UNIVERSITA DEGLI STUDI DI TORINO

FACOLTA' DI SCIENZE MATEMATICHE FISICHE E NATURALI
CORSO DI LAUREA IN MATEMATICA

L' Advanced Encryption Standard

Relatore
Prof. Umberto Cerruti

Candidato
Chiara Capitani

ANNO ACCADEMICO 2005 - 2006

Indice

Introduzione

Scopo di questa tesi è dare una breve ma esaustiva spiegazione del funzionamento dell'algoritmo crittografico Advanced Encryption Standard.
Nei quattro capitoli vedremo, rispettivamente, la "storia" di AES ed i motivi per i quali venne adottato, le basi matematiche richieste per comprendere al meglio l'implementazione e le operazioni dell'algoritmo, la descrizione dettagliata di tutte le operazioni svolte all'interno di AES e, infine, la sua rappresentazione polinomiale.
E' inoltre presente un'appendice utile come ripasso di alcuni concetti utilizzati nella spiegazione delle basi matematiche, quali le definizioni e le proprietà principali riguardanti anelli, campi e polinomi.

Capitolo 1

L'Advanced Encryption Standard

In Crittografia, l'Advanced Encryption Standard (AES), conosciuto anche come Rijndael, è un algoritmo di cifratura a blocchi utilizzato come standard dal governo degli Stati Uniti d'America. Data la sua sicurezza e le sue specifiche pubbliche si presume che in un prossimo futuro venga utilizzato in tutto il mondo come è successo al suo predecessore, il Data Encryption Standard (DES)[1]. AES è stato adottato dall'agenzia del dipartimento del commercio National Institute of Standards and Technology (NIST), che ha il compito di approvare gli standard federali per il governo degli Stati Uniti, e dalla US FIPS PUB 197 nel novembre del 2001 dopo 5 anni di studi e standardizzazioni.
L'algoritmo è stato sviluppato da due crittografi Belgi, Joan Daemen e Vincent Rijmen, che lo hanno presentato al processo di selezione per l'AES con il nome di "Rijndael", nome derivato dai nomi degli inventori.
Rijndael è un'evoluzione del primo algoritmo sviluppato da Daemen e Rijmen, Square.
A differenza del DES, Rijndael è una rete a sostituzione e permutazione, non una rete di Feistel[2]. AES è veloce sia se sviluppato in software sia se sviluppato in hardware, è relativamente semplice da implementare, e richiede poca memoria.

1.1 Le origini dell'AES

Per molti anni il predecessore dell'AES, il Des, rappresentò lo standard per la cifratura e l'autenticazione di documenti. Il National Institute of Standard and Technology (NIST) lo scelse come standard di cifratura nel 1977 con validità quinquennale. Fu riaffermato nel 1983, nel 1988 ed infine nel 1993 con una clausola che ne confermava la validità fino al dicembre del 1998. La clausola specificava che, allo scadere di tale data, il NIST avrebbe dovuto prendere in considerazione alternative che offrissero un maggior grado di sicurezza.
Sin dall'inizio il DES fu oggetto di molte critiche che misero in discussione la sua forza crittografica; in riferimento alla sicurezza dello schema e a dispetto delle 256 chiavi necessarie per un attacco brutale, la metà di esse, in media, ne permetteva la violazione. Ciò portò alla considerazione che la lunghezza della chiave era troppo piccola.
In riferimento all'utilizzo delle S-Box[3] nelle procedure di cifratura e decifratura si sospettava che queste strutture nascondessero delle trapdoor[4].
In vista della data di scadenza della validità del DES ed a causa di tali critiche , il NIST considerò alternative che offrissero un livello di sicurezza maggiore; una di queste, il triplo DES, venne considerato un valido sostituto, ma il NIST volle selezionare come nuovo standard un algoritmo di cifratura più sicuro ed efficiente. A tal fine, il 12 settembre del 1997, indisse un concorso pubblico per la nomina dell'Advanced Encryption Standard: l'obiettivo fondamentale del NIST era quello di stabilire un nuovo standard che diventasse un punto di riferimento, nel secolo successivo, nel campo della sicurezza. Ai partecipanti fu richiesto un completo pacchetto di documentazione che contenesse quanto segue:

i. una completa descrizione dell'algoritmo, comprendente tutte le equazioni matematiche, le tabelle, i diagrammi ed i parametri necessari all'implementazione dell'algoritmo;
ii. una stima dell'efficienza computazionale, comprendente le seguenti informazioni:
    - una descrizione della piattaforma utilizzata per generare tale stima;
    - un'analisi dell'algoritmo rispetto agli attacchi di crittoanalisi più conosciuti;

- un elenco dei vantaggi e dei limiti dell'algoritmo;
- un'implementazione di riferimento in ANSI C[5] con appropriati commenti;
- un'implementazione ottimizzata dall'algoritmo in ANSI C e Java[6].

1.2 Requisiti per l'AES

Il NIST richiese per gli algoritmi candidati i seguenti requisiti:

i. l'algoritmo doveva implementare un cifrario a chiave simmetrica;
ii. l'algoritmo doveva appartenere alla classe dei cifrari a blocchi;
iii. l'algoritmo doveva supportare chiavi con taglia compresa tra 128 e 256 bit e lunghezza del testo in chiaro di 128 bit (era preferibile che l'algoritmo fosse in grado di gestire ulteriori taglie)
iv. la struttura dell'algoritmo doveva essere tale da permetterne l'implementazione su smart-card[7];
v. l'algoritmo doveva essere disponibile a livello mondiale, senza esclusive.

Ognuno dei progetti dei candidati venne sottoposto ad una serie di test ed analisi atti a valutare le caratteristiche computazionali e di sicurezza. I criteri di valutazione scaturirono dai commenti pubblici e dalle discussioni tenutesi il 15 aprile 1997, presso il NIST.
A tal proposito si individuarono tre categorie principali che riportiamo di seguito:

i. Sicurezza
La sicurezza rappresentò il fattore più importante della valutazione. Ogni algoritmo, per essere definito minimamente sicuro, doveva essere sottoposto alle seguenti prove:
   - la sicurezza dell'algoritmo doveva essere confrontata con quella degli altri candidati;
   - l'output dell'algoritmo doveva essere indistinguibile da una permutazione casuale del blocco in input.

ii. Costo
L'algoritmo candidato doveva avere una serie di attributi che ne determinassero un basso costo computazionale. Tale costo fu valutato in base ai seguenti criteri:
   - efficienza computazionale: la valutazione dell'efficienza computazionale dell'algoritmo doveva riferirsi sia all'implementazione hardware che software;
   - requisiti di memoria: si doveva considerare la grandezza del codice e la memoria richiesta.

iii. Algoritmo e caratteristiche di implementazione
La terza area di valutazione riguardò l'algoritmo e le caratteristiche implementative, quali la flessibilità, l'adattabilità software ed hardware e la semplicità. L' algoritmo doveva:
   - gestire dimensioni della chiave e dei blocchi superiori alle taglie richieste;
   - essere implementato in modo sicuro ed efficiente in diversi ambienti;
   - essere implementato come uno stream cipher[8], un algoritmo di hashing[9] e fornire servizi crittografici addizionali.

1.3 Processo di selezione dell'AES

Il 2 gennaio 1997 il NIST, propose un bando per la realizzazione del nuovo algoritmo di cifratura a blocchi che il governo americano avrebbe utilizzato come standard per la cifratura dei dati sensibili e non classificati e indisse quindi un concorso pubblico per la nomina dell' AES, sollecitando la comunità crittografica mondiale per la presentazione di nuovi algoritmi crittografici.
Il 20 Agosto 1998, in California, si tenne la prima conferenza per la candidatura all' AES, durante la quale, il NIST annunciò i 15 candidati ufficiali, elencati nella
seguente tabella:

Algoritmo	Autori
Mars	IBM
RC6	RSA Laboratories
RIJNDAEL	Joan Daemen,Vincent Rijmen
SERPENT	Ross Anderson,Eli Biham,Lars Knudsen
TWOFISH B.	Schneider,J. Kelsey,D. Whiting,D. Wagner, C. Hall,N., Ferguson
CAST-256	Entrust Technologies,INC.
CRYPTON	Future System,INC.
DEAL	R.Outerbridge,L.Knudsen
DFC	CNRS
E2	Nippon Telegraf and Telephone Corp.
FROG	TecApro International S.A.
HPC	L.Brown,J.Pieprzyk,J.Seberry
LOKI97	L.Brown,J.Pieprzyk,J.Seberry
MAGENTA	Detsche Telekom AG
SAFER+	Cylink Corp.

Questi algoritmi furono sottoposti alla comunità crittografica mondiale e, in questa conferenza, il NIST sollecitò commenti pubblici sui candidati. Una seconda conferenza si tenne il 22 Marzo 1999 per discutere i risultati delle analisi condotte dalla comunità crittografica mondiale sugli algoritmi candidati. Il periodo dello scrutinio pubblico riguardante le recensioni iniziali si concluse il 15 Aprile del 1999. Utilizzando le analisi ed i commenti ricevuti, il NIST selezionò 5 finalisti dei 15 algoritmi presentati, qui elencati nella seguente tabella:

Algoritmo	Autori
Mars	IBM
RC6	RSA Laboratories
RIJNDAEL	Joan Daemen,Vincent Rijmen
SERPENT	Ross Anderson,Eli Biham,Lars Knudsen
TWOFISH B.	Schneider,J. Kelsey,D. Whiting,D. Wagner, C. Hall,N., Ferguson

Dopo l'annuncio dei finalisti, il NIST aprì formalmente il processo di valutazione pubblico accettando fino al 15 Maggio 2000 i commenti sugli algoritmi rimasti in gara. Il NIST ricercò attivamente commenti ed analisi su qualsiasi aspetto degli algoritmi candidati.

Il 13 e 14 Aprile 2000 a New York il NIST sponsorizzò la Third AES Candidate Conference, un forum pubblico, aperto alla discussione delle analisi sui finalisti. Gli autori degli algoritmi finalisti furono invitati ad assistere e partecipare alla discussione riguardante i commenti sui propri algoritmi. Dopo il 15 Maggio 2000, il NIST studiò tutte le informazioni disponibili e il 2 Ottobre 2000 selezionò Rijndael come algoritmo da proporre per l'AES.
Nel Febbraio del 2001 l'AES venne presentato come un Draft-Federal Information Processing Standard (Draft-FIPS) e pubblicato per commenti e critiche. Terminato il periodo di 90 giorni di discussione nel Maggio del 2001, dopo averlo corretto in modo appropriato in risposta ai commenti, il NIST iniziò una fase di recensione, pubblicazione e diffusione.
Dopo questa fase intermedia, il Segretario del Dipartimento del Commercio americano, il 26 Novembre 2001, approvò in via definitiva Rijndael quale nuovo standard AES (con FIPS PUB 197). Lo standard prevede una taglia del blocco di input di 128 bit, mentre per la chiave di cifratura prevede tre possibili lunghezze 128, 192 e 256 bit (rispettivamente AES-128, AES-192 e AES-256).
Lo standard divenne effettivo il 26 Maggio del 2002, sei mesi dopo l'approvazione.

1.4 Sicurezza ed attacchi

AES è attualmente lo standard di crittazione utilizzato dal governo degli USA per proteggere documenti classificati SECRET, crittati con chiave a 128 bit, e TOP SECRET, crittati con chiave a 192 o 256. Infatti ad oggi non sono noti attacchi ad AES andati a buon fine né è stata evidenziata alcuna chiave debole o semi debole come per il DES. Gli unici successi sono stati ottenuti da attacchi "parziali", ovvero portati ad implementazioni di AES con un numero di round inferiore a quello previsto dallo standard. I risultati migliori sono stati i seguenti:

- (2000) Ferguson et al.: AES a 7 round su chiave a 128 bit e AES a 9 round su chiave a 256 bit tramite l'attacco Square
- (2003) Jakimoski et al.: AES a 8 round su chiave a 192 bit tramite un attacco di crittanalisi differenziale basato sulle chiavi.

In particolare l'attacco Square è una tipologia di attacco che Rijndael ha ereditato dal suo predecessore, l'algoritmo di cifratura Square appunto, e che è stato documentato dagli stessi autori al momento della presentazione di Rijndael al concorso del NIST. L'attacco Square è un attacco di tipo chosen plaintext, ovvero in cui il crittoanalista può scegliere una serie di testi in chiaro e i rispettivi testi cifrati per poi risalire alla chiave studiando le differenze prodotte dalla cifratura dei diversi testi. Esso si basa sulla struttura orientata al byte di Rijndael, utilizzando degli insiemi di state tutti uguali tranne che per un byte, detto "attivo". Seguendo l'evoluzione della posizione e delle trasformazioni di tale byte, l'algoritmo è in grado di eliminare alcuni valori possibili della chiave riducendo lo spazio di ricerca. L'attacco Square pur essendo valido, rimane comunque un attacco parziale e necessita di un numero di plain text pari a 2^{32}. Alcuni crittografi hanno invece avanzato dubbi sulla sicurezza dell'AES a causa della sua struttura matematica. Infatti a differenza di altri algoritmi, AES possiede una struttura molto semplice e molto ben definita e documentata. Proprio su questo spunto si basa una nuova tipologia di attacco che ha suscitato grande clamore in relazione ad AES, ovvero l'attacco chiamato XSL. L'attacco XSL è un attacco di tipo algebrico virtualmente applicabile a quasi tutti i principali algoritmi di cifratura a blocchi, quindi non soltanto ad AES ma anche ad esempio a SERPENT, ovvero un altro dei cinque algoritmi finalisti del concorso del NIST e in particolare quello riconosciuto come il più sicuro. In particolare XSL sfrutta la struttura matematica di AES, riducendo il problema

di recuperare la chiave di cifratura al problema di risolvere un sistema di equazioni quadratiche multivariate, un sistema che risulta inoltre essere particolarmente sparso. Secondo uno studio del 2002 combinando la tecnica XSL ad un sistema per la rappresentazione concisa di AES (BES), si può arrivare ad una complessità di 2^{100} operazioni. Comunque al di là del clamore suscitato dagli studi che hanno proposto l'attacco XSL, sono stati avanzati numerosi dubbi sull'efficacia di tale tecnica. Inoltre, pur essendo teoricamente corretta, non è al momento implementabile né si sa se lo sarà mai. Ad oggi quindi, considerando che la ricerca esaustiva dovrebbe operare su un numero di possibilità pari a 3.4×10^{38} su chiave a 128 bit, l'AES risulta essere inviolato e rappresenta ancora una delle scelte più sicure nel panorama degli algoritmi di crittografia a chiave privata.

Capitolo 2

Basi Matematiche

2.1 I campi di Galois

Le operazioni presenti in AES sono definite a livello di byte dove i bit vengono interpretati come coefficienti dei polinomi del campo finito $GF(2^8)$, dove GF sta per la locuzione inglese "*Galois Field*". Seguono quindi cenni matematici utili per comprendere al meglio l'implementazione delle operazioni dell'algoritmo.

Definizione. Si dice *campo fondamentale di Galois* il seguente campo di ordine p:

$$(\mathbb{Z}_p,+,\cdot)$$

La particolarità di questo tipo di campi sta nel fatto che la struttura $(\mathbb{Z}_n,+,\cdot)$ è un anello finito, commutativo, unitario; inoltre, è privo di divisori dello zero se e solo se $n=p$ è primo.

Teorema. La caratteristica di un campo finito K è un numero primo.
Dimostrazione. Sia p la caratteristica di un campo finito. Supponiamo, per assurdo, che p non sia primo, cioè $p=rs$, con $1<r,s<p$. Allora risulta $0=pu=ru\cdot su$.
Poiché K ha caratteristica p, si ha $ru\neq 0$ e $su\neq 0$, il che è assurdo, essendo K privo di divisori dello zero.

□

Teorema. Un qualsiasi campo finito $(K,+,\cdot)$ avente caratteristica p contiene un sottocampo $(K',+,\cdot)$ contenente p elementi.
Dimostrazione. Sia u l'unità moltiplicativa del campo. Gli elementi dell'insieme $K'=\{u,2u,3u,...,(p-1)u,pu\}$ sono tutti distinti ed è facile provare che $(K',+,\cdot)$ è un campo, quindi segue la tesi.

□

Il campo K' costruito nella dimostrazione precedente coincide "essenzialmente" con $\mathbb{Z}_p$ (nel senso che esiste un isomorfismo di K' in $\mathbb{Z}_p$). Possiamo allora dire che ogni campo finito K avente caratteristica p contiene il sottocampo $\mathbb{Z}_p=GF(p)$, che prende il nome di *campo base* del campo K.

Teorema (*). Sia $(K,+,\cdot)$ un campo e sia $f(x)\in K[x]$ con $\partial(f(x))>0$. Una classe qualsiasi $[p(x)]$, cioè un qualsiasi elemento di $K[x]/(f(x))$, contiene uno ed uno solo polinomio $r(x)$ che è zero oppure ha grado minore di $f(x)$.

Teorema. Sia K un campo. Se $f(x)$ è un elemento di $K[x]$ (anello dei polinomi a coefficienti in K), con $\partial(f(x))>0$, allora $(K[x]/(f(x)),+,\cdot)$ è un anello commutativo con unità ed è un campo se e solo se $f(x)$ è irriducibile in $K[x]$.

Esempio. Consideriamo l'anello dei polinomi a coefficienti reali, cioè $\mathbb{R}[x]$. Il polinomio $f(x) = x^2 + 1$ è irriducibile in $\mathbb{R}[x]$. Denotiamo con i la classe di resto che contiene x, ossia $[x] = i$. La struttura $\mathbb{R}[x]/(x^2+1)$ è un campo. Ogni elemento del campo può essere espresso come un polinomio in i di grado minore di 2, cioè nella forma $a + ib$, con $a, b \in \mathbb{R}$. Questo è un modo per descrivere il campo dei numeri complessi.

Teorema. Sia $K = GF(p)$ e sia $f(x)$ un polinomio irriducibile di grado n di $K[x]$. Allora il campo $K[x]/(f(x))$ ha p^n elementi.
Dimostrazione. Dal teorema (*) segue che gli elementi di $K[x]/(f(x))$ sono tanti quanti i polinomi $r(x) = r_0 + r_1 x + ... + r_{n-1} x^{n-1}$, essendo gli r_i elementi di $GF(p)$. Poiché ogni r_i, $i = 0,...,n-1$, può assumere p valori distinti, segue che gli elementi di $K[x]/(f(x))$ sono p^n.

□

Definizione. Sia fissato un polinomio $f(x)$ a coefficienti in $K = GF(p)$, irriducibile in $K[x]$ e di grado n. Il campo $K[x]/(f(x))$ si dice *estensione di grado n del campo* K, o *ampliamento algebrico di grado n del campo* K; è un campo di Galois di ordine p^n e si denota con $GF(p^n)$.

Teorema. Il polinomio $x^{q-1} - 1$ ha come radici tutti i $q-1$ elementi non nulli di $GF(q)$.

Corollario (Teorema di Fermat). Ogni elemento α di un campo $K = GF(q)$ soddisfa l'identità $\alpha^q = \alpha$, ovvero è radice dell'equazione $x^q - x$.

Teorema. I $p^n - 1$ elementi non nulli di $GF(p^n)$ formano un gruppo ciclico rispetto alla moltiplicazione.

Definizione. Un generatore del gruppo moltiplicativo $(GF(p^n) - \{0\}, \cdot)$ è detto *elemento primitivo* di $GF(p^n)$.

Teorema. Se α è un generatore di $GF(p^n) - \{0\}$, allora tutti gli altri generatori sono dati dalle potenze α^h, con $0 < h < p^n - 1$ e $(h, p^n - 1) = 1$. Essi sono quindi in numero di $\varphi(p^n - 1)$.

Per completezza enunciamo i seguenti teoremi:

Teorema . Dato un numero primo q ed un numero intero $n > 0$, esiste un campo con q^n elementi, che è il campo di spezzamento del polinomio $x^{q^n} - x$ in $\mathbb{Z}_q[x]$.

Teorema. $x^{q^n} - x$ si fattorizza in modo unico in $F[x]$, $F = GF(q)$, nel prodotto di tutti i polinomi monici irriducibili in $F[x]$ che hanno grado d, dove $d \mid n$.

Dimostrazione. Come prima cosa, partiamo dall'ipotesi seguente: sia $f(x)$ un polinomio monico irriducibile in $\mathbb{Z}_q[x]$, di grado d e tale che $f(x) \mid (x^{q^n} - x)$ e dimostriamo la tesi seguente: $d \mid n$.

Sia β uno zero di $f(x)$; per ipotesi β è anche uno zero di $x^{q^n} - x$.

Poiché $\{zeri\ di\ x^{q^n} - x\} = GF(q^n)$ questo implica che $\beta \in GF(q^n)$, $F \subseteq GF(q^n)$, $F(\beta) \subseteq GF(q^n)$ e quindi $F = GF(q^n)$.

A questo punto, si ha:

$$[F(\beta) : F] = \deg(f(x)) = d\ ,\ [GF(q^n) : F] = n\ .$$

Allora:

$$F \subseteq F(\beta) \subseteq GF(q^n) \Rightarrow \underbrace{[GF(q^n) : F]}_{n} = \underbrace{[GF(q^n) : F(\beta)]}_{m}\underbrace{[F(\beta) : F]}_{d} \Rightarrow$$
$$\Rightarrow dm = n \Rightarrow d \mid n\ .$$

Passiamo alla seconda parte della dimostrazione: partiamo dall'ipotesi seguente: sia $f(x)$ un polinomio monico irriducibile in $F[x]$, di grado d, dove $d \mid n$ e dimostriamo che

$$f(x) \mid (x^{q^n} - x) \text{ in } F[x]\ .$$

Sia β uno zero di $f(x)$. Considero $F(\beta) = GF(q^d)$; abbiamo:

$$[F(\beta) : F] = \deg(f(x)) = d\ .$$

Sappiamo che $\beta^{q^n} = \beta$, quindi β è uno zero di $x^{q^d} - \lambda$, allora:

$$\beta^{q^n} = \beta \text{ perché } d \mid n \Rightarrow \beta \text{ è zero di } x^{q^n} - x \Rightarrow f(x) \mid (x^{q^n} - x)\ .$$

E con questo la dimostrazione è terminata.

□

2.2 I byte in AES

In questa sezione cercheremo di dare una spiegazione esaustiva sulle operazioni che avvengono all'interno dell'algoritmo; il primo passo per fare questo è spiegare come vengono interpretati i byte all'interno dell'algoritmo.

In AES ogni byte (sequenza di 8 bit) viene interpretato come un polinomio in $GF(2^8)$ nel seguente modo:

$$\{b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0\} \Rightarrow b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

cioè, ogni bit equivale ad un coefficiente del polinomio sopraindicato.

Esempio. Conversione da esadecimale (base 16) a $GF(2^8)$:
$(49)_{16} = 4 \cdot 16 + 9 \cdot 16^0 = (73)_{10} = (1001001)_2 = (x^7 + x^4 + 1)_{GF(2^8)}$

2.2.1 Somma e moltiplicazione tra byte

Poiché la somma di due elementi nel campo finito $GF(2^8)$ si ottiene sommando $\bmod 2$ i coefficienti delle corrispondenti potenze dell'indeterminata x nei due polinomi, ne segue che in AES la somma di due byte si ottiene effettuando la somma $\bmod 2$ dei bit corrispondenti.

Esempio.

Notazione polinomiale: $(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = (x^7 + x^6 + x^4 + x^2)$
Notazione binaria: $(01010111)_2 \oplus (10000011)_2 = (11010100)_2$
Notazione esadecimale: $(57)_{16} \oplus (83)_{16} = (D4)_{16}$

Il simbolo sopraindicato, $\oplus$, indica l'operazione XOR, cioè l'implementazione della somma tra due byte effettuata bit a bit.

La moltiplicazione di due elementi (polinomi)in $GF(2^8)$ si ottiene moltiplicando i due polinomi modulo un polinomio irriducibile di ottavo grado, che nell'AES è:

$$m(x) = x^8 + x^4 + x^2 + x + 1.$$

Il polinomio $m(x)$ coincide con $(01)_{16}(1b)_{16}$: sono necessari due byte perché, nella conversione che stiamo utilizzando, con un byte si possono rappresentare al massimo polinomi di settimo grado; ovviamente, per quanto detto fin qui, $m(x) \notin GF(2^8)$.
Questo tipo di moltiplicazione gode della proprietà associativa ed esiste l'elemento neutro: $(01)_{16}$. Inoltre, esiste l'inverso moltiplicativo di ogni elemento diverso da zero e di grado inferiore a 8:
sia $b(x) \in GF(2^8)$; posso ricavare il suo inverso $b^{-1}(x)$ attraverso il seguente procedimento: utilizzo l'algoritmo euclideo per trovare $a(x)$ e $c(x)$ tali che

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

Quindi, poiché $a(x)\cdot b(x) \bmod m(x) = 1$, è possibile ricavare

$$b^{-1}(x) = a(x) \bmod m(x)$$

Non esiste un'operazione semplice a livello di byte per effettuare questo tipo di moltiplicazione, ma si può realizzare un algoritmo partendo da un caso particolare: la moltiplicazione di un polinomio per x:
moltiplicando un polinomio $f(x) \in GF(2^8)$ per x (con x in base 2 od in base 16), si ottiene

$$f(x)\cdot x = p(x) = b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x$$

Dobbiamo però analizzare due casi:

i. $\partial f(x) < 7 \Rightarrow \partial p(x) < 8 \Rightarrow p(x) \in GF(2^8)$;
ii. $\partial f(x) = 7$ (e quindi si avrà $b_7 = 1$) $\Rightarrow p(x) \notin GF(2^8)$;

nel secondo caso sarà quindi necessario ridurre $p(x) \bmod m(x)$, procedimento che può essere effettuato sottraendo $p(x)$ a $m(x)$.
Poiché ci troviamo nell'insieme $\mathbb{Z}_2$ e, dato che una sottrazione equivale ad una somma, possiamo sommare $p(x)$ a $m(x)$.
Il prodotto per x può essere implementato con un *left shift*, cioè moltiplicando il byte per $(02)_{16}$, e successivamente, nel caso in cui $b_7 = 1$, con uno XOR (l'operazione vista in precedenza, $\oplus$) con $(1b)_{16}$, cioè la rappresentazione esadecimale di $m(x)$.

Definiamo la funzione che effettua il prodotto di un polinomio per x, ovvero la moltiplicazione di un byte per $(02)_{16}$, come *xpol()*.
Una volta definita la precedente funzione *xpol()*, è possibile definire un algoritmo che risolva il prodotto tra due polinomi qualsiasi, $f(x)$ e $g(x)$, utilizzando la proprietà distributiva di cui gode la moltiplicazione.
Notiamo che, essendo in grado di effettuare la moltiplicazione $f(x)\cdot x$, si è anche in grado di effettuare $f(x)\cdot x^n$: è, infatti, sufficiente applicare n volte la funzione *xpol()*; inoltre, poiché $g(x)$ è somma di potenze di x , siamo in grado di calcolare il prodotto di $f(x)$ per ogni termine di $g(x)$, preso singolarmente.

Definiamo $f(x)$ e $g(x)$:

$$f(x) = x^6 + x^4 + x^2 + x + 1 = (57)_{16}, \qquad g(x) = x^4 + x + 1 = (13)_{16};$$

effettuiamo il prodotto $f(x)\cdot g(x)$, utilizzando la proprietà distributiva del prodotto rispetto alla somma e otteniamo:

$$f(x)\cdot g(x) = f(x)\cdot(x^4 + x + 1) = \left[f(x)\cdot x^4\right] + \left[f(x)\cdot x\right] + \left[f(x)\cdot 1\right]$$

dove ogni termine può essere calcolato utilizzando la funzione definita sopra.
Traduciamo ora l'algoritmo appena descritto a livello di byte:

$$f(x)\cdot x = (57)_{16}\cdot(02)_{16} = xpol((57)_{16}) = (\mathrm{AE})_{16}$$
$$f(x)\cdot x^2 = (57)_{16}\cdot(04)_{16} = xpol(xpol((57)_{16})) = xpol((\mathrm{AE})_{16}) = (47)_{16}$$
$$f(x)\cdot x^3 = (57)_{16}\cdot(08)_{16} = xpol((47)_{16}) = (8\mathrm{E})_{16}$$
$$f(x)\cdot x^4 = (57)_{16}\cdot(10)_{16} = xpol((8\mathrm{E})_{16}) = (07)_{16}$$

ed effettuiamo i calcoli:

$$f(x)\cdot g(x) = (57)_{16}\cdot(13)_{16} = (57)_{16}\cdot[(10)\oplus(02)\oplus(01)] =$$
$$= [(57)_{16}\cdot(10)_{16}]\oplus[(57)_{16}\cdot(02)_{16}]\oplus[(57)_{16}\cdot(01)_{16}] =$$
$$= (07)_{16}\oplus(\mathrm{AE})_{16}\oplus(57)_{16} = (\mathrm{FE})_{16}.$$

2.3 Le word in AES

Le word in AES sono delle quadruple di elementi del tipo $[a_3, a_2, a_1, a_0]$; per effettuare operazioni tra word si utilizzano polinomi di 4 termini a coefficienti in $GF(2^8)$:

$$[a_3, a_2, a_1, a_0] = a(x) = a_3x^3 + a_2x^2 + a_1x + a_0;$$

i coefficienti a_i $(i = 0,...,3)$, vengono trattati come byte e i calcoli tra coefficienti vengono effettuati come abbiamo spiegato nel paragrafo precedente.

2.3.1 Somma e moltiplicazione tra word

Consideriamo due polinomi, $a(x)$ e $b(x)$:

$$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0, \qquad b(x) = b_3x^3 + b_2x^2 + b_1x + b_0,$$

la loro somma si effettua in $GF(2^8)$ nel seguente modo:

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

quindi, data una coppia di word, la loro somma viene effettuata calcolando lo XOR tra ogni byte della prima word e il corrispettivo della seconda:

$$[a_3, a_2, a_1, a_0] + [b_3, b_2, b_1, b_0] = [(a_3 \oplus b_3), (a_2 \oplus b_2), (a_1 \oplus b_1), (a_0 \oplus b_0)].$$

Per il prodotto dei due polinomi, $a(x)$ e $b(x)$, si calcola come prima cosa il loro prodotto algebrico, e ottenendo cioè il polinomio:

$$p(x) = p_6x^6 + p_5x^5 + p_4x^4 + p_3x^3 + p_2x^2 + p_1x + p_0.$$

Per ottenere una sua rappresentazione tramite una quadrupla di byte è necessario ridurre quest'ultimo polinomio modulo un polinomio di quarto grado (questo perché il polinomio da ottenere dovrà avere al massimo grado 3 ed avere 4 coefficienti).
Nell'AES il polinomio di quarto grado utilizzato è il seguente:

$$x^4 + 1,$$

dobbiamo quindi effettuare la "riduzione" $p(x) \bmod (x^4 + 1)$;
se $x^i \bmod (x^4 + 1) = x^{i \bmod 4}$, allora

$$p(x) = p_6x^2 + p_5x + p_4 + p_3x^3 + p_2x^2 + p_1x + p_0$$

cioè

$$p(x) = p_3x^3 + (p_6 \oplus p_2)x^2 + (p_5 \oplus p_1)x + (p_4 \oplus p_0).$$

Per effettuare il prodotto tra due polinomi, è inoltre possibile definirne il prodotto modulare, indicato con $\otimes$:

$$d(x) = a(x) \otimes b(x) = d_3x^3 + d_2x^2 + d_1x + d_0,$$

in cui:

$$d_0 = (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3)$$
$$d_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a_2 \cdot b_3)$$
$$d_2 = (a_2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3)$$
$$d_3 = (a_3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3);$$

I coefficienti del polinomio $d(x)$ possono essere scritti nella seguente forma:

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

Capitolo 3

Descrizione dell'algoritmo

3.1 Descrizione dell'algoritmo di cifratura

L'AES è un cifrario a blocchi, ovvero il testo in chiaro (espresso in bit), viene diviso in blocchi di lunghezza fissa cifrati autonomamente che vengono successivamente riuniti per dare il messaggio cifrato; sulla base delle specifiche del concorso, gli autori dell'AES decisero che la dimensione dei blocchi del cifrario dovesse essere di 128 bit, mentre, la chiave, sempre in base alle specifiche, poteva assumere tre lunghezze diverse: 128, 192 o 256 bit, dando quindi origine a tre versioni diverse dell'algoritmo. L'input e l'output dell'algoritmo sono sequenze di bit, che nell'implementazione vengono trattati come array, mentre nella cifratura sono considerati come matrici i cui elementi sono byte: l'unità di elaborazione dell'algoritmo quindi non è tanto il bit quanto il byte, perciò la matrice in input, dato che 128 bit sono 16 byte, sarà una matrice con 16 elementi, ovvero una matrice quadrata 4×4. Ne segue che la chiave sarà rappresentata, nelle tre versioni, da matrici 4×4, 4×6 o 4×8: vengono mantenute 4 righe per consentire operazioni con il blocco in input, come si vedrà in seguito.
Nel seguito utilizzeremo la seguente notazione:

word per indicare una singola colonna di un blocco (sia di quello in input che di quello della chiave);
Nb per indicare il numero di word di un blocco in input;
Nk per indicare il numero di word della chiave;
Nr per indicare il numero di round in cui avverrà la codifica.

Osservazione. La versione ufficiale dell'AES presentava: $Nb = 4$ (in quanto il blocco in input è di 128 bit), e $Nk = 4, 6, 8$ a seconda del tipo di chiave usata.

Abbiamo introdotto Nr, ovvero il numero di round, poiché il messaggio in chiaro viene cifrato più volte, a seconda di quale versione della chiave si utilizza:

chiave a 128 bit $\Rightarrow$ 10 round
chiave a 192 bit $\Rightarrow$ 12 round
chiave a 256 bit $\Rightarrow$ 14 round

Il numero elevato di round conferisce una maggior sicurezza poiché aumentano le possibili combinazioni di crittazione.

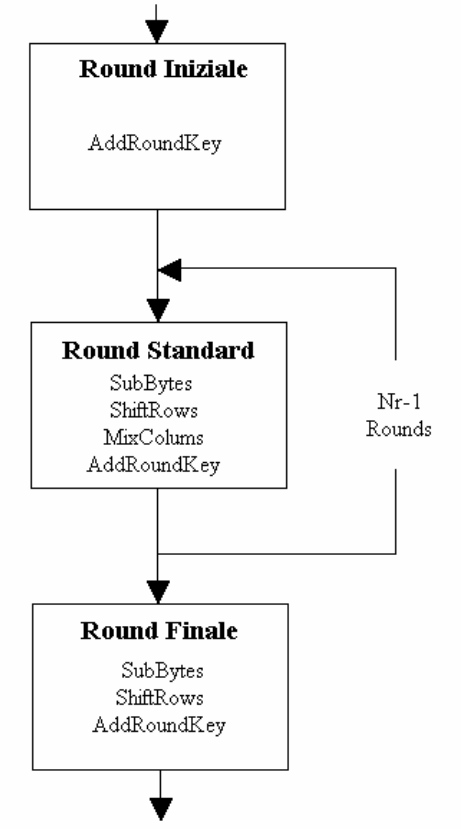
L'algoritmo è formato da due funzioni principali: *KeyExpansion* e *Chiper*; Chiper è la funzione che effettivamente codifica il messaggio, mentre KeyExpansion è un generatore di chiavi: la crittazione avviene utilizzando una chiave diversa per ogni round, e la prima funzione fornisce alla seconda tutte le chiavi necessarie generandole a partire da quella principale.
Come spiegheremo in seguito, il numero delle chiavi è dato dal numero di round più una; inoltre, le operazioni non vengono effettuate sull'input, che all'inizio è un array, ma su una matrice di appoggio detta State (o matrice di Stato), su cui l'input viene "copiato" e da cui

verrà estratto l'output: la copiatura avverrà per colonne, ovvero i primi 4 byte dell'input formeranno la prima word di State e così via.

La funzione Chiper implementa gli Nr round, di cui $Nr-1$ sono identici, mentre uno è diverso dagli altri; gli $Nr-1$ round identici codificano il messaggio attraverso l'applicazione di quattro funzioni in cascata:

- *SubBytes* operante su ogni byte
- *ShiftRows* operante sulle righe di byte
- *MixColumns* operante sulle word
- *AddRoundKey* cifra utilizzando la chiave;

il round rimanente è diverso dagli altri poiché sostituisce a *MixColumns* un'altra occorrenza di *AddRoundKey*, e necessita quindi di due chiavi; inoltre, le funzioni vengono applicate in ordine diverso: il round diverso inizia prima degli altri e termina dopo gli altri, come si può meglio comprendere osservando lo schema a blocchi dell'algoritmo:



In questo schema il round diverso dagli altri è formato da *Round Iniziale*, con l'applicazione di AddRoundKey, e *Round Finale*, con l'applicazione di SubBytes, ShiftRows e AddRoundKey.
Dallo schema a blocchi, si può intuire la necessità di realizzare un ciclo per effettuare gli Nr-1 round.

Poiché abbiamo definito le operazioni da eseguire possiamo scrivere lo pseudocodice di Cipher:

```
Pseudo – Codice. Cipher

Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]

    state = in

    AddRoundKey(state, w[0,Nb-1])

    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

    out = state
end
```

In input la funzione prende tre array, *in*, *out* e *w*, definiti come segue:

- *in* è il blocco in input, di dimensioni $4 \cdot Nb$ ovvero 16 byte.;
- *out* è il blocco in output, che ha le stesse dimensioni di quello in input e che è ovviamente vuoto poiché dovrà essere "riempito" al termine della funzione;
- *w* è un array di word, cioè un array di colonne, di dimensione $Nb \cdot (Nr+1)$, ossia quattro colonne per ogni round più uno: la chiave infatti è composta da quattro word, e, poiché un round ha bisogno di due chiavi, queste saranno $Nr+1$.
  w viene passato a Cipher da KeyExpansion, perciò a Cipher non viene influenzato dal metodo utilizzato per generare le chiavi, ma richiede di avere tutte le chiavi di cui necessita.

Riassumendo, le prime operazioni di Cipher sono creare lo State e "copiarvi sopra" l'input; in seguito vengono implementati i round, nello stesso modo in cui sono descritti nello

schema a blocchi: per gli $Nr-1$ round viene realizzato un ciclo con un'istruzione for. Infine nell'array di output viene copiato lo State e la funzione restituisce il testo cifrato.
Quindi, una descrizione generale dell'algoritmo, può essere schematizzata come segue:

1. Dato un plaintext x (testo in chiaro), inizializza $State = x$ ed esegue un operazione chiamata AddRoundKey, che esegue lo XOR tra RoundKey e lo State.
2. Per ognuno dei primi $Nr-1$ round, esegue un'operazione sullo State chiamata SubBytes usando una S-Box; esegue una permutazione dello State: ShiftRows; esegue un'operazione sullo State: MixColumns; esegue AddRoundKey.
3. Esegue SubBytes; esegue ShiftRows; esegue AddRoundKey.
4. Pone il chipertext (testo cifrato) $y = State$.

Vediamo ora, in dettaglio, le operazioni eseguite da AES:

3.1.1 SubBytes

La funzione *SubBytes* (dove Sub sta per substitution, ovvero sostituzione) che si occupa di sostituire i byte, presenta due caratteristiche principali: è invertibile, in quanto è necessario poter realizzare la funzione inversa, ed è non lineare, caratteristica che costituisce un punto di forza dell'AES, dato che la non-linearità è un grave problema per il crittoanalista.
La sostituzione avviene in maniera molto semplice: si fa uso di una tabella, detta *Substitution - Box*, oppure *S-Box*, che è una permutazione di $\{0,1\}^8$, in cui ogni possibile valore di un byte ha un valore corrispondente; indichiamo questa tabella con π_S.
Per analizzare π_S rappresentiamo i byte in notazione esadecimale: π_S sarà allora una matrice 16 × 16 (poiché un byte può assumere 256 valori, la S-Box dovrà avere 256 elementi), dove le righe e le colonne sono indicizzate da cifre esadecimali e l'entrata nella riga X e la colonna Y è $\pi_S(XY)$. Inoltre, come già accennato, la formulazione della S-Box di AES coinvolge operazioni in un campo finito:

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

La funzione SubBytes non fa altro che sostituire ad ogni byte il suo corrispettivo presente nella S-Box; vediamo come avviene la conversione di un elemento del campo in un byte:
l'elemento del campo, $\sum_{i=0}^{7} a_i x^i$, corrisponde al byte $a_7a_6a_5a_4a_3a_2a_1a_0$, dove $a_i \in \mathrm{Z}_2$, per $0 \le i \le 7$. Così π_S risulta essere definita dall'algoritmo nel quale gli otto bit di input $a_7a_6a_5a_4a_3a_2a_1a_0$ sono sostituiti dagli otto bit di output $b_7b_6b_5b_4b_3b_2b_1b_0$.
Quindi: $SubBytes(a_7a_6a_5a_4a_3a_2a_1a_0) = b_7b_6b_5b_4b_3b_2b_1b_0$.

Il problema di implementazione sta nel costruire una S-Box in modo da garantire la non linearità e l'invertibilità della funzione: per costruirla si parte da una matrice 16×16 i cui elementi sono tutti i possibili valori ordinati di un byte, ai quali si applicano le seguenti trasformazioni:

i. ad ogni elemento viene sostituito il suo inverso in $GF(2^8)$ (il byte 00000000 rimane tale)

ii. ad ogni bit del byte viene applicata la seguente trasformazione affine:

$$a_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} c_i$$

dove c_i è l'i-esimo bit del byte 01100011, valore fisso utilizzato per ogni byte.

In forma matriciale, la trasformazione affine degli elementi della S-Box possono essere espressi come:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Applicando le precedenti trasformazioni viene realizzata la S-Box, che si può osservare nella seguente tabella :

		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

I valori in tabella sono espressi in notazione esadecimale; in particolare, l'insieme X delle righe rappresenta i primi 4 bit, mentre l'insieme Y delle colonne rappresenta gli ultimi 4, ossia i bit meno significativi.
Riassumendo, la funzione SubBytes trova i byte negli insiemi x e y e sostituisce l'elemento corrispondente.

Per illustrare meglio l'operazione SubBytes, facciamo un esempio:
supponiamo di avere il numero $(53)_{16}$ e verifichiamo che $(53)_{16}$ in binario è rappresentato dal byte 01010011:
$(53)_{16} = 5 \cdot 16 + 3 = (83)_{10}$ e, poiché

$(83)_{10} = 0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 2^0$

abbiamo $(53)_{16} = (01010011)_2$

che rappresentato come elemento del campo diventa:

$x^6 + x^4 + x + 1$.

L'inverso moltiplicativo (nel campo $GF(2^8)$) è:

$x^7 + x^6 + x^3 + x$.

Quindi in notazione binaria abbiamo:

$(a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0) = (11001010)$.

Calcoliamo poi:

$$b_0 = a_0 + a_4 + a_5 + a_6 + a_7 + c_0 \mod 2 =$$
$$= 0 + 0 + 0 + 1 + 1 + 1 \mod 2 =$$
$$= 1$$
$$b_1 = a_1 + a_5 + a_6 + a_7 + a_0 + c_1 \mod 2 =$$
$$= 1 + 0 + 1 + 1 + 0 + 1 \mod 2 =$$
$$= 0$$

$\vdots$

$\vdots$

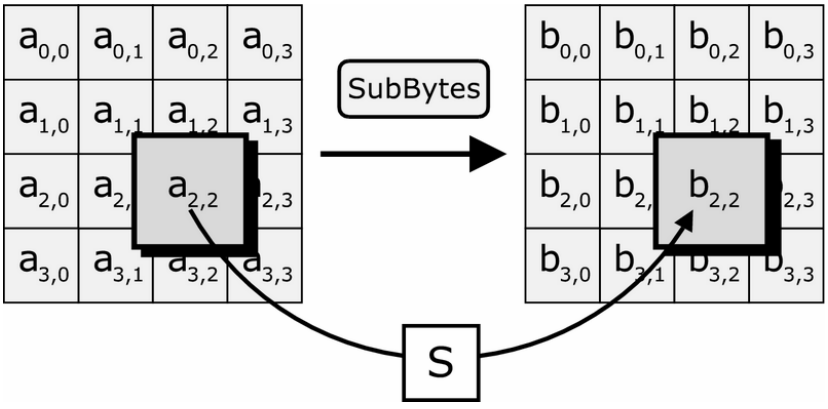
e così via.

Il risultato è quindi:

$(b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0) = (11101101)_2$

In notazione esadecimale 01010011 è ED.

Questo calcolo può essere verificato controllando che, nella tabella precedente, l'elemento della riga 3 e della colonna 5 è proprio ED.



3.1.2 ShiftRows

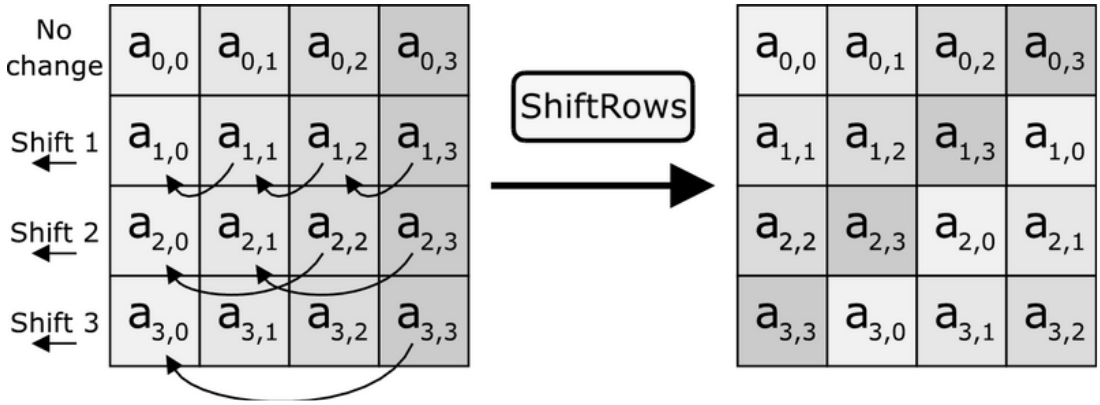
La funzione *ShiftRows* viene applicata a State dopo la funzione MixColumns e provvede a traslare le righe di byte di un parametro dipendente dal numero di riga; nel caso, ad esempio, della versione a 128 bit abbiamo:

$$a_{r,c} = b_{r,(c-Shift(r,Nb)) \bmod Nb}$$

dove $0 < r < 4$, $0 \le c \le Nb$; la "funzione" shift assume i seguenti valori:

$Shift(1,4) = 1$
$Shift(2,4) = 2$
$Shift(3,4) = 3$

Nelle altre due versioni avremo ovviamente valori analoghi per S*hift*.



In pratica, la prima riga resta uguale (shift di zero posizioni), la seconda riga si muove verso sinistra di una posizione, la terza riga di due posizioni e la quarta riga di tre posizioni. L'operazione è modulo 4, quindi i byte che muovendosi a sinistra "escono" dalla matrice, rientrano da destra.

3.1.3 MixColumns

La funzione *MixColumns* opera sulle colonne di State: ogni byte della matrice viene trattato come un polinomio in $GF(2^8)$ ed anche la singola colonna viene considerata come un polinomio, che ha come coefficienti i byte della colonna.
Ogni colonna viene moltiplicata modulo $x^4 + 1$ per un polinomio fissato $c(x)$, dato da:

$$c(x) = (03)_{16} x^3 + (01)_{16} x^2 + (01)_{16} x + (02)_{16}$$

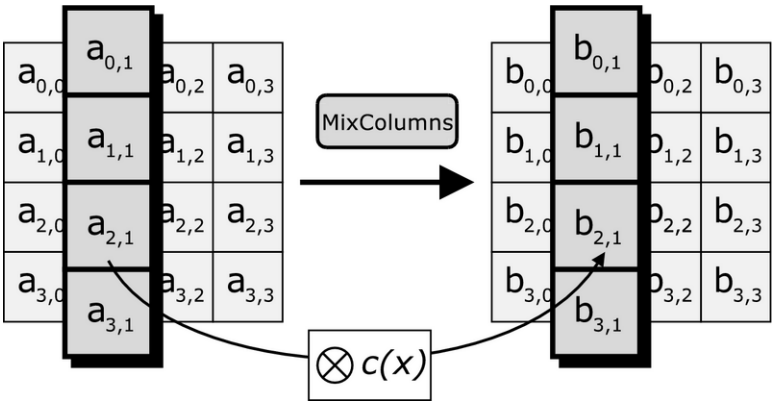
$c(x)$ ha le stesse caratteristiche delle singole colonne, è un polinomio con coefficienti che sono polinomi in $GF(2^8)$.
La moltiplicazione avviene secondo le regole viste nelle basi matematiche, e può essere schematizzata come segue:

$$\begin{bmatrix} a_{0,c} \\ a_{1,c} \\ a_{2,c} \\ a_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} b_{0,c} \\ b_{1,c} \\ b_{2,c} \\ b_{3,c} \end{bmatrix} \text{ per } 0 \le c < Nb$$

ossia:

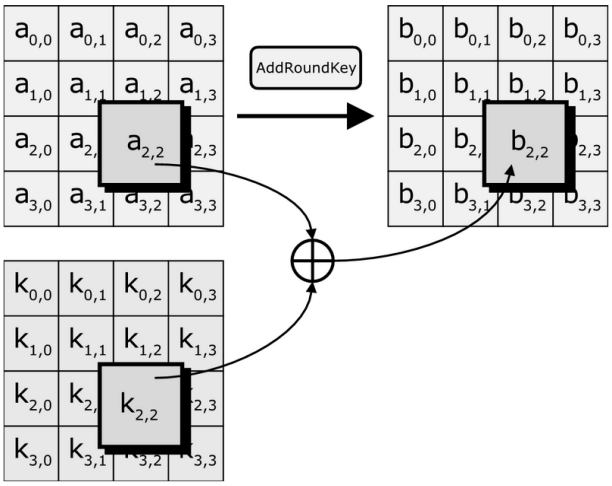
$$a_{0,c} = ((02)_{16} \bullet b_{0,c}) \oplus ((03)_{16} \bullet b_{1,c}) \oplus b_{2,c} \oplus b_{3,c}$$

$$a_{1,c} = b_{0,c} \oplus ((02)_{16} \bullet b_{1,c}) \oplus ((03)_{16} \bullet b_{2,c}) \oplus b_{3,c}$$
$$a_{2,c} = b_{0,c} \oplus b_{1,c} \oplus ((02)_{16} \bullet b_{2,c}) \oplus ((03)_{16} \bullet b_{3,c})$$
$$a_{3,c} = ((03)_{16} \bullet b_{0,c}) \oplus b_{1,c} \oplus b_{2,c} \oplus ((02)_{16} \bullet b_{3,c})$$



3.1.4 AddRoundKey

La funzione *AddRoundKey* è l'ultima operazione effettuata in un round.
Come si può vedere nello pseudo-codice, AddRoundKey viene chiamata ricevendo in input quattro delle colonne dell'array w: queste colonne andranno a formare la chiave, non ancora utilizzata nelle altre funzioni. La chiave viene aggiunta allo State semplicemente con un operazione di XOR su ogni byte, come si può vedere dallo schema seguente:



Un singolo round (ossia, uno degli $Nr-1$ round uguali), dopo aver applicato AddRoundKey, termina; alla fine di tutti i round il testo in chiaro sarà effettivamente crittato.

3.1.5 KeyExpansion

Ci rimane da descrivere come viene generato l'insieme delle chiavi di AES. Diamo questa descrizione della versione di AES a 10 round, ossia la versione che usa una chiave a 128 bit.

Ci servono quindi un numero di chiavi adatto a 11 round, ognuna delle quali è costituita da 16 byte. L'algoritmo che genera le chiavi è *word–oriented*, dove una parola (word appunto) è formata da 4 byte o, equivalentemente, da 32 bit; quindi, ogni chiave di un round è costituita da 4 parole.
L'insieme delle chiavi di tutti i round è detto *chiave espansa* (*expanden key*), è formato da 44 parole ed è denotato con $w[0],...,w[43]$, dove ogni $w[i]$ è una parola.
La chiave espansa è costruita usando la funzione *KeyExpansion*, che è presentata nell'algoritmo seguente:

Algoritmo. KeyExpansion

Funzioni Esterne: *RotWord*, *SubWord*

$01000000 \rightarrow Rcon[1]$
$02000000 \rightarrow Rcon[2]$
$04000000 \rightarrow Rcon[3]$
$08000000 \rightarrow Rcon[4]$
$10000000 \rightarrow Rcon[5]$
$20000000 \rightarrow Rcon[6]$
$40000000 \rightarrow Rcon[7]$
$80000000 \rightarrow Rcon[8]$
$1B000000 \rightarrow Rcon[9]$
$36000000 \rightarrow Rcon[10]$

for $i \rightarrow 1$ to 3
  do $(key[4i], key[4i+1], key[4i+2], key[4i+3]) \rightarrow w[i]$

for $i \rightarrow 0$ to 43
  do $w[i\text{-}1] \rightarrow temp$
  if $i \equiv 0 \ (\bmod 4)$
    $SubWord(RotWord(temp)) \oplus Rcon[i/4] \rightarrow temp$
    $w[i\text{-}4] \oplus temp \rightarrow w[i]$

return $(w[0],...,w[43])$

L'input di questo algoritmo è la chiave di 128 bit, *key*, che è utilizzata come un vettore di byte, $key[0],..key[15]$; l'output è il vettore delle parole, w, che abbiamo introdotto precedentemente.

La funzione KeyExpansion incorpora altre due funzioni, che sono chiamate *RotWord* e *SubWord*; la funzione RotWord esegue uno spostamento (shift) ciclico a sinistra dei quattro byte B_0, B_1, B_2, B_3, cioè:

$$RotWord(B_0, B_1, B_2, B_3) = (B_1, B_2, B_3, B_0);$$

mentre la funzione $SubWord(B_0, B_1, B_2, B_3)$ applica la S-Box ad ognuno dei quattro byte B_0, B_1, B_2, B_3, cioè:

$$SubWord(B_0, B_1, B_2, B_3) = (B_0', B_1', B_2', B_3')$$

dove $B_i' = SubBytes(B_i)$, $i = 0,...,3$.

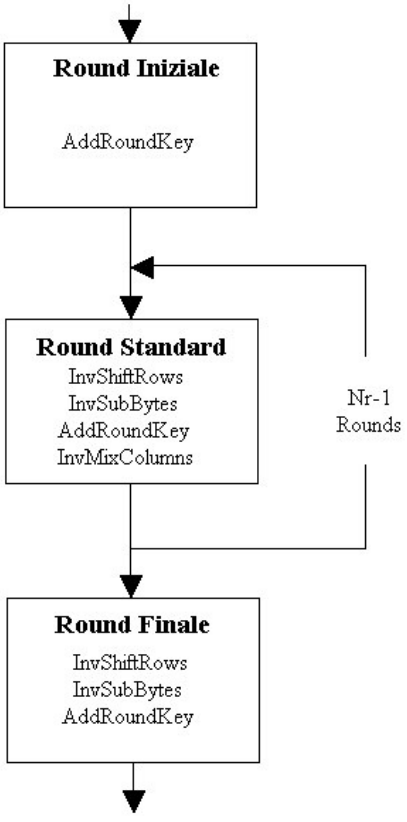
Inoltre, *Rcon* è un vettore di 10 parole, indicato con $Rcon[1],...,Rcon[10]$.

Tutte queste sono costanti definite in notazione esadecimale all'inizio dell'algoritmo.

3.2 Descrizione dell'algoritmo di decifratura

Nell' analisi dell' algoritmo di decifratura di AES, anche detto *Inverse Cipher*, è facile notare come vengano ripercorse le stesse strutture dell'algoritmo di cifratura. InvCipher infatti, come Cipher opera su box da 128 bit (o 16 byte), quindi l'unità fondamentale di operazione è ancora una volta il byte e, la chiave è la proprietà che distingue le tre versioni dell' algoritmo inverso a seconda se viene utilizzata a 128,192 o 256 bit (standard a 128 bit). Inoltre anche InvCipher distingue i box, considerati come matrici, in box Input/Output e box di Stato, ovvero la matrice che viene modificata dalle iterazioni dell' algoritmo. Anche le unità di riferimento sono le stesse:

Nb per indicare il numero di word di un blocco in input;
Nk per indicare il numero di word della chiave;
Nr per indicare il numero di round



Un altro elemento invariante è lo schema dell'algoritmo, infatti nonostante le funzioni siano differenti, i round di InvCipher e Cipher conservano la stessa struttura: InvCipher implementa Nr round; di questi $Nr-1$ sono eseguiti ciclicamente, mentre il round iniziale e il round finale compongono un unico passo che comincia prima e termina solo dopo le iterazioni del ciclo.

L'algoritmo è composto da 4 funzioni:

- *InvShiftRows*
- *InvSubBytes*
- *InvMixColumns*
- *AddRoundKey*

Come si può capire, le prime tre funzioni sono caratterizzanti l'InvCipher, mentre l'ultima funzione è la stessa incontrata in Cipher.
Analizziamo lo pseudo-codice di InvCipher:

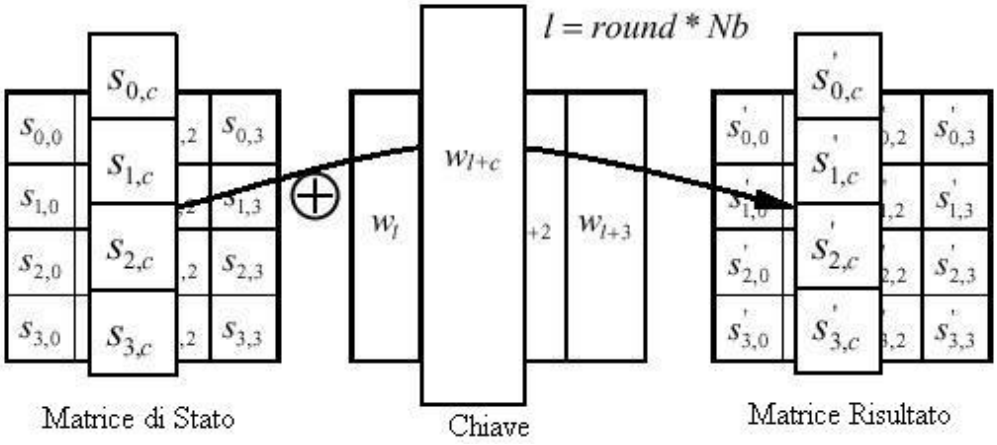
```
Pseudo – Codice.  InvCipher

InvCipher(Byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
Begin
∴  byte state[4, Nb]
∴  state = in
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0,Nb-1])
∴ out = state
end
```

Come si può notare, nella prima riga del codice vengono definiti gli input della funzione, i tre array in, out e w; in e out hanno le stesse dimensioni, cioè 128 bit, e rappresentano, rispettivamente, la matrice di ingresso e la matrice di uscita; L'array w, formato da word, ha dimensione $Nb(Nr+1)$ e viene "passato" ad InvCipher mediante il codice di KeyExpansion.

Le righe del codice precedute dal simbolo ∴, indicano come l'algoritmo definisca le matrici di stato e, successivamente, definisca questa come Input all'inizio e come Output alla fine del codice.

3.2.1 AddRoundKey

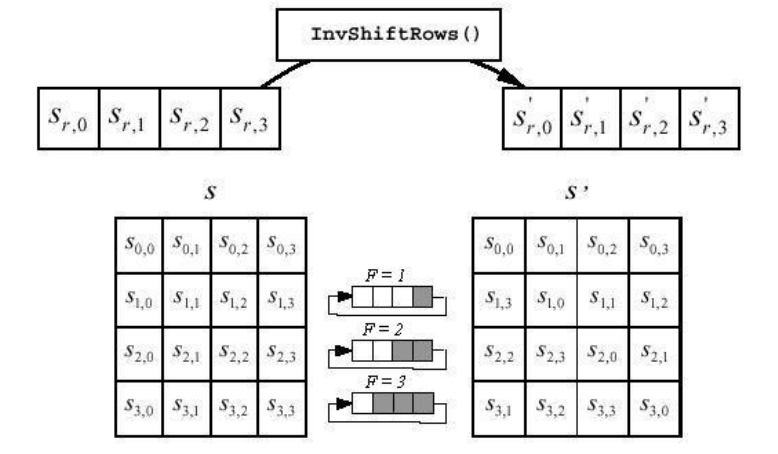
La funzione AddRoundKey riceve ad ogni iterazione modificata dal proprio codice, la matrice di Stato e la Chiave come parametri. Osserviamo la seguente figura:



Si nota subito come AddRoundKey consideri la State una colonna alla volta e la ponga in XOR con una colonna della matrice Chiave: il risultato, ovviamente è una nuova matrice di Stato.

3.2.2 InvShiftRows

La funzione *InvShiftRows* opera sulle righe della matrice di Stato, "traslandole" di un offset a seconda della riga considerata: se la riga considerata è la prima, cioè la numero zero, il valore di offset F sarà zero, ovvero non cambierà, mentre se la riga considerata è la numero 1, si avrà $F = 1$ e così via fino ad avere $F = 3$.



L'analogia con la funzione ShiftRows, sta nel fatto che questa, cioè l'inversa presenta la traslazione invertita e cioè:

$$S'_{r,(c+Shift(r,Nb))} \bmod Nb = S_{r,c}$$

dove $0 \le c < Nb$, $0 < r < 4$.
Nella versione di AES a 128 bit, i valori di Shift sono i seguenti:

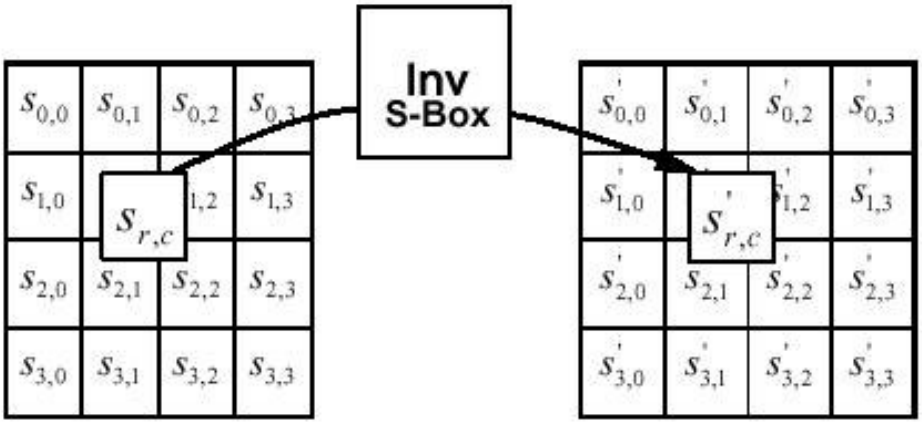
- $Shift(4,1) = 1$
- $Shift(4,2) = 2$
- $Shift(4,3) = 3$

3.2.3 InvSubBytes

La funzione *InvSubBytes* realizza una sostituzione dei byte applicando l'inverso della S-Box ad ogni byte dell'array State. Questa S-Box inversa è costruita dalla composizione di due trasformazioni:

i. applicando l'inverso della trasformazione affine descritta nella funzione SubBytes
ii. prendendo l'inverso moltiplicativo nel campo finito $GF(2^8)$

La trasformazione di sostituzione procede in modo del tutto analogo a quello descritto nella fase di cifratura:



Nella matrice di Stato, i valori dei singoli byte sono rappresentati da numeri esadecimali di due cifre, le quali indicano, rispettivamente, un'ascissa ed un ordinata di una tabella di sostituzione. Questa tabella è detta Inverse S-Box e la si può osservare nella seguente figura:

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	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

3.2.4 InvMixColumns

La funzione *InvMixColumns* opera sulla matrice di Stato da colonna a colonna, e considera ogni colonna come un polinomio di 4 termini. Le colonne rappresentano polinomi con coefficienti nel campo finito $GF(2^8)$, i quali vengono moltiplicati modulo x^4+1 con un fissato polinomio $a^{-1}(x)$, dato da:

$$a^{-1}(x) = (0B)_{16}x^3 + (0D)_{16}x^2 + (09)_{16}x + (0E)_{16}.$$

Questo prodotto può essere scritto in forma matriciale:

$$S'(x) = a^{-1}(x) \oplus S(x)$$

ossia

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}, \text{ per } 0 \le c < Nb$$

Capitolo 4

Rappresentazione polinomiale dell'AES

In questo paragrafo descriviamo l'AES in forma polinomiale: ogni operazione compiuta nei round dell'algoritmo viene fatta su un anello ed è, quindi, possibile rappresentare ogni funzione applicata nei round con un polinomio, o come composizione di più polinomi su un anello.

4.1 Descrizione dell'algoritmo nell'anello R

Sia $\mathbb{Z}_2 = \{0,1\}$ il campo binario e consideriamo il polinomio irriducibile

$$p(z) = z^8 + z^4 + z^3 + z + 1 \in \mathbb{Z}_2[z]$$

Sia $\mathbb{G} = \mathbb{Z}_2[z]/\langle p(z)\rangle = GF(2^8)$ il campo di Galois di 2^8 elementi e consideriamo l'ideale:

$$I = \langle x^4 + 1, y^4 + 1, p(z)\rangle \subset \mathbb{Z}_2[x, y, z]$$

Descriveremo l'algoritmo dell'AES attraverso una serie di manipolazioni polinomiali dentro l'anello finito:

$$R = \mathbb{Z}_2[x, y, z]/\, I = \mathbb{G}[x, y]/\langle x^4 + 1, y^4 + 1\rangle$$

Da questa descrizione abbiamo che l'anello R ha contemporaneamente sia la struttura di una $\mathbb{Z}_2$-algebra finita che la struttura di una $\mathbb{G}$-algebra finita.
I monomi:

$$\{x^i y^j z^k : 0 \le i, j \le 3, 0 \le k \le 7\}$$

formano una $\mathbb{Z}_2$-base dell'anello (algebra) R. In particolare, abbiamo $\dim_{\mathbb{Z}_2} R = 128$, cioè $|R| = 2^{128}$.

Nell'anello R le operazioni sono eseguite come segue: la somma è fatta componente per componente ed il prodotto è attuato attraverso il prodotto in $\mathbb{Z}_2[x, y, z]$ seguita dalla riduzione modulo l'ideale I.

Sia $r \in R$ un elemento e siano $r_{i,j} \in \mathbb{G}$ e $r_j \in \mathbb{G}[x]/\langle x^4 + 1\rangle$, possiamo definire r come segue:

$$r = \sum_{i=0}^{3}\sum_{j=0}^{3} r_{i,j} x^i y^j = \sum_{j=0}^{3}\left(\sum_{i=0}^{3} r_{i,j} x^i\right) y^j = \sum_{j=0}^{3} r_j y^j$$

A livello teorico, un crittosistema a chiave segreta è dato da un'applicazione di cifratura:

$$\varepsilon : M \times K \to C$$

ed un'applicazione di decifratura

$$\delta : C \times K \to M$$

tali che $\delta(\varepsilon(m,k),k) = m$, con $m \in M$ e $k \in K$ e dove M è lo spazio dei messaggi in chiaro (*plain-text*), C è lo spazio dei messaggi cifrati (*cipher-text*) e K è lo spazio delle chiavi.

Nel crittosistema AES abbiamo, come già detto in precedenza, la possibilità di lavorare con chiavi segrete di 128 bit, 192 bit o 256 bit; per semplicità illustreremo il crittosistema utilizzando la chiave a 128 bit, ossia quando $|K| = 2^{128}$.

Per l'algoritmo di AES definiamo

$$K = M = C = R.$$

Per la nostra descrizione è di fondamentale importanza il seguente polinomio, che rappresenta la S-Box di AES:

$$\begin{aligned}\varphi(u) = {} & (z^2+1)u^{254} + (z^3+1)u^{253} + (z^7+z^6+z^5+z^4+z^3+1)u^{251} + \\ & + (z^5+z^2+1)u^{247} + (z^7+z^6+z^5+z^4+z^2)u^{239} + u^{223} + \\ & + (z^7+z^5+z^4+z^2+1)u^{191} + (z^7+z^3+z^2+z+1)u^{127} + \\ & + (z^6+z^5+z+1) \in \mathbb{G}[u]\end{aligned}$$

Supponiamo che due utenti, Alice e Bob, condividano una chiave segreta $k \in R$ ed Alice voglia cifrare il messaggio $m \in R$; come prima cosa, sia Alice che Bob devono eseguire un'espansione della chiave (*key-expansion*) che darà 11 elementi $k^t \in R$, con $t = 0,...10$.
Questo calcolo è fatto ricorsivamente, nel seguente modo:

$$k^{(0)} = k;$$

$$k_0^{(t+1)} = \left(\sum_{i=0}^{3} \varphi(k_{i,3}^{(t)} x^i) \right) x^3 + z^t + k_0^{(t)}, \text{ per } t = 0,...,9$$

$$k_i^{(t+1)} = k_{i-1}^{(t+1)} + k_i^{(t)}, \text{ per } i = 1,2,3 \text{ e } t = 0,...,9.$$

Per descrivere l'algoritmo di cifratura dobbiamo definire l'elemento dell'anello:

$$\gamma = (z+1)x^3 + x^2 + x + z \in R$$

che rappresenta l'operazione MixColumns in AES.
Usando la chiave del round $k^{(t)} \in R$ e partendo con il messaggio in chiaro $m \in R$, Alice calcola ricorsivamente:

$$m^{(0)} = m + k^{(0)}$$

$$m^{(t+1)} = \gamma \sum_{i=0}^{3} \sum_{j=0}^{3} \varphi(m_{i,j}^{(t)}) x^i y^{3i+j} + k^{(t+1)}, \text{ per } t = 0,...,8$$

$$c = m^{(10)} = \sum_{i=0}^{3} \sum_{j=0}^{3} \varphi(m_{i,j}^{(9)}) x^i y^{3i+j} + k^{(10)}$$

Il messaggio cifrato che Alice trasmetterà a Bob, è denotato con c.

Osserviamo che nel decimo round non si hanno prodotti per γ: questo assicura che le operazioni di cifratura e di decifratura possono essere descritte formalmente dalle stesse operazioni algebriche.

Sia f una funzione definita come segue:

$$\begin{aligned} f : GF(2^n) &\to GF(2^n) \\ a &\mapsto b = f(a) \end{aligned}$$

Tutte le funzioni su $GF(2^n)$ possono essere espresse come un polinomio su $GF(2^n)$ di grado al più $2^n - 1$:

$$f(a) = \sum_{i=0}^{2^n-1} c_i a^i$$

Esiste un'unica permutazione polinomiale $\psi(u) \in \mathbb{G}[u]$ di grado al più 255 ($2^8 - 1 = 256 - 1$) tale che $\varphi \circ \psi = \psi \circ \varphi = id_{\mathbb{G}}$.
L'elemento $\gamma \in R$ è invertibile e vale:
$\gamma^{-1} = (z^3+z+1)x^3 + (z^3+z^2+1)x^2 + (z^3+1)x + (z^3+z^2+z) \in R$.

Usando l'applicazione ψ, l'elemento γ^{-1} e la chiave del round $k^{(t)}$, Bob può decifrare il messaggio m, mandatogli da Alice, attraverso i seguenti passaggi:

$$c^{(0)} = c + k^{(10)}$$

$$c^{(t+1)} = \gamma^{-1} \sum_{i=0}^{3} \sum_{j=0}^{3} \psi(c_{i,j}^{(t)}) x^i y^{i+j} + \gamma^{-1} k^{(9-t)}$$

$$c^{(10)} = \sum_{i=0}^{3} \sum_{j=0}^{3} \psi(c_{i,j}^{(9)}) x^i y^{i+j} + k^{(0)}$$

Si verifica facilmente che $m = c^{(10)}$.

Notiamo che formalmente sia lo schema di cifratura che quello di decifratura seguono la stessa sequenza di trasformazioni: il polinomio φ è semplicemente rimpiazzato da ψ, il prodotto per γ è sostituito dal prodotto per γ^{-1} e l'insieme delle chiavi è cambiato sostituendo $k^{(t)}$, per $t = 0,...,10$, con $k^{(10)}, \gamma^{-1}k^{(9)}, ..., \gamma^{-1}k^{(1)}, k^{(0)}$.

Osservazione. Nella descrizione originale dell'algoritmo di AES, l'anello R non veniva utilizzato; gli insiemi di elementi aventi 128 bit erano descritti da matrici 4×4, ognuna delle quali aveva elementi di dimensione di un byte (cioè 8 bit). La descrizione assegnava ad ogni elemento $r = \sum_{i=0}^{3} \sum_{j=0}^{3} r_{i,j} x^i y^j$, la matrice 4×4:

$$\begin{bmatrix} r_{0,0} & r_{0,1} & r_{0,2} & r_{0,3} \\ r_{1,0} & r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,0} & r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,0} & r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix}$$

dove ogni elemento $r_{i,j} \in \mathbb{G}$ rappresenta un byte.

Usando uno schema specifico, l'algoritmo di AES esegue le seguenti operazioni: *SubBytes*, *ShiftRows*, *MixColumns*, *AddRoundKey*. Lo schema con cui vengono eseguite le operazioni è il seguente: nel round zero si aggiunge al testo in chiaro la chiave $k^{(0)}$, quindi si esegue solo *AddRoundKey*; dal primo al nono round si eseguono le operazioni *SubBytes*, *ShiftRows*, *MixColumns* e *AddRoundKey*. Nel decimo round si eseguono solamente *SubBytes*, *ShiftRows* e *AddRoundKey*.

Appendice

Gli anelli ed i gruppi: definizione e principali proprietà

Definizione. Si dice *anello* l'insieme $(A, +, \cdot)$, dotato di due operazioni binarie, indicate con + e ·,

$$\begin{array}{ll} A \times A \to A & A \times A \to A \\ (a,b) \mapsto a+b & (a,b) \mapsto a \cdot b \end{array}$$

che prendono il nome di addizione e sottrazione, tali che valgano le seguenti condizioni:

i. + è associativa, ossia $(a+b)+c = a+(b+c)\ \forall a,b,c \in A$;
ii. esiste un elemento 0 che è neutro rispetto a +, ossia $a+0=0+a=a\ \forall a \in A$;
iii. $\forall a \in A$ esiste un elemento, $-a$, tale che $a+(-a)=0$ (esistenza dell'opposto);
iv. + è commutativa, ossia $a+b=b+a\ \forall a,b \in A$;
v. · è associativa, cioè $(a \cdot b) \cdot c = a \cdot (b \cdot c)\ \forall a,b,c \in A$;
vi. valgono le seguenti proprietà distributive:
$a \cdot (b+c) = a \cdot b + a \cdot c \qquad (a+b) \cdot c = a \cdot c + b \cdot c \qquad \forall a,b,c \in A$.

Un insieme dotato di un'operazione che gode di i., ii., iii., prende il nome di *gruppo*: quindi $(A,+)$ è un gruppo; il punto iv. dice che il gruppo è *abeliano* cioè commutativo.

Per un anello valgono inoltre le seguenti proprietà:

1. un anello in cui la moltiplicazione sia commutativa prende il nome di *anello commutativo*;
2. un anello A si dice *unitario* o *con unità* se esiste 1 in A tale che $1 \cdot a = a \cdot 1\ \forall a \in A$;
3. un anello commutativo si dice *dominio d'integrità* se non possiede divisori dello zero, cioè se $ab = 0 \Rightarrow a = 0$ o $b = 0$;
4. un *campo* è un anello commutativo con unità che contiene l'inverso di ogni elemento non nullo;
5. un *corpo* è un anello (non necessariamente commutativo) con unità che contiene l'inverso di ogni elemento non nullo.

Proposizione. Un dominio d'integrità finito D è un campo.
Dimostrazione. È sufficiente provare che esiste $1 \in D$ tale che $1 \cdot a = a \cdot 1$ per ogni $a \in D$ e che ogni $a \neq 0$ è invertibile in D.
Sia $D = \{a_1, a_2, ..., a_n\}$ con gli a_i tutti diversi tra di loro. Sia $a = a_k \neq 0$.
Allora gli elementi:

$$aa_1, aa_2, ..., aa_n$$

sono anch'essi tutti distinti (infatti, se $i \neq j$, $aa_i = aa_j \Rightarrow a_i = a_j$).
Allora l'applicazione (iniettiva):

$$\begin{array}{c} \Psi : D \to D \\ a_i \mapsto aa_i \end{array}$$

è anche suriettiva (poiché D è finito) e quindi biunivoca.
Questo significa che ogni elemento di D si scrive come aa_i, ossia come prodotto di a per qualche elemento $a_i \in D$. In particolare, a stesso si scriverà come:

$$a = aa_{i_0} = a_{i_0} a \qquad \text{per qualche } a_{i_0} \in D$$

Abbiamo quindi che a_{i_0} è elemento unità per D, sia infatti $x = aa_i$ un qualunque elemento in D; allora:

$$x = aa_i = (aa_{i_0})a_i = (a_{i_0} a)a_i = a_{i_0}(aa_i) = a_{i_0} x$$

Indichiamo tale elemento unità a_{i_0} con 1; quindi, poiché $1 \in D$, 1 si scriverà come $1 = aa_j$ per qualche a_j in D. Da questo segue che a è invertibile.

□

Definizione. Un sottoinsieme *F* di un campo *K* si dice sottocampo del campo *K* se *F* è chiuso rispetto alle operazioni del campo *K*, $+$ e $\cdot$, e se con queste operazioni costituisce esso stesso un campo.

Polinomi:definizione e principali proprietà

Definizione. Una *funzione polinomiale* o *funzione razionale intera* sul campo K è una funzione p di K in sé tale che esistano un intero $n \geq 0$ ed elementi $a_i \in K$ in modo che:

$$p : x \in K \mapsto \sum_{i=0}^{n} a_i x^i \in K, \ \forall x \in K$$

Una funzione polinomiale p è pertanto una funzione da K in sé di natura particolare , indicata nel seguente modo:

$$p(x) = \sum_{i=0}^{n} a_i x^i .$$

Sia $\mathcal{F}$ l'insieme di tutte le funzioni polinomiali di K in sé. Trattandosi di funzioni da un campo K in sé, si possono definire addizione e moltiplicazione di tali funzioni utilizzando nella definizione l'addizione e la moltiplicazione del campo K:

$$(p+q)(x) \overset{\text{def}}{=} p(x) + q(x), \qquad (p \cdot q)(x) \overset{\text{def}}{=} p(x) \cdot q(x), \ \forall x \in K ;$$

quindi, se $p(x)$ e $q(x)$ sono le funzioni polinomiali

$$p(x) = \sum_{i=0}^{n} a_i x^i , \qquad q(x) = \sum_{j=0}^{m} b_j x^j$$

allora, tenendo presenti le proprietà soddisfatte da addizione e moltiplicazione di elementi di un campo, risulterà:

$$(p+q)(x) = p(x) + q(x) = \sum_{h=0}^{m} (a_h + b_h) x^h$$

e

$$(p \cdot q)(x) = p(x) \cdot q(x) = \sum_{h=0}^{n+m} \Big(\sum_{i+j=h} a_i b_j \Big) x^h .$$

L'insieme $\mathcal{F}$ costituito dalle funzioni polinomiali di K in sé, dotato delle due operazioni, $+$ e $\cdot$, ora introdotte, è un anello commutativo con unità; l'elemento neutro rispetto alla somma, ossia lo zero, è la funzione polinomiale che associa ad ogni $x \in K$ l'elemento 0 di

K, la funzione polinomiale opposta della funzione polinomiale p è la funzione polinomiale, che indichiamo con $-p$, tale che $(-p)(x) = -p(x)$ per ogni $x \in K$. Infine, l'unità di $\mathcal{F}$ è la funzione polinomiale che associa l'elemento 1 (unità del campo K) ad ogni $x \in K$, ossia la funzione costante uguale ad 1.

Definizione. Un *polinomio* $p(x)$ a coefficienti in un campo K è un'espressione formale del tipo

$$p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n, \qquad a_i \in K$$

dove x è un'indeterminata.

Dalla definizione segue che due polinomi $p(x) = \sum_{i=0}^{n} a_i x^i$ e $q(x) = \sum_{j=0}^{m} b_j x^j$, $a_i, b_j \in K$, sono *uguali* se e solo se $a_i = b_i \ \ \forall i$ (in particolare, se $m > n$, allora $b_{n+1} = b_{n+2} = ... = b_m = 0$).

L'insieme di tutti i polinomi a coefficienti in K si indica con $K[x]$.

Un fatto importante da notare è che le funzioni polinomiali e i polinomi vengono indicati allo stesso modo ma si tratta invece di due concetti diversi; due funzioni polinomiali sono uguali quando assumono gli stessi valori in corrispondenza di ogni $x \in K$, cioè quando si "comportano" allo stesso modo mentre due polinomi sono uguali se e solo se hanno la stessa scrittura formale.

Siano $p(x) = \sum_{i=0}^{n} a_i x^i$ e $q(x) = \sum_{j=0}^{m} b_j x^j$ due elementi di $K[x]$. Possiamo utilizzare per $K[x]$ le stesse definizioni di addizione e moltiplicazione di $\mathcal{F}$, cioè se $m \geq n$:

$$p(x) + q(x) \overset{\text{def}}{=} \sum_{h=0}^{m} (a_h + b_h) x^h$$

e

$$p(x)q(x) \overset{\text{def}}{=} \sum_{h=0}^{n+m} \left(\sum_{i+j=h} a_i b_j \right) x^h$$

Rispetto a queste operazioni, $K[x]$ diventa un anello commutativo con unità.
In questo caso dobbiamo però notare che il polinomio nullo non viene definito come quello che associa ad ogni elemento di K lo zero di K, ma come polinomio con tutti i coefficienti nulli; così l'opposto del polinomio $p(x) = \sum_{i=0}^{n} a_i x^i$ è il polinomio che ha come coefficienti gli opposti dei coefficienti a_i, ecc.

Definizione. Si definisce *grado del polinomio* $p(x) = \sum_{i=0}^{n} a_i x^i$ l'intero n, se $a_n \neq 0$. Si indica con $\deg p(x)$, oppure con $\partial p(x)$. Il coefficiente a_n prende il nome di coefficiente direttivo di $p(x)$.

Tra i gradi di due polinomi a coefficienti in un campo e i gradi della loro somma e del loro prodotto sussistono le seguenti relazioni:

$$\partial(p(x)+q(x)) \le \max(\partial p(x), \partial q(x)), \qquad \partial(p(x)q(x)) = \partial p(x) + \partial q(x)$$

Proposizione. L'anello $K[x]$ è un dominio d'integrità.

Dimostrazione. È sufficiente provare che $K[x]$ è privo di divisori dello zero.

Siano $p(x) = \sum_{i=0}^{n} a_i x^i$ e $q(x) = \sum_{j=0}^{m} b_j x^j$ due polinomi non nulli. Supponiamo che $\partial p(x) = n$ e $\partial q(x) = m$; ciò significa che $a_n \neq 0$ e $b_m \neq 0$.

Dalla definizione di polinomio prodotto $p(x)q(x)$, risulta che il coefficiente di x^{m+n} è $a_n b_m$, che è diverso da zero poiché anche a_n e b_m lo sono ed inoltre sono elementi di un campo, in cui non esistono divisori dello zero. Quindi $p(x)q(x)$ non può essere il polinomio nullo.

□

Proposizione (L'algoritmo della divisione tra polinomi). Siano $f(x)$, $g(x) \in K[x]$ due polinomi, con $g(x) \neq 0$. Allora esistono, e sono univocamente individuati, due polinomi $q(x)$ e $r(x)$ in $K[x]$ tali che

$$f(x) = g(x) \cdot q(x) + r(x), \text{ con } \partial r(x) < \partial g(x) \text{ oppure } r(x) = 0.$$

Definizione. Si dice che un polinomio $g(x) \in K[x]$ *divide* un polinomio $f(x) \in K[x]$, e si scrive $g(x) \mid f(x)$, se esiste un $q(x) \in K[x]$ tale che

$$f(x) = g(x) \cdot q(x).$$

Definizione. Un elemento $f(x)$ in $K[x]$ si dice *invertibile* se esiste un polinomio $g(x)$ in $K[x]$ tale che $f(x) \cdot g(x) = 1$.

Definizione. Siano $f(x)$ e $g(x)$ due polinomi appartenenti a $K[x]$ e non entrambi nulli. Si definisce *massimo comun divisore tra* $f(x)$ *e* $g(x)$, $(MCD(f(x), g(x))$, un polinomio $d(x) \in K[x]$ tale che:

a) $d(x) \mid f(x)$, $d(x) \mid g(x)$
b) se $d'(x) \mid f(x)$, $d'(x) \mid g(x)$, allora $d'(x) \mid d(x)$

Lo stesso procedimento che garantiva l'esistenza del MCD tra interi, ossia l'algoritmo euclideo delle divisioni successive, vale per i polinomi, e garantisce quindi l'esistenza di un massimo comun divisore di due polinomi non entrambi nulli.

Definizione. Due elementi $f(x)$ e $g(x) \in K[x]$ si dicono *associati* se esiste un elemento invertibile a di $K[x]$ tale che $f(x) = g(x) \cdot a$.

Osservazione. La relazione "essere associati" è una relazione d'equivalenza e in ogni classe di polinomi tra loro associati se ne può sempre scegliere uno monico (cioè con coefficiente direttivo uguale ad 1); se due elementi soddisfano entrambi la definizione di massimo comun divisore per due polinomi, essi sono associati e si può dire che il massimo comun divisore tra questi è l'unico massimo comun divisore monico.

Inoltre, anche per i polinomi vale la cosiddetta *identità di Bézout*:

se $f(x)$ e $g(x)$ stanno in $K[x]$, detto $d(x)$ il loro massimo comun divisore, esistono $h(x)$ e $k(x)$ in $K[x]$ tali che

$$d(x) = h(x) \cdot f(x) + k(x) \cdot g(x).$$

Definizione. Due polinomi $f(x)$ e $g(x)$ si dicono *coprimi* se

$$MCD(f(x), g(x)) = 1.$$

Definizione. Un polinomio $f(x)$ in $K[x]$, che non sia il polinomio nullo e non sia invertibile, si dice *irriducibile* su K se

$$f(x) = g(x)h(x),\ g(x), h(x) \in K[x] \Rightarrow g(x) \text{ o } h(x) \text{ è invertibile}$$

Se non è irriducibile, il polinomio si dice *riducibile*.

Definizione. Un polinomio $f(x)$ in $K[x]$ che non sia il polinomio nullo e non sia invertibile si dice *primo* se, ogni volta che $f(x)$ divide un prodotto $g(x)h(x)$ (con $g(x)$ e $h(x) \in K[x]$), allora divide uno dei due fattori.

Proposizione. Un polinomio in $K[x]$ è irriducibile se e solo se è primo.

Campi e loro estensioni

Definizione. Un'*estensione* di un campo F è un qualunque campo K che contenga F.

Definizione. Sia F un campo e sia K una sua estensione. Si definisce *grado* dell'estensione K sul campo F, e i indica con $[K:F]$, la dimensione di K come spazio vettoriale su F.

Definizione. Un'estensione K di un campo F si dice *finita* se il suo grado $[K:F]$ è finito; si dice *infinito* in caso contrario.

Teorema. Sia L un'estensione finita di K e sia K un'estensione finita di F, abbiamo cioè $L \subseteq K \subseteq F$. Allora:

i. L è un'estensione finita di F
ii. $[L:F] = [L:K][K:F]$.

Corollario. Sia L un'estensione finita di F e K un sottocampo di L contenente F. Allora:

$$[K:F] | [L:F].$$

Proposizione. Sia *K* un'estensione di *F* e sia $a \in K$; allora:

$$F(a) = \left\{ \frac{\alpha_0 + \alpha_1 a + \alpha_2 a^2 + ... + \alpha_s a^s}{\beta_0 + \beta_1 a + \beta_2 a^2 + ... + \beta_t a^t}, \alpha_i, \beta_j \in F, s, t \in \mathrm{N}, \ \beta_0 + \beta_1 a + ... + \beta_t a^t \neq 0 \right\}.$$

Definizione. Sia *F* un campo e *K* una sua estensione. Un elemento $a \in K$ si dice *algebrico* su *F* se $f(a) = 0$ per qualche polinomio non nullo $f(x) \in F[x]$, in caso contrario a si dice *trascendente* su *F*.

Note

1.DES. Il Data Encryption Standard**,** o DES, è un algoritmo di cifratura scelto come standard dal Federal Information Processing Standard (FIPS) per il governo degli Stati Uniti d'America nel 1976 e in seguito diventato di utilizzo internazionale. Questo algoritmo all'inizio ha suscitato molte discussioni per via della sua chiave di codifica corta e per via di alcune scelte progettuali che erano secretate; si supponeva che dietro queste scelte vi fosse la National Security Agency (NSA) e l'inserimento di una backdoor. Di conseguenza il DES è stato soggetto a una intensa analisi di tipo accademico che ha prodotto le ricerche che sono alla base dei moderni algoritmi di cifratura e delle moderne tecniche di crittanalisi.
DES è considerato insicuro per moltissime applicazioni. La sua insicurezza deriva dalla chiave utilizza per cifrare i messaggi che è di soli 56 bit. Macchine specializzate sono in grado di esaminare tutte le possibili chiavi e decifrare il messaggio in meno di 24 ore. Dal punto di vista teorico esistono delle tecniche crittanalitiche in grado di forzare il DES ma queste tecniche non sono applicabili praticamente. Per rendere sicuro il DES si sono sviluppate delle evoluzioni come il Triple DES: questo algoritmo espande la chiave impedendo per il momento un attacco di forza bruta sebbene renda l'algoritmo in teoria vulnerabile ad attacchi che però non sono attuabili nella pratica.
In molti documenti per indicare il DES si utilizza anche la sigla DEA (Data Encryption Algorithm).

2.Rete di Feistel. In crittologia, un cifrario di Feistel è un algoritmo di cifratura a blocchi con una particolare struttura sviluppata dal crittologo dell'IBM Horst Feistel e da lui questa struttura ha preso il nome di rete di Feistel. Moltissimi algoritmi di cifratura a blocchi utilizzano questa struttura incluso il Data Encryption Standard (DES). La struttura inventata da Feistel ha il vantaggio che la cifratura e decifratura sono operazioni molto simili, spesso identiche e che basta invertire il funzionamento del gestore della chiave. Quindi i circuiti di cifratura e decifratura spesso sono gli stessi. Il meccanismo di criptatura ricorda le operazioni in cascata di Enigma.

3.S-Box. In Crittografia, le S-Box (o Substitution-Box) sono dei componenti base degli algoritmi a chiave simmetrica. Nei blocchi di cifratura le S-Box vengono utilizzate per oscurare relazioni tra il testo in chiaro e il testo cifrato e, spesso, vengono appositamente progettate per resistere alla crittanalisi.

4.Trapdoor. Trapdoor significa letteralmente “botola”, o “passaggio segreto”.
Una funzione one-way trapdoor e' una funzione iniettiva, tale che:

- $f(x)$ e' facile da calcolare
- $f^{-1}(x)$ esiste, ma e' intrattabile se non si conosce un “segreto” (trapdoor)

5.ANSI C. Lo standard ANSI C è stato definito nel 1989 dall'American National Standard Institute, come standard del linguaggio C ed è stato successivamente adottato dalla International Standard Organization come standard internazionale con la sigla ISO/IEC 9899:1990, e va anche sotto il nome di standard ISO C.
Scopo dello standard è quello di garantire la portabilità dei programmi C fra sistemi operativi diversi, ma oltre alla sintassi ed alla semantica del linguaggio C (operatori, parole chiave, tipi di dati) lo standard prevede anche una libreria di funzioni che devono poter essere implementate su qualunque sistema operativo.

6.Java. Il linguaggio Java è un linguaggio di programmazione orientato agli oggetti, creato da James Gosling e altri ingegneri di Sun Microsystems. Il gruppo iniziò a lavorare nel 1991, il linguaggio inizialmente si chiamava Oak. Il nome fu successivamente cambiato in Java a causa di un problema di copyright (il linguaggio di programmazione Oak esisteva già nel 1991). Java fu annunciato ufficialmente il 23 maggio 1995 a SunWorld. La piattaforma di programmazione Java è fondata sul linguaggio stesso, sulla Java Virtual Machine (JVM) e sulle API.
7.Smart Card. La smart card è una carta di identificazione, normalmente usata per l'autenticazione forte di coloro che la possiedono.
Si compone di un supporto plastico, di dimensioni e fattura normalmente identiche a quelle delle carte di credito (formato ISO), e di un microprocessore che contiene dati sia protetti che di libero accesso.
Per accedere ai dati protetti non c'è altro modo che fornire un codice numerico (PIN) al microprocessore per identificarsi.
Senza l'uso del PIN (Personal Identification Number), i dati non possono essere letti e, dopo un certo numero di tentativi errati nell'immettere il PIN, le funzioni della smart card si bloccano. Normalmente una carta bloccata può essere sbloccata tramite uno specifico codice (PUK, PIN Unblocking Number, numero di sblocco del PIN).

8.Stream Cipher. Uno stream cipher è un algoritmo di tipo simmetrico e la principale caratteristica di questo tipo di algoritmi è l'eccezionale velocità, di molte volte superiore a qualsiasi tipo di block Al contrario di altri "cipher" che operano su blocchi di dimensione prefissata gli stream cipher crittano il messaggio "al volo". Uno stream cipher lavora su piccoli stock di dati che vengono combinati con una chiave attraverso un operatore logico.

9.Algoritmo di Hashing. Si tratta di un algoritmo che partendo da un documento di qualsiasi dimensione lo elabora e produce un codice di misura fissa. Il metodo d'elaborazione è tale che, se il documento fosse cambiato in qualunque sua parte, questo codice cambierebbe. Per esemplificare s'immagini un algoritmo che calcola il numero di lettere, il numero di parole, la frequenza d'ogni lettera (etc.), se cambia una qualsiasi lettera o parola anche il risultato cambia. Si può pensare al codice prodotto dall'algoritmo di hashing come ad un'impronta del documento. Dall'impronta non è possibile risalire al documento, però se questo cambia, anche solo in minima parte, allora cambia anche l'impronta.

Bibliografia

[1] Berardi L., Algebra e Teoria dei Codici Correttori,
[2] Menezes A., Oorschot P., Vanstone S., Handbook of Applied Cryptography
[3] Piacentini Cattaneo G.M., Algebra, Zanichelli
[4] http://alpha01.dm.unito.it/personalpages/cerruti/crypto2/Rijndael.pdf
[5] http://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[6] http://www.cryptosystem.net/aes/
[7] http://www.nist.gov
[8] http://csrc.nist.gov/CryptoToolkit/aes/rijndael
[9] http://www.nsa.gov
[10] http://it.wikipedia.org/wiki/DES