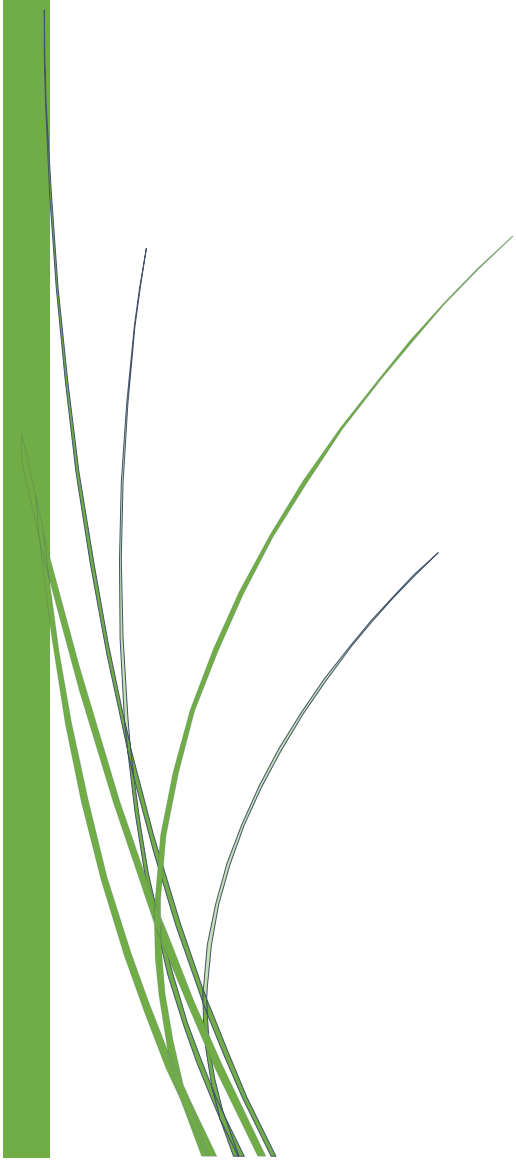
CRIPTOGRAFIA DE WHITE BOX

Aplicació dels atacs d'injecció de falta.



Alexis Gutiérrez Mercader
Grau d'enginyeria informàtica 2016-2017
Consultor: Cristina Pérez Solà
Data: 7 de juny del 2017

Agrair a la meva família i en especial als meus pares José María i Rosa María i a la meva parella Vanesa per la seva paciència i comprensió que han mostrat en aquests mesos de gran treball i nervis. Sense el seu esforç mai hauria arribat fins aquí. Gràcies per tots aquest valors que m'heu ensenyat i per recordar-me que per arribar als objectius que et proposes s'ha de continuar lluitant dia a dia.

A més agrair a tot l'equip *d'Applus Laboratories* per la seva ajuda i en especial en David Hernández i a l'Arnau Vives, ja que sense ells aquest treball no hagués estat possible.

Per últim, només em queda esmentar a Cristina Pérez, la meva tutora del treball per donar-me les directrius necessàries per seguir treballant i arribar a aquest resultat final del que em sento molt orgullós.

Moltes gràcies a tots.

Resum

Aquest treball forma part dels estudis del grau en enginyeria informàtica de la Universitat Oberta de Catalunya. Pertany a l'àrea de seguretat informàtica i la part pràctica d'aquest s'ha portat a terme a Applus Laboratories i és de caràcter confidencial. Durant el treball s'estudia la possibilitat d'aplicar un atac de falta a un algoritme AES en un entorn White Box (WB) en el que l'atacant té un control absolut sobre el dispositiu i l'execució de l'aplicació.

Durant el present escrit a més d'aprofundir en la realització de l'atac en qüestió, s'estudia l'estat de l'art pel que fa a l'algoritme AES i a les tècniques per la seva translació a un entorn WB. L'estudi s'ha realitzat per obtenir els coneixements necessaris per poder arribar al resultat desitjat.

Per recolzar l'explicació del treball i poder entendre millor tant com funciona l'algoritme AES com l'atac escollit, es realitza una implementació pràctica pròpia en Python que s'entrega juntament amb aquesta memòria.

Paraules clau

Seguretat informàtica, Criptografia, White Box, AES, atacs de falta.

Abstract

This work is part of computer engineering degree of the Universitat Oberta de Catalunya. It belongs to the area of computer security and the practical part of this has been carried out at Applus Laboratories and it is confidential. In this paper we study the possibility of applying a fault attack in AES algorithm in White Box Cryptography (WBC). In this context the attacker has complete control over the device and the application.

Besides, this report examines the state of the art of AES algorithm and techniques for transfer it to a WB context. This study is performed to obtain the necessary knowledge to reach the desired result.

We developed a practical implementation of AES algorithm in Python to support the explanation of the work and therefore understand the behavior of AES algorithm and the attack. This implementation is delivered with this report.

Taula de continguts

Taula de figures

1. Introducció

1.1. Justificació del treball i context

Vivim en un món en que els nous sistemes d'informació han guanyat cada vegada més importància en les nostres vides. Milions i milions de terabytes de dades viatgen cada dia a través de la xarxa i la seguretat d'aquestes esdevé cada vegada més important. La criptografia és l'encarregada de jugar aquest paper.

A l'hora de definir la criptologia és molt útil la pròpia etimologia de les paraules. L'origen de les paraules prové dels termes grecs *kryptós* (ocult) i *graphein* (escriure) o *logia* (estudi) respectivament [1]. Per tant quan es parla de criptologia s'apunta a la ciència que estudia com amagar informació. Es poden trobar dues branques relacionades amb la criptologia, per una banda la criptografia que s'encarrega de l'estudi del xifratge de missatges, i per una altra la criptoanàlisi que s'encarrega d'estudiar els mètodes per trencar la seguretat obtinguda a través dels diferents xifratges.

Segons el principi de Kerckhoffs (àmpliament acceptat) [2], la seguretat de la criptografia es basa en l'ús de claus secretes i no en ocultar el funcionament dels algoritmes. Un algoritme és segur quan encara coneixent al detall el seu funcionament és impossible desxifrar el missatge sense la clau corresponent. Per tant els algoritmes més utilitzats en l'actualitat són totalment públics i amb una recerca senzilla qualsevol persona pot estudiar el seu funcionament.

Fins fa pocs anys es podia assumir que les operacions criptogràfiques de xifratge i desxifratge a través d'una clau es realitzaven en un entorn segur, de manera que ningú podia accedir i observar el procés de l'algoritme en qüestió des de dins. Amb aquestes condicions, si un atacant volia accedir a la informació havia d'atacar des de fora (model de caixa negra o *black box*) ja sigui amb atacs de *side channel*, injecció de faltes o qualsevol altre atac físic o lògic [3]. Però què succeeix quan l'atacant pot ficar-se dins del funcionament de l'algoritme i observar tots els seus processos? Continuen sent segurs aquests algoritmes? Aquestes i d'altres preguntes formen part de l'abast de la criptografia de *White Box (WBC)*.

La WBC respon a la necessitat de protegir claus criptogràfiques en entorns oberts, com per exemple un ordinador, *tablet* o *smartphone*, on un atacant té control total del dispositiu i, per tant, tota la informació emmagatzemada és extremadament vulnerable.

En els últims anys cada vegada es disposa de més solucions a partir d'implementacions exclusives en software i aquest fet fa que cada vegada la WBC tingui més importància. Durant aquest treball s'estudiaran quines són les estratègies que utilitza la WBC per aconseguir mantenir la confidencialitat en el pitjor entorn d'atac possible. A més es tractarà l'aplicació d'alguns dels atacs presents en la criptografia clàssica a un entorn *White Box (WB)*. Sobretot es dirigirà el focus als atacs d'injecció de faltes i a si és possible aplicar aquest tipus d'atacs a un entorn *White Box* per vulnerar la seguretat i d'aquesta manera estudiar fins a quin punt són segures les estratègies utilitzades en els entorns *WB*.

1.2. Objectius

L'objectiu principal d'aquest treball és avaluar la viabilitat dels atacs d'injecció de falta contra implementacions criptogràfiques software en un entorn *WB.* L'algoritme escollit per realitzar aquesta anàlisi és l'AES (*Advanced Encryption Standard*), per ser l'algoritme simètric més utilitzat en aplicacions de seguretat i, per tant, un dels més implementats en entorns *WB*. Per arribar a aquest objectiu principal s'hauran d'aconseguir altres objectius intermedis. La consecució d'aquests objectius farà que durant l'elaboració del treball es puguin reflectir els coneixements necessaris per arribar a la fita proposada.

En primer lloc, el primer objectiu serà entendre al detall com funciona el xifratge i desxifratge de l'algoritme AES, ja que aquest coneixement serà molt útil per l'objectiu final. Per tal de demostrar que s'ha assolit aquest coneixement amb èxit es desenvoluparà una eina que pugui xifrar missatges amb un algoritme AES clàssic. D'aquesta manera també s'estudiaran cadascun dels processos d'aquest algoritme per tal de posteriorment entendre la seva adaptació a la *WB*.

El segon pas serà estudiar què és la WBC. D'aquesta manera es tindrà una visió general de quines són les seves particularitats i la importància que pot tenir en el futur. Una vegada assolit aquest objectiu ja es disposarà del coneixement necessari de l'entorn en el que es desenvoluparà tot el treball.

A continuació es tractaran quines són les estratègies que s'utilitzen per adaptar els algoritmes a un entorn tant insegur com el de la *WB*. Saber quines són aquestes estratègies i en què consisteixen serà molt útil per donar el següent pas que consistirà en estudiar una implementació de l'AES a la *WB*.

Una vegada obtinguts els objectius anteriors, el treball es centrarà en l'estudi dels possibles atacs als que pot estar exposada la criptografia clàssica. El focus principal seran els atacs de falta i s'estudiarà el funcionament d'aquest tipus d'atac en l'algoritme AES (a través de la implementació realitzada) per després poder aplicar-ho a la WB.

Amb tots aquests objectius intermedis acomplerts arribarà el moment d'afrontar l'objectiu final d'atacar la implementació de l'AES escollida per tal d'avaluar la viabilitat d'aquest tipus d'atacs i poder explicar amb deteniment totes les tècniques utilitzades per aconseguir-ho. Amb la consecució de tots els objectius proposats es reflectiran alguns dels coneixements obtinguts durant el grau i s'aprofundirà en d'altres que no en formaven part.

Tot el treball s'ha portat a terme en el laboratori d'IT d'Applus Laboratories i les conclusions i el treball fet serviran per poder, en un futur, desenvolupar eines que serveixin per analitzar la seguretat d'implementacions d'algoritmes a la White Box.

1.3. Enfocament i metodologia

En el present treball es combina l'estudi teòric de l'estat de l'art del tema escollit amb l'aplicació pràctica d'aquests coneixements per arribar al producte final.

El treball es basa en una metodologia deductiva en la que es passa dels conceptes necessaris més generals (criptografia i *White Box*) fins arribar a un objectiu final concret, atacar un algoritme AES a la *White Box*. Les tasques s'han desenvolupat seqüencialment però s'han anat millorant posteriorment per donar lloc a un treball final homogeni i uniforme.

La memòria de treball s'ha anat formant de manera paral·lela a la realització del treball per tal de reflectir totes les tasques realitzades.

1.4. Planificació del projecte

Dels objectius fixats i la metodologia es desprenen les següents tasques que s'han portat a terme durant la realització del treball:

- Estudi general de l'estat de l'art de la WBC.

- *Advanced Encryption Standard.* Estudi en profunditat del seu funcionament.

- *Advanced Encryption Standard*. Implementació.

- Estratègies lligades a la WBC. Estudi en profunditat.

- Elecció de la implementació WB de l'AES pel posterior estudi i atac.

- Atacs a la criptografia clàssica. Revisió de l'estat de l'art.

- Atacs de injecció de faltes a la *WB*.

- Elecció de l'atac a la implementació de l'AES escollida.

- Tècniques utilitzades durant l'atac. Revisió.

- Anàlisi dels resultats i conclusions.

En els annexos es pot trobar el diagrama de Gantt associat a la planificació.

1.5. Productes obtinguts

El producte final d'aquest Treball de Fi de Grau és aquesta memòria de treball en la que es reflecteix l'estat de l'art de cadascun tels temes tractats i el desenvolupament pràctic de la implementació de l'AES i de l'atac realitzat. Juntament amb aquest text s'entrega la implementació Python de l'algoritme AES i els scripts que s'han realitzat per poder efectuar els atacs amb èxit.

1.6. Breu descripció dels capítols

Seguidament en el capítol 2 es fa una explicació del que és un **algoritme criptogràfic,** distingint entre els algoritmes simètrics i els asimètrics.

El capítol 3 es centra en analitzar l'**AES**, que és un conegut algoritme simètric. Durant el capítol es pot veure en profunditat el seu funcionament veient detingudament cadascuna de les seves rondes. L'explicació es recolza amb captures de pantalla de la implementació pròpia de l'algoritme.

En el capítol 4 el treball s'endinsa l'altre concepte clau , l'entorn White Box. Amb aquesta motivació es fa una diferenciació entre els **diferents escenaris** que s'apliquen a l'hora d'estudiar la criptografia moderna.

El capítol 5 fa un recull de les **diferents estratègies** que es segueixen per tal de **traslladar un algoritme AES a un entorn White Box.**

En el capítol 6 s'aprofundeix en els **atacs més comuns en la criptografia actual**. En concret en els atacs de Side Channel i de falta.

Al capítol 7 es dedica a l'explicació **del desenvolupament de l'atac escollit contra l'AES**.

El capítol 8 és el centre de l'objectiu del present treball, és a dir, **l'aplicació de l'atac escollit sobre un entorn White Box** per tal d'estudiar la seva viabilitat.

Finalment el capítol 9 presenta les **conclusions** extretes i les línies de **treball futur**.

2. Algoritmes criptogràfics

En aquest apartat s'explica que és un algoritme criptogràfic i les seves principals característiques. A més es distingeixen els diferents tipus d'algoritmes criptogràfics que es poden trobar i en què consisteix cadascun.

Un algoritme criptogràfic és un conjunt d'operacions que modifiquen unes dades d'entrada per tal d'aconseguir un o més requeriments de seguretat. Els objectius més comuns d'aquests algoritmes són els següents [2]:

-**Confidencialitat**. Garantir la impossibilitat de que un atacant pugui accedir a la informació.

- **Autenticació**. Garantir que les dades enviades són propietat de l'usuari o dispositiu correcte.

- **Integritat**. Garantir que la informació que rep el destinatari és la mateixa que va enviar el remitent sense cap modificació.

Un algoritme es considera **computacionalment segur** quan un atacant no pot aconseguir desxifrar les dades d'entrada amb un mètode més àgil que la cerca exhaustiva de la clau. Les claus han de ser secretes i han de ser d'una longitud suficient per aconseguir que la cerca exhaustiva sigui inviable a nivell pràctic amb la tecnologia disponible fins al moment [4]. Per arribar a aquest objectiu els algoritmes criptogràfics han de garantir dues propietats [1]:

- **Difusió**. El compliment d'aquesta propietat fa que qualsevol modificació de la clau o del missatge d'entrada es propagui per la major part del missatge encriptat.

- **Confusió**. La relació entre el text sense xifrar, la clau i el missatge xifrat ha de ser el més complicada possible i per tant no lineal.

Els dos tipus d'algoritmes criptogràfics més utilitzats són els simètrics i els asimètrics [5]:

- **Algoritmes simètrics.**

  En els algoritmes simètrics la seguretat recau en una mateixa clau secreta amb la que es xifren i desxifren les dades. El DES i l'AES són exemples d'algoritmes simètrics.

  A la **Figura 1** es pot veure el funcionament d'aquest tipus d'algoritmes:

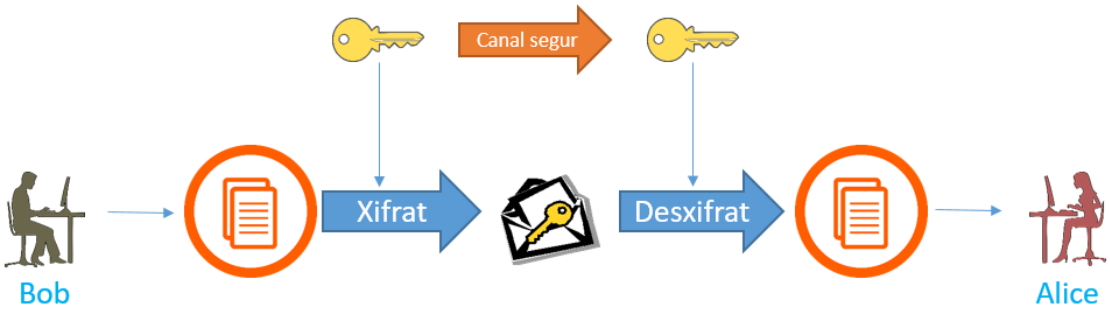


Figura 1. Funcionament algoritme simètric. Bob envia un missatge xifrat amb una clau simètrica a Alice.

- **Algoritmes asimètrics.**

Els algoritmes asimètrics utilitzen un parell de claus per xifrar. Una d'elles, amb la que es xifra el missatge, és pública, i l'altra, amb la que es desxifra el missatge, és privada i només coneguda pel destinatari del missatge. Aquestes dues claus estan relacionades matemàticament per tal de que el xifratge i desxifratge sigui possible i d'aquesta manera s'evita que la clau privada pugui ser interceptada per un atacant, ja que en cap moment serà comunicada. Com a exemple d'algoritmes de criptografia asimètrica es poden destacar el RSA o el DSA.

A la **Figura 2** es pot observar el funcionament d'un algoritme asimètric:

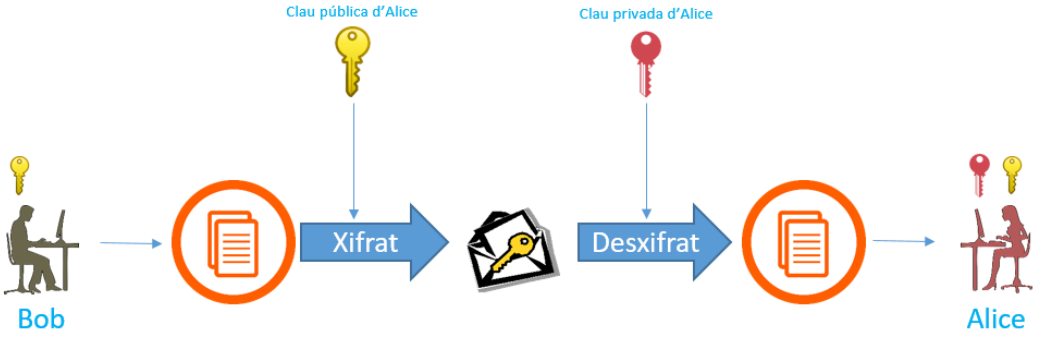


Figura 2. Funcionament algoritme asimètric. Bob envia un missatge xifrat a Alice amb la clau pública d'aquesta. Ella ho desxifra amb la seva clau privada.

En el següent capítol es detalla el funcionament de l'AES ja que és el focus principal del present treball.

3. Advanced Encryption Standard (AES)

L'AES (Advanced Encryption Standard) va néixer al 2001 als Estats Units arrel d'una competició promoguda pel NIST (National Institute of Standards and Technology) per trobar un algoritme simètric que atorgués una seguretat més elevada que el DES. L'algoritme va ser dissenyat per Joan Daemen i Vincent Rijmen [6].

En aquest cas l'AES és un algoritme de xifratge simètric de blocs de 128 bits amb tres versions amb diferents mides de clau (128, 192 i 256 bits). Segons la mida de la clau el nombre de rondes de l'algoritme serà de 10, 12 o 14 rondes respectivament [6]. Durant tot el treball l'estudi es centrarà en la versió amb clau de 128 bits ja que el funcionament és anàleg per la resta de mides de clau, per tant totes les referències a l'algoritme d'aquí en endavant es referiran a l'AES-128. A continuació podem veure el funcionament de l'AES en detall. Per l'explicació detallada de cadascun dels processos interns de l'algoritme utilitzarem una implementació pròpia de l'AES desenvolupada en Python.

Com ja s'ha comentat amb anterioritat l'AES és un algoritme que consta de 10 rondes. Cadascuna de les rondes compta amb quatre processos: SubBytes, ShiftRows, MixColumns i AddRoundKey. S'ha de tenir en compte que abans de començar la primera ronda s'efectua un AddRoundKey previ i que l'última ronda no compta amb el MixColumns [6]. Per tant l'esquema de xifratge de l'algoritme queda de la següent manera:

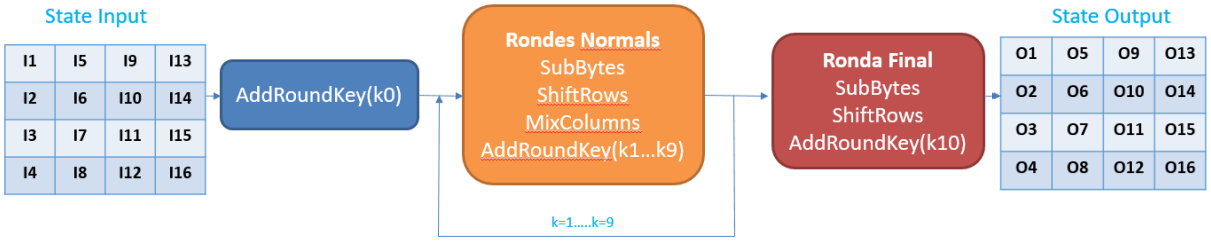


Figura 3. Esquema xifratge AES.

A l'esquema es pot observar que l'input i l'output de l'algoritme tenen forma de matriu de 4x4 (en aquest cas denominada State). Totes les operacions de l'algoritme es realitzen sobre aquesta matriu donant lloc a noves matrius State. De la mateixa manera, els bits de les subclaus també s'organitzen en forma de matriu per fer els càlculs

pertinents. A la **Figura 4** , es pot observar un exemple de clau i input a la implementació AES i la formació dels States corresponents.

Figura 4. Formació dels States.

A continuació es presenten els diferents mètodes que formen part del xifratge a través de l'algoritme AES. Però abans d'entrar en detall s'ha de tenir en compte que l'AES és un algoritme que treballa amb bytes representats en un camp de Galois (GF) de 2^8. Això permet que el resultat dels càlculs de multiplicacions i sumes mai sobrepassi la mida d'un byte sense necessitat d'arrodoniments ni truncaments.

Els valors d'un byte es representen com un polinomi amb coeficients per x^7,x^6,x^5,x^4,x^3,x^2,x^1 i 1. Per exemple, vegem com es representa el primer byte de la clau utilitzada en la **Figura 4:**

$$valor\ hexadecimal: 2B$$

$$valor\ binari: 0010\ 1011 \quad (1)$$

$$forma\ polinòmica:\ x^5+x^3+x+1$$

Per realitzar una suma s'ha de tenir en compte que la suma de dos bytes serà igual a la suma dels coeficients del polinomi mòdul 2. Això equivaldrà a realitzar una XOR tant per una suma com una resta. Ja que si els coeficients són iguals donaran com a resultat d'una suma o una resta un 0, i si són diferents donaran com a resultat un 1.

Pel que fa a la multiplicació es realitza mòdul un polinomi irreductible de grau 8. En l'AES aquest polinomi és el següent:

$$x^8+x^4+x^3+x+1 \quad (2)$$

Vegem un exemple de multiplicació:

$$multiplicació\ hexadecimal: 0x83 \cdot 0x57$$

$$x^7+x+1 \cdot x^6+x^4+x^2+x+1$$
$$= x^{13}+x^{11}+x^9+x^8+2x^7+x^6+x^5+x^4+x^3+2x^2+2x+1\ mod\ 2$$
$$= x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1 \quad (3)$$
$$x^{13}+x^{11}+x^9+x^8+x^6+x^5+x^4+x^3+1\ mod\ x^8+x^4+x^3+x+1 = x^7+x^6+1$$
$$= 1100\ 0001b = 0xC1$$

3.1. AddRoundKey

El mètode AddRoundKey consisteix en calcular una XOR de l'State d'entrada al mètode amb la subclau de la ronda corresponent. Aquestes subclaus són calculades a través del mètode de derivació de claus que es presenta al capítol 3.5. Es tracta d'una operació ràpida però que farà que quedin afectats tots els bytes de l'State [6]. En la següent **Figura 5** continuem amb l'exemple anterior, en aquest cas el primer pas serà realitzar una XOR entre l'State d'entrada i la primera subclau k_0:

6B	2E	E9	73
C1	40	3D	93
BE	9F	7E	17
E2	96	11	2A

⊕

2B	28	AB	09
7E	AE	F7	CF
15	D2	15	4F
16	A6	88	3C

=

40	06	42	7ª
BF	EE	CA	5C
AB	4D	6B	58
F4	30	99	16

Figura 5. Mètode AddRoundKey (k_0).

3.2. SubBytes

La transformació SubBytes consisteix en aplicar a cada element de l'State (x) la següent funció en el camp de Galois (2^8) [7]:

$$f(x) = \begin{cases} a \cdot x^{-1} \oplus b & per\ x \neq 0 \\ b & per\ x = 0 \end{cases} \quad (4)$$

Sent $b = 0x63$ i a la matriu afí següent:

$$a = \begin{matrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{matrix} \quad (5)$$

Els càlculs de les multiplicacions i sumes es realitzen tal i com es veia al començament d'aquest apartat. Per una altra banda, $a \cdot x^{-1}$ equival a la inversa multiplicativa en el GF (2^8), aquesta operació consisteix en buscar un polinomi que multiplicat per a i mòdul $x^8+x^4+x^3+x+1$ sigui igual a 1.

Estem davant d'un mètode no lineal que té per objectiu atorgar la propietat de confusió a l'algoritme. És molt habitual la implementació d'aquest mètode amb una taula de cerca, en la que per cada byte d'entrada hi ha un byte de sortida corresponent. Aquesta taula de cerca s'acostuma a conèixer amb el nom de S-Box. A la següent **Figura 6** podem veure la S-Box[6]:

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 6.S-Box AES.

Aplicant la transformació al nostre exemple obtenim el següent resultat:

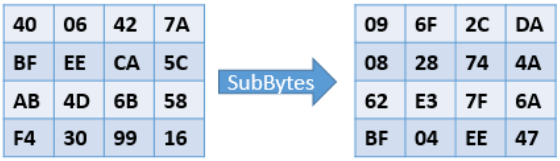


Figura 7. Mètode SubBytes de la ronda 1.

3.3. ShiftRows

Una vegada realitzat el pas anterior, ShiftRows serà el mètode a aplicar. Aquest mètode juntament amb el següent (MixColumns) proporcionaran la propietat de difusió a l'AES [6]. El funcionament del ShiftRows és molt senzill. La primera fila quedarà inalterada, la segona es desplaçarà 1 columna cap a l'esquerra, la tercera ho farà 2 columnes i la quarta, 3. En l'exemple proporcionat, el mètode ShiftRows donaria el següent resultat:

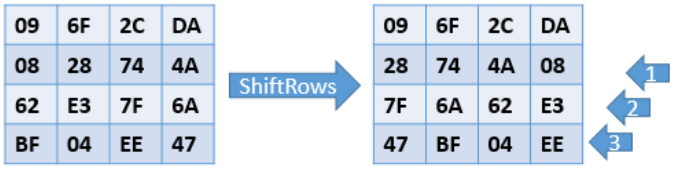


Figura 8. Mètode ShiftRows de la ronda 1.

3.4. MixColumns

El mètode MixColumns calcula cada columna individualment multiplicant-la per una matriu, en concret per cadascun dels valors de la columna es realitzarien els següents càlculs en el camp de Galois (2^8) :

$$\begin{cases} O_0 = 2I_0 + 3I_1 + I_2 + I_3 \\ O_1 = I_0 + 2I_1 + 3I_2 + I_3 \\ O_2 = I_0 + I_1 + 2I_2 + 3I_3 \\ O_3 = 3I_0 + I_1 + I_2 + 2I_3 \end{cases} \quad (6)$$

I_x i O_x es corresponen respectivament amb l'input i l'output del mètode MixColumns corresponent a una columna de l'State, i per cadascuna de les 4 files. Es pot comprovar que els càlculs en el GF(2^8) que s'explicaven a la introducció de l'AES són equivalents a les següents expressions [6]:

$$2I = \begin{cases} I \ll 1 \ \ quan\ el\ primer\ bit\ de\ I\ és\ igual\ a\ 0 \\ I \ll 1 \oplus 0x1B\ quan\ el\ primer\ bit\ de\ I\ és\ igual\ a\ 1 \end{cases} \quad (7)$$

$$3I = 2I \oplus I$$

Agafem la primera columna de la sortida de l'operació ShiftRows de l'exemple per veure com seria el càlcul:

09
28
7F
47

$$\begin{cases} O_0 = 2 \cdot 0x09 \oplus 3 \cdot 0x28 \oplus 0x7F \oplus 0x47 \\ O_1 = 0x09 \oplus 2 \cdot 0x28 \oplus 3 \cdot 0x7F \oplus 0x47 \\ O_2 = 0x09 \oplus 0x28 \oplus 2 \cdot 0x7F \oplus 3 \cdot 0x47 \\ O_3 = 3 \cdot 0x09 \oplus 0x28 \oplus 0x7F \oplus 2 \cdot 0x47 \end{cases} \quad (8)$$

El càlcul del valor de la primera fila serà el següent:

$$O_0 = 2 \cdot 0x09 \oplus 3 * 0x28 \oplus 0x7F \oplus 0x47 \quad (9)$$

En primer lloc la multiplicació de $2 \cdot 0x09$ al camp de Galois (2^8) es correspondrà amb un shift d'un bit cap a l'esquerra, ja que el primer bit és igual a 0:

$$0x09 = 00001001 << 1 \quad => \quad 00010010 = 0x12 \quad (10)$$

La multiplicació de $3 \cdot 0x28$ al camp de Galois (2^8) es correspondrà amb la mateixa operació que l'anterior (ja que el primer bit també és igual a 0) però fent la XOR del resultat amb $0x28$ ja que estem multiplicant per 3:

$$0x28 = 00101000 << 1 \quad => \ 01010000 = 0x50$$

$$0x50 \oplus 0x28 = 0x78 \quad (11)$$

Per arribar al primer output de la primera columna ja només ens queda fer les XOR corresponents:

$$O_1 = 0x12 \oplus 0x78 \oplus 0x7F \oplus 0x47 = 0x52 \quad (12)$$

Fent el mateix per totes les columnes i els seus valors corresponents arribaríem a l'output del mètode MixColumns. Per acabar la primera ronda del AES ja només restarà fer l'AddRoundKey amb la subclau de ronda corresponent, en aquest cas k_1. Podem veure el resultat final de la primera ronda de l'exemple executant la funció MixColumns en la nostra implementació, seguida de la funció AddRoundKey(k_1):

```
 State despres de mixColumns:
['52', '97', 'E0', 'BA']
['9F', '86', '1A', '1A']
['16', '15', 'AE', '26']
['C2', 'CA', '54', '59']

State despres d' addRoundKey
['F2', '1F', 'C3', '90']
['65', 'D2', 'B9', '76']
['E8', '39', '97', '50']
['D5', '7B', '6D', '5C']
```

Figura 9. Resultat de la primera ronda de l' AES.

El xifratge AES repetirà els mètodes vistos en els anteriors apartats seguint l'esquema de la **Figura 3** fins a arribar a l'State de sortida i per tant al missatge xifrat. A la **Figura 10** es pot veure el resultat final del missatge xifrat. Per arribar a aquests valors de sortida hem fet ús de la implementació pròpia en Python.

```
Missatge xifrat:
['3A', '0D', 'A8', '24']
['D7', '7A', '9E', '66']
['7B', '36', 'CA', 'EF']
['B4', '60', 'F3', '97']
```

Figura 10. Resultat del xifratge.

3.5. Derivació de claus.

A la **Figura 4** es pot observar que s'utilitzen 11 claus de ronda o subclaus diferents (k_n), sent k_0 la primera i la que es correspon amb la clau mestra de l'algoritme. La derivació d'aquestes subclaus també segueix un procés. En primer lloc s'agafa l'última columna de la clau de la ronda anterior i se li aplica una rotació (RotBytes), és a dir el byte que era el primer passarà a ser l'últim i la resta guanyarà una posició. A continuació se li aplica la mateixa S-Box que s'ha utilitzat en el mètode SubBytes de l'encriptació (veure **Figura 6**). Fent una XOR entre la primera columna de la clau de la ronda anterior (en aquest cas la clau k_0), la columna resultant del procés anterior i la columna de Rcon corresponent a la ronda trobarem la primera columna de la subclau$_1$ (k_1). En les següents figures es pot observar el procés per obtenir la primera columna de k_1 per l'exemple proporcionat a la **Figura 4**.

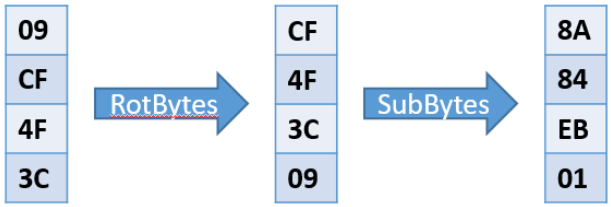


Figura 11. Derivació de la clau, pas 1.

Rcon

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
r1	r2	r3	r4	r5	r6	r7	r8	r9	r10

Figura 12. Matriu Rcon utilitzada per la derivació de la clau.

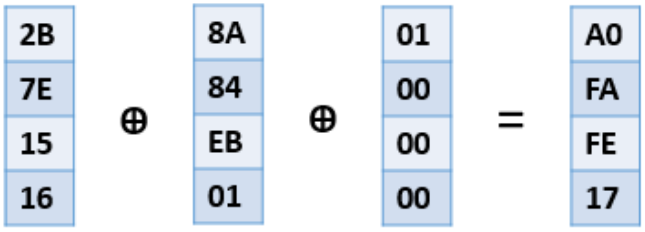


Figura 13.Obtenció de la primera columna de k_1.

Una vegada obtinguda la primera columna de k_1, la resta de columnes s'obtindran a partir del càlcul d'una XOR entre la columna corresponent de k_{i-1} (en aquest cas k_0) i l'última columna de k_1 calculada:

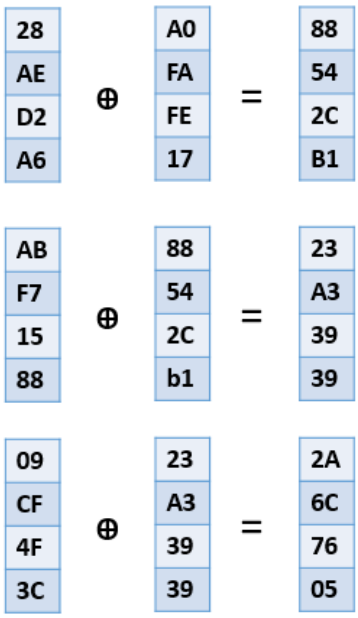


Figura 14. Obtenció de la resta de columnes de k_1.

Comprovem que si obtenim la subclau$_1$ amb el mètode *getRoundKey(1)* de la nostra implementació en Python arribem al resultat vist a l'exemple:

```
La subclau 1 (k1) es:
['A0', '88', '23', '2A']
['FA', '54', 'A3', '6C']
['FE', '2C', '39', '76']
['17', 'B1', '39', '05']
```

Figura 15. Implementació de la Subclau$_1$.

Seguint el mateix procés s'obtindran totes les subclaus que formen part del procés de xifratge. S'ha de tenir en compte que el procés presentat en aquestes línies és vàlid per AES-128. En el cas en que les claus fossin de 192 o 256 bits el funcionament seria similar però amb algunes diferències ja que el nombre de rondes seria diferent (12 i 14 respectivament). Per exemple en AES-256 les dues primeres claus de ronda (k_0 i k_1) venen determinades directament per la clau mestra de l'algoritme i el procés de derivació comença en la primera columna de k_2.

4. Diferents escenaris en la criptografia moderna

Com ja s'introduïa al primer apartat del present treball, fins fa pocs anys s'assumia que els algoritmes criptogràfics treballaven en entorns segurs en els que un atacant no podia accedir al funcionament intern de les operacions criptogràfiques. En els últims anys la perspectiva ha canviat degut a l'auge de les tecnologies mòbils i de les solucions software i es poden observar escenaris molt més insegurs a l'hora de mantenir la confidencialitat [3][8]. En aquest apartat es detallen els tres tipus d'escenaris criptogràfics basats en el poder amb el que compta l'atacant per aconseguir els seus objectius [9][10][11].

4.1. Escenari Black Box

Aquest és el model de seguretat tradicional. En aquest escenari s'assumeix que l'atacant no té accés físic a la clau ni a cap processament intern de l'algoritme de xifratge. L'única forma de realitzar un atac en aquestes condicions és a través de l'observació de l'input i l'output de l'algoritme o amb una cerca exhaustiva de la clau.

4.2. Escenari Gray Box

En aquest segon escenari l'atacant augmenta el seu poder al ser capaç d'extreure informació dels sistemes que realitzen les operacions de xifratge. L'atacant tampoc té accés a l'interior del xifratge però pot extreure dades que generen els sistemes al realitzar les operacions criptogràfiques i que el poden ajudar a esbrinar la clau. Aquesta informació pot ser el temps d'operació, el consum de potència, o la radiació electromagnètica. Aquest tipus d'informació és la base dels atacs de Side-Channel que seran explicats en detall a l'apartat d'atacs (veure capítol 6.1).

4.3. Escenari White Box

Per últim, l'escenari de White Box és el més insegur a l'hora de xifrar o desxifrar un missatge ja que l'atacant té un poder total sobre el sistema de manera que pot accedir a les direccions de memòria o pot alterar l'execució del programa. Aquest escenari suposa un nou repte per la criptografia i en els últims anys han sorgit algunes estratègies per tal d'adaptar els algoritmes criptogràfics existents a aquest nou model de seguretat.

5. Estratègies per la implantació d'algoritmes criptogràfics a la White Box

Com ja s'ha observat en l'apartat anterior les condicions de seguretat en la WB fan que les operacions criptogràfiques siguin molt més vulnerables. Per tant si es vol adaptar un algoritme criptogràfic conegut com l'AES a la WBC s'hauran d'implantar algunes estratègies per tal d'intentar dificultar l'accés a la informació confidencial.

L'estratègia ideal per aconseguir que un atacant no pogués accedir de cap forma a la clau seria implantar l'algoritme com una enorme taula de cerca en la que per cada valor d'entrada hi hagués un valor de sortida, de manera que la implementació es comportés com una Black-Box ideal. Aquesta implementació malauradament no és possible a la pràctica, ja que faria que l'algoritme necessités un espai extraordinàriament gran per tal de que la taula de cerca contingués tots els possibles valors. Com a exemple, una implementació AES-128 d'aquesta mena requeriria un espai al disc d'uns $5 \cdot 10^{27}$ terabytes [12].

Encara que l'aplicació d'una taula de cerca única és inviable, això no vol dir que la seguretat d'un algoritme no pugui ser millorada per afrontar un escenari WB. En aquest apartat es troben les estratègies utilitzades fins ara per aconseguir implementacions pràctiques a la WB d'algoritmes existents, amb una explicació del seu funcionament en AES i els seus objectius [1][2][3][13][14][15][16][17][18][19].

En aquest cas utilitzarem un esquema d'AES equivalent al que vèiem en la **Figura 4** amb petites modificacions, d'aquesta manera es col·loca l'AddRoundKey(k_0) dintre de les rondes normals i AddRoundKey (k_9) passa a la ronda final (veure **Figura 16**).

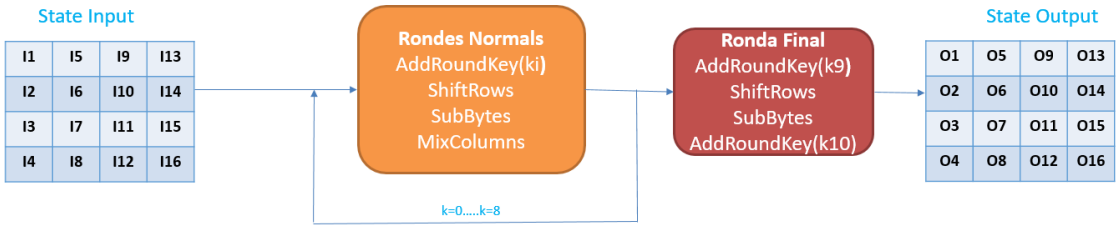


Figura 16. Esquema AES modificat.

5.1. Avaluació parcial

Aquesta primera estratègia busca barrejar l'efecte de la clau dintre de l'algoritme fent una precomputació d'aquest. D'aquesta manera la clau no està disponible explícitament al sistema i per tant es dificulta la seva obtenció per part d'un atacant.

En el cas de l'AES aquesta estratègia consistirà en barrejar l'efecte del mètode AddRoundKey amb el SubBytes. Amb la precomputació tindríem una sèrie de taules S-Box en la que ja estaria inclòs l'efecte de la clau de ronda i per tant durant el xifratge del missatge en cap moment es faria una operació amb la clau. Com a contrapartida, en aquest mètode, com que cada subclau de l'algoritme és diferent, ara ja no tindrem una única S-Box per tot l'AES sinó moltes més per cada ronda i cada byte. S'ha de tenir en compte que en l'última ronda s'inclouran les dues últimes subclaus (k_9 i k_{10}). Aquestes noves taules de cerca són conegudes com T-Boxes i per la implementació de l'AES es necessiten 160 T-Boxes diferents, que es corresponen a 16 T-Boxes per cadascuna de les rondes, una per cadascun dels bytes de l'State (una T-Box es comporta igual que les S-Box que ja vam veure en el capítol 3.2, és a dir amb un byte d'entrada i un byte de sortida).

Si seguim amb l'esquema plantejat a la **Figura 16** sorgeix el problema de que l'operació ShiftRows es troba entre les dues operacions que es poden fusionar a les T-Box. L'esquema pot ser novament modificat tenint en compte que ShiftRows es pot posar davant del mètode AddRoundKey sempre que la clau també passi pel mètode ShiftRows, quedant com a resultat l'esquema següent:

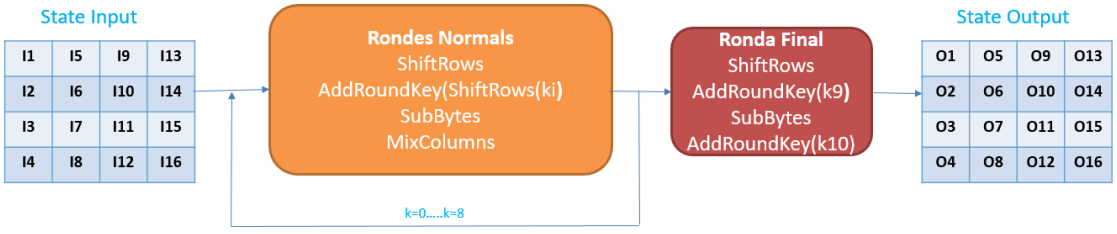


Figura 17.Esquema AES modificat canviant la posició de l'operació ShiftRows.

Seguint amb aquest exemple les T-Box contindrien el resultat de realitzar les operacions AddRoundKey(ShiftRows(k_i)) i SubBytes de manera que la T-Box tampoc tindrà una relació directa amb la clau, sinó que la precomputació de les T-Box es realitzarà amb les

subclaus ja passades pel mètode ShiftRows. Una vegada aplicades les T-Box ens quedaria el següent esquema de l'algoritme:

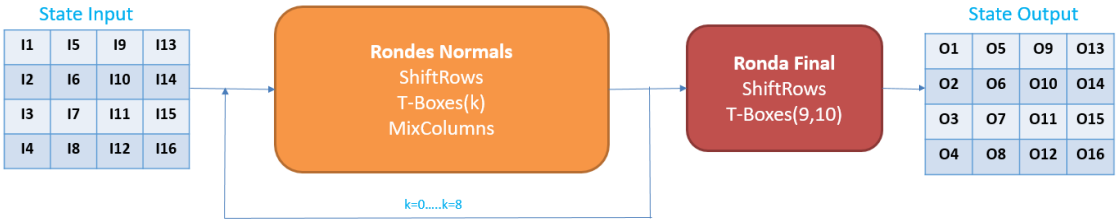
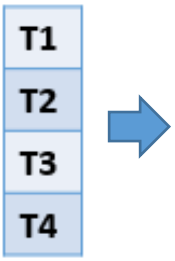


Figura 18. Esquema WB AES utilitzant la solució T-Box.

5.2. Implantació de taules de cerca

A la introducció d'aquest apartat ja es parlava de que la manera ideal d'implementar un algoritme a la WB seria una gran taula de cerca que substituís totes les operacions, però com ja hem vist, això a la pràctica no és viable. No obstant, sí es poden substituir les operacions per cadenes de petites taules de cerca que aportin confusió però que siguin implementables a la pràctica.

En el cas de l'AES si seguim amb l'estratègia que vèiem a l'apartat anterior, encara es pot aportar més confusió transformant el mètode MixColumns en sèries de taules de cerca. Si recordem el que vam veure al capítol 3.4, el mètode MixColumns realitzava els càlculs amb columnes. En aquest cas, si agafem com exemple la primera columna de la primera ronda, aquesta seria igual als 4 bytes de sortida de les quatre primeres T-Boxes (recordem que teníem 16 T-Boxes per cadascuna de les rondes):



$$\begin{cases} O_0 = 2 \cdot T1 \oplus 3 \cdot T2 \oplus T3 \oplus T4 \\ O_1 = T1 \oplus 2 \cdot T2 \oplus 3 \cdot T3 \oplus T4 \\ O_2 = T1 \oplus T2 \oplus 2 \cdot T3 \oplus 3 \cdot T4 \\ O_3 = 3 \cdot T1 \oplus T2 \oplus T3 \oplus 2 \cdot T4 \end{cases} \quad (13)$$

D'aquí podem treure 4 taules de cerca denominades Ty-Boxes que transformen 1 byte d'entrada en 4 de sortida seguint el raonament següent:

$$\begin{cases} Ty1 = T1 \cdot [02\ 01\ 01\ 03] \\ Ty2 = T2 \cdot [03\ 02\ 01\ 01] \\ Ty3 = T3 \cdot [01\ 03\ 02\ 01] \\ Ty4 = T4 \cdot [01\ 01\ 03\ 02] \end{cases} \quad (14)$$

El resultat del mètode MixColumns seria la XOR dels valor de les Ty-Boxes anteriors:

$$Ty1 \oplus Ty2 \oplus Ty3 \oplus Ty4 \quad (15)$$

La XOR també la podem construir com una taula de cerca per evitar els càlculs, una manera de fer-ho seria realitzar una X0R-Box que tingués com entrada una parella de valors de 4 bits i com a sortida el resultat de la seva XOR. Per cadascuna de les columnes necessitarem consultar la XOR-Box 24 vegades i per tant necessitarem 96 consultes per ronda (864 en total).

L'esquema de l'algoritme AES seguint aquesta estratègia seria el següent:

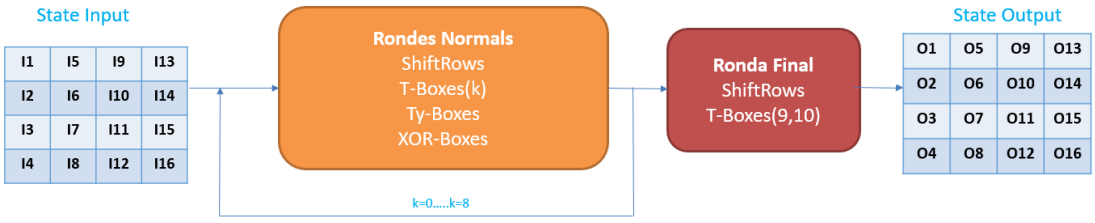


Figura 19.Esquema White Box AES incloent la solució Ty-Box.

Per fer que hi hagi menys accessos a taules de cerca es podria fer una composició de les T-Boxes amb les Ty-Boxes. Ja que els outputs de les T-Boxes entren directament a les Ty-Boxes.

5.3. Codificacions internes

Aplicant totes les estratègies anteriors l'algoritme encara estaria davant el perill de que l'atacant pogués recuperar la clau a partir de l'anàlisi de les taules de cerca [17]. Ja que només hi haurà 256 possibilitats diferents per la composició d'una T-Box i una Ty-Box segons el valor del byte de la clau en qüestió (aquest atac seria aplicable a la ronda 1).

Per evitar això es pot utilitzar una estratègia denominada codificacions internes. Aquesta estratègia tracta d'aplicar un xifratge als valors d'una taula de cerca per tal d'ofuscar-los. Aquesta operació ha de ser reversible. Per poder aplicar aquestes codificacions i que el comportament de l'algoritme no es vegi afectat es poden utilitzar codificacions i descodificacions en taules de cerca consecutives de la següent manera:

$$T' = g\left(f(T)\right)^{-1}$$
$$U' = f\left(g(U)\right)^{-1} \quad (16)$$

On T i U son taules consecutives i $f(x)$ i $g(x)$ són codificacions internes.

D'aquesta manera s'ofuscarien els valors de les taules T i U sense modificar el comportament de l'algoritme. Ja que en cadascuna de les taules s'aplicarien les codificacions internes de manera inversa a la taula veïna.

5.4. Aplicar transformacions lineals invertibles

Amb la incorporació de les codificacions internes s'aconsegueix que cadascuna de les taules que incorporen part de les subclaus de ronda al xifratge tinguin confusió. La següent estratègia serveix per aconseguir que cadascuna de les taules també obtingui la difusió com a propietat. L'estratègia consisteix en la multiplicació de les taules per una matriu de bijecció singular. A l'hora d'escollir aquestes matrius s'ha d'assegurar que siguin reversibles per poder garantir el funcionament de l'algoritme. Les transformacions s'introduirien abans de l'aplicació de les codificacions internes per tal de propagar aquestes codificacions per tota la taula [15].

5.5. Codificacions externes

Si s'implementa un AES a la WB amb totes les estratègies anteriors, no obstant, l'entrada i la sortida de l'algoritme segueixen estant desprotegides. De manera similar a les codificacions internes es pot codificar l'entrada i la sortida de l'algoritme. D'aquesta manera s'aconsegueix que en cap moment el missatge a protegir estigui en text en clar. En aquest cas es necessitaria una matriu de 128x128 bits però que també es pot descompondre en matrius més petites tal com es veia a l'apartat d'implantació de taules de cerca. S'utilitza aquest sistema de taules de cerca per evitar un masking senzill que pugui ser fàcilment esbrinable.

6. Tipus d'atacs

Arribat a aquest punt del present treball ja s'ha pogut comprovar com funciona el xifratge en l'algoritme AES i com es pot adaptar aquest algoritme a un entorn WB amb les tècniques de l'apartat anterior. En aquest apartat s'introdueixen els tipus d'atacs més freqüents en la criptografia actual. Les dues famílies d'atac descrites en aquest capítol provenen de l'entorn hardware. En aquest projecte es demostra la metodologia i viabilitat de la seva aplicació en un entorn software WB. D'aquesta manera es pot veure què és i com funciona un atac de falta per tal de poder aplicar-ho després a l'algoritme AES i posteriorment a una implementació d'aquest algoritme a la WB (objectiu principal del treball).

6.1. Atacs de Side-Channel

Al capítol 4.2 quan es defineix l'escenari Gray Box ja es parla dels atacs de Side-Channel. Aquests atacs es basen en l'estudi de la informació involuntària que retorna un hardware quan realitza operacions. Les informacions més utilitzades per aquest tipus d'atacs són el temps d'execució, l'energia consumida i la radiació electromagnètica generada. Si la informació involuntària generada per un hardware que executa operacions criptogràfiques correla amb la clau secreta que utilitza és possible esbrinar aquesta clau obtenint un nombre suficient de traces. La paraula traça fa referència a la senyal obtinguda per cada execució.

Per saber l'instant de temps en que es realitza la instrucció que fa l'operació que interessa a l'atacant, les tècniques més utilitzades són el SPA (Simple Power Analysis) i el SEMA (Simple Electromagnetic Analysis) [20]. Aquestes tècniques consisteixen en la identificació d'una operació a través de l'observació d'una o poques traces de potència (en el cas de SPA) o de radiació electromagnètica (en el cas de SEMA). Per exemple un hardware que executa una codificació amb l'algoritme AES podria mostrar les 10 rondes del xifratge com a 10 patrons que es repeteixen seqüencialment en la traça de consum de potència (o radiació electromagnètica) i d'aquesta manera un atacant podria saber en quin moment s'està realitzant una operació concreta.

L'analogia de les tècniques de SPA en l'entorn WB serà molt útil a l'hora d'identificar les instruccions utilitzades pel codi de l'algoritme. Això es veurà en detall en el capítol 8.1.

El DPA (Differential Power Analysis) i el DEMA (Differential Electromagnetic Analysis) són atacs que realitzen una anàlisi estadística comparant dos grups de traces. Cada grup de traces es correspon amb una mateixa operació criptogràfica però treballant amb dues variables diferents [20][21]. Per exemple, es podria aplicar aquest atac a la sortida de la operació SubBytes de l'AES. Les anàlisis DPA i DEMA necessiten computar l'estadística d'un gran nombre de traces (centenars de milers típicament) ja que en dispositius hardware la informació útil acostuma a estar emmascarada pel soroll. Aquest és un handicap que quan treballem en un entorn WBC purament software, com veurem més endavant, no tindrem.

També existeixen atacs que partint del DPA o DEMA fan ús del coeficient de correlació de Pearson per analitzar la correlació entre les traces i les dades assumint un cert model de potència [20].

Aquest treball no aprofundirà en el funcionament dels atacs de Side-Channel ja que queden fora de l'abast dels objectius establerts. No obstant, alguna d'aquestes tècniques serà útil a l'hora de trobar el moment en el que cal injectar la falta en l'algoritme WB a atacar.

6.2. Atacs d'injecció de falta

El primer document acadèmic en el qual s'introdueixen els atacs d'injecció de falta per obtenir informació confidencial és del 2001 [22]. Aquests tipus d'atacs es basen en intervenir el hardware durant el seu funcionament normal. Aquestes intervencions puntuals són denominades faltes i es poden realitzar de diverses formes. Els mètodes més típics d'injecció de falta són pics de tensió, pujades de freqüència, polsos electromagnètics i pertorbacions amb l'ajuda d'un làser [23].

Un atac d'injecció de falta pot tenir com a objectiu saltar-se una certa instrucció del codi que s'està executant o canviar el valor d'una certa variable. Per exemple, si a un dispositiu hardware dissenyat per treballar a una freqüència determinada de cop i volta se li augmenta molt la freqüència de treball durant alguns cicles de rellotge, molt probablement es crearà una pertorbació en l'execució normal del codi fent que el programa que s'està executant en aquell hardware es salti instruccions. Alternativament, il·luminar els registres de la CPU amb llum ultraviolada és una manera d'alterar el seu estat i, en conseqüència, el valor que hi guarden.

Existeixen atacs d'injecció de falta per poder vulnerar la seguretat dels algoritmes criptogràfics més utilitzats, l'AES no és una excepció i s'han trobat atacs d'injecció de falta que poden aconseguir recuperar la clau utilitzada [7][23][24][25]. En el següent apartat podem veure com funciona un d'aquests atacs, concretament el que utilitzarem per demostrar la viabilitat de realitzar aquest tipus d'atacs que provenen del món del hardware en un entorn software WB.

7. Atac de falta contra l'AES

En aquest apartat s'explica amb detall com funciona l'atac de falta escollit contra l'AES. En concret, s'ha optat per un atac que consisteix en inserir una falta en el resultat del mètode ShiftRows de la ronda 9 [7][24]. Per realitzar aquest atac és necessari poder accedir al resultat final del desenvolupament normal de l'algoritme (xifratge correcte) i al resultat del xifratge quan s'aconsegueix inserir una falta. S'ha escollit aquest atac per ser un dels més efectius que es pot desenvolupar contra un AES, ja que l'objectiu és analitzar si es pot traslladar un atac de falta a l'entorn WB. S'ha utilitzat com a base la implementació pròpia de l'algoritme en Python que ja havia servit per realitzar l'explicació del funcionament de l'algoritme en el capítol 3. La implementació d'aquest atac ha servit per comprendre com realitzar-ho i poder traslladar-ho a una implementació White Box en futurs apartats.

7.1. Funcionament de l'atac escollit

Com ja s'ha introduït, l'objectiu és inserir una falta a la sortida del mètode ShiftRows de la ronda 9, que és la última ronda en la que es realitzen tots els mètodes de l'algoritme. Vegem en el següent exemple el que implica la introducció d'aquesta falta. En cadascuna de les etapes es pot observar com canvia l'State en el cas d'haver-hi una falta.

En les següents taules podem veure el State després del mètode ShiftRows de la novena ronda. La primera matriu es correspon amb el desenvolupament normal de l'execució, mentre que en la segona hi ha hagut una falta en el primer byte, i el valor 0x83 s'ha convertit en 0x5B.

83	FF	C1	77
33	15	91	0E
F0	A6	B4	81
AF	ED	09	5E

5B	FF	C1	77
33	15	91	0E
F0	A6	B4	81
AF	ED	09	5E

Figura 20. Inserció de la falta després de l'operació ShiftRows.

Si recordem l'explicació de com funciona el mètode MixColumns que es detalla al capítol 3.4, el valor corresponent a la primera fila i primera columna intervé en el càlcul de tota la primera columna en l'State de sortida del mètode. Per tant després de l'operació

MixColumns tota la primera columna passa a estar infectada com a conseqüència de l'error introduït:

17	91	8C	23
41	C9	36	AD
A1	91	38	82
18	68	6F	AA

BC	91	8C	23
99	C9	36	AD
79	91	38	82
6B	68	6F	AA

Figura 21. Propagació de la falta després de l'operació MixColumns.

El següent pas és afegir amb una XOR la clau de ronda corresponent (en aquest cas k_9). Com es tracta d'una XOR, després d'aquest pas continuaran infectats els mateixos bytes que en el mètode anterior:

BB	88	A4	7A
36	33	E7	F1
C7	40	11	82
EB	49	2E	C4

10	88	A4	7A
EE	33	E7	F1
1F	40	11	82
98	49	2E	C4

Figura 22. Propagació de la falta després de l'operació AddRoundKey.

Una vegada arribats a aquest punt comença l'última ronda de l'algoritme que només executarà els mètodes SubBytes, ShiftRows i AddRoundKey. En començar aquesta última ronda, el mètode SubBytes a través de les S-Box fa que cadascun dels bytes canviï per un altre. Queda clar que després d'aquesta operació la infecció segueix controlada a la primera columna:

EA	C4	49	92
05	C3	94	A1
C6	E3	82	13
E9	3B	31	1C

CA	C4	49	92
28	C3	94	A1
C0	E3	82	13
46	3B	31	1C

Figura 23. Propagació de la falta després de l'operació SubBytes.

Seguidament, el mètode ShiftRows degut al seu funcionament farà que els bytes infectats canviïn de posicions. Seguiran sent quatre bytes però en aquest cas estaran dispersats per la resta de columnes:

EA	C4	49	92
C3	94	A1	05
82	13	C6	E3
1C	E9	3B	31

CA	C4	49	92
C3	94	A1	28
82	13	C0	E3
1C	46	3B	31

Figura 24. Propagació de la falta després del mètode ShiftRows.

Després d'aquest pas ja només queda afegir la clau de ronda 10 per arribar al missatge xifrat, a continuació vegem la diferència entre el missatge xifrat amb falta i sense falta:

3A	0D	A8	24
D7	7A	9E	66
7B	36	CA	EF
BA	60	F3	97

1A	0D	A8	24
D7	7A	9E	4B
7B	36	CC	EF
BA	CF	F3	97

Figura 25. Comparació entre el missatge xifrat amb falta i sense falta.

S'ha de tenir en compte que qualsevol falta en la primera columna hagués comportat una variació en el text xifrat dels mateixos bytes, ja que després de l'operació MixColumns de la ronda 9 els bytes afectats serien igualment els pertanyents a la primera columna.

Si restem al missatge xifrat el resultat del xifratge amb falta obtenim la matriu d'error següent (recordem que el resultat d'una resta és equivalent a una XOR):

20	00	00	00
00	00	00	2D
00	00	06	00
00	AF	00	00

Figura 26. Matriu d'error.

Les faltes introduïdes en la primera columna ens serviran per calcular els valors de la clau k_{10} corresponents a les posicions que s'han vist afectades per l'error en la computació (bytes 1,8,11,14).

Del funcionament dels mètodes que formen part de l'AES es desprenen les següents equacions:

$$\begin{aligned} SubBytes(b(1) \oplus 2Ein(1)) &= SubBytes(b(1)) \oplus Eout(1) \\ SubBytes(b(2) \oplus Ein(2)) &= SubBytes(b(2)) \oplus Eout(14) \\ SubBytes(b(3) \oplus Ein(3)) &= SubBytes(b(3)) \oplus Eout(11) \\ SubBytes(b(4) \oplus 3Ein(4)) &= SubBytesB(b(4)) \oplus Eout(8) \end{aligned} \quad (17)$$

On Ein es correspon amb els errors introduïts al byte corresponent i Eout es correspon amb la diferència en el byte corresponent entre l'output correcte i l'output amb falta. Els valors de b són els valors del bytes de l'State a l'entrada de l'operació SubBytes de la ronda 10 quan no hi ha hagut falta. Les posicions de cada variable es corresponen amb les vistes anteriorment (per exemple 14 és el valor de la segona fila i quarta columna).

Per tant les equacions anteriors es formen simplement igualant el còmput de l'operació SubBytes de la ronda 10 començant des de la falta introduïda al mètode ShiftRows de la ronda 9 (part esquerra de l'equació) i posteriorment tirant enrere l'algoritme des de l'output (part dreta de l'operació).

En aquestes equacions comptem amb dues incògnites. L'error introduït que no sabem de quan serà i els valors dels bytes b. El que sí que tenim és l'error de sortida que ja podíem veure en la matriu d'error introduïda anteriorment com a diferència entre el xifrat correcte i el xifrat incorrecte.

L'atac consisteix en buscar tots els valors possibles de la falta (Ein) i els bytes b que compleixen les equacions. Un cop obtinguts, el valor correcte de la subclau k_{10} complirà la següent equació (on x és el número del byte corresponent):

$$k(x)_{10} = ShiftRows\big(SubBytes(b(x))\big) \oplus Output(x) \quad (18)$$

Com tindrem diversos valors que compliran les equacions, també obtindrem diversos valors possibles per $k(x)_{10}$. Si es realitzen diverses faltes consecutives en el mateix byte, el nombre de candidats a ser la subclau correcta k_{10} serà cada vegada menor.

7.2. Programació de l'atac de falta en la implementació pròpia en Python

Una vegada estudiat el funcionament de l'atac en qüestió, s'ha realitzat la seva implementació en l'algoritme programat en Python que ja s'havia introduït al capítol 3.

A les següents captures de pantalla es pot veure com la implementació pròpia de l'atac va reduint els candidats a ser el valor correcte d'un byte de la subclau 10 a mesura que es van realitzant faltes en el primer byte de l'State de sortida del mètode ShiftRows de la ronda 9. El programa introdueix una falta aleatòria en el byte corresponent i va trobant els bytes candidats a formar part de la subclau 10 a través de les equacions que s'han vist al subapartat anterior. En concret els bytes candidats que s'observen en les 4 files de les següents imatges es corresponen amb els bytes 1, 14, 11 i 8 respectivament.

El procediment consisteix a descartar tots aquells candidats de k_{10} que no es repeteixin com a resultat de les diverses faltes (ja que independentment de la falta que es provoca el valor de la subclau sempre ha de ser el mateix).

```
Candidats per subclau 10
['FA', '88', '63', 'FE', '67', '5D', '3F', '01', '05', '9C', 'A2', 'A6', '3B', 'C0', 'C4', 'EE', 'EA', '77', '98', '2F', '73', '2B', '4D', '11', '15', '49', 'B6', 'B2', 'D4', 'D0']
['BD', 'F8', 'DB', '8B', '5E', '63', '68', '80', '26', '7D', '95', '9E', '40', 'A3', 'A8', 'E6', 'C5', 'ED', 'CE', '2D', '76', '38', '0E', '4B', '10', 'F3', '55', '33', '1B', 'D0']
['BA', '60', '62', 'FC', 'FE', '1F', '83', '81', '5B', '24', '26', '48', 'C7', 'C5', 'ED', 'EF', '0C', '73', '71', '0E', '90', '92', '4A', '59', '37', '35', 'D6', 'D4', 'B8', '1D']
['70', 'FA', '62', '89', 'FF', '67', '81', '84', '7D', '06', '9B', '9E', 'E8', 'E5', '78', 'E0', 'ED', '0B', '75', '0E', '93', '96', '11', 'F2', '14', '6F', 'F7', '19', '8C', '1C']
```

Figura 27. Candidats a k_{10} després de primera falta.

```
Candidats per subclau 10
['01', '15', '9C', '88', '4D', 'D0', 'C4']
['5E', '63', '38']
['24', '0C', '81']
['62', '89']
```

Figura 28. Candidats a k_{10} després de segona falta.

```
Candidats per subclau 10
['4D', 'D0', 'C4']
['63']
['0C']
['89']
```

Figura 29. Candidats a k_{10} després de tercera falta.

```
Candidats per subclau 10
['D0']
['63']
['0C']
['89']
```

Figura 30. Candidats a k_{10} després de quarta falta.

Com es pot observar ha sigut suficient amb quatre faltes per trobar els bytes corresponents de k_{10}. Per esbrinar la resta de bytes de la subclau 10 només faria falta fer el mateix procés introduint faltes a la resta de columnes de la matriu State a la sortida de l'operació ShiftRows de la ronda 9. En aquest cas al tractar-se d'una implementació pròpia la dificultat d'inserir faltes és mínima i es pot controlar el byte que es vol afectar. Si el byte afectat no es pogués controlar s'augmentaria el nombre de faltes necessàries per poder realitzar l'atac amb èxit. S'ha de tenir en compte també que si en comptes d'afectar al primer byte afectem el segon les equacions a complir estaran en un altre ordre, ja que el mètode MixColumns afectaria d'una manera diferent, però l'efectivitat de l'atac restaria igual.

Una vegada s'ha arribat al valor correcte de k_{10} l'únic que queda és aconseguir la clau mestra a partir de la subclau 10. Per això és necessari seguir el següent procés:

$$\begin{gathered}
Valors\ dels\ bytes\ de\ la\ primera\ columna: \\
k_{i-1}(1) = k_i(1) \oplus k_i(10) \oplus k_i(14) \oplus RCON_i \\
k_{i-1}(2) = k_i(2) \oplus k_i(11) \oplus k_i(15) \oplus RCON_i \\
k_{i-1}(3) = k_i(3) \oplus k_i(12) \oplus k_i(16) \oplus RCON_i \\
k_{i-1}(4) = k_i(4) \oplus k_i(9) \oplus k_i(13) \oplus RCON_i \\
Valors\ de\ la\ resta\ de\ bytes: \\
k_{i-1}(x) = k_i(x) \oplus k_i(x-4)
\end{gathered} \tag{19}$$

Aquest procés equival a fer el procés invers al que es feia en la derivació de clau que s'explica a l'apartat 3.5 .

A la implementació pròpia que s'ha fet de l'atac es pot veure com es retorna la clau correcta de l'algoritme a partir dels candidats obtinguts:

```
Subclau 10 formada a partir dels candidats:
['D0', 'C9', 'E1', 'B6']
['14', 'EE', '3F', '63']
['F9', '25', '0C', '0C']
['A8', '89', 'C8', 'A6']

Clau:
2B7E151628AED2A6ABF7158809CF4F3C
Execucio finalitzada. Premi intro per sortir
```

Figura 31. Clau obtinguda a partir de l'atac de falta

Una vegada s'ha estudiat en profunditat l'atac, s'arriba a la suposició de que aquest atac és exportable a un entorn WB ja que, encara que es puguin utilitzar T-Boxes o Ty-Boxes per tal d'amagar la clau, el valor de sortida d'una T-Box seria el mateix que el d'aplicar els mètodes corresponents i les dades segueixen sent tractades byte a byte. En el següent apartat es tractarà de confirmar aquesta hipòtesi fent front a un cas real.

8. Atac sobre la implementació White Box AES escollida

En aquest punt es podrà observar com traslladar l'atac de falta estudiat a l'apartat anterior a un entorn White Box en el que es té control total sobre el sistema. En aquest cas s'ha optat per realitzar l'atac contra una implementació AES senzilla en ARM que utilitzem al laboratori per testar. Aquest punt es centrarà en la metodologia emprada per aconseguir atacar la implementació en ARM.

L'atac s'ha desenvolupat sobre un dispositiu mòbil amb un sistema operatiu Android i amb permisos root. S'ha de tenir en compte que tenir permisos root és un requisit indispensable per poder tenir control total del dispositiu i per dur a terme l'atac. Recordem que estem davant d'un entorn WB i per tant tenir un control total del dispositiu és una precondició. No obstant, avui dia no és gens desgavellat considerar que un dispositiu mòbil es pot rootejar. El procediment descrit a continuació s'ha realitzat amb l'eina de debugging GDB (veure annex 2). Bàsicament un procés de debugging consisteix en poder monitoritzar l'execució d'un programa pas a pas per identificar algun error en la seva execució o per modificar el seu comportament d'alguna forma, en aquest cas en l'atac que es proposa es modificaran certs valors de registres del processador en un moment donat.

L'objectiu principal per desenvolupar l'atac amb èxit serà trobar el moment de l'execució de l'algoritme en el que s'ha d'inserir la falta (equivalent a la sortida del ShiftRows de la ronda 9 de l'apartat anterior) i trobar quin és el registre o registres del processador que s'han de modificar per aconseguir la falta desitjada [26][27]. Cal recordar que les faltes necessàries per obtenir la clau de ronda 10 són les que resulten en un xifratge de sortida que es diferencii en 4 bytes del xifratge de sortida correcte. A més, s'ha de tenir en compte que la clau d'una implementació AES sol estar inserida en un altre mètode, tal com vèiem al capítol 5.1 del present treball. Si no fos així la clau es podria obtenir directament amb tècniques de debugging.

8.1. Anàlisi de l'execució de l'algoritme

La metodologia per trobar el moment en el que inserir la falta es desenvolupa a través de l'estudi de l'evolució del Program Counter (PC) durant l'execució de l'aplicació [28] [29]. El PC és un registre que conté l'adreça de memòria de la següent instrucció que

serà executada, per exemple quan un programa conté un bucle el valor del PC va repetint-se a mesura que s'executen de nou les mateixes instruccions.

Es realitza la suposició de que durant l'execució de l'algoritme es podrà identificar un bucle ja que l'AES disposa de 10 rondes i té operacions que es repeteixen en cadascuna d'aquestes. Aquest raonament seria similar al que es realitza quan es realitza un SPA o SEMA dels que es parlaven en l'apartat 6.1.

En una primera execució hem vist que el final de l'aplicació el PC té un valor de 0x16504. Per obtenir amb GDB l'evolució del PC durant tota l'execució fem ús del següent codi, la comanda *set args* controla el valor d'entrada que es xifrarà, per que el GDB reconegui aquest codi l'hem de cridar a partir de la comanda *source*:

**No publicat per raons de confidencialitat.*

Figura 32. Script GDB per analitzar l'execució de l'algoritme.

Si es representen gràficament els valors obtinguts, sent l'eix X el nombre d'instruccions i l'eix Y el valor del Program Counter, es pot arribar a la representació de la **Figura 33**:

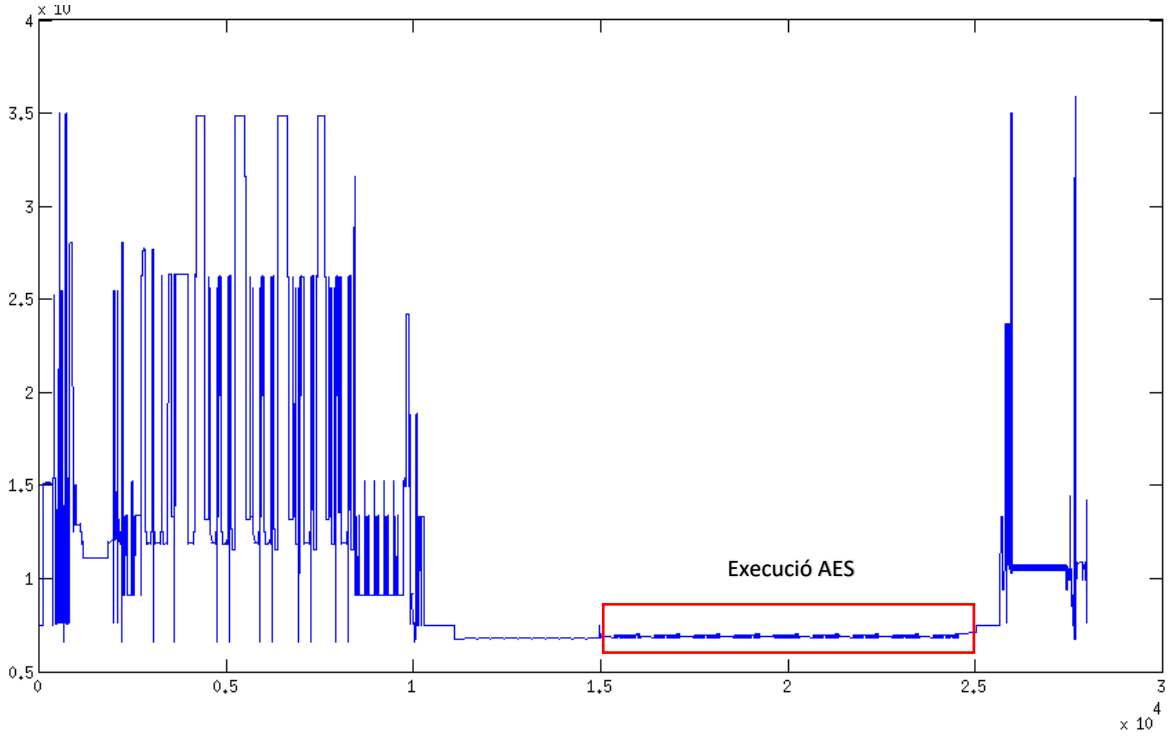


Figura 33. Evolució del Program Counter durant l'execució de la implementació AES.

Si s'observa la imatge amb deteniment es pot apreciar amb claredat com el registre PC pren valors molt alts al principi i final de l'execució, i valors més baixos al centre de l'execució. Podem relacionar els valors alts amb crides a funcions de sistema. Per tant

farem “zoom” a la regió mitja de l’execució on s’intueix que s’està codificant l’AES. En aquesta regió s’identifica un bucle que es repeteix 9 vegades. Aquest fet encaixa plenament amb l’execució de l’AES ja que l’última ronda de l’algoritme ha de ser diferent a la resta al no executar-se el mètode MixColumns. En la **Figura 34** es pot veure el bucle de 9 rondes al que es fa referència:

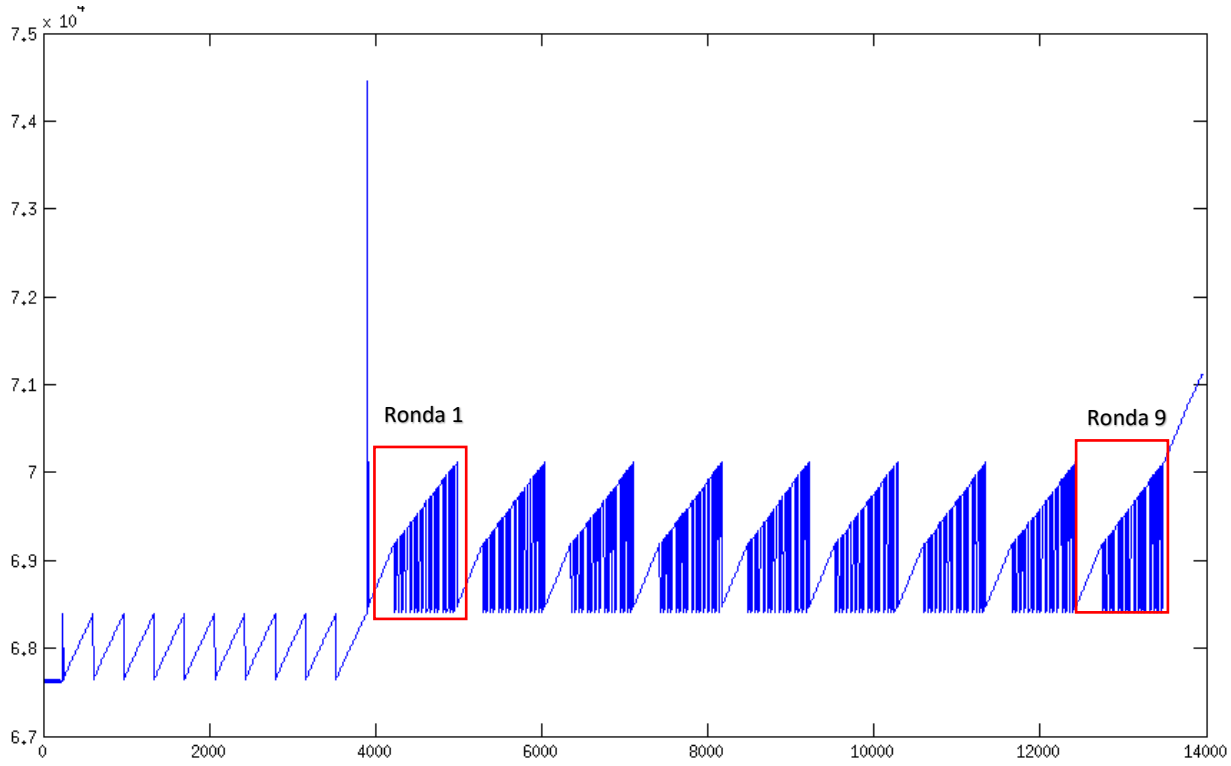


Figura 34. Identificació de les rondes de l'AES a través del Program Counter.

Encara es pot anar més lluny en aquesta primera anàlisi i tractar de comprendre què és el que passa en cadascuna de les rondes. En la **Figura 35** es pot veure un gràfic del que s’interpreta com una ronda de l’AES. Es pot distingir que a la part baixa de la figura hi ha uns valors de Program Counter que es repeteixen 16 vegades, aquest valor podria fer referència al mètode MixColumns ja que aquest mètode seria igual en cadascuna de les rondes (i com s’observava en la **Figura 34** després de la ronda 9 no torna a haver-hi una baixada cap a aquests valors de Program Counter). Les 16 vegades que l’execució visita cadascun d’aquests valors es poden justificar perquè l’operació es realitza byte a byte i l’State està formada per aquesta quantitat de bytes. Seguint amb aquesta suposició es

pot arribar a la conclusió que els valors superiors de la **Figura 35** es corresponen amb la resta de les operacions i que en aquest cas són diferents en cada byte perquè els mètodes tenen inclosos els diferents valors de la clau (veure T-Boxes del capítol 5.2)

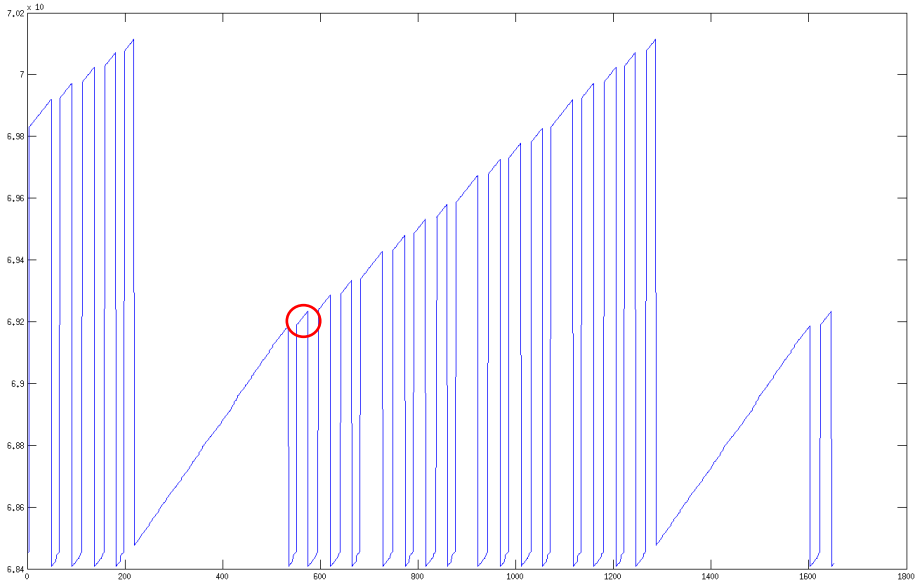


Figura 35. Representació d'una ronda de l'AES a través del Program Counter.

8.2. Identificació dels registres d'interès

L'objectiu principal del projecte és poder inserir una falta en el primer byte dintre de l'execució de la ronda 9 i aconseguir que aquesta falta es propagui per 4 bytes de la sortida xifrada de l'algoritme tal com passava a l'atac que es detallava al capítol 7. Vist l'anàlisi anterior ja es pot veure com saber quins són els valors de Program Counter interessants per realitzar l'atac (es poden veure ressaltats en la **Figura 35**). En concret l'objectiu de l'atac serà el moment en que es repeteixin per novena vegada aquests valors de Program Counter, ja que significarà que estem a la ronda 9 de l'algoritme.

En aquest moment es necessita esbrinar quins són els registres del processador que han de ser atacats per aconseguir canviar el valor que suposarà que l'atac sigui un èxit. Amb aquest objectiu es poden seguir diverses estratègies:

-**Atac de força bruta**. Intentar l'atac amb cadascun dels registres del processador fins a trobar el que afecta a la sortida de l'algoritme de la manera desitjada. Aquest seria l'escenari real en el cas que un atacant tingués el dispositiu rootejat i instal·lés el GDB per no tingués cap coneixement sobre el codi del programa ni tingués cap eina que el pogués donar informació sobre aquest.

- **Fer ús de tècniques d'enginyeria inversa**. D'aquesta manera es pot arribar a obtenir la part del codi en la que s'està executant el mètode objectiu de l'atac i observar els registres utilitzats en cadascuna de les instruccions. A partir d'aquí, la idea seria mapejar els valors de PC amb les instruccions amb els registres corresponents.
- **Known Key Analysis**. Si ja s'ha obtingut la clau per altres mètodes (com per exemple un atac diferencial com els vistos en el capítol 6.1), es pot fer una anàlisi per veure quins són els registres que tenen el valor correcte en un cert punt de l'execució de l'algoritme. Per aconseguir aquesta informació es recopilen els valors de tots els registres per cadascun dels valors de PC. Com es coneix la clau de l'algoritme és poden calcular tots els valors intermedis de l'AES en cada ronda i comprovar quins registres i en quin moment contenen els valors correctes d'un cert byte de l'State.

En aquest projecte s'han dut a terme la segona tècnica i la tercera, només s'ha descartat la primera alternativa per qüestió de temps i eficiència. S'ha fet ús d'un software per a Linux anomenat Hopper que permet obtenir el codi a partir d'un binari (veure annex 2). En la **Figura 36** es pot veure una captura de pantalla del programa veient els valors de Program Counter que es corresponen amb la zona d'atac.

*No publicat per raons de confidencialitat.

Figura 36. Codi obtingut amb Hopper de la zona escollida per l'atac.

D'aquesta manera es poden veure les instruccions i registres del processador que s'utilitzen en l'operació que es vol atacar. En aquest punt de l'atac es necessita certa experiència en programació per identificar les instruccions d'interès. Hem de tenir en compte que quan s'analitza l'execució del binari i els valors que prenen els registres durant l'execució, ens trobem que el byte que volem atacar manté el mateix valor durant una sèrie d'instruccions. Això és degut a que els registres del processador no s'actualitzen en cada instrucció, sinó que, un cop actualitzats, mantenen el valor enregistrat fins que aquest és utilitzat per alguna altra instrucció.

Per tant, per aconseguir inserir la falta en el moment correcte s'haurà d'esperar a que el Program Counter desitjat es doni per novena vegada (en aquest cas 0x10b9c) i llavors canviar el valor del registre correcte. Per aconseguir esbrinar el moment exacte i el registre correcte s'han realitzat anàlisis i proves a partir del codi font proporcionat pel Hopper i s'ha arribat a la conclusió de que les instruccions afectades havien de ser les que tenien un valor de Program Counter igual a 0x10b9c, 0x10b9e, 0x10ba0 i 0x10ba2, i el registre objectiu de l'atac havia de ser r2, ja que era el registre que podia contenir el valor que es volia afectar.

En el següent codi (**Figura 37**) es pot observar com es realitza la inserció de la falta:

*no publicat per raons de confidencialitat

Figura 37. Script utilitzat per la inserció de la falta.

En la **Figura 38** es pot observar la sortida que proporciona l'algoritme després d'inserir aquesta falta, i en la **Figura 39** una execució normal de l'algoritme. Per considerar que l'atac s'ha desenvolupat amb èxit els valors de sortida d'ambdues execucions haurien de variar en 4 bytes.

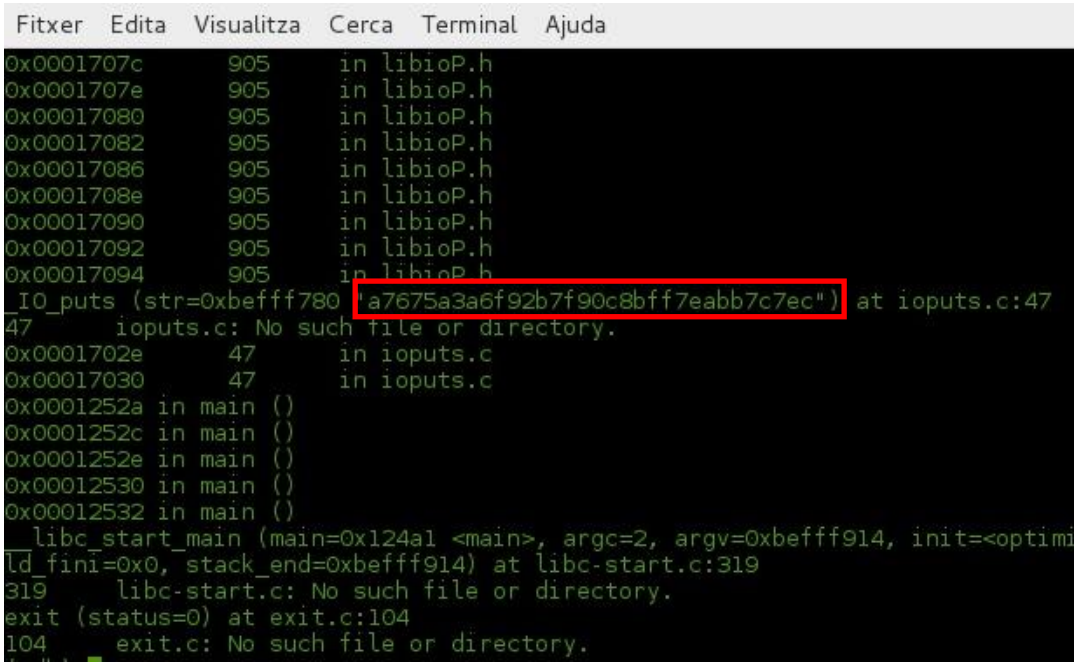


Figura 38. Sortida algoritme AES amb falta.

Figura 39. Sortida algoritme AES en una execució normal.

Com es pot veure els xifratges amb falta i sense falta són els següents:

CC	6F	0C	AB
67	92	8B	B6
5A	B7	18	C7
3A	4D	7E	EC

A7	6F	0C	AB
67	92	8B	B7
5A	B7	FF	C7
3A	F9	7E	EC

Figura 40. Comparació xifratge amb falta i sense falta WBAES.

Per comprovar que hem afectat a l'algoritme de la manera que buscàvem hem comparat aquest resultat amb el resultat d'inserir una falta en la nostra implementació d'AES utilitzant el mateix text per xifrar i la mateixa clau i s'ha comprovat que la falta es correspon a haver assignat el valor 0x00 al primer byte de l'State a la sortida de l'operació ShiftRows. Per tant presumiblement, el valor de PC on hem parat l'execució es correspon amb aquesta operació de la ronda 9 de l'AES. En aquest cas com estem en un entorn WB i no sabem quina és la implementació real de l'algoritme, aquest valor de PC també es podria correspondre amb la sortida de l'operació SubBytes de la ronda 9 ja que el registre r2 tindria el mateix valor en ambdues operacions i l'anàlisi del codi que retorna el Hopper no és tan específic com per poder distingir si estem davant d'una operació equivalent al mètode ShiftRows o al SubBytes d'un algoritme AES clàssic.

8.3. Recuperació del valor de la clau

Com s'ha explicat en el capítol 7, un cop aconseguim una falta i resolem les equacions corresponents obtenim varis candidats pel valor correcte de la subclau 10. En aquest punt, provocant diverses faltes, i tenint en consideració que el valor de la subclau no canvia, anem reduint el nombre de possibilitats fins arribar a la clau correcta. Fent això en 4 bytes diferents per obtenir els 16 bytes de la subclau i seguin el mateix raonament explicat en el capítol 7.1 es pot arribar al valor de la subclau de la ronda 10 i a partir d'aquesta obtenir la clau mestra de l'algoritme que en aquest cas és la següent:

4A 1C 6D F3 39 66 42 76 C5 96 7E 20 DC 2B EF EC

No obstant, val la pena analitzar que aquesta metodologia d'anar provocant faltes per reduir l'espai de possibilitats es duu a terme en el món del hardware perquè mai sabem

quina és la falta que hem provocat. Així provem tots els possibles valors que compleixen les equacions. En el nostre cas, en canvi, quan avaluem un algoritme en un entorn White Box, nosaltres sí tenim control total sobre el valor que escrivim al registre i, per tant, coneixem la falta que hem provocat exactament.

Per tant, podem fer ús d'aquesta informació per calcular el valor exacte de b (de les equacions (17) de l'apartat 7.1) coneixent Ein i Eout. A partir d'aquí, el valor correcte de la subclau k_{10} s'obté amb una sola falta (una falta per columna, de fet, per acabar obtenint els 16 bytes de la subclau).

Per tant l'atac en un entorn WB és més eficient que en el seu entorn hardware original pel qual es va concebre l'atac.

9. Conclusions i treball futur

Com a conclusió del treball realitzat es pot afirmar que és possible aplicar un atac de falta en un entorn White Box encara que la dificultat per realitzar-ho dependrà de les contramesures adoptades per l'aplicació objectiu de l'atac. S'ha de tenir en compte que qualsevol mesura de protecció del binari que faci més complicada l'enginyeria inversa o la manipulació (per exemple mitjançant debugging) de l'aplicació en qüestió dificultarà en gran mesura l'atac. L'ofuscació de codi faria que fos més difícil entendre el codi amb aplicacions d'enginyeria inversa com el Hopper Dissassembler utilitzat. Per altra banda, una execució diferent de l'algoritme podria donar lloc a que no es poguessin veure tan clares les rondes de la seva execució. Per exemple, es podrien donar rondes falses enmig de les veritables per tal d'aconseguir que l'atacant no trobi el moment en el que inserir la falta. Per tant per poder traslladar l'atac a una altra implementació, caldria un estudi previ anàleg al realitzat en aquest projecte per tal d'analitzar el seu funcionament.

Una altra conclusió important de l'estudi realitzat és que s'ha demostrat que en un entorn WB, el fet de tenir control total de l'execució, fa que l'atac testejat sigui molt més eficient que en un entorn hardware en que mai sabem on estem atacant i/o què estem modificant. Això ens obre un món de possibilitats a l'hora de pensar nous atacs específics per un entorn WB que valdria la pena explorar més enllà del que la criptografia clàssica defineix.

Com a treball futur i tenint com a punt de partida el treball realitzat es seguirà treballant estudiant diverses implementacions de White Box Cryptography amb l'objectiu d'aplicar la metodologia apresa per poder comprovar la seguretat d'altres implementacions que arribin al laboratori. L'objectiu és avaluar si la metodologia i els procediments descrits en aquest projecte són traslladables a altres implementacions de WB-AES "més protegides".

D'aquesta manera el laboratori podrà desenvolupar una metodologia més eficient per aconseguir valorar la seguretat de les aplicacions de molts dels nostres possibles clients futurs.

Veient que els dispositius mòbils cada vegada tenen més importància en el dia a dia i que, per tant, les implementacions de White Box Cryptography seran cada vegada més

presents, tenir un coneixement més profund del funcionament d'aquest tipus d'implementacions i de la metodologia per analitzar-les farà que ens puguem adaptar a les noves tecnologies ràpidament i contribuir a l'augment de la seguretat en sistemes de seguretat criptogràfica a través de tecnologies mòbils.

10. Bibliografia.

[1] Wyseur, B. (2009). *White-Box Cryptography*. *PhD Thesis*. Retrieved from http://www.cosic.esat.kuleuven.be/publications/talk-98.pdf

[2] Mulder, Y. De. (2014). *White-Box Cryptography. Analysis of White-Box AES* Implementations.

[3] Sanfelix, E., Mune, C., & Haas, J. De. (2015). *Unboxing the White-Box. Black Hat EU 2015.*

[4] Mulder, Y. De, Roelse, P., & Preneel, B. (n.d.). Lepoint. *Cryptanalysis of the Xiao-Lai white-box AES implementation.*

[5] Gao H. (2014). *White Box Cryptography to protect Cryptographic keys in software*.

[6] Hall, C., & Ferguson, N. (2001*). Chapter 7 The Advanced Encryption Standard* (AES), (November), 58–73.

[7] Dusart, P., Letourneux, G., & Vivolo, O. (2002). *Differential Fault Analysis on A.E.S.* https://eprint.iacr.org/2003/010.pdf

[8] Saxena, A., Wyseur, B., & Preneel, B. (2009). *Towards security notions for white-box cryptography. Lecture Notes in Computer Science* (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 5735 LNCS (1), 49–58. https://doi.org/10.1007/978-3-642-04474-8_4

[9] Joye, M. (2008). *On White-box Cryptography and Obfuscation*, 7–12. Retrieved from http://arxiv.org/abs/0805.4648

[10] Saxena, A., & Wyseur, B. (2008*). On White-box Cryptography and Obfuscation*, 7–12. Retrieved from http://arxiv.org/abs/0805.4648

[11] Thiebeauld, H. (2016). *In the depth of whitebox cryptography algorithms*. *White Paper*, (February).

[12] Chow, S., Eisen, P. a, Johnson, H., Van Oorschot, P. C., & Oorschot, P. C. Van. (2002). *White-box Cryptography and an AES Implementation*. Proc. of Selected Areas in Cryptography (SAC 2002), Lecture Notes in Computer Science 2595, 250–270. https://doi.org/10.1007/3-540-36492-7_17

[13] Billet, O., Gilbert, H., & Ech-chatbi, C. (2005). *Cryptanalysis of a White Box AES Implementation. Sac '04*, 227–240. https://doi.org/10.1007/978-3-540-30564-4_16

[14] De Mulder, Y., Roelse, P., & Preneel, B. (2013). *Revisiting the BGE Attack on a White-Box AES Implementation. Eprint.iacr.org*, 1–16. Retrieved from http://eprint.iacr.org/2013/450.pdf

[15] Klinec, D. (2013). *White-box attack resistant cryptography*. *Thesis*. Retrieved from http://is.muni.cz/th/325219/fi_m/thesis.pdf%5Cnhttps://is.muni.cz/th/325219/fi_m/thesis.pdf

[16] Lepoint, T., & Rivain, M. (2013*). Another Nail in the Coffin of White-Box AES Implementations. Eprint.Iacr.Org*. Retrieved from https://eprint.iacr.org/2013/455.pdf

[17] Muir, J. a. (2013). *A Tutorial on White-box AES*. Retrieved from https://eprint.iacr.org/2013/104.pdf

[18] Sasdrich, P., Moradi, A., & Güneysu, T. (2016). *White-Box Cryptography in the Gray Box – A Hardware Implementation and its Side Channels –.*

[19] Wyseur, B. (2012). *White-Box Cryptography. Hiding Kyeys on Software.* Retrieved from www.whiteboxcrypto.com/files/2012_misc.pdf

[20] Brier, E., Clavier, C., Oliver, F., *Correlation Power Analysis with a Leakage Model*, *CHES 2004*, 11-13 Aug. 2004.

[21] Kocher, P.; Joshua, J.; Jun, B., *Differential Power Analysis*, *Journal of Cryptographic Engineering,* Mar. 2011.

[22] Boneh D., DeMillo R. and Lipton R. *On the Importance of Checking Cryptographic Protocols for Faults*, Journal of Cryptology, Springer-Verlag, Vol. 14, No. 2, pp. 101-119, 2001

[23] Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., & Whelan, C. (2006). *The Sorcerer's Apprentice Guide to Fault Attacks. Proceedings of the IEEE*, *94*(2), 370–382. https://doi.org/10.1109/JPROC.2005.862424

[24] Giraud, C. (2005). *DFA on AES. 4th International Conference, AES 2004*, 27–41. Retrieved from: https://doi.org/10.1007/11506447_4

[25] Jacob, M., Boneh, D., & Felten, E. (2003). *Attacking an obfuscated cipher by injecting faults*. *Digital Rights Management*, 1–15. Retrieved from http://link.springer.com/chapter/10.1007/978-3-540-44993-5_2

[26] Sysk. (2012). *Practical Cracking of White-Box Implementations*. Retrieved from http://www.phrack.org/issues/68/8.html

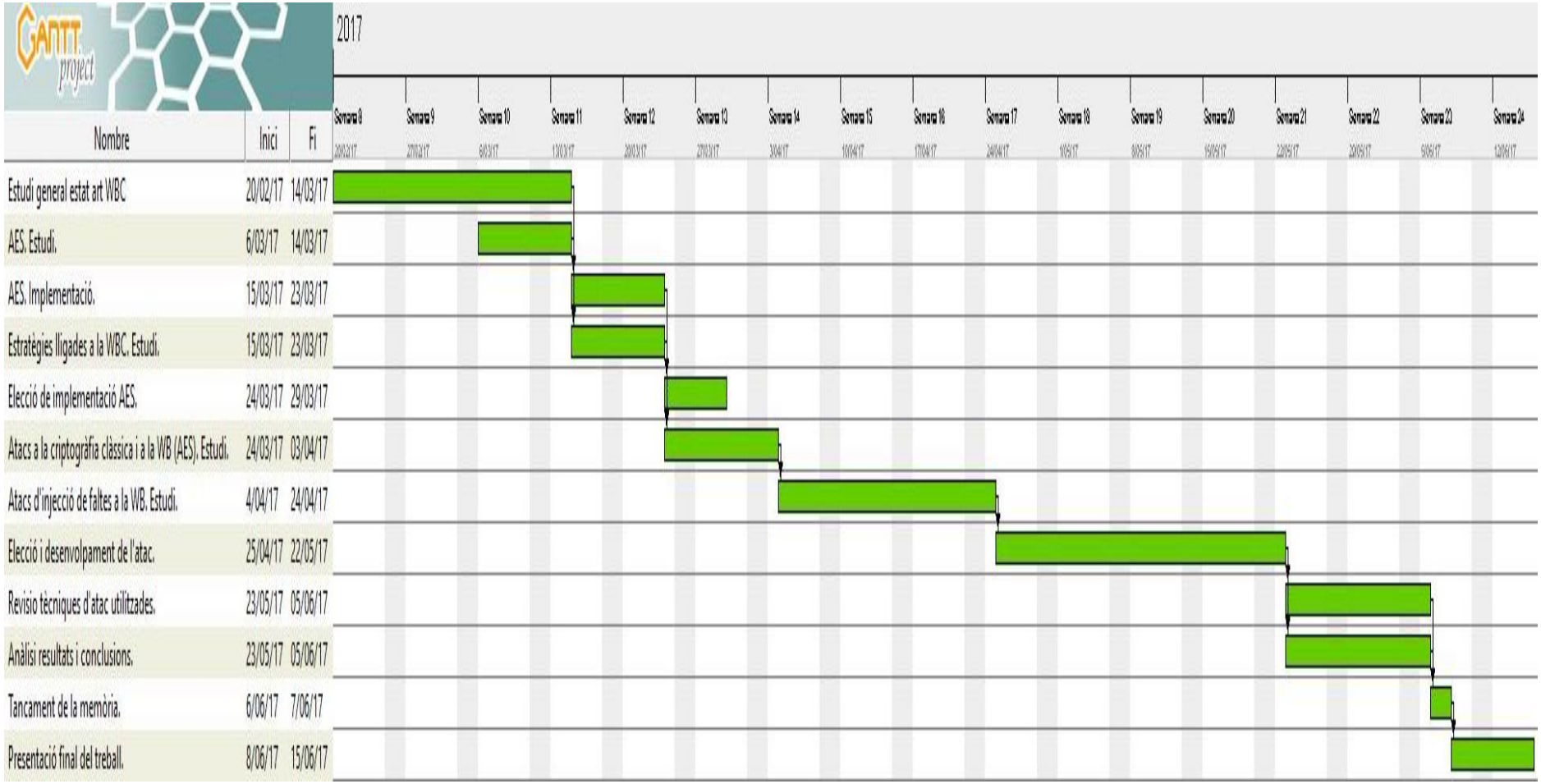
[27] Teuwen, P., & Hubain, C. (2016). *Differential Fault Analysis on White-box AES Implementations AES encryption DFA on AES-128 Encryption*. Retrieved from http://blog.quarkslab.com/differential-fault-analysis-on-white-box-aes-implementations.html

[28] Allibert, J., Feix, B., Gagnerot, G., Kane, I., Thiebeauld, H., & Razafindralambo, T. (2015). *Chicken or the Egg - Computational Data Attacks or Physical Attacks*, 1–26. Retrieved from https://eprint.iacr.org/2015/1086

[29] Bos, J. W., Hubain, C., Michiels, W., & Teuwen, P. (2015). *Differential Computation Analysis: Hiding your White-Box Designs is not enough. IACR 753*, 1–22. Retrieved from https://eprint.iacr.org/2015/753

Annexos

Annex 1 – Diagrama de Gantt



Annex 2 - Revisió eines utilitzades per l'atac a la implementació White Box de l'AES

Per poder aconseguir arribar a realitzar l'atac que s'ha pogut observar en l'apartat 10 del present treball, s'ha hagut de fer ús de diferents eines que s'introdueixen en aquest punt. El present apartat no pretén ser un anàlisis exhaustiu d'aquestes eines sinó veure com aquestes han servit per arribar a l'objectiu final amb èxit.

- **GDB debugger**. GDB és un depurador que permet executar l'execució d'un programa pas a pas i modificar el seu funcionament provocant canvis en les variables internes del programa.

  Les comandes més rellevants utilitzades en el present treball han sigut les següents:

  - *break*: permet aturar l'execució del programa en un cert punt determinat. Si es disposa del codi es pot introduir la línia en la que es vol posar el breakpoint, si no és així es pot aturar l'execució per un cert valor d'un registre o del Program Counter.
  - *step n*: permet reprendre l'execució i tornar a aturar el programa n instruccions després. Gràcies a aquesta comanda es pot executar l'aplicació pas a pas i veure les variacions que suposa cada instrucció.
  - *info registers*: dóna informació sobre els valors dels diferents registres en un moment donat.
  - *set:* permet modificar el valor d'un registre o d'una variable en un moment donat de l'execució. Aquesta és la comanda utilitzada per inserir les faltes de l'atac realitzat en el present treball.
  - *print:* mostra per pantalla el valor d'un registre o variable en un moment donat.
  - *set logging file:* defineix un fitxer de sortida en el que s'emmagatzemen les dades definides amb la comanda *p/x.* Per activar aquesta característica s'ha d'introduir la comanda *set logging on.*
  - *Source:* permet cridar a un script en llenguatge gdb guardat en un fitxer amb anterioritat.
  - *set args:* permet definir els arguments d'entrada del programa que s'està analitzant.

Per més informació o per la descàrrega del programa es pot acudir a:

https://www.gnu.org/software/gdb/

-**Termux**. En un primer moment per la realització es va fer ús del GDB de manera remota amb GDB server. D'aquesta manera des d'un ordinador amb Linux es podia connectar amb el dispositiu ARM per tal de realitzar l'execució i fer debugging del programa. Aquest mètode tenia el problema de ser molt més lent que l'execució en local i per poder aconseguir desenvolupar l'atac en local es va fer ús de *Termux*.

Aquesta aplicació (disponible a la Play Store de Google) permet disposar d'un terminal en un dispositiu mòbil i poder executar el GDB de manera local. Amb aquesta decisió es va aconseguir que el procés d'anàlisi i atac de la implementació WBAES fos molt més ràpid.

La direcció de la seva pàgina web per obtenir més informació és:

https://termux.com/

- **Hopper Disassembler**. Aquesta és una aplicació que permet fer enginyeria inversa d'un programa a partir del seu binari, per tal d'analitzar-ne les seccions, funcions declarades i les instruccions en llenguatge màquina d'aquestes. Suporta diferents arquitectures com ARM , x86, entre d'altres i té una interfície gràfica avançada que permet moure's per les zones del binari amb molta comoditat i analitzar-ho amb molt detall. La seva efectivitat dependrà també del nivell d'ofuscació que tingui el programa objectiu de l'anàlisi.

En la seva pàgina web es troba més informació del seu funcionament:

https://www.hopperapp.com/

Glossari

AES. Advanced Encryption Standard. Algoritme de criptografia simètrica desenvolupat als EEUU l'any 2001 per Daemen i Rijmen. Va ser l'escollit en una competició promoguda pel NIST.

ARM. Fa referència a una arquitectura desenvolupada per ARM Holdings. És l'arquitectura majoritària en l'actualitat en els dispositius mòbils.

Algoritme Asimètric. Algoritme criptogràfic en el que s'utilitzen dues claus diferents pel xifratge i el desxifratge (clau pública i clau privada).

Algoritme Simètric. Algoritme criptogràfic en el que s'utilitza una mateixa clau pel xifratge i el desxifratge.

Camp de Galois (GF). També conegut com camp finit, és un cos definit sobre un nombre finit d'elements. Són molt utilitzats en criptografia, geometria i teoria de nombres.

Coeficient de correlació de Pearson. Càlcul molt utilitzat en estadística que retorna la relació lineal entre dues variables.

Criptoanàlisi. Vessant de la criptologia que s'encarrega d'estudiar els mètodes per trencar la seguretat obtinguda a través dels diferents xifratges.

Criptografia. Vessant de la criptologia que s'encarrega de l'estudi del xifratge de missatges.

Criptologia. Ciència que estudia com amagar informació d'interès perquè sigui il·legible per un tercer.

Debugging. És un procés que permet executar un programa pas a pas per identificar algun error en la seva execució o per modificar el seu comportament d'alguna forma. Hi ha moltes eines que permeten realitzar aquest tipus de tasques. En el present treball GDB ha sigut l'eina seleccionada per portar a terme aquesta tasca.

Differential Electromagnetic Analysis (DEMA). Tipus d'atac que utilitza una gran quantitat de traces de radiació electromagnètica emesa per un hardware per tal d'aplicar un càlcul estadístic entre dos grups diferents de traces que permeti aconseguir informació confidencial.

Differential Power Analysis (DPA). Tipus d'atac que utilitza una gran quantitat de traces de potència elèctrica generada per un hardware per tal d'aplicar un càlcul estadístic entre dos grups diferents de traces que permeti aconseguir informació confidencial.

National Institute of Standards and Technology (NIST). Agència americana creada amb l'objectiu de promoure la innovació i la competència als Estats Units.

Program Counter (PC). Registre del processador de un computador que indica la posició del processador en la seva seqüència d'instruccions.

Root. En Android ser usuari root implica que es disposa de tots els drets per realitzar canvis en el dispositiu.

Simple Electromagnetic Analysis (SEMA). Anàlisi amb poques traces de la radiació electromagnètica emesa per un hardware per tal d'identificar alguna operació concreta.

Side Channel. Tipus d'atacs que utilitzen informació retornada per un hardware involuntàriament.

Simple Power Analysis (SPA). Anàlisi amb poques traces de la potència elèctrica generada per un hardware per tal d'identificar alguna operació concreta.

State. Durant l'execució de l'AES es denomina State a cadascuna de les matrius d'entrada o de sortida dels diferents mètodes que formes l'algoritme.

Taula de cerca. Estructura de dades que retorna un valor per un cert valor d'entrada. Són molt utilitzades en informàtica ja que poden implementar funcions sense la necessitat de realitzar càlculs matemàtics i per tant de manera més ràpida. Les S-Box, T-Box i Ty-Box són exemples d'aquest tipus d'estructura.

White Box (WB). Entorn en el que l'atacant pot tenir control total del sistema. Es tracta de l'escenari més insegur i en el que calen més mesures de seguretat a l'hora de portar a terme operacions confidencials.

White Box Cryptography (WBC). Branca de la criptografia que s'encarrega de l'estudi del xifratge de missatges en un entorn WB.