CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

Solving CSAW'25 ESC Challenges

Meowginal Leaks

Pouya Narimani, Kseniia Rogova, Ali Abbasi | November 2025

Set 1

1. GateKeeper 1 & 2

GateKeeper 1 & 2

Challenge:

- Two passwords (8 and 12 characters) to be recovered
- The password verification loop has input-dependent execution time
- Adaptive delay based on the index of the character
- Timing from the OS wasn't accurate

```
trigger_high();

for(uint8_t i = 0; i < password_len[c]; i++) {
    if(data[i] == password[c][i]) {
        matched_chars++;
        for(volatile uint32_t j = 0; j < (uint32_t) (5000 / (c+1)) - i*125; j++);
    } else {
        break;
    }
}

trigger_low();
```

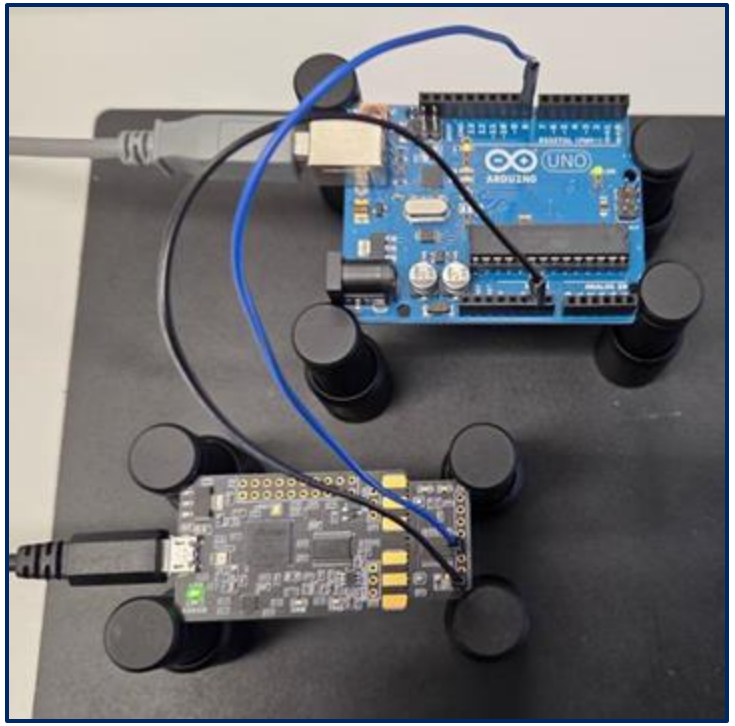
GateKeeper 1 & 2

Test setup:

- We use an Arduino Uno to capture the trigger window (*trigger_high* to *trigger_low* from GPIO4)

Approach:

- Compute adaptive timing thresholds for each character position
- Test possible characters and measure execution time
- If timing exceeds the threshold → mark the character as correct
- Repeat the process until the full password is recovered



gk1{l0g1npwn} → 511 queries (< 45 s)

gk2{7rU3ncrYkIND} → 671 queries (< 37 s)

Set 1

2. Sorters Song 1 & 2

Sorters Song 1 & 2

Challenge:

- Two secret 15-byte arrays (with 8-bit and 16-bit values) to be recovered
- Secret arrays are stored and sorted by the target
- The sorting function leaks power patterns per element comparison
- The attacker can control copied elements (get_pt) and trigger sort calls

```
void sort8(uint8_t* arr, uint8_t len) {
    uint8_t i, j, key_sort;

    for (i = 1; i < len; i++) {
        key_sort = arr[i];
        j = i;

        while (j > 0 && arr[j - 1] > key_sort) {
            arr[j] = arr[j - 1];
            j--;
        }
        arr[j] = key_sort;
    }
}
```

Sorters Song 1 & 2

Approach:

- Capture a reference trace by inserting the smallest value (0x00)
  - This shows the “no shift” case.
- For each key element, run a binary search over possible values.
- For every guess:
  - Capture the trace and compute Euclidean distance to the reference
- Use the distance to decide whether to move the lower or upper bound
- Repeat until the element’s value is found

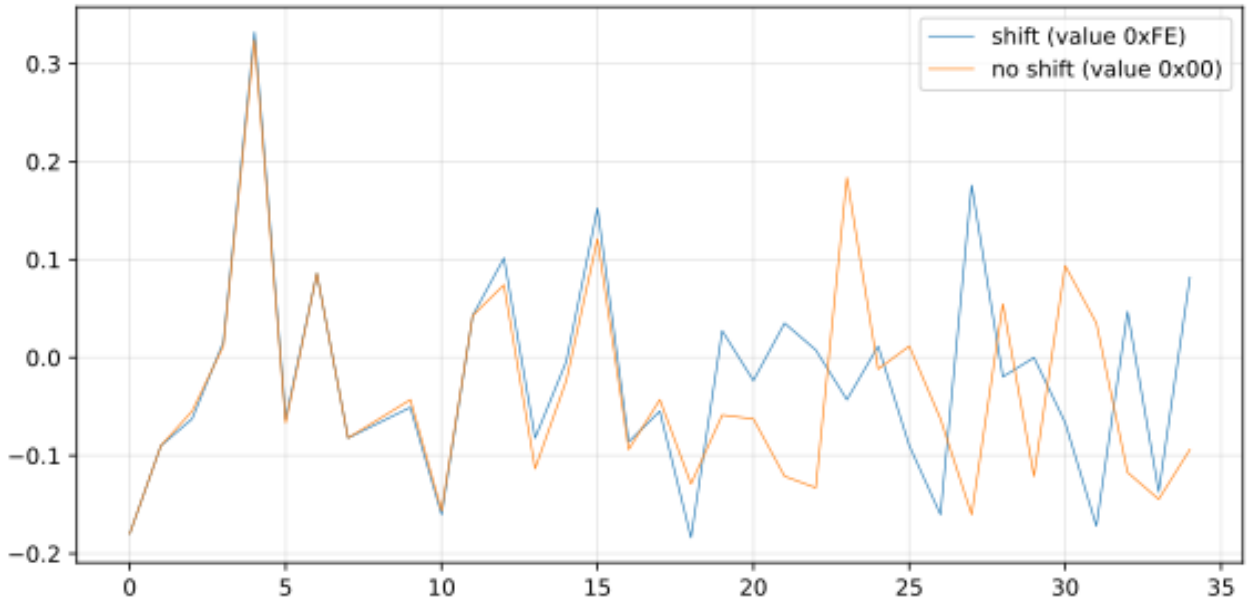
Sorters Song 1 & 2

Complexity:

- For n-bit values → ≈ 2×⌈log2(2^n)⌉+2 queries per byte

Results:

- ss1{y0u_g0t_it_br0!} → 241 queries (8-bit array)
- ss2{!AEGILOPS_chimps → 497 queries (16-bit array)



Set 1

3. Critical Calculation

Critical Calculation

Challenge:

- Simple counter implemented with two nested for-loops
- Boot continues only if the counter reaches the expected final value
- If the counter value differs, the system halts and returns a secret flag
- Task: force a mismatch in the counter to trigger the flag

```
trigger_high();
for(i = 0; i < 100; i++){
    for(j = 0; j < 40; j++){
        cnt += 2;
    }
}
trigger_low();
```

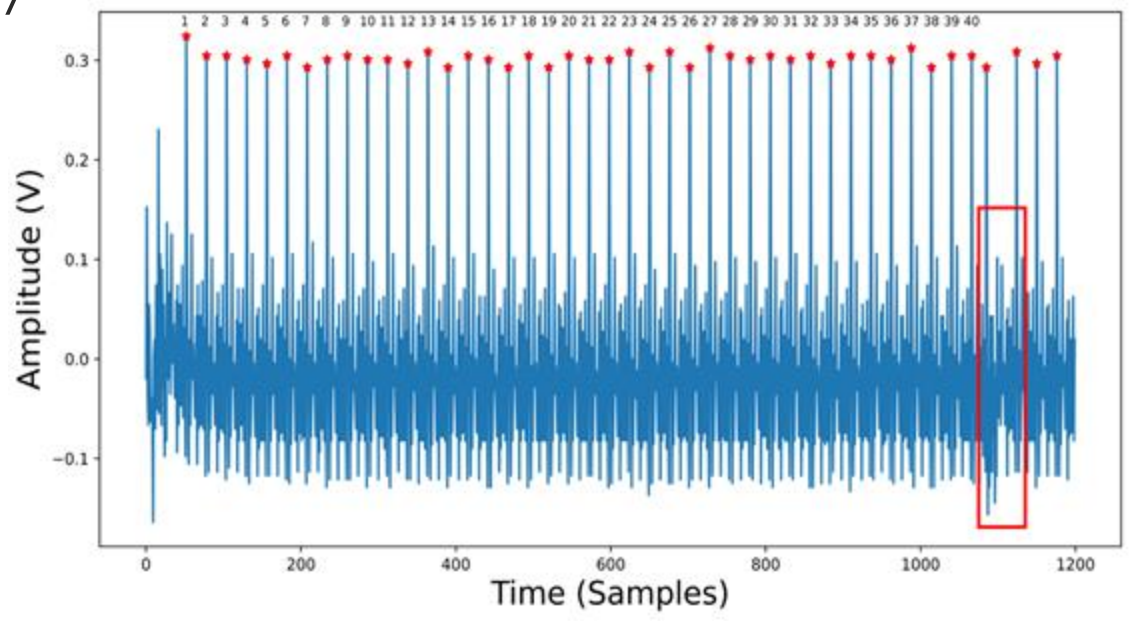
Critical Calculation

Approach:

- Analyze the power trace to locate loop branch instructions
- Inject a voltage glitch at the branch to skip an inner-loop iteration
- Tune glitch parameters (width/repeat) with an oscilloscope and CWNANO:
  - *scope.glitch.repeat* = 3
  - Example location for fault injection: 207

Results:

- cc1{C0RRUPT3D C4LCUL4T10N}



Set 2

1. Dark GateKeeper

Dark GateKeeper

Challenge:

- Recover a 12-character master key
- The password verification loop is almost constant-time
- We employ power analysis to recover the key
- When the if condition is false, the execution pattern is slightly different

```
trigger_high();

for (i = 0; i < KEY_LENGTH; i++) {
    if (incoming_key[i] != master_key[i])
{
        access_granted = 0 ;
    }
}

trigger_low();
```

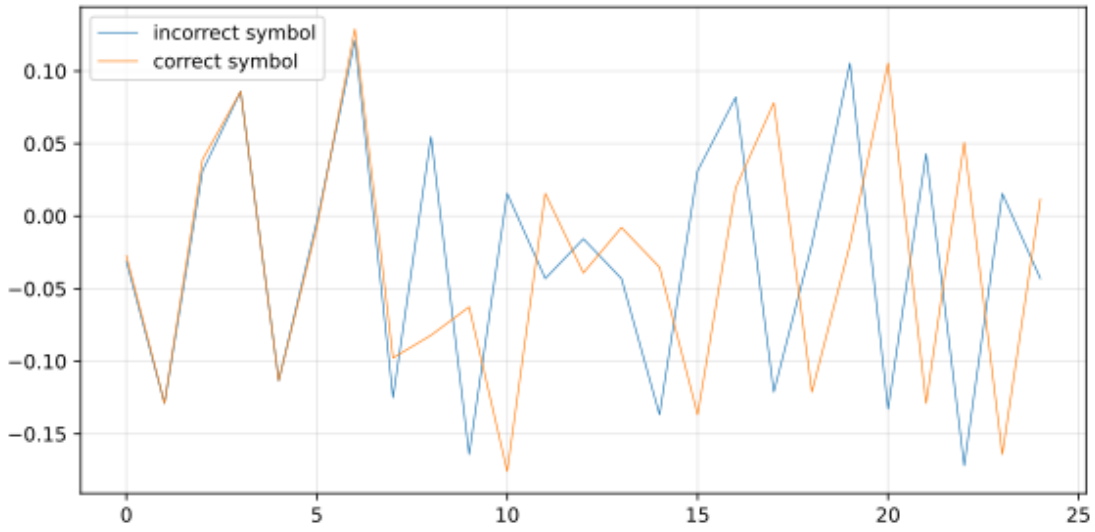
Dark GateKeeper

Approach:

- Capture a reference trace by inserting the smallest value (0x00)
- For each key byte, brute force the possible values
- For every guess:
  - Capture the trace and compute the Euclidean distance to the reference
- Use the distance to decide whether the input was correct or not
- Repeat until the correct key is recovered

Results:

- ESC{J0lt_Th3_G473} → 83 queries



Set 2

2. Hyperspace Jump Drive

Hyperspace Jump Drive

Challenge:

- Recover a 12-byte secret and provide it to capture the flag
- The *invert_polarity()* function calculates the XOR of the input with the secret
- The Correlation Power Analysis (CPA) can recover the key

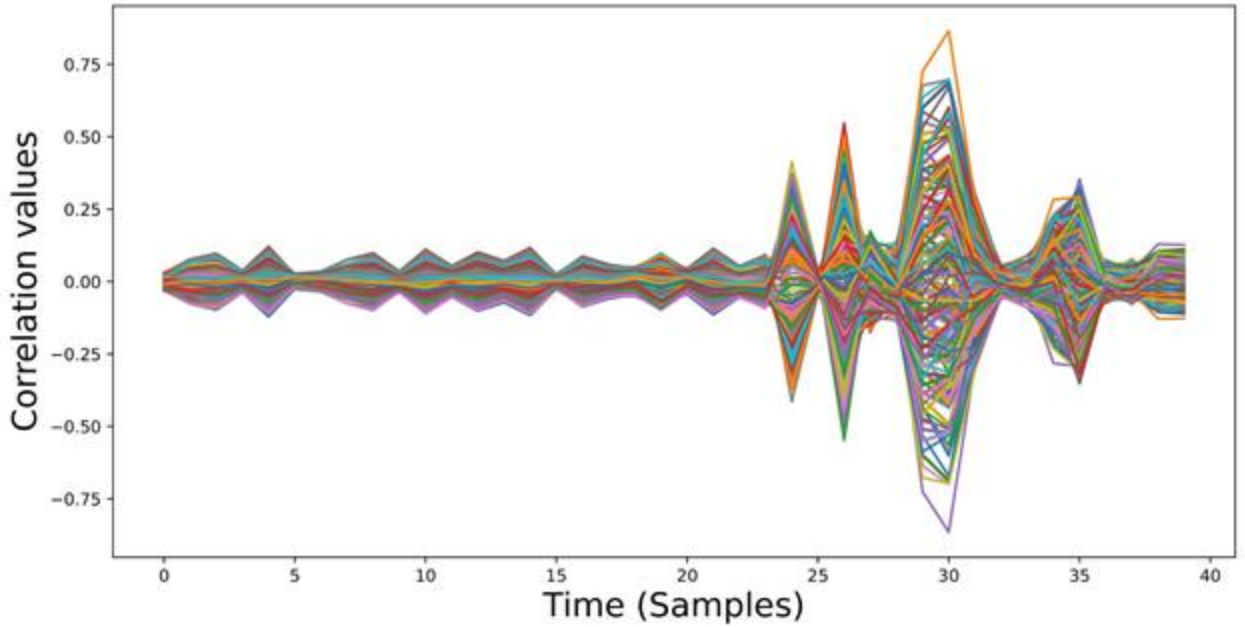
```
trigger_high();

volatile uint8_t* secret_bytes = (uint8_t*)secret_ignition_sequence;
int total_bytes = sizeof(secret_ignition_sequence);
volatile uint8_t temp_result ;

for (int i = 0; i < total_bytes; i++) {
    temp_result = inversion_mask ^ secret_bytes[i];
    simple_delay_separator();
}
total_bytes += temp_result ;

trigger_low();
```

Hyperspace Jump Drive

We gather 25600 power traces (100 per input)

Results:

- [0x3, 0x7D, 0x39, 0x5D, 0xB4, 0xCF, 0x56, 0x66, 0x6F, 0x27, 0xBF, 0x5B]
- b64{c3RhcmR1c3Q=}



Set 3

1. Alchemist Infuser

Alchemist Infuser

Challenge:

- Recover a 16-byte secret and decrypt the message to capture the flag
- The *encrypt()* function calculates the XOR of the input with the secret
- The Correlation Power Analysis (CPA) can recover the key
- The leakage model should change for the second 8 bytes
  - For the first 8 bytes: *plaintext ⊕ key guess*
  - For the second 8 bytes: *(plaintext ⊕ recovered key) ⊕ key guess*

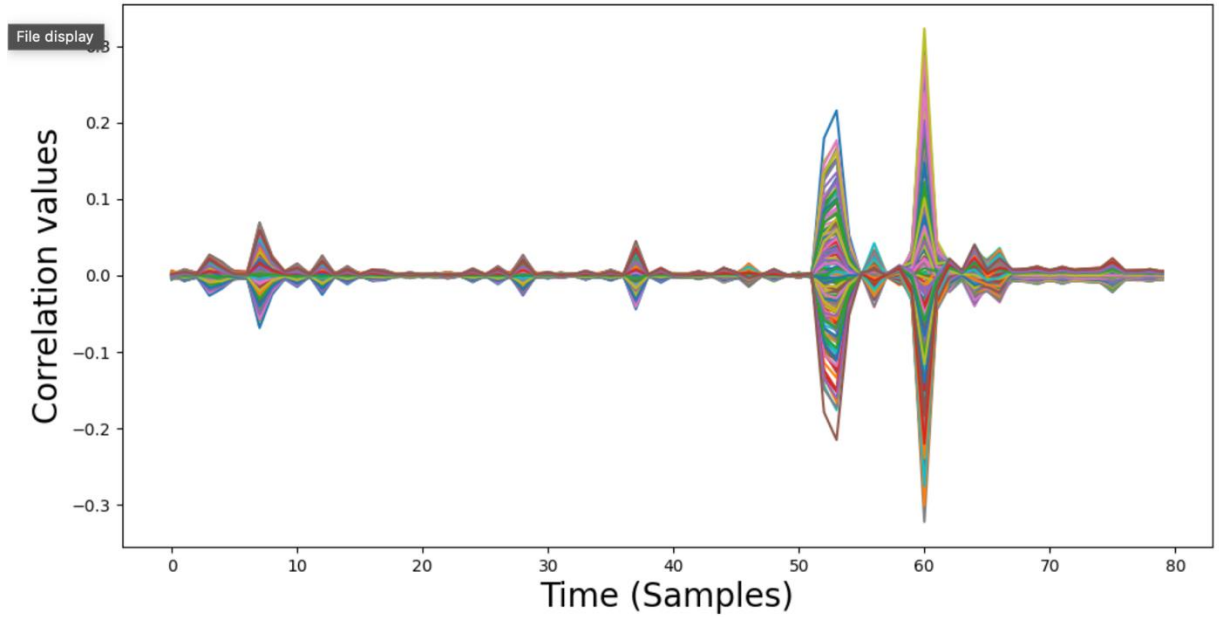
```
trigger_high();
volatile uint8_t temp;
for (int i = 0; i < 16; i++) {
  temp = data[i % dlen] ^ ((uint8_t*)xxtea_key)[i];
  delay(15);
  data[i%dlen] = temp ;
}
trigger_low();
```

Alchemist Infuser

We gather 102400 power traces (400 per input)

Results:

- [0x4e, 0x61, 0x4a, 0x2d, 0x55, 0x67, 0x52, 0x64, 0x32, 0x70, 0x58, 0x6b, 0x38, 0x76, 0x35, 0x73]
- a1c{Wh1teDragonT}



Set 3

2. Echoes of Chaos

Echoes of Chaos

Challenge:

- Similar to the Sorters Song challenge

Approach:

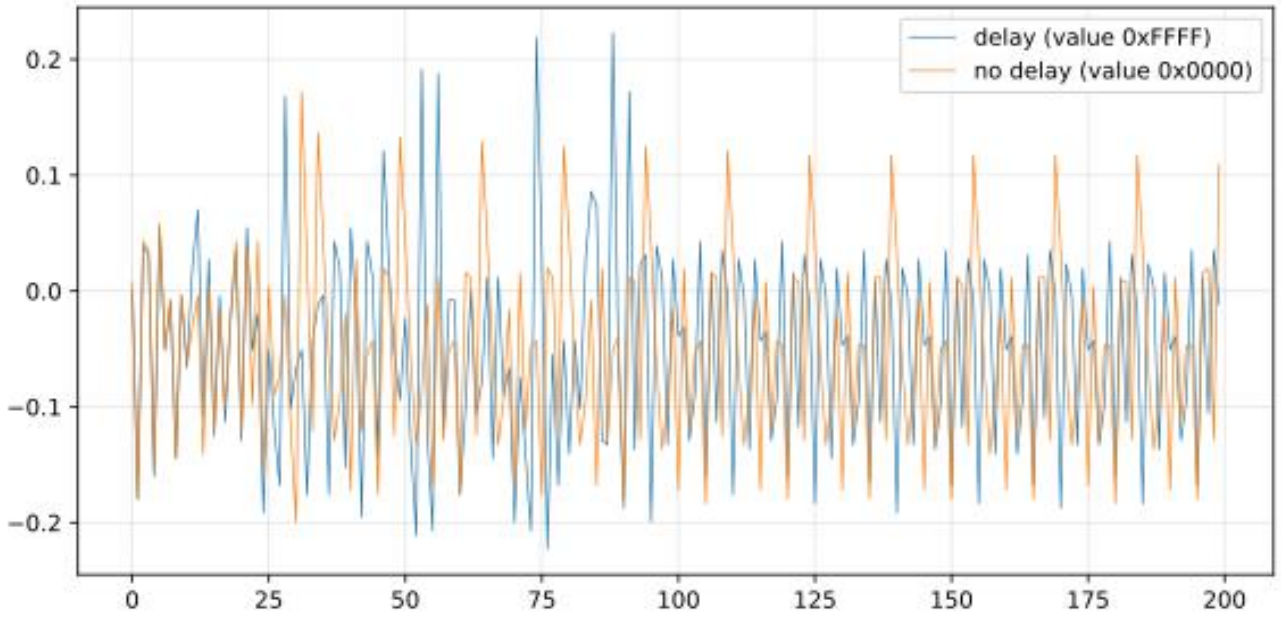
- For each guessed value, capture a new sorting trace for the first left comparison
- Compare it to the reference trace using the Euclidean distance
- Based on the distance, we decide whether to adjust the lower or upper bound of the search range
- Repeat this process for all 15 key values
- Each comparison involves a *reset()* (no new query) and *get_pt()* call

Echoes of Chaos

Complexity: (15 * (16 + 1)) + 1 = 256

Results:

- [0x2B9, 0x1988, 0x2384, 0x369F, 0x4CA1, 0x6582, 0x6D3C, 0x73A0, 0x8CF0, 0xA56D, 0xB3EC, 0xBFF4, 0xE594, 0xFCEC, 0xFDDA]
- eoc{th3yreC00ked}



Thank you ☺

Code for our artifacts can be found here:
github.com/pouya13/csaw2025_ESC_solutions