Universitat de Girona
Escola Politècnica Superior

Treball Final de Màster

Estudi: Màster en Ciència de Dades

Títol: Side channel attack against the Mbed TLS implementation of the RSA algorithm.

Document: Memòria

Alumne: Victor Micó Biosca

Tutor: David Juher Barrot
Departament: Departament d'Informàtica, Matemàtica Aplicada i Estadística
Àrea: Matemàtica Aplicada

Convocatòria (mes/any): Juny 2023

Universitat de Girona
Escola Politècnica Superior

TREBALL FINAL DE MÀSTER

Side channel attack against the Mbed TLS implementation of the RSA algorithm.

Autor:
Victor MICÓ BIOSCA

Juny 2023

Màster en Ciència de Dades

Tutor:
David JUHER BARROT

Resum

L'operació principal de l'RSA és l'exponenciació modular. Depenent de la implementació és possible realitzar atacs de canal lateral com DPA. Per protegir aquesta operació, els desenvolupadors de llibreries criptogràfiques implementen contramesures com l'ofuscament de l'exponent, que impedeix utilitzar múltiples traces. Com a conseqüència, els atacs s'han de realitzar utilitzant una única traça. En aquest treball ataquem la implementació d'RSA de la llibreria Mbed TLS la qual utilitza un algoritme d'exponenciació de finestra mòbil. Per mitjà d'SPA i de tècniques de correspondència de patrons s'ha aconseguit extreure els dos exponents privats de la clau RSA, d_p i d_q utilitzant una única traça de potència.

El codi i la traça utilitzats en aquest treball es poden trobar a la següent adreça de Github. victormico/sca-mbedtls-rsa

Agraïments

Per començar, vull expressar el meu sincer agraïment a Colin O'Flynn i al personal de *NewAE Technology Inc.* per haver seleccionat i donat suport a la proposta de treball que ha culminat en aquesta tesi.

A més, desitjo agrair de manera especial al meu tutor, David Juher, per la seva inestimable ajuda i col·laboració.

També m'agradaria expressar la meva profunda gratitud a la meva parella, Marina, a la meva mare, Pura, i a la meva germana, Maria, pel constant suport i paciència que m'han brindat al llarg d'aquest procés.

Així mateix, vull agrair de manera especial al meu amic Alberto Marcos per les enriquidores converses sobre criptografia i ciència de dades, les quals han estat una autèntica font d'inspiració per a aquest treball.

Finalment, vull dedicar aquest treball a la memòria del meu pare, que estic segur que l'hauria gaudit llegint.

Sense el suport incondicional de tots ells, aquest treball no hauria estat possible.

Índex

Índex de figures

Índex de taules

List of Algorithms

CAPÍTOL 1

Introducció

La criptografia respon a la necessitat d'enviar informació sensible a una altra persona sense que terceres persones en puguin conèixer el contingut. La història de la criptografia es remunta al 4000 AC a Egipte, on els jeroglífics servien per codificar els missatges que es volien transmetre. Durant l'època de l'emperador romà Juli Cèsar, s'utilitzava el que avui es coneix com a xifra de Cèsar. La xifra de Cèsar consisteix a substituir cada lletra per la corresponent lletra desplaçada un nombre de posicions determinat.

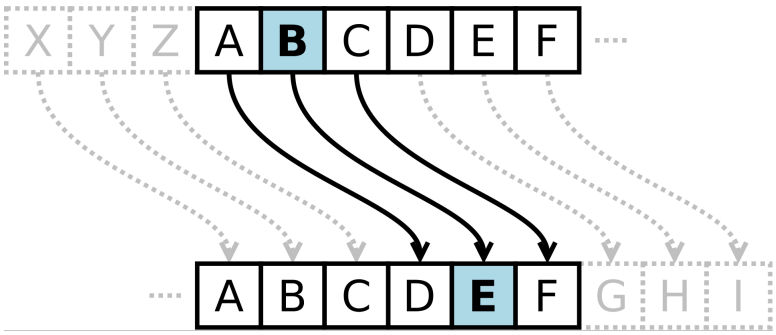

Figura 1.1: Substitució de caràcters de la xifra de Cèsar

Va ser necessari que transcorreguéssin 1500 anys fins que s'introduís una millora en aquest algoritme de xifra. [Aumasson 2017] La xifra de Vigenère suposa una modificació de l'algoritme del Cèsar on en comptes de substituir cada lletra utilitzant un nombre fix, s'utilitza un nombre de posicions que canvia per a cada lletra.

Aquest algoritme va ser utilitzat fins a La Primera Guerra Mundial. Després, amb l'inici de la computació i sobretot durant La Segona Guerra Mundial, es van desenvolupar múltiples mecanismes de xifrat. En particular, el bàndol aliat va interceptar les comunicacions del bàndol enemic en descobrir com funcionava la màquina Enigma. Això va suposar un punt clau per al desenvolupament posterior del conflicte.

La criptografia, per tant, era utilitzada per organitzacions, governs i nacions, amb finalitats principalment militars. En l'actualitat, amb l'ús generalitzat dels ordinadors i internet, la criptografia és una peça clau per assegurar la confidencialitat de la informació que tractem en el nostre dia a dia.

Actualment hi ha algorismes públics i estandarditzats com el DES (Data Enciption Standard) o el seu successor, l'AES (Advanced Encriptyon Standard).

Tant el DES com l'AES basen la seva seguretat en combinar el missatge inicial amb una clau i després realitzar permutacions i substitucions de bits.

En el cas del DES, la clau té una mida de 56 bits. Això permet que amb el hardware actual sigui possible una atac de força bruta. Aquest tipus d'atacs consisteixen a provar totes les possibles claus fins a obtenir el missatge correcte. Amb un màquina formada per un *array* de 120 FPGA anomenada COPACOBANA, amb un cost aproximat d'uns 9000$, és possible en 6,4 dies de mitjana. [van Tilborg 2014]

En al cas de l'AES, la seva longitud de 128 bits impossibilita un atac de força bruta amb la capacitat de computació actual.

Tots els algorismes mencionats són de clau simètrica, això implica que s'utilitza la mateixa clau per xifrar i desxifrar.

En el 1976 Diffie i Hellman introdueixen per primer cop el concepte de clau pública i privada, posant les bases per a la criptografia asimètrica. L'intercanvi de claus pública i privada basa la seguretat en el problema del logaritme discret. Tot i això, Diffie i Hellman no van proposar cap sistema de xifrat concret. Dos anys després, l'any 1978, Rivest, Shamir i Adleman van descobrir un sistema de xifrat de clau pública, el qual es va anomenar RSA, amb les inicials de cadascun d'ells. L'RSA es basa en un altre problema matemàtic, la dificultat de factoritzar nombres primers.

Amb l'extensió d'usos dels algoritmes de xifrat, ben aviat va ser necessari l'ús de dispositius per a emmagatzemar adequadament les claus de xifrat, perquè emmagatzemar les claus directament a l'ordinador o al mòbil pot suposar un problema de seguretat. Per això són necessaris dispositius externs com les *smartcards* o les carteres digitals. Alguns dels elements quotidians que fan servir dispositius criptogràfics incrustats (*embedded*) són les targetes de crèdit, les targetes SIM dels telèfons mòbils, el DNI o el passaport.

Els dispositius criptogràfics són capaços de rebre un missatge a través d'una interfície, xifrar el contingut del missatge i transmetre el missatge xifrat. Generalment, també són capaços de fer l'operació a la inversa: rebre un missatge xifrat, desxifrar-lo i transmetre el missatge en text pla.

La seguretat d'aquests dispositius no només es basa en la robustesa dels seus algoritmes sinó en la impossibilitat teòrica d'extraure'n les claus criptogràfiques que contenen. L'acció d'intentar obtenir les claus d'un dispositiu criptogràfic sense autorització s'anomena atac.

Els atacs a dispositius criptogràfics es poden dividir en dues grans categories:

- Atacs actius: Un atac actiu consisteix a manipular els *inputs* o l'entorn del dispositiu amb l'objectiu que funcioni de forma errònia o diferent de les

condicions normals. Amb la injecció de faltes (*fault injection*) és possible fer passar un PIN dolent per bo o extreure claus criptogràfiques, entre d'altres.

- Atacs passius: En un atac passiu, l'atacant extreu informació del dispositiu a través de canals laterals (*side-channel*) mentre que el dispositiu funciona en condicions normals. Aquests canals laterals poden ser el consum elèctric, la radiació electromagnètica o, fins i tot, el so o la temperatura.

Obtenir les claus dels dispositius mencionats anteriorment pot suposar aconseguir l'accés al compte bancari d'una persona, poder interceptar les seves comunicacions o suplantar-ne la identitat. És per això que aquests tipus d'atacs tenen un gran interès entre la comunitat acadèmica i contínuament se'n publiquen de nous i contramesures per a evitar-los.

L'objectiu d'aquest treball es realitzar un atac de canal lateral a una implementació de codi obert de l'algoritme RSA.

Tot i que a l'estat de l'art es mencionen moltes publicacions sobre com realitzar atacs de canal lateral, la majoria es centren en els algoritmes de clau simètrica. Entre les publicacions de clau asimètrica, en són poques les que publiquen de manera oberta les dades i el codi per reproduir l'atac.

Les motivacions del treball són les següents:

1. Enrobustir una llibreria de codi lliure.

2. Fer difusió dels mètodes de captura, processament i atac d'un algoritme de clau asimètrica.

3. Publicar de manera oberta l'anàlisi i les traces.

El resultat d'aquest treball és l'extracció completa de la clau de RSA per mitjà d'una única traça de consum de potència.

CAPÍTOL 2

Preliminars

2.1 Domini

2.1.1 Criptografia de clau pública

La criptografia de clau pública utilitza dues claus diferents per a xifrar i desxifrar dades: una clau pública i una clau privada. La clau pública es pot compartir obertament amb altres persones, mentre que la clau privada es manté en secret per l'usuari.

El procés de xifrat implica utilitzar la clau pública per transformar un missatge en un text xifrat que no pot ser llegit sense la clau privada corresponent. Aquest text xifrat es pot enviar de manera segura a través d'un canal públic. Per desxifrar el text xifrat i obtenir el missatge original, es fa servir la clau privada.

A més de la xifra, la criptografia de clau pública té altres aplicacions importants com ara la signatura digital i l'autenticació. Amb la signatura digital es pot generar una firma digital única d'un missatge utilitzant la clau privada, la qual cosa permet verificar-ne l'autenticitat i la integritat. Això és útil per verificar l'autoria d'un missatge i assegurar-se que no ha estat modificat durant la transmissió.

La criptografia de clau pública es basa en problemes matemàtics complexos que són difícils de resoldre, com ara la factorització de nombres grans (en el cas de l'algoritme RSA) o el problema del logaritme discret en corbes el·líptiques (en el cas de l'algoritme ECC). Aquests problemes proporcionen la seguretat del sistema, ja que requereixen un esforç computacional significatiu per a ser resolts.

2.1.1.1 RSA

Com hem introduït prèviament al capítol 1, l'RSA va ser dissenyat per Rivest, Shamir i Adleman i basa la seva seguretat en la dificultat de factoritzar nombres grans.

Primitives criptogràfiques

- **Xifrat**: $c = m^e \mod n$ on m és el missatge, e és la clau pública i c és el text xifrat (*ciphertext*).
- **Desxifrat**: $m = c^d \mod n$ on c és el text xifrat, d és la clau privada i m és el missatge.
- **Firma**: En el procés de signatura, l'autor del missatge utilitza la seva clau privada per generar una firma $s = m^d \mod n$.
- **Verificació de la firma**: La firma s d'un missatge m es verifica computant $m' = s^e \mod n$. Si $m = m'$, llavors la firma és valida.

Procés de generació de la clau RSA Les claus pública i privada del RSA es generen de la següent manera:

1. Es generen dos nombres primers, p i q grans, distints i amb una longitud en bits similar.
2. Es calcula el mòdul $n = p \cdot q$
3. Es calcula el totient de n, i.e. $\varphi(n) = (p-1) \cdot (q-1)$
4. Es tria un nombre enter positiu que siguicoprimer amb $\varphi(n)$ i que compleixi $1 < e < \varphi(n)$. El parell (n, e) serà la clau pública.
5. Es calcula l'exponent privat d realitzant una operació d'aritmètica modular anomenada inversa multiplicativa. Ha de satisfer que $d \cdot e \equiv 1 \mod \varphi(n)$. L'exponent d serà la clau privada.

2.1.1.2 Algoritmes d'exponenciació modular

D'entre totes les operacions necessàries per computar qualsevol de les primitives del RSA, l'exponenciació modular és la més fonamental. L'algoritme més bàsic per a computar m^e consisteix a multiplicar m per si mateix e vegades, i.e. $m \cdot m \cdot \ldots \cdot m$. Amb nombres molt grans com és el cas del RSA, realitzar l'exponenciació d'aquesta manera és molt ineficient, en concret es necessiten $e-1$ multiplicacions. Una clau de RSA típica és de 1024 bits. Si pensem que el nombre de multiplicacions seria de l'ordre de 2^{1024} i que el nombre estimat d'àtoms a l'univers és de l'ordre de 2^{300}, podem veure que no és possible computar l'exponenciació d'aquesta forma. [Menezes 2007].

Amb la intenció de reduir el nombre d'operacions hi ha diversos algoritmes per realitzar aquesta operació. Aquests algoritmes combinen multiplicacions amb quadrats per realitzar l'exponenciació. En el cas de l'algoritme 1 *left-to-right binary exponentiation*, també conegut com *square and multiply*, es tracta

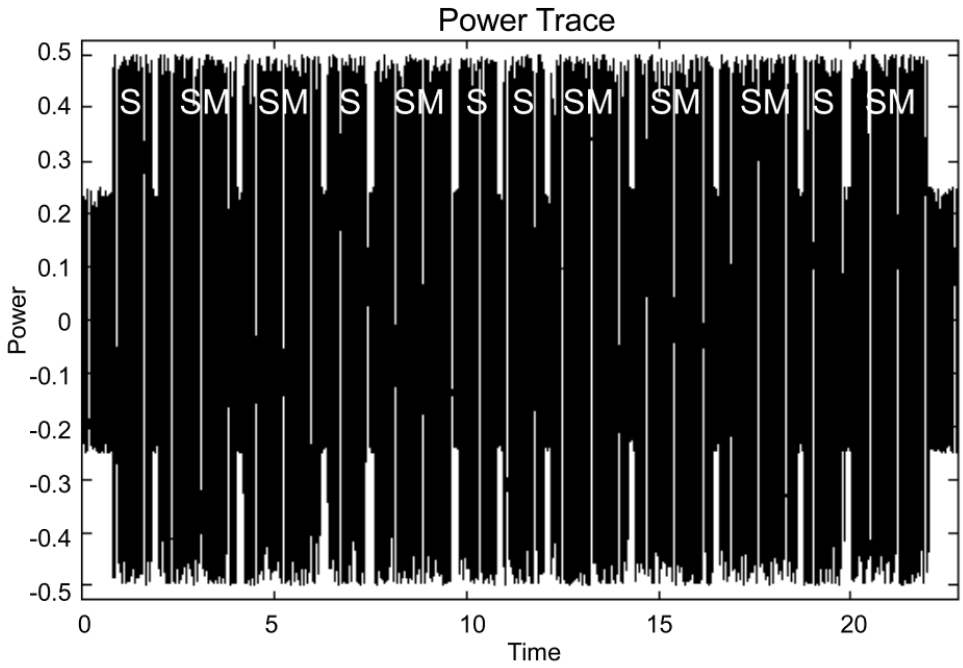

Figura 2.1: Traça de potència de RSA [Paar 2010]

l'exponent observant cada bit d'esquerra a dreta. Si el bit és '0' es fa un quadrat i si el bit és '1' es fa un quadrat i una multiplicació. Per tant, podem observar que si de mitjana tenim tants 1 com 0 a una clau aleatòria de RSA, el nombre necessari d'operacions per a una clau de 1024 bits seria de $1.5 \cdot 1024 = 1.536$, el còmput del qual ja està a l'abast de qualsevol dispositiu.

L'elecció de l'algoritme d'exponenciació no és només rellevant en termes d'eficiència, sinó que també és important per a la seguretat. Si podem distingir multiplicacions de quadrats, és possible extreure'n l'exponent directament. [Kocher 1999].

A la figura 2.1 podem un exemple d'una traça de potència on es pot identificar la seqüencia d'operacions modulars. Els quadrats es marquen amb una S (*square*) i les multiplicacions amb una M.

En el capítol 3 veurem amb detall els atacs de canal lateral i, com a través del consum de potència, podem observar aquestes diferències.

Una contramesura fàcil i eficient consisteix a realitzar una multiplicació addicional en cas que el bit sigui '0', però descartant-ne el resultat [Coron 1999]. En l'algoritme 2 podem veure aquesta modificació.

Algorithm 1 Left-to-right binary exponentiation

Require: m as message
Require: $(e = (e_t e_{t-1} \dots e_1 e_0)_2)$ for $e_i \in (0,1)$
Ensure: m^e
  $A \leftarrow 1$
  for $i \leftarrow t$ to 0 **do**
    $A \leftarrow A \cdot A$ {Square}
    if $e_i = 0$ **then**
      $A \leftarrow A \cdot m$ {Multiply}
  return A

Algorithm 2 Left-to-right multiply always binary exponentiation

Require: m as message
Require: $(e = (e_t e_{t-1} \dots e_1 e_0)_2)$ for $e_i \in (0,1)$
Ensure: m^e
  $A \leftarrow 1$
  for $i \leftarrow t$ to 0 **do**
    $A \leftarrow A \cdot A$ {Square}
    if $e_i = 0$ **then**
      $A \leftarrow A \cdot m$ {Multiply}
    else
      $T \leftarrow A \cdot m$ {Multiply and discard}
  return A

Tot i aquesta contramesura, en cas que existeixi una petita diferència, per exemple en la gestió de l'exponent, és possible extreure'n la clau. [Amiel 2009].

Una altre tipus d'algoritmes d'exponenciació són els de finestra, els quals, en comptes de tractar l'exponent bit a bit, el tracten en finestres de k bits. Aquests algoritmes necessiten precalcular alguns valors utilitzant el missatge a exponenciar. En l'algoritme 3 es precalculen les potències de m de 0 a $2^k - 1$.

Per exemple, per a $k = 3$, es precalculen les potències $m^0, m^1, \dots, m^6, m^7$. Un cop calculades, es tracta l'exponent en finestres de tres bits (000, 001, 010, 011, 100, 101, 110, 111) o el que és el mateix (0, 1, 2, 3, 4, 5, 6, 7). Per a cada finestra, es fan 3 quadrats i una multiplicació pel valor precalculat corresponent.

En aquest algoritme, no és possible extreure l'exponent distingint únicament multiplicacions de quadrats, ja que és independent de la seqüència d'operacions. Per tant, l'ús d'aquests algoritmes es pot considerar una contramesura. És necessari l'ús d'atacs horitzontals de canal lateral per a atacar-los [Järvinen 2017], que explicarem amb detall a la secció 3.

Algorithm 3 Left-to-right k-ary exponentiation

Require: m as message
Require: $(e = (e_t e_{t-1} ... e_1 e_0)_b)$ for e_i where $b = 2^k$ for some $k > 1$
Ensure: m^e
  $m_0 \leftarrow 1$
  for $i \leftarrow 1$ to $(2^k - 1)$ **do**
    $m_i \leftarrow m_{i-1} \cdot m$ {Thus $m_i = m^i$}
  $A \leftarrow 1$
  for $i \leftarrow t$ to 0 **do**
    $A \leftarrow A \cdot A^{2^k}$ {k Squares}
    $A \leftarrow A \cdot m_{e_i}$ {Multiply}
  return A

Per últim, introduirem l'algoritme 4 de finestra mòbil (*sliding window*). Es diferencia de l'anterior en el fet que necessita menys càlculs previs i menys multiplicacions. [Menezes 2007].

Aquest algoritme va recorrent l'exponent bit a bit. Mentre els bits són 0, computa quadrats. En canvi, quan troba un 1 selecciona una finestra de bits de mida k i computa k quadrats i una multiplicació per al valor precalculat corresponent. Per això es diu que és de finestra mòbil. Aquest algoritme és l'implementat per

Algorithm 4 Sliding-window exponentiation

Require: m as message
Require: $(e = (e_t e_{t-1} ... e_1 e_0)_2)$ with $e_t = 1$ and integer $k \geq 1$
Ensure: m^e
  $m_1 \leftarrow m$
  $m_2 \leftarrow m^2$
  for $i \leftarrow 1$ to $(2^{k-1} - 1)$ **do**
    $m_{2i+1} \leftarrow m_{2i-1} \cdot m_2$
  $A \leftarrow 1$
  $i \leftarrow t$
  while $i \geq 0$ **do**
    if $e_i = 0$ **then**
      $A \leftarrow A \cdot A$ {Square}
      $i \leftarrow i - 1$
    else {Find the longest bitstring $e_i e_{i-1} \dots e_l$ such that $i - l + 1 \geq k$ }
      $A \leftarrow A \cdot A$ {Square}
      $A \leftarrow A \cdot m_{(e_i e_{i-1} ... e_l)2}$ {Multiply}
      $i \leftarrow l - 1$
  return A

la llibreria Mbed TLS i és l'objecte d'atac d'aquest treball.

2.1.1.3 RSA-CRT

Per poder implementar l'RSA d'una manera encara més eficient, s'utilitza el mode RSA-CRT. Per simplicitat només veurem com s'aplica el Teorema xinès del residu (CRT, *Chinese Reminder Theorem*) per accelerar el desxifrat $m = c^d \mod n$ o la firma d'un missatge $s = m^d \mod n$. Per poder-nos referir indistintament a les dues operacions, per a explicar el CRT farem servir la següent nomenclatura: $y = x^d \mod n$.

Utilitzant el CRT, en comptes de fer una operació amb un mòdul n llarg podem substituir-ho per dues exponenciacions més curtes amb els nombres primers p i q. En tractar-se d'un tipus de transformació aritmètica, tindrem tres passos: transformar al domini de CRT, realitzar l'operació dins del domini de CRT i, finalment, fer la inversa de la transformació. [Paar 2010]

Transformació de l'input al domini de CRT Per a transformar al domini de CRT únicament hem de reduir modularment l'element base x als dos factors p i q del mòdul n. Obtenim, així, la representació modular de x.

$$x_p \equiv x \mod p$$
$$x_q \equiv x \mod q$$

Exponenciació en el domini CRT Amb les versions reduïdes de x realitzem les següents dues exponenciacions:

$$y_p \equiv x_p^{d_p} \mod p$$
$$y_q \equiv x_q^{d_q} \mod q$$

on:

$$d_p \equiv d \mod (p-1)$$
$$d_q \equiv d \mod (q-1)$$

Transformació inversa Per a combinar i realitzar la transformació inversa del CRT haurem de fer l'operació següent:

$$h = qInv \cdot (y_p - y_q) \mod p$$

$$y = y_q + h \cdot q$$

on:

$$qInv = q^{-1} \mod p$$

A la pràctica, es precalculen d_p , d_q i $qInv$, i es guarda la clau privada com els següents cinc paràmetres: (p, q, d_p , d_q, $qInv$).

2.2 Set-up

2.2.1 ChipWhisperer

ChipWisperer és un projecte *open source* creat per Colin O'Flynn com a part de la seva tesi doctoral [O'Flynn 2017]. Es tracta d'una plataforma estandarditzada per a capturar i analitzar el consum de potència de dispositius criptogràfics. L'objectiu del projecte és fer accessible a investigadors i desenvolupadors de *hardware* les eines necessàries per poder dur a terme atacs de canal lateral i d'injecció de faltes. Amb aquest objectiu, es va fundar l'empresa NewAE que proporciona *hardware* de captura, dispositius *target* i l'entorn de *software*.

2.2.1.1 ChipWhisperer Husky

El ChipWhisperer Husky és una eina per realitzar atacs de canal lateral i injecció de faltes. És capaç d'agafar mostres d'un senyal de potència en mode *streaming*, el que ho fa ideal per a capturar algoritmes asimètrics amb temps d'execució llargs, com l'RSA.

Característiques:

- **Sample Rate & ADC**: 200 MS/s, 12-bit
- **Sample Buffer Size**: > 80K Sample
- **Streaming Support (limited by computer buffer)**: >20 MS/s, 8-bit data can stream back for unlimited capture sizes.
- **Voltage Glitching**: 2-size Crowbar glitch
- **Clock Glitching**: High-resolution glitch generation based on phase-shift architecture (sub nS resolution)
- **I/O Pins**: ChipWhisperer 20-pin header, additional 8 data + 1 clock line. All I/O pins 3.3 V
- **FPGA**: Artix A35

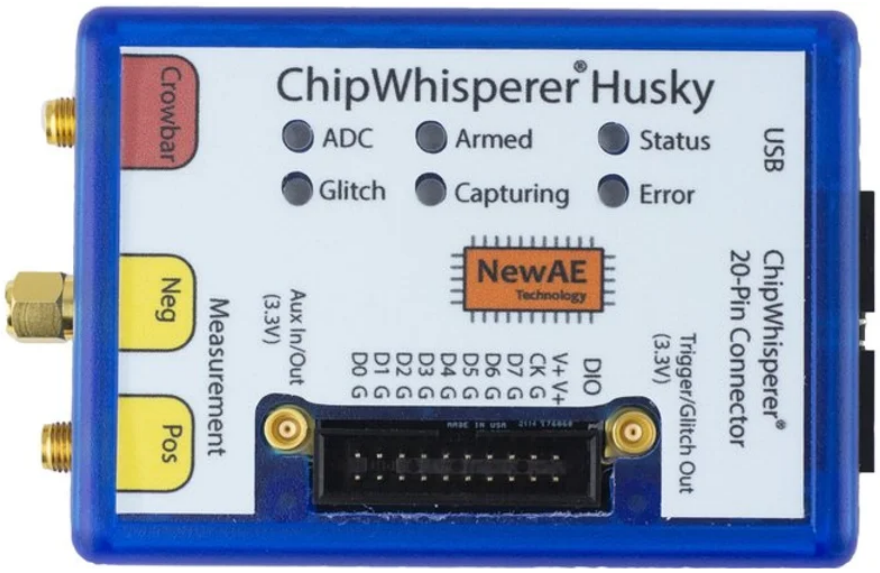

Figura 2.2: ChipWhisperer Husky [NewAE 2023d]

2.2.1.2 CW308 UFO Target Board

La placa CW308 està dissenyada per a muntar-hi a sobre diverses plaques *target*. Disposa de reguladors de potència, filtres oscil·ladors i totes les interfícies necessàries per extreure el senyal del microcontrolador i enviar-lo al dispositiu de captura.

Característiques:

- 1.2V, 1.8V, 2.5V, 3.3V and V-ADJ (1.25V - 3.5V range) power supplies.
- Oscillator driver with crystal socket to allow use of most 2 or 3-pin crystals to drive target device or ChipWhisperer.
- On-board LC low-pass filter to provide “clean” power supply for resistive shunt measurement.
- Diode protection on I/O lines to allow voltage glitch insertion on target with less risk to connected devices.
- Soft-start on input power to avoid disconnecting ChipWhisperer-Lite USB when switching power on/off.
- Includes 8-bit Atmel XMEGA and 32-bit STM32F3 (Cortex M3) target devices

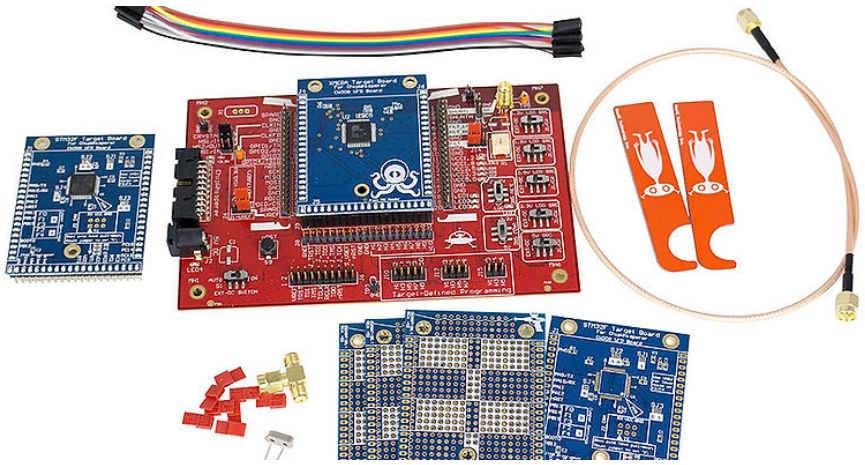

Figura 2.3: CW308 UFO Target board i accessoris [NewAE 2023b]

2.2.1.3 STM32F3 Target Board

La placa STM32F3 allotja el microcontrolador d'ST i permet connectar-lo a la placa CW308. En aquesta placa s'allotja i s'executa el codi de la llibreria Mbed TLS.

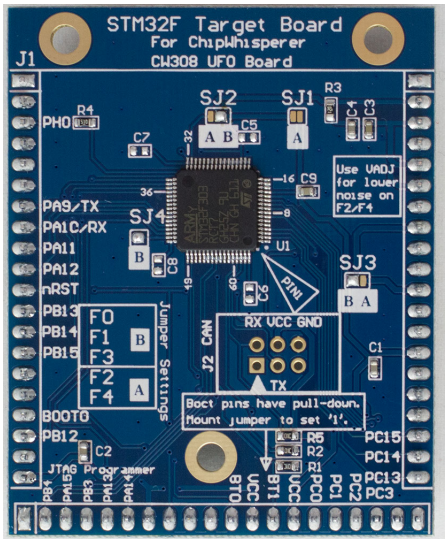

Figura 2.4: STM32F3 Target Board [NewAE 2023c]

2.2.2 Llibreria criptogràfica Mbed TLS

Mbed TLS és una llibreria criptogràfica de codi obert que proporciona funcionalitats de seguretat per a aplicacions de xarxa i de dispositius incrustats. Ori-

ginalment coneguda com PolarSSL, va ser adquirida per ARM i posteriorment rebatejada com Mbed TLS.

La llibreria ofereix una implementació eficient i fiable d'algoritmes criptogràfics com ara xifrat simètric, xifrat asimètric, algoritmes de resum, protocols de seguretat de capes (TLS/SSL) i altres funcionalitats relacionades amb la seguretat. Està dissenyada per ser lleugera, modular i fàcil d'integrar en diferents plataformes i entorns.

Mbed TLS és amplament utilitzada en una varietat d'aplicacions, com ara dispositius IoT (Internet de les coses), servidors web, dispositius de xarxa, sistemes *embedded* i aplicacions de comunicació segura en general

Mbed TLS ofereix suport per a diferents protocols de seguretat, com ara TLS 1.2 i 1.3, i proporciona eines i funcionalitats per a la gestió de certificats i claus criptogràfiques. També implementa diverses mesures de seguretat per protegir contra atacs com ara les vulnerabilitats de *side channel* i les vulnerabilitats relacionades amb la gestió de memòria.

La implementació del RSA d'aquesta llibreria fa servir l'algoritme 4.. El codi d'aquesta implementació es pot consultar a l'annex 7.3.

Per a realitzar l'atac farem servir la versió **2.5.1** ja que és la versió integrada al software de ChipWhisperer.

CAPÍTOL 3

Estat de l'art

3.1 Side-channel attacks

3.1.1 Simple Power analysis

Quan un dispositiu electrònic executa un codi o implementa una operació criptogràfica, realitza diferents operacions. Cada operació té un cost computacional diferent, algunes són més complexes i requereixen més potència, i d'altres menys. Cada operació pot tenir un perfil diferenciat. L'anàlisi directa d'una o diverses traces de potència s'anomena *Simple Power Analysis* (SPA).

Mitjançant SPA, un atacant pot deduir quines operacions està realitzant un dispositiu criptogràfic. Per exemple, si un atacant sap que un dispositiu fa una operació criptogràfica, però no sap quina, i observa deu patrons similars, pot deduir que l'operació es tracta de l'AES, ja que l'algorisme consta de deu rondes.

Els atacs d'SPA poden ser molt potents si es disposa del codi font que està executant el dispositiu, perquè llavors és possible anar mapejant cada operació del codi amb els patrons identificats a la traça de potència.

El primer atac publicat d'SPA el van realitzar Kocher *et al.* [Kocher 1999], que van mostrar com es podia atacar l'RSA utilitzant traces de consum de potència. L'algoritme d'exponenciació era el *binary exponentiation* 1 . Com hem vist al capítol 2, van ser capaços de distingir quadrats de multiplicacions i, per tant, extreure l'exponent complet de l'RSA.

3.1.2 Correlation Power Analysis

En una anàlisi de correlació de consum de potència, o CPA, es relacionen les dades tractades pel dispositiu criptogràfic i les traces de consum de potència. Es diferencia de l'SPA en el fet que es necessita una gran quantitat de traces, a vegades de l'ordre de milions. Per a realitzar un atac de CPA, s'observa un instant concret de la traça de potència en el qual s'executa l'operació d'interès. En el cas del DES i de l'AES, aquest instant es correspon habitualment amb l'ús de les caixes de substitució (S-Box) de la primera ronda de l'algoritme. [Mangard 2007]. En el cas de l'RSA, els punts habituals per a atacar són la reducció modular [Boer 2002], l'exponenciació [Messerges 1999] i la recombinació [Witteman 2009] .

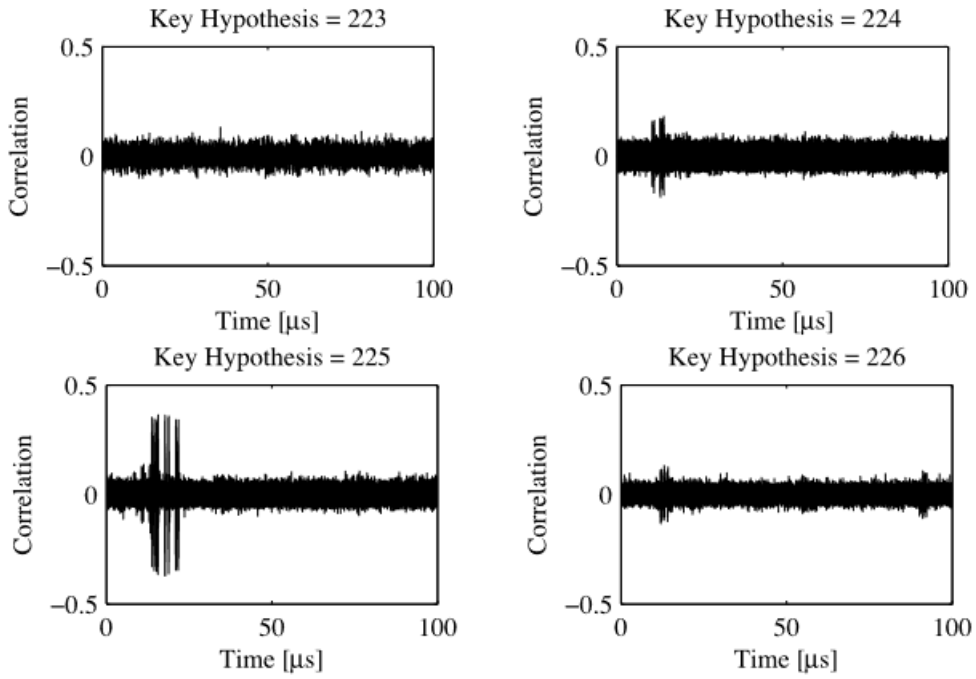

Figura 3.1: Resultat de CPA per a diferents hipòtesis de valor intermedi [Mangard 2007]

L'estratègia general per a realitzar un CPA consisteix en cinc passos [Mangard 2007]:

1. Escollir un resultat intermedi de l'algoritme criptogràfic.

2. Mesurar el consum de potència

3. Calcular els valors intermedis hipotètics.

4. Aplicar un model de potència, per exemple calcular el *Hamming Weight*, als valors intermedis hipotètics.

5. Comparar els hipotètics valors de consum de potència amb els valors de consum de potència de les traces.

A la figura 3.1 podem veure el resultat d'un CPA per a quatre claus hipotètiques. En la clau que correspon amb el valor correcte, es poden observar els pics de correlació.

3.1.3 Contramesures contra Side-channel aplicades a RSA

3.1.3.1 Ofuscació de l'exponent

Els atacs de CPA ataquen l'exponent, que és fix en múltiples traces. Per a evitar-ho, és possible ofuscar l'exponent en cada nova execució afegint-hi una màscara additiva. L'exponent secret és aleatoritzat utilitzant la següent equació [Clavier 2013]:

$$d' \leftarrow d + r \cdot \phi(n)$$

On r és un nombre aleatori i $\phi(n)$ és el totient d'Euler aplicat al mòdul n. Utilitzant l'exponent ofuscat s'obté el mateix missatge xifrat, i.e. $m^d \equiv m^{d'}$.

3.1.3.2 Ofuscació del missatge

Els atacs de CPA també aprofiten que es pot controlar el missatge o bé que el missatge és conegut. Per tal que això no passi, podem ofuscar el missatge abans de la xifra. Per fer-ho, es genera un nombre aleatori r i amb aquest es calculen r_1 i r_2 encarregats de fer impredictible el missatge d'entrada i de corregir el resultat final respectivament [Clavier 2013]:

$$r_1 = r^e \mod n$$
$$r_2 = r^{-1} \mod n$$

Llavors durant l'operació d'RSA.

$$x' = x \cdot m_1$$
$$y' = x'^d \mod n$$
$$y = y' \cdot m_2$$

3.1.4 Atacs horitzontals vs. atacs verticals

Les anàlisis com el CPA exploten la informació de forma vertical, és a dir, ataquen el mateix instant de temps entre múltiples traces.

Les anàlisis horitzontals, per contra, analitzen per a una única traça diferents instants de temps.

La figura 3.2 il·lustra aquests dos conceptes:

Els atacs horitzontals només necessiten una traça per extraure l'exponent. Per aquest motiu, la contramesura d'ofuscament de l'exponent és ineficaç perque tal com hem vist anteriorment, el exponent ofuscats son equivalents als originals per xifrar un missatge. segons $m^d \equiv m^{d'}$.

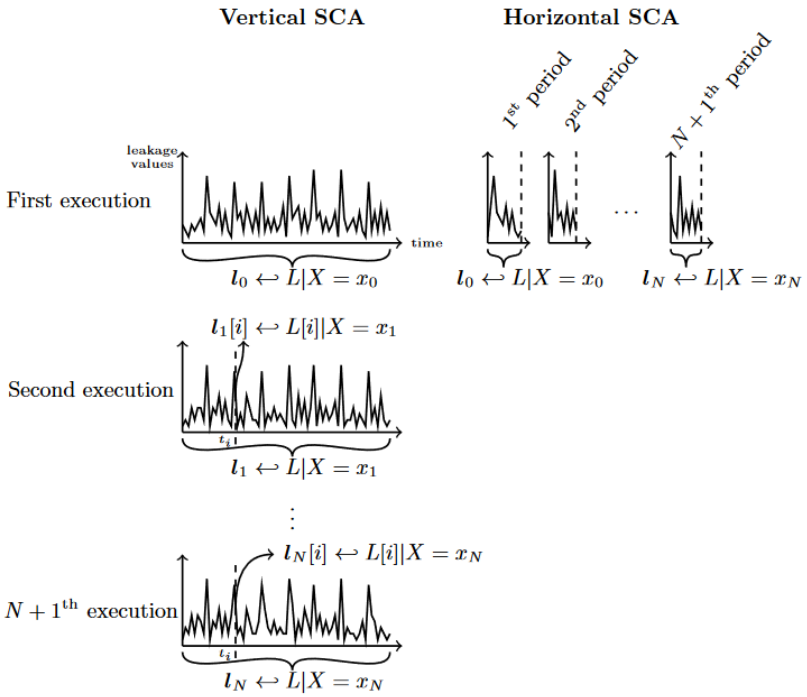

Figura 3.2: Atacs verticals i horitzontals. [Bauer 2013]

A continuació farem una breu descripció dels principals atacs horitzontals.

3.1.4.1 Big Mac attack

El conegut com a *Big Mac attack* es considera el primer atac horitzontal. Els autors expliquen així perquè van triar aquest curiós nom per a l'atac:

A well known brand product is so generously large as to be impossible to have a bite taken out of the whole at one go - like the method of attack, it must be nibbled at and consumed by tackling individual layers one by one in any order. [Walter 2001]

L'objectiu principal d'aquest atac és trobar col·lisions entre les operacions d'exponenciació realitzades en l'algoritme RSA. L'atac compara aquestes operacions per identificar quines són més semblants entre elles. Això permet distingir entre multiplicacions amb diferents valors precalculats, així com entre multiplicacions i quadrats.

Per a realitzar aquesta comparació, l'atac aprofita una característica dels algoritmes de multiplicació de nombres grans. Assumeix que s'utilitza un algoritme similar a la multiplicació "de llibre de text", on el nombre que s'està multiplicant s'inclou en la traça de consum de potència. Aquesta informació filtrada pot ser extreta mitjançant el que s'anomena *leakage*, fent mitjanes de les parts de la

traça i comparant els vectors resultants amb distàncies euclidianes. Les operacions amb menor distància euclidiana entre elles es consideren com a pertanyents al mateix valor precalculat.

Cal destacar que aquest atac assumeix que l'algoritme de multiplicació és conegut o es pot deduir per SPA.

3.1.4.2 Horizontal Correlation Analysis

A diferència de l'atac anterior, on només es feia ús de la traça, en l'atac proposat per Clavier *et al.* [Clavier 2010] s'utilitza el coneixement del missatge per realitzar hipòtesis sobre l'exponent i provar d'obtenir correlacions sobre els valors intermedis de l'operació de multiplicació.

L'atac és menys realista i en la pràctica no es pot utilitzar si el missatge està ofuscat amb un nombre aleatori suficientment gran.

3.1.4.3 Cross-correlation

Una altra proposta d'atac horitzontal consisteix a reduir tota l'operació modular a un únic valor mitjançant una operació de compressió. Un cop reduïda la traça a un vector en el qual cada mostra és una operació modular, es realitza una operació de correlació creuada, on es comparen fent ús de la correlació de Pearson totes les mostres entre elles. Les mostres que tenen una correlació alta comparteixen el mateix operand. En la figura 3.3 es pot veure el resultat d'aquesta correlació creuada per a l'algoritme 1. Cada quadrat es correspon amb la comparació entre una operació i una altra. Les úniques operacions que poden compartir un operand són les multiplicacions, ja que sempre multipliquen el registre acumulador amb el missatge inicial. Per tant, és possible llegir directament l'exponent. A l'article fan la prova també amb l'algoritme 2 i detecten les operacions de *discard*, demostrant que aquest tipus d'atac és capaç de derrotar la contramesura de *Square and multiply always*. [Witteman 2011]

En una conferència més recent [Vadnala 2017], es va provar la mateixa tècnica en algoritmes d'exponenciació de finestra com l'algoritme 4.

3.1.4.4 Clustering Analysis

L'ús d'algoritmes de *clustering* com *k-means* ha estat la tendència principal en l'estat de l'art durant la dècada dels 10. Els primers autors que proposen utilitzar aquesta tècnica no supervisada són Heyszl *et al.* [Heyszl 2013]. Utilitzant aquesta tècnica aconsegueixen reduir l'entropia per fer un atac de força bruta. Un any més tard, Perin *et al.* [Perin 2014] proposen un *framework* per realitzar aquest tipus d'atac que es divideix en quatre fases diferenciades:

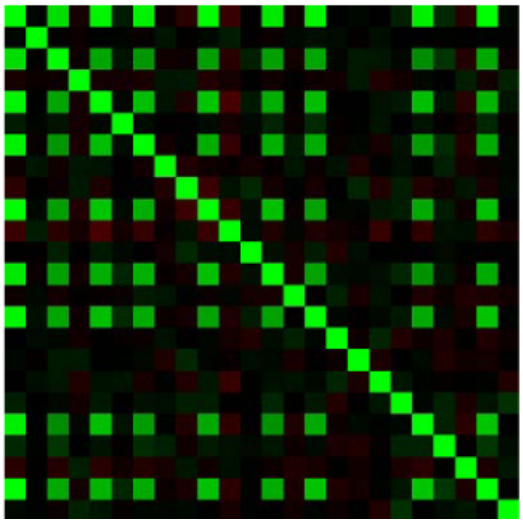

Figura 3.3: Correlació creuada entre operacions modulars.

1. Preprocessar la traça en les sub-traces necessàries
2. Trobar els punts d'interès
3. Classificació de grups mitjançant un algoritme no supervisat
4. Recuperació de l'exponent

Aquest complet *framework* proporciona instruccions clares sobre com fer aquest tipus d'atac amb eficàcia.

Els anys posteriors, el mateix autor proposa millores addicionals al *framework*, com l'ús de *Fuzzy k-Means* o *Expectation-Maximization (EM) algorithm* en comptes de *k*-means, entre d'altres millores, amb la intenció de parametritzar l'atac i fer-lo menys manual. [Perin 2015]

Nascimento *et al.* [Nascimento 2017] fan servir el *framework* de Perin per atacar Corbes El·líptiques en comptes del RSA. A més, afegeixen una capa de correcció d'errors.

És important remarcar que tots aquests articles analitzen algorismes d'exponenciació binaris, és a dir, que treballen bit a bit i per tant "només"han de distingir entre zeros i uns. Tenint això en compte, Järvinen i Balasch [Järvinen 2017] proposen un mètode per ampliar els atacs de clustering a mètodes d'exponenciació de finestra com els algorismes 3 i 4.

Finalment, al 2020, Perin li torna a donar una volta al seu *framework* afegint-hi un algoritme de *deep learning* iteratiu, el qual assegura que és capaç d'anar reduint l'error a cada iteració fins aconseguir extraure tots els bits de l'exponent. [Perin 2020]

CAPÍTOL 4

Planificació i Metodologia

4.1 Metodologia àgil

En aquest projecte s'ha seguit una adaptació de la metodologia àgil basada en un *backlog* i en l'assignació de punts d'història a les tasques. La metodologia àgil proporciona un enfocament flexible que permet adaptar-se als canvis i entregar de forma incremental funcionalitats.

4.1.1 Iteracions i lliuraments incrementals

La realització de l'anàlisi s'ha dividit en iteracions curtes, conegudes com a *sprints*, amb una durada de dues setmanes de feina cada una. Com no hem pogut estar dedicats completament al projecte, s'ha optat per associar una tasca a cada *sprint* i contar el nombre d'hores aproximat dedicat a cada tasca.

4.1.2 Priorització i gestió del backlog

El *backlog* és una part integral de la metodologia àgil. Es refereix a una llista ordenada de tasques, funcionalitats o requisits que representen el conjunt de treball que encara ha de ser realitzat i que està pendent de ser abordat.

S'ha mantingut un *backlog* de tasques i s'ha prioritzat cada tasca segons la seva importància.

4.1.3 Punts d'història

Els punts d'història són una unitat de mesura utilitzada en la metodologia àgil per estimar la complexitat o la mida d'una tasca o funcionalitat. Representen una forma relativa de comparar les tasques en funció del seu esforç o dificultat.

Habitualment es fa servir la seqüencia de Fibonacci (1, 2, 3, 5, 8, 13, 21, ...) ja que és una seqüència creixent que proporciona valors progressivament més grans. Aquesta progressió reflecteix l'augment en la complexitat relativa de les tasques a mesura que augmenten els punts d'història assignats. L'ús d'aquesta seqüència ajuda a evitar l'assignació de valors massa específics o detallats, ja que pot ser difícil i imprecís diferenciar entre tasques que són molt similars en termes de complexitat.

4.2 Backlog

A les taules següents podem veure les tasques del *backlog*, amb els corresponents punts d'història, descripció, subtasques (si escau) i lliurable associat.

Títol de Tasca	Realitzar una anàlisi de llibreries públiques de criptografia asimètrica
Punts d'història	3
Descripció	Realitzar una anàlisi de les llibreries públiques disponibles per a la implementació de l'algorisme RSA. Aquesta tasca té com a objectiu identificar i avaluar les llibreries més populars i utilitzades i comprendre les seves característiques, funcionalitats i consideracions de seguretat.
Subtasques	- Analitzar llibreria μecc - Analitzar llibreria Mbed TLS
Lliurable	Seleccionar una llibreria amb la qual desenvolupar el treball.

Taula 4.1: Tasca 1

Títol de Tasca	Configurar CW-Husky per capturar traça de RSA
Punts d'història	5
Descripció	Aquesta tasca implica configurar l'entorn i els dispositius necessaris per capturar la traça d'una operació RSA utilitzant l'eina CW-Husky.
Subtasques	- Connectar correctament els dispositius - Programar target board - Configurar paràmetres de captura
Lliurable	Traça de RSA.

Taula 4.2: Tasca 2

Títol de Tasca	Realitzar SPA
Punts d'història	5
Descripció	Realitzar una anàlisi visual de la traça de potència, identificar les regions del RSA i les operacions modulars.
Subtasques	- Identificar seccions del RSA: Precomputacions, exponenciacions i recombinació. - Identificar operacions modulars
Lliurable	Jupyter notebook amb regions identificades.

Taula 4.3: Tasca 3

Títol de Tasca	Distingir quadrats de multiplicacions
Punts d'història	8
Descripció	Utilitzar una tècnica d'anàlisi del senyal per distingir quadrats de multiplicacions. En l'algoritme Slidingwindow, si som capaços de distingir els quadrats de les multiplicacions, ja podem obtenir els bits de l'exponent que queden fora de les finestres.
Lliurable	Exponent parcial

Taula 4.4: Tasca 4

Títol de Tasca	Classificar finestres segons el valor
Punts d'història	13
Descripció	Utilitzar una tècnica d'anàlisi del senyal per distingir quins valors es tracten a cada finestra (10000, 10001, 10010, ... 11110, 11111)
Lliurable	Exponent complet

Taula 4.5: Tasca 5

4.3 Costos associats al projecte

En la següent taula 4.6 es detallen els costos associats al projecte.

Concepte	**Quantitat**	**Valor Unitari**	**Cost**
Hores científic de dades	200	50 €/hora	10000 €
Cost d'amortització de l'ordinador	200	0.11 €/hora	22 €
CW308 Target base board i targets	1	306 €	306 €
CW Husky	1	550 €	550 €
Altres materials i recursos	1	100 €	100 €
Total			**10978 €**

Taula 4.6: Costos associats al projecte

CAPÍTOL 5

Contribució Metodològica

5.1 Definició de la proposta

La idea inicial del projecte es va presentar al *NewAE ChipWhisperer Contest 2021*. [Micó Biosca 2021]. L'objectiu era generar *datasets* per a analitzar la viabilitat de realitzar atacs de *clustering* sobre algoritmes de finestra com el k-ary 3 utilitzant tres mides de k diferents (2, 3 i 4) .

La proposta va estar guardonada amb:

- ChipWhisperer-Husky [NewAE 2023d]
- CW305 Artix FPGA 7A35 Target Board [NewAE 2023a]
- Una copia signada del llibre *The Hardware Hacking Handbook* [Woudenberg 2021]

La primera tasca va consistir a buscar implementacions públiques d'RSA o ECC per a una FPGA. Tot i trobar-ne, la dificultat de programar una FPGA per adaptar-ne el codi va fer que desestimés aquesta proposta.

Després de parlar-ho amb Jean-Pierre Thibault *(Senior Security Engineer)* i Colin O'Flynn(*CEO)* a *NewAE Technology Inc.* vaig decidir encaminar el treball a atacar una llibreria pública i presentar-ne els resultats en aquest treball.

5.2 Mètode proposat per a l'anàlisi

1. Analitzar les diferents llibreries públiques d'RSA i seleccionar-ne una que utilitzi un algoritme d'exponenciació de finestra per a realitzar l'atac
2. Capturar una traça
3. Realitzar una anàlisi visual de la traça de potència, identificar les regions de l'RSA i les operacions modulars
4. Desenvolupar un mètode per a distingir quadrats de multiplicacions
5. Desenvolupar un mètode per a distingir els diferents valors precalculats
6. En cas d'obtenir resultats satisfactoris, informar als desenvolupadors de la llibreria de la vulnerabilitat.

5.3 Repetibilitat de l'anàlisi

El codi i la traça utilitzats en aquest treball es poden trobar a la següent adreça de Github: victormico/sca-mbedtls-rsa

CAPÍTOL 6

Resultats

En aquest capítol mostrarem l'atac realitzat a la implementació de l'RSA-CRT de la llibreria *Mbed TLS*. Aquesta llibreria utilitza com a mètode d'exponenciació l'algoritme *Sliding Windows* 4 amb una finestra de *5* bits.

La llibreria està instal·lada a una placa STM32F3 2.4, la qual està muntada sobre la placa mare CW308 2.3, fet que facilita l'extracció del consum de potència.

Com a dispositiu de captura s'ha fet servir el Chipwhisperer Husky 2.2 configurat per a capturar de manera síncrona i en mode *streaming*. S'han capturat **1.092.0000** mostres a una taxa de mostreig de **29,48MS/s**.

En la secció 6.1 expliquem l'SPA realitzat i com hem identificat el patró d'inici de cada finestra de bits. Aquest patró l'utilitzem per a realitzar una anàlisi de correspondència de patrons a la traça completa, que expliquem a la secció 6.2. Mitjançant aquesta anàlisi, som capaços de distingir quines operacions modulars pertanyen a la finestra de de bits i quines no. Amb aquest primer pas, ja som capaços d'aconseguir aproximadament el 30% dels bits dels exponents dp i dq.

Finalment, hem identificat la càrrega en memòria dels 5 bits de la finestra. Hi ha un patró diferenciat per a la càrrega d'un 0 i per a la càrrega d'un 1. Utilitzant la mateixa tècnica de coincidència de patrons, podem identificar els 5 bits de cada finestra i, per tant, extreure aproximadament el 100% dels bits dels exponents d_p i d_q.

6.1 SPA

La primera fase de l'SPA consisteix a identificar les dues exponenciacions de l'RSA-CRT. A la figura 6.1 veiem la traça completa, amb les dues exponenciacions identificades.

Si ampliem l'inici de la primera exponenciació, tal com mostrem a la figura 6.2, podem veure uns 15 pics que sobresurten. Es corresponen amb les 15 multiplicacions necessàries per a calcular els valors precalculats.

A la zona inicial de l'exponenciació, que mostrem ampliada a la figura 6.3, podem veure uns pics petits que es van repetint. També observem uns pics diferents. Com que la traça en cru no ens permet veure més diferències a aquest nivell, apliquem un filtre *lowpass* 5 al senyal.

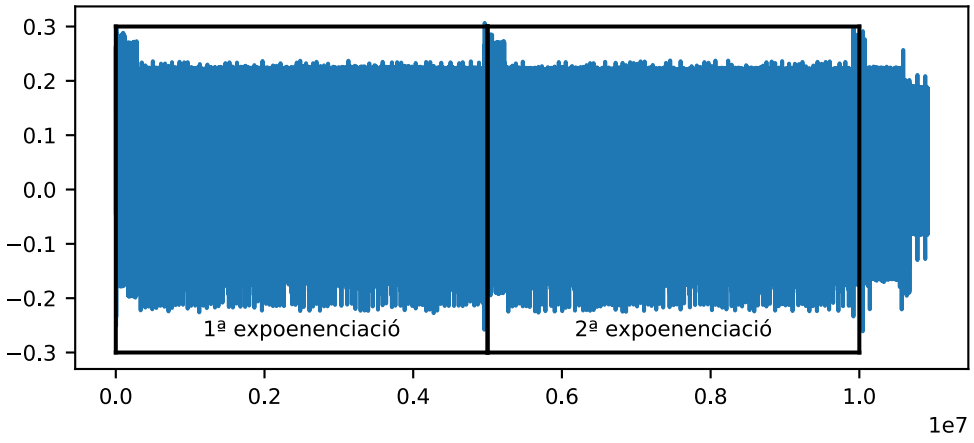

Figura 6.1: Traça RSA completa

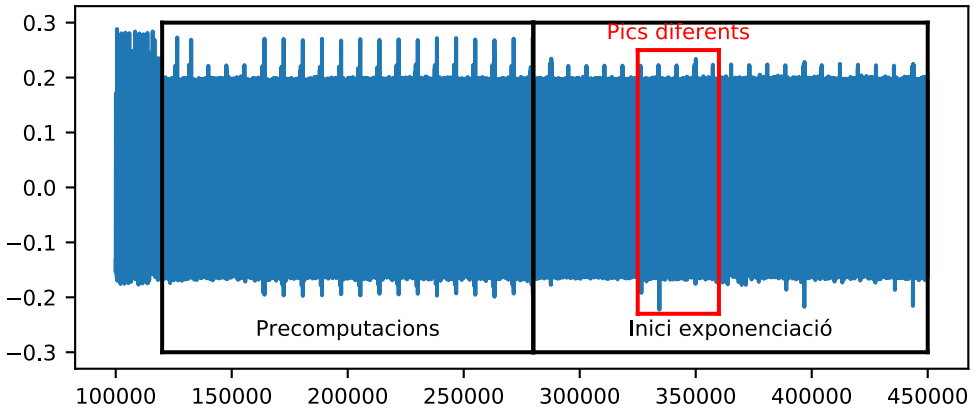

Figura 6.2: Precomputacions i inici de l'exponenciació

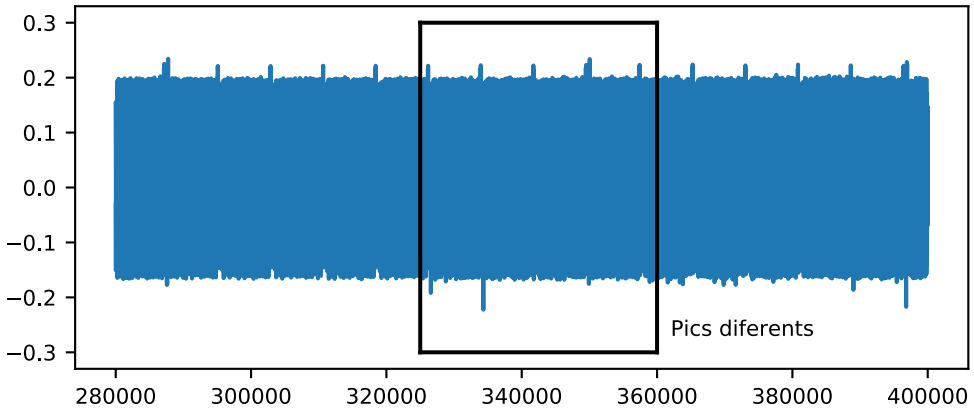

Figura 6.3: Pics diferents entre les operacions modulars

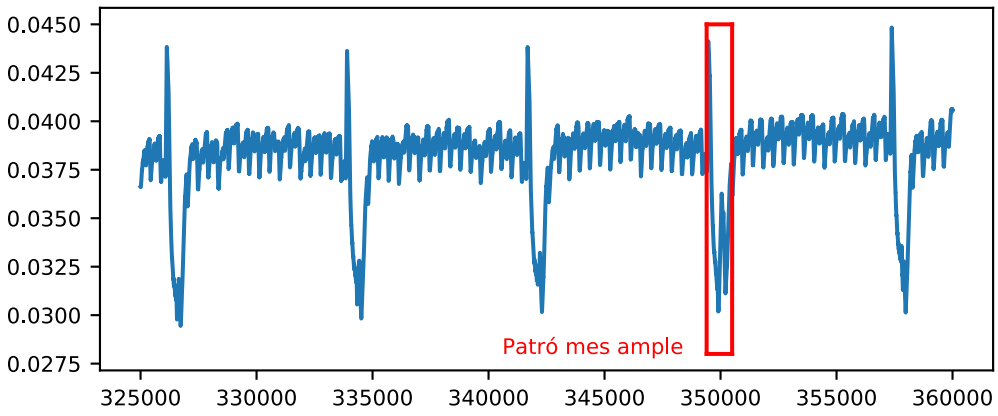

Figura 6.4: Traça filtrada amb *lowpass*

Després de provar diferents valors, ens quedem amb el filtre *lowpass* amb pes 100. A la figura 6.4 mostrem ampliada la zona indicada com a pics diferents de la figura anterior i amb el filtre *lowpass* aplicat. En aquesta figura destaquem un patró que es diferencia dels altres per ser més ample.

Observant la freqüència de repetició del patró ample, establim la hipòtesi que es tracta de l'inici de la finestra de bits. Per comprovar-ho, fem una anàlisi de correspondència de patrons.

6.2 Correspondència de patrons

En una anàlisi de correspondència de patrons, es compara un patró amb la resta de la traça. L'algoritme 6 en descriu el procés. El patró i la secció de la traça corresponent es comparen mitjançant el coeficient de correlació de Pearson [SciPy 2023b]. Aquest procés es va repetint movent la secció de la traça en increments d'una mostra. El resultat d'aquesta anàlisi és una traça en la qual cada mostra indica la probabilitat que aquella regió s'assembli al patró .

6.2.1 Identificació de quadrats i multiplicacions

La figura 6.5 mostra en detall el patró que havíem indicat a la figura 6.4. Aquest patró s'ha comparat amb tota la traça per obtenir els instants en els quals ocorre la mateixa operació durant l'exponenciació.

En la figura 6.6 veiem el resultat de la correspondència de patrons aplicada a la primera exponenciació. Podem veure clarament com el patró es va repetint. Per extraure l'índex, fem servir la funció *find peaks* de la llibreria *SciPy* [SciPy 2023a], a la qual li pots indicar un llindar i et retorna l'índex on ocorren els pics

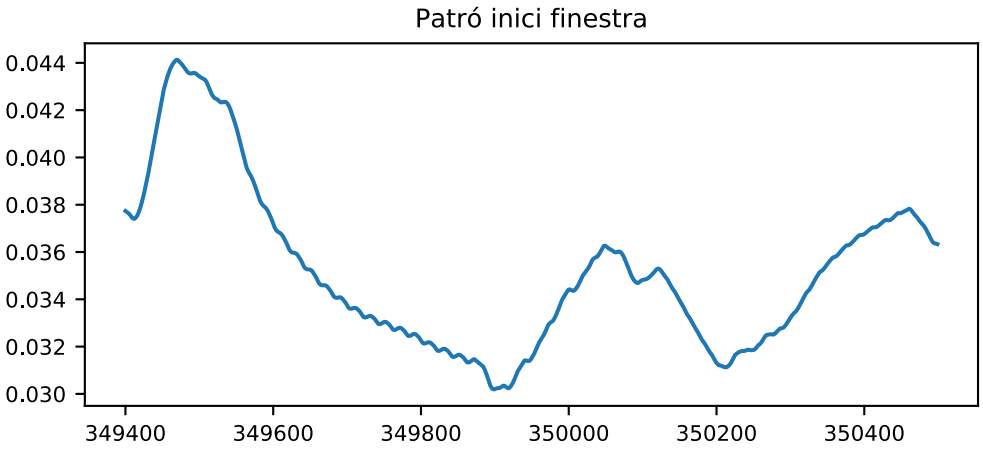

Figura 6.5: Patró inci de finestra

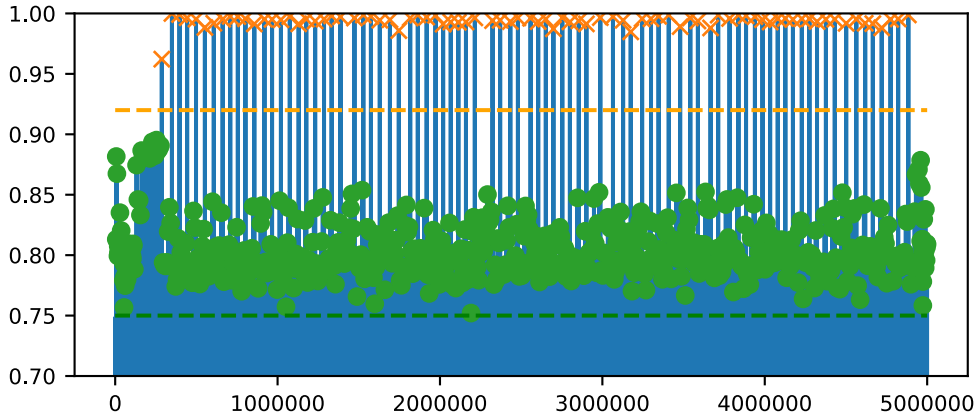

Figura 6.6: Resultat de la correspondència de patrons.

superiors a aquest llindar. En aquest cas, hem extret dos tipus de pics. D'una banda, els pics que estan per sobre del llindar de $0,9$, (marcat en taronja a la figura). Aquests pics ens indiquen l'inici de cada finestra. El segon tipus de pics seran els que estan entre $0,75$ i $0,9$ (marcat en verd a la figura). En aquest cas, ens indiquen l'inici d'una operació modular (quadrat o multiplicació).

Comptant el nombre d'operacions entre l'inici de cada finestra, podem obtenir quines operacions són quadrats o multiplicacions. Observem que el nombre mínim de pics verds (operacions modulars) entre dos pics taronja (inici finestres) és de 6, que són les operacions necessàries en un *sliding windows* de finestra de 5 bits, SQ-SQ-SQ-SQ-SQ-MUL (on *SQ* és un quadrat i *MUL* una multiplicació). D'aquesta manera confirmem la hipòtesi que el patró es corresponia amb l'inici de la finestra. Si hi ha punts verds addicionals, significa que tenim multi-

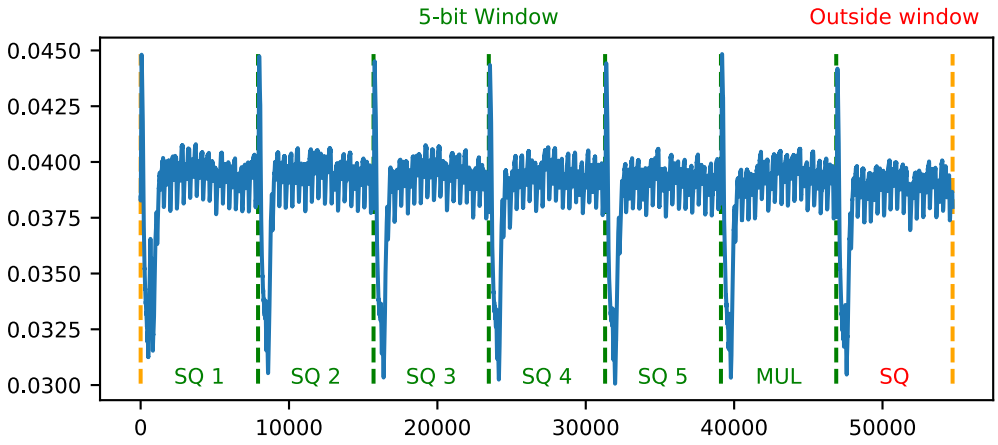

Figura 6.7: Identificació d'operacions modulars.

plicacions fora de la finestra de *sliding windows*, corresponent a quan es tracta un 0 a l'exponent.

A la figura 6.7 podem veure aquest procés més clarament. En la figura està representada la traça filtrada. Les línies discontínues indiquen el punt identificat amb el mètode de correspondència de patrons. En taronja per a l'inici de la finestra i en verd per a les operacions modulars.

Amb aquest mètode ja hem pogut obtenir alguns dels bits de cada exponenciació: els que corresponen al primer bit de la finestra que sempre és 1 i els que estan fora de les finestres, que són 0.

6.2.1.1 Bits obtinguts de la primera exponenciació

1xxxx001xxxx1xxxx1xxxx01xxxx01xxxx01xxxx01xxxx1xxxx1xxxx1xxxx01x
xxx001xxxx1xxxx01xxxx01xxxx01xxxx1xxxx01xxxx01xxxx1xxxx01xxxx000
001xxxx01xxxx1xxxx01xxxx001xxxx01xxxx0001xxxx1xxxx01xxxx01xxxx1x
xxx1xxxx1xxxx1xxxx01xxxx000000001xxxx1xxxx001xxxx1xxxx00001xxxx1
xxxx1xxxx1xxxx01xxxx1xxxx01xxxx1xxxx000001xxxx00001xxxx001xxxx1x
xxx0001xxxx01xxxx1xxxx001xxxx0001xxxx001xxxx1xxxx00001xxxx1xxxx0
001xxxx01xxxx1xxxx01xxxx1xxxx1xxxx1xxxx1xxxx1xxxx01xxxx1xxxx1xxx
x01xxxx01xxxx0001xxxx001xxxx01xxxx1xxxx01xxxx01xxxx1xxxx00xxxxxx

6.2.1.2 Bits obtinguts de la segona exponenciació

1xxxx01xxxx1xxxx1xxxx1xxxx1xxxx1xxxx0001xxxx1xxxx1xxxx1xxxx00001
xxxx1xxxx1xxxx01xxxx1xxxx1xxxx00001xxxx01xxxx01xxxx1xxxx1xxxx1xx
xx1xxxx1xxxx01xxxx01xxxx1xxxx1xxxx1xxxx1xxxx1xxxx1xxxx01xxxx1xxx

x1xxxx1xxxx01xxxx1xxxx001xxxx1xxxx1xxxx000001xxxx01xxxx1xxxx1xxx
x1xxxx00001xxxx0001xxxx0001xxxx1xxxx01xxxx01xxxx1xxxx1xxxx1xxxx0
1xxxx0001xxxx1xxxx1xxxx1xxxx1xxxx01xxxx001xxxx1xxxx0001xxxx1xxxx
1xxxx0001xxxx1xxxx1xxxx1xxxx1xxxx1xxxx00001xxxx01xxxx1xxxx1xxxx1
xxxx01xxxx01xxxx01xxxx01xxxx01xxxx0001xxxx001xxxx00001xxxxxxxxx

6.2.2 Identificació de bits dins d'una finestra

Una vegada identificades cada operació modular, va ser possible determinar la regió de la traça on es produïa la càrrega individual dels bits de la finestra. La càrrega es correspon amb la següent línia de codi:

```
wbits |= ( ei << ( wsize - nbits ) );
```

On:

- *wbits* és una variable que s'utilitza per emmagatzemar una finestra de bits.
- *ei* és el bit actual que s'està processant.
- *wsize* és la mida de la finestra, que determina quants bits es processen junts en cada iteració, en aquest cas 5.
- *nbits* és el nombre de bits processats a la finestra actual.

La línia realitza una operació de bits OR (|) entre *wbits* i (*ei* « (*wsize - nbits*)). L'expressió (*ei* « (*wsize - nbits*)) desplaça el bit ei a l'esquerra en la posició adequada dins de la finestra, segons el nombre de bits processats. Llavors, el resultat es combina amb el contingut actual de *wbits* mitjançant l'operació OR, i el resultat s'emmagatzema de nou a *wbits*.

Aquest processat produeix un consum diferent quan s'emmagatzema un zero o un u.

A la figura 6.8 podem veure els patrons corresponents.

Mitjançant el mètode de correspondència de patrons, podem distingir cada bit de la finestra. A la figura 6.9 podem veure el resultat de comparar els patrons anteriors amb el procés de càrrega dels bits.

El resultat és la completa identificació dels bits de cada finestra. Tal com es mostra a la figura 6.10

Automatitzant el procés per a cada finestra i combinant els bits que havíem extret anteriorment obtenim el següent resultat:

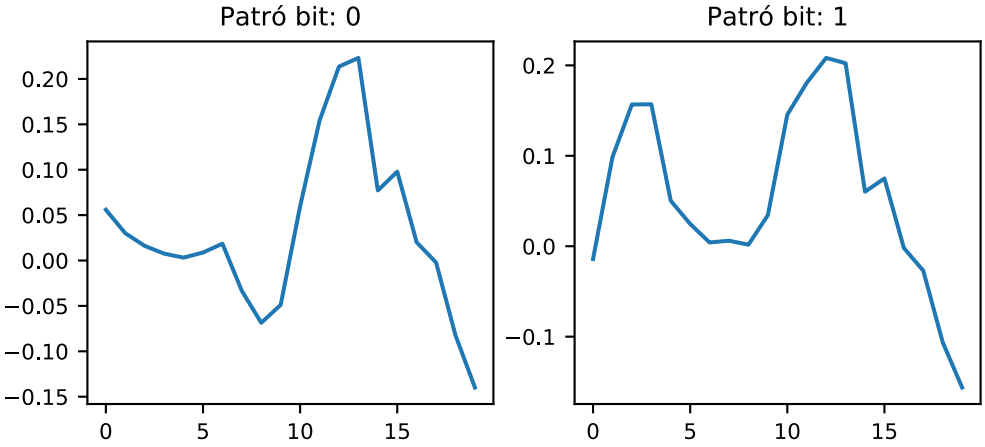

Figura 6.8: Patrons corresponents a la càrrega d'un zero i d'un u.

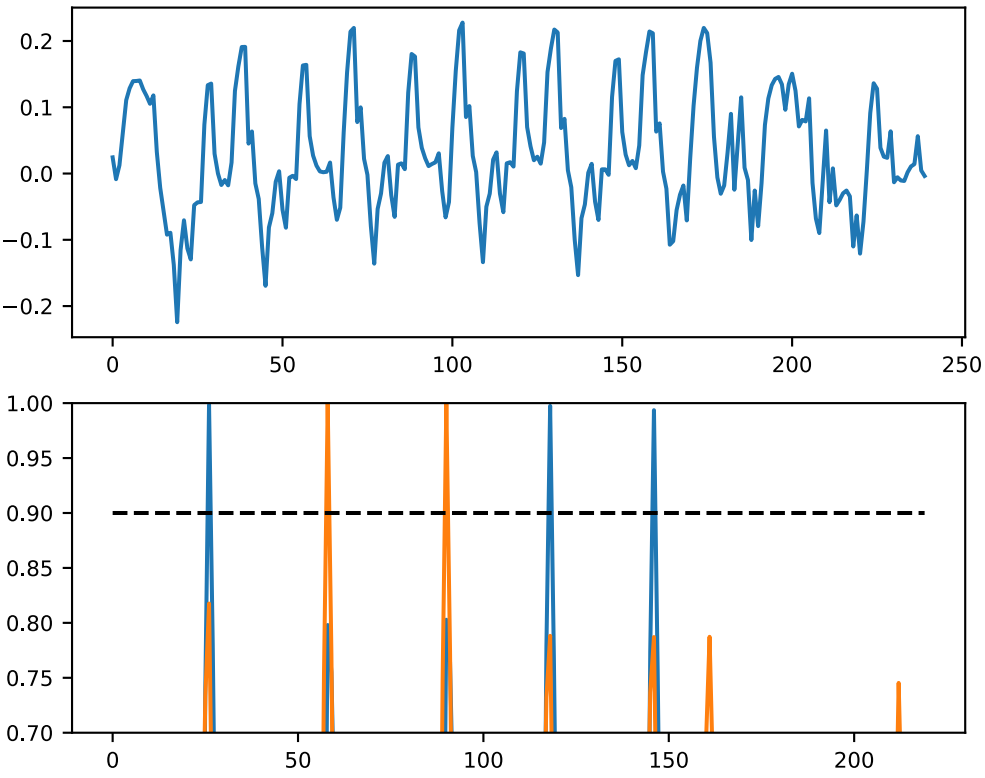

Figura 6.9: Dalt: Segment de traça corresponent a la càrrega dels bits d'una finestra Baix: Resultat de la correspondència de patrons per als bits zero i u.

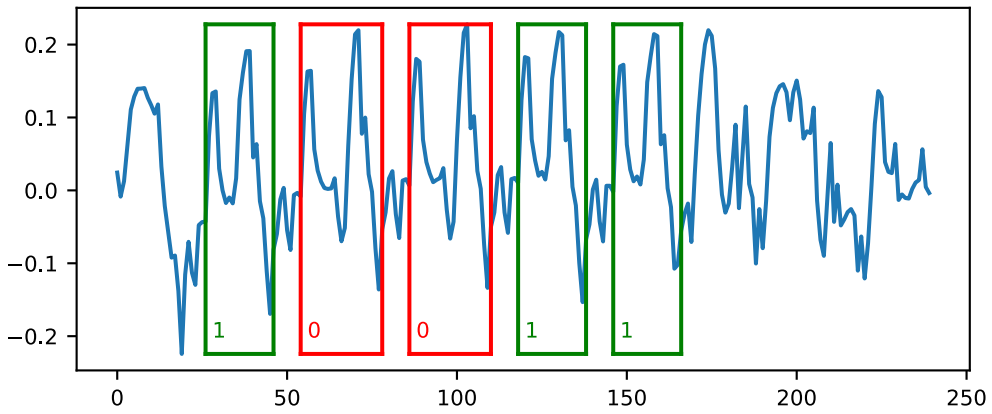

Figura 6.10: Identificació de càrrega individual de cada bit de la finestra.

6.2.2.1 Bits obtinguts de la primera exponenciació

1100000110101100111101010110011101010110010000100111010011111011
0000011110100000101110111010110101011101001001101110001010011000
0011110010010100110100100010001010001000101011001101011101100011
1111110110001110010101100000000011101101010010100111000000101101
1111100001000001100110001010010111110000011011000010111001110010
0010001101101010111001001001000011001001001111110000011111111000
0011110011000101010110011100110100100111101111000010110100011100
1010010010001000100000010001011100110000101110110101011100100102

6.2.2.2 Bits obtinguts de la segona exponenciació

1001001010110011100101111110110111110001111111011110111010100001
0101110111001101001011101111100000101010101010101111001110001111
1011000101010101110101001010110011101001001111010100010100001100
0100011001001110010111001001111011111010000010110010010101011110
1111100000111110001001100010011110010101100100101000111000110110
1011100010100100011101111111100101100100101111010100010111101100
1000000011110100001101111010111011011000001011001001011100100111
1011010001011010011001010100011100000111110010110000011010101111

6.3 Resum de resultats

La següent taula mostra un resum dels resultats obtinguts per a les dues exponenciacions durant les dues fases de l'atac.

	1ª exponenciació	2ª exponenciació
Distingir quadrats de multiplicacions	33,98%	30,91%
Distingir bits de cada finestra	99,80%	100%

Taula 6.1: Resum de resultats

CAPÍTOL 7

Conclusions i treball futur

7.1 Conclusions

En aquest treball s'ha dut a terme un atac a la versió 2.5.1 de la llibreria Mbed TLS, que implementa un algoritme d'exponenciació RSA-CRT utilitzant el mètode de *Sliding windows*.

L'atac es basa en la captura d'una traça de l'operació RSA i s'ha realitzat un SPA per identificar les dues exponenciacions. A través de l'anàlisi de correspondència de patrons, s'ha aconseguit inferir la seqüència d'operacions modulars, distingint entre quadrats i multiplicacions.

A partir d'aquesta fase, s'ha obtingut aproximadament el 30% dels bits de cada exponent. Identificant l'inici de cada finestra, s'ha pogut localitzar la regió on es carreguen els bits. Emprant la tècnica de correspondència de patrons, s'ha determinat la totalitat dels bits de tots dos exponents.

Aquest treball demostra que els atacs de canal lateral son factibles, es poden realitzar amb un pressupost ajustat i amb mètodes relativament senzills de processat de senyal.

7.2 Treball futur

Com a treball futur es proposa:

1. Actualitzar el codi de la llibreria Mbed TLS a l'última versió per comprovar si és possible explotar aquesta vulnerabilitat.

2. Provar altres dispositius *target* alternatius a l'STM32F3.

3. Utilitzar altres tècniques per extreure els valors de l'exponent, com algoritmes de *clustering*.

Aquestes iniciatives permetran ampliar el coneixement sobre la vulnerabilitat detectada i buscar solucions més efectives per protegir les implementacions criptogràfiques basades en la llibreria Mbed TLS.

Bibliografia

[Amiel 2009] Frederic Amiel, Benoit Feix, Michael Tunstall, Claire Whelan and William P. Marnane. *Distinguishing Multiplications from Squaring Operations*. In Roberto Maria Avanzi, Liam Keliher and Francesco Sica, editeurs, Selected Areas in Cryptography, Lecture Notes in Computer Science, pages 346–360, Berlin, Heidelberg, 2009. Springer. (Cited on page 8.)

[Aumasson 2017] Jean-Philippe Aumasson and Matthew D. Green. Serious cryptography: a practical introduction to modern encryption. No Starch Press, San Francisco, 2017. OCLC: ocn986236585. (Cited on page 1.)

[Bauer 2013] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff and Justine Wild. *Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations*. In Ed Dawson, editeur, Topics in Cryptology – CT-RSA 2013, Lecture Notes in Computer Science, pages 1–17, Berlin, Heidelberg, 2013. Springer. (Cited on pages vii and 18.)

[Boer 2002] Bert Boer, Kerstin Lemke and Guntram Wicke. *A DPA attack against the modular reduction within a CRT implementation of RSA*. volume 2523, pages 228–243, August 2002. (Cited on page 15.)

[Clavier 2010] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet and Vincent Verneuil. *Horizontal Correlation Analysis on Exponentiation*. In Miguel Soriano, Sihan Qing and Javier López, editeurs, Information and Communications Security, Lecture Notes in Computer Science, pages 46–61, Berlin, Heidelberg, 2010. Springer. (Cited on page 19.)

[Clavier 2013] Christophe Clavier and Benoit Feix. *Updated Recommendations for Blinded Exponentiation vs. Single Trace Analysis*. In Emmanuel Prouff, editeur, Constructive Side-Channel Analysis and Secure Design, Lecture Notes in Computer Science, pages 80–98, Berlin, Heidelberg, 2013. Springer. (Cited on page 17.)

[Coron 1999] Jean-Sébastien Coron. *Resistance Against Differential Power Analysis For Elliptic Curve Cryptosystems*. In Çetin K. Koç and Christof Paar, editeurs, Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, pages 292–302, Berlin, Heidelberg, 1999. Springer. (Cited on page 7.)

[Heyszl 2013] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis and Georg Sigl. *Clustering Algorithms for Non-Profiled Single-Execution Attacks on Exponentiations*. Technical report 438, 2013. (Cited on page 19.)

[Järvinen 2017] Kimmo Järvinen and Josep Balasch. *Single-Trace Side-Channel Attacks on Scalar Multiplications with Precomputations*. In Kerstin Lemke-Rust and Michael Tunstall, editeurs, Smart Card Research and Advanced Applications, Lecture Notes in Computer Science, pages 137–155, Cham, 2017. Springer International Publishing. (Cited on pages 8 and 20.)

[Kocher 1999] Paul Kocher, Joshua Jaffe and Benjamin Jun. *Differential Power Analysis*. In Michael Wiener, editeur, Advances in Cryptology — CRYPTO' 99, Lecture Notes in Computer Science, pages 388–397, Berlin, Heidelberg, 1999. Springer. (Cited on pages 7 and 15.)

[Mangard 2007] Stefan Mangard, Elisabeth Oswald and Thomas Popp. Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer US, 2007. (Cited on pages vii, 15 and 16.)

[Menezes 2007] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone. Handbook of applied cryptography. CRC Press, 2007. (Cited on pages 6 and 9.)

[Messerges 1999] Thomas S. Messerges, Ezzy A. Dabbish and Robert H. Sloan. *Power Analysis Attacks of Modular Exponentiation in Smartcards*. In Çetin K. Koç and Christof Paar, editeurs, Cryptographic Hardware and Embedded Systems, Lecture Notes in Computer Science, pages 144–157, Berlin, Heidelberg, 1999. Springer. (Cited on page 15.)

[Micó Biosca 2021] Victor Micó Biosca. *Clustering attacks on k-ary implementations of public-key crypto algorithms*. https://github.com/newaetech/chipwhisperer-contest-2021/issues/4, December 2021. original-date: 2021-12-02T15:10:31Z. (Cited on page 25.)

[Nascimento 2017] Erick Nascimento and Łukasz Chmielewski. *Applying Horizontal Clustering Side-Channel Attacks on Embedded ECC Implementations (Extended Version)*. Technical report, 2017. (Cited on page 20.)

[NewAE 2023a] NewAE. *NAE-CW305*. https://www.newae.com/products/NAE-CW305, 2023. (Cited on page 25.)

[NewAE 2023b] NewAE. *NAE-CW308*. https://www.newae.com/products/NAE-CW308, 2023. (Cited on pages vii and 13.)

[NewAE 2023c] NewAE. *NAE-CW308T-STM32F3*. https://www.newae.com/ufo-target-pages/NAE-CW308T-STM32F3, 2023. (Cited on pages vii and 13.)

[NewAE 2023d] NewAE. *NAE-CWHUSKY*. https://www.newae.com/products/NAE-CWHUSKY, 2023. (Cited on pages vii, 12 and 25.)

[O'Flynn 2017] Colin O'Flynn. *A Framework for Embedded Hardware Security Analysis*. July 2017. (Cited on page 11.)

[Paar 2010] Christof Paar and Jan Pelzl. Understanding cryptography: a textbook for students and practitioners. Springer, Heidelberg ; New York, 2010. OCLC: ocn527339793. (Cited on pages vii, 7 and 10.)

[Perin 2014] Guilherme Perin, Laurent Imbert, Lionel Torres and Philippe Maurine. *Attacking Randomized Exponentiations Using Unsupervised Learning*. In Emmanuel Prouff, editeur, Constructive Side-Channel Analysis and Secure Design, volume 8622, pages 144–160. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science. (Cited on page 19.)

[Perin 2015] Guilherme Perin and Lukasz Chmielewski. *A Semi-Parametric Approach for Side-Channel Attacks on Protected RSA Implementations*. 2015. (Cited on page 20.)

[Perin 2020] Guilherme Perin, Lukasz Chmielewski, Lejla Batina and Stjepan Picek. *Keep it Unsupervised: Horizontal Attacks Meet Deep Learning*. Technical report 891, 2020. (Cited on page 20.)

[SciPy 2023a] SciPy. *scipy.signal.find_peaks — SciPy v1.10.1 Manual*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html, 2023. (Cited on page 29.)

[SciPy 2023b] SciPy. *scipy.stats.pearsonr — SciPy v1.10.1 Manual*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html, 2023. (Cited on page 29.)

[Vadnala 2017] Praveen Vadnala and Lukasz Chmielewski. *Attacking OpenSSL using Side-channel Attacks*, August 2017. (Cited on page 19.)

[van Tilborg 2014] H.C.A. van Tilborg and S. Jajodia. Encyclopedia of cryptography and security. Encyclopedia of Cryptography and Security. Springer US, 2014. (Cited on page 2.)

[Walter 2001] C. D. Walter. *Sliding Windows Succumbs to Big Mac Attack*. In Çetin K. Koç, David Naccache and Christof Paar, editeurs, Cryptographic Hardware and Embedded Systems — CHES 2001, Lecture Notes in Computer Science, pages 286–299, Berlin, Heidelberg, 2001. Springer. (Cited on page 18.)

[Witteman 2009] Marc Witteman. *A DPA attack on RSA in CRT mode*. 2009. (Cited on page 15.)

[Witteman 2011] Marc F. Witteman, Jasper G. J. van Woudenberg and Federico Menarini. *Defeating RSA Multiply-Always and Message Blinding Countermeasures*. In Aggelos Kiayias, editeur, Topics in Cryptology – CT-RSA 2011, Lecture Notes in Computer Science, pages 77–88, Berlin, Heidelberg, 2011. Springer. (Cited on page 19.)

[Woudenberg 2021] Jasper van Woudenberg and Colin O'Flynn. The Hardware Hacking Handbook: Breaking Embedded Security with Hardware Attacks. No Starch Press, December 2021. Google-Books-ID: DEqatAEACAAJ. (Cited on page 25.)

Annexos

7.3 Funció d'exponenciació modular de la llibreria Mbed TLS

Listing 7.1: Mbed TLS modular exponentiation function

```
/*
 * Sliding-window exponentiation: X = A^E mod N
(HAC 14.85)
 */
int mbedtls_mpi_exp_mod( mbedtls_mpi *X, const mbedtls_mpi *A,
const mbedtls_mpi *E, const mbedtls_mpi *N, mbedtls_mpi *_RR )
{
int ret;
size_t wbits, wsize, one = 1;
size_t i, j, nblimbs;
size_t bufsize, nbits;
mbedtls_mpi_uint ei, mm, state;
mbedtls_mpi RR, T, W[ 2 << MBEDTLS_MPI_WINDOW_SIZE ], Apos;
int neg;

if( mbedtls_mpi_cmp_int( N, 0 ) < 0 || ( N->p[0] & 1 ) == 0 )
    return( MBEDTLS_ERR_MPI_BAD_INPUT_DATA );

if( mbedtls_mpi_cmp_int( E, 0 ) < 0 )
    return( MBEDTLS_ERR_MPI_BAD_INPUT_DATA );

/*
 * Init temps and window size
 */
mpi_montg_init( &mm, N );
mbedtls_mpi_init( &RR ); mbedtls_mpi_init( &T );
mbedtls_mpi_init( &Apos );
memset( W, 0, sizeof( W ) );

i = mbedtls_mpi_bitlen( E );
```

```
wsize = ( i > 671 ) ? 6 : ( i > 239 ) ? 5 :
        ( i >  79 ) ? 4 : ( i >  23 ) ? 3 : 1;

if( wsize > MBEDTLS_MPI_WINDOW_SIZE )
    wsize = MBEDTLS_MPI_WINDOW_SIZE;

j = N->n + 1;
MBEDTLS_MPI_CHK( mbedtls_mpi_grow( X, j ) );
MBEDTLS_MPI_CHK( mbedtls_mpi_grow( &W[1],  j ) );
MBEDTLS_MPI_CHK( mbedtls_mpi_grow( &T, j * 2 ) );

/*
 * Compensate for negative A (and correct at the end)
 */
neg = ( A->s == -1 );
if( neg )
{
    MBEDTLS_MPI_CHK( mbedtls_mpi_copy( &Apos, A ) );
    Apos.s = 1;
    A = &Apos;
}

/*
 * If 1st call, pre-compute R^2 mod N
 */
if( _RR == NULL || _RR->p == NULL )
{
    MBEDTLS_MPI_CHK( mbedtls_mpi_lset( &RR, 1 ) );
    MBEDTLS_MPI_CHK( mbedtls_mpi_shift_l( &RR, N->n * 2 * biL ) );
    MBEDTLS_MPI_CHK( mbedtls_mpi_mod_mpi( &RR, &RR, N ) );

    if( _RR != NULL )
        memcpy( _RR, &RR, sizeof( mbedtls_mpi ) );
}
else
    memcpy( &RR, _RR, sizeof( mbedtls_mpi ) );

/*
 * W[1] = A * R^2 * R^-1 mod N = A * R mod N
 */
```

```
if( mbedtls_mpi_cmp_mpi( A, N ) >= 0 )
    MBEDTLS_MPI_CHK( mbedtls_mpi_mod_mpi( &W[1], A, N ) );
else
    MBEDTLS_MPI_CHK( mbedtls_mpi_copy( &W[1], A ) );

MBEDTLS_MPI_CHK( mpi_montmul( &W[1], &RR, N, mm, &T ) );

/*
 * X = R^2 * R^-1 mod N = R mod N
 */
MBEDTLS_MPI_CHK( mbedtls_mpi_copy( X, &RR ) );
MBEDTLS_MPI_CHK( mpi_montred( X, N, mm, &T ) );

if( wsize > 1 )
{
    /*
     * W[1 << (wsize - 1)] = W[1] ^ (wsize - 1)
     */
    j =  one << ( wsize - 1 );

    MBEDTLS_MPI_CHK( mbedtls_mpi_grow( &W[j], N->n + 1 ) );
    MBEDTLS_MPI_CHK( mbedtls_mpi_copy( &W[j], &W[1]
) );

    for( i = 0; i < wsize - 1; i++ )
        MBEDTLS_MPI_CHK( mpi_montmul( &W[j], &W[j], N, mm, &T ) );

    /*
     * W[i] = W[i - 1] * W[1]
     */
    for( i = j + 1; i < ( one << wsize ); i++ )
    {
        MBEDTLS_MPI_CHK( mbedtls_mpi_grow( &W[i], N->n + 1 ) );
        MBEDTLS_MPI_CHK( mbedtls_mpi_copy( &W[i], &W[i - 1] ) );

        MBEDTLS_MPI_CHK( mpi_montmul( &W[i], &W[1], N, mm, &T ) );
    }
}

nblimbs = E->n;
bufsize = 0;
```

```
nbits   = 0;
wbits   = 0;
state   = 0;

while( 1 )
{
    if( bufsize == 0 )
    {
        if( nblimbs == 0 )
            break;

        nblimbs--;

        bufsize = sizeof( mbedtls_mpi_uint ) << 3;
    }

    bufsize--;

    ei = (E->p[nblimbs] >> bufsize) & 1;

    /*
     * skip leading 0s
     */
    if( ei == 0 && state == 0 )
        continue;

    if( ei == 0 && state == 1 )
    {
        /*
         * out of window, square X
         */
        MBEDTLS_MPI_CHK( mpi_montmul( X, X, N, mm, &T ) );
        continue;
    }

    /*
     * add ei to current window
     */
    state = 2;

    nbits++;
```

```
        wbits |= ( ei << ( wsize - nbits ) );

        if( nbits == wsize )
        {
            /*
             * X = X^wsize R^-1 mod N
             */
            for( i = 0; i < wsize; i++ )
                MBEDTLS_MPI_CHK( mpi_montmul( X, X, N, mm, &T ) );

            /*
             * X = X * W[wbits] R^-1 mod N
             */
            MBEDTLS_MPI_CHK( mpi_montmul( X, &W[wbits], N, mm, &T ) );

            state--;
            nbits = 0;
            wbits = 0;
        }
    }

    /*
     * process the remaining bits
     */
    for( i = 0; i < nbits; i++ )
    {
        MBEDTLS_MPI_CHK( mpi_montmul( X, X, N, mm, &T ) );

        wbits <<= 1;

        if( ( wbits & ( one << wsize ) ) != 0 )
            MBEDTLS_MPI_CHK( mpi_montmul( X, &W[1], N, mm, &T ) );
    }

    /*
     * X = A^E * R * R^-1 mod N = A^E mod N
     */
    MBEDTLS_MPI_CHK( mpi_montred( X, N, mm, &T ) );

    if( neg && E->n != 0 && ( E->p[0] & 1 ) != 0 )
    {
```

```
    X->s = -1;
    MBEDTLS_MPI_CHK( mbedtls_mpi_add_mpi( X, N, X ) );
}

cleanup:

for( i = ( one << ( wsize - 1 ) ); i < ( one << wsize ); i++ )
    mbedtls_mpi_free( &W[i] );

mbedtls_mpi_free( &W[1] );
mbedtls_mpi_free( &T );
mbedtls_mpi_free( &Apos );

if( _RR == NULL || _RR->p == NULL )
    mbedtls_mpi_free( &RR );

return( ret );
}
```

7.4 Algoritmes de processat de senyal

Algorithm 5 Lowpass filter

Require: t as Trace to filter
Require: $weight$ as Weight of the lowpass filter
Ensure: $result$ Trace filtered
  $weight_1 \leftarrow weight + 1$
  $N \leftarrow length(trace)$
  for $i \leftarrow 1 to N$ **do**
    $result[i] \leftarrow (result[i] + weight * result[i-1])/weight_1$
  $i \leftarrow N - 2$
  while $i \geq 0$ **do**
    $result[i] \leftarrow (result[i] + weight * result[i+1])/weight_1$
  return $result$

Algorithm 6 Pattern match

Require: t as trace
Require: ref as Reference pattern
Ensure: $scores$
  $N \leftarrow length(trace)$
  $n \leftarrow length(ref)$
  for $i \leftarrow 1 to N$ **do**
    $score[i] \leftarrow corr(ref, trace(i, i+n)$
  return $score$