Correlation Power Analysis (CPA) Attack on RSA Using Hamming Weight Model with Key Verification

Sumantra Dutta and Debopama Basu

September 22, 2025

1 Introduction

This report describes the methodology, analysis techniques, and results of a Correlation Power Analysis (CPA) attack on a small 15-bit RSA implementation using the Hamming Weight model. The objective was to recover unknown bits of the RSA private key from measured power traces and verify the recovered key using a ChipWhisperer target.

2 Hardware Setup

The following hardware components were used in the experiment:

- **ChipWhisperer-Lite** – Used as the side-channel analysis platform, providing power measurement and capture capability.
- **CW308 Target Board** – Target board on which the microcontroller is mounted.
- **STM32F Microcontroller** – Microcontroller device running the cryptographic algorithm under test.
- **SMA Cable** – Connects the ChipWhisperer-Lite to the oscilloscope input for power trace acquisition.
- **20-Pin Header Connector** – Provides the interface between the ChipWhisperer-Lite and the STM32F for clock, reset, and measurement signals.

3 RSA Square-and-Multiply Algorithm

Algorithm 1 Square-and-Multiply for RSA Decryption

```
1: function SquareAndMultiply($C, d, n$)
2:     $Result \leftarrow 1$
3:     for each bit $b$ in $d$ (MSB to LSB) do
4:         $Result \leftarrow (Result \times Result) \bmod n$          ▷ Square (always)
5:         if $b = 1$ then
6:             $Result \leftarrow (Result \times C) \bmod n$          ▷ Multiply (conditional)
7:         end if
8:     end for
9:     return $Result$
10: end function
```

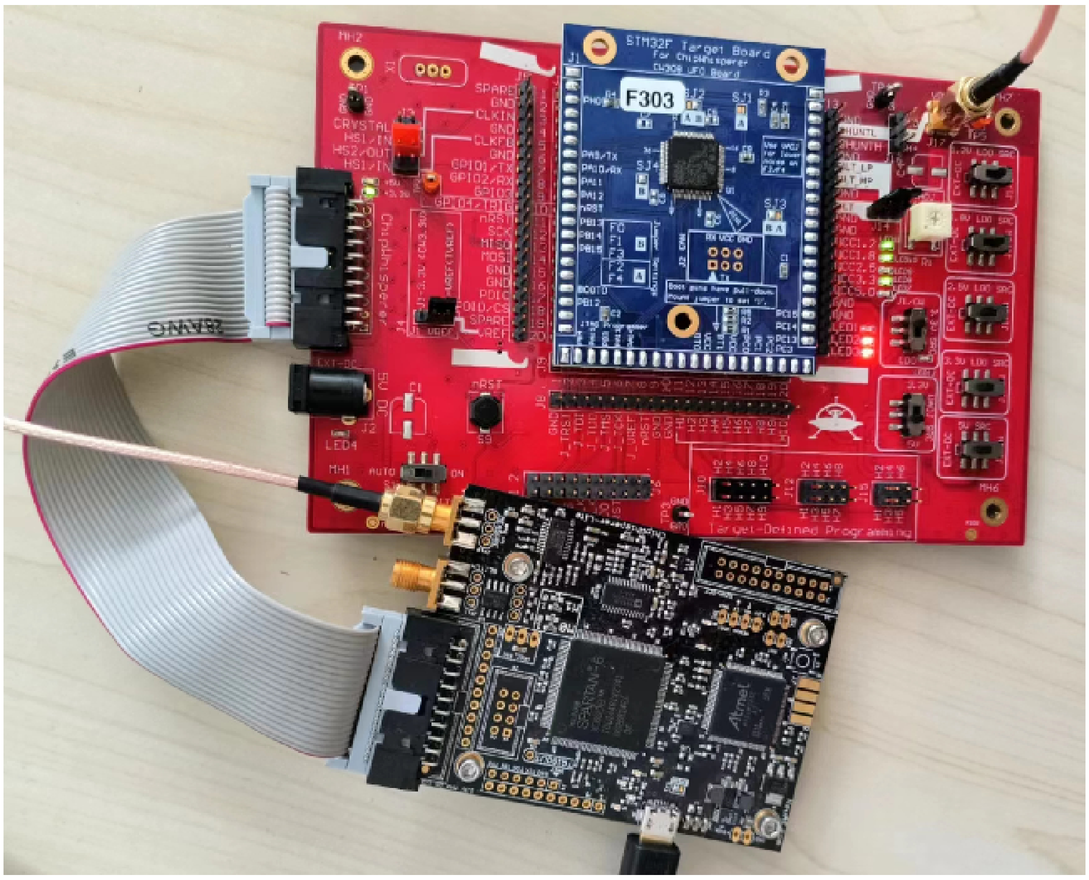


Figure 1: ChipWhisperer Setup

4 Methodology and Analysis

- **RSA Algorithm:** The RSA Algorithm uses a Square and Multiply operation. The security of the RSA algorithm is built on the fact that the the modular exponentiation is done with a very large prime number N whose factorisation is not possible. Hence instead of trying to break the algorithm mathematically we do a side channel attack. By side channel attack we mean collecting physical leakages like EM Radiations, timings and power values for recovering the secret key when the algorithm is running.

- **Trace Collection:** 5000 power traces were collected using the ChipWhisperer by dumping the RSA hex file which contains the simpleserial protocol commands for communicating with the host pc and the target device during the multiplication operation which happens when bit = 1. The CSV file contains ciphertexts and their corresponding power traces.
- **Hamming Weight Model:** The instantaneous power consumption of the device is assumed to correlate with the number of '1's in the binary representation of intermediate values.
- **CPA Attack:** For each unknown bit of the private key, intermediate values were simulated under two hypotheses (0 or 1), converted to Hamming weights, and correlated with measured power traces using Pearson correlation. The hypothesis with the higher absolute correlation was selected as the guessed bit.
- **Key Reconstruction:** After recovering all unknown bits, the final 15-bit private key was reconstructed by combining the known MSB, recovered bits, and LSBs.
- **Key Verification:** The recovered key was verified by sending it to the RSA device and reading the plaintext output. Successful verification confirmed that the CPA attack correctly recovered the key.

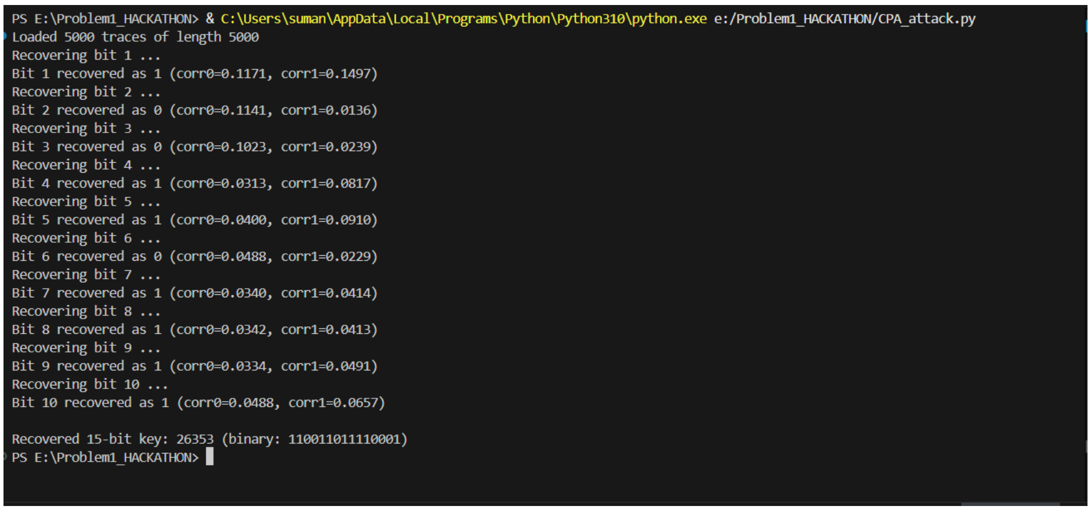
5 Challenges Faced

- Selecting the appropriate window for correlation was critical to correctly recover the key bits.
- Some correlation values were very close, requiring careful comparison of absolute correlations to guess the correct bit.
- Ensuring correct byte formatting when sending the recovered key back for verification.

6 Results

The CPA attack successfully recovered the 15-bit RSA private key.

Final Recovered 15-bit Private Key (Decimal): `26353`
Binary Representation: `110011011110001`



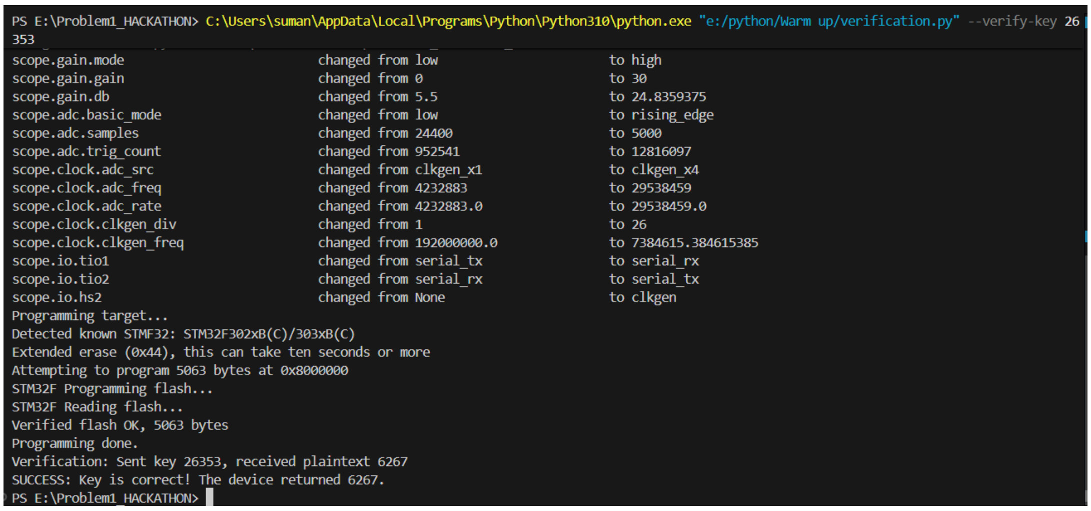
```
PS E:\Problem1_HACKATHON> & C:\Users\suman\AppData\Local\Programs\Python\Python310\python.exe e:/Problem1_HACKATHON/CPA_attack.py
Loaded 5000 traces of length 5000
Recovering bit 1 ...
Bit 1 recovered as 1 (corr0=0.1171, corr1=0.1497)
Recovering bit 2 ...
Bit 2 recovered as 0 (corr0=0.1141, corr1=0.0136)
Recovering bit 3 ...
Bit 3 recovered as 0 (corr0=0.1023, corr1=0.0239)
Recovering bit 4 ...
Bit 4 recovered as 1 (corr0=0.0313, corr1=0.0817)
Recovering bit 5 ...
Bit 5 recovered as 1 (corr0=0.0400, corr1=0.0910)
Recovering bit 6 ...
Bit 6 recovered as 0 (corr0=0.0488, corr1=0.0229)
Recovering bit 7 ...
Bit 7 recovered as 1 (corr0=0.0340, corr1=0.0414)
Recovering bit 8 ...
Bit 8 recovered as 1 (corr0=0.0342, corr1=0.0413)
Recovering bit 9 ...
Bit 9 recovered as 1 (corr0=0.0334, corr1=0.0491)
Recovering bit 10 ...
Bit 10 recovered as 1 (corr0=0.0488, corr1=0.0657)

Recovered 15-bit key: 26353 (binary: 110011011110001)
PS E:\Problem1_HACKATHON>
```

Figure 2: ChipWhisperer console output during key Recovery.

6.1 Verification Output

To confirm the recovered key was correct we used the ChipWhisperer target's verification command that accepts the recovered key and returns the decrypted plaintext. The console output (screenshot and raw lines below) shows the device accepted the key and returned a plaintext, and the verification routine declared success.



```
PS E:\Problem1_HACKATHON> C:\Users\suman\AppData\Local\Programs\Python\Python310\python.exe "e:/python/Warm up/verification.py" --verify-key 26
353
scope.gain.mode                        changed from low                     to high
scope.gain.gain                        changed from 0                       to 30
scope.gain.db                          changed from 5.5                     to 24.8359375
scope.adc.basic_mode                   changed from low                     to rising_edge
scope.adc.samples                      changed from 24400                   to 5000
scope.adc.trig_count                   changed from 952541                  to 12816097
scope.clock.adc_src                    changed from clkgen_x1               to clkgen_x4
scope.clock.adc_freq                   changed from 4232883                 to 29538459
scope.clock.adc_rate                   changed from 4232883.0               to 29538459.0
scope.clock.clkgen_div                 changed from 1                       to 26
scope.clock.clkgen_freq                changed from 192000000.0             to 7384615.384615385
scope.io.tio1                          changed from serial_tx               to serial_rx
scope.io.tio2                          changed from serial_rx               to serial_tx
scope.io.hs2                           changed from None                    to clkgen
Programming target...
Detected known STMF32: STM32F302xB(C)/303xB(C)
Extended erase (0x44), this can take ten seconds or more
Attempting to program 5063 bytes at 0x8000000
STM32F Programming flash...
STM32F Reading flash...
Verified flash OK, 5063 bytes
Programming done.
Verification: Sent key 26353, received plaintext 6267
SUCCESS: Key is correct! The device returned 6267.
PS E:\Problem1_HACKATHON>
```

Figure 3: ChipWhisperer console output during key verification (programming, verification and final success message).

The important lines from the console (verbatim) are:

```
Verification: Sent key 26353, received plaintext 6267
SUCCESS: Key is correct! The device returned 6267.
```

Explanation of verification output:

- `Verification: Sent key 26353` — the host sent the recovered decimal key value (26353) to the target device as the candidate private key.

- `received plaintext 6267` — the device used the supplied key to decrypt (or to perform the RSA operation) and returned the resulting plaintext value 6267.
- `SUCCESS: Key is correct!  The device returned 6267.` — the target's verification routine checked the decrypted plaintext (likely against an expected known value or by verifying a protocol handshake) and declared the supplied key correct.

Because the device accepts the recovered key, performs the RSA operation with it, and returns the expected plaintext and a success message, we conclude the CPA-recovered key is indeed the device's private key.

7 Repository

All commented scripts related to trace collection, CPA attack, and key verification are available in the public GitHub repository:
RSA by Team Destra

8 Conclusion

This experiment demonstrates the effectiveness of CPA attacks on small RSA implementations. By analyzing the correlation between Hamming weights of simulated intermediate values and actual power traces, unknown key bits were successfully recovered and verified on a real target device. The explicit verification step — sending the recovered key and receiving the expected plaintext plus an affirmative success message — provides strong evidence of a correct key recovery.