

Exploration of Cache-Based Side-Channel Attacks in FPGA SoCs

Matteo Oldani

Under the supervision of

Dina Mahmoud
Dr. Mirjana Stojilović

19/01/2023

Introduction

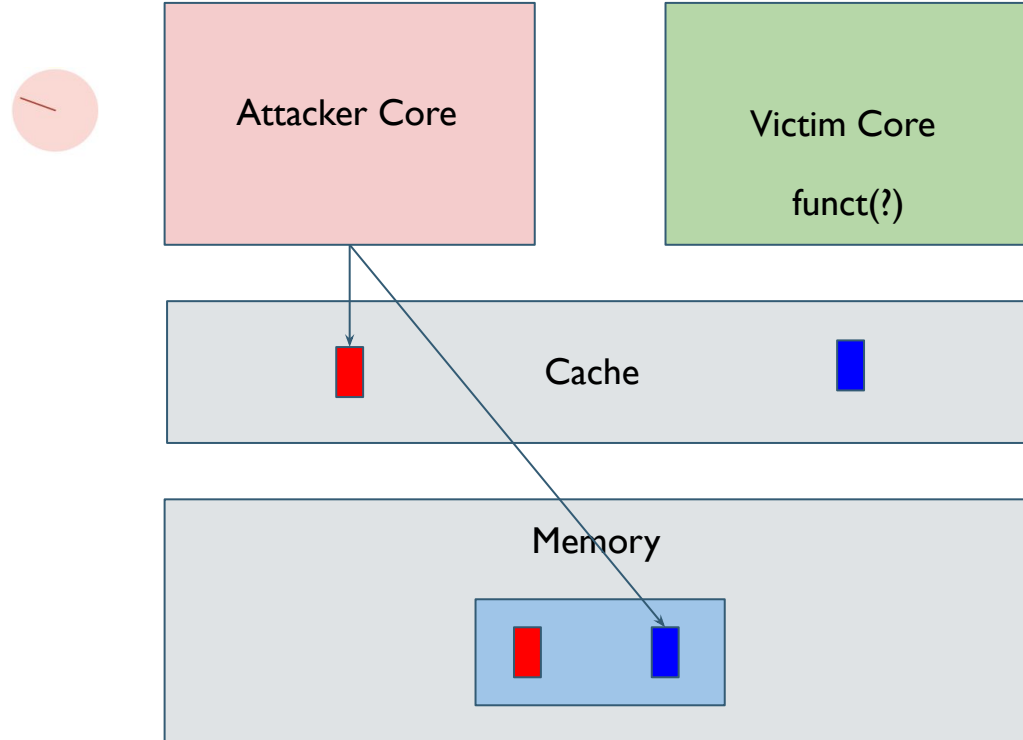
- Increasing demand for specialized hardware
- High level of parallelism
- Application in industry
 - Healthcare
 - Automotive
 - Aerospace




Roadmap


- Background
 - Cache attacks
 - AES T-Table implementation
- Experimental Setup
- Threat Models
- Exploration
- Conclusions

Cache Attacks



```
func(secret_bit)
{
    if(secret_bit == 1)
        *addr1;
    else
        *addr2;
}
```

 addr1

 addr2

Requirements:

- Ability to time access
- Ability to control the cache
 - Flush a cache line
 - Evict a cache line
- Shared memory

Types of Cache Attacks:

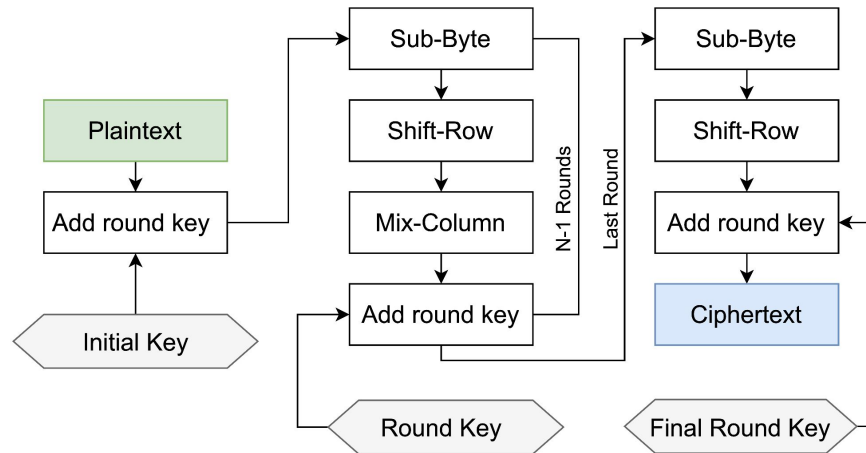
- FLUSH + RELOAD
- EVICT + RELOAD
- FLUSH + FLUSH
- PRIME + PROBE

AES T-Table Implementation

AES Steps:

- Add round key
- Sub-Byte
- Shift-Row
- Mix-Column

AES 128 → 10 rounds



AES T-Table Implementation

- T-Table is an optimized implementation of AES
- AES steps are done using transformation tables
- Each round is composed of 16 lookups
- T-Table implementations of AES are vulnerable to side-channel attacks

AES T-Table Implementation

AES encryption is done as follow:

$$\begin{aligned}(x_0^{r+1}, x_1^{r+1}, x_2^{r+1}, x_3^{r+1}) &\leftarrow Te_0[x_0^r] \oplus Te_1[x_5^r] \oplus Te_2[x_{10}^r] \oplus Te_3[x_{15}^r] \oplus k_0^{r+1}, \\(x_4^{r+1}, x_5^{r+1}, x_6^{r+1}, x_7^{r+1}) &\leftarrow Te_0[x_4^r] \oplus Te_1[x_9^r] \oplus Te_2[x_{14}^r] \oplus Te_3[x_3^r] \oplus k_1^{r+1}, \\(x_8^{r+1}, x_9^{r+1}, x_{10}^{r+1}, x_{11}^{r+1}) &\leftarrow Te_0[x_8^r] \oplus Te_1[x_{13}^r] \oplus Te_2[x_2^r] \oplus Te_3[x_7^r] \oplus k_2^{r+1}, \\(x_{12}^{r+1}, x_{13}^{r+1}, x_{14}^{r+1}, x_{15}^{r+1}) &\leftarrow Te_0[x_{12}^r] \oplus Te_1[x_1^r] \oplus Te_2[x_6^r] \oplus Te_3[x_{11}^r] \oplus k_3^{r+1}.\end{aligned}$$

where Te_i are the transformation tables used for encryption and:

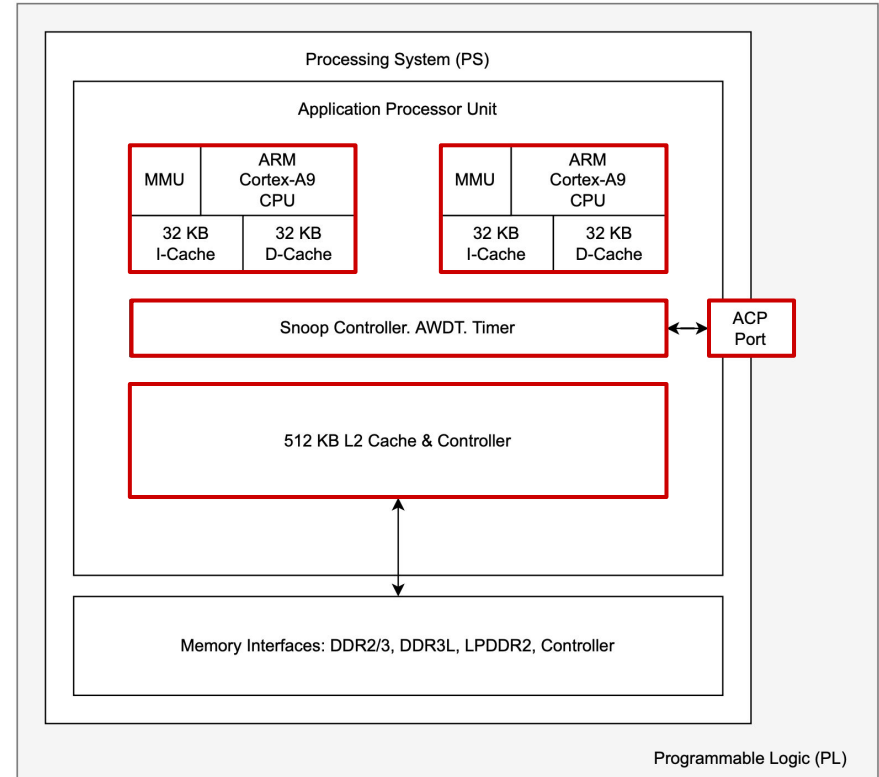
$$x_i^0 = Plaintext_i \oplus key_i.$$

Roadmap

- Background
- Experimental Setup
- Threat Models
- Exploration
- Conclusions

Experimental Setup

- Zedboard Development Board
- AMD Xilinx Zynq®-7000 SoC
- Non-inclusive last-level cache (L2)



AXI Protocol

Versions

- AXI4
- AXI4-Lite
- AXI4-Stream

Channels

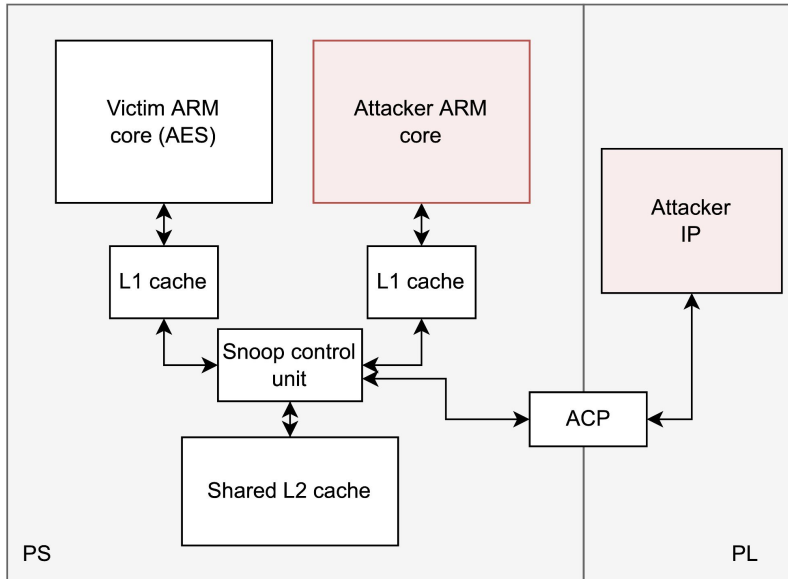
- Read Address Channel
- Read Data Channel
- Write Address Channel
- Write Data Channel
- Write Response Channel

Roadmap

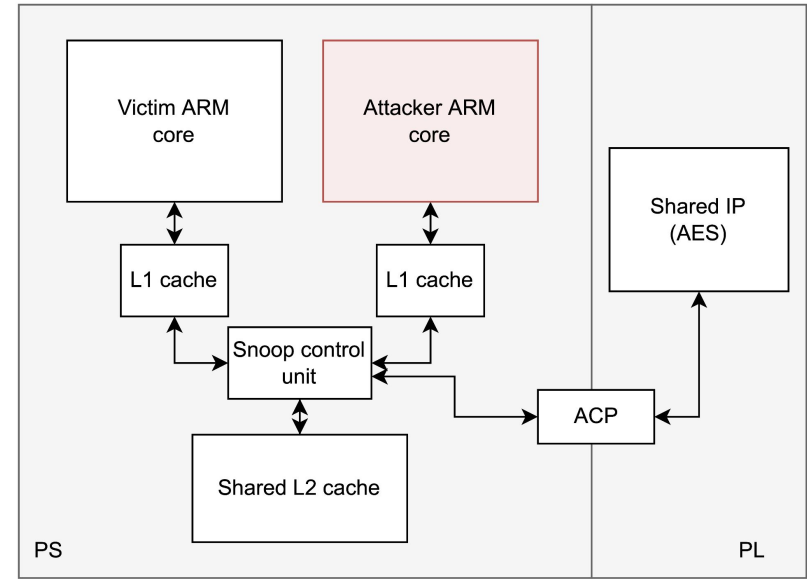
- Background
- Experimental Setup
- Threat Models
- Exploration
- Conclusions

Threat Models

FPGA-to-CPU



CPU-to-FPGA



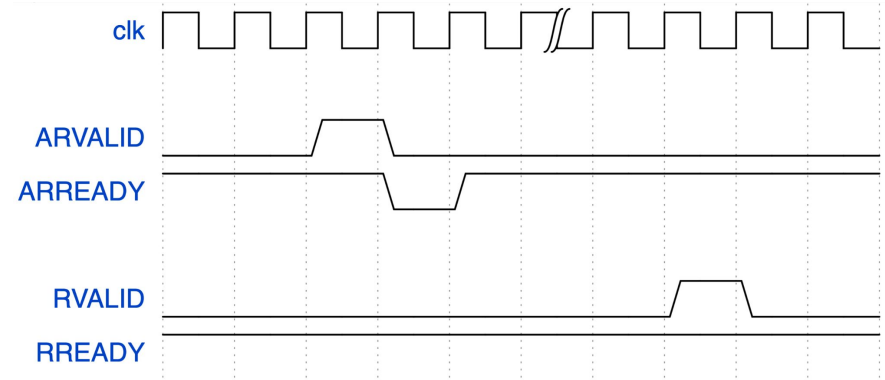
Roadmap

- Background
- Experimental Setup
- Threat Models
- Exploration
 - Read side-channel
 - Write side-channel
 - CPU to FPGA
- Conclusions

Timing Difference

The AXI Read handshake length depends on the position of data in the cache as shown by Bossuet et al.*

- Data in L1 cache → 12 cycles
- Data in L2 cache → 13 cycles



*Bossuet et al. "Performing Cache Timing Attacks from the Reconfigurable Part of a Heterogeneous SoC - An Experimental Study", Applied Sciences 2021

Eviction

The WSTRB signal controls which byte is valid when a write transaction is issued from the FPGA side to the CPU one.

A cache line is invalidated from L1 if a write is issued to the same address.

To evict a cache line from L1 without changing its value, we can issue a write transaction with the WSTRB to b'0000'.

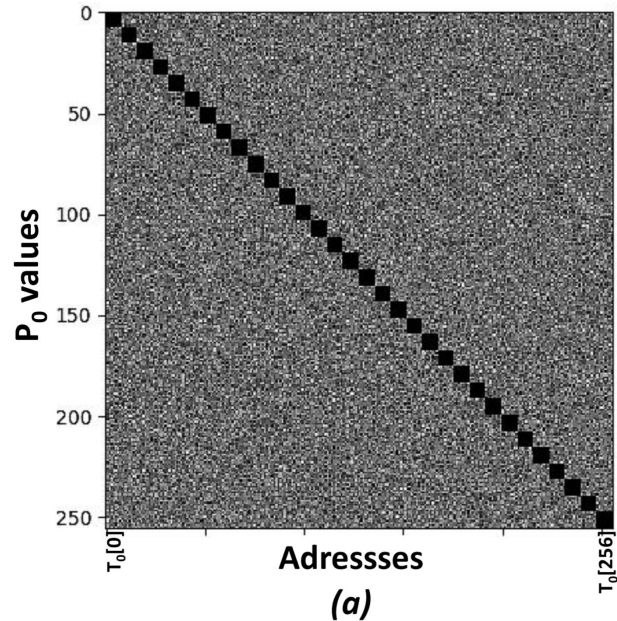
Key Recovery Attack

Attack steps for a partial key recovery:

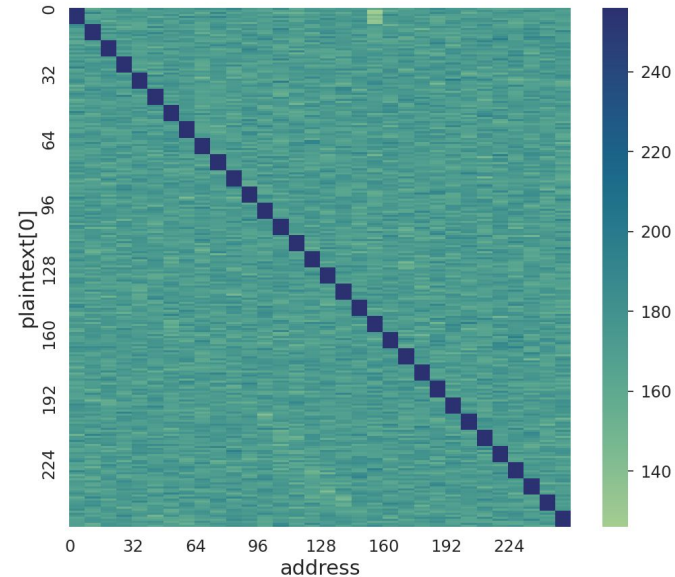
1. Consider a plaintext of 16 bytes. For every byte, test all the possible values while setting the remaining bytes in the plaintext at random.
2. Check if the table has its first 8 bytes in the cache. If so, this plaintext is kept in the candidates, otherwise, it is removed.
3. Iterate the process until the set of candidates is reduced to only 8 values.

Read Channel Attack Results

Reference paper results



Our results

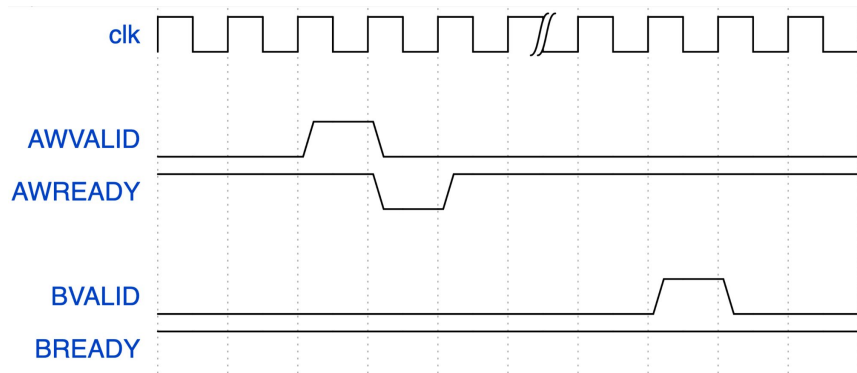


Write Side-channel

We focused on the write transaction of the AXI protocol.

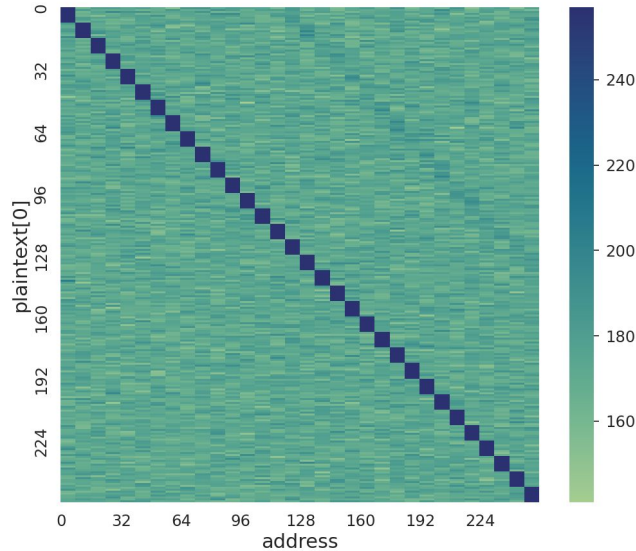
The write handshake length depends on the data location in cache:

- Data in L1 cache → 15+ cycles
- Data in L2 cache → 14 cycles

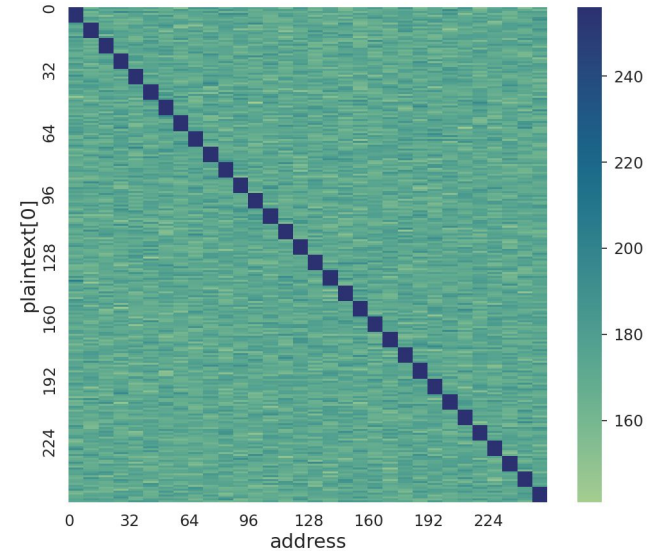


Write Channel Attack Results

Slow version

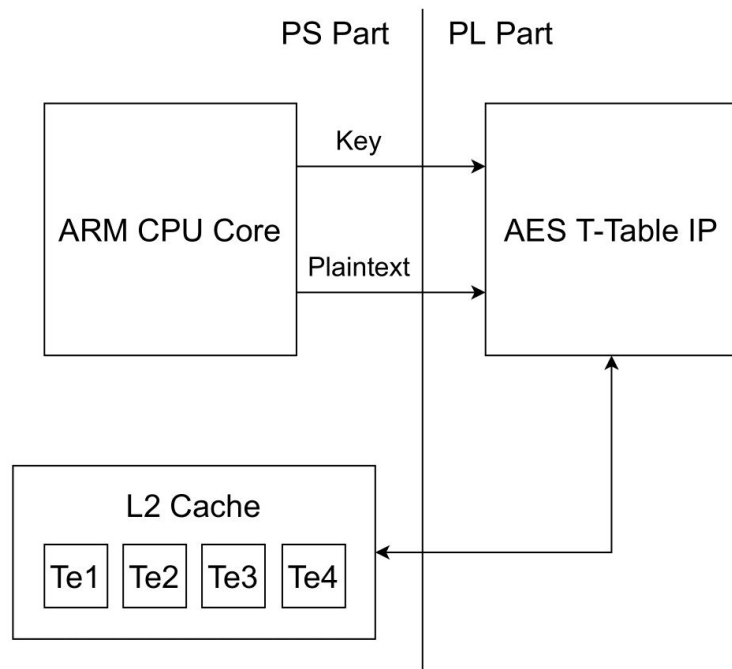


Optimized version



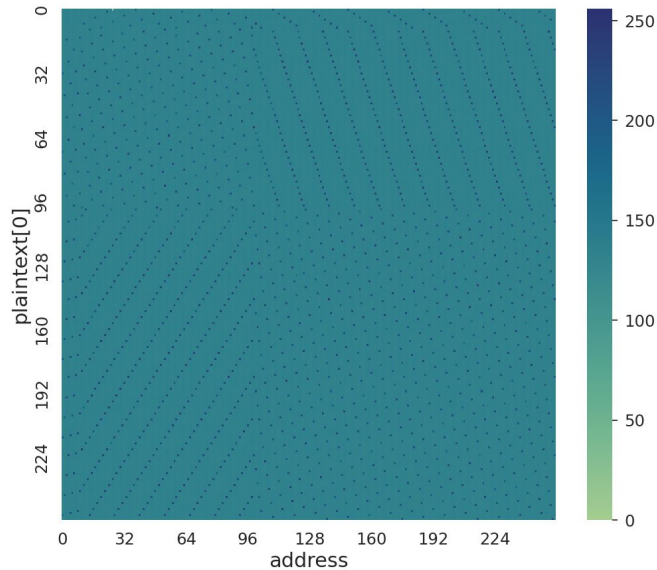
CPU to FPGA

- T-Table implementation of AES as a custom IP
- Tables are stored in shared memory (L2 cache)
- Key and plaintext are provided by the user
- We implemented a timer on the ARM CPU to time accesses to shared T-Tables

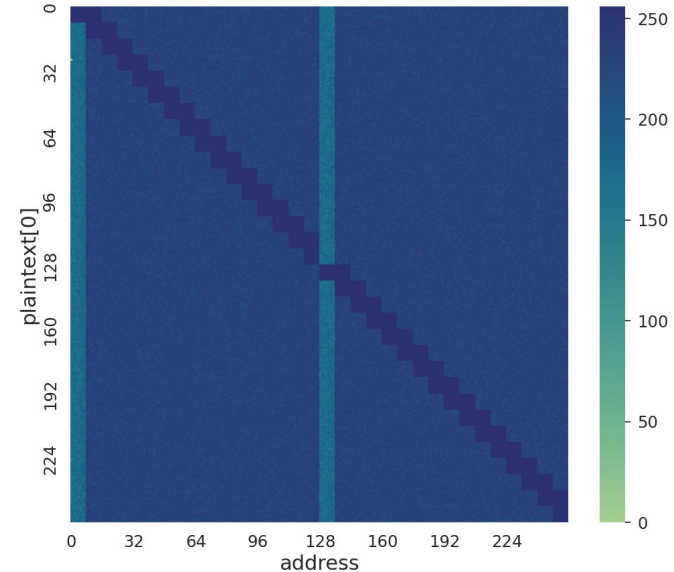


CPU to FPGA Attack Results

Memory Pattern



Attack Results



Conclusions

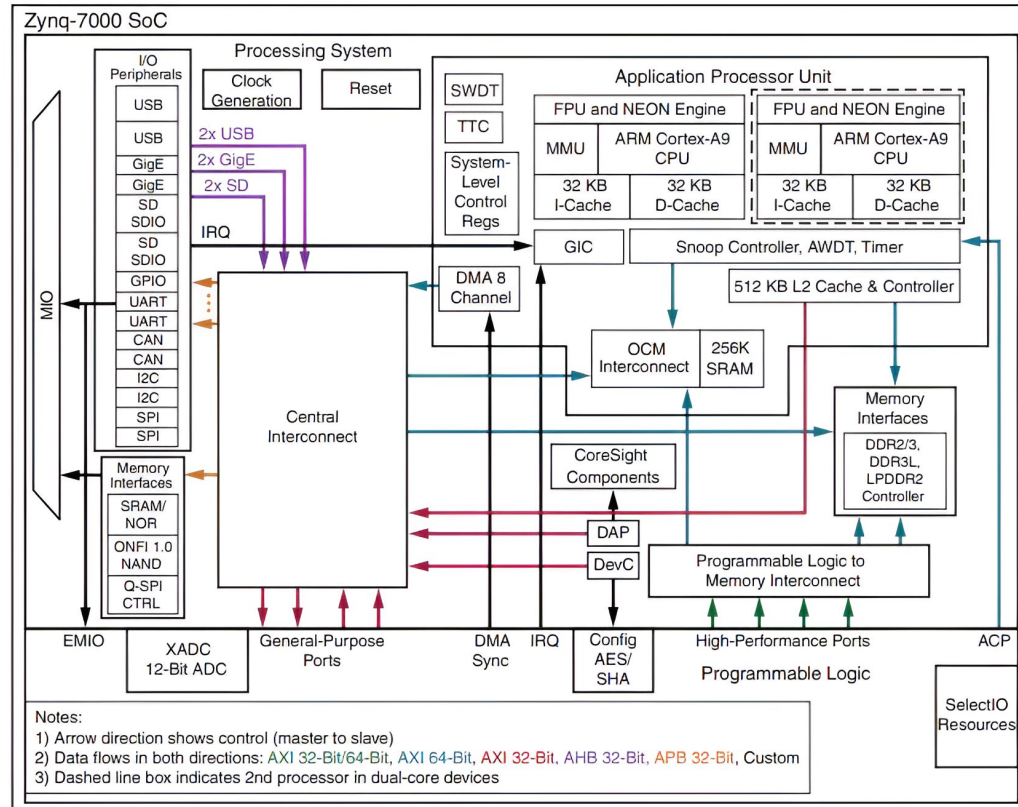
- We assessed the work done by Bossuet et al. showing the feasibility of a key recovery attack on the T-Table implementation of AES
- We found a new side channel on the write transaction
- We replicated the attack on AES
- We built an optimized version of that attack
- We tested the attack on a different threat model: CPU to FPGA

Thank You!

For more information please contact me at
matteo.oldani@epfl.ch



Complete Experimental Setup



VHDL Timer

```
if rising_edge(Clk) then
    counter_valid <= counter_valid_i;
    counter <= std_logic_vector(ticks);

    if awready = '1' and awvalid = '1' then
        ticks <= (others => '0');
        counter_valid_i <= '0';
        countable <= '1';
    elsif rready = '1' and rvalid = '1' then
        counter_valid_i <= '1';
        countable <= '0';
    end if;

    if countable = '1' then
        ticks <= ticks + 1;
    end if;

end if;
```

```
static __inline__ unsigned time_read_mem(unsigned char * addr){  
  
    unsigned cc;  
  
    __asm__ __volatile__ ("mcr p15, 0, %0, c9, c12, 2" :: "r"(1<<31));  
    __asm__ __volatile__ ("mcr p15, 0, %0, c9, c12, 0" :: "r"(5));  
    __asm__ __volatile__ ("mcr p15, 0, %0, c9, c12, 1" :: "r"(1<<31));  
    __asm__ __volatile__ ("ldr r9, [%0]" :: "r"(addr) : "r9");  
    __asm__ __volatile__ ("mov r8, r9" : : : "r9");  
    __asm__ __volatile__ ("mrc p15, 0, r10, c9, c13, 0");  
    __asm__ __volatile__ ("mov %0, r10" : "=r"(cc) : : "r9");  
  
    return cc;  
}
```

Related Work

Beyond Cache Attacks

The work done by Sepúlveda et al. exploited contention on the bus-based SoC communication to execute the attack exactly after the first round of encryption.

JackHammer

Weissman et al. analysed rowhammer and cache attacks on Intel based SoCs. In particular their work showed cache attacks on inclusive last level of caches opening the possibility for PRIME + PROBE attacks.

Attack Example

```
##### STARTING THE ATTACK #####
```

```
The address of te0 is: 0x102EC0
```

```
The address of te1 is: 0x1032C0
```

```
The address of te2 is: 0x1036C0
```

```
The address of te3 is: 0x103AC0
```

```
Possible bytes for key[ 0] = 15:  8  9 10 11 12 13 14 15 --> Encryption required: 195
Possible bytes for key[ 1] =  0:  0  1  2  3  4  5  6  7 --> Encryption required: 149
Possible bytes for key[ 2] = 73: 72 73 74 75 76 77 78 79 --> Encryption required: 37
Possible bytes for key[ 3] = 214: 208 209 210 211 212 213 214 215 --> Encryption required: 136
Possible bytes for key[ 4] = 97: 96 97 98 99 100 101 102 103 --> Encryption required: 122
Possible bytes for key[ 5] = 20: 16 17 18 19 20 21 22 23 --> Encryption required: 151
Possible bytes for key[ 6] =  3:  0  1  2  3  4  5  6  7 --> Encryption required: 254
Possible bytes for key[ 7] = 111: 104 105 106 107 108 109 110 111 --> Encryption required: 95
Possible bytes for key[ 8] = 139: 136 137 138 139 140 141 142 143 --> Encryption required: 89
Possible bytes for key[ 9] = 202: 200 201 202 203 204 205 206 207 --> Encryption required: 133
Possible bytes for key[10] = 70: 64 65 66 67 68 69 70 71 --> Encryption required: 114
Possible bytes for key[11] = 62: 56 57 58 59 60 61 62 63 --> Encryption required: 158
Possible bytes for key[12] = 103: 96 97 98 99 100 101 102 103 --> Encryption required: 128
Possible bytes for key[13] = 249: 248 249 250 251 252 253 254 255 --> Encryption required: 37
Possible bytes for key[14] =  4:  0  1  2  3  4  5  6  7 --> Encryption required: 172
Possible bytes for key[15] = 46: 40 41 42 43 44 45 46 47 --> Encryption required: 104
```

```
█
```