AES: Square Attack

Sharp Thomas (1986413)

16 Luglio 2024

Repo Github

1 Introduzione

Come progetto di esame tra le tracce disponibili ho scelto quella riguardante crittoanalisi. Partendo dalla definizione, crittoanalisi è il processo in cui si cerca di scoprire delle informazioni riguardo il sistema crittografico, in genere il plaintext o addirittura la chiave che viene usata per cifrare.
Come algoritmo di cifratura a chiave simmetrica ho deciso di attaccare l'Advanced Encrypting Standard, d'ora in poi AES, il cui suo scopo è stato quello di sostituire DES e 3DES con un'algoritmo più sicuro ed efficiente, e per la precisione la versione che attaccherò sarà AES128, cioè la versione di AES con chiave da 128bit (anche AES192 e AES256 sono disponibili ma in questo caso la complessità di tempo sarebbe stata il fattore principale da battere).
La crittoanalisi però è solo un tipo di attacco, che comprende vari modi in cui svolgerlo, e come metodo ho scelto lo Square Attack, anche noto come crittoanalisi integrale. Quindi in nostri obbiettivi saranno di implementare:

1. l'algoritmo per AES
2. una procedura per riuscire ad ottenere la chiave di cifratura

2 AES

2.1 Introduzione

AES è un'algoritmo di cifratura a blocchi, cioè che prende blocchi di plaintext e li cifra restituendo un ciphertext della stessa dimensione, e nel caso dell'AES i blocchi presi in input sono di 128bit. Nel processo di generazione del ciphertext il blocco attraversa varie trasformazioni, principalmente permutazioni e sostituzioni, raggruppate a round, in genere 10 round. Per ognuno di questi round viene usata una subkey (o roundkey) generata dalla chiave originale, mentre la chiave originale viene usata per un whitening (un semplice XOR) prima del primo round.

2.2 Implementazione

Il plaintext durante la sua trasformazione in uno ciphetext viene contenuto in uno State, lo state è semplicemente una matrice di byte, da 16 elementi, in cui la prima word (32bit, 4 byte) del testo è posta nella prima colonna e cosi via per le restanti. Per comodità e facilità d'uso rappresenteremo la key da 128bit nella stessa

maniera.

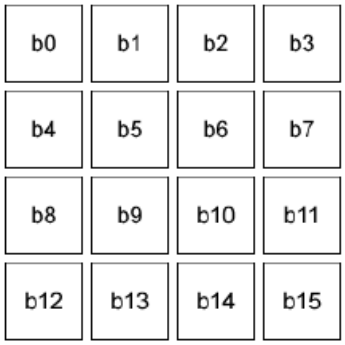

Figure 1: Rappresentazione usata per stati e chiavi

I passi da fare per creare una semplice implementazione dell'AES sono:

1. implmentare l'espansione della chiave, fase prima del cifraggio che prende in input la chiave originaria e ritorna il numero di chiavi necessarie per generare il ciphetext.

2. implementare una funzione che iteri sui vari round dell'AES avanzando nella varie fasi di cifratura.

2.2.1 Key Scheduler

L'espansione della chiave si divide in due operazioni, generazione della prima word (colonna) della subchiave, e generazione delle successive. Le due sono operazioni separate perchè operano in maniera diversa.
La prima colonna della subkey è calcolata applicando:

1. una rotazione a sinistra di un byte alla colonna precedente (l'ultima della chiave precedente)

2. la colonna risultante subisce una sostituzione tramite S-Box, una semplice matrice 16×16 dove le righe sono indicate dai 4 bit più significativi del byte e i 4 meno siginificativi indicano le colonne.

3. ciò che ci rimane infine deve attraversare due XOR, uno con la prima colonna della chiave precedente e uno con la costante di round (una word in cui il valore è posto negli 8bit più significativi).

x \ y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
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	BI	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

Figure 2: Rappresentazione del key scheduler.

Mentre le colonne restanti sono calcolate facendo lo XOR tra la colonna direttamente precedente e la colonna di stesso inidice della chiave precedente.

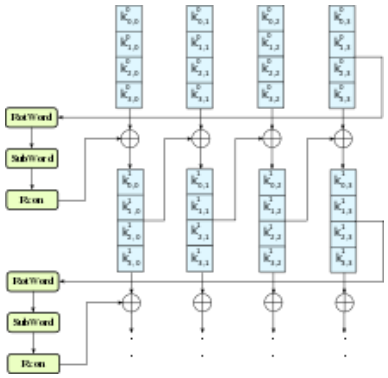

Figure 3: Rappresentazione del key scheduler.

Questo processo viene ripetuto per il numero di chiavi richieste dall'AES e ci fornisce quindi una lista di chiavi da utilizzare per ogni round.

2.2.2 Algoritmo

La prima cosa che viene fatta dall'algoritmo di cifratura è apliare la chiave tramite il key scheduler come appena descritto.
Successivamente vengono ripetuti $n-1$ round composti dei seguenti passaggi:

1. SubByte, in questo passaggio viene usata la S-box per sostituire i byte

2. RotRow, ogni riga viene ruotato di i posizioni a sinistra dove i è l'indice della riga (quindi la prima riga viene ruotata di 0, la seconda di 1 e cosi via).

3. MixColumns, cambia ogni byte a seconda di ogni altro byte presente nella colonna, in pratica fa una moltiplicazione colonna-matrice con la seguente matrice:

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

4. AddRoundKey, infine viene fatto lo XOR bitwise tra la chiave del round e lo stato risultante dai passaggi precedenti.

Notare bene che questo viene ripetuto per $n-1$ round perché l'ultimo round effettua ogni operazione tranne che la MixColumns, e sopratutto che l'algoritmo è di facile comprensione.

Quello che segue è il codice della funzione di cifraggio:

```
    def encrypt(state: State,
    key: str, nrounds: int= 10):
    '''Funzione che esegue il
    cifraggio di uno State data
    una certa key
    con nrounds rounds'''
    round_keys= ke.key_expansion
    (key, nrounds)
    add_roundkey(state,
    round_keys[0])
    for x in range(nrounds-1):
        round(state, round_keys[
    x+1])
    sub_state(state)
    shift_rows(state)
    add_roundkey(state,
    round_keys[nrounds])
```

Il risultato di questi passaggi è un ciphertext completo che corrisponde al blocco di plaintext.

3 Square Attack

3.1 Introduzione

Come detto in precedenza lo square attack è un tipo di crittoanalisi volta allo scoprire la chiave utilizzata per cifrare un messaggio tramite AES. Prende il nome dal precursore dell'AES per cui è stato ideato, l'algoritmo di cifratura Square.

Fortunatamente, essendo un sistema di cifratura sicuro, l'AES non può essere "rotto" nella sua forma normale a 10 round, non per un'impossibilità a livello di disponibilità di algoritmi per scoprire una chiave, ma più che altro per una questione di complessità di tempo di suddetti algoritmi, e tutto ciò vale anche per lo Square Attack.
Visto che ci sono limiti di tempo per cui una certa informazione, come la chiave, ha valore il nostro Square Attack funzionerà solo per un AES con cifratura a 4 rounds.

3.2 Algoritmo

Il processo di scoperta della chiave richiede un elemento fondamentale che rende questo tipo di attacco un chosen-plaintext attack, un *oracolo*, che teoricamente sarebbe una blackbox a cui viene fornito un plaintext e che ci ritorna indietro il ciphertext usando la chiave da noi ricercata.
Per motivi dimostrativi andremo noi a crearci il nostro oracolo a cui forniamo la chiave con cui generiamo i testi cifrati.

Questo oracolo ci servirà per generare i *DeltaSet*, cioè degli insiemi di contenenti 256 stati, definiti in un modo specifico. Ognuno di questi stati appartenenti al DeltaSet varia su solo il primo byte, che assume tutti i valori possibili di un byte $(0-255)$attraverso i vari stati dell'insieme, e i byte restanti sono uguali per ogni stato nel DeltaSet. Utilizzare questo particolare insieme di stati è importante perché quel singolo byte se viene operato in XOR con tutti gli altri 255 di suo stesso indice ci darà come risultato 0, e questa peculiarità oltre ad essere mantenuta viene pure espansa a tutti gli altri byte per la fine del terzo round.

Specificati i fondamentali, i seguenti sono i passaggi per ottenere la chiave da un cifraggio AES al quarto round tramite oracolo:

1. Trovare la chiave del quarto round, richiede:
   (a) Generare un DeltaSet e cifrarlo tramite l'oracolo
   (b) Generare una chiave contenente solo un byte non-zero
   (c) Decifrare applicando AddRoundKey e l'inverso di RotRow e SubBytes per tornare all'inizio del terzo round e usare la proprietà del DeltaSet per verificare che il byte della chiave sia corretto, altrimenti passare ad un'altro valore.
   (d) ripetere per ogni $i-$esimo byte della chiave fino a non aver scoperto ogni byte della chiave
2. Effettuare l'inverso delle operazioni del KeyScheduler per ottenere tutte le chiavi generate da quella originale, e quella originale stessa.

Di seguito è presente il codice delle funzioni dei punti 1. e 2.:

```
def fourth_key(k: str) -> str:
'''Funzione che prende in input
    una chiave in esadecimale,
esegue una cifratura tramite AES
     fino al quarto round, e poi
     la rompe.'''
set= setup(k)
key= ""
for index in range(16):
    key= key + find_guess(set,
    index, k)
    if index % 4 == 0: print(
    index+1, "-esimo byte
    trovato")
return key

def find_guess(dset: DeltaSet, i
    : int, k: str) -> str:
'''Funzione che per un certo
    indice (byte) della chiave
    del quarto round
trova qual'e' il byte effettivo
    della round key'''
valids= []
for x in range(256):
    copy= dset.__deepcopy__()
    guess= hex(x).removeprefix("
    0x")
    reverse_state(guess, i, copy
    )
    if copy.check(i):
        valids.append(guess)
    del copy
if len(valids) != 1:
```

```
    while len(valids) != 1:
        for b in valids:
            states= setup(k)
            reverse_state(b, i,
    states)
            if not states.check(
    i):
                valids.remove(b)
return valids[0]
```

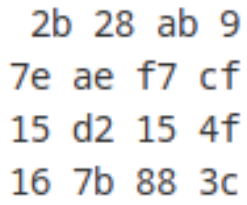

Figure 4: Risultato dello Square Attack

Questo processo e implementazione ci porta a scoprire la chiave di cifratura dei messaggi, sempre a patto di avere a disponibilità qualcosa che ci permetta a partire dal plaintext di ottenere i ciphertext che vogliamo.

Il processo completo ci mette intorno i 3 minuti per scoprire la chiave ma è sicuramente migliorabile andando a fare alcune ovvie modifiche al codice. Facendo un test con input come chiave "*2b7e151628aed27babf7158809cf4f3c*" vedremo che correttamente otteniamo come risultato:

4 Conclusioni

Considerando che AES è un algoritmo quantum-proof, con i metodi attuali semplicemente non è possibile ottenere la chiave di un sistema con cifratura tramite AES in tempi ragionevoli, lo Square Attack a livello di praticità termina al quarto round, anche se ampliabile al quinto round dovendo indovinare più bytes (5, effettuando modifiche all'esecuzione del quarto round, altrimenti 8) per indovinare un singolo byte, e al sesto andando ad aggiungere un round prima invece che alla fine, ma entrambi in ogni caso iniziano a mostrare problemi di complessità di tempo.