PP Smartcard: Midterm Presentation

Team 1

Laurenz Altenmüller
Carlo van Driesten
Markus Hofbauer
Kevin Meyer
Johannes Schreiner

Agenda

1. Organization
   a. Project structure
   b. Reviews
2. DPA
   a. Method
   b. Results
3. Implementation of Card OS + Protocol
   a. Reverse Engineering approach and results
   b. UART Sampling & Transmission strategy
   c. APDU protocol implementation
4. AES Implementation
5. Outlook: Preparations for Hiding Countermeasure

Agenda

1. Organization
   a. Project structure
   b. Reviews
2. DPA
   a. Method
   b. Results
3. Implementation of Card OS + Protocol
   a. Reverse Engineering approach and results
   b. UART Sampling & Transmission strategy
   c. APDU protocol implementation
4. AES Implementation
5. Outlook: Preparations for Hiding Countermeasure

Organization

Teams

- **Attacking** Markus, Kevin
- **Cloning** Laurenz, Carlo, Johannes

Tools

Result: Milestone reached

May 15th, 2015

31 days early

Team Review Sessions

Goals

- Exchanging knowledge
- Fixing bugs
- Developing ideas

Procedure

- Team gets together
- Developer explains code
- Reviewers critically question the implementation
- Blackboard lists of open questions and todos
- Suggestions and information is saved in tickets

Agenda

1. Organization
   a. Project structure
   b. Reviews
2. DPA
   a. Method
   b. Results
3. Implementation of Card OS + Protocol
   a. Reverse Engineering approach and results
   b. UART Sampling & Transmission strategy
   c. APDU protocol implementation
4. AES Implementation
5. Outlook: Preparations for Hiding Countermeasure

Method

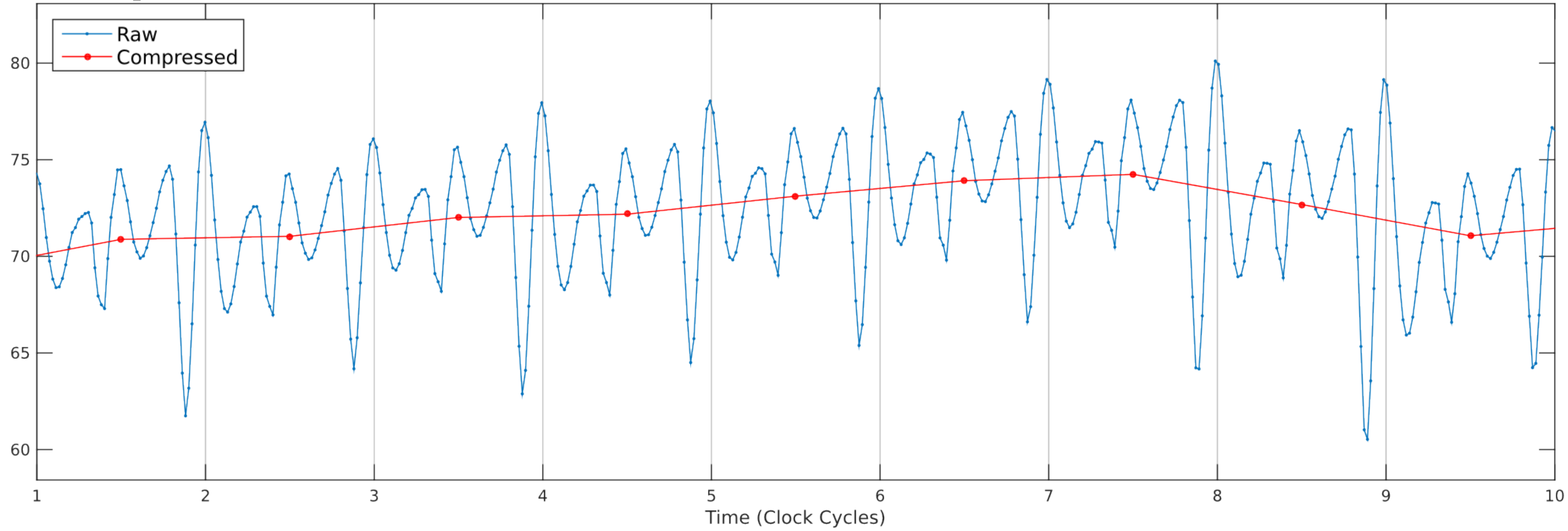
1. **Measurement**
   - AES Recording time: 0.5 ms
   - Sample rate: 250 MHz

2. **Compression**

3. **Differential Power Analysis**

4. **Testing**
   - Test calculated key
   - Using one pair of cipher- and plaintext
   - Decrypt ciphertext with AES-128 and compare result to expected plaintext

Compression



Idea

- Remove redundancy
- Speed up computation
- Reduce noise

Realization

- Take mean of samples for each clock cycle (clock: 4.8 MHz)
- 125,000 → 2,404 samples/trace (reduction by ~98%)

Differential Power Analysis (1)

Idea

- **Intermediate value**: last round's S-Box input

  $$r = \text{s-box}\left(\text{plaintext} \oplus \text{round-key}\right)$$

- Apply **power model**:

  $$H = \text{hamming-weight}(r)$$

- Calculate **Correlation Coefficient**:

  $$R = \text{corr-coef}(H, T)$$

- $\max(R) \Rightarrow \text{round-key}$
- Last decryption round

  $$\Rightarrow \text{round-key} = \text{master-key}$$

Differential Power Analysis (2)

Implementation (1)

- Python (numpy)
- Optimizations:
    - Hamming Weight: Usage of 8-bit lookup table
    - Merging S-Box and Hamming Weight lookup table:

$$H = \text{hamming-weight}\left(\text{s-box}\left(\text{plaintext} \oplus \text{round-key}\right)\right)$$

$$\Rightarrow \text{hamming\_s-box} = \text{hamming-weight}\left(\text{s-box}\right)$$

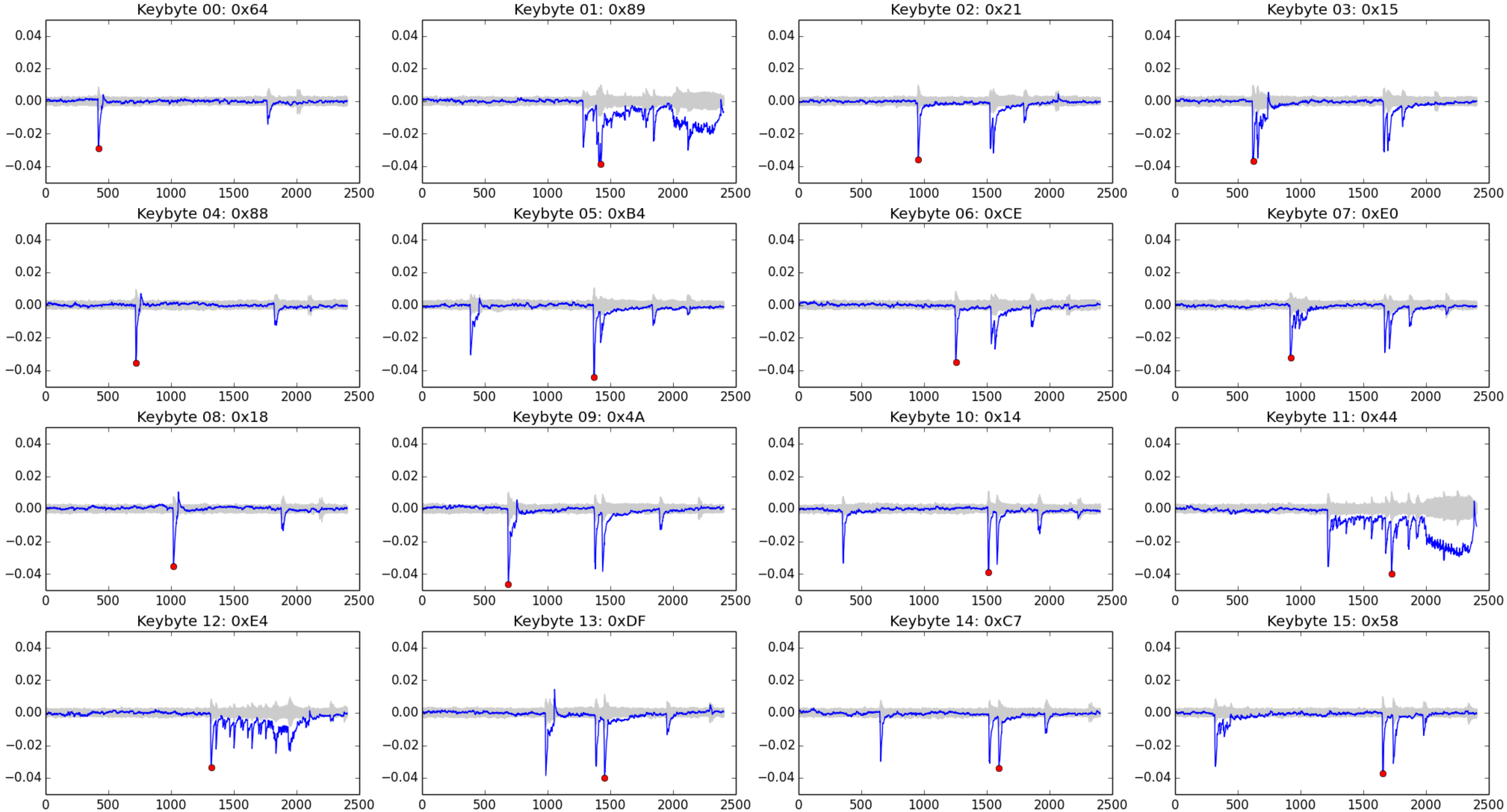
    - Pre-calculations for Correlation Coefficient:
        - $T_{\text{diff}} = T - \overline{T}$
        - $\sum {T_{\text{diff}}}^2$

Differential Power Analysis (3)

Implementation (2)

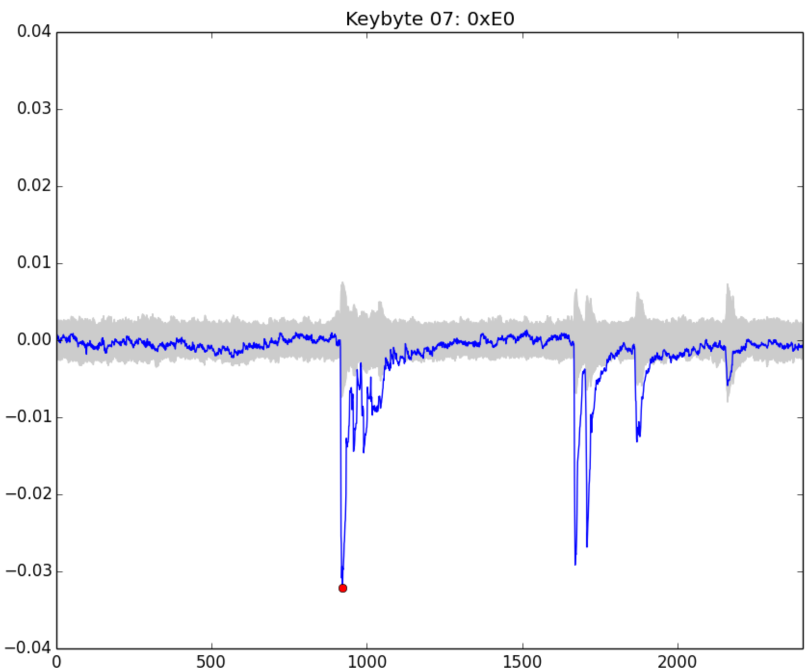
- Calculations for each byte:
  - Calculation of hypothetical intermediate values
  - Calculation of Correlation Coefficient (CC)
  - $H_{\text{diff}} = H - \overline{H}$
  - $CC = \dfrac{T_{\text{diff}} \cdot H_{\text{diff}}}{\sqrt{\sum {T_{\text{diff}}}^2 \cdot \sum {H_{\text{diff}}}^2}}$
  - Comparing the hypothetical power consumption values with the power traces (CC)

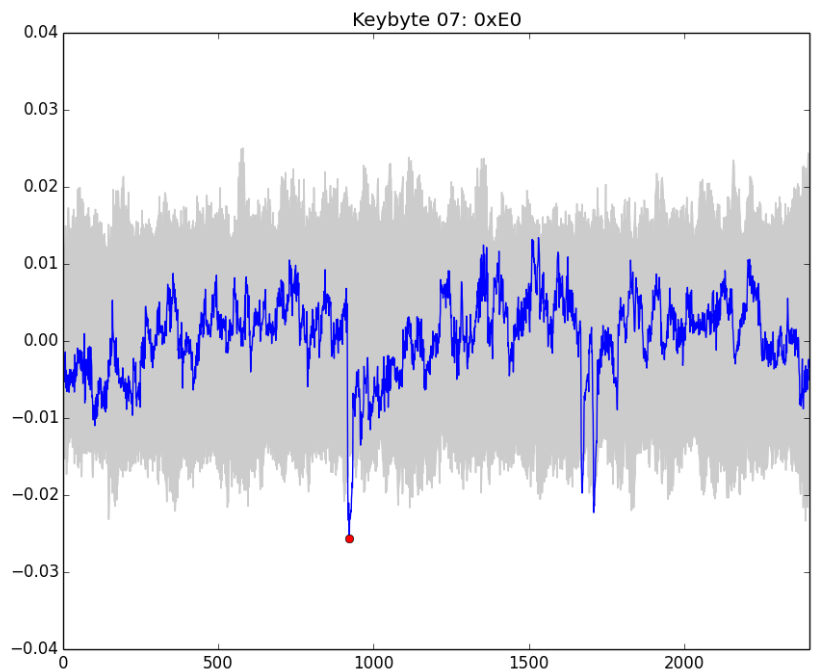
Results (1)


Keybyte 00: 0x64
Keybyte 01: 0x89
Keybyte 02: 0x21
Keybyte 03: 0x15
Keybyte 04: 0x88
Keybyte 05: 0xB4
Keybyte 06: 0xCE
Keybyte 07: 0xE0
Keybyte 08: 0x18
Keybyte 09: 0x4A
Keybyte 10: 0x14
Keybyte 11: 0x44
Keybyte 12: 0xE4
Keybyte 13: 0xDF
Keybyte 14: 0xC7
Keybyte 15: 0x58
0.04
0.02
0.00
−0.02
−0.04
0
500
1000
1500
2000
2500

Results (2)

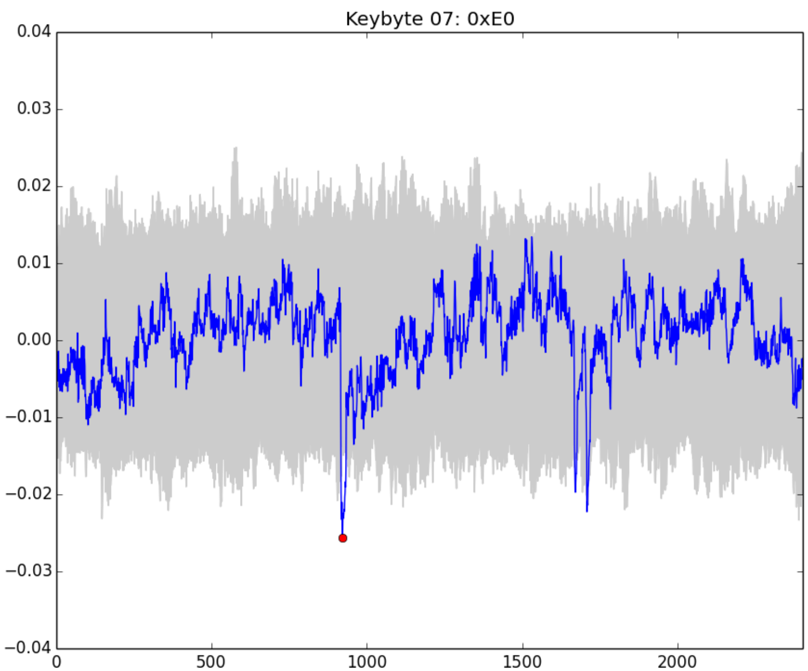
# traces	5500	130
# samples	2,404 (1 per cycle)	2,404 (1 per cycle)

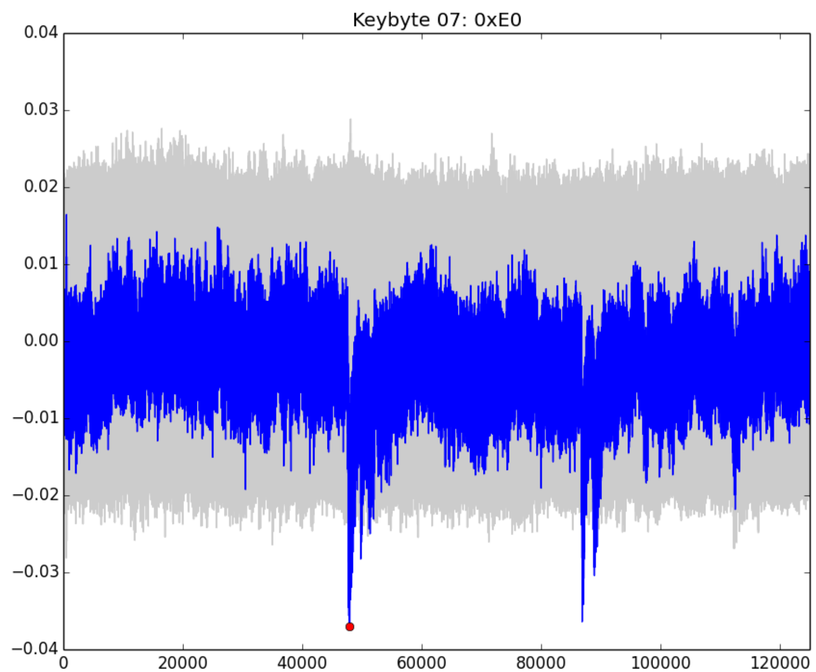




Results (3)

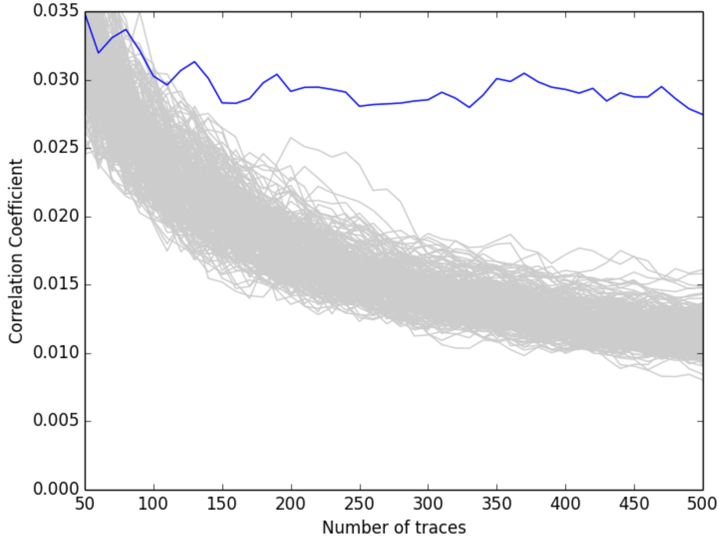
# traces	130	130
# samples	**2,404 (1 per cycle)**	**125,000 (52 per cycle)**





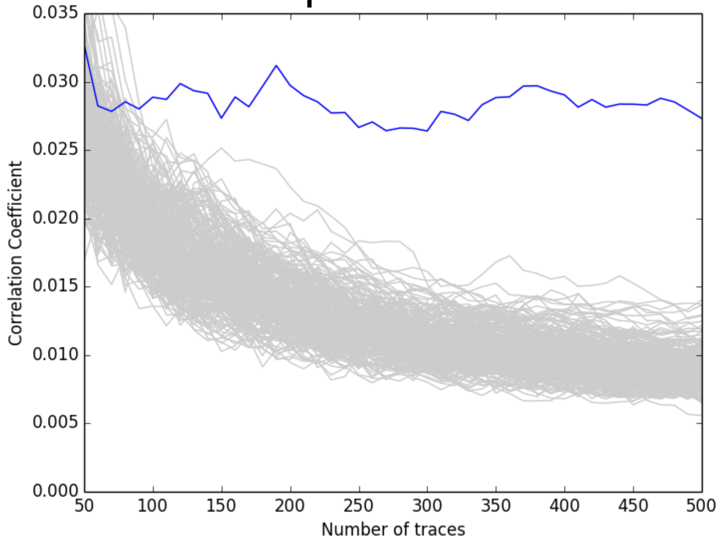
Performance

Raw Data



# of traces	time (per byte)	success rate
100	32.68s (2.04s)	0%
120	36.58s (2.29s)	12%
150	39.02s (2.44s)	41%
200	48.04s (3.00s)	93%

Compressed Data

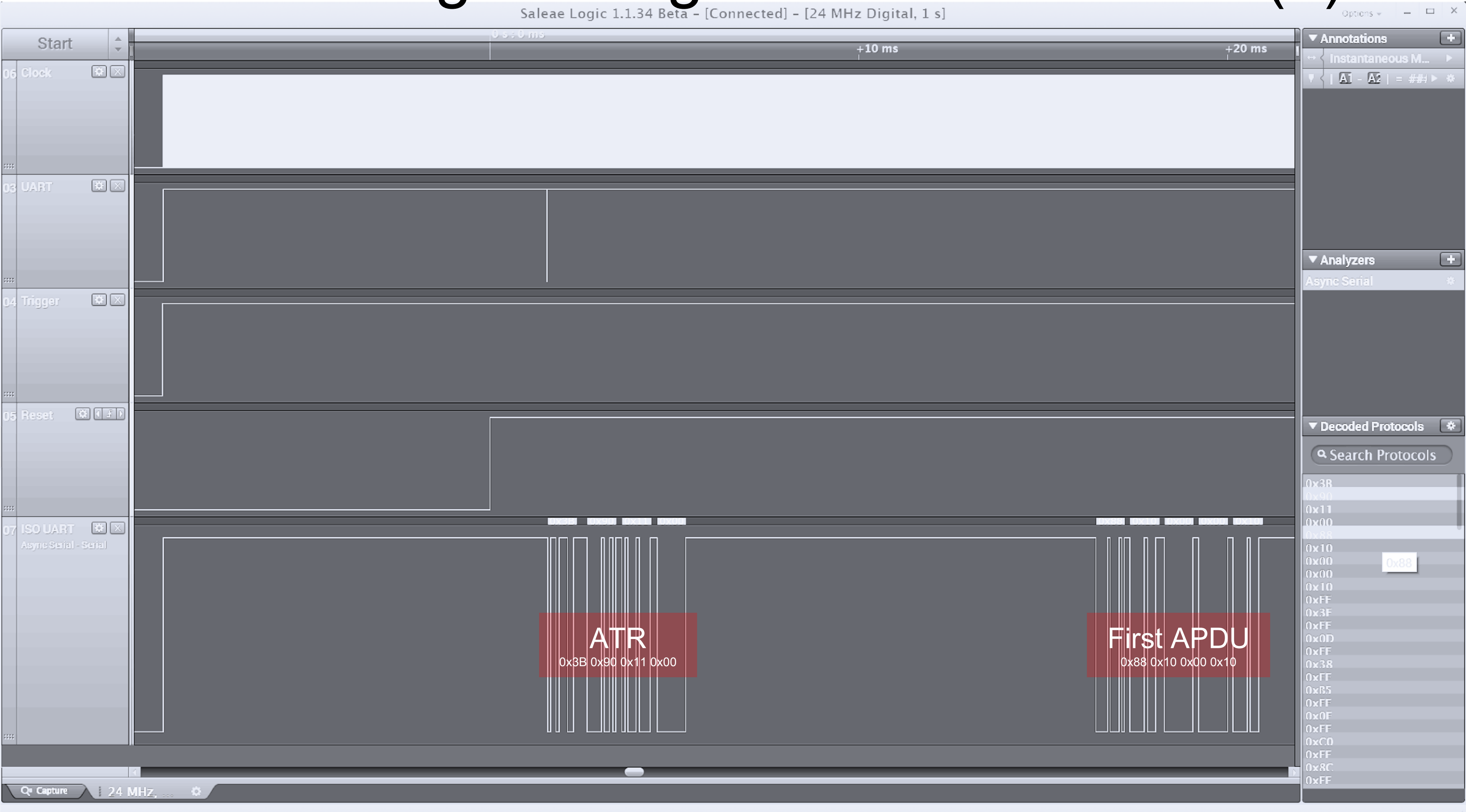


# of traces	time (per byte)	success rate
100	0.49s (0.03s)	18%
120	0.54s (0.03s)	55%
150	0.67s (0.04s)	87%
200	0.85s (0.05s)	99%

Agenda

1. Organization
   a. Project structure
   b. Reviews
2. DPA
   a. Method
   b. Results
3. Implementation of Card OS + Protocol
   a. Reverse Engineering approach and results
   b. UART Sampling & Transmission strategy
   c. APDU protocol implementation
4. AES Implementation
5. Outlook: Preparations for Hiding Countermeasure

Reverse Engineering of Communication (1)



Reverse Engineering of Communication (2)

Smartcard

Terminal

Data	***Interpretation***
0x3B	**ATR** (High Z / Active Low, LSB first)
0x90	**ATR** (enable TA1, TD1)
0x11	**ATR** (ETU = 372)
0x00	**ATR** (T=0 protocol)
new instruction...	
0x88	Class Byte
0x10	Instruction (**DECRYPT BLOCK**)
0x00	Param1 (unused)
0x00	Param2 (unused)
0x10	Param3 (number of bytes)
0xEF	Ready to receive byte
??	Encrypted block (challenge) byte 1
0xEF	Ready to receive byte
??	Encrypted block (challenge) byte 2
0xEF	Ready to receive byte
...	
??	Encrypted block (challenge) byte 15
0xEF	Ready to receive byte
??	Encrypted block (challenge) byte 16
0x61	Status (success, pending response)
0x10	Status (number of response bytes)

(continued)	
new instruction...	
0x88	Class Byte
0xC0	Instruction (**GET RESPONSE**)
0x00	Param1 (unused)
0x00	Param2 (unused)
0x10	Param3 (number of bytes)
0xC0	ACK
??	Decrypted block (response) byte 1
??	Decrypted block (response) byte 2
...	
??	Decrypted block (response) byte 15
??	Decrypted block (response) byte 16
0x90	Status (success)
0x00	Status (success)
new instruction...	
0x88	Class Byte
0x10	Instruction (**DECRYPT BLOCK**)
0x00	Param1 (unused)
0x00	Param2 (unused)
0x10	Param3 (number of bytes)
0xEF	Ready to receive byte
??	Encrypted block (challenge) byte 1
...	

APDU Protocol implementation (ISO7816-4)

Main Program

```
send_atr()
while true:
    get_apdu_DECRYPT_BLOCK()
    aes_decrypt()
    get_apdu_GET_RESPONSE()
```

Assumption

Terminal alternatingly issues `DECRYPT_BLOCK`, `GET_RESPONSE` requests with known params

Allows

Implementing only needed parts of the APDU protocol

Drawback

Unsupported APDUs lead to incorrect behaviour

→ Unexpected bytes trigger debug output on UART

```c
#define SIZE(a) sizeof(a)/sizeof(a[0])

void get_apdu_DECRYPT_BLOCK(uint8_t* auth_challenge)
{
    const uint8_t expected_request[] = {0x88, 0x10, 0x00, 0x00, 0x10};
    receive_expected(expected_request, SIZE(expected_request));

    for(uint8_t i = 0; i < 16; i++)
    {
        transmit_byte(0xEF);
        auth_challenge[i] = comm_receive_byte();
    }

    const uint8_t response[] = {0x90, 0x00};
    transmit_bytes(response, SIZE(response));
}

void get_apdu_GET_RESPONSE(uint8_t* auth_response)
{
    const uint8_t expected_request[] = {0x88, 0xC0, 0x00, 0x00, 0x10};
    receive_expected(expected_request, SIZE(expected_request));

    transmit_byte(0xC0);
    transmit_bytes(auth_response, 16);

    const uint8_t response[] = {0x90, 0x00};
    transmit_bytes(response, SIZE(response));
}
```

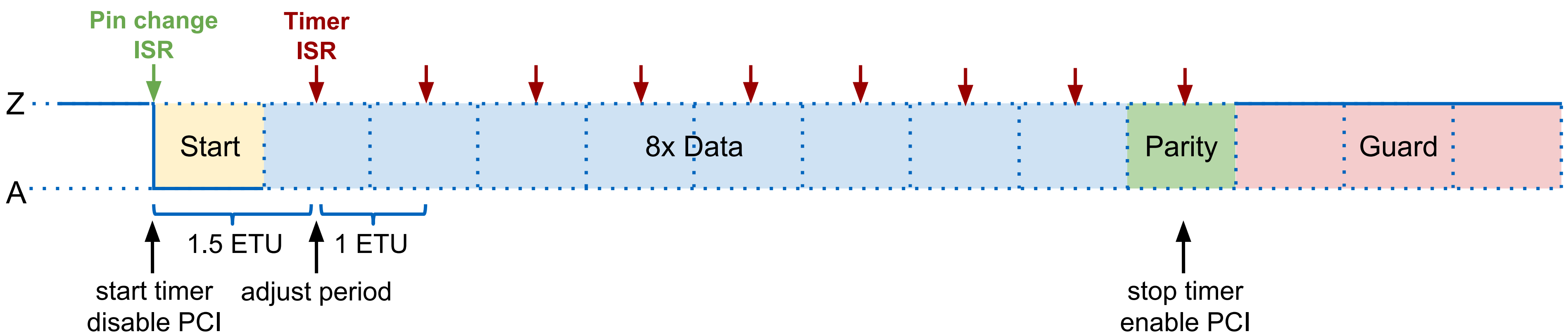
UART Sampling Strategy (ISO7816-3)

Key challenges for precise sampling

- Detection of start bit
- Precise sampling every 372 cycles (= 1 ETU) at middle of bit

Solution

- Detect start bit via **pin change interrupt** (PCI) during idle
  → No busy-waiting/polling
- Pin change interrupt sets up **timer interrupt**
  - Triggers 9 times for every received byte
  - Reception of parity bit disables timer, re-enables PCI
  → Precise, equidistant sampling points



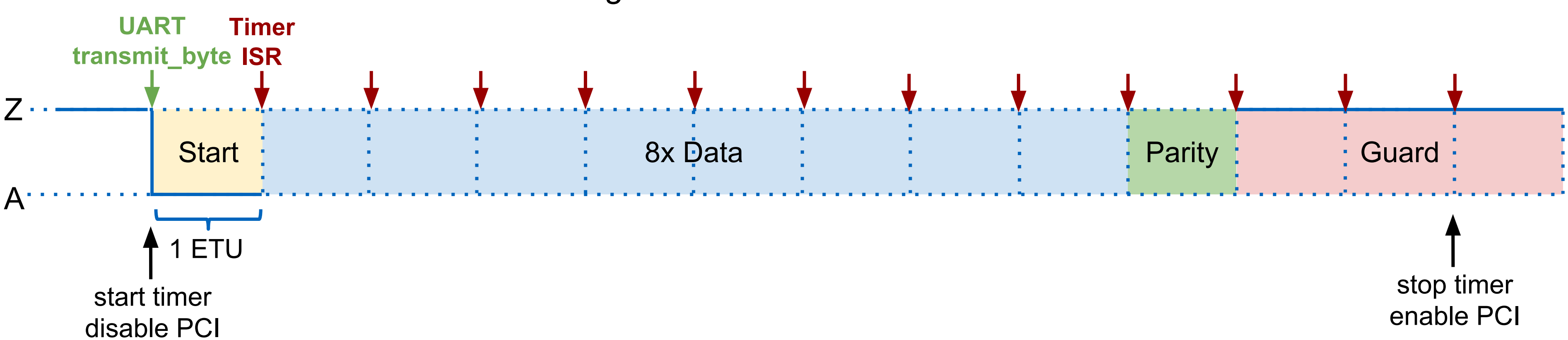
UART Transmission Strategy (ISO7816-3)

Key challenges for transmission

- Avoid conflicts with UART receiver pin change interrupt
- Precise bit change every 372 cycles (= 1 ETU)

Solution

- transmit_byte
  - disables pin change interrupt
  - sets start bit level to low
  - enables timer interrupt
- Timer ISR
  - changes data line level
  - reenables PCI after last guard bit



Agenda

1. Organization
   a. Project structure
   b. Reviews
2. DPA
   a. Method
   b. Results
3. Implementation of Card OS + Protocol
   a. Reverse Engineering approach and results
   b. UART Sampling & Transmission strategy
   c. APDU protocol implementation
4. **AES Implementation**
5. Outlook: Preparations for Hiding Countermeasure

AES Implementation

Procedure

- Research existing implementations
  - Gather ideas
  - Compare implementations
  - Get deep knowledge of AES
- Decision against C++ AES
  - Less overhead

→ **AVR Crypto Lib**

Benefits

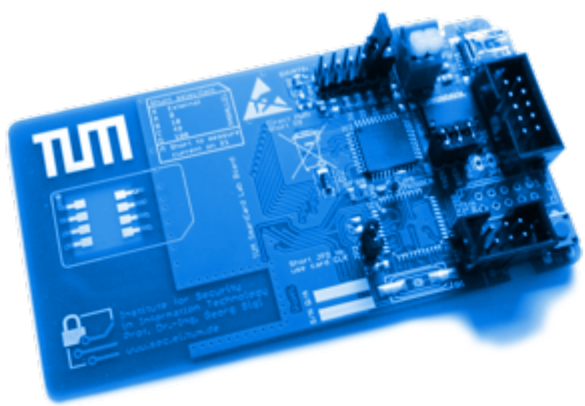
- Lucid
- Benchmarks available
- Optimized for AVR
- Galois Field multiplication in Assembler (fast)

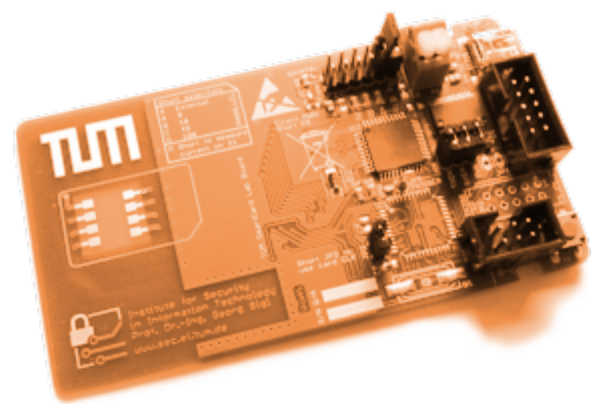
Modifications

- Removed
  - 198/256 bit decryption
  - encryption
  - generic function calls
- Added detailed comments

Clone Comparison

	Original	**Clone** *(no extra features)*	
Total code size	*unknown*	4064 B	**6.2%** used
Total data size	*unknown*	300 B	**7.3%** used
AES execution time	**4.604 ms**	**4.505 ms**	2.15% faster
AES code size	*unknown*	1816 B	45% of code
AES data size	*unknown*	192 B	64% of data





Agenda

1. Organization
   a. Project structure
   b. Reviews
2. DPA
   a. Method
   b. Results
3. Implementation of Card OS + Protocol
   a. Reverse Engineering approach and results
   b. UART Sampling strategy
   c. APDU protocol implementation
4. AES Implementation
5. **Outlook: Preparations for Hiding Countermeasure**

Preparations for Hiding Countermeasure (1)

Hiding Countermeasures require **random** noise, delays, etc.

Problem	No HW-RNG (no source of **entropy**)
Solution	1) Exploit **Jitter** between main clock and WDT for entropy generation 2) Seed Software-implemented CSPRNG
Problem	Entropy must be available shortly after reset, but WDT-entropy **generation** takes ~0.5s
Solution	1) **Seed** PRNG from EEPROM 2) Overwrite EEPROM with first new pseudo-random
Problem	Entropy values in EEPROM must be hard to predict
Solution	1) Overwrite EEPROM with first WDT-entropy byte 2) **Reseed** PRNG with newly available WDT-entropy bytes

Preparations for Hiding Countermeasure (2)

We want many interruptions and delays during AES

Problem	Large amount of cryptographically secure RNG is needed Little computational power available (PRNG takes 0.3ms)
Solution	1) Use **idle time** to compute pseudo-randoms 2) **Buffer** the results in SRAM

Thank you!