Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Matematica

CRITTOGRAFIA SIMMETRICA: IL SISTEMA AES

Tesi di Laurea in Algoritmi della Teoria dei Numeri e Crittografia

Relatore:
Davide Aliffi

Presentata da:
Andrea Tasini

II Sessione
Anno Accademico 2010-2011

Indice

Introduzione

In questo elaborato viene fatta una descrizione del sistema di crittografia simmetrica *Advanced Encryption Standard*, AES. Il sistema fu creato da due crittografi belgi, Joan Daemen e Vincent Rijmen, con il nome di algoritmo ¨Rijndael˝. Come si dirà in seguito, l'algoritmo venne presentato ad un concorso indetto dal *National Institute of Standards and Technology* (NIST) nel 1997, per sostituire l'ormai obsoleto DES. Una volta nominato vincitore del concorso nel 2001, l'algoritmo ¨Rijndael˝divenne conosciuto in tutto il mondo come nuovo standard AES.
Rijndael è un cifrario a blocchi, come il NIST richiedeva, perciò è un algoritmo che opera su dei singoli blocchi di testo in chiaro (in questo caso da 128 bit). Nonostante presenti questa caratteristica prefissata, differisce totalmente dal precedente standard DES (*Data Encryption Standard*) sia per la sua innovativa struttura, che per la sua resistenza alla maggior parte degli attacchi noti. Innanzitutto Rijndael si avvale di una chiave molto più lunga e quindi più resistente ad attacchi a forza bruta di DES. Le chiavi dei sistemi crittografici simmetrici vengono spesso classificate in base alla loro lunghezza in bit (essi possono essere identificati come elementi del campo algebrico $\mathbb{Z}_2$). Per esempio, DES sfrutta una chiave di 56 bit, mentre AES ne adotta una di lunghezza variabile tra 128, 192, 256 bit (tipicamente 128). Per effettuare un attacco a forza bruta, si compie una semplice ricerca nello spazio delle chiavi, provando chiave per chiave a forzare il sistema. Ovviamente questo tipo di attacco richiede un tempo di esecuzione tanto maggiore quanto più è ampia la dimensione dello spazio in cui si ricerca la chiave stessa, ossia di 2^{56} possibili chiavi (combinazioni di bit) per quanto riguarda DES e di 2^{128} per quanto riguarda AES. Da quest'ultima considerazione si desume immediatamente che AES è estremamente più sicuro e resistente a questo tipo di attacco rispetto a DES.
Oltre alla chiara differenza nella lunghezza delle chiavi, Rijndael presenta una struttura nuova rispetto a DES. Quest'ultimo è un esempio di cifrario di *Feistel*: gli algoritmi di cifratura e decifrazione del messaggio sono uguali e sono composti da una serie di passaggi in cui vengono ripetutamente scambiate la

metà di destra del blocco considerato e la metà sinistra. I designer di Rijndael, invece, non utilizzano questo tipo di modello, ma creano una struttura innovativa approfittando delle proprietà dei campi algebrici finiti. Queste proprietà, che verranno discusse più dettagliatamente nei prossimi capitoli, si basano sulla struttura algebrica dei campi finiti del tipo $GF(2^n)$ (nel caso di AES $n = 8$). Questi campi, che prendono il nome di *Galois Fields*, sono dei particolari campi finiti con cardinalità pari alla potenza di un primo e che, per questo, differiscono dai classici $\mathbb{Z}_p$ con p primo. Sono particolarmente utili per la descrizione dell'algoritmo, in quanto si può sfruttare la visione polinomiale dei *bytes* sui quali si opera. I *bytes*, come verrà detto in seguito, sono una sequenza di 8 bit, per esempio $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ e hanno perciò una forma particolarmente adattabile ad una notazione polinomiale del tipo:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_ix^i.$$

Questo grazie ad un particolare isomorfismo che associa i *Galois Fields* ai quozienti degli anelli dei polinomi a coefficienti in $\mathbb{Z}_p$

$$GF(p^n) \cong \mathbb{Z}_p[x]/a(x) \quad \forall a(x) \quad \textit{polinomio irriducibile} \quad : \quad deg(a(x)) = n.$$

Accantonata l'ipotesi di un attacco a forza bruta, impossibile da applicare in tempi decenti con la potenza delle macchine attuali, i creatori di Rijndael si concentrarono di più sul rendere l'algoritmo resistente ad altri tipi di attacco, come quelli con testo in chiaro noto. La crittoanalisi moderna, infatti, sfrutta soprattutto questo modello di attacco, considerando tutte le possibili correlazioni che ci potrebbero essere tra i bit in input (in chiaro) e quelli in output (cifrati). La struttura di AES è finalizzata a ridurre al minimo questa correlazione, cancellando in pratica la possibilità di essere attaccato per mezzo di questi metodi. La complessità algebrica delle operazioni effettuate, ripetute per dieci volte, dona al cifrario una notevole ed immediata diffusione dei *bytes* rendendo di fatto minima la correlazione tra input e output. Nonostante tutto, è stata effettuata di recente una ricerca da parte di alcuni crittoanalisti al fine di trovare un attacco degno di AES. Andrey Bogdanov, Dmitry Khovratovich e Christian Rechberger sono riusciti infatti a ridurre considerevolmente i tempi per "craccare"il cifrario ma, come ammette Bogdanov stesso: "*non siamo nemmeno vicini a forzare AES, al momento*". Infatti sebbene calati di ben cinque volte rispetto al normale, i tempi restano ancora troppo lunghi per un'utilità pratica; rendendo di fatto l'algoritmo ancora inviolato.

Capitolo 1

Storia di AES

Per diversi anni, lo standard più utilizzato in tutto il mondo della crittografia fu DES, *Data Encryption Standard*. Questo algoritmo venne scelto dall'ente per la tecnologia, il *National Istitute of Standards and Technology* (NIST), già dagli anni '70. Alla fine degli anni '90, però, il sistema era fortemente vulnerabile ad attacchi di forza bruta. Infatti, la chiave che, per le macchine contemporanee all'ideazione di DES, era resistente a questi tipi di attacchi, per le macchine più recenti è diventata semplice da trovare grazie alle sue ridotte dimensioni (soli 56 bit). Si cercò così di rendere più sicuro il sistema grazie a 3DES. Questo nuovo sistema si presentava più sicuro di DES grazie ad una chiave triplicata rispetto alla precedente (168 bit), ma era poco efficiente. Il NIST indisse, così, il 12 settembre 1997 un concorso pubblico al fine di selezionare un nuovo standard che fosse sicuro come 3DES (ma più efficiente) e che avrebbe preso il nome di *Advanced Encription Standard* (AES). L'anno seguente, all'apertura del concorso, il NIST annunciò i 15 candidati ufficiali, che furono sottoposti poi all'attenzione della comunità crittografica mondiale. Il 15 Aprile 1999 il NIST proclamò i 5 finalisti del concorso e il 2 Ottobre 2000 selezionò ufficialmente Rijndael come algoritmo da proporre. Dopo diverse verifiche, il Segretario del Dipartimento di Commercio americano approvò definitivamente Rijndael come nuovo standard AES.

1.1 Metodi di valutazione di AES

Il NIST usò due diversi metri di giudizio nelle due fasi di valutazione degli algoritmi. In primo luogo si basò sulla verifica di requisiti fondamentali.

- **SICUREZZA**: fa riferimento all'impegno necessario per analizzare crittograficamente un algoritmo. L'enfasi nella valutazione era sulle

possibilità pratiche di un attacco. Poichè in AES la chiave ha dimensioni minime di 128 bit, gli attacchi a forza bruta con le tecnologie attuali e future non vennero neppure considerati.

- **COSTO**: il NIST richiese che l'algoritmo AES potesse essere impiegato per un'ampia gamma di applicazioni. Di conseguenza, doveva avere un'elevata efficienza computazionale e doveva essere utilizzabile in applicazioni ad alta velocità, come per esempio i collegamenti a banda larga.

- **CARATTERISTICHE DELL'ALGORITMO E DELL'IMPLEMENTAZIONE**: questa categoria includeva vari aspetti fra cui la flessibilità, la possibilità di impiego in varie implementazioni hardware e software e la semplicità, per facilitare l'analisi della sicurezza.

Per la valutazione dei cinque finalisti i criteri vennero aggiornati e ampliati per fare un'analisi ancora più attenta. I tre criteri fondamentali precedenti divennero più approfonditi e dettagliati in diversi punti secondo quanto segue:

- **Sicurezza generale**: la comunità crittografica valutò questo criterio tramite una pubblica analisi degli algoritmi proposti. Questo è stato il primo cifrario in cui la valutazione di un algoritmo crittografico candidato a diventare uno standard è stata resa pubblica.

- **Implementazioni software**: velocità di esecuzione, prestazione in varie piattaforme e variazione della velocità in base alle dimensioni della chiave.

- **Ambienti con spazio limitato**: gli algoritmi devono richiedere alla macchina una quantità di memoria che renda possibile l'esecuzione in ambieti con memoria ristretta (per esempio Smart Card).

- **Implementazioni hardware**: per questo tipo di implementazione hanno una maggiore importanza le dimensioni che si traducono direttamente in costi.

- **Attacchi alle implementazioni** questi attacchi, a differenza della analisi crittografica, non sfruttano le proprietà matematiche impiegate nell'algoritmo, bensì le sue caratteristiche intrinseche e le funzionalità dipendenti all'implementazione.

- **Crittografia e decrittografia**: considerazioni generali sugli algoritmi di crittografia e decrittografia, qualora siano diversi e la decrittografia richieda ulteriore spazio hardware.

- **Agilità della chiave**: verifica la capacità di cambiare rapidamente la chiave del sistema utilizzando la minima quantità di risorse.
- **Versatilità e flessibilità**: facilità di supporto di chiavi e blocchi di diverse dimensioni.
- **Potenzialità di sfruttamento del parallelismo a livello delle istruzioni**: capacità di sfruttare le funzionalità di esecuzione parallela nei processori attuali e futuri.

In base a questi criteri venne scelto e poi approvato dal Segretario del Dipartimento di Commercio americano l'algoritmo Rijndael. Esso infatti rispetta praticamente tutti i canoni richiesti dal NIST e, soprattutto, è resistente a tutti gli attacchi a testo in chiaro noto (*known attacks*) ad un costo computazionale relativamente molto basso.

Capitolo 2

Elementi matematici di base

2.1 I campi finiti

Definizione 2.1. Un campo è una struttura algebrica composta da un insieme K e da due operazioni binarie dette *addizione* e *moltiplicazione*, indicate con i simboli $+$ e $*$, tali che $(K, +)$ e $(K\backslash\{0\}, *)$ siano due gruppi abeliani con elementi neutri rispettivamente 0 e 1, che ogni elemento di K abbia il suo inverso moltiplicativo e che la moltiplicazione sia distributiva rispetto all'addizione.

I campi finiti sono, come suggerisce il nome, dei campi che contengono un numero finito di elementi. Essi sono classificabili in base alla loro cardinalità nel seguente modo:

- ogni campo finito ha p^n elementi, per qualche numero primo p e qualche numero naturale $n \geq 1$.
- per ogni numero primo p e naturale $n \geq 1$, esiste un solo campo finito con p^n elementi, a meno di isomorfismi.

Un generico campo finito di p^n elementi viene così indicato con la notazione $GF(p^n)$.
Un caso particolare lo si può riscontrare per $n = 1$, quando le operazioni di $GF(p)$ vengono definite dalla normale aritmetica modulare. In questo frangente, il campo assume comunemente la notazione di $\mathbb{Z}_p$, indicando appunto il campo delle classi di resto modulo p definito dal quoziente $\mathbb{Z}/p\mathbb{Z}$.
Per $n > 1$ si entra nella casistica più rilevante dal punto di vista della crittografia e, quindi, di AES. Questi campi non sono più rappresentabili dall'aritmetica modulare come sopra.[1] Essi, infatti, vengono definiti da un'aritmetica

[1] $\mathbb{Z}_p$ è un campo se e solo se p è un numero primo, quindi, essendo p^n non primo, $\mathbb{Z}_{p^n}$ non è un campo

di tipo polinomiale, in cui ogni elemento viene espresso come un polinomio di grado al più $n-1$ a coefficienti in $\mathbb{Z}_p$. Particolarmente importanti per la crittografia sono i campi del tipo $GF(2^n)$, come verrà mostrato nei paragrafi seguenti.

2.2 Il Byte

Definizione 2.2. Il *Byte* è definito come una sequenza di bit (tipicamente 8). Spesso, perciò, viene utilizzata la notazione *8-bit byte*.

In AES può essere visto, inoltre, come un array di bit rappresentabile nel seguente modo:

$$\{b_7b_6b_5b_4b_3b_2b_1b_0\} \quad b_i \in \{0,1\} \quad 0 \leq i \leq 7$$

dove i b_i sono appunto i singoli bit che compongono il *byte*. In AES è fondamentale considerare il *Byte* come una singola entità, poichè è l'unità base di tutti i processi dell'algoritmo. Quest'ultimo prende in input, infatti, un blocco di una quantità variabile di bit, che verrà poi organizzato in un array bidimensionale di *bytes* (matrice) per applicare in un secondo momento diverse operazioni. A questo scopo è utile anche avere una visione del *Byte* diviso in due porzioni da 4 bit ciascuno e, quindi, dare la seguente definizione.

Definizione 2.3 (Nibble)**.** Un *nibble* è una delle due metà di un singolo *Byte*, quindi una sequenza di 4 bit. In particolare, si chiameranno i 4 bit di sinistra *nibble* più significativo e i 4 più di destra *nibble* meno significativo.

Ogni *nibble* può essere visto a sua volta come una rappresentazione in base esadecimale, in quanto il massimo di combinazioni distinte possibili di 4 bit è proprio sedici. Di conseguenza in AES un *Byte* viene spesso trattato come una coppia di valori compresi in $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$. Infine, un'ultima rappresentazione del *Byte*, che viene spesso usata nel cifrario descritto, è quella polinomiale. Si può osservare che un singolo *Byte* è esprimibile come un polinomio di settimo grado a coefficienti in $\mathbb{Z}_2$.

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_ix^i$$

Dove i coefficienti b_i del polinomio sono i bit che compongono il *byte* e gli x^i vengono usati come segnaposto, per indicare l'ordine dei bit stessi. A questo proposito è possibile sfruttare l'aritmetica di un particolare campo finito. Tra i campi del tipo $GF(2^n)$ gli ideatori dell'algoritmo Rijndael decisero di usare $n = 8$, vista la struttura del *Byte* sopra definita.

2.3 Il campo finito $GF(2^8)$

Genericamente parlando, i campi finiti del tipo $GF(2^n)$ sono estensioni di grado n su $\mathbb{Z}_2$; è infatti possibile osservare un particolare isomorfismo

$$GF(2^n) \simeq \mathbb{Z}_2[x]/p(x) \quad \forall p(x)\ irriducibile\ \in \mathbb{Z}_2[x] \quad deg(p(x)) = n.$$

Nel caso di AES, i suoi ideatori Deamen e Rijmen pensarono di dare ad ogni *byte* b in input una corrispondenza con un elemento in

$$GF(2^8) \simeq \mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1)$$

sfruttando, appunto, la formulazione del *byte* in senso polinomiale descritta nel paragrafo precedente.[2]
La scelta di utilizzare un'aritmetica in un campo è fondamentale in ogni algoritmo di crittografia, in particolare se all'interno dell'algoritmo compare un'operazione quale la divisione. É anche logico, in secondo luogo, usare un'aritmetica su di un campo i cui elementi rappresentano esattamente l'intera configurazione dei byte usati nell'algoritmo. Se, per esempio, si considerasse un algoritmo di crittografia convenzionale, ossia che opera su *8-bit byte* e che sfrutta la divisione - come nel caso di AES, si vorrebbe sfruttare un campo che operi su tutti i numeri compresi tra 0 e $2^8 - 1$, per evitare sia mancati utilizzi di interi (nel caso si scegliesse un campo con cardinalità minore di $2^8 = 256$), sia sprechi di configurazioni binarie (nel caso si scegliesse una cardinalità maggiore). Tuttavia, 256 non è un numero primo, pertanto $\mathbb{Z}_{256}$, nonostante sia composto dagli interi necessari per sviluppare un'aritmetica adatta ad AES, non è un campo. Si farà riferimento, anche per questo motivo, al campo finito $GF(2^8)$, poiché risponde a tutte le esigenze del caso. Sviluppando più in dettaglio le proprietà di questo particolare campo, si esamineranno le due operazioni (addizione e moltiplicazione) nel loro funzionamento.

2.3.1 Addizione

Definizione 2.4. Per due *bytes* $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$ e $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$ l'operazione di addizione viene indicata con il simbolo $\oplus$ ed è definita nel seguente modo:

$$\{a_7a_6a_5a_4a_3a_2a_1a_0\} \oplus \{b_7b_6b_5b_4b_3b_2b_1b_0\} = \{c_7c_6c_5c_4c_3c_2c_1c_0\}$$

con $c_i = (a_i + b_i) mod 2 \quad \forall i :\ 0 \leq i \leq 7$.

[2] L'isomorfismo è semplicemente dato dalla mappatura di ogni *byte*, considerato come sequenza di bit, nel relativo polinomio avente come coefficienti la sequenza stessa dei bit: $(b_7b_6b_5b_4b_3b_2b_1b_0) \longrightarrow \sum_{i=0}^{7} b_i x^i$.

La somma modulo 2 tra bit viene anche chiamata *XOR bit-a-bit* e spesso indicata semplicemente con il simbolo $\oplus$.
Ovviamente si possono utilizzare anche altri tipi di notazione oltre a quella del *byte*. Per esempio, le seguenti espressioni sono equivalenti:

- $(x^7 + x^6 + x^3 + x + 1) \oplus (x^6 + x^5 + x^3 + x^2 + 1) = (x^7 + x^5 + x^2 + x)$ (notazione polinomiale)
- $\{11001011\} \oplus \{01101101\} = \{10100110\}$ (notazione binaria)
- $\{CB\} \oplus \{6D\} = \{A6\}$ (notazione esadecimale).

2.3.2 Moltiplicazione

Definizione 2.5. Dati due *bytes* $\{a_7a_6a_5a_4a_3a_2a_1a_0\}$ e $\{b_7b_6b_5b_4b_3b_2b_1b_0\}$, l'operazione di moltiplicazione viene indicata con il simbolo $\bullet$ ed è definita in notazione polinomiale nel seguente modo:

$$\{a_7a_6a_5a_4a_3a_2a_1a_0\} \bullet \{b_7b_6b_5b_4b_3b_2b_1b_0\} = (\sum_{i=0}^{7} a_i x^i) \cdot (\sum_{i=0}^{7} b_i x^i) mod(m(x))$$

dove $m(x)$ è il polinomio irruducibile di ottavo grado $x^8 + x^4 + x^3 + x + 1$, scelto dai creatori di Rijndael.

In questo modo il risultato, che viene calcolato in forma polinomiale, sarà ridotto modulo $m(x)$ e avrà grado minore di otto. Pertanto è assicurata la possibilità di trasformare il polinomio in una sequenza di bit e nel relativo *byte*.
Si considerino, per esempio, due *bytes* in forma esadecimale $\{57\}$ e $\{83\}$ dove $\{57\} \bullet \{83\} = \{C1\}$. In forma polinomiale questo prodotto risulta essere:

$$(x^6+x^4+x^2+x+1)(x^7+x+1) = x^{13}+x^{11}+x^9+x^8+x^7+x^7+x^5+x^3+x^2+x+x^6+x^4+x^2+x+1.$$

Trattandosi di polinomi a coefficienti in $\mathbb{Z}_2$ il risultato è dato da

$$x^{13} + x^{11} + x^9 + x^8 + x^5 + x^4 + x^3 + 1.$$

Riducendo ora modulo $m(x)$ si ha che:

$$(x^6 + x^4 + x^2 + x + 1)(x^7 + x + 1) mod(m(x)) = x^7 + x^6 + 1.$$

Ossia $\{11000001\} = \{C1\}$.
La moltiplicazione definita in questo modo ha elemento neutro, ossia $\{01\}$ ed è associativa. Inoltre, essendo $GF(2^8)$ un campo, ogni elemento ha il suo inverso moltiplicativo.

Inverso moltiplicativo

Per ogni elemento di $GF(2^8)$ non nullo in notazione polinomiale è possibile trovare l'inverso moltiplicativo tramite l'algoritmo euclideo esteso come segue. Data il polinomio $b(x)$ e denotando con $b^{-1}(x)$ il suo inverso:

$$b(x)a(x) + m(x)c(x) = 1.$$

Da cui $a(x) \bullet b(x) mod(m(x)) = 1$ che significa:

$$b^{-1}(x) = a(x) mod(m(x)).$$

Un'alternativa è di ricercare l'inverso di un dato *byte* x sfruttando la proprietà del campo finito per cui

$$x^{-1} = x^{254}.$$

2.3.3 Polinomi a coefficienti in $GF(2^8)$

Molto usati in AES sono anche i polinomi a coefficienti in $GF(2^8)$. Nella costruzione e gestione della chiave del cifrario, riccorre più volte l'utilizzo di una notazione che raggruppa in un array quattro *bytes*. Il suddetto array è denotato con *Word* che può essere indicato in forma polinomiale come un polinomio di quattro termini (terzo grado) avente per coefficienti dei *bytes*, perciò elementi di $GF(2^8)$ che vengono a loro volta denotati in forma esadecimale. Si possono illustrare le operazioni di addizione e di moltiplicazione anche per questi polinomi ma in modo differente dai precedenti, poichè bisogna tener conto del fatto che i coefficienti non sono più dei semplici bit ma dei veri e propri *bytes*.
Dati due polinomi a coefficienti in $GF(2^8)$ $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ e $b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$, il polinomio somma $c(x)$ sarà dato dalla seguente operazione:

$$c(x) = a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a_2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0).^3$$

Per la moltiplicazione, invece, è necessario compiere un ulteriore passaggio. É infatti lecito richiedere che il polinomio prodotto abbia lo stesso grado e forma dei due fattori. Questo permette di ottenere una nuova *Word* come risultato ed è importante per mantenere l'operazione chiusa rispetto all'anello di polinomi in cui si sta lavorando.

[3] Il simbolo $\oplus$ denota la normale operazione tra *bytes* sopra definita tramite lo *XOR bit-a-bit*.

Dapprima si compie la normale moltiplicazione, che aumenterà il grado del polinomio prodotto rispetto a quello dei due fattori:

$$c(x) = a(x) \cdot b(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0.$$

Dove:

$c_6 = (a_3 \bullet b_3)$
$c_5 = (a_3 \bullet b_2 \oplus a_2 \bullet b_3)$
$c_4 = (a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3)$
$c_3 = (a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3)$.
$c_2 = (a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2)$
$c_1 = (a_1 \bullet b_0 \oplus a_0 \bullet b_3)$
$c_0 = (a_0 \bullet b_0)$

Come volevasi dimostrare il polinomio non ha più solo quattro termini, bensì sette. A questo proposito, si esegue un ulteriore passaggio per completare l'operazione e per ottenere una *Word* come risultato, ossia una riduzione modulare per un polinomio di quarto grado. Nel caso di Rijndael è stato scelto $(x^4 + 1)$. É interessante notare che $(x^4 + 1)$ non è irriducibile in $GF(2^8)[x]$ per questo la moltiplicazione non sarà sempre invertibile. AES però effettua questo genere di operazione solo quando uno dei due fattori è un polinomio fissato. Nello specifico, quindi, fissa sempre dei polinomi che rendono possibile l'inversione. Un esempio è quello usato nella funzione *Mix Columns*, che verrà illustrata nel dettaglio nel capitolo seguente. Il polinomio scelto è

$$p(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

che ha inverso moltiplicativo

$$p^{-1}(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

Nel caso di moltiplicazione per un polinomio fissato l'operazione (inclusa la riduzione modulare) può essere anche denotata dal simbolo $\otimes$ e ha come riultato un polinomio a quattro termini $d(x) = a(x) \otimes b(x) = d_3x^3 + d_2x^2 + d_1x + d_0$ con

$$d(x) = c(x) mod(x^4 + 1). \tag{2.1}$$

Osservazione 2.1. La forma $d(x) = c(x) mod(x^4 + 1)$ può essere vista in forma matriciale come

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

dove $a(x)$ e $b(x)$ rispettano le notazioni precedenti.

Dimostrazione. Considerando (2.1) e basandosi sulla seguente equazione

$$x^i mod(x^4+1) = x^{imod4}$$

si può notare che il polinomio $d(x)$ può essere descritto in questo modo:

$$\begin{aligned} d(x) = c(x)mod(x^4+1) &= [c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0]mod(x^4+1) \\ &= c_3x^3 + (c_2 \oplus c_6)x^2 + (c_1 \oplus c_5)x + (c_0 \oplus c_4). \end{aligned}$$

Da qui segue, esplicitando i coefficienti c_i

$$\begin{aligned} d_0 &= a_0 \bullet b_0 \oplus a_3 \bullet b_1 \oplus a_2 \bullet b_2 \oplus a_1 \bullet b_3 \\ d_1 &= a_1 \bullet b_0 \oplus a_0 \bullet b_1 \oplus a_3 \bullet b_2 \oplus a_2 \bullet b_3 \\ d_2 &= a_2 \bullet b_0 \oplus a_1 \bullet b_1 \oplus a_0 \bullet b_2 \oplus a_3 \bullet b_3 \\ d_3 &= a_3 \bullet b_0 \oplus a_2 \bullet b_1 \oplus a_1 \bullet b_2 \oplus a_0 \bullet b_3. \end{aligned}$$

e quindi, in forma matriciale, l'assunto

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

□

Capitolo 3

Cifratura e decifrazione di AES

3.1 Cifratura di AES

I due designers Joan Daemen e Vincent Rijmen proposero e progettarono l'algorimo Rijndael in modo tale che rispettasse tre caratteristiche fondamentali:

- Resistenza contro tutti gli attacchi noti
- Velocità e compattezza del codice su un'ampia gamma di piattaforme
- Semplicità progettuale

Proposero quindi un cifrario a blocchi con lunghezza del blocco e della chiave indipendenti l'una dall'altra. AES in particolare può avere chiave con lunghezza variabile di 128, 192 o 256 bit, ma blocco di lunghezza fissa di 128 bit. La descrizione del cifrario in questo elaborato è pensata in modo particolare per il sistema con chiave a 128 bit, in quanto probabilmente la più rilevante, poiché usata più su ampia scala rispetto alle altre.

Definizione 3.1. Il risultato intermedio delle operazioni fatte durante l'algoritmo prende il nome di *State*, rappresentabile come un array di bytes.

Subito dopo aver copiato il blocco da 128 bit di testo in chiaro in *State*, dividendolo in sedici 8-bit bytes da porre in una matrice 4x4, comincia la codifica, che consiste fondamentalmente in dieci *rounds* (fasi). Ogni *round* è composto da quattro operazioni:

- Substitute Bytes
- Shift Rows

- Mix Columns
- Add Round Key

Eccezione è l'ultimo round, in cui non compare l'operazione di *Mix Columns*. Tutte le operazioni vengono effettuate in sequenza all'interno di ogni round, tranne un'iniziale *Add Round key* al di fuori di ogni fase. Ora verrà mostrato in dettaglio il funzionamento di ogni operazione e della sua inversa. AES infatti ha un sistema di decifrazione che funge da inverso della cifratura con lo stesso numero di fasi, operazioni inverse e disposte con un altro ordine. Si considera ora nei particolari la composizione di un round dell'algoritmo.

3.1.1 Substitute bytes

La SubBytes è una trasformazione (permutazione) non lineare di bytes che vengono mappati tramite una tabella definita in AES stesso. La tabella, detta *S-box*, contiene 16x16 bytes e sfrutta l'algebra del gruppo finito GF(2^8), permutando tutti i 256 elementi del suddetto gruppo. La trasformazione utilizza quindi una mappatura su tabella, che diventa in pratica una semplice sostituzione di bytes dell'array *State* con i corrispettivi valori ricercati sulla S-box, da cui il nome *SubBytes*. In questo modo, degli 8 bit che compongono un byte di *State*, si trasformano i quattro più a sinistra e i quattro più a destra (che verranno chiamati rispettivamente *nibble* più significativo e *nibble* meno significativo) in due valori da 0 a 15. Il *nibble* più significativo determinerà la riga della S-box e il *nibble* meno significativo la colonna; si avrà, così, un nuovo valore che sostituirà il precedente in *State*. Per fare un semplice esempio, il valore A4 deriverà dal byte {10100100}, ossia 164 in GF(2^8), e verrà sostituito dal valore in riga A e colonna 4 della *S-box*, ossia 49 ({01001001} in binario)

Tabella *S-box*

Per costruire la tabella occorrono fondamentalmente due trasformazioni. Dopo aver inizializzato una tabella 16x16 di bytes tale che le righe siano in sequenza ascendente, ossia la prima $00, 01, ..., 0F$, la seconda $10, 11, ..., 1F$ e così via fino all'ultima, assegnando, quindi, ad un generico byte nella riga x e colonna y il valore xy, si compiono le seguenti operazioni:

1. ogni valore della tabella viene sostituito con il suo inverso moltiplicativo in GF(2^8) (00 verrà lasciato invariato).

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	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4B	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

2. a ciascuno degli 8 bit di cui è composto ogni byte della S-box viene applicata una trasformazione lineare tale che:

$$b'_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i \qquad (3.1)$$

dove: b_i è l'i-esimo bit del byte considerato nella *S-box*, b'_i è l'i-esimo bit del byte trasformato e c_i è l'i-esimo bit del byte con valore 63, ossia (01100011).

AES esprime l'operazione sopra citata in forma matriciale in questo modo:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

dove le operazioni tra gli elementi dei vettori sono in $\mathbb{Z}_2$. Per quanto si è detto, segue che l'azione di *SubBytes* su un *byte* x è la seguente:

$$SubBytes(x) = Ax^{-1} + B$$

dove A e B sono le matrici precedenti secondo questa notazione

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

L'unica parte non lineare è quindi la trasformazione $y = x^{-1} = x^{254}$, in $GF(2^8)$. Questa parte è in realtà l'unica parte non lineare di tutto l'algoritmo, ma, come si vedrà in seguito, è sufficiente per rendere il sistema difficilmente attaccabile per via algebrica.

3.1.2 Shift Rows

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} \Longrightarrow \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,2} & a_{2,3} & a_{2,4} & a_{2,1} \\ a_{3,3} & a_{3,4} & a_{3,1} & a_{3,2} \\ a_{4,4} & a_{4,1} & a_{4,2} & a_{4,3} \end{pmatrix}$$

Lo Shift Rows, come si può osservare, è un semplice scorrimento di bytes nell'array *State*. La prima riga rimane invariata, dalla seconda alla quarta viene sempre eseguito uno scorrimento circolare a sinistra di uno, due e tre bytes rispettivamente.

3.1.3 Mix Columns

La trasformazione Mix Columns opera sulla matrice *State* colonna per colonna, considerando ognuna di esse come un polinomio a quattro termini in GF(2^8) e moltiplicandola per un polinomio fissato $c(x) = 03x^3 + 01x^2 + 01x + 02$ modulo $(x^4 + 1)$. Questa operazione può essere vista come una moltiplicazione matriciale per una matrice data

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix} \tag{3.2}$$

dove $a_{i,j}$ con $0 \leq i, j \leq 3$ è la matrice di *State*. Per un'unica colonna di *State* si può esplicitare l'operazione in questo modo

$$
\begin{aligned}
b_{0,j} &= (2 \bullet a_{0,j}) \oplus (3 \bullet a_{1,j}) \oplus a_{2,j} \oplus a_{3,j}, \\
b_{1,j} &= a_{0,j} \oplus (2 \bullet a_{1,j}) \oplus (3 \bullet a_{2,j}) \oplus a_{3,j}, \\
b_{2,j} &= a_{0,j} \oplus a_{1,j} \oplus (2 \bullet a_{2,j}) \oplus (3 \bullet a_{3,j}), \\
b_{3,j} &= (3 \bullet a_{0,j}) \oplus a_{1,j} \oplus a_{2,j} \oplus (2 \bullet a_{3,j}).
\end{aligned}
$$

dove l'operazione $\bullet$ è la moltiplicazione in $\mathrm{GF}(2^8)$ e $\oplus$ è l'operazione XOR bit-a-bit, che in $\mathrm{GF}(2^8)$ corrisponde alla somma.

3.1.4 Add Round Key

Questa non è altro che la fase in cui viene inserita la chiave segreta che rende il cifrario sicuro (le altre trasformazioni sono note, per cui facilmente invertibili). La chiave da 128 bit (16 bytes) viene inserita con una semplice operazione di somma bit-a-bit modulo 2 a *State*, similmente al modo in cui viene utilizzato in un cifrario di Vernam.[1] Ad ogni round la chiave aggiunta è diversa e ricavata dalle precedenti ricorsivamente.

3.1.5 Gestione della chiave

La chiave segreta del cifrario viene gestita principalmente in due passaggi, prima di essere aggiunta a *State* nella fase finale del *Round*. Dapprima viene espansa in una chiave che, alla fine di tutta la cifratura, risulterà avere un numero di bit pari al numero di bit di un singolo blocco moltiplicato per il numero di round dell'algoritmo più uno $(128 * (10 + 1) = 1408bit)$, ossia ben 176 bytes organizzati in un array lineare di 44 *Words*.[2] In secondo luogo si compie una scelta per determinare quale sia la parte di chiave da usare nel round corrente all'interno dell'array di *Word*.

Espansione della chiave

Una volta inizializzato l'array di *Word* $W[i]$ per $0 \leq i \leq 43$ e copiata nelle prime quattro *Word* la chiave stessa, ad ogni chiamata della funzione si aggiungono quattro *Word* all'array. Esse vengono aggiunte ricorsivamente

[1] Cifrario a chiave privata lunga quanto il testo stesso generata casualmente e poi aggiunta vettorialmente al testo per ottenere la cifratura

[2] array colonna da quattro *bytes* ciascuno

ed ognuna dipende dalla precedente e da quella che si trova quattro posizioni prima.

$$W[i] = W[i-1] \oplus W[i-4]$$

Unica eccezione risultano essere quelle che si trovano ad una posizione i multipla di quattro. In questo caso si ha un riguardo particolare e si applicano delle funzioni aggiuntive non lineari, diverse dal solito XOR bit-a-bit. Esistono, infatti, due funzioni che vengono applicate alla *Word* precedente a quella considerata; in seguito viene eseguita un operazione XOR bit-a-bit tra il risultato ottenuto ed una costante di fase che dipenderà, quindi, dalla posizione in cui si trova la *Word* presa in esame nell'array.

$$Subword(Rotword(w[i-1])) \oplus Rcon[i/4]$$

Dove:

- Subword è una sottofunzione che utilizzando la S-box di AES (componente non lineare) sostituisce ogni byte della *Word* in input,
- Rotword è una sottofunzione che compie uno scorrimento circolare a sinistra di un byte sulla *Word* (simile allo scorrimento di *Shift rows*),
- Rcon è una costante di fase aggiunta al risultato delle due sottofunzioni.

Di seguito si può osservare lo pseudocodice dell'algoritmo della funzione di espansione della chiave.

```
KeyExpansion(byte Key[16] word W[44])
{
   for(i = 0; i < 4; i++)
      W[i] = (key[4*i],key[4*i+1],key[4*i+2],key[4*i+3]);
   for(i = 4; i < 44); i++)
   {
      temp = W[i - 1];
      if (i mod4 == 0)
         temp = SubWord(RotWord(temp)) +  Rcon[i / 4];
      else if (i mod4 == 0)
         temp = SubByte(temp);
      W[i] = W[i - 4] + temp;
   }
}
```

3.2 Decifrazione di AES

AES non è un cifrario di *Feistel*, al contrario del precedente standard DES; pertanto, la cifratura e la decifrazione utilizzano due algoritmi diversi. Tuttavia, entrambi sono composti da 10 *Round* e la programmazione della chiave con la funzione di estensione sopra descritta viene svolta allo stesso modo. Nonostante ciò, ogni funzione della cifratura, eccettuato l'*AddRoundKey* che rimane invariato, ha una sua funzione inversa:

- Inverse sub bytes
- Inverse shift rows
- Inverse mix columns

Un altro cambiamento nella decifrazione lo si può riscontrare nell'ordine delle funzioni all'interno di un singolo round. La prima trasformazione eseguita sarà *InvShiftRows*, inversa della funzione Shift Rows sopra descritta; a seguire si trovano *InvSubBytes*, *AddRoundKey* e, infine, *InvMixColumns*. Si osservi ora lo pseudocodice dell'algoritmo di decifrazione che mette in luce le differenze con l'algoritmo precedente di cifratura.

```
InvCipher(byte in[16], byte out[16], word w[44])
{
   byte state[4,4]
   state = in
   AddRoundKey(state, w[40, 33])
   for (round = 9 ; round >= 1 ; round--)
   {
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*4, (round+1)*3])
      InvMixColumns(state)
   }
   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, 3])
   out = state
}
```

3.2.1 Trasformazione Shift Rows inversa

La trasformazione inversa dello Shift Rows applica semplicemente alla matrice *State* uno scorrimento circolare nelle righe allo stesso modo della

corrispettiva funzione di cifratura, ma nella direzione opposta, ossia verso destra. In questo modo è molto semplice dimostrare che *InvShiftRows* è l'inversa di *ShiftRows*

3.2.2 Trasformazione Mix Columns inversa

La funzione è definita dalla seguente moltiplicazione tra matrici, similmente alla sua corrispettiva diretta.

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

É facile dimostrare che questa sia l'inversa di *MixColumns* poichè

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inoltre, come già affermato nella sezione della trasformazione Mix Columns, vi è un'equivalenza tra il prodotto matriciale sopra riportato ed il prodotto di ciascuna colonna di *State* per un polinomio fissato modulo (x^4+1). Anche in questo caso si può considerare questo tipo di operazione con il polinomio dato

$$b(x) = 0Bx^3 + 0Dx^2 + 09x + 0E$$

dove è facile dimostrare che $b(x) = c(x)^{-1} mod(x^4+1)$, con $c(x)$ il polinomio usato in *MixColumns*.

3.2.3 Trasformazione Substitute Bytes inversa

Funziona esattamente allo stesso modo della sua corrispettiva diretta. L'unica differenza risiede nella *S-box*. In questa funzione si usa infatti una tabella che definisce la permutazione inversa a quella usata in *SubBytes*.

Tabella S-box inversa

La S-box inversa viene costruita applicando l'inversa della trasformazione applicata per la S-box seguita dall'inversa moltiplicativa in GF(2^8). Questa trasformazione inversa è:

$$b'_i = b_i \oplus b_{(i+2)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+7)mod8} \oplus d_i$$

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

con $d = 05$, ossia (00000101). AES rappresenta tale applicazione in forma matriciale come visto sopra:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Equivalentemente a quanto detto per la corrispettiva funzione in cifratura, preso un qualsiasi *byte* y:

$$InvSubBytes(y) = (A^{-1}y - B)^{-1} = (A^{-1}y - B)^{254}$$

dove A e B sono rispettivamente la matrice e il vettore utilizzati nella precedente equazione.

3.2.4 Cifratura inversa equivalente

Come risulta evidente dalla descrizione dell'algoritmo, cifratura e decifrazione di AES non sono uguali. Questo, come verrà detto anche in seguito,

rappresenta uno svantaggio dal punto di vista implementativo. Tuttavia, esiste una versione equivalente all'algoritmo di decrittografia, che utilizza sempre le funzioni inverse della cifratura ma nello stesso ordine di quest'ultima. Considerando che l'ordine nella cifratura è *SubBytes, ShiftRows, MixColumns* e *AddRoundKey* mentre nella decifrazione è *InvShiftRows, InvSubBytes, AddRoundKey* e *InvMixColumns*, per avere un algoritmo di decifrazione equivalente a quello di cifratura sarà necessario scambiare le prime due trasformazioni tra loro così come le seconde due.

Scambio tra *InvShiftRows* e *InvSubBytes*

InvShiftRows altera la sequenza dei *bytes* di *State* ma non il suo contenuto, mentre *InvSubBytes* ne altera il contenuto ma non la sequenza dei *bytes*. Per questa ragione, è assolutamente indifferente l'ordine delle funzioni e, quindi, sono scambiabili. Infatti per un determinato *State* S_i:

$$InvShiftRows[InvsubBytes(S_i)] = InvSubBytes[InvshiftRows(S_i)].$$

Scambio tra *AddRoundKey* e *InvMixColumns*

Considerando la chiave come una sequenza di *Word* allora entrambe le funzioni operano su *State* colonna per colonna. Per poterle scambiare è necessario osservare la linearità che le caratterizza. Infatti per un determinato *State* S_i e una determinata chiave di fase w_j si ha:

$$InvMixColumns(S_i \oplus w_j) = [InvMixColumns(S_i)] \oplus [InvMixColumns(w_j)]$$

Quindi, per poter applicare lo scambio, basta applicare *InvMixColumns* alla chiave di fase prima di aggiungerla allo *State* corrente. In questo modo l'algoritmo di decrittografia risulta strutturato equivalentemente all'algoritmo di crittografia con un'unica accortezza: applicare *InvMixColumns* alla chiave di fase prima di aggiungerla.

Capitolo 4

AES e sicurezza: cenni

Come si è potuto constatare, all'interno dell'algoritmo vengono utilizzate diverse funzioni e costanti matematiche. Ognuna di queste è stata implementata per rendere l'algoritmo il più efficiente e sicuro possibile, in accordo con i termini del NIST. Per quanto riguarda la sicurezza, i criteri usati dall'istituto per valutare gli algoritmi candidati a diventare il nuovo standard fanno riferimento all'impegno necessario ad analizzare crittograficamente un algoritmo e alle possibilità pratiche di un attacco. La valutazione finale fu condotta dalla comunità dei crittografi che diedero particolare enfasi all'analisi della robustezza nei confronti dei *known plaintext attacks*,[1] per esempio di tipo differenziale e lineare. Il NIST si pronunciò così sull'algoritmo Rijndael: *"Non esiste alcun attacco noto alla sicurezza di Rijndael. Rijndael utilizza una S-box come componente non lineare. Rijndael sembra avere un margine di sicurezza adeguato, ma ha ricevuto qualche critica che suggerisce che la sua struttura matematica potrebbe essere soggetta ad attacchi. D'altro canto, la semplicità della sua struttura dovrebbe aver facilitato l'analisi della sicurezza durante il processo di sviluppo dello standard AES."*
Di seguito si potrà vedere più nel dettaglio perchè i due crittografi Deamen e Rijmen progettarono l'algoritmo utilizzando proprio quelle funzioni e quelle costanti e in che modo quelle scelte ne accrebbero la sicurezza.

4.1 Scelte algoritmiche e loro motivazioni

Substitute Bytes e S-box

In generale esistono diverse *S-box* che renderebbero Rijndael sicuro. Infatti durante la creazione della tabella si potrebbero applicare mappature

[1] attacchi in cui sono note una parte di testo in chiaro e la relativa parte di testo cifrato

differenti da quella scelta e si otterrebbe lo stesso risultato, ossia la minor correlazione possibile tra bit di input e bit di output. In pratica viene scelta, però, una sola di esse: la mappatura nell'inverso moltiplicativo in $GF(2^8)$. Definire la tabella *S-box* tramite questa funzione dona all'algoritmo maggiore semplicità di esecuzione e allo stesso tempo diventa l'unica parte non lineare del cifrario, ossia quella che rende il cifrario difficilmente accessibile per via algebrica. Inoltre la costante dell'equazione (2.1) è stata selezionata in modo tale che la *S-box* non abbia alcun punto fisso e nessun punto fisso opposto.[2] Un'ovvia caratteristica della tabella è la sua invertibilità. Grazie a questo è possibile costruire la tabella inversa come già mostrato nel paragrafo (2.2.3). Inoltre, la scelta della funzione $x \mapsto x^{-1}$ è dovuta alle caratteristiche favorevoli che possiede: una ˝non linearità˝elevata e, quindi, una non facile approssimabilità con funzioni lineari. Nella loro proposta al NIST, Daemen e Rijmen osservarono che l'azione della *S-box* nei *bytes* b_{ij} poteva essere scritta come segue ([3]):

$$S - box(b_{ij}) = \lambda_8 + \sum_{d=0}^{7} (\lambda_d b_{ij}^{255-2d}).$$

Mix Columns

La funzione *Mix Columns* gioca un ruolo fondamentale nella dispersione fra i byte di ciascuna colonna, generata da un codice lineare su cui si basano i coefficienti dell'equazione (2.2). La scelta della riduzione modulo $(x^4 + 1)$ è rilevante dal punto di vista aritmetico, poichè, come già spiegato, è necessario mantenere il risultato in forma di *Word* (array di 4 *bytes*). In questo modo vengono anche rispettati diversi criteri che rendono la funzione stessa più efficiente quali:

- linearità in $GF(2^8)$
- simmetria
- semplicità descrittiva.

Infine, la scelta dei coefficienti molto piccoli è importante dal punto di vista dell'implementazione. Con tali coefficienti si rende infatti l'algoritmo più veloce e meno complesso. Un'ulteriore motivo che ha portato alla scelta di tali coefficienti è la necessità che la funzione sia invertibile. Come già mostrato nel paragrafo (2.3.3), la riduzione modulare per $(x^4 + 1)$ non dà

[2] $(S - box)(a) = a$ oppure $(S - box)(a) = \bar{a}$ con $\bar{a}$ il complemento bit-a-bit di a

origine ad un campo, quindi era necessario scegliere un polinomio che fosse invertibile; per tutti questi motivi il polinomio fissato risulta essere alla fine

$$\{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}.$$

Shift Rows

La *Shift Rows*, combinata con *Mix Columns*, è fondamentale per la dispersione dei byte. Dopo pochi rounds infatti, tutti i bit di output dipendono interamente dai bit di input.

Espansione della chiave

L'algoritmo di espansione della chiave è stato formulato in base ai seguenti criteri:

1. la conoscenza di una parte della chiave di cifratura o della chiave di round non consente di calcolare molti altri bit della chiave di base;

2. la trasformazione è invertibile, ovvero la conoscenza di quattro *Word* consecutive della chiave espansa consente la rigenerazione dell'intera chiave espansa;

3. uso di costanti di round per eliminare le simmetrie;

4. diffusione delle differenze della chiave di cifratura nelle chiavi di round; ovvero ciascun bit della chiave influenza più bit della chiave di round;

5. sufficiente non-linearità da impedire la determinazione completa delle differenze della chiave di round utilizzando solo le differenze della chiave di cifratura;

6. facilità di descrizione.

Numero di Round

Il numero di Round è determinato in modo tale da rendere impossibile un attacco di forza bruta, ossia un attacco basato sulla ricerca esaustiva della chiave. Il numero 10 è in realtà un margine di sicurezza, poichè per il cifrario a blocchi da 128 bit non sono possibili attacchi di questo tipo già con più di 6 rounds. Le ragioni di questa scelta sono principalmente due:

1. all'algoritmo Rijndael sono sufficienti due soli rounds per ottenere una "diffusione completa "dei *bytes*, grazie al fatto che i bit di *State* dipendono dai bit dello *State* di due rounds precedenti a quello corrente. In questo senso, aggiungere quattro rounds significa avere una "diffusione completa"di *bytes* all'inizio e al termine dell'algoritmo. Ovviamente, questa diffusione così veloce è dovuta a tutte le funzioni sopra citate, che fanno differire notevolmente AES da un comune cifrario di *Feistel*. Infatti, quest'ultimo, ha una diffusione più lenta causata dalla modalità di esecuzione dell'algoritmo che lavora su una metà del blocco alla volta.

2. attacchi di crittografia lineare e differenziale sfruttano le correlazioni tra output e input di n round per attaccare un cifrario di $n+1$ o $n+2$ round. Ma già sfruttando le correlazione di 4 round è difficile attaccare il cifrario per questa via, in più sono stati aggiunti ulteriori 4 round ai 6 che ne avrebbero garantito la sicurezza per raddoppiare la difficoltà dell'attacco che da 4 passa a 8 round.

Infine per le versioni di AES con chiave più lunga il numero di round accresce di uno per ogni 32 bit addizionali nella chiave del cifrario.

4.2 Resistenza contro i *known attacks*

4.2.1 Crittoanalisi lineare e differenziale

La crittoanalisi differenziale è di solito un attacco con testo in chiaro scelto. Il metodo base utilizza coppie di testo in chiaro relazionate da una costante detta differenza: la differenza può essere definita in vari modi, ma l'operazione più usata è generalmente lo XOR. L'attaccante può calcolare le differenze dei corrispondenti testi cifrati, sperando di intercettare delle regolarità statistiche nella loro distribuzione. La coppia delle differenze tra input e output è detta un differenziale. Le proprietà statistiche dei differenziali dipendono dalla natura delle S-box usate per la cifratura. L'attacco permette di recuperare la chiave più velocemente della ricerca esaustiva di tutte le possibili combinazioni. Dato che la crittoanalisi differenziale è divenuta di pubblico dominio, è divenuta anche una preoccupazione di base dei crittografi: ci si aspetta infatti che i nuovi sistemi siano accompagnati da un'analisi da cui si evinca che l'algoritmo è resistente a questo tipo di attacco, e per molti di essi, incluso AES, è stato matematicamente provato che lo sono. Infatti, per poter effettuare un attacco di crittoanalisi differenziale, è necessaria una

prop ratio[3] significativamente superiore a 2^{1-n}. Per AES è provato che non esistono *4-round differential trails*[4] con *prop ratio* superiore a 2^{150}, perciò è resistente all'attacco.
Un attacco di crittoanalisi lineare, invece, si basa principalmente su due fasi. La prima consiste nel trovare delle equazioni lineari che leghino testo in chiaro, testo cifrato e bit della chiave. In un cifrario reale queste equazioni lineari potrebbero cambiare di passaggio in passaggio e quindi sono variabili, per questo si parla spesso di correlazione lineare tra input e output e approssimazione lineare. Ovviamente le approssimazioni vengono costruite in maniera differente da cifrario a cifrario, però, più comunemente, l'analisi si concentra sulla *S-box*, poichè è l'unica parte del cifrario non lineare. Una volta ottenuta un'approssimazione lineare, è possibile applicare un semplice algoritmo usando coppie note di testo in chiaro e testo cifrato per prevedere i valori dei bit della chiave coinvolti nell'approssimazione.
Per far sì che un attacco di crittoanalisi lineare sia possibile, tutte le *linear trails* devono avere un indice di correlazione molto alto, tale che la somma di tutti sia significativamente maggiore a $2^{n/2}$ (dove n è la lunghezza del blocco). Per AES è dimostato che non esistono *4-round linear trails* con coefficiente di correlazione maggiore a 2^{75}, perciò è resistente all'attacco.

4.2.2 Altri tipi di attacchi

Esistono altri tipi di *known attacks* eseguibili contro i cifrari a blocchi: di seguito ne verranno citati alcuni.

- **Differenziale troncato**: normalmente nei cifrari le differential trails tendono a raggrupparsi (*clustering*). Se, per certi settaggi di input difference pattern e output difference pattern, il numero di differential trail è estremamente alto, il fenomeno di clustering diventa più evidente. Per questo la probabilità che una differential trail si possa trovare nel bordo del raggruppamento può essere calcolata indipendentemente dalla prop ratio delle singole differential trails. Questo tipo di attacco è forte negli algoritmi in cui vengono fatte operazioni direttamente nello *State*. Nonostante questo, Rijndael è resistente anche in questo caso. Infatti dopo sei round è provato essere impossibile applicare un attacco che non sia la ricerca esaustiva della chiave.

[3] L'ammontare relativo di tutte le coppie di input che per una data differenza danno origine ad un differenziale in output

[4] Con *differential trails* si intende il percorso e il modo in cui si modifica un differenziale lungo il cifrario dall'input all'output

- **Attacco dell'interpolazione**: attacco molto utile contro cifrari che fanno uso di espressioni algebriche. Si basa sul fatto che qualora vengano usati polinomi con basso grado bastano poche coppie di testo in chiaro e relativo testo cifrato per stabilire i coefficienti del polinomio da cui dipende la chiave. Per Rijndael è difficilmente applicabile a causa della complessità del polinomio interpolatorio, che rappresenta la *S-box*, in $GF(2^8)$. La *S-box* di AES può essere rappresentata nel seguente modo ([1] ¨Submission di AES˝):

  $$63+8Fx^{127}+B5x^{191}+01x^{223}+F4x^{239}+25x^{247}+F9x^{251}+09x^{253}+05x^{254}$$

- ***The Square attack***: lo *Square attack* di Lars Knudsen viene chiamato così poichè è stato creato appositamente per l'algoritmo *Square*. Quest'ultimo era un cifrario a blocchi basato su una rete di sostituzioni e permutazioni creato da Daemen e Rijmen; Rijndael, infatti, riprende molte proprietà dallo *Square*. Per questo lo *Square attack* potrebbe essere applicato ad AES e, infatti, è un'ottima alternativa alla ricerca esaustiva della chiave fino a sei rounds. Purtroppo però, è dimostrato che dal settimo round in poi è impossibile applicare l'attacco in un tempo accettabile.

Bibliografia

[1] Joan Daemen and Vincent Rijmen, AES submission document on Rijndael, June 1998.

[2] Federal Information Processing Standards Publication 197, Announcing the ADVANCED ENCRYPTION STANDARD (AES), November 26, 2001

[3] Susan Landau, Polynomials in the Nation's Service: Using Algebra to Design the Advanced Encryption Standard, 2004

[4] Harris Nover, Algebraic Cryptanalysis of AES: an Overview

[5] William Stallings, Crittografia e sicurezza delle reti, McGraw-Hill 2004

[6] Wikipedia