

# Side-Channel Attack

Simple Power Analysis on  
RSA Algorithm

Yeoh Wei Yang (U2121112A)  
Tay Chee Yong (U2121657K)



# Table of Content

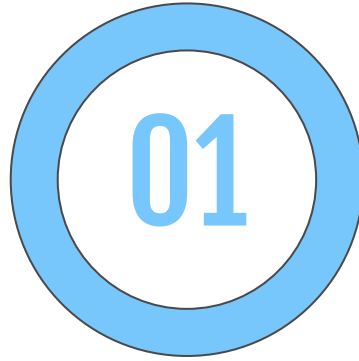


**01** Project Overview

**02** Simple Implementation of  
SPA using Arduino

**03** Code Explanation

**04** Review of LibgCrypt  
RSA Implementation



# Project Overview

The purpose of this project is to see how we can attack RSA by analyzing the power consumption during cryptographic operations.

...

# Project Overview

$\frac{\text{num}^{\text{exp}}}{r=1}$

for each bit of exp:

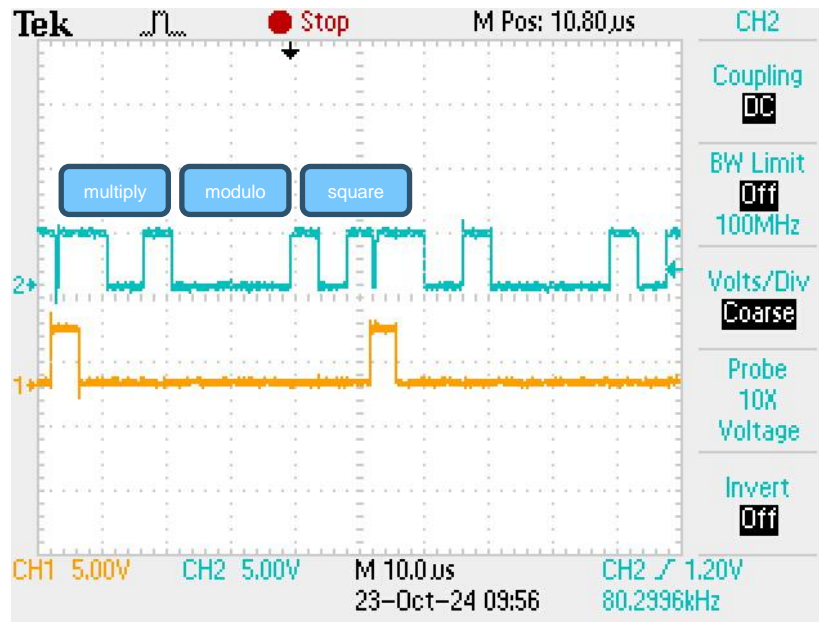
$r = r \cdot r \% n$

if bit == 1:

$r = r \cdot \text{num} \% n$

return r

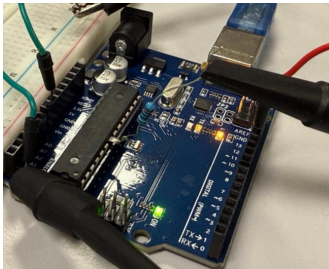
square  
square  
square  
multiply  
square  
multiply



# 02

## Simple Implementation of SPA (Arduino)

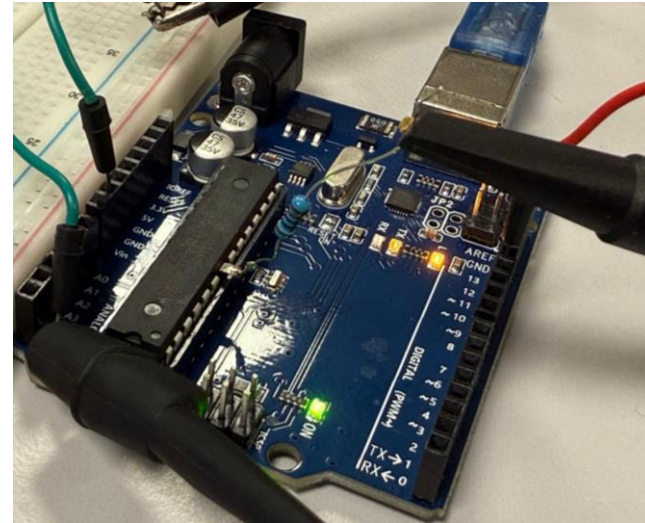
Ideally, ChipWhisperer would be the best option but it was too costly hence we found an alternative way which is to use Arduino instead



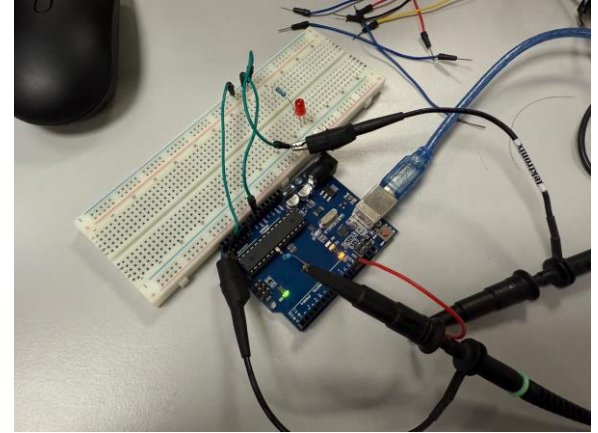
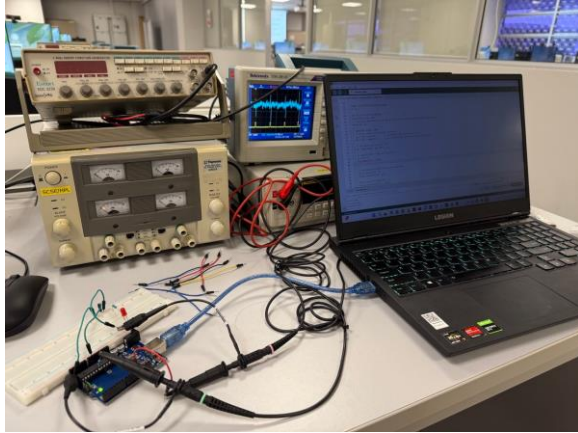
...

# Simple Implementation of SPA (Arduino)

(PCINT14/RESET) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 ( $\overline{SS}$ /OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)



# Simple Implementation of SPA (Arduino)

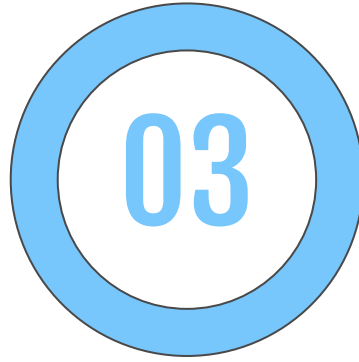


# Simple Implementation of SPA (Arduino)

```
1  uint64_t n, p, q, e, d, M, received_encrypted_message, temp;
2
3  int intarr[64]; // Array of bits of decryption key, d
4  int index = 0; // Length (in bits) of decryption key, d
5
6  void setup() {
7      // put your setup code here, to run once:
8      Serial.begin(115200);
9      pinMode(11, OUTPUT);
10     pinMode(12, OUTPUT);
11     digitalWrite(11, HIGH);
12     digitalWrite(12, HIGH);
13
14     // RSA Parameters
15     // To the attacker, p, q, d are unknown. n & e make up the public key
16     p = 5; q = 11; e = 33; d = 17;
17
18     n = p * q; // the mod value
19
20     // Original Message
21     M = 2;
22
23     // Simulate receiving of encrypted msg
24     received_encrypted_message = 52; // 2^33 mod 55
25
26     // Calculate d in binary form and insert it into an array for ease of future calculations
27     while (d > 0) {
28         intarr[index] = d % 2;
29         d = d >> 1;
30         index = index + 1;
31     }
32 }
33 }
```

```
34
35 void loop() {
36     // put your main code here, to run repeatedly:
37
38     temp = received_encrypted_message;
39     M = 1;
40     d = 17;
41
42     Serial.println("Running RSA...");
43
44     digitalWrite(11, LOW);
45     // Modular Exponentiation
46     for (int i = index - 1; i >= 0; i--)
47     {
48         digitalWrite(12, LOW);
49         // Do the squaring first
50         M = (M * M) % n;
51
52         // If the bit is a 1, then an additional multiplication should be done
53         if (intarr[i]) {
54             M = (M * temp) % n;
55         }
56
57         digitalWrite(12, HIGH);
58     }
59     digitalWrite(11, HIGH);
60 }
```



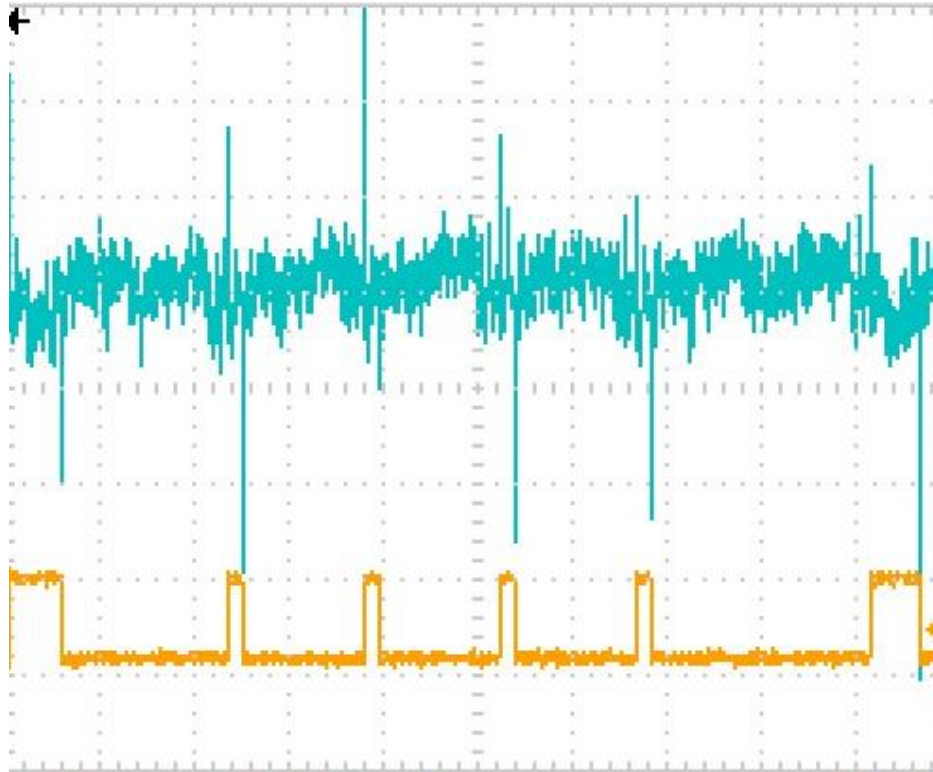


# Code Explanation

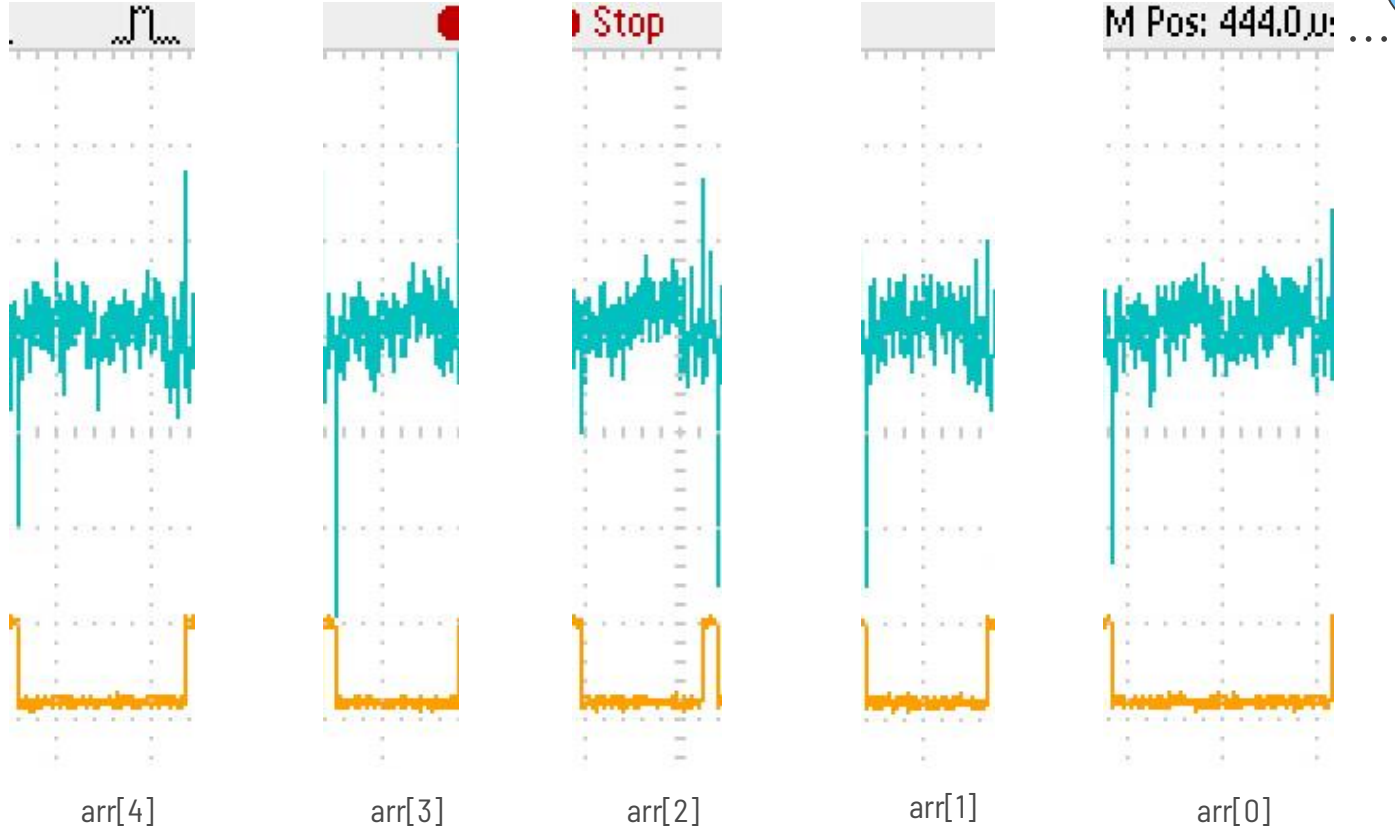
-Moving over to Google Colab-

...

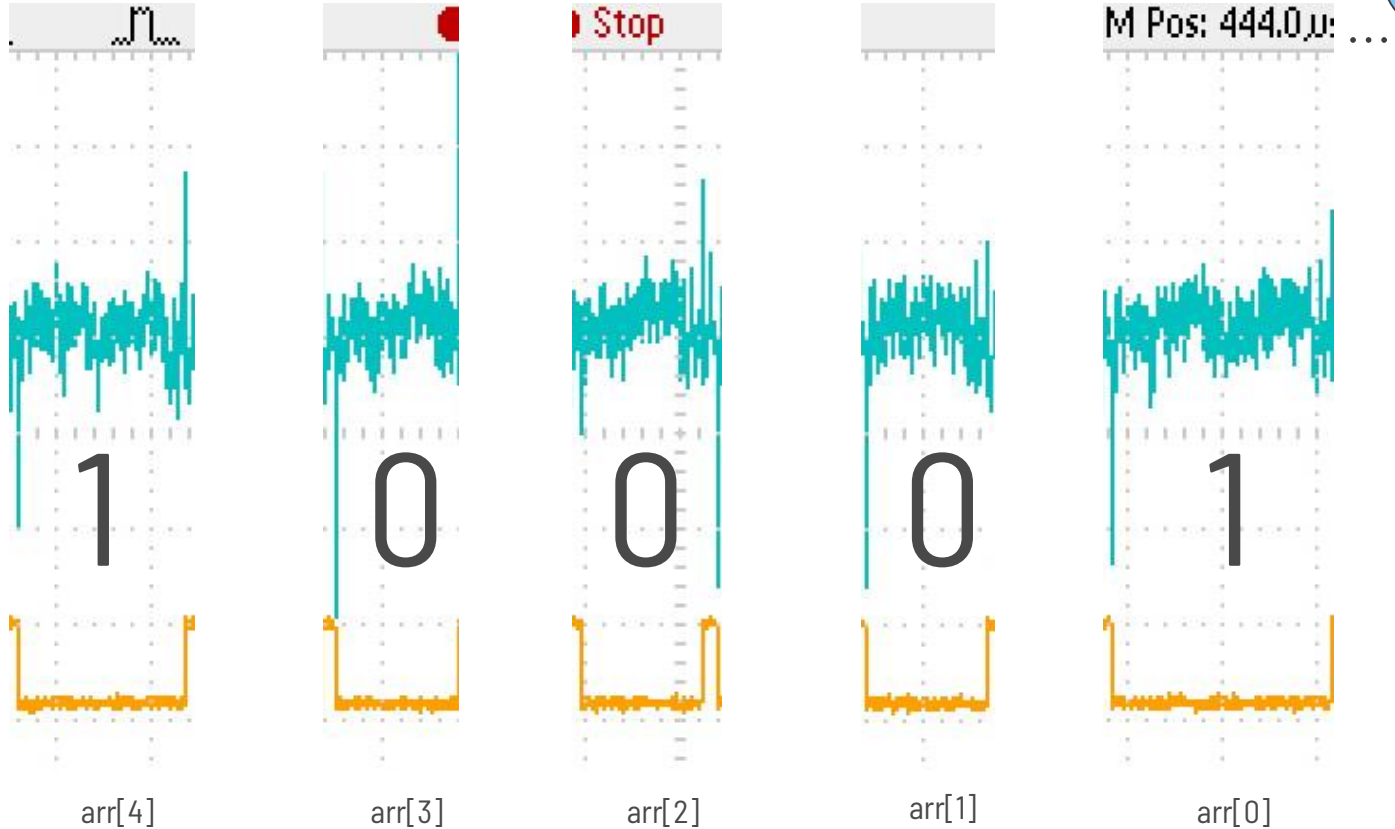
# Segmenting the Power Trace



# Segmenting the Power Trace



# Segmenting the Power Trace





# Review of Libgcrypt RSA Implementation

Blind-Folded: Simple Power Analysis Attacks using Data with a  
Single Trace and no Training

...

# Libgcrypt RSA(traditional) Implementation

```
53 void loop() {
54
55     temp = received_encrypted_message;
56     M = 1;
57     d = 17;
58
59     Serial.println("Running RSA...");
60
61     digitalWrite(11, LOW);
62     // Libgcrypt implementation of the traditional exponentiation function, simplified
63     for (int i = index - 1; i >= 0; i--)
64     {
65         digitalWrite(12, LOW);
66         // Do the square and multiply function
67         rp = (rp * rp) % n;      // rp contains the squared only value
68         xp = (rp * temp) % n;    // xp contains the squared and multiplied value
69
70         // Determine which one to take
71         rp = SetCond(rp, xp, intarr[i] );
72     }
73
74     digitalWrite(11, HIGH);
75 }
```

# Libgcrypt RSA(traditional) Implementation

**Algorithm 5** Simplified iteration of the main loop of the traditional exponentiation function in Libgcrypt. Bold variables indicate large integers.

▷ ***rp*** contains the current result

1:  $xp \leftarrow rp \times rp$

▷ Squaring

2:  $xp, rp \leftarrow rp, xp$

▷ Swapping ***rp*** and ***xp***

3:  $xp \leftarrow rp \times b$

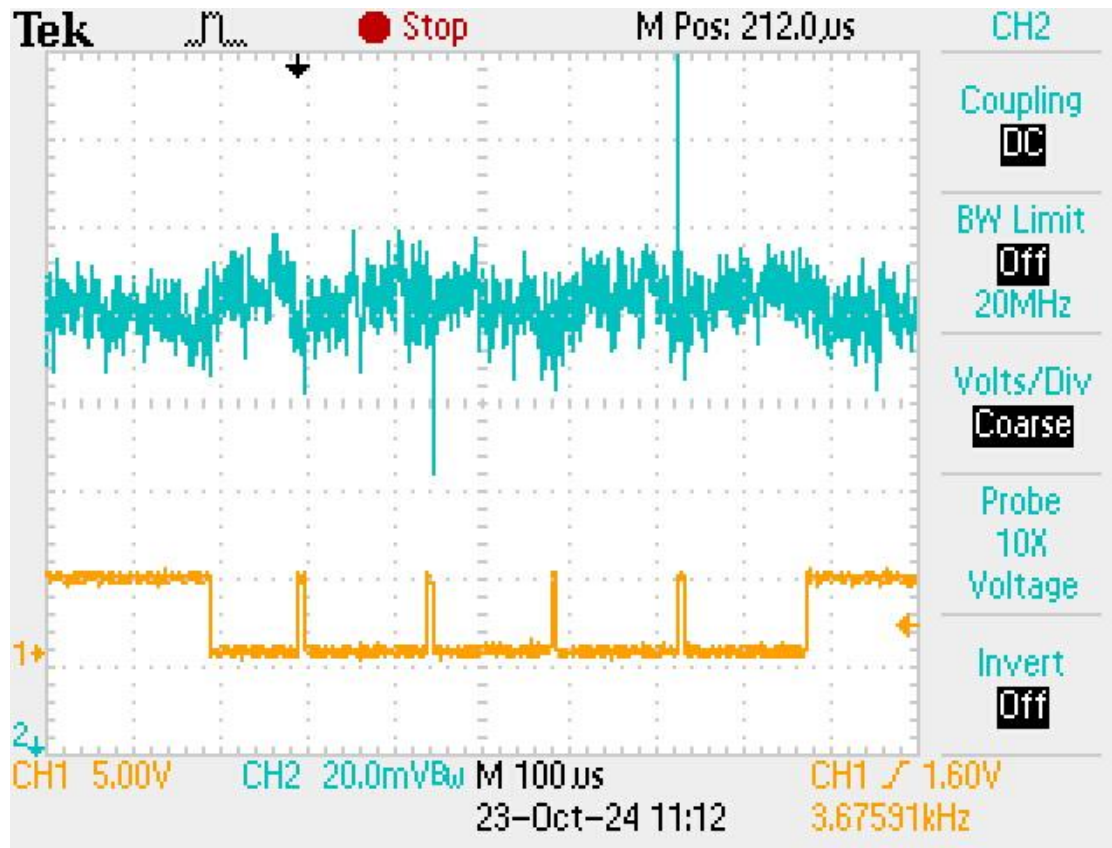
▷ Multiplication

4:  $rp \leftarrow \text{SETCOND}(rp, xp, e_i)$

▷ Keeping correct result depending on exponent bit

```
44 // Simplified version of libgcrypt implementation
45 uint64_t SetCond(uint64_t square_only, uint64_t square_and_multiply, uint64_t exponent){
46
47     mask0 = wzero - exponent;
48     mask1 = exponent - wone;
49
50     return (mask0 & square_and_multiply) | (mask1 & square_only);
51 }
```

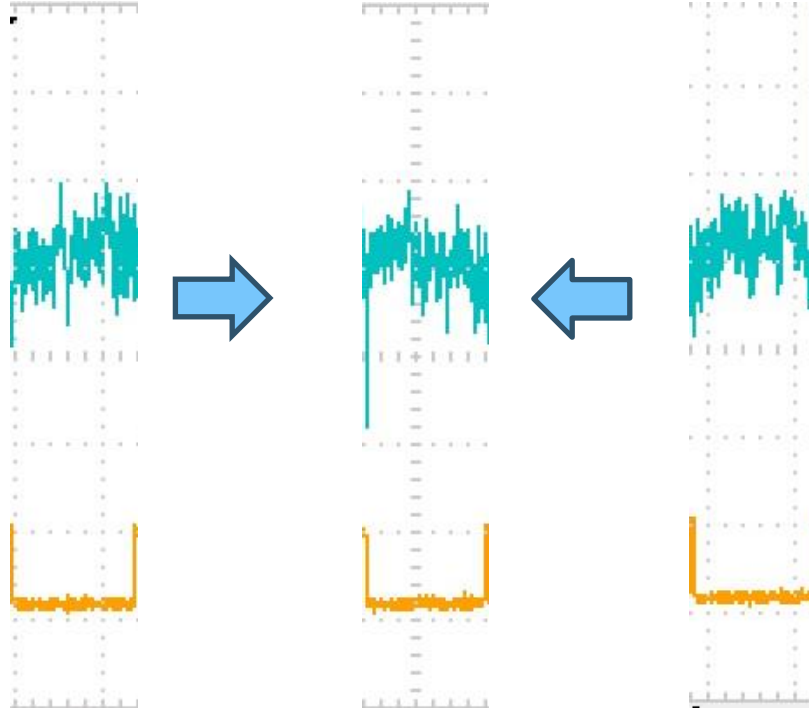
# Libgcrypt RSA(traditional) Implementation



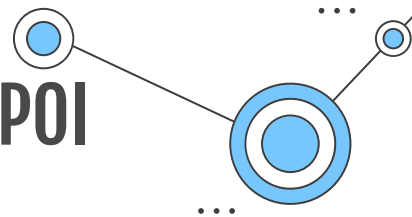


# Manually find a few ROI. Overlay them

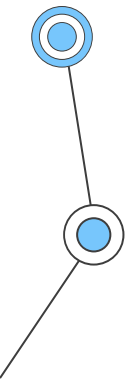
- ROI - Region Of Interest
  - Segment of the Trace that correspond to one bit



# Libgcrypt RSA(traditional) – Locating POI



-Back over to Google Colab-



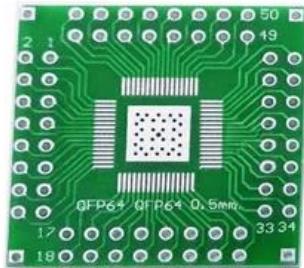
# Limitation/improvement for future



## Isolate chip

Capacitors and noise influence the power trace

...



## Hardware Limitations

Oscilloscope  
Accuracy



## RSA (Windowed) & ECDSA

Other attacks using  
SPA covered in the  
paper.



# Thanks!

Do you have any questions?

**CREDITS:** This presentation template was created by [Slidesgo](#), including icons by [Flaticon](#), infographics & images by [Freepik](#) and illustrations by [Stories](#)

