;
  y=1;
  for (int n=1;n<=NBITS;n++)
  { if ( (bitn & 0x1) == 0x1)
    y = (y*gi[n-1]) % PRIME;
    bitn = bitn >>1;}
  v = (y-1);
  // if address v exists,
  // { read address a[0]=A(v)
  //  tohash=true ;}
  if (tohash) Keccak.update(a);
  cT=read_timer();
}
Keccak.dofinal(bMAC);
cT=read_timer();
```

64 cycles resolution


Approximate Normal Distribution
Average= 1735563 (9,256.336s)
Standard Deviation= 778 (4,149 ms)
2136 different values /
3915 samples
max-min = 5970
Time Unit : 64/12.000.000
(5,33 µs)
min
max
atmega8
bMAC
computing
time
0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98

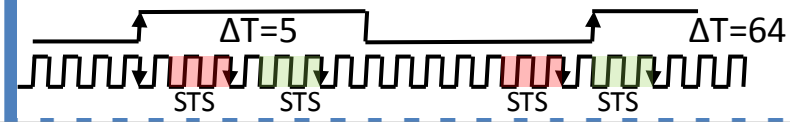
Stop & Start Timer Attack (Arduino NANO)

1163 measures
min= 980
max= 1245
max-min=265
193 different values
Average= 1121
Standard Deviation= 40

Time Unit =
64/16.000.000= 4 µs

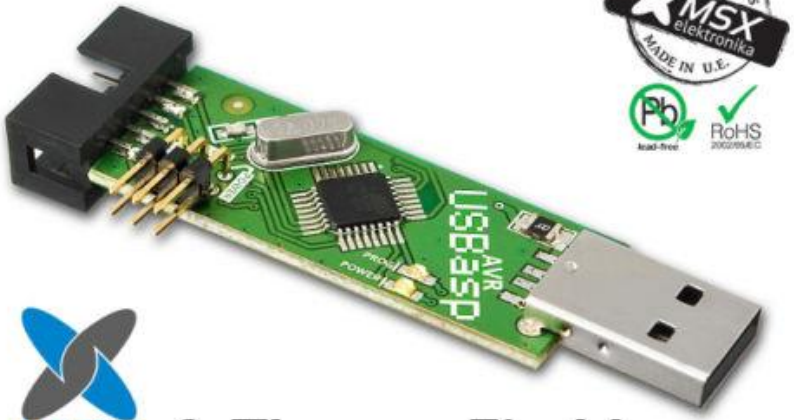
Average per computation =
1121*64/34812 = 2,06 cycles

```
TCCR1B = 0 ; // Stop Timer
TCCR1B = 3 ; // Start Timer
```

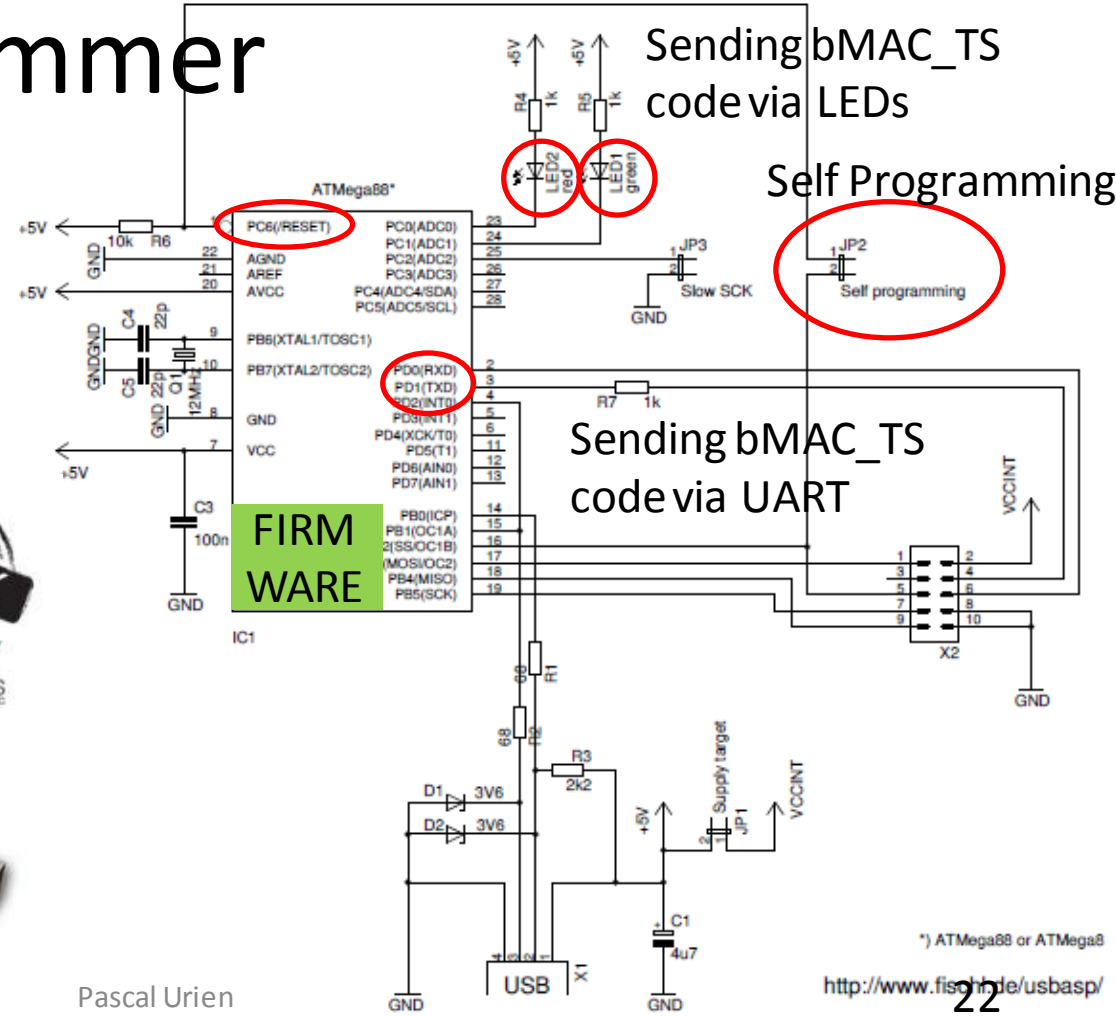


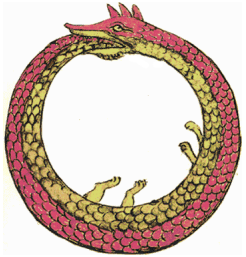
Open SPI Programmer

- Thomas Fischl (2011)
  - USBasp - *USB programmer for Atmel AVR controllers*



MSX & Thomas Fischl





Ouroboros

A flasher can be flashed

- Ouroboros* : a bootloader compatible with USBasp** driver
  - USBasp applications are downloaded thanks to the Ouroboros bootloader.

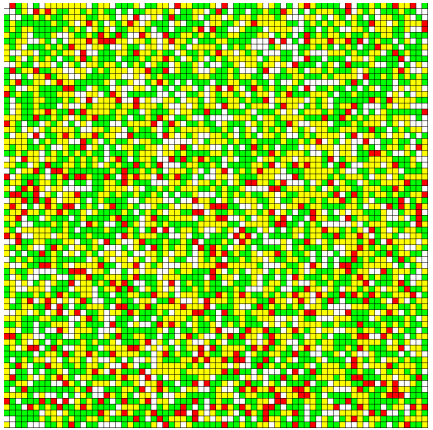
**USBasp (*USB programmer for Atmel AVR controllers*), Thomas Fischl 2011-2019

*Ouroboros Project, Jonathan Thomson, 2011-2019

Integrity Probes check bootloader

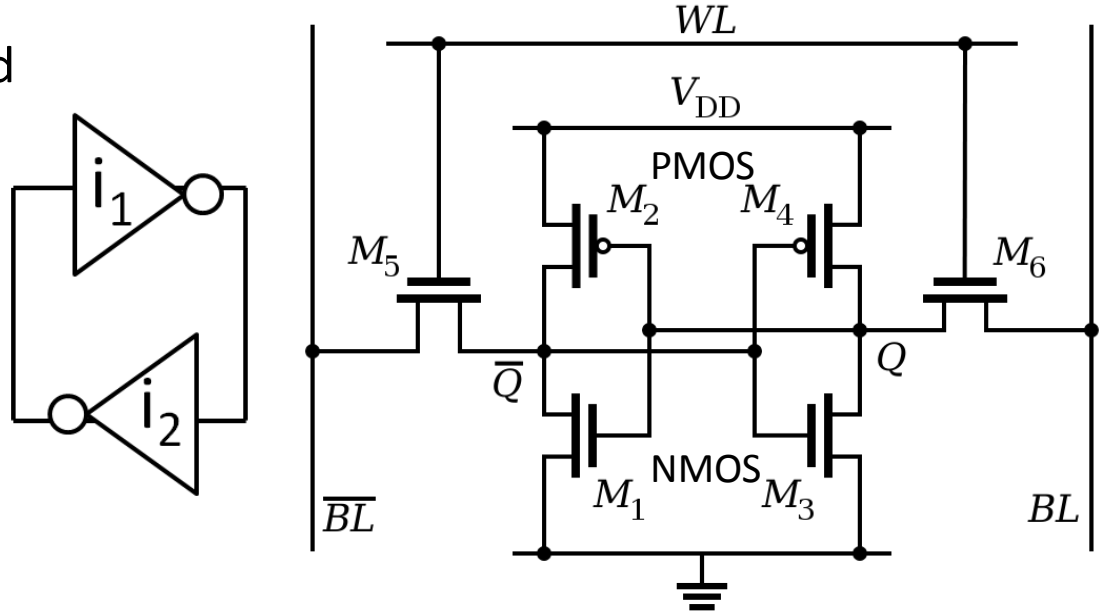
In classical systems applications trust bootloader

SRAM PUF & Flipping-bits



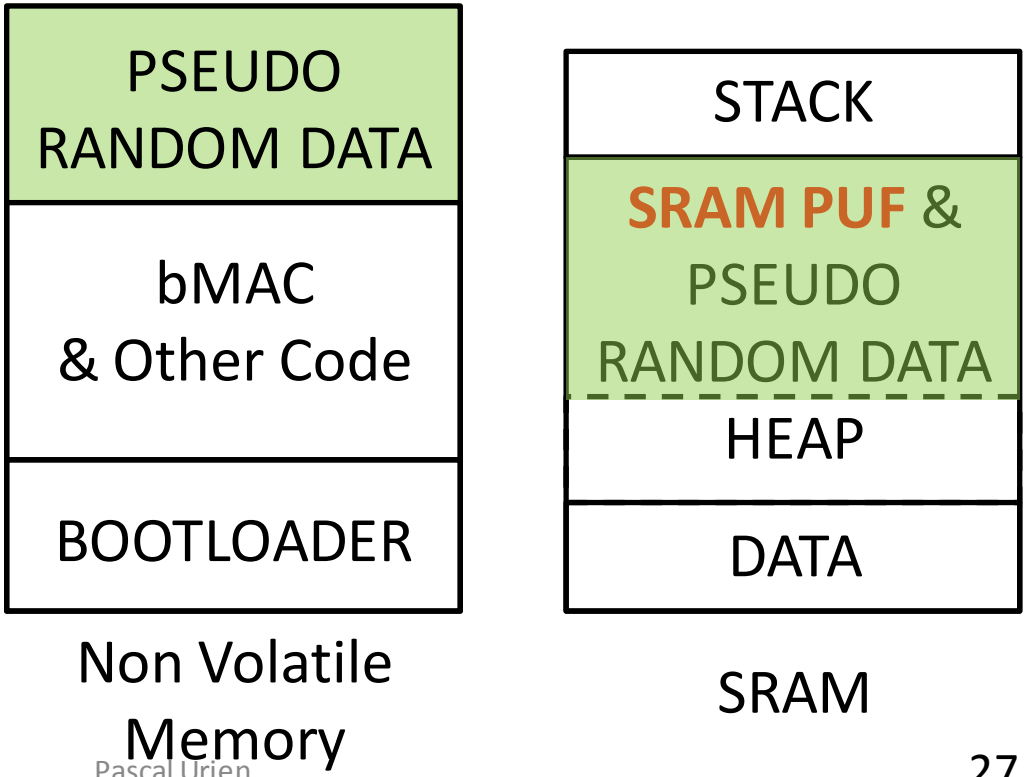
About SRAM PUF

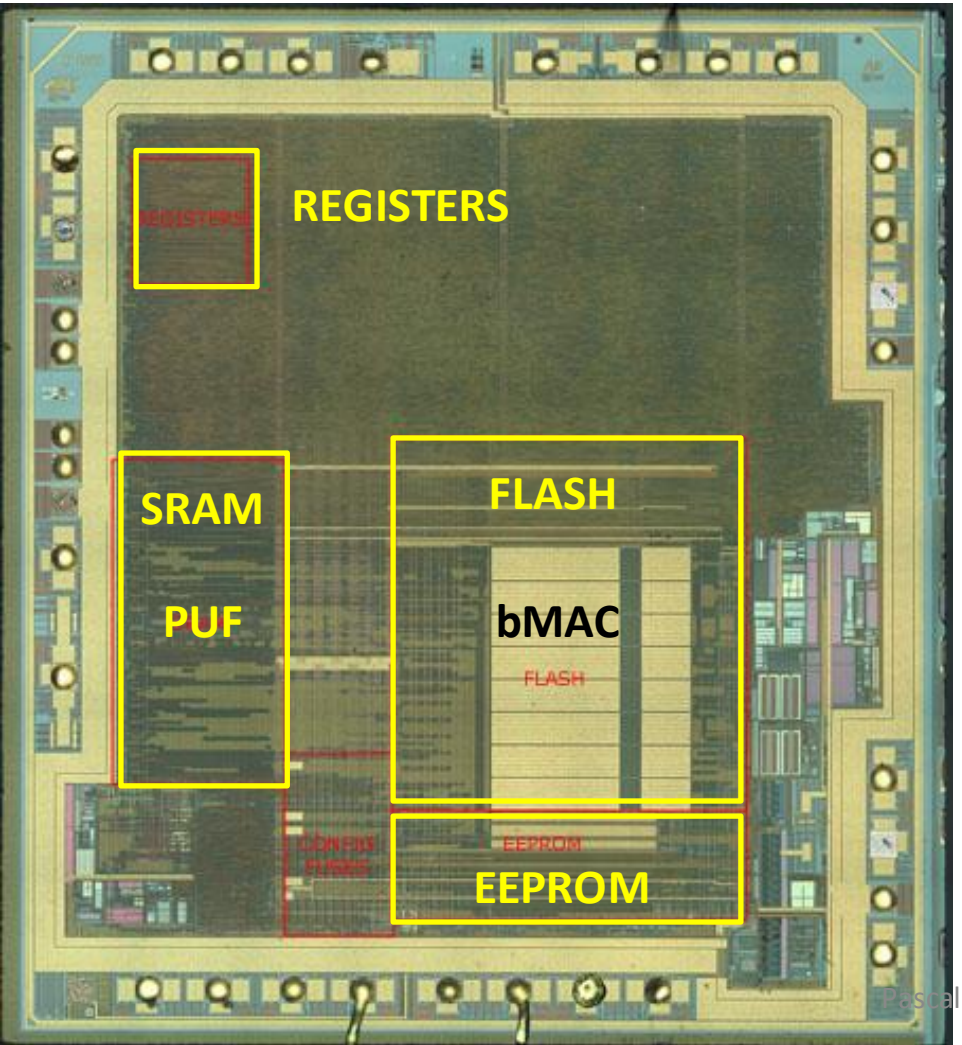
- A SRAM memory is made with 6 CMOS transistors, and includes two invertors (i1 and i2) connected in series (i.e. head to tail).
- Due to transistors physical and electrical dissymmetry, some memory cells take a fixed value (non random) after powering up.
- This effect (SRAM PUF) may be used for micro controller unit (MCU) authentication purposes.

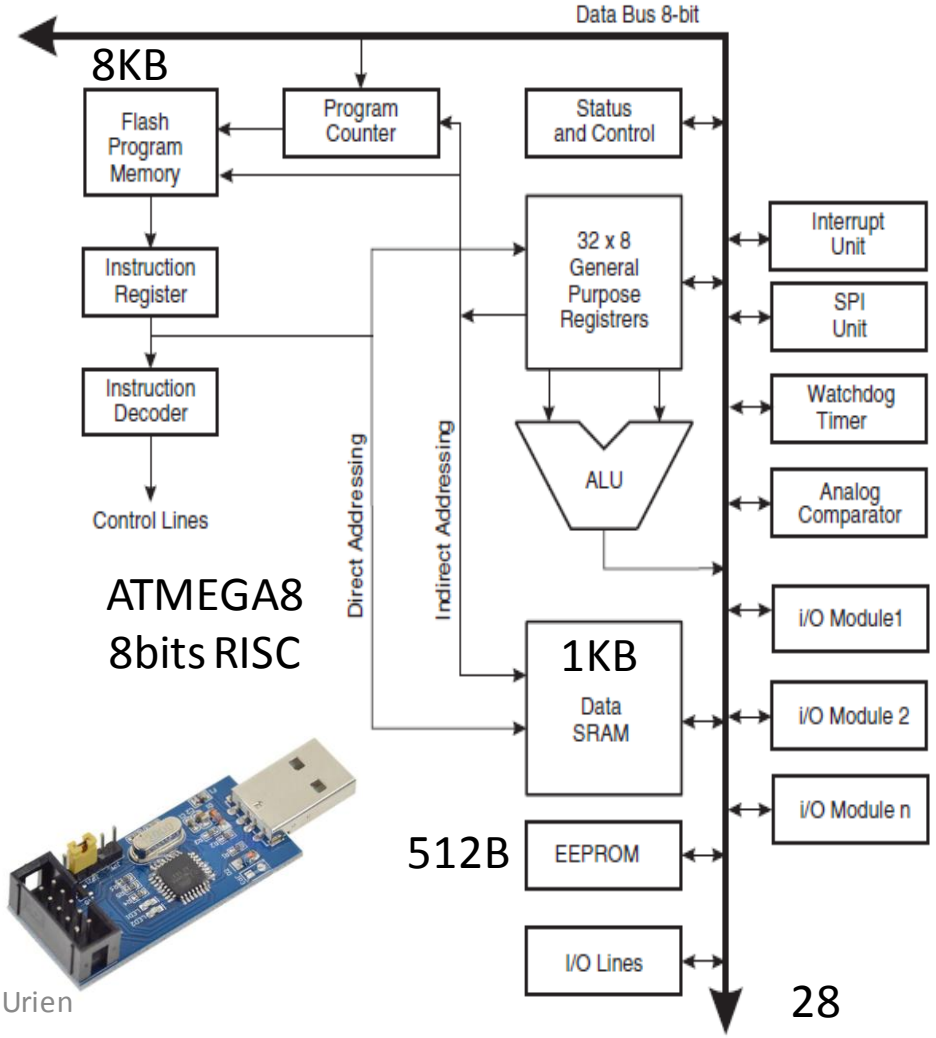


bMAC & SRAM PUF

- Idea: Binding bMAC to device
- How: bMAC is computed with SRAM PUF

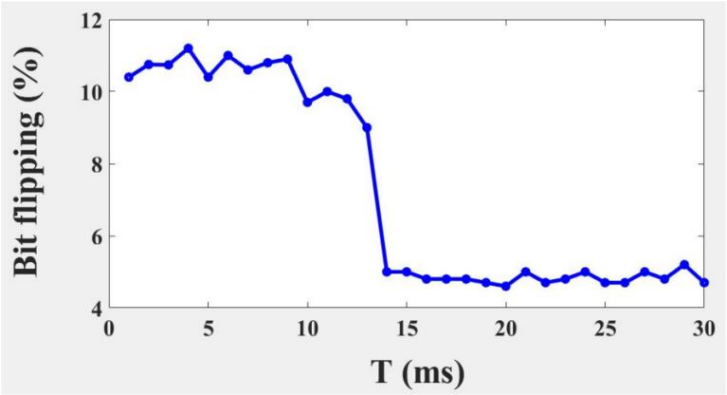



REGISTERS
SRAM
PUF
FLASH
bMAC
FLASH
EEPROM
EEPROM


Data Bus 8-bit
8KB
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
1KB
Data SRAM
512B
EEPROM
I/O Lines
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
i/O Module1
i/O Module 2
i/O Module n
ATMEGA8
8bits RISC

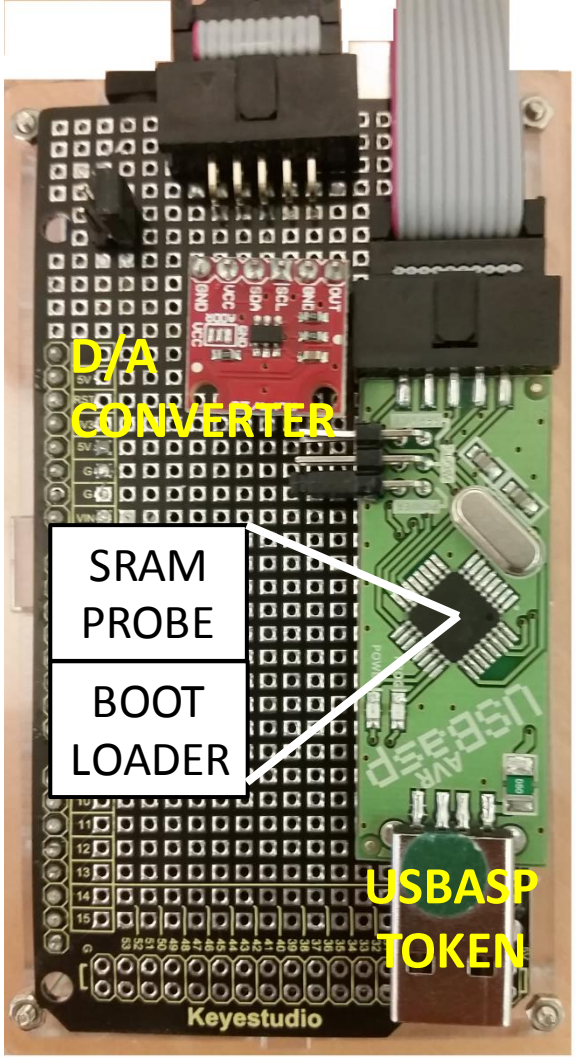
SRAM PUF Extraction

- A SRAM Probe is downloaded in the ATMEGA8
- It has been demonstrated* that SRAM PUFs are dependant from the voltage rising time and more generally from the powering-up signal.

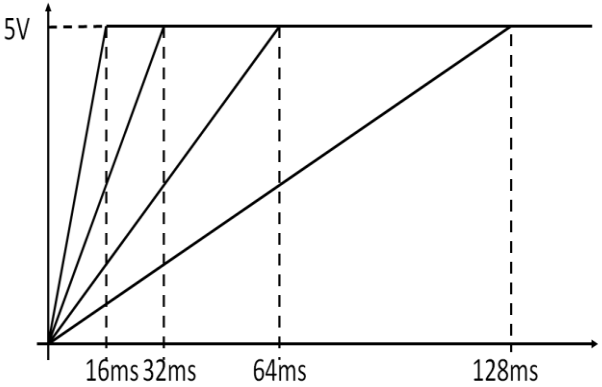


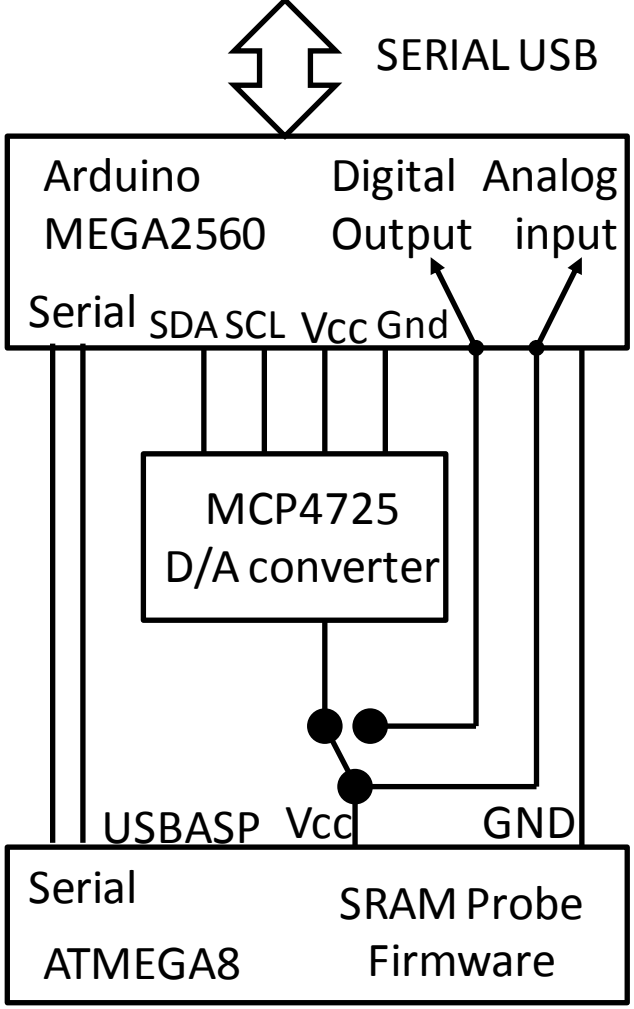
*Chayanika Roy Chaudhuri, "Effects of Temporal Variations on Delay based Physical Unclonable Functions", Master's Thesis, 2016

*Abdelrahman T. Elshafiey ; Payman Zarkesh-Ha ; Joshua Trujillo, "The effect of power supply ramp time on SRAM PUFs", IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), 2017



Urien, P.; "Innovative ATMEGA8 Microcontroler Static Authentication Based on SRAM PUF", IEEE CCNC 2020





Notations

- For N measures, performed on a device k, we note $M^N k(i)$ the number of 1 occurrences for a memory cell i.
- We call ***domain*** $D^N k$ the set of cells that are :
  - always seen at 1, $H^N k = \{ i \mid M^N k(i) = N \}$
  - always seen at 0, $L^N k = \{ i \mid M^N k(i) = 0 \}$,
- Other cells are referred as *noisy* $N^N k$
- $M^N k = H^N k \cup L^N k \cup N^N k$
- $D^N k = L^N k \cup H^N k$

- For two devices and two set of measures $M^{N2} k_1$ and $M^{N2} k_2$ we call ***common domain*** $C^{N1,N2} k_1,k_2$ the cells that belong to $D^{N1} k_1 \cap D^{N2} k_2$.
  - $C^{N1,N2} k_1,k_2 = D^{N1} k_1 \cap D^{N2} k_2$
- For two devices and two set of measures, we call ***flipping bits*** the cells $F^{N1,N2} k_1,k_2$ that belong to the common domain $C^{N1,N2} k_1,k_2$ but with different values, i.e. :
  - $F^{N1,N2} k_1,k_2 = (H^N k_1 \cap L^N k_2) \cup (L^N k_1 \cap H^N k_2)$

Graphical convention:
- $H^N k$ in green
- $L^N k$ in yellow
- $N^N k$ in white
- $F^{N1,N2} k_1,k_2$ in red

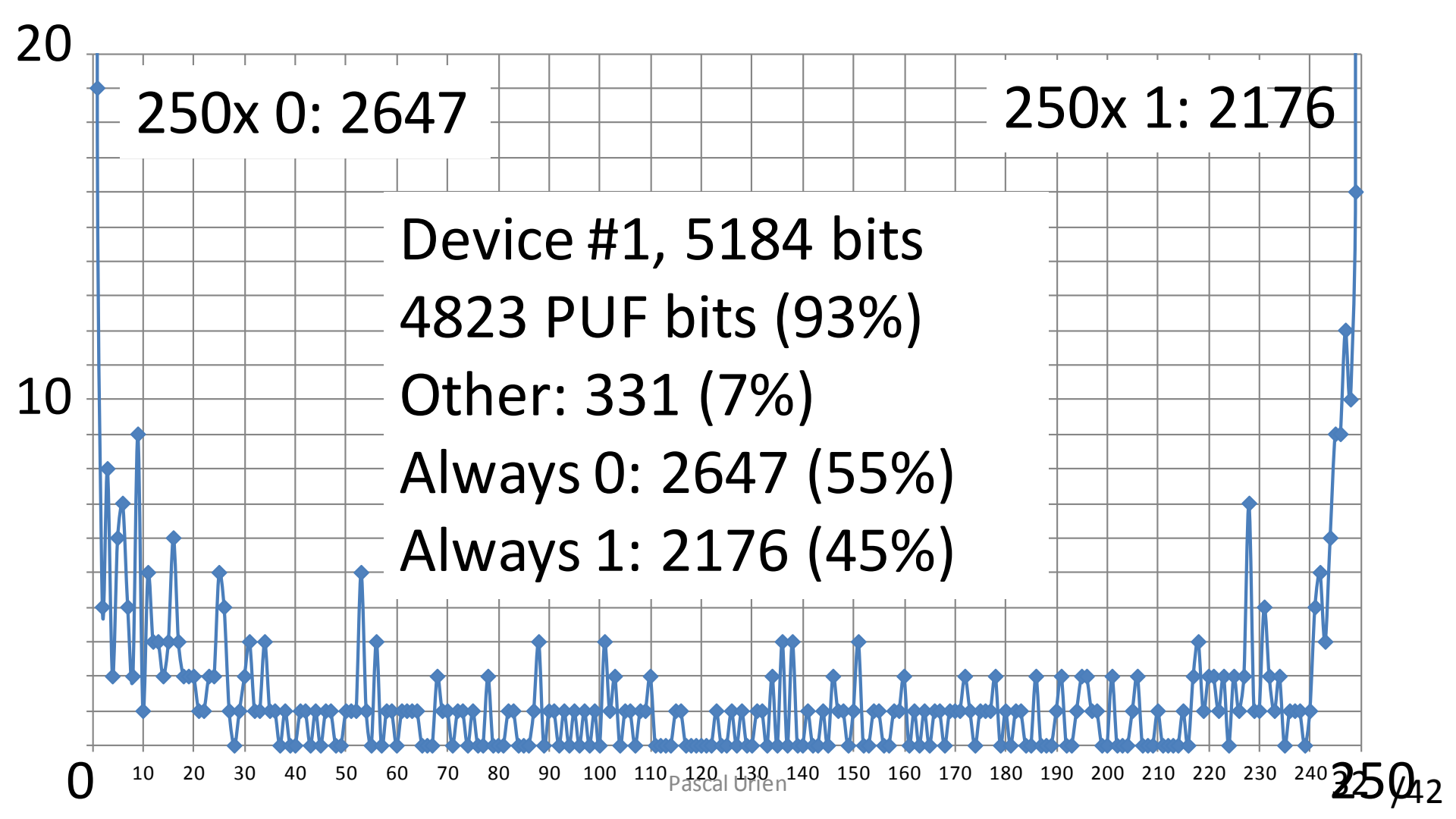
Graphical Comparaison:

$H^1 \wedge H^2 \rightarrow H$ $N^1 \wedge X^2 \rightarrow N$

$L^1 \wedge L^2 \rightarrow L$ $X^1 \wedge N^2 \rightarrow N$

$H^1 \wedge L^2 \rightarrow F$

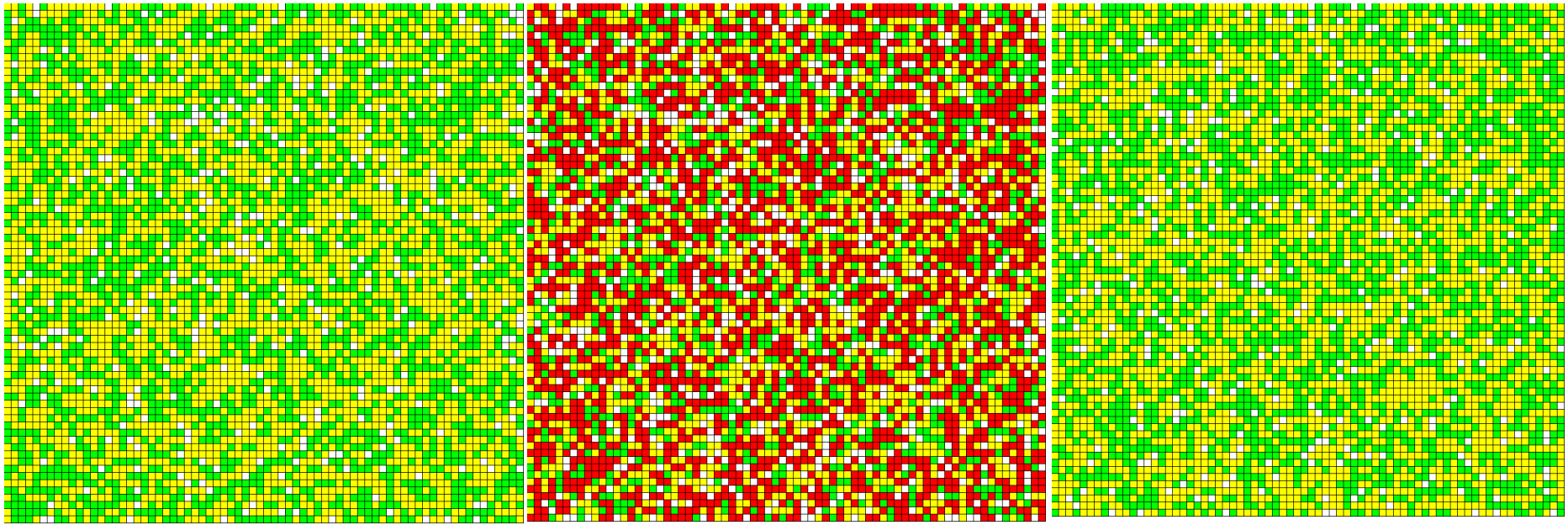
$L^1 \wedge H^2 \rightarrow F$


20
10
0
250x 0: 2647
250x 1: 2176
Device #1, 5184 bits
4823 PUF bits (93%)
Other: 331 (7%)
Always 0: 2647 (55%)
Always 1: 2176 (45%)
10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200 210 220 230 240 250

4823 bits (93%)
1: 45% - 0:55%

Common Domain 4517 bits
Match: 2324 (51 %) – NoMatch: 2193 (49%)

4856 bits (93%)
1:46% - 0: 54%

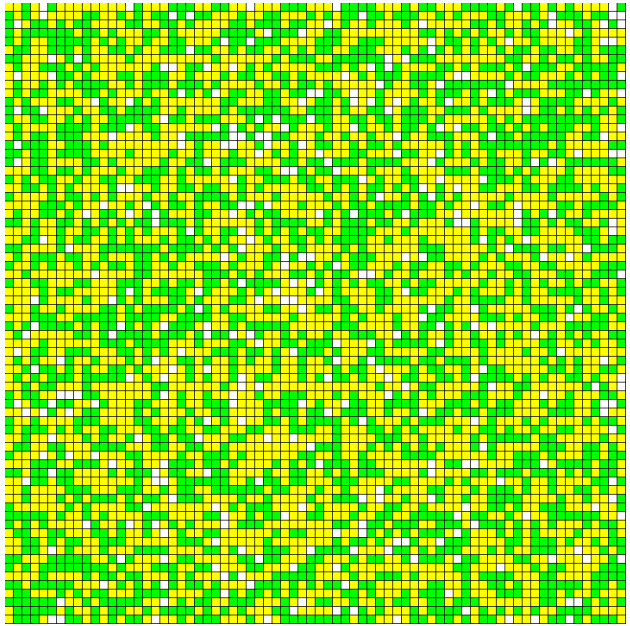


DEVICE#1
250 MEASURES

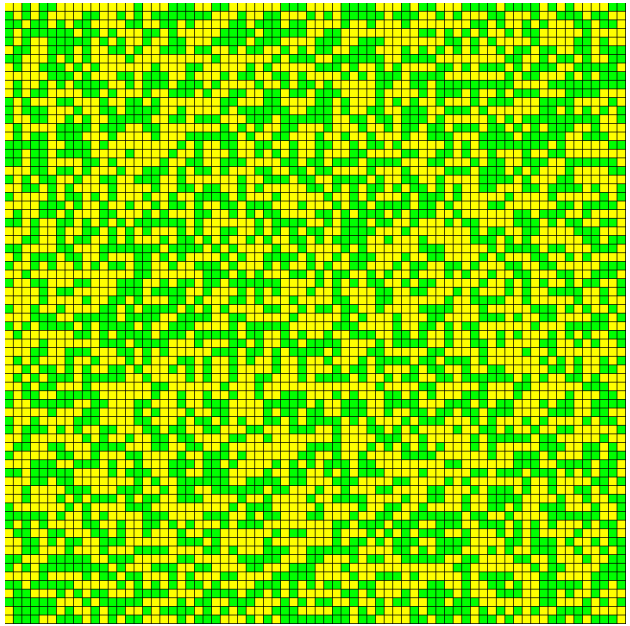
Flipping bits, red
H match, green
L match, yellow
Other, white

DEVICE#2
250 MEASURES

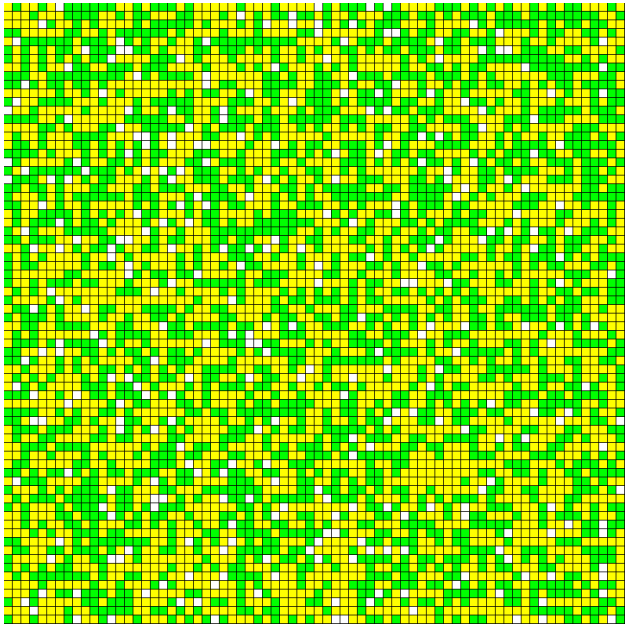
Static Authentication



DEVICE#1, 250 MEASURES



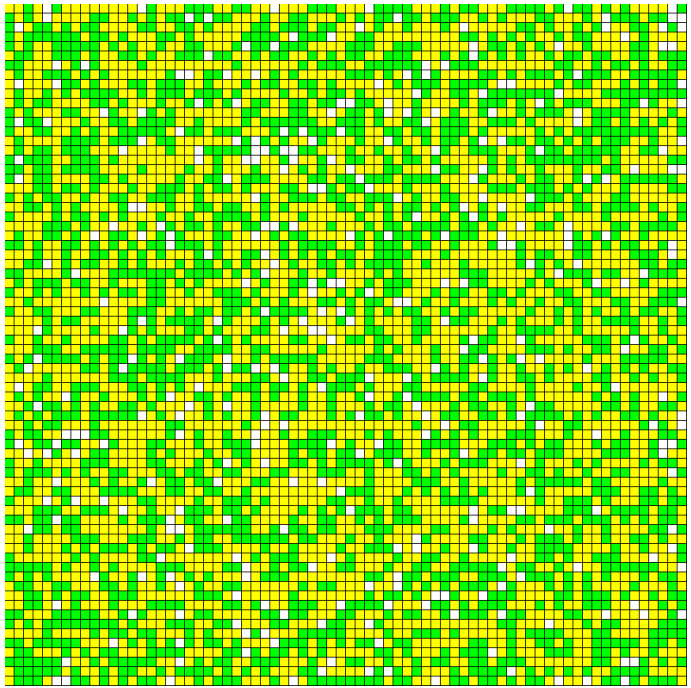
DEVICE#X 1 MEASURE



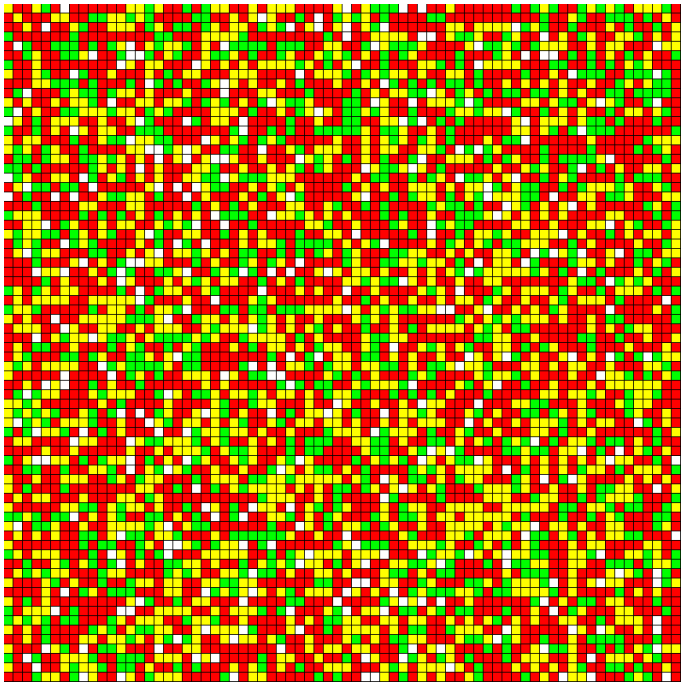
DEVICE#2, 250 MEASURES

Graphical Static Authentication

Flipping bits, red
H match, green
L match, yellow
Other, white



DEVICE#1



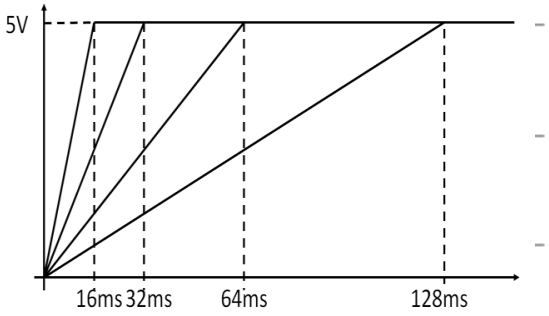
DEVICE#2

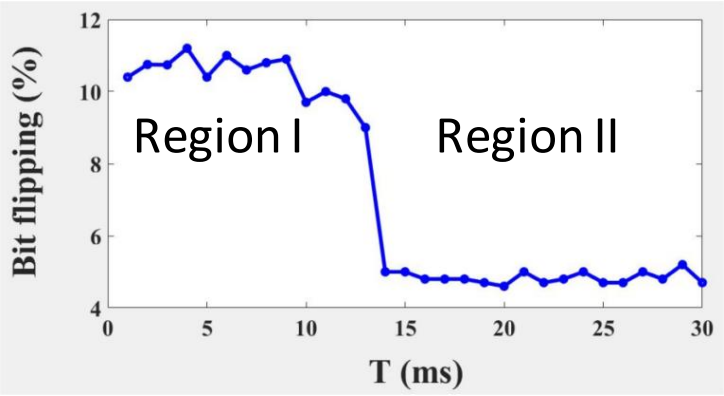
Authentication Algorithm

We note p=1-ε the estimated probability of a memory cell, to get a zero or one value, ε being a small value. We consider two measures $M^{N1}k_1$ and $M^{N2}k_{2,}$ the first being a reference for device k_1, i.e. $N_1 >> 1$.
We take $(1- \varepsilon)^{N1} < 1/N1$, i.e. $\varepsilon < \ln(N1)/N1$, $\ln(250)/250 = 0{,}022$
for example $\varepsilon=10^{-3}$

- for N2 << N1 (for example N2=1)
  - check that $\#D^{k1}_{N1} < \#D^{k2}_{N2}$
  - the number of flipped elements should be less than $\varepsilon^{N2} \times \#D^{k1}_{N1}$
- for N2 ~ N1 (for example N2=N1)
  - check that $| \#D^{k1}_{N1} - \#D^{k2}_{N2} | < \varepsilon \#D^{k1}_{N1}$
  - the number of flipped elements ($\#F^{N1,N2}$ k_1,k_2) should be less than $\varepsilon \times \#D^{k1}_{N1}$.

Number of flipped bits according to the rising time in the common domain, For 250 measures

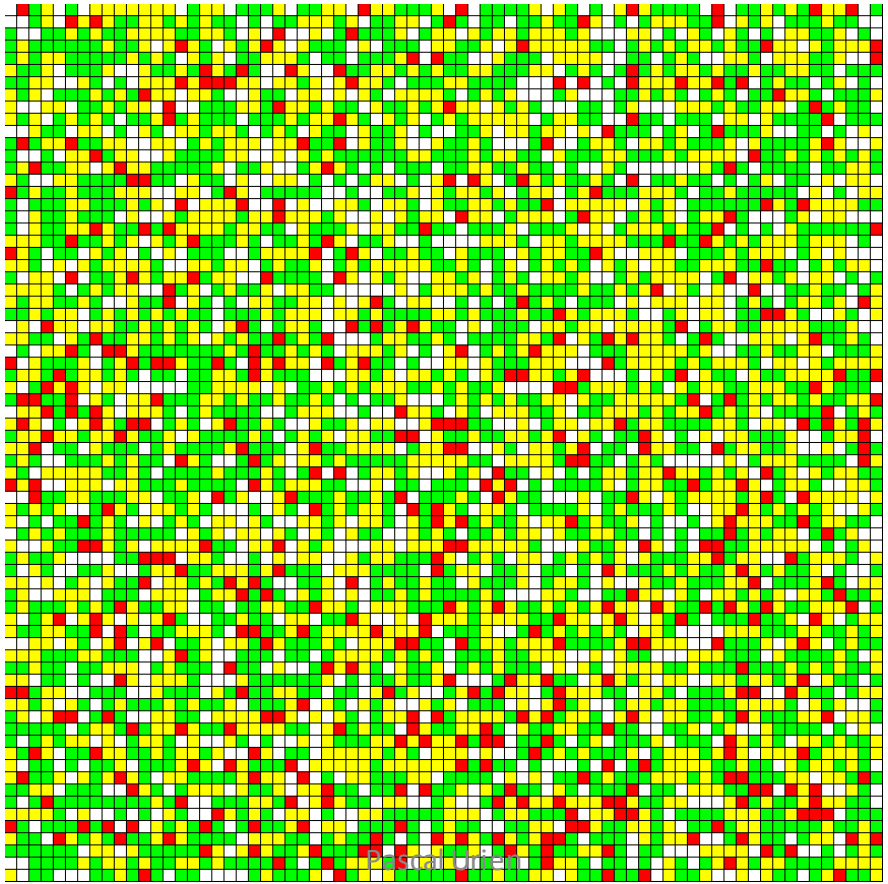




Flipped bits device#1 (250 measures)

Reference 1024 ms
0ms,1ms,2ms,4ms,8ms
, 16ms,32ms,64ms,
128ms,256ms,512ms

Flipping bits, red
H match, green
L match, yellow
Other, white



PUF and BMAC

- PUF bits (located in the SRAM) may be included in the bMAC calculations
  - It is possible to bind BMAC to processor
  - It is possible to define dynamic BMAC if flipping bits are included in BMAC calculation

Device #1 SRAM DUMP

1024 ms x 250

```
0608 .D .. 25 .7 49 .A CA 10 B8 47 .C .B E. C. C2 94
0624 9. 63 .5 B7 5. 0C 86 90 0. .A 44 B. 94 .4 7. BB
0640 40 .5 B. B2 .. 84 44 8. .6 83 65 EA .E A. 6. .4
0656 .2 .C 57 .6 34 02 06 .. 00 .8 83 1. .8 .D 0. 9.
0672 2. 47 .1 E2 .8 62 .. 8. 91 C. .. F3 B8 46 .3 ..
0688 86 22 43 .B 93 6. 87 .3 09 87 .1 C. 7A 4. .7 51
0704 68 5D 19 9B 31 4E AC 90 25 .3 .4 .A .6 .C 4. 38
0720 41 DF 20 66 3C 66 02 7. 54 .5 C1 59 .1 C. 90 0B
0736 4C C2 03 B9 D8 38 .7 B. 85 .A 06 .. .8 6F .B D0
0752 .. 43 A6 0E C. EE 6A D. 80 F. .3 C6 61 28 9. .0
0768 84 E3 34 BF 91 1. 46 97 2C 44 5D 7. 48 C8 4. 2.
0784 0F .9 81 C2 0. 7. 1. 32 .1 CB .1 5F 0A 87 D. 1.
0800 99 .F .4 .F 15 .. 2. 3. C0 E. .C AB 68 66 2C 9.
0816 81 .6 5. .7 2A 44 AC 39 70 4. 40 .B 3. EC D2 .C
0832 88 87 C. F. .. 1D E5 A. E3 .3 3. .8 18 .D .7 .8
0848 .. F5 2F CE 3. .. 00 30 A. E1 10 DA 28 8. 0F B9
0864 A0 C7 F0 FB A. 6. .1 .A 06 26 CB B7 46 8D 87 B.
0880 82 45 2. CE 12 .F FA D. 1C 3. 69 76 C0 A. BE 34
0896 BD 8A .. 8. .E 2A 0F .E D5 D. .1 FF .6 AD .. 7.
0912 .0 D9 D6 7. .. 40 .C .A 0. 45 .D 57 A0 72 7. .C
0928 89 .5 .1 .6 .9 8. 90 65 B1 .6 C. BA 0. E4 75 5.
0944 A0 A. .1 A8 B8 ED 2E 10 A0 C6 .4 .2 .8 B5 A. .6
```

64ms x 250

```
0608 DD 3. 2. 6. .9 EA C. 1. B. 57 .C .B E. C. D3 94
0624 9E 63 .7 B7 .. 0D 8E .0 0. .A 4. .. 9. .E 7. B.
0640 50 A5 .. .2 A8 84 .. .B .6 8B .. EA 7E A1 .. D4
0656 C2 EC 5F .. F. 02 .. .A .. .8 .3 1. B. .D .F .2
0672 24 4F B1 E2 .C .. .9 .. .1 C7 .. .. B. .6 .3 .4
0688 8E 3. .3 9B 93 6. .7 .B 0. .7 .. C2 7A 4. .. 51
0704 78 5D 1B 9. 3. 5E A. .2 6. .3 94 .. .. .C 46 .8
0720 C5 .. .. .6 3. .E .. 7. 5C C5 C. 59 D. D2 .0 0B
0736 .. .2 .. .. D. 38 .7 .. 8. FA 06 .8 78 6F FB D0
0752 9F .. .6 0E C. .E 6A DC 8. F6 .3 .6 .. .. 9. ..
0768 .7 E. 3. .. D. 3. 46 .7 2. 44 5D 7B 4. .. 6. 2.
0784 0F C9 .1 C. .1 .. 1. 2. 2. DB 99 5F .E 83 D. ..
0800 99 DD .. 1F .5 66 .. 3. 65 .. .. A. 68 .2 2C 8.
0816 8. .. 5. 27 .2 4. AC B9 F4 40 E0 .. 2. .C .2 8E
0832 8. .7 C. F9 E1 1D F. E4 .3 7. 3. FC 1. C. 7F .8
0848 CD FD .F 4. 35 .. 2. 36 B. E. .0 .. .. 8. 0F .9
0864 AC C7 .. B. A. .. F0 6. .6 26 CF B7 .6 .F 87 F7
0880 A. .4 2F .. .2 .F F. C. .E 34 7. 7. .0 A. B. 34
0896 FD 8. 82 .. B. 22 0F CE .D D0 19 DF .6 B. 8. ..
0912 E6 D. DE .1 9A .2 6C .A 0E 45 .. .7 E2 7. 71 AC
0928 .. E5 .. A6 .9 .E .. 6. .1 7E .7 B. .. .. .7 50
0944 A. A. 79 A8 B. FD .E 1. A0 C6 7C 72 .C B5 AD .6
```

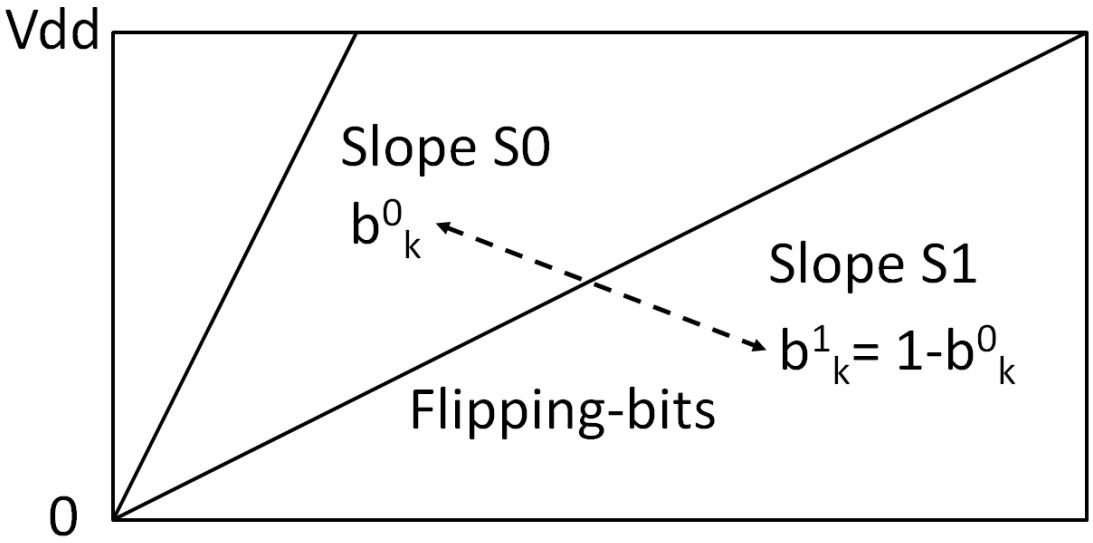
Surrounded : super constant bytes

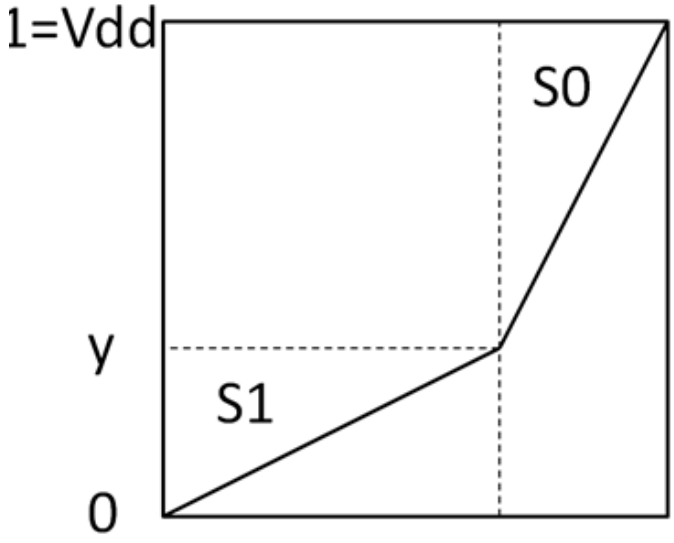
Red : flipping bytes

Dynamic PUF

Sy waveform

- Let's consider a flipping bit and the following power up waveform (Sy)



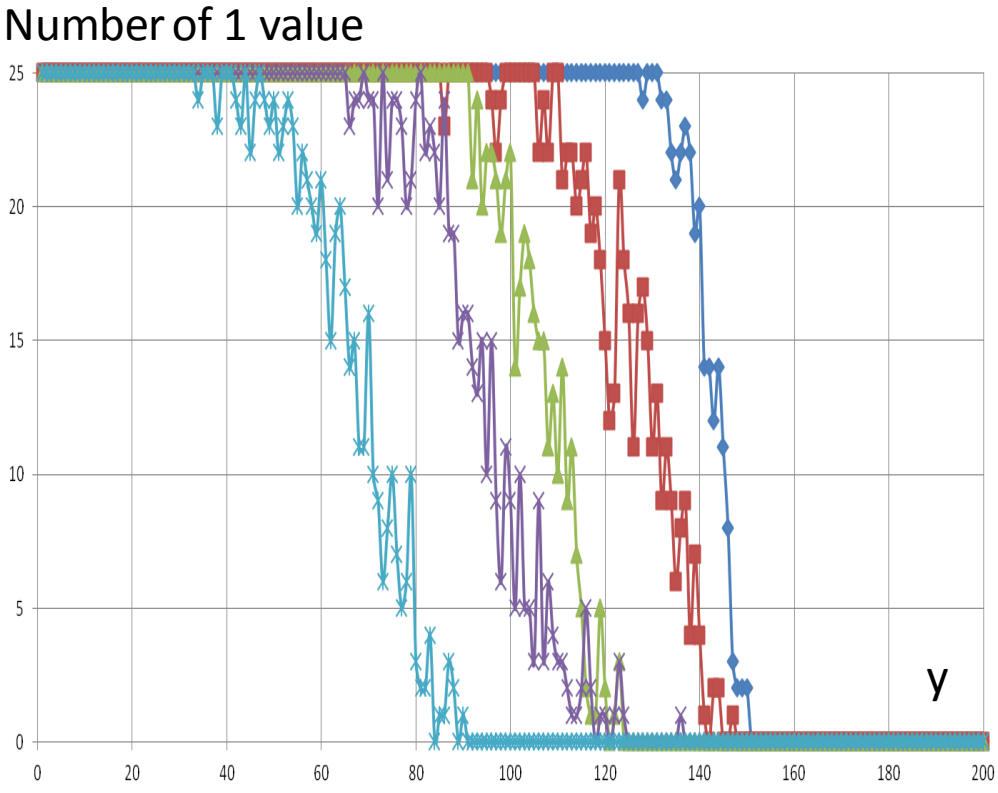


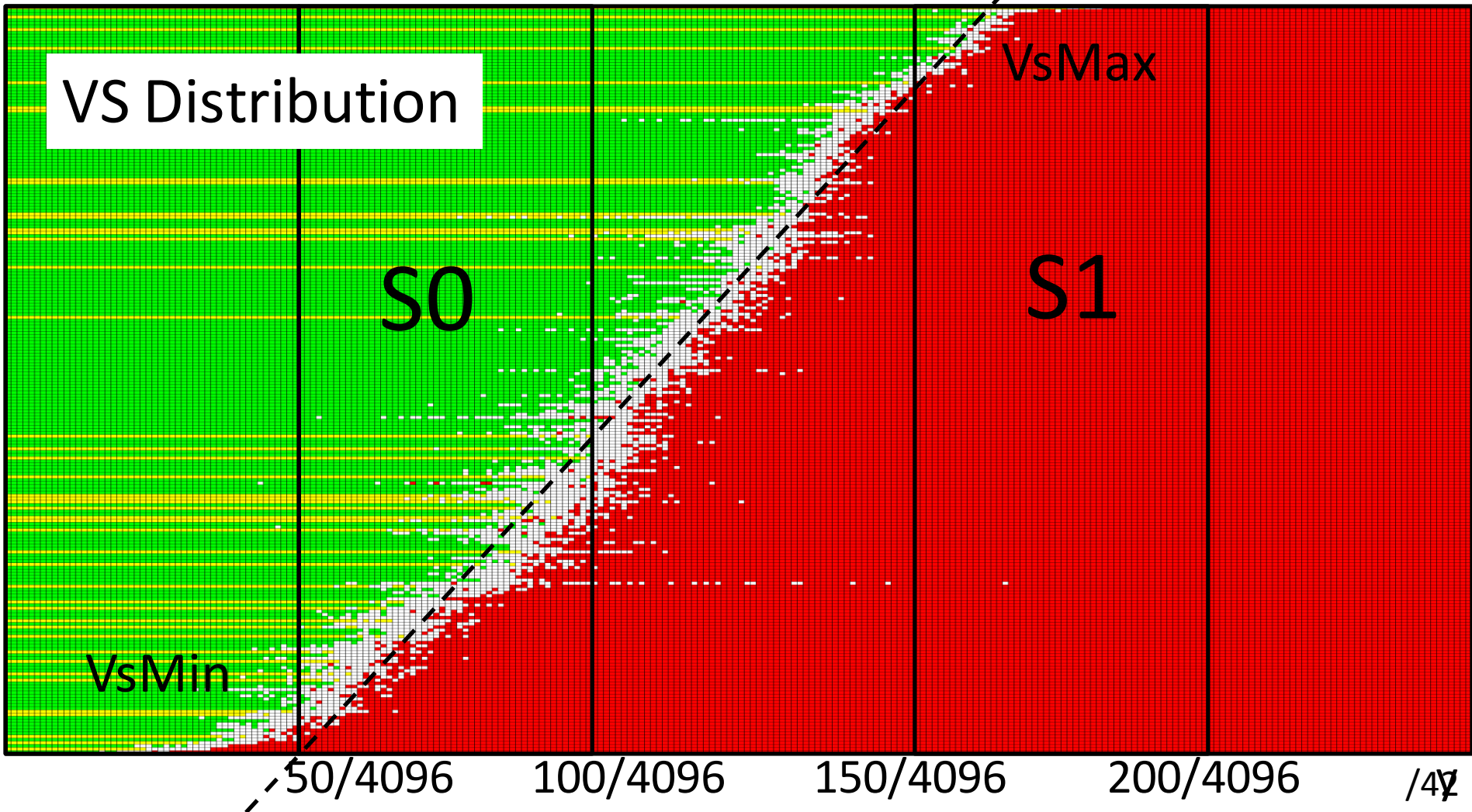
Sy Waveform

- For y=0 (slope S0) we get b^0_k
- For y=1 (slope S1) we get $b^1_k = 1 - b^0_k$
- Therefore it exist a threshold voltage (Vs) Vs=Vdd*y, for which the bit value change.

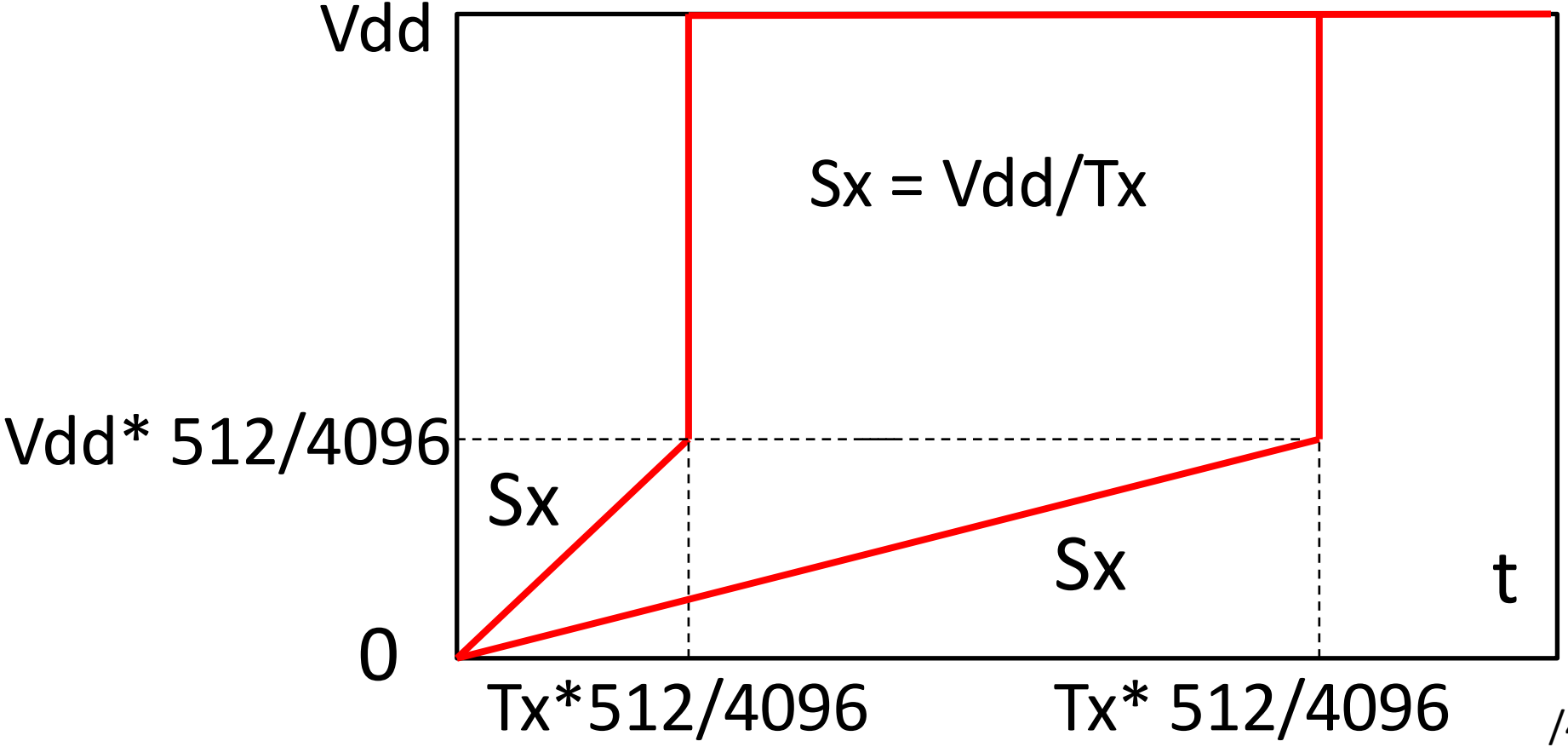
Sy Threshold value distribution

- We performed for each flipping bit, 25 measures, for y parameter ranging for 0/4096 to 250/4096
- Obviously the switching region is noisy.
- The Vs value average is 100/4096x5v



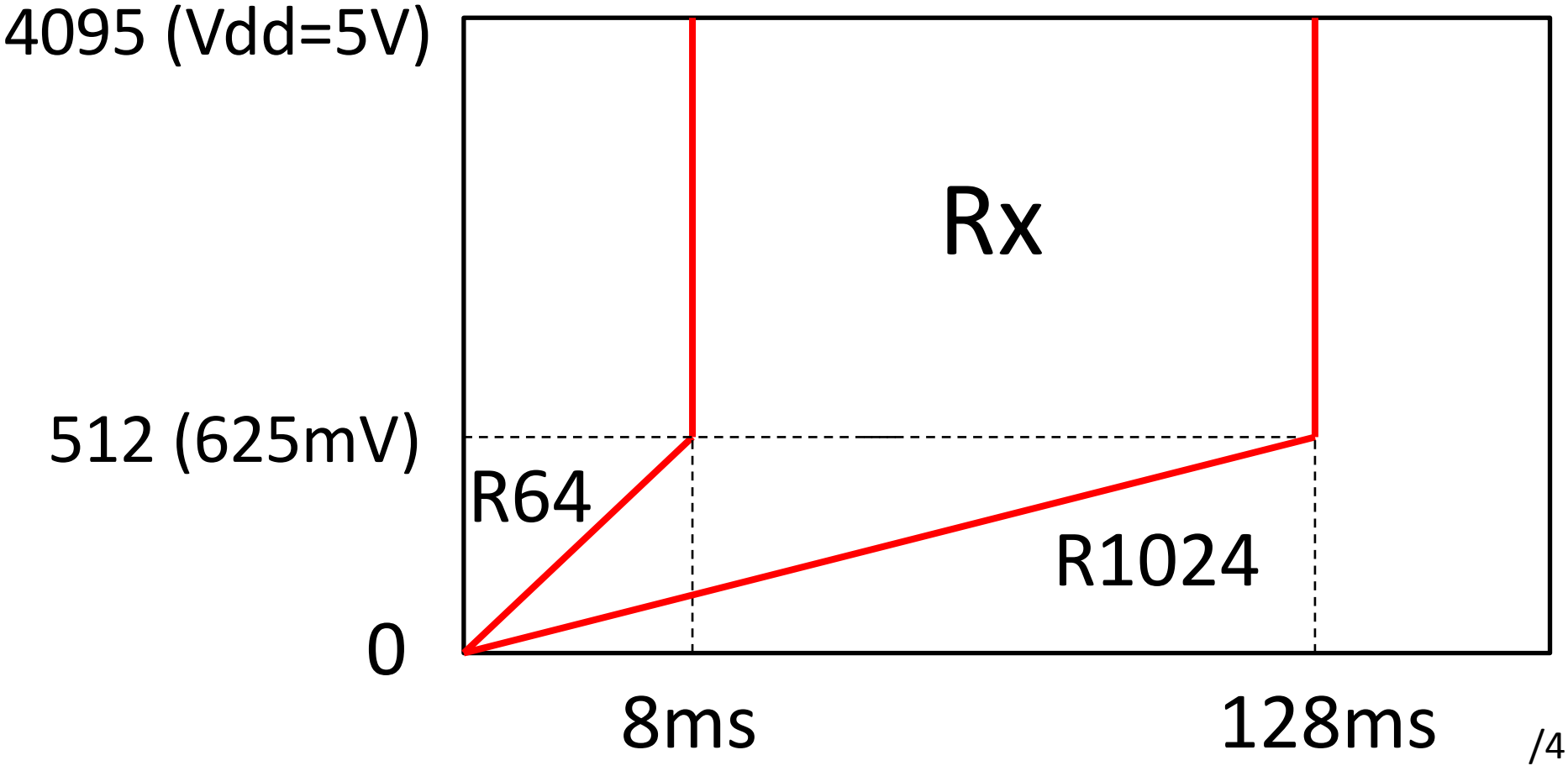

VS Distribution
VsMax
S0
S1
VsMin
50/4096
100/4096
150/4096
200/4096

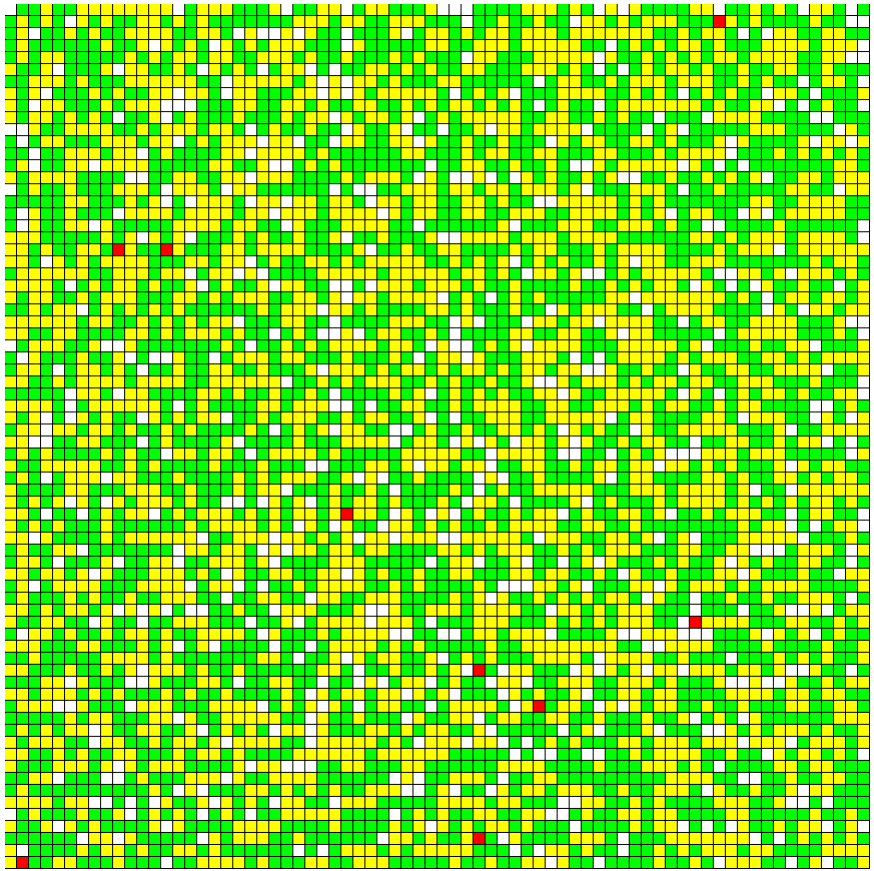
Rx power-up waveform



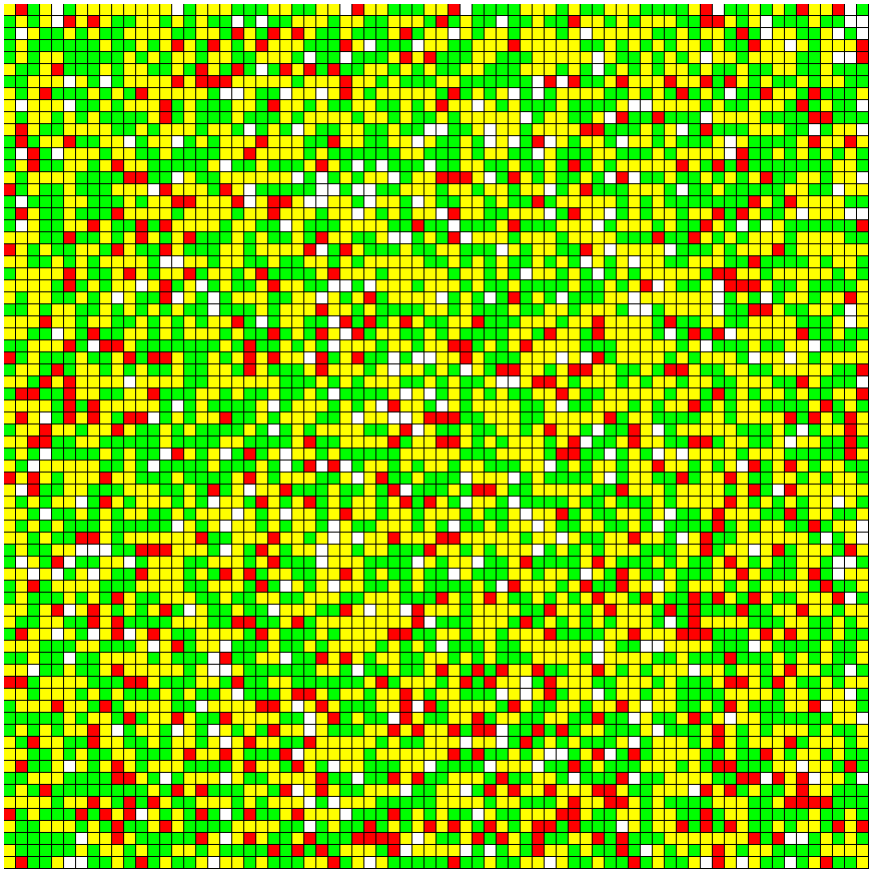
About Rx power-up waveform

- Rx power-up wave form creates flipping bit at low voltage
- The software don’t know the Rx waveform used for power-up

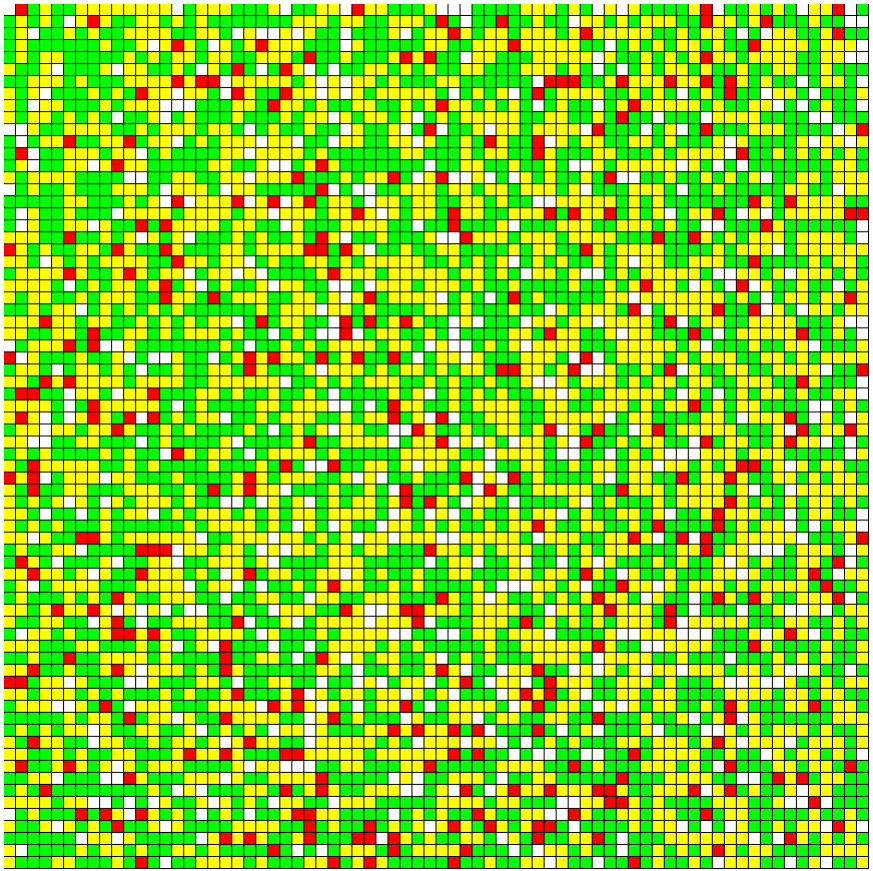

4095 (Vdd=5V)
512 (625mV)
0
Rx
R64
R1024
8ms
128ms



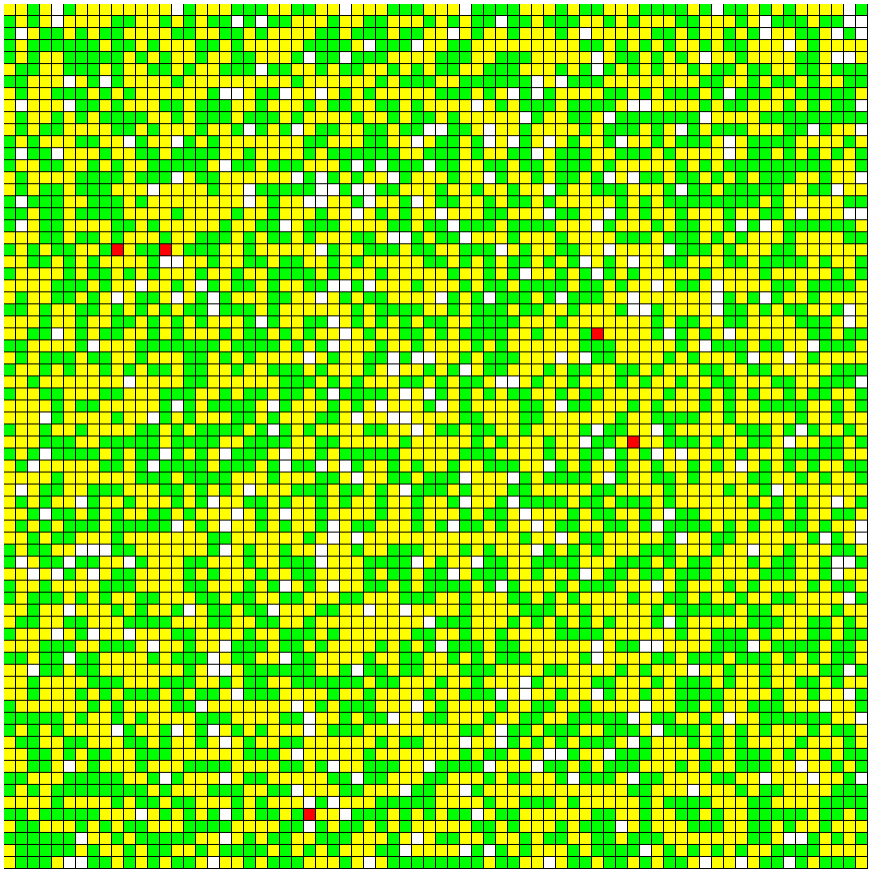
$R^{1}_{64} / S^{250}_{64}$



$R^{1}_{64} / S^{250}_{1024}$



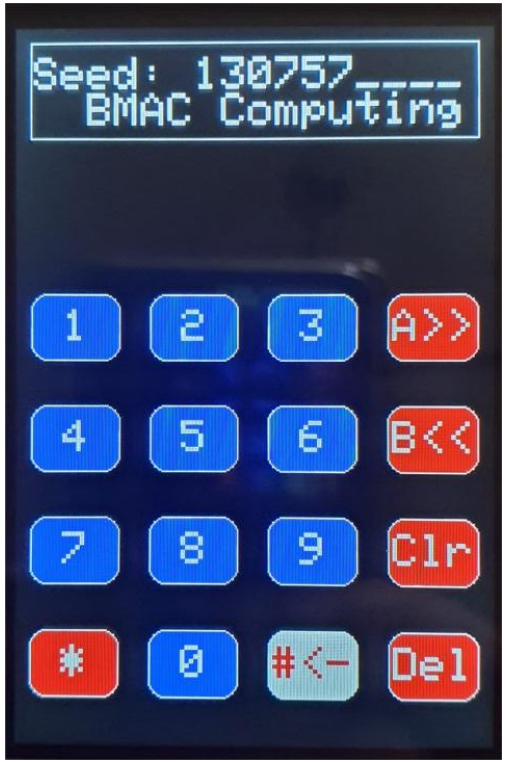
$R^{1}_{1024} / S^{250}_{64}$



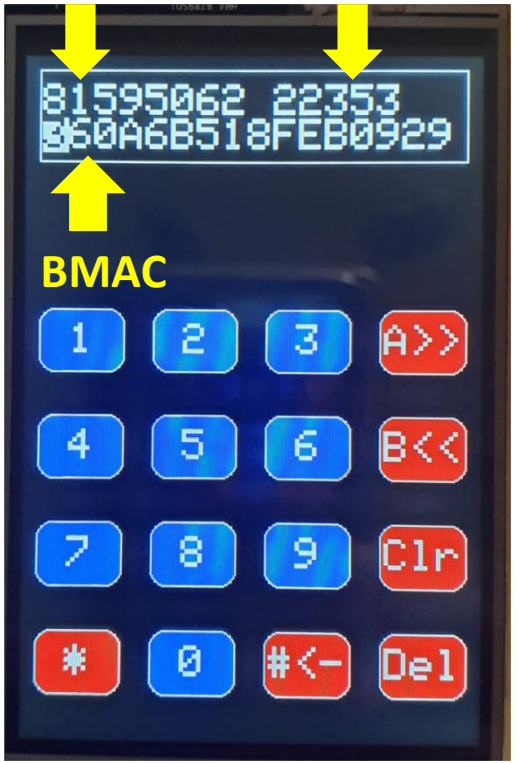
$R^{1}_{1024} / S^{250}_{1024}$

Proposed Dynamic PUF protocol

- Luke has SRAM contents for two Rx power-up waveforms: R64 and R1024.
- The R64 SRAM content has about 200 flipping-bits.
- These contents are determined at low voltage (512mV), before Luke and Vador have a digital life.
- Vador and Leia know these SRAM contents. In order to authenticate Luke, Leia uses power-up waveforms either R64 or R1024, in a random order.
- Luke will always produce the right response, while Vador will make a random choice; so after n tries so probability of zero error for Vador will be 1/2^n…


Seed: 130757____
BMAC Computing
1
2
3
A>>
4
5
6
B<<
7
8
9
Clr
*
0
#<-
Del

Questions ?


COMPUTING TIME
(326.380.248 µs)
TIME STAMPED
BMAC (16 bits)
81595062 22353
960A6B518FEB0929
BMAC
1
2
3
A>>
4
5
6
B<<
7
8
9
Clr
*
0
#<-
Del