PV181 Laboratory of security and applied cryptography

Seminar 12:
Crypto-libraries protected against hardware attacks

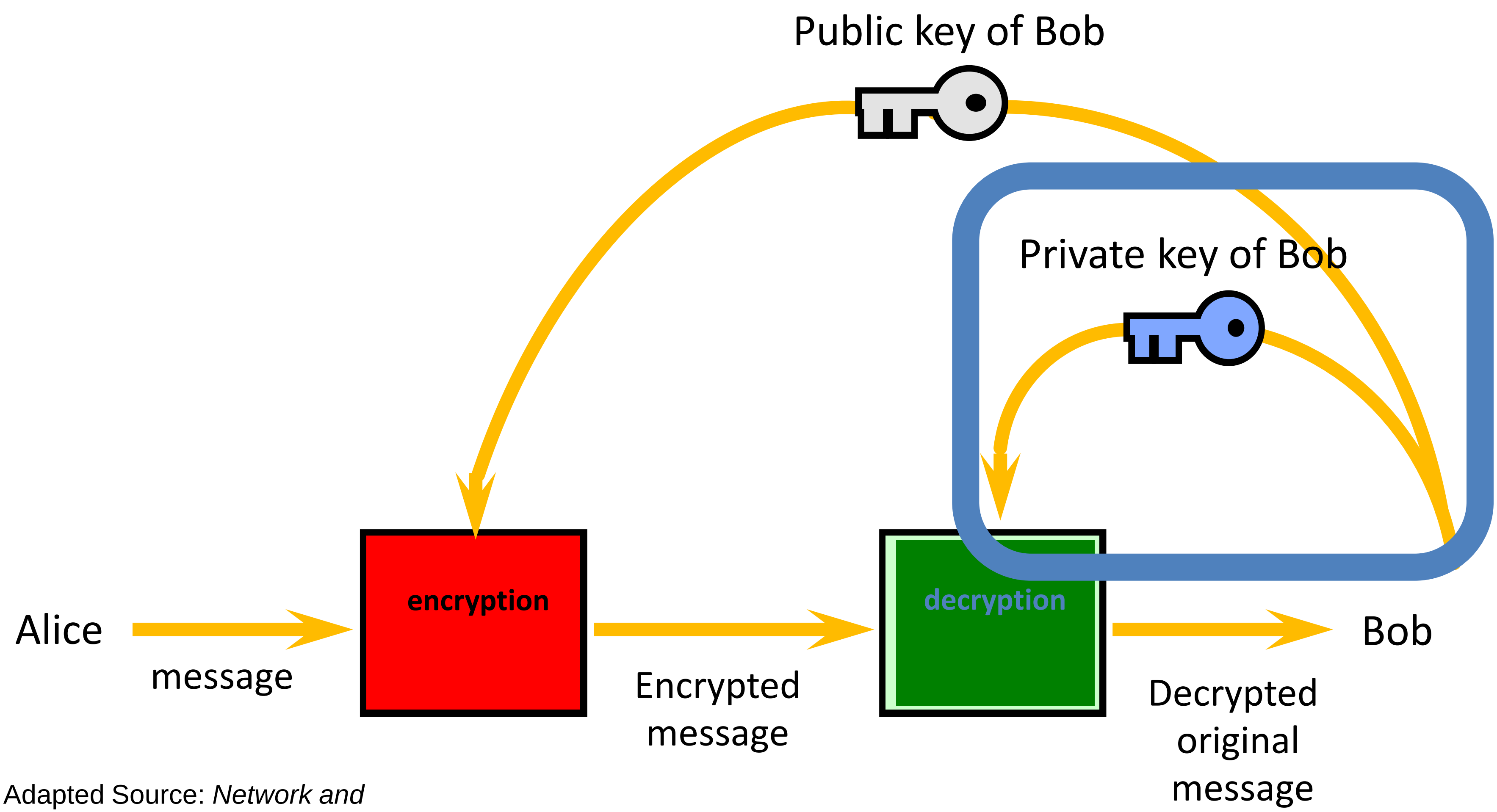
Łukasz Chmielewski
chmiel@fi.muni.cz

CRoCS
Centre for Research on Cryptography and Security

Outline

- Recall + goal of this seminar
  - Digital signatures
  - RSA vs. ECC
- Side Channel + Fault Injection speed run
- Secured X25519 library: sca25519
  - Optionally: Demo
- Assignment this week:
  - Securing RSA execution

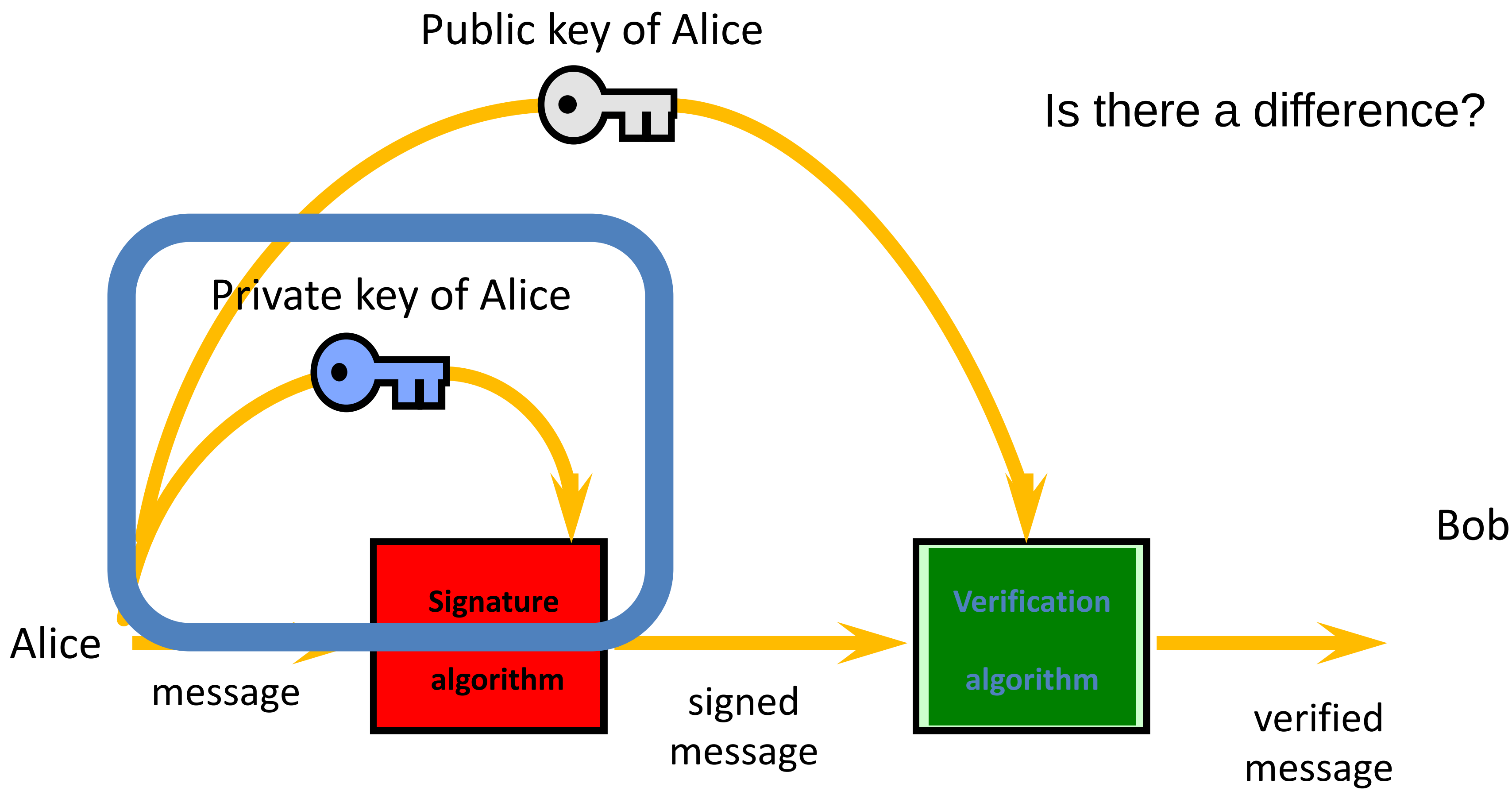
Recall: Asymmetric cryptosystem



Adapted Source: *Network and Internetwork Security* (Stallings)

Recall: Digital signature scheme

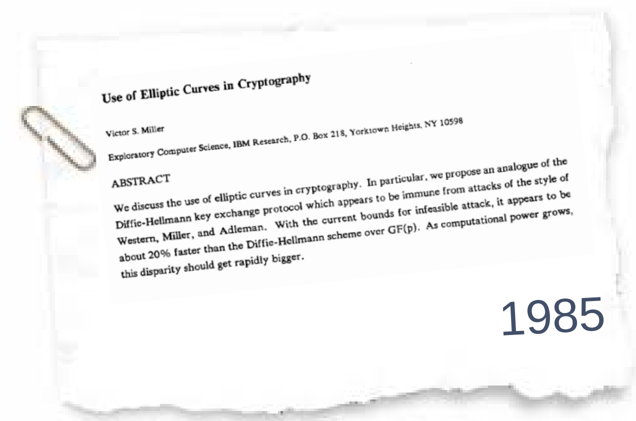
Is there a difference?



Source: *Network and Internetwork Security* (Stallings)

Recall: RSA vs. ECC

- exponentiation ≈ scalar multiplication
- multiplication ≈ points addition
- squaring ≈ point doubling



Use of Elliptic Curves in Cryptography

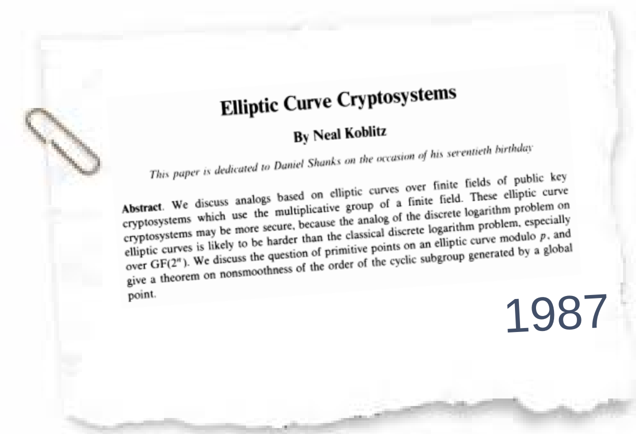
Victor S. Miller

Exploratory Computer Science, IBM Research, P.O. Box 218, Yorktown Heights, NY 10598

ABSTRACT

We discuss the use of elliptic curves in cryptography. In particular, we propose an analogue of the Diffie-Hellmann key exchange protocol which appears to be immune from attacks of the style of Western, Miller, and Adleman. With the current bounds for infeasible attack, it appears to be about 20% faster than the Diffie-Hellmann scheme over GF(p). As computational power grows, this disparity should get rapidly bigger.

1985



Elliptic Curve Cryptosystems

By Neal Koblitz

This paper is dedicated to Daniel Shanks on the occasion of his seventieth birthday

Abstract. We discuss analogs based on elliptic curves over finite fields of public key cryptosystems which use the multiplicative group of a finite field. These elliptic curve cryptosystems may be more secure, because the analog of the discrete logarithm problem on elliptic curves is likely to be harder than the classical discrete logarithm problem, especially over $GF(2^n)$. We discuss the question of primitive points on an elliptic curve modulo p, and give a theorem on nonsmoothness of the order of the cyclic subgroup generated by a global point.

1987

Why is hardware security important?

Card / Money Theft

Identity Theft

- **Premium**

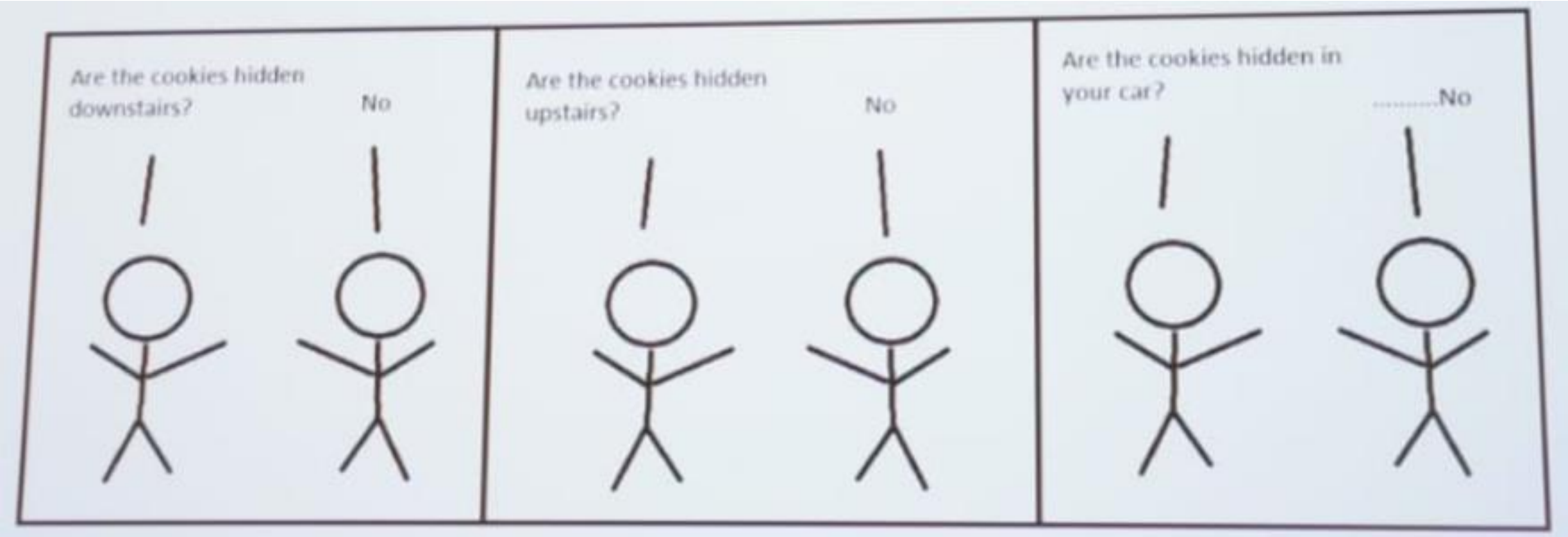
Phone / Money Theft

Impersonation

Side-Channel Analysis


copyright www.kovanleeuwen.nl

Cookies Example



Passive vs Active Side Channels

Passive: analyze device behavior

Active: change device behavior

Recent Practical Attacks

November 13, 2019

May 28, 2020


LadderLeak: Side-channel security flaws exploited to break ECDSA cryptography

Existing attacks refined to attack elliptic curve algorithm

SCA Titan: January 7, 2021

October 3, 2019

Researchers Discover ECDSA Key Recovery Method

October 3, 2019 • Add Comment • by Emma Davis

December 12, 2019


PLUNDERVAULT —

Intel's SGX coughs up crypto keys when scientists tweak CPU voltage

Install fixes when they become available. Until then, don't sweat it.

DAN GOODIN - 12/10/2019, 11:41 PM

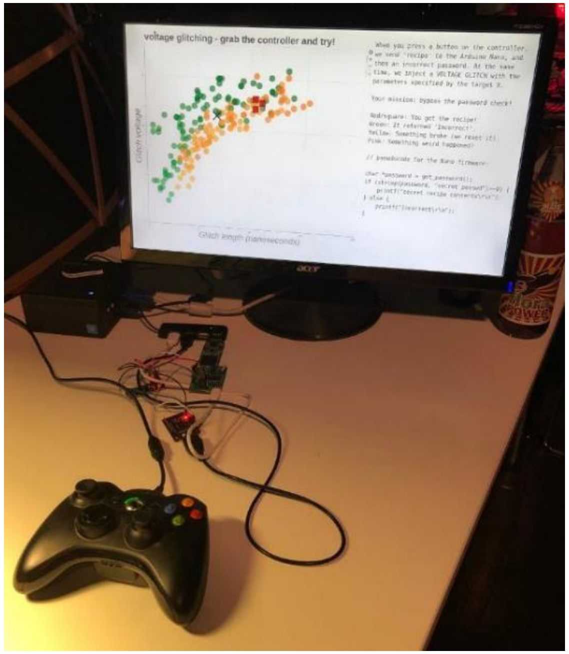
Side Channels

- Time
- Power
- Electro Magnetic Emanations
- Light
- Sound
- Temperature
- …

What can be attacked & why?

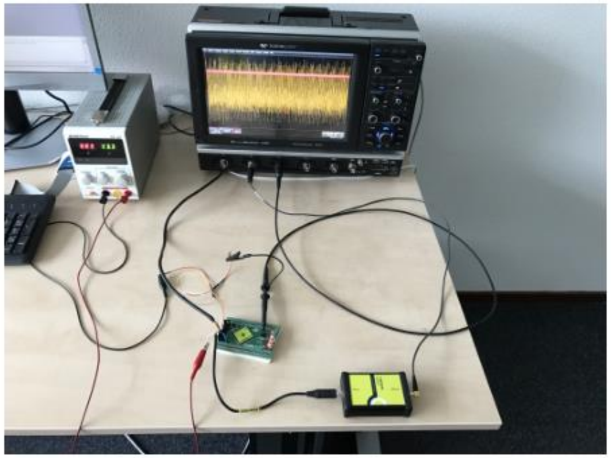
- Type of device?
- What kind of primitive?
- How much control do you have?
- What can you access?
- What would be the attacker's goal?
- What is your goal?
- Where is the money?
- …

Practical Setup Spectrum

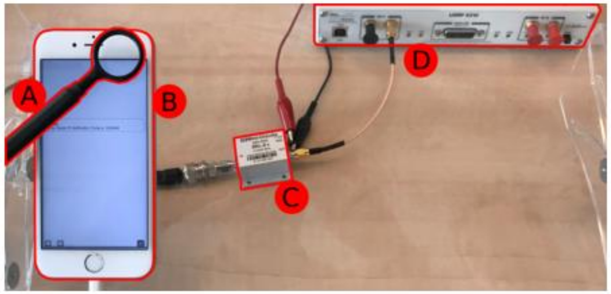


Some Other Practical Setups

DPA setup with ARM CortexM4

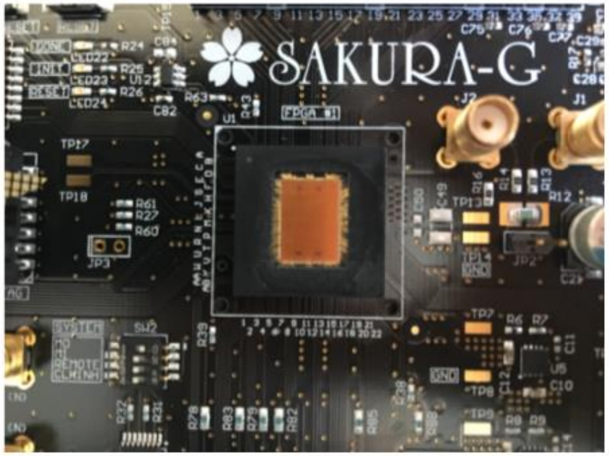


Tempest

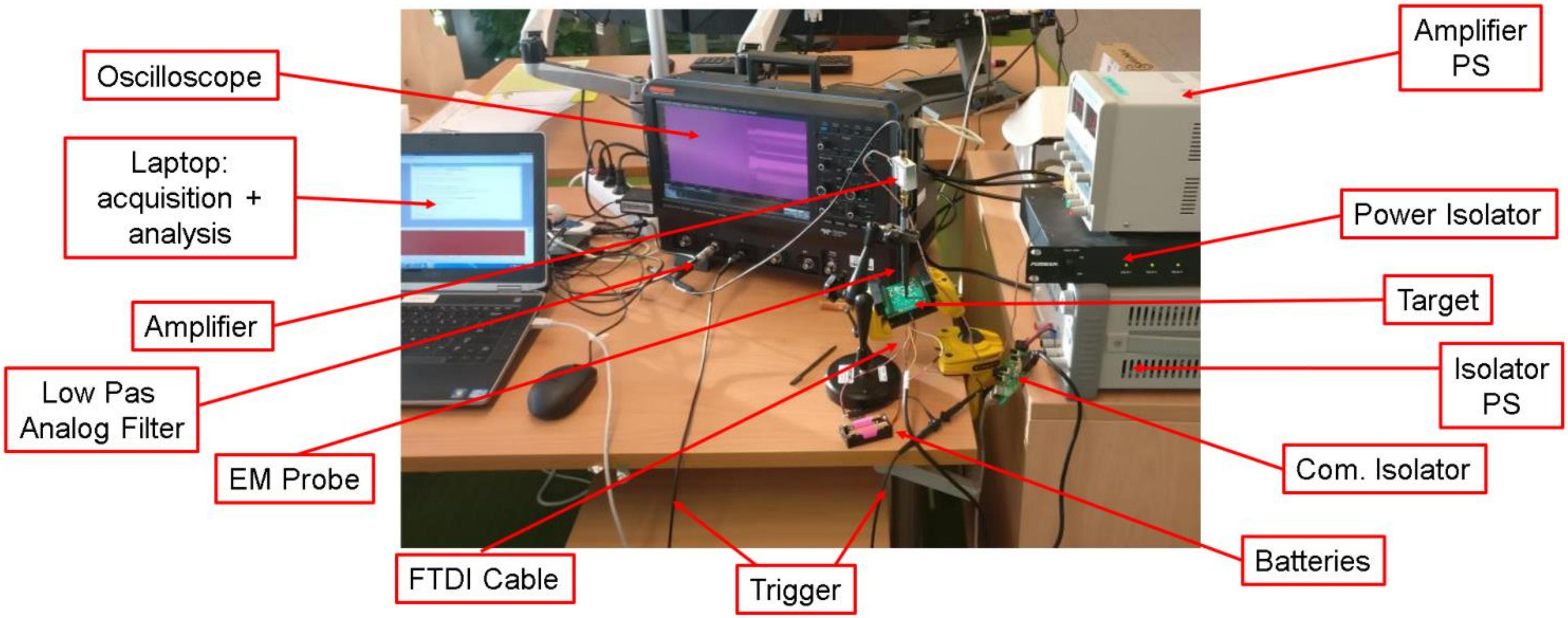


FA setup

FPGA board for SCA



Actual (overcomplicated?) setup



Simple Power Analysis (SPA) on RSA

```
ModExp(c) {
    A = 1
    for ( i = n-1; i≥0; i– –)
        A = A² mod N                 S
        if (d_i = =1)
            A = A*c mod N            M
        end if
    end for
    Return A = c^d mod N
}
```


Timing Attacks on Implementations of
Diffie-Hellman, RSA, DSS, and Other Systems

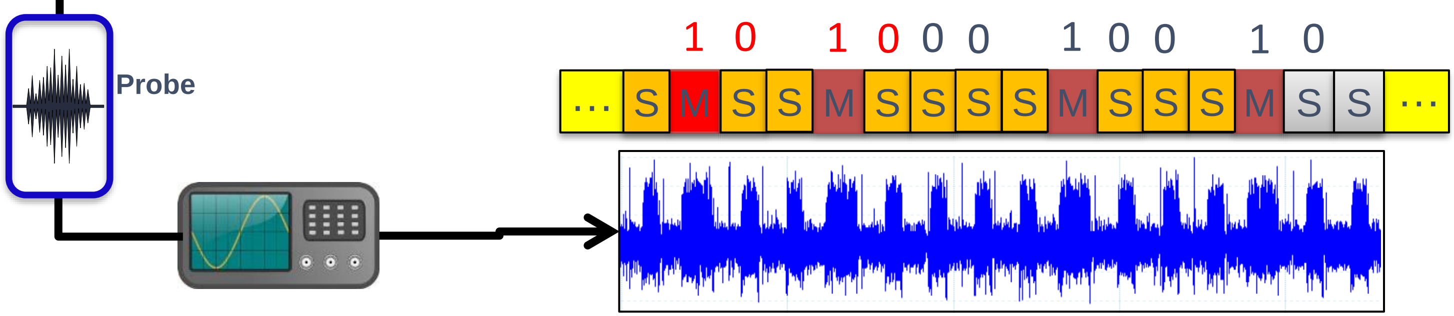
Paul C. Kocher

Cryptography Research, Inc.
607 Market Street, 5th Floor, San Francisco, CA 94105, USA.
E-mail: paul@cryptography.com.

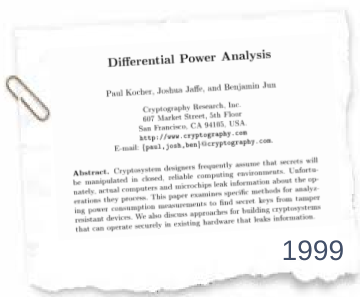
Abstract. By carefully measuring the amount of time required to perform private key operations, attackers may be able to find fixed Diffie-Hellman exponents, factor RSA keys, and break other cryptosystems. Against a vulnerable system, the attack is computationally inexpensive and often requires only known ciphertext. Actual systems are potentially at risk, including cryptographic tokens, network-based cryptosystems, and other applications where attackers can make reasonably accurate timing measurements. Techniques for preventing the attack for RSA and

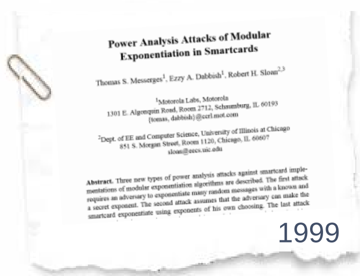
1996.

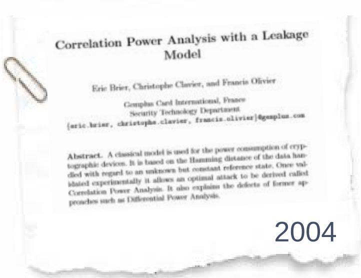
"By carefully measuring the ***amount of time*** required to perform private key operations, attackers may be able to find […] RSA keys."

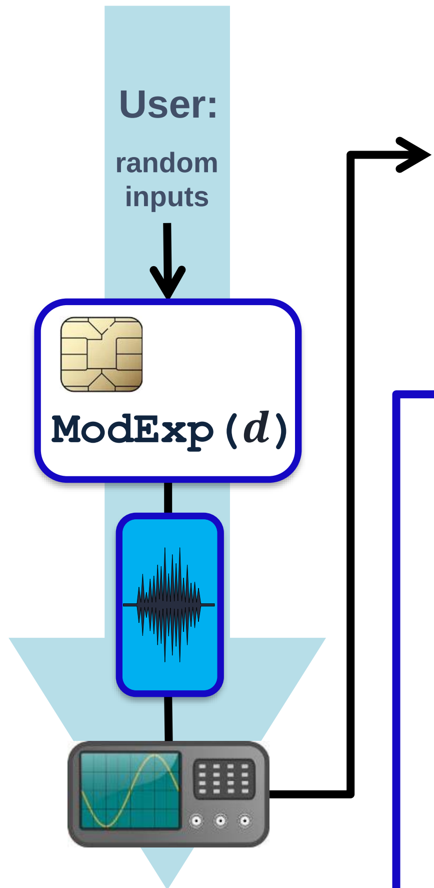


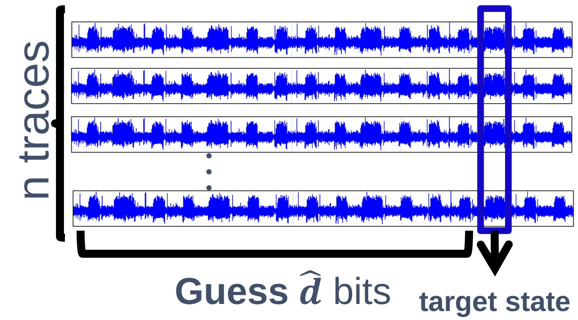
Differential (Correlation) Power Analysis











f_i = **Selection Function**(random inputs, $\hat{d}$, target state)

- HW of a register
- HD between current and previous register state
- ID model (value of a register)

$$f_i = \begin{cases} 0 & if\ HW \leq 16 \\ 1 & if\ HW > 16 \end{cases} \qquad f_i = HW(reg\_state)$$

DPA = Difference of Means

CPA = Pearson correlation

Profiled Attacks

- Problems with the above approaches:
  - can we attack the key directly?
  - we often do not get many traces with the same secret
  - can we use an unprotected device of the same model?
- (Possible) Solution:
  - We profile, i.e. template the unprotected device
  - We use the profile to break the protected device
- Procedure:
  1. Choose a model that describes the power consumption
  2. Profile the unprotected device to create the template (Template Building)
  3. Use the template to break the protected device (Template Matching)
- The same steps are always performed but the model can be different.
  - So often we will not learn the secret but the hamming weight of the secret.
- Neural Networks can be used instead of Template Attacks
- Attacking Single Trace or Multiple Traces? Both are possible.

Controlled unprotected device

Attacked protected device

Goals of Fault Injection

- The goal is to change a critical value or to change the flow of a program.
- Faults can be injected in several ways:
  - Power glitches can disturb the power supply to the processor, resulting in wrong values read from memory.
  - Optical glitches with laser can force any elementary circuit to switch, enabling the attacker to achieve a very specific change of data values or behavior.
  - Clock manipulation by introducing a few very short clock cycles which may lead to the device misinterpreting a value read from memory.
  - Cutting the power to the processor while performing important computations, hoping to either prevent the system from taking measures against a detected attack or get the system into a vulnerable state when the power is back.
- Differential Fault Analysis (DFA)

Fault Injection Example:
the “unlooper” device

Fault Injection Example: the "unlooper" device

Warm-up question (1): where to glitch?

```
void entry() {
    void* start  = 0x80000000;
    void* length = 0x00400000;

    serial_puts("Start Secure Boot...\n");

    loadOSFromHardDrive(start);

    if (! authenticateOS(start,length) )
        do {} while(1);

    serial_puts("Run OS\n");

    boot_next_stage(start);
    //starts executing at the address start
}
```

RSA-CRT: Differential Fault Analysis

- Optimization of computing a signature giving about 3 or 4-fold speed-up
- Precompute the following values:
  - Find $d_p = d \pmod{p-1}$, computed as $d_p = e^{-1} \pmod{p-1}$
  - Find $d_q = d \pmod{q-1}$
  - Compute $i_q = q^{-1} \pmod{p}$
- Computations using $m_p = m \pmod{p}$ and $m_q = m \pmod{q}$
- Signature or encryption (forgetting about hashing):
  - $s_p = m^{d_p} \pmod{p}$
  - $s_q = m^{d_q} \pmod{q}$
  - Garner's method (1965) to recombine s_p and s_q:
    - $s = s_q + q \cdot (i_q(s_p - s_q) \pmod{p})$
- Due to a limited time, we need to skip the math details on how to recover p and q, but it is possible with one fault!
  - If you are interested, ask me after the seminar; it is a so-called Bellcore attack, see for example: https://eprint.iacr.org/2012/553.pdf

How to protect against FI?

- You have to check that the operations was correctly executed, for example:
  - Duplication of operations;
  - For signature generation you can verify the result
  - Some SCA countermeasures will work even for FI
    - But not all

Warm-up Question (2):
Software for PIN code verification

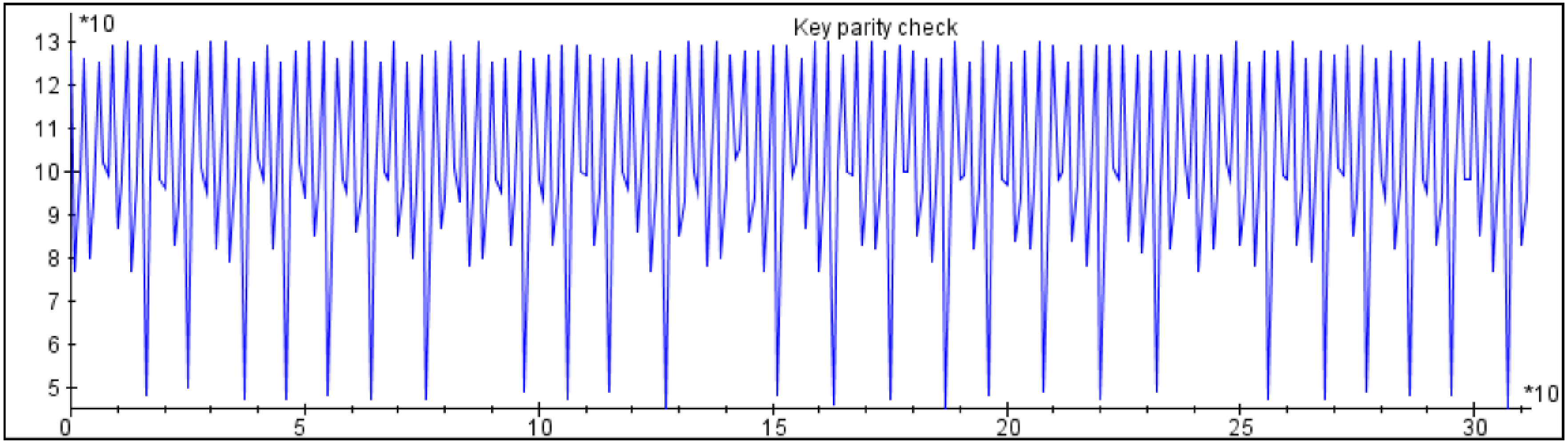
```
Input: 4-digit PIN code
Output: PIN verified or rejected
Process CheckPIN (pin[4])
int pin_ok=0;
if (pin[0]==5)
    if (pin[1]==9)
        if (pin[2]==0)
            if (pin[3]==2)
                pin_ok=1;
            end
        end
    end
end
return pin_ok;
EndProcess
```

- What is the problem here?
- What are the execution times of the process for PIN inputs?
    - [0,1,2,3], [5,3,0,2], [5,9,0,0]
- The execution time increases as we get closer to
    - [5,9,0,2]

Warm-up Task – parity check for DES key

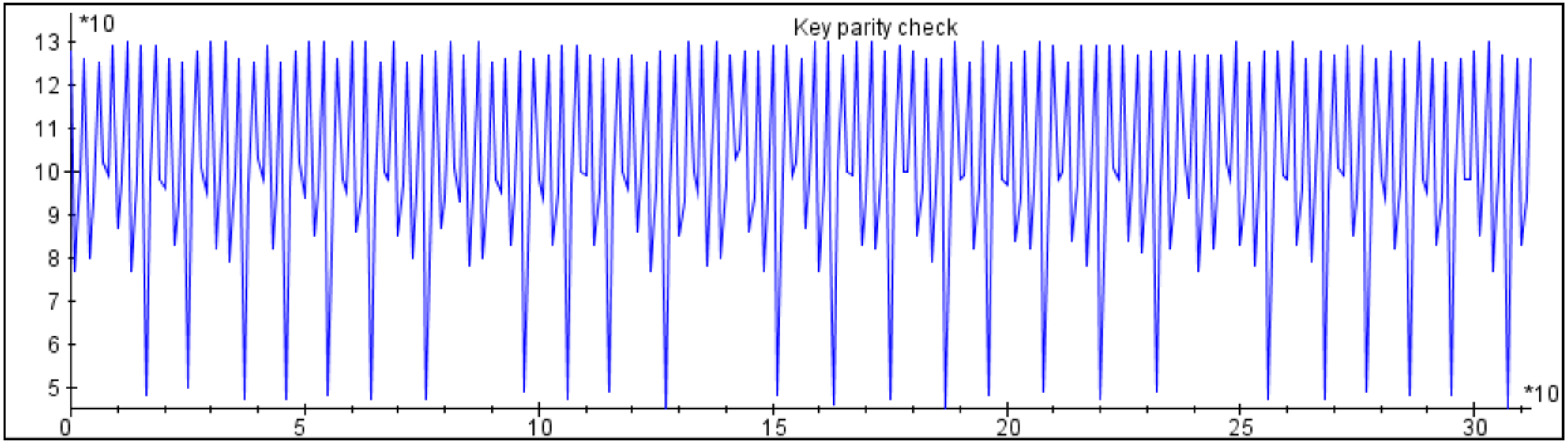
```
public static boolean checkParity ( byte[]key, int offset) {
    for (int i = 0; i < DES_KEY_LEN; i++) { // for all key bytes
        byte keyByte = key[i + offset];
        int count = 0;
        while (keyByte != 0) { // loop till no '1' bits left
            if ((keyByte & 0x01) != 0) {
                count++; // increment for every '1' bit
            }
            keyByte >>>= 1; // shift right
        }
        if ((count & 1) == 0) { // not odd
            return false; // parity not adjusted
        }
    }
    return true; // all bytes were odd
}
```

Warm-up Task – parity check for DES key
cont'd



Tell me what is the key ☺

Warm-up Task – parity check for DES key
cont'd



1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1

0x45 0x08 0xEF 0x23

Question 1:
faster and more secure modexp - Montgomery ladder

```
x0=x; x1=x^2
for j=k-2 to 0 {
  if dj=0
    x1=x0*x1; x0=x0^2
  else
    x0=x0*x1; x1=x1^2
  x1=x1 mod N
  x0=x0 mod N
}
return x0
```

Both branches with the same number and type of operations (unlike square and multiply on previous slide)

Is it constant-time & secure? Why?

Question 2:
even more secure modexp

x_0=x; x_1=x^2
for j=k-2 to 0 **{**
  b=d_j
  $x_{(1-b)}$=x_0*x_1; x_b=x_b^2
  x_1=x_1 mod N
  x_0=x_0 mod N
}
return x_0

Memory access often is not constant time!
Especially in the presence of caches.

Is it constant-time & secure? Why?

Question 3:
even more secure modexp

```
x_0=x; x_1=x^2
for j=k-2 to 0 {
  b=d_j
  x_(1-b)=x_0*x_1; x_b=x_b^2
  x_1=x_1 mod N
  x_0=x_0 mod N
}
return x_0
```

Memory access often is not constant time!
Especially in the presence of caches.

Is it constant-time & secure? Why?

Question 4:
even more more secure modexp

```
x0=x; x1=x^2; sw = 0
for j=k-2 to 0 {
  b=dj
  cswap(x0,x1,b⊕sw)
  sw = sw⊕di
  x1=x0*x1; x0=x0^2
  x1=x1 mod N
  x0=x0 mod N
}
return x0
```

Constant-time? Depends on the cswap… but it can be ☺
Other-side channels? Depends 😐

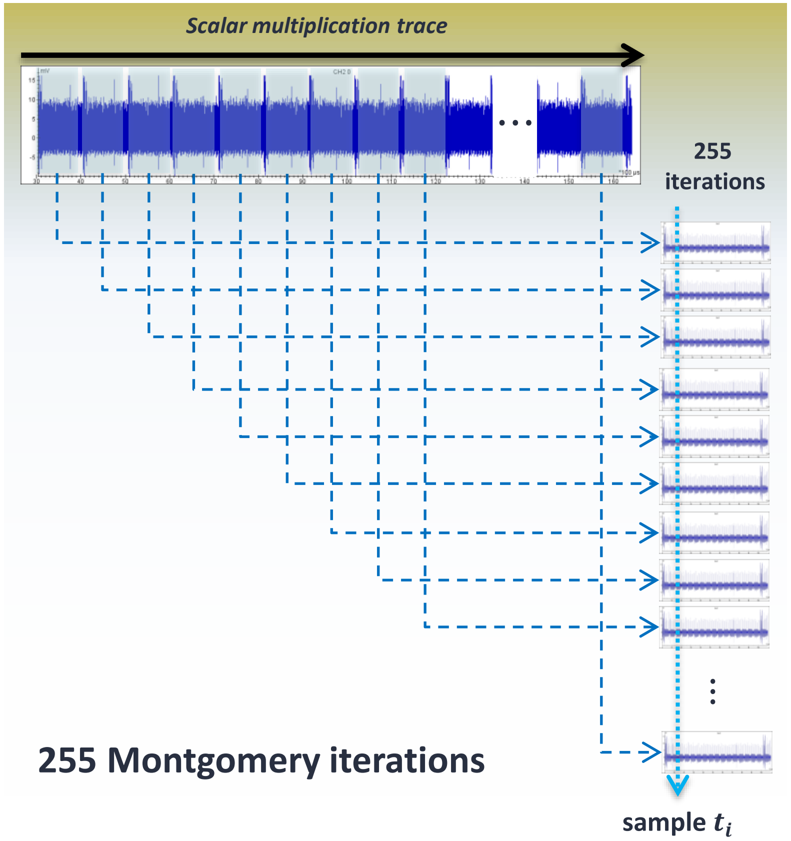
Is it constant-time & secure? Why?

Question 5:

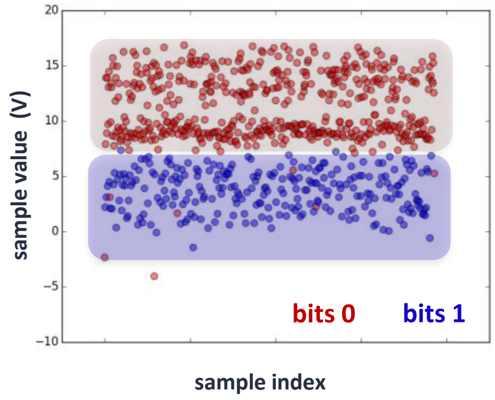
Arithmetic Cswap – constant-time?

```
void fe25519_cswap(fe25519* in1, fe25519* in2, int condition)
{
    int32 mask = condition;
    uint32 ctr;
    mask = -mask;
    for (ctr = 0; ctr < 8; ctr++)
    {
        uint32 val1 = in1->as_uint32[ctr];
        uint32 val2 = in2->as_uint32[ctr];
        uint32 temp = val1;
        val1 ^= mask & (val2 ^ val1);
        val2 ^= mask & (val2 ^ temp);
        in1->as_uint32[ctr] = val1;
        in2->as_uint32[ctr] = val2;
    }
}
```

Question 5:
Arithmetic Cswap – secure against other side-channels?



Apply clustering (e.g. **k-means**), Template Attack, Deep Learning to the set of 255 samples:



255 Samples

Message and exponent blinding

$$c = m^d mod\ N$$

1. $m_r = m.r^{-e} mod\ N$	message blinding
2. $d_r = d + r * \varphi(n)$	exponent blinding
3. $c_r = {m_r}^{d_r}\ mod\ n$	blinded exponentiation
4. $c = c_r * r\ mod\ n$	message "unblinding"

The sequence of operations (S, M) is related to the exponent bits.

However:

- If ***d*** is random: the sequence of exponent bits changes for every RSA execution
- If ***m*** is random: Intermediate data is random (masked) → hardly predicted!

DPA is based on the prediction of intermediate data.

Thesis: *Any side-channel attack requiring **multiple traces** are repelled by message **and** exponent blinding countermeasures.*

For ECC there are corresponding countermeasures: coordinate blinding, scalar blinding, blinded scalar multiplications, and no unblinding ☺

Message and exponent blinding for CRT?

$c = m^d mod\ N$

1. $m_r = m.r^{-e} mod\ N$	message blinding
2. $d_r = d + r * \varphi(n)$	exponent blinding
3. $c_r = {m_r}^{d_r}\ mod\ n$	blinded exponentiation
4. $c = c_r * r\ mod\ n$	message "unblinding"

- Message blinding is the same!
- Exponent blinding needs to be done twice:

  $s_p = m^{d_p}$ (mod p) = $m^{d_p + r*(p-1)}$ (mod p)

  $s_q = m^{d_q}$ (mod q) = $m^{d_q + r*(q-1)}$ (mod q)

- That does not stop FI attacks!

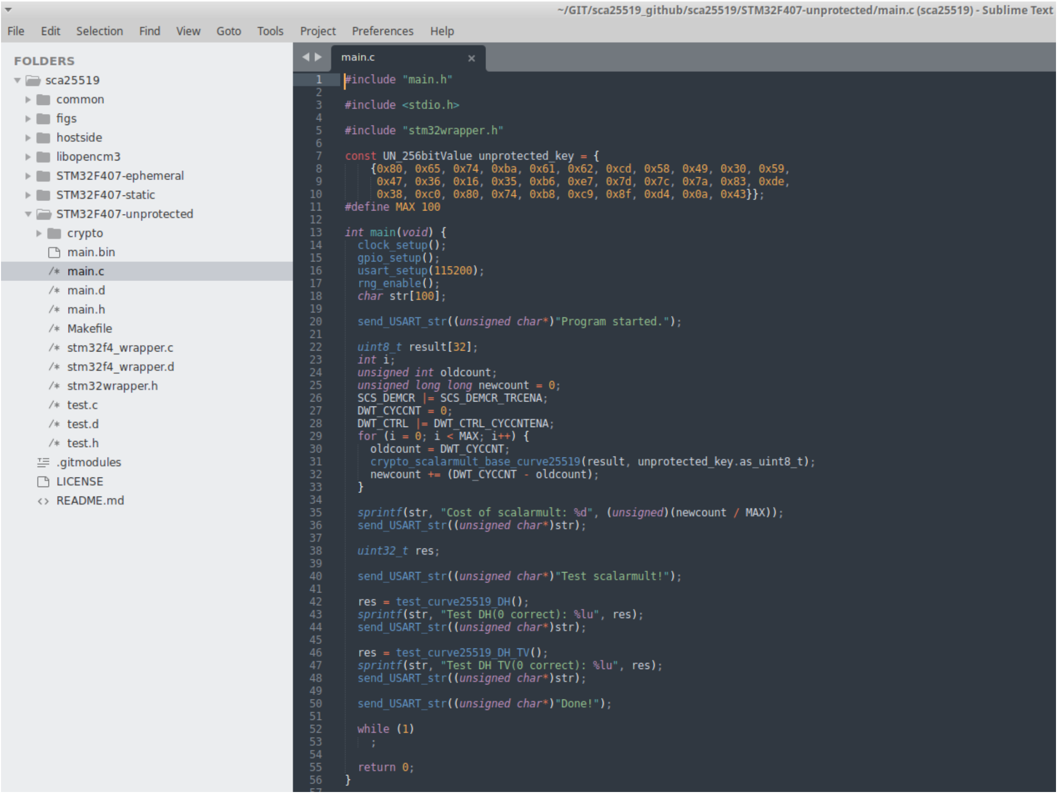
SCA&FI-protected Elliptic Curve library

- A protected library for ECDH
  - key exchange & session key establishment
  - It will be published in TCHES2023 volume 1 and
    - presented at Ches 2023 in Prague
- Code library available from GitHub
- Useful links:
  - https://eprint.iacr.org/2021/1003
  - https://github.com/sca-secure-library-sca25519/sca25519
- Taking care of ECDSA:
  - https://eprint.iacr.org/2022/1254
  - I will add it to the repository later on.

What to do first

- Download (or clone) the code from:
  - https://github.com/sca-secure-library-sca25519/sca25519
- If you do not know C then it will be tricky but in this case try to be intuitive.
- Task 1: have a look at the STM32F407-unprotected:
  - Please find the starting point.
  - Please find the scalar multiplication function.
    - And the scalar multiplication loop.
  - What the code is doing?

Task 1: Unprotected Crypto Library



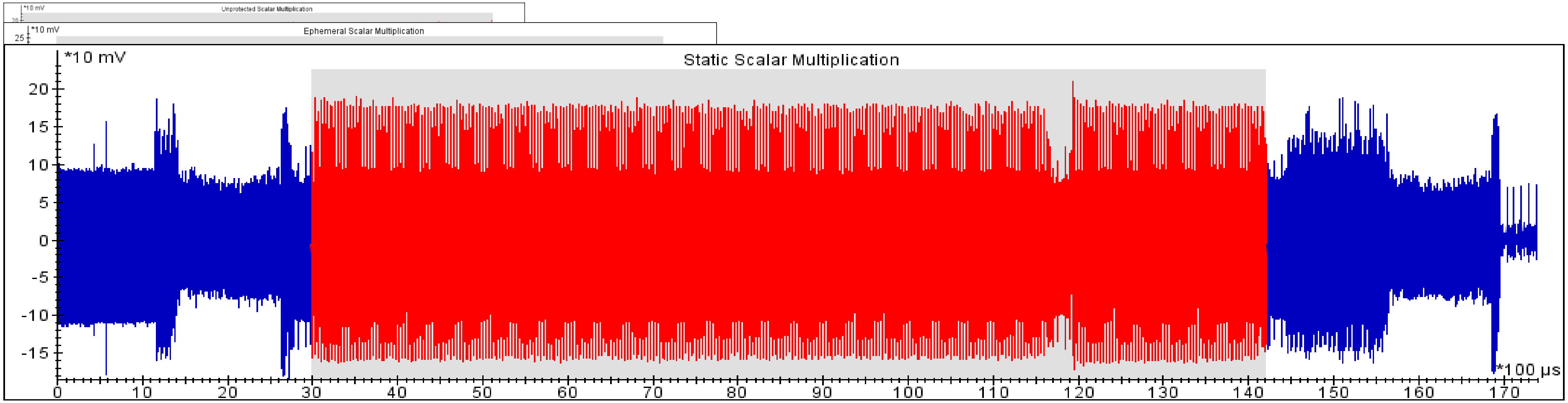
Task 1: Unprotected Crypto Library cont'd

```
int crypto_scalarmult_curve25519(
```

```

  state.previousProcessedBit = 0;

  // Process all the bits except for the last three where we explicitly double
  // the result.
  while (state.nextScalarBitToProcess >= 0) {
    uint8_t byteNo = (uint8_t)(state.nextScalarBitToProcess >> 3);
    uint8_t bitNo = (uint8_t)(state.nextScalarBitToProcess & 7);
    uint8_t bit;
    uint8_t swap;

    bit = 1 & (state.s.as_uint8_t[byteNo] >> bitNo);
    swap = bit ^ state.previousProcessedBit;
    state.previousProcessedBit = bit;
    curve25519_cswap(&state, swap);
    curve25519_ladderstep(&state);
    state.nextScalarBitToProcess--;
  }
```

Protected Crypto Library – other implementations

Ephemeral & Static increase complexity



Task 2: Ephemeral Crypto Library

- Have a look at the STM32F407-ephermeral (and STM32F407-static):
  - Find scalar multiplication functions and the scalar multiplication loops
- Try to find one side-channel countermeasure and one fault injection countermeasure. Have also a look at the list of implemented countermeasures in:
  - https://tches.iacr.org/index.php/TCHES/issue/view/312
- Can you explain the countermeasures?
- If you have time, then try to find one or two more countermeasures

Remark: do not worry – this is a hard exercise.

Task 2: Ephemeral Crypto Library - FI

```c
// ### alg. step 5 ###
INCREMENT_BY_163(fid_counter);

// Double 3 times before we start. ### alg. step 6 ###
curve25519_doublePointP(&state);
curve25519_doublePointP(&state);
curve25519_doublePointP(&state);

// ### alg. step 7 ###
INCREMENT_BY_163(fid_counter);

if (!fe25519_iszero(&state.zp))   // ### alg. step 8 ###
{
  goto fail; // ### alg. step 9 ###
}

// Optimize for stack usage when implementing  ### alg. step 10 ###
fe25519_invert_useProvidedScratchBuffers(&state.zp, &state.zp, &state.xq,
                                         &state.zq, &state.x0);
fe25519_mul(&state.xp, &state.xp, &state.zp);
fe25519_reduceCompletely(&state.xp);

fe25519_cpy(&state.x0, &state.xp);
```

```c
  fe25519_reduceCompletely(&state.xp);

  INCREMENT_BY_163(fid_counter); // ### alg. step 21 ###

  // ### alg. step 22 ###
  if (fid_counter != (163 * 4 + 251 * 9)) {
  fail:
    retval = -1;
    randombytes(state.xp.as_uint8_t, 32);  // ### alg. step 23 ###
  } else {
    retval = 0;
  }
  fe25519_pack(r, &state.xp);
  return retval;
}
```

Find the same countermeasure in the static implementation.

Task 2: Ephemeral Crypto Library - SCA

```c
static void maskScalarBitsWithRandomAndCswap(
        ST_curve25519ladderstepWorkingState *pState, uint32_t wordWithConditionBit,
        uint32_t bitNumber) {
    uint32_t randomDataBuffer[2] = {0, 0};
    randombytes((uint8_t *)randomDataBuffer, sizeof(randomDataBuffer));
    //
    // first combine the scalar bit with a random value which has
    // the bit at the data position cleared
    uint32_t mask = randomDataBuffer[0] & (~(1 << bitNumber));
    wordWithConditionBit ^= mask;

    // Arrange for having the condition bit at bit #0 and random data elsewhere.
    ROTATER(wordWithConditionBit, bitNumber);

    cSwapAndRandomize(wordWithConditionBit, pState->xp.as_uint32_t,
                      pState->xq.as_uint32_t, randomDataBuffer[1]);
    cSwapAndRandomize(wordWithConditionBit, pState->zp.as_uint32_t,
                      pState->zq.as_uint32_t, randomDataBuffer[1]);
}
```

Task 2: Ephemeral Crypto Library – SCA cont'd

```c
static void maskScalarBitsWithRandomAndCswap(
        ST_curve25519ladderstepWorkingState *pState, uint32_t wordWithConditionBit,
        uint32_t bitNumber) {
    uint32_t randomDataBuffer[2] = {0, 0};
    randombytes((uint8_t *)randomDataBuffer, sizeof(randomDataBuffer));
    //
    // first combine the scalar bit with a random value which has
    // the bit at the data position cleared
    uint32_t mask = randomDataBuffer[0] & (~(1 << bitNumber));
    wordWithConditionBit ^= mask;

    // Arrange for having the condition bit at bit #0 and random data elsewhere.
    ROTATER(wordWithConditionBit, bitNumber);

    cSwapAndRandomize(wordWithConditionBit, pState->xp.as_uint32_t,
                      pState->xq.as_uint32_t, randomDataBuffer[1]);
    cSwapAndRandomize(wordWithConditionBit, pState->zp.as_uint32_t,
                      pState->zq.as_uint32_t, randomDataBuffer[1]);
}
```

Task 3: Static Crypto Library – SCA

- Find scalar splitting (similar to blinding):
  1. Generate 64-bit r and computer r^{-1}
  2. Compute P’ = $[r^{-1}*k]*P$
  3. Compute [r]*P’ = [k]P
- Does it work?
- Find this countermeasure in the static SCA code: Steps 2 and 3.

Exercise: Protected Crypto Library 3

Step 2

```
    // ### alg. step 22 ###
    while (state.nextScalarBitToProcess >= 0) {
      uint8_t limbNo = 0;
      uint8_t bitNo = 0;
  #ifdef MULTIPLICATIVE_CSWAP
      {
        limbNo = (uint8_t)(state.nextScalarBitToProcess >> 5);
        bitNo = state.nextScalarBitToProcess & 0x1f;
        // ### alg. step 22 and ###

        maskScalarBitsWithRandomAndCswap(&state, state.s.as_uint32_t[limbNo],
                                          bitNo);
      }
  #else
      {
        limbNo = (uint8_t)(state.nextScalarBitToProcess >> 5);
  #ifdef ITOH_COUNTERMEASURE
        uint32_t temp = state.s.as_uint32_t[limbNo] ^ itoh.as_uint32_t[limbNo];
        curve25519_cswap_asm(&state, &temp);
        state.s.as_uint32_t[limbNo] <<= 1;
        itoh.as_uint32_t[limbNo] <<= 1;
  #else
        curve25519_cswap_asm(&state, &state.s.as_uint32_t[limbNo]);
  #endif
      }
  #endif
      if (state.nextScalarBitToProcess >= 1)  // ### alg. step 24
      {
        curve25519_ladderstep(&state); // alg. step 25

        INCREMENT_BY_NINE(fid_counter); // alg. step 27

  #ifdef MULTIPLICATIVE_CSWAP
  #ifdef ITOH_COUNTERMEASURE
        maskScalarBitsWithRandomAndCswap(&state, itohShift.as_uint32_t[limbNo],
                                          bitNo); // ### alg. step 26
  #endif
  #else
  #ifdef ITOH_COUNTERMEASURE
        curve25519_cswap_asm(&state, &itohShift.as_uint32_t[limbNo]);
  #endif
  #endif
      }
```

Step 3

```
    while (state.nextScalarBitToProcess >= 0) // ### alg. step 37
    {
      uint8_t limbNo = 0;
      uint8_t bitNo = 0;

  #ifdef MULTIPLICATIVE_CSWAP
      {
        limbNo = (uint8_t)(state.nextScalarBitToProcess >> 5);
        bitNo = state.nextScalarBitToProcess & 0x1f;

        // ### alg. step 38
        maskScalarBitsWithRandomAndCswap(&state, state.r.as_uint32_t[limbNo],
                                          bitNo);
      }
  #else
      {
        limbNo = (uint8_t)(state.nextScalarBitToProcess >> 5);
  #ifdef ITOH_COUNTERMEASURE64
        uint32_t temp = state.r.as_uint32_t[limbNo] ^ itoh64.as_uint32_t[limbNo];
        curve25519_cswap_asm(&state, &temp);
        state.r.as_uint32_t[limbNo] <<= 1;
        itoh64.as_uint32_t[limbNo] <<= 1;
  #else
        curve25519_cswap_asm(&state, &state.r.as_uint32_t[limbNo]);
  #endif
      }
  #endif

      if (state.nextScalarBitToProcess >= 1) // ### alg. step 39
      {
        curve25519_ladderstep(&state); // ### alg. step 40
        INCREMENT_BY_NINE(fid_counter); // ### alg. step 42

  #ifdef MULTIPLICATIVE_CSWAP
  #ifdef ITOH_COUNTERMEASURE64
        maskScalarBitsWithRandomAndCswap(&state, itoh64Shift.as_uint32_t[limbNo],
                                          bitNo); // ### alg. step 41
  #endif
  #else
  #ifdef ITOH_COUNTERMEASURE64
        curve25519_cswap_asm(&state, &itoh64Shift.as_uint32_t[limbNo]);
  #endif
  #endif
      }
      state.nextScalarBitToProcess--;
```

Efficiency Demo (Optionally)

Demo Instructions

- Open in a browser: https://github.com/sca-secure-library-sca25519/sca25519
- And follow the instructions from there
  - There are some issues related to the libopencm3 library
- You need a Discover board and an FTDI cable
- git clone https://github.com/sca-secure-library-sca25519/sca25519.git

Assignment 9 – Countermeasures

- This is a programming assignment. Please upload your scripts/code and the required analysis via the course webpage.
- The deadline for submission is Dec. 13, 2023, 8:00.
  - -3 points for each started 24h after the deadline.
- Your code should be contained in one .py file. Please name the submission file as <uco_number>_hw9.zip. Put there both the python code, the analysis document, and all data produced during analysis (as long as the size is reasonable).
- The code must contain comments so that it is reasonably easy to understand how to run the script for evaluating each answer.

Assignment 9 - Tasks

1. Task 1: protect the CRT implementation with exponent blinding in the function TCR_protected! First, test and then modify the code (the result should be the same). In a separate report (max 2 pages), write why the countermeasure works (does not affect the correctness of the result). Then, perform a useful analysis of the efficiency cost of the countermeasure (repeat the experiment a number of times and report a percent increase). **[3.5 points]**
2. Task 2: protect the CRT implementation with message blinding! Note that this will require knowledge of e. In the document, write why the countermeasure works. Then, perform a useful analysis of the cost of the countermeasure. **[3.0 points]**
3. Task 3: protect the CRT implementation against fault injection! Any countermeasure is OK. In the document, write why the countermeasure works. Then, perform a useful analysis of the cost of the countermeasure. **[2.5 points]**
4. Task 4: combine all the countermeasures and measure the time of all additional countermeasures and how well they work. Write that in the report. **[1 points]**
5. **Bonus (3 points):**
   (a) Instead of exponent blinding, implement exponent splitting. How does it compare to blinding? **[1 point]**
   (b) Implement another extra countermeasure (any, it can be either SCA or FI). What is its cost? **[1 point]**
   (c) Implement yet another extra countermeasure (any, either SCA or FI). What is its cost? **[1 point]**

Remark: we are securing Python code and, for the sake of this exercise, assume that the code is directly executed by the processor (and not interpreted etc.)

Consultation: Monday at 13:00 in A406.

Good luck!!!