Bit-by-Bit : The Side Channel Hackathon

Hackathon Challenge 1

Side-Channel Shenanigans: The RSA Key Recovery Challenge

September 21, 2025

The Challenge

Greetings, elite hackers!

A rogue AI, "Hexacrypt," has locked away its ultimate secrets using a custom-built, ultra-lightweight 16-bit RSA crypto-module. While its encryption seems solid, our spies report a critical flaw: the module is a **blabbermouth**. During decryption, it leaks sensitive information through its power consumption! This is a classic side-channel vulnerability, and it's our only way in.

Your mission is to interface with the hardware, listen to the chip's "power whispers" as it works, analyze these whispers to reconstruct the AI's secret private key, and unlock the data.

The Setup: You are targeting an **STM32F303 microcontroller** on a **CW308 target board**. The device is running firmware that performs RSA decryption. All power measurements and communication will be done using a **ChipWhisperer-Lite**.

Mission Objectives

1. **Establish a Comms Link:** Write a Python script to communicate with the target device using the provided `simpleserial_rsa-CW308_STM32F3.hex` firmware.
2. **Gather Intel:** Use your script to send random ciphertexts to the device and capture its power consumption traces during the decryption process.
3. **Crack the Key:** Analyze the collected power traces using side-channel analysis techniques to recover the secret 15-bit RSA private key, d.
4. **Verify & Celebrate:** Confirm you have the right key using a special verification sequence.

Part 1: Intel Gathering (Trace Acquisition)

The provided firmware expects a 2-byte ciphertext and returns a 2-byte plaintext. **A crucial rule for this RSA implementation is that any ciphertext integer c you send must be less than the modulus N (64507)**. The firmware will reject values greater than or equal to N.

Starter Python Script

This script will help you program the board and communicate with it. Your first task is to extend it to capture and save the power traces.

```python
import chipwhisperer as cw
import random
import csv
import os
import numpy as np

# --- 1. CONNECT TO HARDWARE ---
scope = cw.scope()
scope.default_setup()
target = cw.target(scope)

# --- 2. PROGRAM THE TARGET ---
prog = cw.programmers.STM32FProgrammer
print("Programming target...")
cw.program_target(scope, prog, "simpleserial_rsa-CW308_STM32F3.hex")
print("Programming done.")

# --- 3. CONFIGURE CAPTURE SETTINGS ---
scope.clock.adc_src = "clkgen_x1"
scope.adc.samples = 5000 # You can adjust this value

# --- 4. PREPARE FOR CAPTURE ---
RSA_N = 64507
NUM_TRACES = 500 # YOU decide how many traces you need!
OUT_CSV = "my_traces.csv"

random.seed(0xCAFEBABE)
rows = []

# --- 5. CAPTURE LOOP ---
for i in range(NUM_TRACES):
    # Per RSA rules, ciphertext must be < N.
    c_int = random.randint(0, RSA_N - 1)
    ct_bytes = c_int.to_bytes(2, 'big')

    scope.arm()
    target.simpleserial_write('p', ct_bytes)

    ret = scope.capture()
    if ret:
        print(f"Capture timed out for trace {i+1}")
        continue

    # Optional: Read the plaintext response back from the target
    resp = target.simpleserial_read('r', 2)

    trace = scope.get_last_trace()
    # TODO: Save the trace, ciphertext, and plaintext
```

ChipWhisperer API Cheat Sheet

- `scope = cw.scope()`: Connects to the ChipWhisperer capture hardware.
- `target = cw.target(scope)`: Connects to the target device.
- `cw.program_target(...)`: Flashes firmware onto the target.
- `scope.arm()`: Arms the scope, preparing it to capture on a trigger.
- `target.simpleserial_write('p', ct_bytes)`: Sends data to the target.
- `ret = scope.capture()`: Starts the capture. Returns `False` on success.
- `trace = scope.get_last_trace()`: Retrieves the captured power trace.

- `resp = target.simpleserial_read('r', 2)`: Reads data back from the target.

Part 2: Cracking the Code (The Attack)

Now for the real fun! You have the traces; it's time to find the key.

Understanding RSA

The RSA algorithm relies on modular exponentiation.

- **Encryption:** $Ciphertext = Plaintext^e \pmod{n}$
- **Decryption:** $Plaintext = Plaintext^d \pmod{n}$

The pair (e, n) is the public key, and (d, n) is the private key. Your target device is computing the decryption operation using a secret private key d.

The Vulnerability: Square-and-Multiply

To compute $C^d \pmod{n}$ efficiently, the device uses a classic algorithm called **Square-and-Multiply**.

Algorithm 1 Square-and-Multiply Algorithm

```
Function SquareAndMultiply(base C, exponent d, modulus n)
Result ← 1
for all bit b in exponent d (from MSB to LSB) do
    Result ← (Result × Result) mod n          ▷ SQUARE operation (always happens)
    if bit b is 1 then
        Result ← (Result × C) mod n           ▷ MULTIPLY operation (conditional)
return Result
```

The vulnerability is in the **If** statement. The device performs an extra multiplication **only when the secret key bit is a '1'**. This creates a detectable difference in power consumption.

Hints for the Attack

- The secret key d is **15 bits** long.
- **Known Bits:** The Most Significant Bit (MSB) is **1**. The 4 Least Significant Bits (LSBs) are **0001**.
- The key looks like this: `1 ? ? ? ? ? ? ? ? ? ? 0 0 0 1`. You only need to recover the **10 unknown bits**.
- Your goal is to use side-channel analysis to distinguish the power consumption of a '1' bit from a '0' bit.

The Final Check (Verification)

Once you believe you have recovered the full 15-bit key, how do you know you're right? Hexacrypt's module has a quirky feature.

If you send the correct private key d as a ciphertext input, the module will output the specific plaintext value of `6267`.

Use your communication script to send your recovered key to the device. If it responds with `6267`, you have successfully completed your mission!

Submission

To complete the challenge, please prepare the following:

- A brief report (PDF format) describing your methodology, analysis techniques, and any challenges you faced.
- Clearly state the final **recovered 15-bit private key in decimal format** within your report.
- A link to a **public GitHub repository** containing all your commented scripts.

Good luck, and may the side channels be ever in your favor! Happy Hacking!