# CSAW'21
# Embedded Security Challenge
## UMBC - SECRETS

Dr. Naghmeh Karimi (Advisor)

Md Toufiq Hasan Anik  Mohammad Ebrahimabadi  Javad Bahrami  Suhee Sanjana Mehjabin

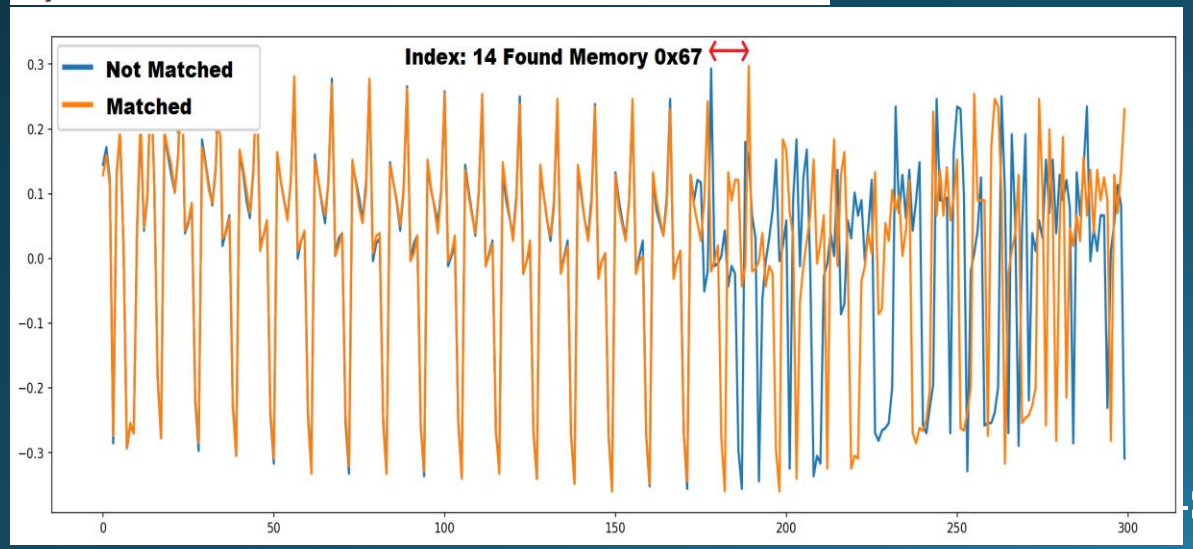
UMBC
University of Maryland Baltimore County

CSAW'21

# Set 1: Recall

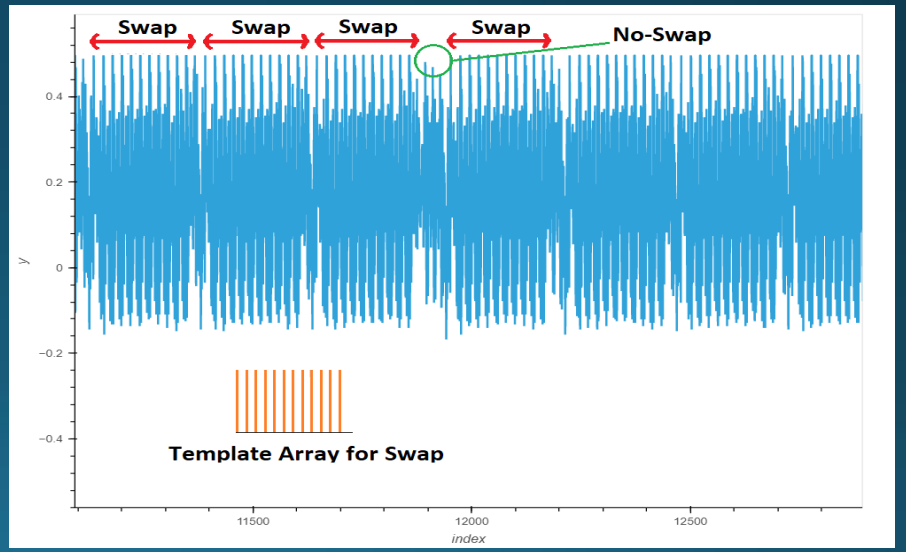
- 16-byte hidden memory
- Comparing the hidden memory with a given message
- Vulnerability: Verify function and break
- Side-Channel Attack: Time Analysis
  - 256 possible power traces for each byte element
  - Correlation among each of the 256 power traces
  - Finding the lowest correlation value
  - As soon as a correct memory is found with lowest correlation, we move to next memory index.

```
for (uint8_t i = 0; i < sizeof(correct_mem); i++) {
  if (correct_mem[i] != data[i]) {
    mem_different = 1;
    break;
  }
```




# Set 1: Fizzy

- Order of memory in a predefined array
- Vulnerability: Dummy loop inside the swap function
- Side-Channel Attack: Power-Time Analysis
  - Analysis of swap function & dummy multiplier
    - 12 power peaks when swapping happens
    - 10 power peaks for 10 iteration of multiplication
    - 2 power peaks for swapping
  - Creating a Template Array for multiplication and find swap
  - Sort in reverse order



# Set 1: Error

- Determining the win-code
- Vulnerability: Calling crc32 function twice from the same hash_loop
- Side-Channel Attack: Fault-Injection
  - Introducing voltage glitch repeatedly for same input to corrupt one or both of the crc32 function calls
  - Storing the set of unmatched output
  - Repeat the process for 5 different inputs
  - Finding the intersection between the 5 sets of unmatched outputs

```
trigger_high();
crc32(buf, sizeof(buf), &crc);
crc32(buf, sizeof(buf), &crc_2);
trigger_low();

if (crc != crc_2) {
  simpleserial_put('r',4, win_code);
} else {
  simpleserial_put('r',4 , (uint8_t*)&crc);
}
```

# Set 1: CRT

- Finding two large prime numbers (P and Q) used for generation of N (related to the public and private keys of the RSA algorithm)
- Vulnerability: Susceptible to fault injection on power supply (VCC) & power/timing attack
- Side-Channel Attack: Fault Injection & Power-Time Analysis
  - Fault is injected in s1 modulo exponentiation.
  - Two modulo exponentiations are concatenated and it produces a faulty one.
  - Based on the grabbed faulty value, P is determined and then Q is determined accordingly.

$$S_1 = (m^{dp})\ mode\ (p) \qquad S_2 = (m^{dq})\ mode\ (q)$$

$$Cipher = S = \{S_1, S_2\}$$

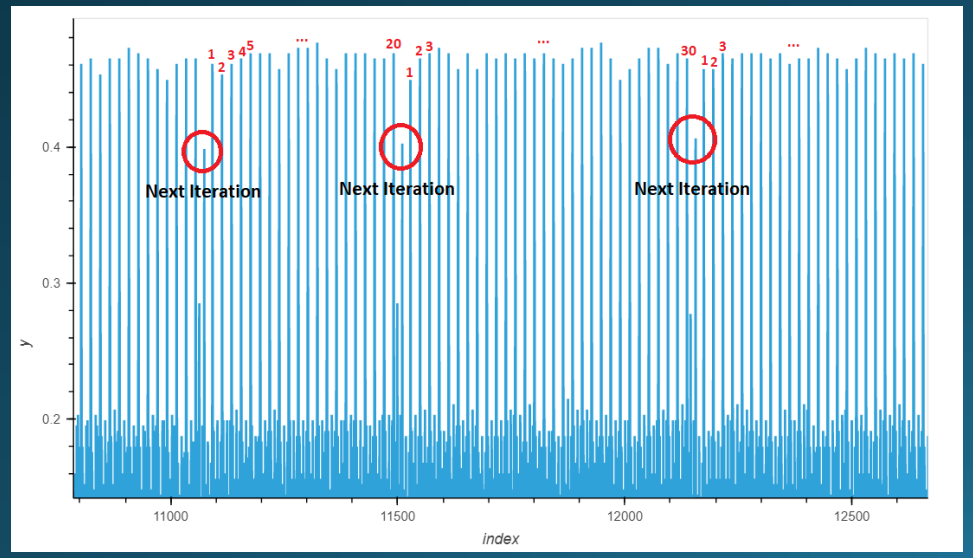
Inject Fault

$$S_1 \rightarrow Faulty\ Cipher = S' = \{S_1', S_2\}$$

$$q = \gcd(m - S'^{e}, N) \qquad p = \frac{N}{q}$$

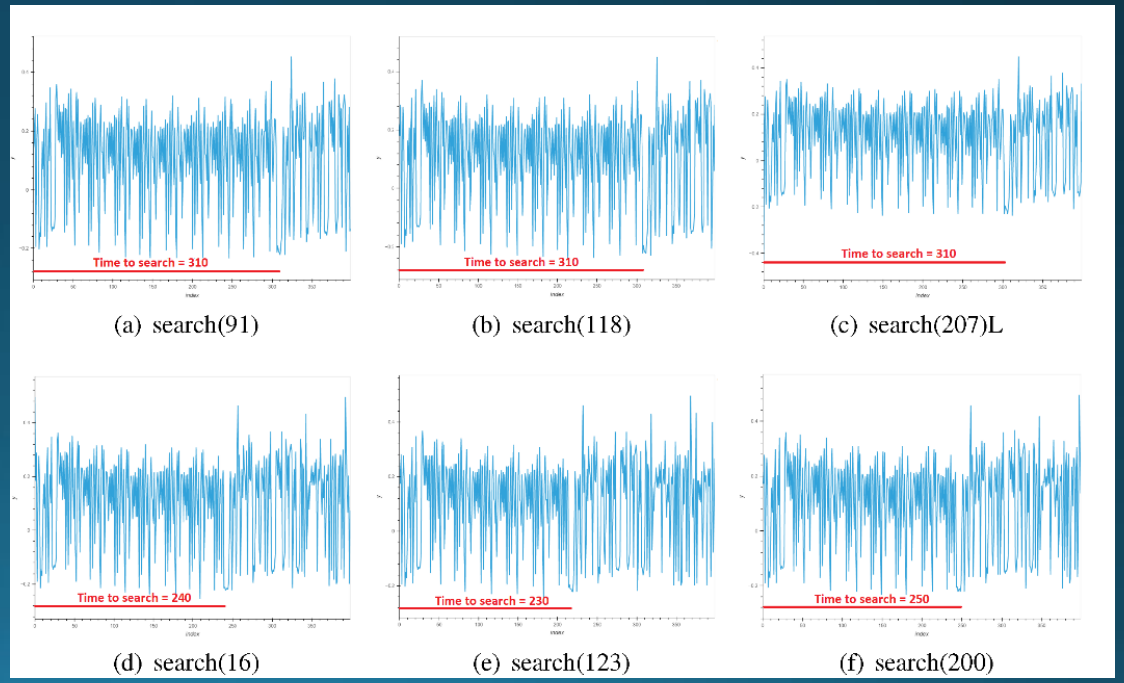
# Set 2: Casino

- Finding the correct order of 15 given elements
- Vulnerability: Number of multiplication in draw function
- Side-Channel Attack: Power timing analysis
  - Computing the power traces
  - Analyzing the draw function
    - Number of multiplication = value of respective element in the array
    - One multiplication per iterations which gives peak in power trace
  - Counting the number of power peaks for each array element location till get ting a drop in power



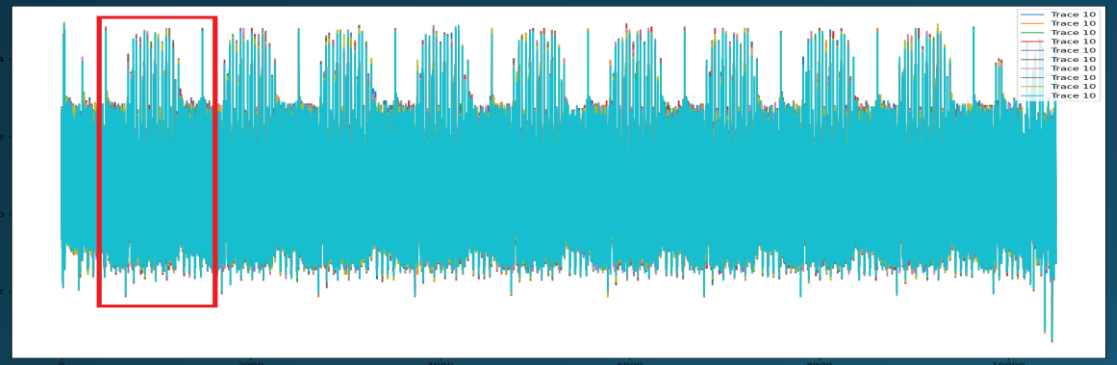

# Set 2: Search

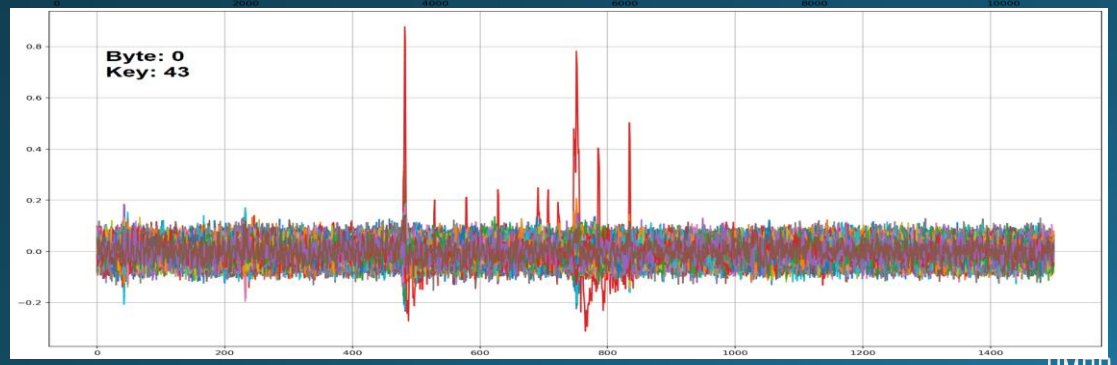
- Finding the 6 moved elements regardless of their order
- Vulnerability: binarySearch function, which divides the array into two parts repeatedly until search is completed
- Side-Channel Attack: Power-Time Analysis
  - Finding power traces for inputs from 0 to $2^8$-1
  - Analyze the duration of power traces
  - The first six elements (in ascending order) or the last six elements (in descending order) are the required elements



# Set 2: FIAsco

- Finding the secret key of AES cipher
- Vulnerability: Side-channel on power
- Side-Channel Attack: Correlation Power Analysis (CPA)
  - Selection of the Intermediate result
  - Power measurement
  - Calculation of hypothetical values
  - Mapping of intermediate values to power traces
  - Comparison of hypothetical values with power traces





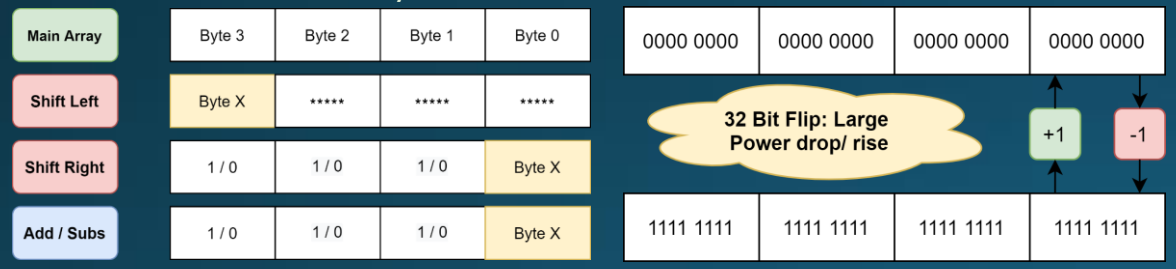

# Set 3: Homebrew

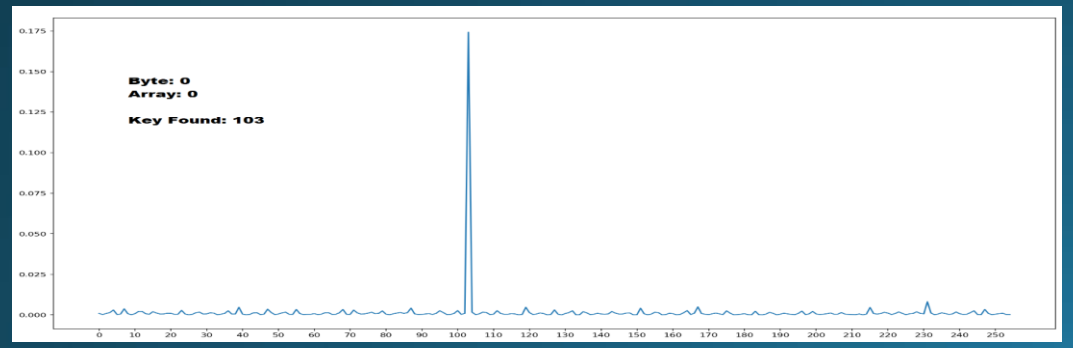
- Finding all different 16 round keys
- Vulnerability: Independency of each block of plaintext from other bytes of key makes brute-force attack's order from $2^{128}$ to $2^8.2^4$
- Side-Channel Attack: Weakness in Architecture
  - Generating an array of random plaintext
  - Generating $2^8$ combinations for all 16 bytes of secret keys
  - Comparing the hardware output with attacker's python script output to determine key candidates
  - Appending new key to key candidate array if the two outputs match
  - After two iterations, insertion of the two key arrays is used as the starting point key array for the next plain text.

```
for (int i = 0; i < 16; i++) {
  for (int j = 0; j < 8; j++) {
    uint8_t key_bit = (key[i] >> j) & 1;
    if (key_bit) {
      data[i] = (sb_1[data[i]] - sb_2[data[i]]);
      data[i] ^= 0xFF;
      data[i] += data[sb_2[data[i]]%16];
    }
    else {
      data[i] *= data[i];
      data[i] *= sb_2[data[i]];
      data[i] -= data[sb_1[data[i]]%16];
    }
  }
```

# Set 3: Calculator

- Finding the array of three elements each of 4 bytes
- Vulnerability: Buffer overflow using the addition and subtraction function (11111111 to 00000000 or vice versa.)
- Side-Channel Attack: Power Analysis
  - Attacking one byte at a time
  - Getting rid of all the other bytes through mult. & div.
  - Overflowing the buffer in both extremities and collecting all the power traces
  - Determining the overflow index by correlating powers
  - Generating two separate arrays of 12 bytes with add/subs
  - Find the element for each indexes with high correction from the arrays of addition or substruction.




# Set 3: NotSoAccessible

- Should find the last element of the round keys
- Vulnerability: injecting fault
- Side-Channel Attack: Fault Injection / Power analysis
  - We brute force 2 bytes and found 0x8d14
  - Only round 25 is there for attack → Many Computation
  - If in round 1 or 32 we could do the Power analysis attacks.
  - Few more vulnerability:
    - If temp is 0 → round key directly goes to output.
    - If fault is in i, it can bypass all the round and leak the information.

```
for(uint8_t i = 0; i < rounds; i++) {
  if (i == 25) {
    trigger_high();
  }
  uint16_t rol_1 = (ctxt[1] << 1) | (ctxt[1] >> (word_size - 1));
  uint16_t rol_2 = (ctxt[1] << 2) | (ctxt[1] >> (word_size - 2));
  uint16_t rol_8 = (ctxt[1] << 8) | (ctxt[1] >> (word_size - 8));

  uint16_t temp = (rol_1 & rol_8) ^ ctxt[0] ^ rol_2;
  ctxt[0] = ctxt[1];
  ctxt[1] = temp ^ round_keys[i];
  if (i == 25) {
    trigger_low();
  }
```