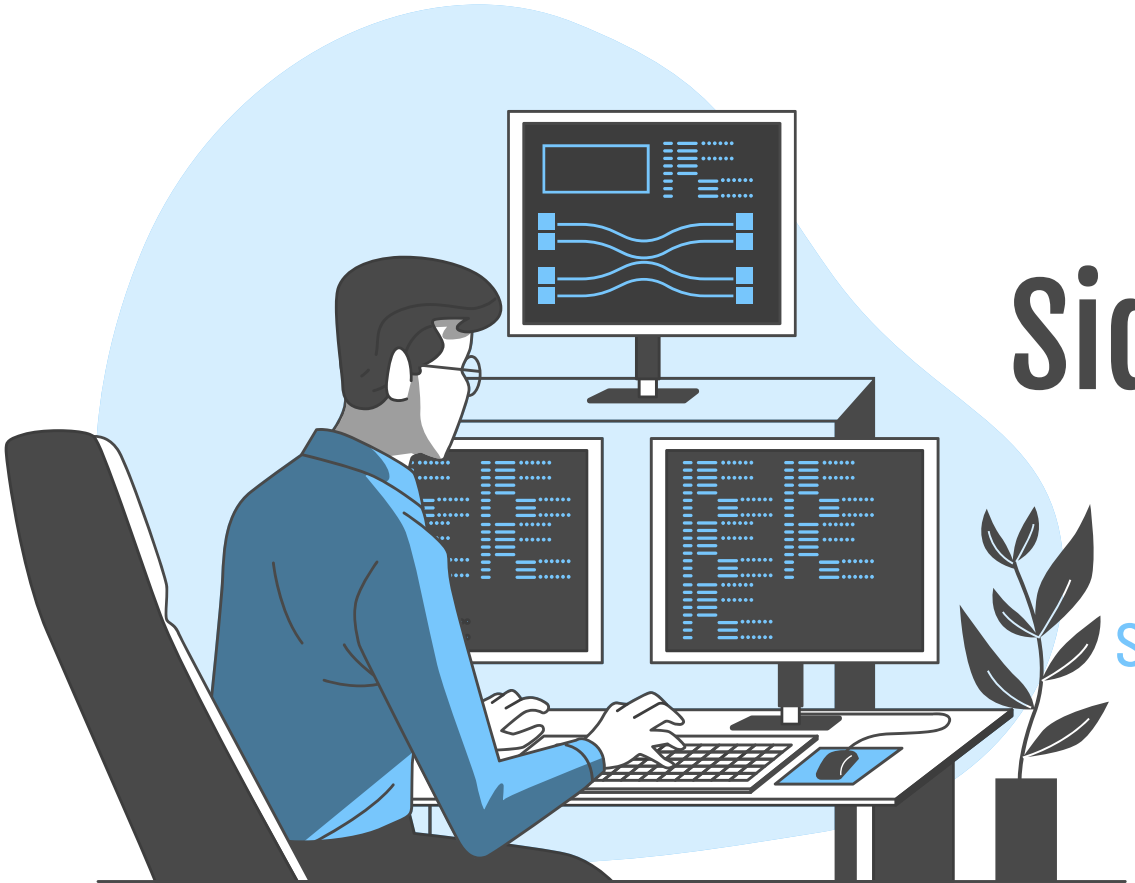
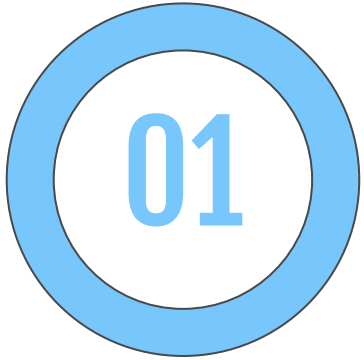


Side-Channel Attack

Simple Power Analysis on RSA Algorithm

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

Table of Content



Project Overview

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

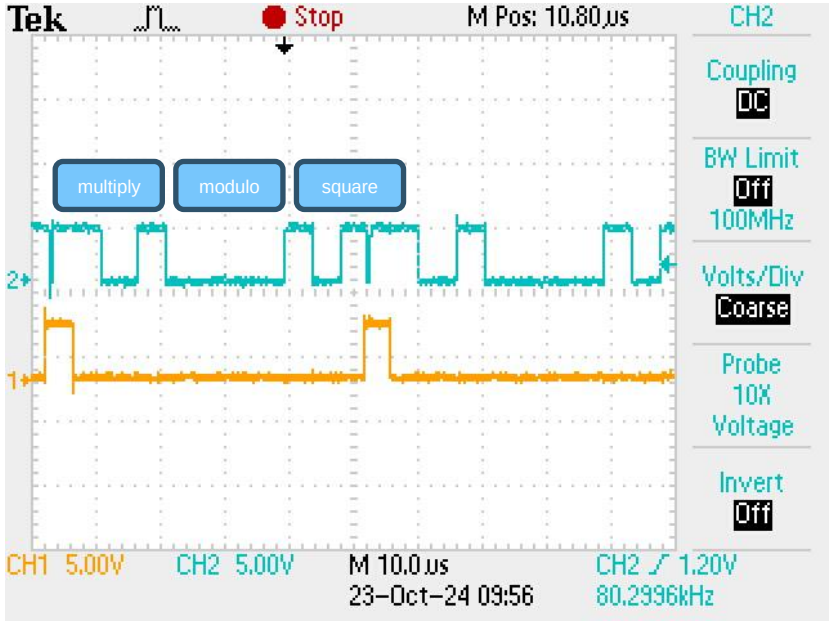
Project Overview

$num^{exp} \mid n$

$r = 1$

```
for each bit of exp:
    r = r * r % n
    if bit == 1:
        r = r * num % n
    return r
```

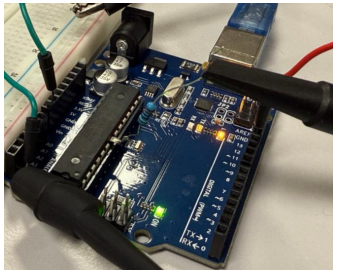
square
square
Square
multiply
square
multiply



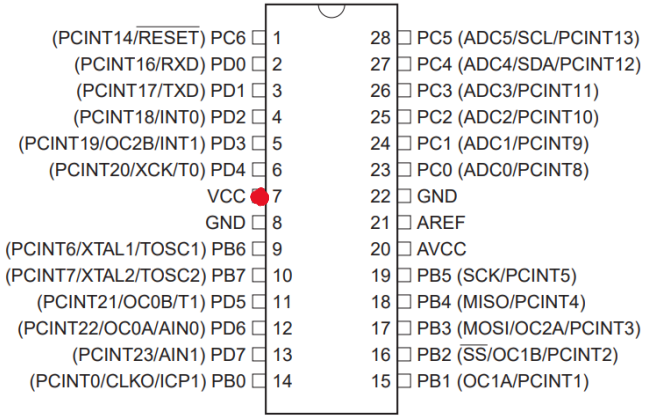
02

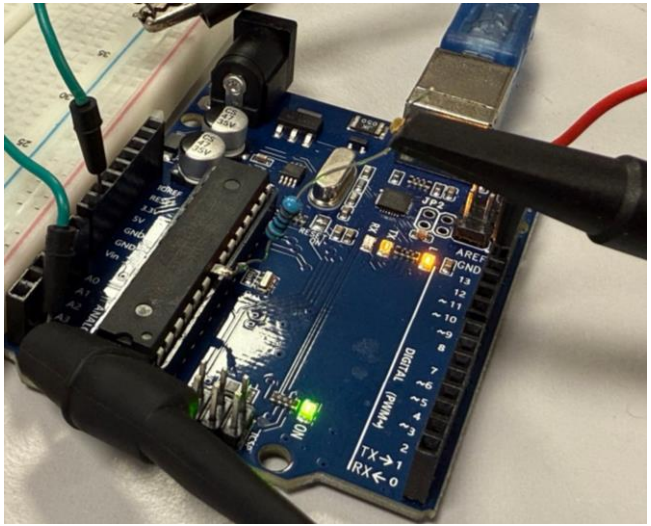
Simple Implementation of SPA (Arduino)

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

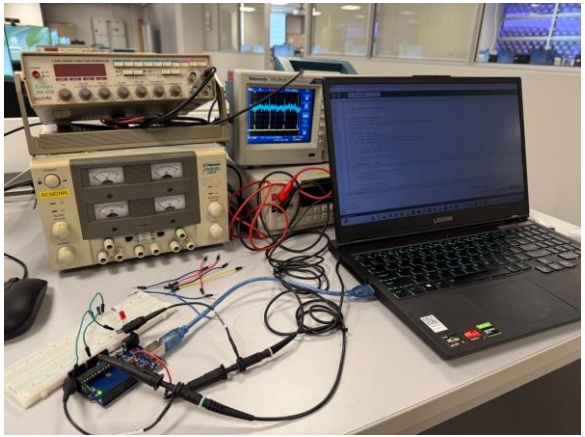


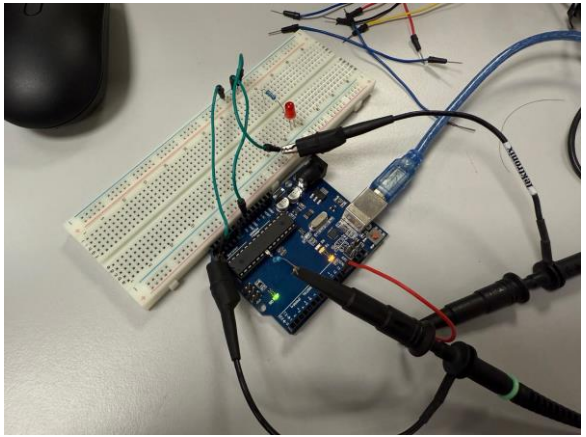
Simple Implementation of SPA (Arduino)





Simple Implementation of SPA (Arduino)





Simple Implementation of SPA (Arduino)

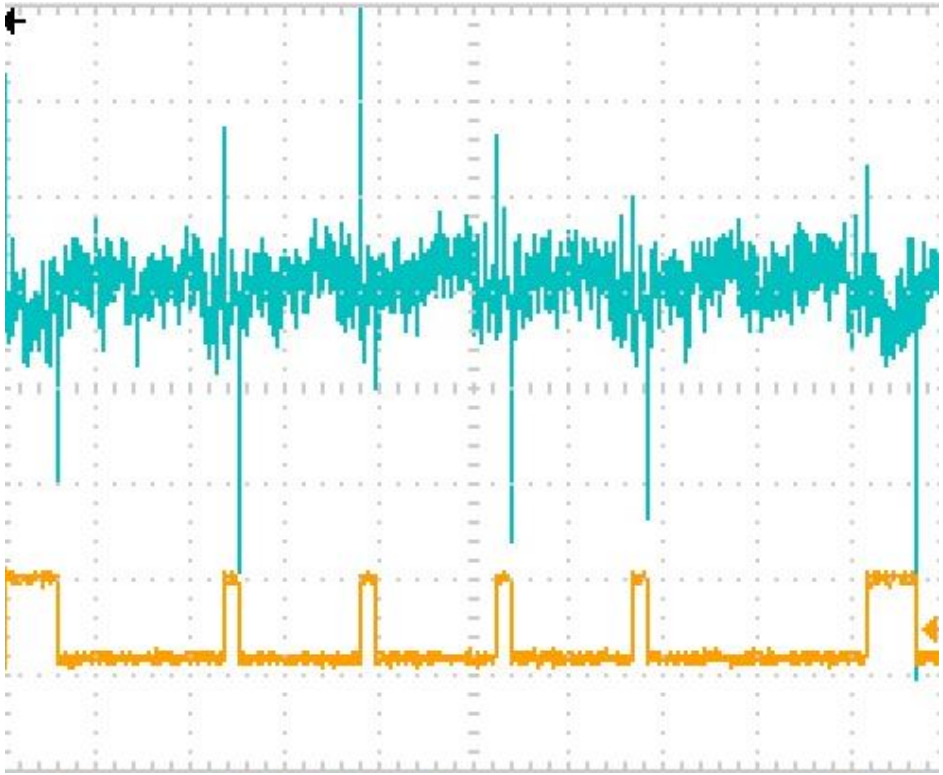
```ino
uint64_t n, p, q, e, d, M, received_encrypted_message, temp;

int intarr[64];   // Array of bits of decryption key, d
int index = 0;    // Length (in bits) of decryption key, d

void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  digitalWrite(11, HIGH);
  digitalWrite(12, HIGH);

  // RSA Parameters
  // To the attacker, p, q, d are unknown. n & e make up the public key
  p = 5; q = 11; e = 33; d = 17;

  n = p * q;  // the mod value

  // Original Message
  M = 2;

  // Simulate receiving of encrypted msg
  received_encrypted_message = 52;   // 2^33 mod 55

  // Calculate d in binary form and insert it into an array for ease of future calculations
  while (d>0){
    intarr[index] = d % 2;
    d = d >> 1;
    index = index + 1;
  }
}
```

```ino

void loop() {
  // put your main code here, to run repeatedly:

  temp = received_encrypted_message;
  M = 1;
  d = 17;

  Serial.println("Running RSA...");

  digitalWrite(11, LOW);
  // Modular Exponentiation
  for (int i = index - 1; i >= 0; i--)
  {
    digitalWrite(12, LOW);
    // Do the squaring first
    M = (M * M) % n;

    // If the bit is a 1, then an additional multiplication should be done
    if (intarr[i]){
      M = (M * temp) % n;
    }

    digitalWrite(12, HIGH);
  }
  digitalWrite(11, HIGH);
}
```

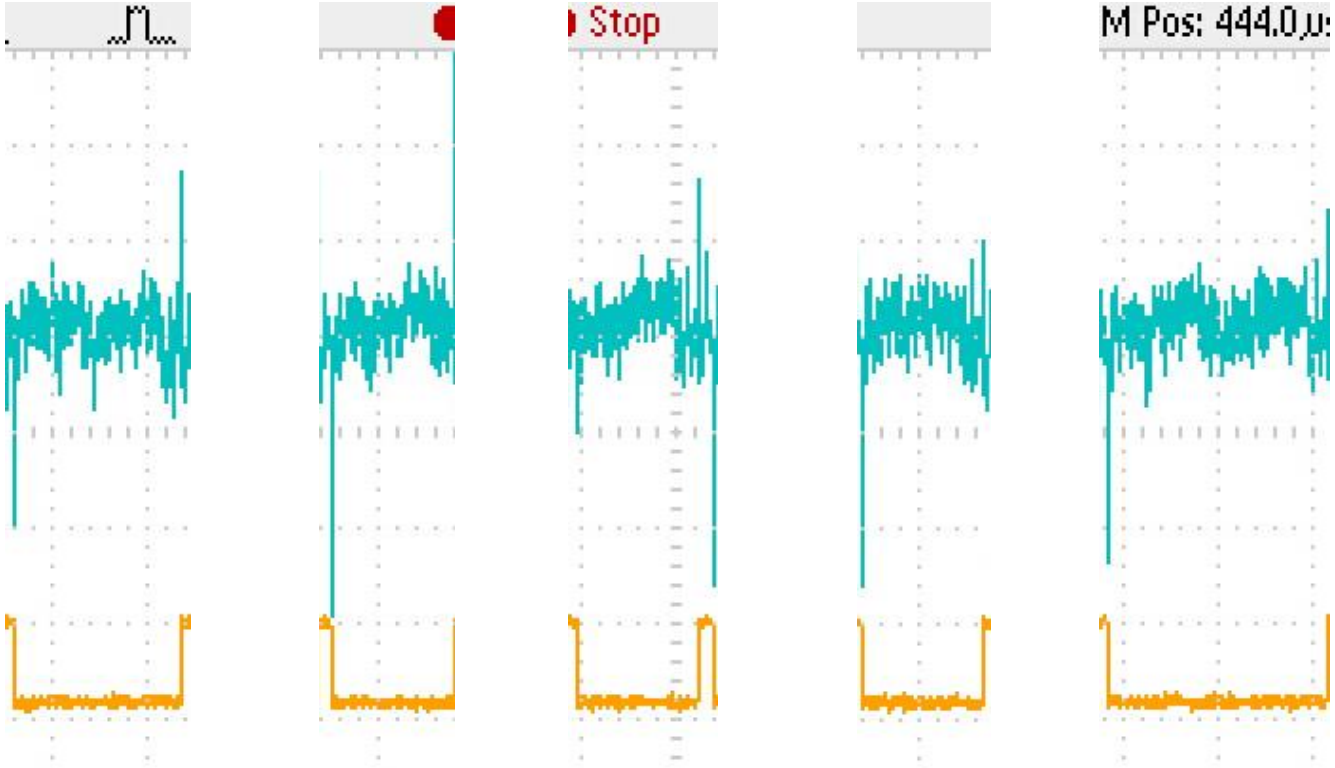
Code Explanation

-Moving over to Google Colab-

Segmenting the Power Trace

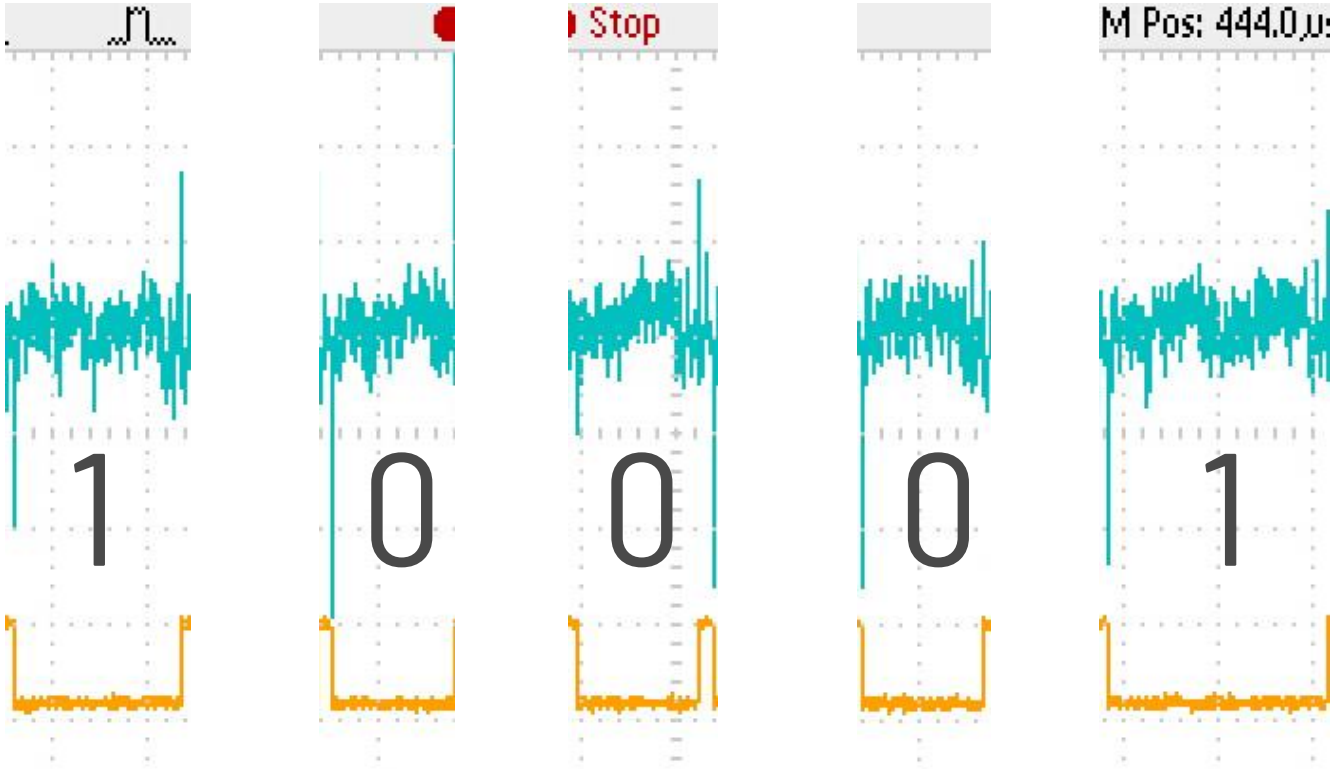


Segmenting the Power Trace



arr[4] arr[3] arr[2] arr[1] arr[0]

Segmenting the Power Trace



arr[4] arr[3] arr[2] arr[1] arr[0]

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

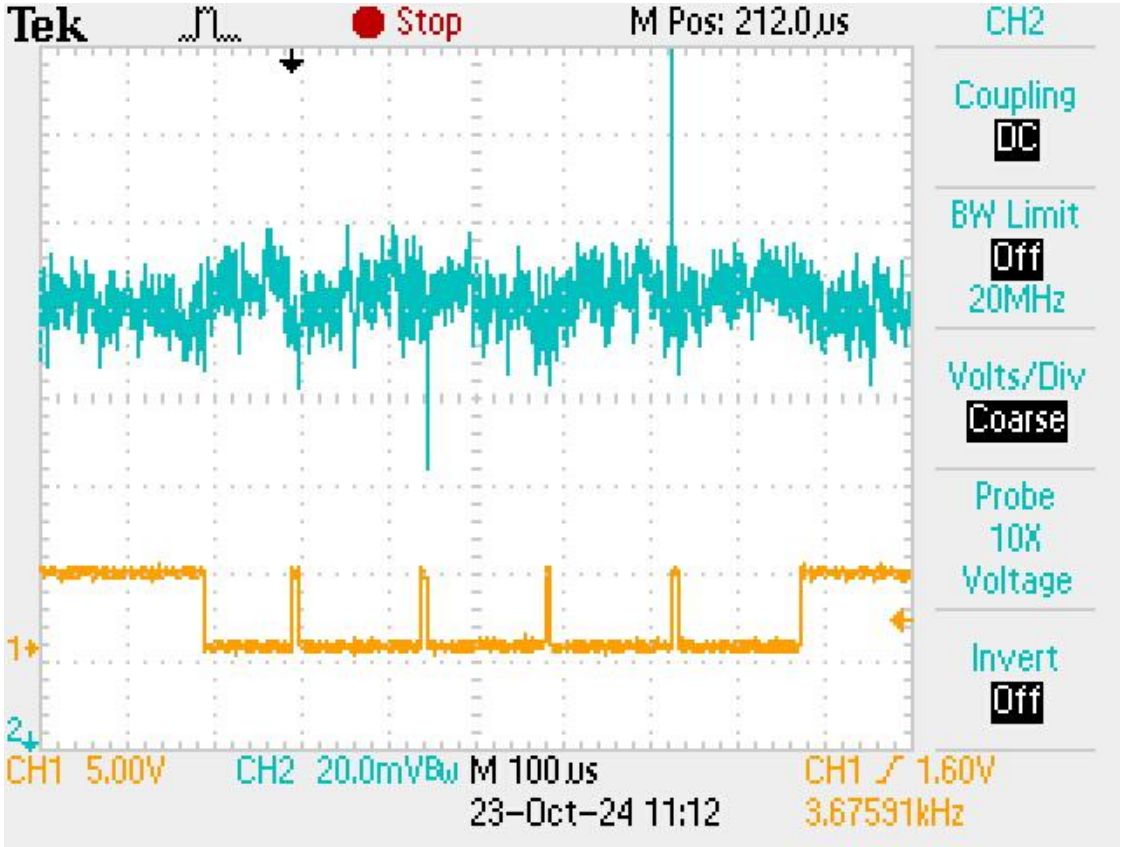
Algorithm 5 Simplified iteration of the main loop of the traditional exponentiation function in `Libgcrypt`. Bold variables indicate large integers.

▷ $\boldsymbol{rp}$ contains the current result

1: $\boldsymbol{xp} \leftarrow \boldsymbol{rp} \times \boldsymbol{rp}$ ▷ Squaring
2: $\boldsymbol{xp}, \boldsymbol{rp} \leftarrow \boldsymbol{rp}, \boldsymbol{xp}$ ▷ Swapping $\boldsymbol{rp}$ and $\boldsymbol{xp}$
3: $\boldsymbol{xp} \leftarrow \boldsymbol{rp} \times \boldsymbol{b}$ ▷ Multiplication
4: $\boldsymbol{rp} \leftarrow \text{SetCond}(\boldsymbol{rp}, \boldsymbol{xp}, \mathtt{e}_i)$ ▷ Keeping correct result depending on exponent bit

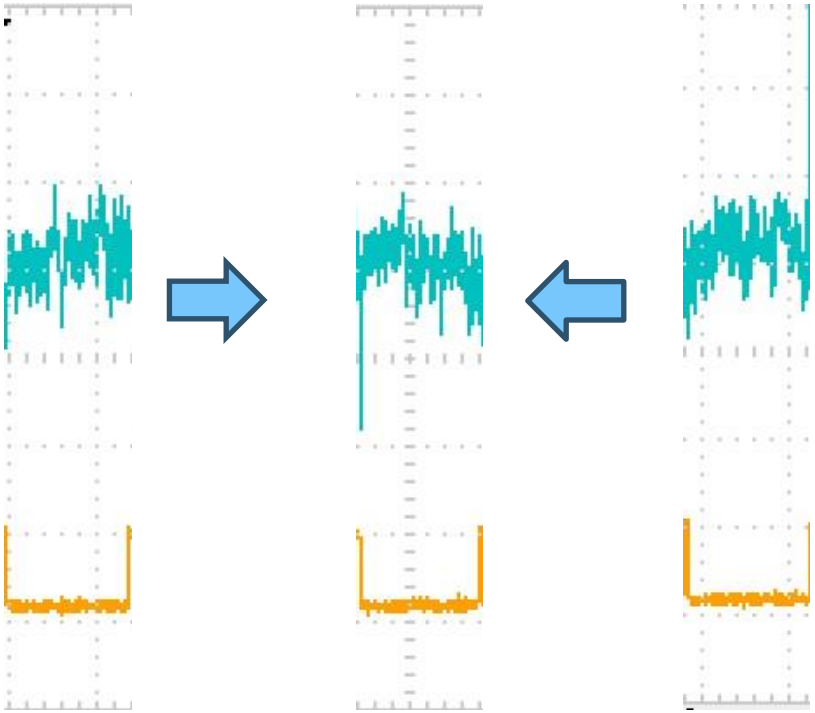
```
44  // Simplified version of libgcrypt implementation
45  uint64_t SetCond(uint64_t square_only, uint64_t square_and_multiply, uint64_t exponent){
46
47    mask0 = wzero - exponent;
48    mask1 = exponent - wone;
49
50    return (mask0 & square_and_multiply) | (mask1 & square_only);
51  }
```

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

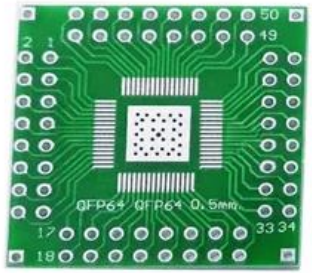
-Back over to Google Colab-

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?