Attaques par canaux auxiliaires

RSA - AES

M. Hostettler

Présentation de TIPE, Juin 2023

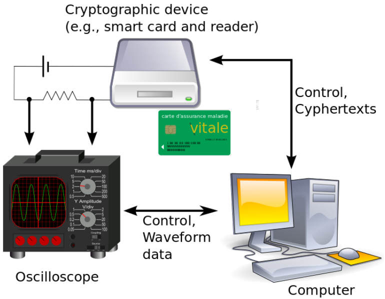
Plan

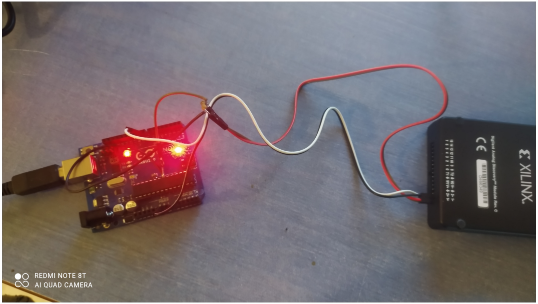
- Définition
- Problématique
- Mise en place du dispositif de mesures
  - Mesure de courant
  - Première trace RSA
- Attaque sur RSA
  - L'algorithme
  - Allure de la courbe
  - Timing attack
- Attaque sur AES
  - L'algorithme
  - Attaque CPA
  - Corrélation de Pearson
- Contre-mesure et conclusion

Définition

Une attaque par canal auxiliaire (en anglais: Side-channel attack) est une attaque qui vise à exploiter l'implémentation de certains systèmes cryptographiques par le biais d'informations "s'échappant" du système.

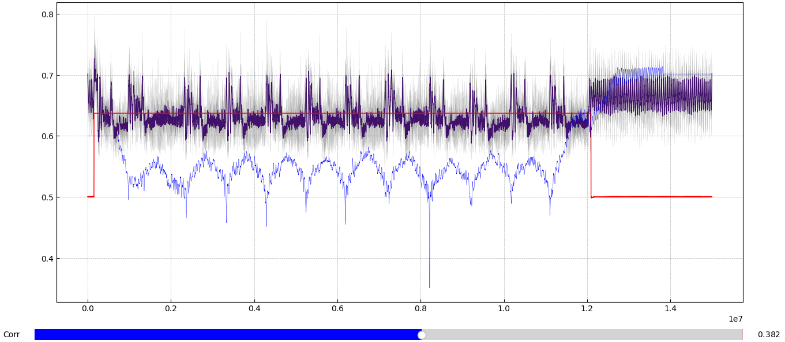
Mesure de courant





Attaque par mesure de courant à travers une résistance de 50Ω

Première Trace RSA



En violet la courbe de consommation avec du filtrage de Savitzky–Golay et en bleu une courbe de corrélation par fenêtre glissante

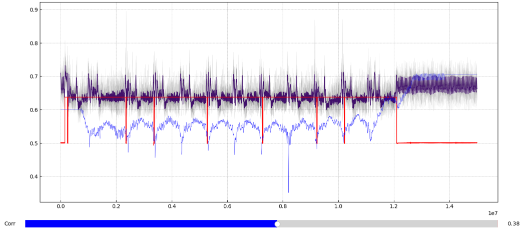
L'algorithme RSA

L'exponentiation rapide est utilisée :

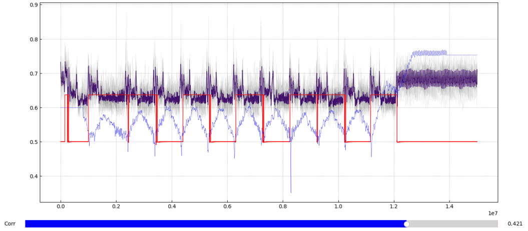
```
int powm(int x, int y, int p){
    int res = 1;
    while (y > 0) {
        if (y & 1){
          res = (res * x) % p;
        }
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}
```

En s'aidant de pulsations

```
int powm(int x, int y, int p){
    int res = 1;
    while (y > 0) {
        PORTB = B00000000;
        PORTB = B11111111;
        if (y & 1){
          res = (res * x) % p;
        }
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}
```




```
int powm(int x, int y, int p){
    int res = 1;
    while (y > 0) {
        PORTB = B00000000;
        PORTB = B11111111;
        if (y & 1){
          PORTB = B00000000;
          res = (res * x) % p;
          PORTB = B11111111;
        }
        y = y >> 1;
        x = (x * x) % p;
    }
    return res;
}
```

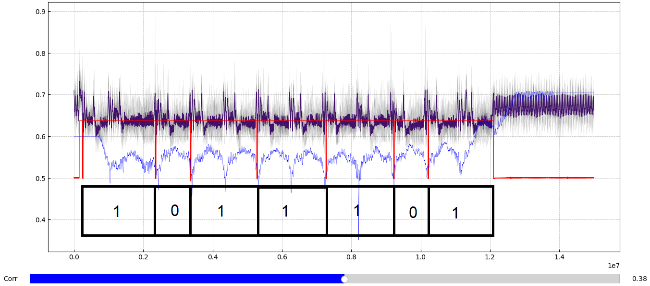


Timing

Idée

Algorithme de Boyer-Moore naïf

```
diffes = []
pt1 = pt.PowerTrace(power_trace.trace).filter()
for i in range(0, power_trace.size - rsa_ref.size):

    diff = pt1[i:(i + rsa_ref.size)] - rsa_ref
    diffes.append(np.sum(np.abs(diff)))
```



L'algorithme AES

AES (Advanced encryption standard) est un algorithme de cryptographie symétrique basé sur le schéma suivant :

```
fun AES(State, Cipherkey)
    KeyExpansion(CipherKey,ExpandedKey)
    AddRoundKey(State,ExpandedKey[0])
    for i = 1 to Nr - 1 do
        Round(State,ExpandedKey[i])
    end for
    FinalRound(State,ExpandedKey[Nr])
```

```
fun Round(State,ExpandedKey[i])
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State,ExpandedKey[i]);

fun FinalRound(State,ExpandedKey[Nr])
    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State,ExpandedKey[Nr]);
```

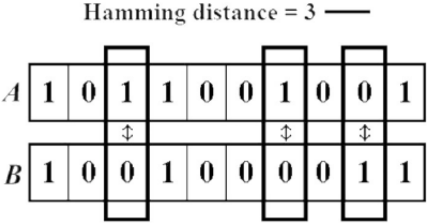
À l'attaque !!

Idée : Construire un modèle de consommation puis utiliser un outil statistique. On "devine" (bruteforce) la sortie de la sbox, donc on en déduit l'entrée car on connaît le text en claire.

Modèle

On considère la consommation du circuit proportionelle au poids de Hamming

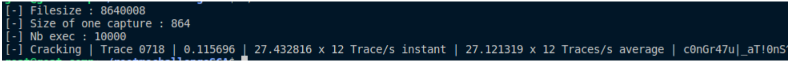
$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$



Le code :

```python
HammingWeight = np.array([bin(n).count("1") for n in range(0, 256)])


def intermediate(pt, keyguess):
    return sbox[pt ^ keyguess]

bestguess = np.zeros(16, dtype=np.float64)

numtraces = numtraces // 10
#meant = np.mean(traces, axis=0, dtype=np.float64)
hyp = np.zeros(numtraces)
for bnum in range(0, 16):
    cpaoutput = [0]*256
    maxcpa = np.zeros(256, dtype=np.float64)
    for kguess in range(0, 256):
        print("Subkey %d, hyp = %02x" % (bnum, kguess))

        sumnum1 = np.zeros(numpoint)
        sumnum2 = np.zeros(numpoint)
        sumnum3 = np.zeros(numpoint)
        sumden1 = np.zeros(numpoint)
        sumden2 = np.zeros(numpoint)

        for tnum in range(0, numtraces):
            hyp[tnum] = HammingWeight[intermediate(kp[tnum][bnum], kguess)]

        for tnum in range(0, numtraces):
            sumnum1 += hyp[tnum] * traces[tnum]
            sumnum2 += hyp[tnum]
            sumnum3 += traces[tnum]
            sumden1 += hyp[tnum] * hyp[tnum]
            sumden2 += traces[tnum] * traces[tnum]

        currnum = numpoint
        cpaoutput[kguess] = (currnum * sumnum1 - sumnum2 * sumnum3) / np.sqrt( (currnum * sumden1 - sumnum2 * sumnum2) * (currnum * s
        maxcpa[kguess] = max(abs(cpaoutput[kguess]))

        print(maxcpa[kguess])
    bestguess[bnum] = np.argmax(maxcpa)

```

Résultats et code en C optimisé :



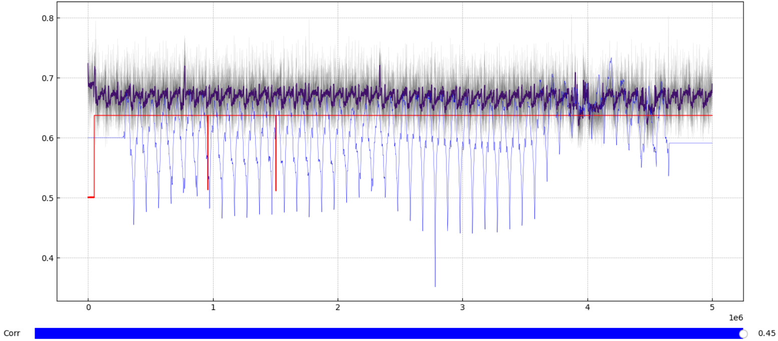
Architecture pour le cas de l'attaque réelle

- Code en C optimisé pour un calcul rapide (library Shared Object)
- Interfaçage avec Python grâce à ctypes
- Script Python communiquant avec l'oscilloscope
- Code Arduino AES en C/C++

Contre-mesure

Exemple de contre mesure pour RSA

On rajoute un : `while (rand() % 10 != 0)`



Impossible de mesurer des temps !!

Conclusion

Merci pour votre attention

Annexe : L'algorithme RSA

RSA (Rivest–Shamir–Adleman) est un algorithme de cryptographie asymétrique basé sur le schéma suivant :
Soit $(p, q) \in \mathbb{P}^2$ et $n = pq, \phi(n) = (p-1)(q-1)$
On prend alors $e \in \mathbb{N}, e \wedge \phi(n) = 1$
La clef privée d est tel que $ed \equiv 1 \pmod{\phi(n)}$ donc d inverse modulaire de e.
Pour un message m on le chiffre par : $c \equiv m^e \pmod{n}$
Et on le déchiffre par : $m \equiv c^d \pmod{n}$

Annexe : Code C calcul I

```c
/*
Original[KEY_SIZE, BLOCK_SIZE, bytes_per_capture]
then you could turn it into Flat[KEY_SIZE * BLOCK_SIZE * bytes_per_capture] by

Flat[x + BLOCK_SIZE * (y + bytes_per_capture * z)] = Original[x, y, z]
*/
#define XYZ(x, y, z) ((z) + ((cracker->bytes_per_capture) * ((x) + ((KEY_SIZE) * (y)))))
// not used anymore for optimisation purposes but makes understanding easier

void AddTrace(cpa_cracker cracker, uint8_t* trace, uint8_t* kp){
    double hyp = 0.0;
    double bestguessed = -INFINITY;
    double guessedIndice = 0;

    #ifdef VERBOSE
        uint32_t nt = 0;
        double cl = omp_get_wtime();
        printf(CLEARLINE "\r[%c] Cracking | Trace %04d |", roll[cracker->nb_trace % 4], (uint32_t)cracker->nb_trace);
    #endif

    cracker->nb_trace++;

    uint32_t coord = 0;
    uint32_t cp = 0;

    #pragma omp parallel for simd collapse(2)
    for (uint8_t bnum = 0; bnum < KEY_SIZE; bnum++){
```

Annexe : Code C calcul II

```c
        for (uint16_t kguess = 0; kguess < BLOCK_SIZE; kguess++){
            #ifdef VERBOSE
                nt = DMAX(nt, omp_get_num_threads());
            #endif

            coord = cracker->bytes_per_capture * (bnum + KEY_SIZE * kguess);
            hyp = (double)((uint32_t)HammingWeight[INTERMEDIATE((uint16_t)kp[bnum],
    kguess)]);

            for (uint32_t i = 0; i < cracker->bytes_per_capture; i++){
                cp = coord + i;
                cracker->sumnum1[cp] += hyp * trace[i];
                cracker->sumnum2[cp] += hyp;
                cracker->sumnum3[cp] += trace[i];
                cracker->sumden1[cp] += hyp * hyp;
                cracker->sumden2[cp] += trace[i] * trace[i];
                //printf("%lf\n", cracker->sumnum2[XYZ(bnum, kguess, i)]);
            }

            for (uint32_t i = coord; i < coord + cracker->bytes_per_capture; i++){

                double inroot = (cracker->sumnum2[i] * cracker->sumnum2[i]
                    - cracker->nb_trace * cracker->sumden1[i])
                    * (cracker->sumnum3[i] * cracker->sumnum3[i]
                    - cracker->nb_trace * cracker->sumden2[i]);

                if (inroot != 0.0)
                    cracker->cpaoutput[i] = (cracker->nb_trace * cracker->sumnum1[i]
```

Annexe : Code C calcul III

```c
                        - cracker->sumnum2[i] * cracker->sumnum3[i] ) / sqrt(fabs(inroot));

                //printf("%lf, %lf \n", inroot, (cracker->nb_trace * cracker->sumnum1[XYZ(bnum, kguess, i)] - cracker->sumnum2[XYZ(bnum, kguess, i)] * cracker->sumnum3[XYZ(bnum, kguess, i)] ) );
            }

            cracker->maxcpa[kguess + bnum*BLOCK_SIZE] = -INFINITY;
            for (uint32_t i = coord; i < coord + cracker->bytes_per_capture; i++)
                cracker->maxcpa[kguess + bnum*BLOCK_SIZE] = DMAX(cracker->maxcpa[kguess + bnum*BLOCK_SIZE], fabs(cracker->cpaoutput[i]));
        }
    }

    for (uint8_t bnum = 0; bnum < KEY_SIZE; bnum++){
        bestguessed = -INFINITY;
        guessedIndice = 0;
        for (uint16_t kguess = 0; kguess < BLOCK_SIZE; kguess++)
            if (cracker->maxcpa[kguess + bnum*BLOCK_SIZE] > bestguessed){
                bestguessed = cracker->maxcpa[kguess + bnum*BLOCK_SIZE];
                guessedIndice = kguess;
            }
        cracker->bestguess[bnum] = guessedIndice;
    }

    #ifdef VERBOSE
        double t = 1 / (double)(omp_get_wtime() - cl);
        cracker->AverageTracePerSec += t;
```

Annexe : Code C calcul IV

```c
            printf(" %lf | %lf x %d Trace/s instant | %lf x %d Traces/s average", cracker
        ->maxcpa[0], t, nt, cracker->AverageTracePerSec / cracker->nb_trace, nt);
            printf(" | ");
            int c = 0;
            for (uint32_t i = 0; i < KEY_SIZE; i++){
                c = (int)cracker->bestguess[i];
                if ((c <= 0x7e) && (c >= 0x20))
                    printf("%c", (char)c);
                else
                    printf(RED "\\x%02X" RESET, (unsigned)c);
            }

            fflush(stdout);
        #endif
}
```

Annexe : Code C Arduino I

```c
#ifndef ECB
  #define ECB 1
#endif

#define AES128 1
//#define AES192 1
//#define AES256 1

#define AES_BLOCKLEN 16 // Block length in bytes - AES is 128b block only

#if defined(AES256) && (AES256 == 1)
    #define AES_KEYLEN 32
    #define AES_keyExpSize 240
#elif defined(AES192) && (AES192 == 1)
    #define AES_KEYLEN 24
    #define AES_keyExpSize 208
#else
    #define AES_KEYLEN 16   // Key length in bytes
    #define AES_keyExpSize 176
#endif

struct AES_ctx
{
  uint8_t RoundKey[AES_keyExpSize];
};
typedef uint8_t state_t[4][4];
static const uint8_t sbox[256] = {
  //0     1    2     3     4    5     6     7      8     9     A      B     C     D
      E     F
```

Annexe : Code C Arduino II

```c
0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
```

Annexe : Code C Arduino III

```
  0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
     0x28, 0xdf,
  0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
     0xbb, 0x16 };

#define Nb 4

#if defined(AES256) && (AES256 == 1)
    #define Nk 8
    #define Nr 14
#elif defined(AES192) && (AES192 == 1)
    #define Nk 6
    #define Nr 12
#else
    #define Nk 4
    #define Nr 10
#endif

static const uint8_t Rcon[11] = {
  0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36 };
struct AES_ctx ctx;



#define getSBoxValue(num) (sbox[(num)])

static void KeyExpansion(uint8_t* RoundKey, const uint8_t* Key)
{
  unsigned i, j, k;
```

Annexe : Code C Arduino IV

```
    uint8_t tempa[4];

    for (i = 0; i < Nk; ++i)
    {
      RoundKey[(i * 4) + 0] = Key[(i * 4) + 0];
      RoundKey[(i * 4) + 1] = Key[(i * 4) + 1];
      RoundKey[(i * 4) + 2] = Key[(i * 4) + 2];
      RoundKey[(i * 4) + 3] = Key[(i * 4) + 3];
    }

    for (i = Nk; i < Nb * (Nr + 1); ++i)
    {
      {
        k = (i - 1) * 4;
        tempa[0]=RoundKey[k + 0];
        tempa[1]=RoundKey[k + 1];
        tempa[2]=RoundKey[k + 2];
        tempa[3]=RoundKey[k + 3];

      }

      if (i % Nk == 0)
      {
        {
          const uint8_t u8tmp = tempa[0];
          tempa[0] = tempa[1];
          tempa[1] = tempa[2];
          tempa[2] = tempa[3];
          tempa[3] = u8tmp;
```

Annexe : Code C Arduino V

```
      }

      {
        tempa[0] = getSBoxValue(tempa[0]);
        tempa[1] = getSBoxValue(tempa[1]);
        tempa[2] = getSBoxValue(tempa[2]);
        tempa[3] = getSBoxValue(tempa[3]);
      }

      tempa[0] = tempa[0] ^ Rcon[i/Nk];
    }
    j = i * 4; k=(i - Nk) * 4;
    RoundKey[j + 0] = RoundKey[k + 0] ^ tempa[0];
    RoundKey[j + 1] = RoundKey[k + 1] ^ tempa[1];
    RoundKey[j + 2] = RoundKey[k + 2] ^ tempa[2];
    RoundKey[j + 3] = RoundKey[k + 3] ^ tempa[3];
  }
}

void AES_init_ctx(struct AES_ctx* ctx, const uint8_t* key)
{
  KeyExpansion(ctx->RoundKey, key);
}


static void AddRoundKey(uint8_t round, state_t* state, const uint8_t* RoundKey)
{
  uint8_t i,j;
  for (i = 0; i < 4; ++i){
```

Annexe : Code C Arduino VI

```
    for (j = 0; j < 4; ++j){
      (*state)[i][j] ^= RoundKey[(round * Nb * 4) + (i * Nb) + j];
    }
  }
}

static void SubBytes(state_t* state)
{
  uint8_t i, j;
  bool fst = true;
  for (i = 0; i < 4; ++i)
  {
    for (j = 0; j < 4; ++j)
    {
      (*state)[j][i] = getSBoxValue((*state)[j][i]);

      fst = false;
    }
  }
}

static void ShiftRows(state_t* state)
{
  uint8_t temp;

  temp           = (*state)[0][1];
  (*state)[0][1] = (*state)[1][1];
  (*state)[1][1] = (*state)[2][1];
  (*state)[2][1] = (*state)[3][1];
```

Annexe : Code C Arduino VII

```c
    (*state)[3][1] = temp;


    temp           = (*state)[0][2];
    (*state)[0][2] = (*state)[2][2];
    (*state)[2][2] = temp;

    temp           = (*state)[1][2];
    (*state)[1][2] = (*state)[3][2];
    (*state)[3][2] = temp;

    temp           = (*state)[0][3];
    (*state)[0][3] = (*state)[3][3];
    (*state)[3][3] = (*state)[2][3];
    (*state)[2][3] = (*state)[1][3];
    (*state)[1][3] = temp;
}

static uint8_t xtime(uint8_t x)
{
  return ((x<<1) ^ (((x>>7) & 1) * 0x1b));
}

static void MixColumns(state_t* state)
{
  uint8_t i;
  uint8_t Tmp, Tm, t;
  for (i = 0; i < 4; ++i)
  {
```

Annexe : Code C Arduino VIII

```c
      t   = (*state)[i][0];
      Tmp = (*state)[i][0] ^ (*state)[i][1] ^ (*state)[i][2] ^ (*state)[i][3] ;
      Tm  = (*state)[i][0] ^ (*state)[i][1] ; Tm = xtime(Tm);  (*state)[i][0] ^= Tm ^ Tmp ;
      Tm  = (*state)[i][1] ^ (*state)[i][2] ; Tm = xtime(Tm);  (*state)[i][1] ^= Tm ^ Tmp ;
      Tm  = (*state)[i][2] ^ (*state)[i][3] ; Tm = xtime(Tm);  (*state)[i][2] ^= Tm ^ Tmp ;
      Tm  = (*state)[i][3] ^ t ;              Tm = xtime(Tm);  (*state)[i][3] ^= Tm ^ Tmp ;
    }
}

#define Multiply(x, y)                                \
      (  ((y & 1) * x) ^                              \
      ((y>>1 & 1) * xtime(x)) ^                       \
      ((y>>2 & 1) * xtime(xtime(x))) ^                \
      ((y>>3 & 1) * xtime(xtime(xtime(x)))) ^         \
      ((y>>4 & 1) * xtime(xtime(xtime(xtime(x))))))   \



static void Cipher(state_t* state, const uint8_t* RoundKey)
{
  uint8_t round = 0;

  AddRoundKey(0, state, RoundKey);
  bool fst = true;
  for (round = 1; ; ++round)
```

Annexe : Code C Arduino IX

```
  {
    if (fst)
    PORTB = B11111111;
    SubBytes(state);

    if (fst)
    PORTB = B00000000;

    //fst = false;
    ShiftRows(state);
    if (round == Nr) {
      break;
    }
    MixColumns(state);
    AddRoundKey(round, state, RoundKey);
    fst = false;
  }
  // Add round key to last round
  AddRoundKey(Nr, state, RoundKey);
}

void AES_ECB_encrypt(const struct AES_ctx* ctx, uint8_t* buf)
{
  // The next function call encrypts the PlainText with the Key using AES algorithm.
  Cipher((state_t*)buf, ctx->RoundKey);
}

byte data[16] = {0};
void setup() {
```

Annexe : Code C Arduino X

```
  // put your setup code here, to run once:
  Serial.begin(57600);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);

  byte key[] = "TIPE_SUCCESS_!!!";
  AES_init_ctx(&ctx, key);
  randomSeed(analogRead(5));
}

void loop() {
  if (Serial.available() > 0) {
    // read the incoming byte:
    int incomingByte = Serial.read();
    if (incomingByte == 's'){
      for (uint32_t i = 0; i < 16; i++)
        data[i] = (byte)Serial.read();
      Serial.read(); // new line
    }
    if (incomingByte == 'c'){
      AES_ECB_encrypt(&ctx, data);
      PORTC |= data[0] & 1;
      Serial.read(); // new line
    }

  }
}
```

Annexe : Code Python CPA-lib I

```python
import ctypes
import os
import numpy as np

path = os.getcwd()
clibrary = ctypes.CDLL(os.path.join(path, 'Trace.so'))

BYTES_PER_KP = 16

class __CPA_Cracker(ctypes.Structure):
    _fields_= [("bytes_per_capture", ctypes.c_uint32),
               ("nb_trace", ctypes.c_uint32),
               ("bestguess", ctypes.POINTER(ctypes.c_double)),
               ("maxcpa", ctypes.POINTER(ctypes.c_double)),
               ("cpaoutput", ctypes.POINTER(ctypes.c_double)),
               ("sumnum1", ctypes.POINTER(ctypes.c_double)),
               ("sumnum2", ctypes.POINTER(ctypes.c_double)),
               ("sumnum3", ctypes.POINTER(ctypes.c_double)),
               ("sumden1", ctypes.POINTER(ctypes.c_double)),
               ("sumden2", ctypes.POINTER(ctypes.c_double)),
               ("AverageTracePerSec", ctypes.c_double)]

_CPA_Cracker = ctypes.POINTER(__CPA_Cracker)
clibrary.InitCracker.argtypes = [ctypes.c_uint32]
clibrary.InitCracker.restype = _CPA_Cracker

clibrary.AddTrace.argtypes = [_CPA_Cracker, ctypes.POINTER(ctypes.c_double), ctypes.
    POINTER(ctypes.c_uint8)]

```

Annexe : Code Python CPA-lib II

```python
class Cracker:
    def __init__(self, bytes_per_cap : int):
        if bytes_per_cap < 1:
            print("Bytes_per_cap must be strictly > 1")
            exit(1)
        self.__cr = clibrary.InitCracker(bytes_per_cap)
    def AddTrace(self, trace, kp):
        temp_trace = list(trace)
        temp_kp = list(kp)
        assert(len(temp_trace) == self.__cr.contents.bytes_per_capture)
        assert(len(temp_kp) == BYTES_PER_KP)

        c_tr = (ctypes.c_double * len(temp_trace))(*temp_trace)
        c_kp = (ctypes.c_uint8 * len(temp_kp))(*temp_kp)
        clibrary.AddTrace(self.__cr, c_tr, c_kp)
```

Annexe : Code Python PowerTrace I

```
import numpy as np
from scipy.signal import savgol_filter


class PowerTrace ():
    def __init__(self, data, name="power"):
        # Basic parameters
        self.trace = np.array(data)
        self.size = len(self.trace)
        self.name = name
        self.polyorder = 3
        self.window = 25
        self.filtered = 0

        # Compute Parameters
        self.mean = np.mean(self.trace)
        self.var = np.var(self.trace)
        self.normalized = self.trace - self.mean
        self.max = np.max(self.trace)
        self.min = np.min(self.trace)
        self.amplitude = np.abs(self.max - self.min)


    def __getitem__(self, index):
        return self.trace[index]

    def __add__(self, x):
        return PowerTrace(self.trace + x, name = self.name)

```

Annexe : Code Python PowerTrace II

```python
    def __sub__(self, x):
        return PowerTrace(self.trace - x, name = self.name)

    def __mul__(self, x):
        return PowerTrace(self.trace * x, name = self.name)

    def __truediv__(self, x):
        return PowerTrace(self.trace / x, name = self.name)

    def __len__(self):
        return self.size

    def filter(self, polyorder = 3, window = 30):
        if polyorder == self.polyorder and window == self.window and self.filtered != 0:
            return self.filtered
        self.polyorder = polyorder
        self.window = window

        self.filtered = PowerTrace(savgol_filter(self.trace, self.window, self.polyorder),
            name = "filtered_" + self.name)

        return self.filtered

    def selfcorrelate(self, polyorder = 3, window = 20):
        self.correlated = np.correlate(self.normalized, self.normalized, 'full')[self.size - 1:]
        self.correlated = self.correlated / self.var / self.size
```

Annexe : Code Python PowerTrace III

```python
        self.correlated = self.correlated / 20 + 1.2*self.mean

        self.correlated = PowerTrace(self.correlated, name = "correlated_" + self.name)

        return self.correlated
```

Annexe : Code Python Attaque finale I

```python
import matplotlib.pylab as plt
import numpy as np
import os
import CpaLib as cpa
from sys import platform, path
import ctypes
from os import sep
import powertrace as pt
from WF_SDK import device, scope # import instruments
from WF_SDK.scope import trigger_source
from binascii import unhexlify

if platform.startswith("win"):
    # on Windows
    dwf = ctypes.cdll.dwf
    constants_path = "C:" + sep + "Program Files (x86)" + sep + "Digilent" + sep + "WaveFormsSDK" + sep + "samples" + sep + "py"
elif platform.startswith("darwin"):
    # on macOS
    lib_path = sep + "Library" + sep + "Frameworks" + sep + "dwf.framework" + sep + "dwf"
    dwf = ctypes.cdll.LoadLibrary(lib_path)
    constants_path = sep + "Applications" + sep + "WaveForms.app" + sep + "Contents" + sep + "Resources" + sep + "SDK" + sep + "samples" + sep + "py"
else:
    # on Linux
    dwf = ctypes.cdll.LoadLibrary("libdwf.so")
    constants_path = sep + "usr" + sep + "share" + sep + "digilent" + sep + "waveforms" + sep + "samples" + sep + "py"
```

Annexe : Code Python Attaque finale II

```python

import numpy as np
from scipy.signal import savgol_filter

import serial
por = "/dev/ttyACM0"
ser = serial.Serial(por, 57600, timeout=10)
import time
time.sleep(3)

import os
import threading

device_data = device.open(config=0)
f =100*10**6
buff_size = 16384//2

timelength = buff_size / f
trigger_percent = 0.001

scope.open(device_data, sampling_frequency=f, buffer_size=buff_size)

v = ctypes.c_double()
dwf.FDwfAnalogInTriggerPositionSet(device_data.handle, ctypes.c_double(timelength*(0.5
     - trigger_percent)))
dwf.FDwfAnalogInTriggerPositionGet(device_data.handle ,ctypes.byref(v))

scope.trigger(device_data, True, trigger_source.analog, channel=2, edge_rising=True,
     level=2)
```

Annexe : Code Python Attaque finale III

```python

constants_path = sep + "usr" + sep + "share" + sep + "digilent" + sep + "waveforms" +
      sep + "samples" + sep + "py"
from sys import path
path.append(constants_path)
import dwfconstants as constants

def record_and_trig(device_data, channel):
    """
        record an analog signal

        parameters: - device data
                    - the selected oscilloscope channel (1-2, or 1-4)

        returns:    - a list with the recorded voltages
    """
    # set up the instrument
    dwf.FDwfAnalogInConfigure(device_data.handle, ctypes.c_bool(False), ctypes.c_bool(
      True))

    ser.write(b"c\n")
    # read data to an internal buffer
    while True:
        status = ctypes.c_byte()    # variable to store buffer status
        dwf.FDwfAnalogInStatus(device_data.handle, ctypes.c_bool(True), ctypes.byref(
      status))

        if status.value == constants.DwfStateDone.value:
            # exit loop when ready
```

Annexe : Code Python Attaque finale IV

```
                break

    # copy buffer
    buffer = (ctypes.c_double * buff_size)()   # create an empty buffer
    dwf.FDwfAnalogInStatusData(device_data.handle, ctypes.c_int(channel - 1), buffer,
     ctypes.c_int(buff_size))

    # convert into list
    buffer = [float(element) for element in buffer]
    return buffer

rec1 = record_and_trig(device_data, 2)


trig_buff = pt.PowerTrace(np.array(rec1))/20 + 0.5

mintrig = trig_buff.min
maxtrig = trig_buff.max
avgtrig = (mintrig + maxtrig) / 2
m = True
i = 1
for p in trig_buff:
    if p > avgtrig and m:
        m = False
    if p < avgtrig and not m:
        break
    i += 1
buff_size = i
timelength = buff_size / f
```

Annexe : Code Python Attaque finale V

```python
trigger_percent = 0.001


scope.close(device_data)
device.close(device_data)
###
device_data = device.open(config=0)
scope.open(device_data, sampling_frequency=f, buffer_size=buff_size)

v = ctypes.c_double()
dwf.FDwfAnalogInTriggerPositionSet(device_data.handle, ctypes.c_double(timelength*(0.5
    - trigger_percent)))
dwf.FDwfAnalogInTriggerPositionGet(device_data.handle ,ctypes.byref(v))

scope.trigger(device_data, True, trigger_source.analog, channel=2, edge_rising=True,
    level=2)

###############################################
#################"""

cracker = cpa.Cracker(buff_size)

try:
    while True:
        kp = os.urandom(16)
        ser.write(b"s" + kp + b"\n")


        tr = list((record_and_trig(device_data, 1)))
```

Annexe : Code Python Attaque finale VI

```python

        cracker.AddTrace(tr, list(kp))

        #import pdb; pdb.set_trace()
except KeyboardInterrupt:
    pass
ser.close()

###############
#traces = parseTraces("traces.bini8")
#numtraces = len(traces)
#numpoint = len(traces[0])
#kp = parseKP("plaintexts.hex")
#
#cracker = cpa.Cracker(numpoint)
#for i in range(numtraces):
#    cracker.AddTrace(traces[i], kp[i])

_ = input("Appuyer to stop")
# reset the scope
scope.close(device_data)


# close the connection
device.close(device_data)
```